BERGISCHE
UNIVERSITÄT
WUPPERTAL

Fakultät für Maschinenbau und Sicherheitstechnik

Masterthesis

im Studiengang **Qualitätsingenieurwesen** beim Fachgebiet für

Verkehrssicherheit und Zuverlässigkeit

zur Erreichung des akademischen Grades

Master of Science

Zuverlässigkeitstechnik in der Industrie 4.0: Klassifikation von Fehlerzuständen bei Zeitreihen mittels maschinellen Lernens

Vorgelegt von
Raphael Willeke

Matrikelnummer:	1431676
Studiengang:	Qualitätsingenieurwesen
Erstprüfer*in	Jun.-Prof. Dr. Tordeux, Antoine
Zweitprüfer*in:	Tim Julitz, M. Sc.
Ausgabe:	10.06.2024
Abgabe:	09.11.2024

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe. Mit Abgabe der Abschlussarbeit erkenne ich an, dass die Arbeit durch Dritte eingesehen und unter Wahrung urheberrechtlicher Grundsätze zitiert werden darf. Ferner stimme ich zu, dass die Arbeit durch das Fachgebiet an Dritte zur Einsichtnahme herausgegeben werden darf.

Hattingen, den 11.10.2024 Unterschrift: 

Kurzfassung

Der Funktionserhalt für Maschinen und Anlagen ist für Unternehmen ein wichtiger Aspekt. Durch Instandhaltungsmaßnahmen wird diese Funktion sichergestellt. Mit neuen Technologien im Rahmen der Industrie 4.0 ergeben sich viele Möglichkeiten der Verbesserung der Instandhaltung insbesondere im Bereich der Fehlerklassifikation unter Berücksichtigung von Datenmessungen mit Zeitbezug.

Im Rahmen dieser Arbeit wird eingehend die Industrie 4.0 beschrieben und die Instandhaltung näher beleuchtet. Es wird resümiert, dass Instandhaltungsstrategien aufgrund hoher Kosten und großer Bedeutung für Unternehmen notwendig sind. Unter Berücksichtigung von hohen Datenverfügbarkeiten sowie besseren Sensoren in der Industrie 4.0 wird die datenbasierte zustandsorientierte Instandhaltung ausgewählt und näher erläutert. Dabei wird der Fokus auf Künstliche Intelligenz gelegt. Verschiedene Algorithmen aus den Bereichen des maschinellen Lernens und Deep Learning werden präsentiert.

Unter Berücksichtigung der Eigenschaften eines ausgewählten Datensatzes von zeitbezogenen Vibrationsdaten und einer Literaturrecherche werden Algorithmen zur automatischen Klassifikation der Daten ausgewählt. Es werden Neuronale Faltungsnetze (CNN) und Rekurrente Neuronale Netze (RNN) auf den Datensatz unter Verwendung der Programmiersprache Python angewendet. Zuvor wird der Datensatz grafisch anhand von Liniendiagrammen sowie Boxplots präsentiert sowie auf Vollständigkeit und unschlüssige Werte untersucht. Ergänzend werden die einzelnen Betriebszustände sowie der Aufbau der Anlage beschrieben.

Die ausgewählten Algorithmen werden dann auf den Datensatz angewendet und anhand von Kennzahlen verglichen. Angewendete Kennzahlen sind die Genauigkeit, Recall, Präzision, F1-Score und MSE. Das beste RNN- und CNN- Modell sind etwas weniger performant als der Referenzwert eines K-Nearest-Neighbour-Algorithmus. Die entwickelten Modelle klassifizieren darüber hinaus einen Fehlerzustand als solchen und liefern mit der Literatur vergleichbare Ergebnisse, sodass geschlussfolgert wird, dass sie sich für eine Anwendung in der zustandsbasierten Instandhaltung eignen.

Abstract

Maintaining the functionality of machines and systems is an important aspect for companies. This function is ensured through maintenance measures. With new technologies in the context of Industry 4.0, there are many possibilities for improving maintenance, particularly in the area of fault classification, taking into account data measurements with time reference.

This thesis describes Industry 4.0 in detail and takes a closer look at maintenance. It is summarized that maintenance strategies are necessary due to high costs and great importance for companies. Taking into account high data availability and better sensors in Industry 4.0, data-based condition-based maintenance is selected and explained in more detail. The focus is placed on artificial intelligence. Various algorithms from the fields of machine learning and deep learning are presented.

Considering the characteristics of a selected data set of time-related vibration data and a literature search, algorithms for the automatized classification of the data are selected. Convolutional neural networks (CNN) and recurrent neural networks (RNN) are applied to the data set using the Python programming language. Beforehand, the data set is presented graphically using line diagrams and box plots and examined for completeness and inconclusive values. In addition, the individual operating states and the structure of the system are described.

The selected algorithms are then applied to the data set and compared using key metrics. The metrics utilized are accuracy, recall, precision, F1-score and MSE. The best RNN and CNN models are slightly less performant than the reference value of a K-Nearest Neighbor algorithm. Furthermore, the developed models classify a fault condition as such and provide results comparable to the literature, so it is concluded that they are suitable for application in condition-based maintenance.

Abkürzungsverzeichnis

AI
  Artificial Intelligence
ANN
  Artificial Neural Network
CAD
  Computer-Aided-Design
CBM
  Condition Based Maintenance
CNN
  Convolutional Neural Network
DL
  Deep Learning
IoT
  Internet of Things
KI
  Künstliche Intelligenz
ML
  Machine Learning bzw. maschinelles Lernen
MSE
  Mean Squared Error
NN
  Neuronales Netz
PdM
  Predicitve maintenance
PHM
  Progonstics and Health Management
ReLU
  Rectified Linear Unit
RNN
  Recurrent Neural Network
SVM
  Support Vector Machine

Abbildungsverzeichnis

Tabellenverzeichnis

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangssituation

In Fertigungs- und Produktionsstätten sind Instandhaltungskosten ein Hauptteil der operativen Kosten. Dabei belaufen sich die Kosten, je nach Industriesektor, auf 15 bis 60 Prozent der Produktionskosten. Darüber hinaus gilt ein Drittel der durchgeführten Instandhaltungsarbeiten aufgrund von nicht notwendiger, ineffizienter oder nicht adäquater Ausführung als verschwendet [30]. Im Bereich der Pharmaindustrie sind 60 % der Produktionskosten abhängig von einer effektiven Instandhaltung. In Deutschland belaufen sich Stand 2019 die jährlichen Instandhaltungskosten auf 3,63 Milliarden Euro [6].
Es ist daher offensichtlich, dass Instandhaltungskosten einen großen Einfluss auf die wirtschaftliche Lage eines Unternehmens respektive einer Produktionsstätte haben. Es gilt diese zu senken und die Instandhaltung zu optimieren, um keine ineffizienten unnötigen Arbeiten durchzuführen. Ineffizient meint ebenso eine zu späte wie zu frühe Durchführung von Instandhaltungsarbeiten und entspricht der Lean-Idee als Vermeidung von Verschwendung [25]. Bei einer Maschine oder Produktionslinie, die einen Stillstand aufgrund eines nicht erkannten Defektes hat, eine Instandhaltung durchzuführen, gilt als ebenso ineffizient wie die Arbeiten bei einer Maschine, die problemlos noch eine gewisse Zeit adäquate Resultate erbringen würde. Ebenso können Fehler die Produktion unterbrechen. Die Verbesserung der Instandhaltung hat also einen direkten Einfluss auf den Grad der Anforderungserfüllung der Produktion und somit auf die Produktionsqualität. Instandhaltungsstrategien sind daher notwendig, um eine sichere kontinuierliche Produktion zu gewährleisten [28].
In der vierten industriellen Revolution bieten das „Condition Monitoring“ und das maschinelle Lernen greifbare Lösungen. Größere Datenverfügbarkeiten und bessere Auswertungskapazitäten können so dazu beitragen ein besseres Verständnis für den Herstellungsprozess und Optimierungsmöglichkeiten herbeizuführen. Sensoren überwachen dabei den Zustand der Maschine oder der Produktionslinie. Die so erhobenen Daten werden durch einen (Lern-)Algorithmus ausgewertet. Dabei werden zum Beispiel Prognosen der noch zu erwartenden Lebensdauer oder für eine Reduktion der Ausfallzeiten erstellt. Basis dieser Prognose sind die erhobenen Daten und der Algorithmus. Dieser wird anhand von vorgefertigten, vorhandenen Daten und Datentypen im Vorfeld trainiert und auf die entsprechenden Problemstellungen bzw. Fehlerfälle angepasst.
Ziel dieser Masterthesis ist es Methoden des Maschinellen Lernens zur Klassifizierung von Zeitreihen in Bezug auf die Verbesserung der Instandhaltung im Rahmen der Industrie 4.0 auszuwählen, welche automatisiert aufgrund von bereits vorliegenden Daten Betriebszustände

erkennen und richtig zuordnen. In Frage kommende Methoden resp. Algorithmen sollen dann auf einen beispielhaften Datensatz angewendet und anhand von Metriken beurteilt werden.

Die Arbeit gliedert sich in sechs Hauptteile. Zu Beginn wird ein Überblick über das Thema Industrie 4.0 gegeben. Es werden technische Entwicklungen, Technologiefelder sowie Trends dargestellt.

Im Anschluss wird eine größere Einsicht in das Thema Instandhaltung gegeben. Verschiedene Instandhaltungsarten werden vorgestellt, von Vor- und Nachteilen betrachtet und auf dessen Grundlage eine Instandhaltungsart ausgewählt. Verschiedene Methoden des Maschinellen Lernens und des Deep Learnings werden vorgestellt sowie Zeitreihen kurz erläutert.

Anschließend wird der Systemaufbau, an dem die später verwendeten Daten gemessen wurden, sowie die unterschiedlichen Betriebszustände kurz vorgestellt. Der Datensatz wird ausführlich präsentiert und beschrieben. Methoden für die späteren Klassifikation der Daten werden unter Berücksichtigung von relevanten Quellen und den Charakteristiken der Daten ausgewählt. Die ausgewählten Methoden werden dann auf den vorgestellten Datensatz angewendet.

Im Anschluss werden die Ergebnisse präsentiert sowie diskutiert und ein Ausblick gegeben.

2 Grundlagen

Zeitreihenanalysen sind aktuelle Werkzeuge im Kontext der Instandhaltung in der Industrie 4.0. Daher wird in den nachstehenden Abschnitten ein Überblick über die Industrie 4.0, Instandhaltung sowie über Zeitreihen und deren Analysen gegeben.

2.1 Industrie 4.0

„Menschen, Maschinen und Produkte sind direkt miteinander vernetzt: die vierte industrielle Revolution hat begonnen“ [8].

Das Zitat zeigt, der Begriff Industrie 4.0 ist eng verknüpft mit dem „Internet of Things“ (IoT) und der vierten industriellen Revolution. Der erstmals im Jahr 1999 aufgekommene Begriff IoT wird als Sammelbegriff für Technologien verwendet, die es ermöglichen, dass Computer, Maschinen, Sensoren, etc. durch das Internet miteinander interagieren können. Dabei fallen nicht nur industrielle Produktionen unter das IoT, sondern auch aktuelle Haushaltsgeräte [36]. Mögliche Einsatzgebiete intelligenter Vernetzung sind zum Beispiel flexible Produktionen, wandelbare Fabriken, kundenzentrierte Lösungen, optimierte Logistik, der Einsatz von Daten oder die ressourcenschonende Kreislaufwirtschaft [8].

Flexible Produktion meint, dass die verschiedenen an einem Produkt beteiligten Firmen eine bessere Abstimmung bei der Entwicklung des Produkts und der Planung der Maschinenauslastungen durch eine digitale Vernetzung erreichen [8].

Die wandelbare Fabrik umfasst eine Modularisierung von Fertigungslinien. So wird eine einfachere Adaption an neue Fertigungsaufgaben, auch für kleinere Stückzahlen, erreicht. Auch der Megatrend der Individualisierung findet durch das Feld der kundenzentrierten Lösung seinen Einzug. Der Kunde wird durch eine digitale Schnittstelle direkt in die Fertigung eingebunden und kann so zum Beispiel am heimischen Computer seine Badeschuhe designen, die direkt vor Ort gefertigt werden können. Auch können smarte Produkte in Echtzeit Felddaten an den Hersteller übermitteln. So ist es möglich direkt Verbesserungen anzustoßen, Auswertungen von Realdaten zu erstellen oder Services zu entwickeln, die auf den Kunden und seine Bedürfnisse zugeschnitten sind. Darüber hinaus ist es durch die Auswertung von Felddaten möglich, Produkte über ihren kompletten Lebenszyklus zu betrachten und einen Beitrag zu einer ressourcenschonenden Kreislaufwirtschaft zu leisten. Weiterführend wird bereits im Design die Art und Weise der Wiederverwertung von Materialien berücksichtigt [8], [34].

Ein weiteres Feld der Industrie 4.0 ist die optimierte Logistik. Durch den Einsatz von Algorithmen ist eine optimale Berechnung des Lieferweges oder auch eine eigenständige Bedarfsmeldung und Bestellung durch vernetze Produktionsmaschinen möglich. Der Materialfluss wird so bestmöglich angepasst [8].

Bezugnehmend auf diese Arbeit ist die Möglichkeit des Einsatzes von Daten am relevantesten. Das Bundesministerium für Wirtschaft und Klimaschutz schreibt auf seiner Homepage, dass durch Dateneinsatz sowie -auswertung und -erhebung, Aussagen zum Zustand eines Produktes oder einer Maschine direkt getroffen werden können. In [8] wird als Beispiel ein Aufzughersteller, welcher durch die Auswertung von Zustandsdaten der Aufzüge seinen Kunden eine optimierte Wartung zur Vorbeugung eines Ausfalls anbieten kann, angeführt.

Erstmals aufgekommen ist der Begriff Industrie 4.0 als Bezeichnung „Plattform Industrie 4.0“. Heutzutage wird er vor allem als Beschreibung der vierten industriellen Revolution verwendet. Daher soll in dem folgenden Abschnitt die vorangegangenen industriellen Revolutionen und der Zusammenhang zur vierten Revolution dargestellt werden.

Die aufeinander aufbauenden Stufen der industriellen Revolutionen sind in Abbildung 2.1 dargestellt. Mit der Mechanisierung der Landwirtschaft sowie der fertigenden Industrie, begann Mitte des 18. Jahrhunderts die erste industrielle Revolution basierend auf der Erfindung der Dampfmaschine durch Thomas Newcomon und der Weiterentwicklung durch James Watt. Zusätzliche Weiterentwicklungen waren der mechanische Webstuhl und neue Verfahren bei der Rohstoffgewinnung [2], [4], [20].

Ende des 19. Jahrhunderts begann durch großflächige Elektrifizierung von Städten, den Ausbau von Eisenbahnen, dem Einsatz von Verbrennungsmotoren bei Fortbewegungsmitteln und der Erfindung des Fließbands sowie der Fließbandfertigung die zweite industrielle Revolution. Ihr zu zuweisen sind auch die Erfindung des Elektromotors, der Telegraphen und Telefone sowie die Herstellung von Kunststoffen. Die bekannteste Entwicklung der zweiten industriellen Revolution ist hingegen die Fließbandfertigung durch Henry Ford [2], [4], [20], [34].

Die dritte industrielle Revolution wurde eingeleitet durch die Erfindung des Computers Mitte des 20. Jahrhunderts. Dieser ermöglichte eine weitestgehende Automatisierung von Maschinen durch einfache Computerprogramme. Hinsichtlich weiterer technischer Errungenschaften sind die Raumfahrt als auch erste Satelliten zu nennen. Basierend auf der fortschreitenden Entwicklung von Mikroelektronik ist der neu entstandene Industriesektor der IT rasant wachsend. Dieser Zeit zu zuordnen sind auch die Entwicklung der Mobiltelefonie und des Internets. Hierdurch vollzog sich ein drastischer Wandel der Kommunikation und bildete die Grundlage für die derzeitige vierte industrielle Revolution [2], [4], [20], [34].

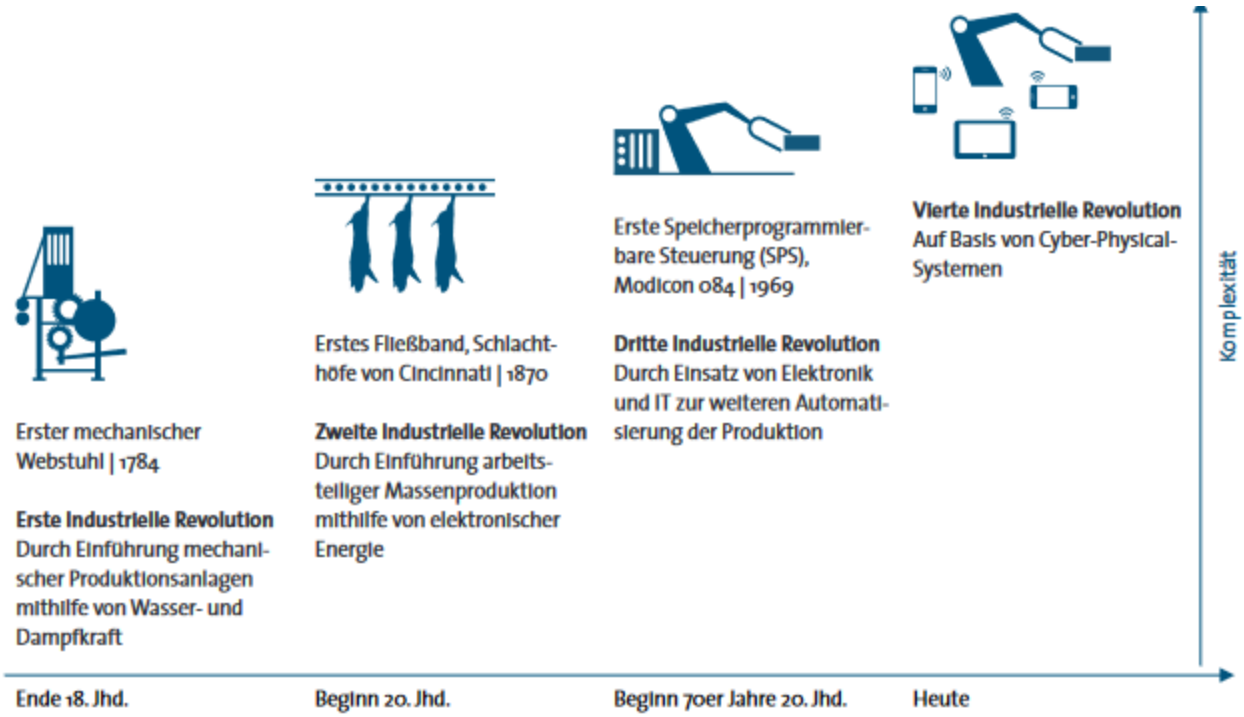


Abbildung 2.1 Stufen der industrieller Revolutionen [4]

2.2 Digitale Transformation

Für ein gesamtheitliches Bild der technologischen Entwicklung, ist es notwendig die digitale Transformation nachfolgend zu beschreiben. Sie weist eine ähnliche schrittweise Entwicklung wie die der industriellen Revolutionen auf.

Anfänglich wurden lediglich Abbildungen analoger Prozesse erschaffen. Händisches Zeichnen von bspw. Fertigungszeichnungen wurde ersetzt durch „Computer-Aided-Design“ (CAD) Programme und andere Softwarelösungen. Beginnend war der Schritt bzw. die Stufe der Digitalisierung in der Mitte der 1950er Jahre mit 2D-Modellen, auch für Fertigungs- und Planungsprozesse, siehe Abbildung 2.2 [5].

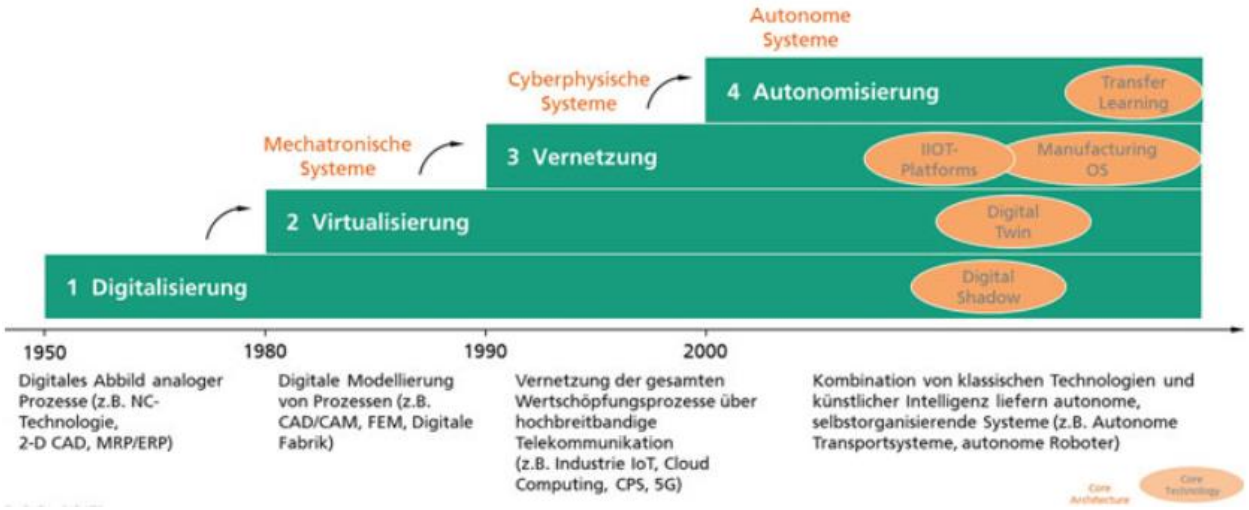


Abbildung 2.2 Digitale Transformation [5]

In den 1980er Jahren begann die Stufe der Virtualisierung. 2D-Ansichten waren nicht mehr ausreichend. Die Intention für eine Weiterentwicklung war ein Modell zu haben, dass sich nach Möglichkeit genauso verhält wie das reale Produkt. Digitale Tests und Simulationen ermöglichten Kosteneinsparungen und Optimierungen bereits vor der Produktion oder der Errichtung von Produktionsstätten oder Fertigungslinien durch frühzeitige Fehlererkennung und -vermeidung. Nach dem Bau der Produktionsstätte oder dem finalen Design des Produktes wurde dann dieses Modell keiner Benutzung mehr zugeführt. Ein weiterer Aspekt der Virtualisierung war die Überführung von Hardwarefunktionen auf die Softwareseite. Dies ermöglichte Kosteneinsparungen aufgrund der nicht mehr notwendigen Beschaffung kostenintensiverer Materialien. Signifikant für diese Stufe war auch, dass auch auf Software während des Feldeinsatzes lediglich mit einem hohen Aufwand zugegriffen werden konnte. Durch die Entwicklung und Einführung des Internets war es möglich eine Vernetzung zu schaffen. Nach Abbildung 2.2 ist Vernetzung die dritte Stufe der Digitalisierung. Es ergibt sich die Möglichkeit physische Systeme mit virtuellen Systemen zu verbinden und sogenannte digitale Zwillinge von bekannten physischen Modellen zu schaffen. Der digitale Zwilling verkörpert den digitalen Schatten des körperlichen Systems und spiegelt virtuell den Echtzeitzustand eines Systems wider. So können im Vorfeld verschiedene Systemzustände virtuell abgebildet und Veränderungen getestet, aber auch weitere Funktionen virtualisiert werden. Virtuelle Systeme können ebenfalls miteinander verbunden werden. Dadurch können gegenseitige Beeinflussungen von Systemverbindungen in Echtzeit getestet werden. Ebenfalls entstehen so neue, erweiterte Systeme und komplexe Systemarchitekturen. Ein großer Teil der Funktionen eines Produktes wird auf die Softwareseite verlagert und durch die Echtzeitdatenermittlung könne Prognosen und Auswertungen innerhalb kürzester Zeit über Cloud-Lösungen durchgeführt werden. Eine Anwendung des gleichen Tools für Echtzeitdatenerhebung ist ebenfalls in der

Fertigung möglich. Eine Vernetzung erfolgt über das sogenannte „Industrial Internet of Things“, eine industrielle Internetverbindung. Über eine unternehmensspezifische zusätzliche Plattform erfolgt die Steuerung der Fertigung [5].

Auf der dritten Stufe der digitalen Transformation aufbauend erfolgt die Autonomisierung. Diese ist die Fähigkeit digitaler Systeme, basierend auf kognitiven Prozessen, selbstständig Entscheidungen zur Erreichung eines Ziels oder Zustands zu treffen. Darunter fallen beispielsweise einfache Assistenzsysteme wie sie bereits in Autos verbaut werden oder auch komplexere Anwendungen bei denen sich zum Beispiel miteinander vernetzte Roboter gegenseitig das Greifen beibringen. Als Basis der Stufe der Autonomisierung werden künstliche Intelligenz und auch maschinelles Lernen angeführt. Kriterium dieser Technologien ist eine angemessene Datenerfassung und -verarbeitung [5].

Es ist ersichtlich, dass erst durch die Entwicklung der digitalen Technologien maschinelles Lernen und künstliche Intelligenz möglich wurden. Dadurch entwickeln sich ebenso neue Geschäfts- wie Aktionsfelder. Einige wurden oben bereits angedeutet. Zusammengefasst lassen sich die in Abbildung 2.3 dargestellten Technologiefelder für die Industrie 4.0 auf Basis der digitalen Transformation identifizieren. Diese werden nachfolgend kurz beschrieben.

Embedded Systems, intelligente Objekte und cyber-physische Systeme (CPS) bilden eine Grundlage für eine smarte Vernetzung. Embedded Systems sind eingebettet informationsverarbeitende Systeme, die in einem größeren Produkt integriert sind. In der Regel bestehen sie aus einer Kombination von Hard- und Software. Darunter fallen zum Beispiel Mikrocontroller, Kommunikationssysteme, Identifikatoren, Sensoren und Aktoren [4], [7].

Sensorik bildet die Grundlage zur Erhebung von Daten. Immer besser werdende Sensoren bilden die Grundlage einer besseren Datenerfassung und somit eine kostengünstigere Methode, um Maschinen oder auf Fertigungslinien zu steuern, zu überwachen oder auch zu optimieren. Gleichzeitig wird durch bessere Sensorik das Feld der „Smart Factory“ ermöglicht. Als Smart Factory wird ein „einzelnes oder [der] Verbund von Unternehmen, das / der IKT [Informations- und Kommunikationstechnologie] zur Produktentwicklung, zum Engineering des Produktionssystems, zur Produktion, Logistik und Koordination der Schnittstellen zu den Kunden nutzt, um flexibler auf Anfragen reagieren zu können“ [20] bezeichnet. Intelligente miteinander vernetze Maschinen tauschen über die Technologiefelder „Cloud Computing“ und Robuste Netze Daten wie Aufträge oder Zustände miteinander aus und können so miteinander interagieren. Termine oder Abläufe können so festgelegt und koordiniert werden, sodass eine Gesamtoptimierung hinsichtlich der Auslastung, Durchlaufzeiten oder auch Qualität erreicht wird. Ein Austausch von Maschinen und wie oben bereits erwähnt, eine angepasste Fertigung, gelingt durch eine Modularisierung von Maschinen und Komponenten. Das Feld der robusten Netze beschreibt das Kommunikationsnetzwerk. Diese erfolgt funk- (WLAN oder Mobilfunk) oder kabelbasiert. Relevant sind Datenübertragungsraten, Datenmengen sowie die Anbindung von

mobilen Endgeräten. Der Bereich Cloud Computing beschreibt die bereits erwähnte Plattform, über die eine intelligente Produktion gesteuert werden oder interagieren kann. Ein Vorteil gegenüber einer lokalen Serverlösung ist die Möglichkeit der Verarbeitung und Speicherung von größeren Datenmengen sowie der Bereitstellung von Applikationen (Apps). Das letzte zu erwähnende Technologiefeld ist die IT-Security. Diese stellt den Daten- als auch Informationsschutz sicher [2], [4], [20].

Das Zusammenwirken dieser Technologiefelder ist ebenso herausfordernd wie jede einzelne Technologie. Im Bereich der Instandhaltung ergeben sich hieraus Chancen zur Verbesserung und Optimierung von Systemen.

Eine dieser Chancen ist die Eröffnung neuer Geschäftsfelder für beispielsweise eine dienstleisterbasierte Auswertung von erhobenen Daten zur Ermittlung von Stillstandzeiten, Wartungs- und Instandhaltungsoptimierung [36].

Im folgenden Kapitel wird daher näher auf die Instandhaltung eingegangen.

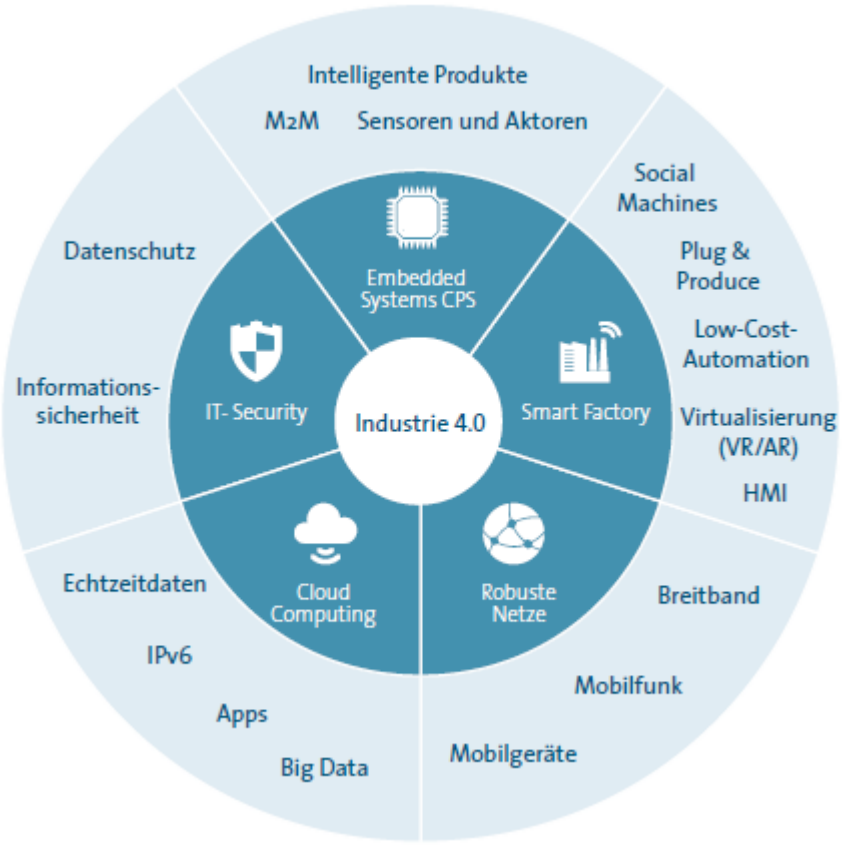


Abbildung 2.3 Technologiefelder Industrie 4.0 [4]

2.3 Instandhaltungsstrategien/ Zuverlässigkeitstechnologien in der Industrie 4.0

In diesem Abschnitt soll zunächst der Begriff der Instandhaltung näher erläutert werden. Diese hat einen signifikanten Einfluss auf die Wirtschaftlichkeit sowie die Wettbewerbsfähigkeit durch Steigerung der Leistungs- als auch der Kostenführerschaft eines Unternehmens. Studien zu diesem Thema haben gezeigt, dass 15 bis 70 Prozent der totalen Produktionskosten durch Instandhaltungstätigkeiten verursacht werden [26]. Fertigungsmaschinen sollen zu dem immer in gleicher Qualität, bei einer maximalen Effizienz mit einer hohen Zuverlässigkeit produzieren [34]. Hinzu kommt im Rahmen der Industrie 4.0 und dem Internet of Things eine immer höhere Anzahl an Komponenten, die in Stand gehaltenwerden müssen. Ebenso steigt die technische Kompliziertheit von Maschinen und Anlagen [5]. Aufgrund der Wichtigkeit der Instandhaltung ist es zunächst sinnvoll grundsätzliche Begrifflichkeiten zu definieren.

Die DIN EN 13306 gibt eine Übersicht über eine Vielzahl an Begriffsdefinitionen, die sich mit dem Begriff Instandhaltung in Verbindung bringen lassen. Instandhaltung ist definiert als: „Kombination aller technischen und administrativen Maßnahmen sowie Maßnahmen des Managements während des Lebenszyklus eines Objekts, die dem Erhalt oder der Wiederherstellung seines funktionsfähigen Zustands dient, sodass es die geforderte Funktion erfüllen kann“ [11]. Die Definition zeigt, dass es ein sehr umfangreiches Gebiet skizziert und weitere Erläuterung nachfolgend notwendig sind. Dabei wird allerdings nur ein Auszug der wichtigsten Definitionen wiedergegeben.

Unter Instandhaltungsmaßnahmen sind Beobachtungen und Analyse des Zustandes eines Objektes zu verstehen, was die Tätigkeiten der Inspektion, Überwachung Prüfung, Diagnose, Prognose usw. einschließt, aber ebenso aktive Instandhaltungsmaßnahmen wie die Instandsetzung als auch die Aufarbeitung [11].

Als Inspektion wird die „Prüfung auf Konformität der maßgeblichen Merkmale eines Objektes durch Messung, Beobachtung oder Prüfung“ [11] verstanden.

Instandsetzung, was gleichzusetzen ist mit dem Begriff Fehlerkorrektur, umfasst eine „physische Maßnahme, die ausgeführt wird, um die Funktion eines fehlerhaften Objektes wiederherzustellen“ [11]. Darunter fallen Tätigkeiten der Fehlerortung, Funktionsprüfung, Vorbereitung der Durchführung, beinhaltend Kalkulation, Terminplanung, Abstimmung, Bereitstellung von Personal, Mitteln und Material, Erstellung von Arbeitsplänen, Auftrag, Auftragsdokumentation und Analyse des Auftragsinhaltes, Vorwegmaßnahmen wie Arbeitsplatzausrüstung, Schutz- und Sicherheitseinrichtungen, Überprüfung der Vorbereitung und der Vorwegmaßnahmen einschließlich der Freigabe zur Durchführung, Durchführung, Abnahme, Fertigmeldung, Auswertung einschließlich Dokumentation, Kostenaufschreibung, Aufzeigen der Möglichkeit von Verbesserungen und Rückmeldung [10], [11], [34].

Im Kontext der Instandhaltung sind wichtige Grundformen neben der Instandsetzung und Inspektion, die Wartung sowie die Verbesserung. Unter Wartung werden „Maßnahmen zur Verzögerung des Abbaus des vorhandenen Abnutzungsvorrats“ [10] bezeichnet. Inspektion hingegen ist nach DIN 31051 definiert als: „Prüfung auf Konformität der maßgeblichen Merkmale eines Objekts, durch Messung, Beobachtung oder Funktionsprüfung“ [10].
Als Verbesserung im Kontext der Instandhaltung wird die „Kombination aller technischen und administrativen Maßnahmen sowie Maßnahmen des Managements zur Steigerung der immanenten Zuverlässigkeit und/oder Instandhaltbarkeit und/oder Sicherheit eines Objekts, ohne seine ursprüngliche Funktion zu ändern“ [10] bezeichnet. Dabei sind Verbesserungen auch Maßnahmen, die im Sinne der Vermeidung einer unsachgemäßen Verwendung oder zur Vermeidung von Ausfällen durchgeführt werden [10].
Zuverlässigkeit ist die „Fähigkeit eines Objektes, eine geforderte Funktion unter gegebene Bedingungen für eine gegebene Zeitspanne zu erfüllen“ [11]. Diese ist nicht mit der Verfügbarkeit gleich zu setzen. Die Verfügbarkeit beschreibt die „Fähigkeit eines Objekts, unter gegebenen Bedingungen und wenn erforderlich in einem Zustand zu sein, eine geforderte Funktion zu erfüllen, vorausgesetzt, dass die erforderlichen externen Ressourcen bereitgestellt sind“ [11]. Wesentlicher Unterschied zwischen diesen beiden Begriffen ist der zeitliche Bezug.
Gründe für die Durchführung von Instandhaltungen sind die Vermeidung von Fehlern und Ausfällen. Ein Fehler bezieht sich auf einen Zustand wohingegen ein Ausfall sich auf ein Ereignis und den Verlust einer Fähigkeit bezieht. Beide betreffen aber eine Nichterfüllung eines Objektes, eine Funktion zu erfüllen. Ein Fehler kann also aufgrund eines Ausfalls auftreten [11].

2.3.1 Instandhaltungsarten

Zu Fehlervermeidung oder -beseitigung können verschiedenen Arten der Instandhaltung herangezogen werden. Eine Übersicht inklusive der Zusammenhänge zwischen den Arten ist in Abbildung 2.4 zu finden. Neben der bereits erläuterten Verbesserung sind als Hauptkategorien die korrektive Instandhaltung und die präventive Instandhaltung zu erwähnen. Im Laufe dieser Arbeit werden die jeweiligen englischen und deutschen Ausdrücke der Instandhaltungsarten als Synonyme verwendet.
Die korrektive Instandhaltung (engl. corrective maintenance) ist eine konventionelle Instandhaltungsart. Diese wird grundsätzlich nach einem Ausfall durchgeführt und kann entweder sofort oder nach einiger Zeit erfolgen. Diese Instandhaltungsart wird auch als reaktive Instandhaltung bezeichnet. Das Motto der reaktiven Instandhaltung ist „run-to-failure“. Sie erfolgt nach einem mehr oder weniger zufälligem Ereignis als korrektive Maßnahme. Im Fokus ist dabei die Reparatur des Systems [11], [26], [28].

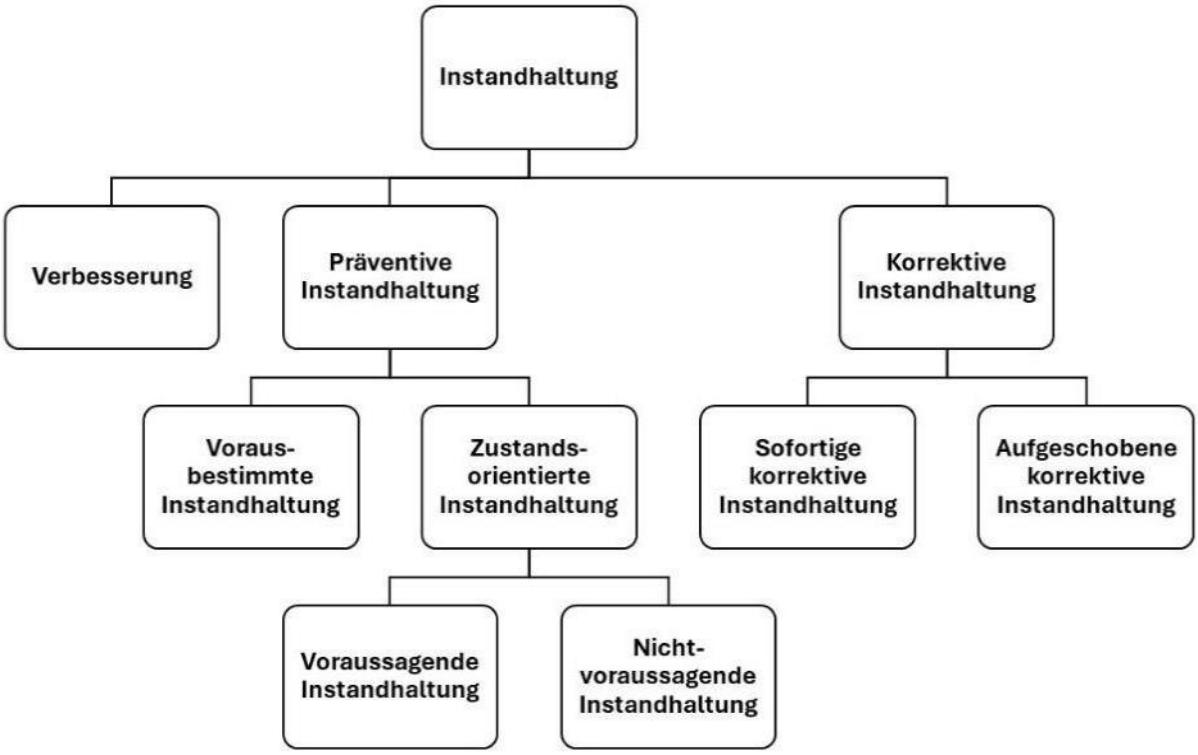


Abbildung 2.4 Instandhaltungsarten nach DIN EN 13306

Eine Instandhaltung vor einem Ausfall durchzuführen, wird als präventive oder auch proaktive Instandhaltung bezeichnet. Die präventive Instandhaltung untergliedert sich in die vorausbestimmte und in die zustandsorientierte Instandhaltung (engl. Condition-based maintenance). Als vorausbestimmt wird sie betitelt, wenn nach festgelegten Zeitabständen oder nach einer festgelegten Anzahl von Nutzungseinheiten, wie z. B. Betriebsstunden oder gefahrenen Kilometern, basierend auf historischen Daten eine Instandhaltung durchgeführt wird. Eine Zustandsermittlung wird im Vorfeld nicht durchgeführt. Eine mögliche Restnutzungsdauer von Komponenten wird also kategorisch nicht berücksichtigt [3], [11], [26].

Im Gegensatz steht die zustandsorientierte Instandhaltung. Von ihr spricht man, wenn eine Instandhaltungsmaßnahme in Abhängigkeit vom Zustand der Maschine durchgeführt wird. Dies kann auf Beobachtungen z. B. von Bedienern, Angestellten oder auch durch eine kontinuierliche Sensorüberwachung von Systemparameter wie Vibrations- oder Temperaturdaten erfolgen. Die zustandsorientierte Instandhaltung unterteilt sich in die voraussagende (engl. predictive maintenance) und auf die nichtvoraussagende Instandhaltung. Unterschieden werden diese beiden Arten, durch eine zusätzliche Prognose zum Fortschreiten des Maschinenabbaus. Die Vorhersage fußt auf wiederholten Analysen oder bekannten Eigenschaften und Bestimmung von bedeutenden Parametern, welche kennzeichnend für den Abbau des Objektes sind. Auf Basis dieser Daten kann eine Klassifikation des Zustands erfolgen [3], [11], [26].

Eine Klassifikation des Zustands der Maschine oder Anlage ist für diese Arbeit von besonderer Relevanz. Daher wird im weiteren Verlauf dieser Arbeit die nichtvoraussagende Instandhaltung sowie die korrektive Instandhaltung nicht weiter thematisiert.
Aus unternehmerischer Perspektive ist eine Entscheidung hinsichtlich der Instandhaltungsart, im Sinne einer Strategie zu treffen. Dazu sind Vor- und Nachteile zu berücksichtigen. Ausgewählte Vor- und Nachteile sind in der nachstehenden Tabelle zu finden sind.

Tabelle 2.1 Instandhaltungsarten Vor- und Nachteile in Anlehnung an [3], [19], [39]

	Vorteile	**Nachteile**
Korrektive Instandhaltung	- Vollständige Ausnutzung der Verfügbarkeit - Geringe Wartungskosten - Geringer Personaleinsatz - Kein Diagnoseaufwand	- Betriebssicherheit wird beeinträchtigt und ggf. Folgeschäden (finanziell & materiell) - Hohes Ausfallrisiko und monetäre Ausfallverluste - Schwierig planbar - Ineffiziente Nutzung von Personal
Vorausbestimmte Instandhaltung	- Gute Verfügbarkeit - Vermeidung plötzlicher Ausfälle - Gute Planbarkeit - Skalierbarer Ressourcenbedarf	- Risiko des zu frühen Austausches - Zunahme des Instandhaltungsaufwands - Höherer Planungsaufwand - Instandhaltungsbedingte Stillstandzeiten - Ausfallverhalten nicht immer statistisch vorhersehbar (Zufallsausfälle)
Zustandsorientierte Instandhaltung	- Optimale Ausnutzung unter Beachtung des tatsächlichen Zustands - Reduziertes Ausfallrisiko - Vermeidung von Stillständen - Optimales Verhältnis von Verfügbarkeit, Sicherheit und Wirtschaftlichkeit	- Hoher Planungs- und Instandhaltungsaufwand - Investitionen in technische Infrastruktur - Risiko der Nichterkennung von Störungen

Des Weiteren sind die Instandhaltungskosten bei der Auswahl der Instandhaltungsart bzw. der Instandhaltungsstrategie zu berücksichtigen. Deutlich wird dies durch die Abbildung 2.5. Aus ihr geht hervor, dass eine zustandsorientierte Instandhaltung unter Berücksichtigung der Verfügbarkeit und Ausfälle im optimalen Bereich liegt.

Aus den Vor- und Nachteilen wird grundsätzlich ersichtlich, dass eine zustandsorientierte Instandhaltung als Optimum anzustreben ist. Gerade mit der Entwicklung neuer Technologien und Trends im Zusammenhang mit der Industrie 4.0, ist es möglich einige der aufgezeigten Nachteile zu kompensieren. Verbesserungen bei den Technologien in Bereichen der Sensorik, Datensysteme oder Cloudinfrastruktur erzielen immer höhere Leistungen, kürzere Verarbeitungszeiten und sinkende Preisniveaus. Nach [28] geben 90 % der Firmen, die in eine vorausschauende Instandhaltung investiert hatten, an, dass sich binnen eines Zeitraumes von zwei Jahren die Investitionen moderat bis signifikant rentiert haben. Daher liegt der Fokus dieser Arbeit auf der zustandsorientierten Instandhaltung, genauer auf der vorausschauenden Instandhaltung.

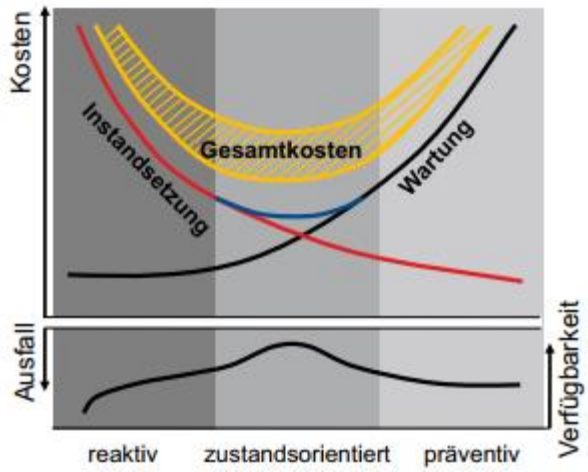


Abbildung 2.5 optimaler Punkt der Instandhaltungsstrategie [34]

2.3.2 Condition based maintenance/ zustandsorientierte Instandhaltung

Im nachfolgenden Kapitel soll näher auf die zustandsorientierte Instandhaltung eingegangen werden. Als Voraussetzung für eine erfolgreiche zustandsorientierte Instandhaltung muss eine Zustandsüberwachung (engl. Condition monitoring) erfolgen. Ziel des Condition monitoring ist es den aktuellen Zustand einer Maschine zu überwachen und zu analysieren, um eine Vorhersage des nächsten Zustandes zu erreichen. Dadurch ist es möglich ggf. präventive Maßnahmen zu ergreifen, sodass Ausfälle vermieden und Fehler nicht eintreten und eine vorausschauende Instandhaltung (engl. predictive maintenance, (PdM)) durchgeführt werden kann.

Für das Condition monitoring lassen sich nach [26] zwei große Arten identifizieren: Zum einen das inspektionsbasierte Monitoring und zum anderen das kontinuierliche Monitoring, worunter auch das sensorüberwachten sowie das online bzw. real Monitoring fällt.
Das inspektionsbasierte Monitoring basiert auf einer nach Zeitintervallen durchgeführten Datensammlung. Die Intervalle sind im Gegensatz zu der vorausbestimmten Instandhaltung nicht im Vorfeld festgelegt und werden je nach Zustand der Maschine angepasst. Der Zustand der Maschine oder Anlage wird zu einem Zeitpunkt ermittelt und kann sensorbasiert erfolgen [26].
Das kontinuierliche Monitoring setzt hingegen auf eine kontinuierliche Erhebung von Daten. Diese erfolgt in der Regel unter Verwendung von Sensoren. Die gesammelte Datenmenge ist dabei wesentlich höher als bei der inspektionsbasierten Überwachung. Zur kontinuierlichen Überwachung lässt sich auch das online bzw. real Monitoring zählen. Dabei werden die Daten der Sensoren kontinuierlich überwacht und in Echtzeit ausgewertet [26].

2.3.3 Predictive maintenance/ voraussagende zustandsorientierte Instandhaltung

Gerade bei zufällig auftretenden Ausfällen und daraus resultierenden Fehlern würden herkömmliche Instandhaltungsarten überwiegend fehlschlagen [26]. PdM bringt u. a. Benefits im Bereich der Sicherheit, Kostenreduktion und Zuverlässigkeit mit sich. Nach [39] ist es möglich Kosteneinsparung im Bereich der Instandhaltung von 25 – 35 % zu erreichen, Stillstände um 70 – 75 % sowie Stillstandzeiten um 35 – 45 % zu senken und die Produktivität um 25 – 35 % zu steigern. In der Literatur werden Condition Based Maintenance (CBM) als auch Prognostics and Health Management (PHM) dem Bereich PdM zugeordnet und teilweise als Synonym verwendet. Allerdings ist in der Literatur nicht eindeutig, wie diese Kategorien zusammenpassen oder sich unterscheiden. Teilweise werden diese Kategorien auch als Erweiterung der predicitive maintenance beschrieben [24]. Darüber hinaus ist in der Literatur als Erweiterung des predicitve maintenance auch die prescriptive maintenance zu finden. Dabei werden Inputparameter des überwachten Systems anhand der gemessenen Sensordaten optimiert. Im Detail gibt die prescricptive maintenance eine Beschreibung ab, was bei einem Ereignis getan werden soll und wie es getan werden soll [28]. Im weiteren Verlauf wird auf diese Instandhaltungsart nicht weiter eingegangen. Der Vollständigkeit wegen seien sie hier kurz erwähnt.
Auf Basis der Datengrundlage, die durch das Condition monitoring in Kapitel 2.3.2 ermittelt wird, ist es dann möglich eine Zustandsklassifikation des betrachteten Systems durchzuführen. Der Zustand einer Maschine oder Anlage ist allerdings meist schwierig zu klassifizieren. Klassifikationen basieren entweder aus einem großen Erfahrungsschatz der Mitarbeitenden bzw. den im Laufe der Zeit eingetretenen Ereignissen (wissensbasiert) oder auf Modellen, die eine Vorhersage (datenbasiert oder auf Basis von physikalischen Modellen) treffen können. Der

Zusammenhang dieser Arten der voraussagenden zustandsorientierten Instandhaltung ist in Abbildung 2.6 zu finden.

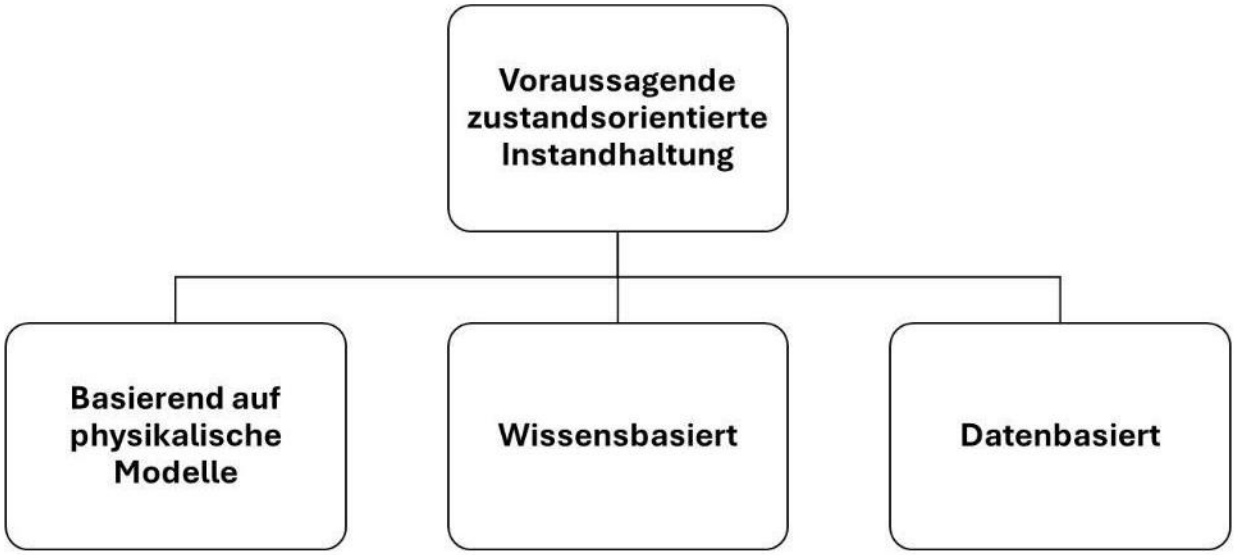


Abbildung 2.6 Arten der voraussagenden zustandsorientierten Instandhaltung [28]

Wissensbasierte Instandhaltung fußt entweder auf Regeln, die getroffen wurden oder Situationen, die bereits eingetreten sind, wofür Lösungen existieren [24]. Erfahrungswerte im Sinne eines gepflegten Datensatzes sind meist schwierig ermittelbar oder liegen kaum oder wenn teilweise unvollständig vor. Ebenso sind Mitarbeitenden, die die verwendeten Maschinen, ihre Muster und Betriebszustände auswendig kennen, nur noch in seltenen Fällen, bedingt durch den demografischen Wandel und den Fachkräftemangel [5], in Betrieben anzutreffen. Ebenso ist es schwierig Vorhersagen aufgrund von Erfahrungswerten zu treffen. Daher ist es naheliegend physikalische oder auch datenbasierte Modelle für eine zustandsabhängige Instandhaltung, insbesondere für eine bessere Unterstützung der Instandhalter zu verwenden. Hinzu kommt, dass Sensorik durch die Entwicklungen der Industrie 4.0 erschwinglicher werden und Datenverarbeitungssysteme immer höhere Leistungen erzielen.

Physikalische Modelle basieren auf den Gesetzen der Physik und erfordern ausgezeichnete Fähigkeiten der Berechnung sowie Modellierung des physikalischen Verhaltens der überwachten Anlage oder Maschine. Erforderlich ist ebenso eine genaue Kenntnis über das Verhalten des Systems, um ein physikalisches Modell zu generieren. Diese Modelle werden weitestgehend für Simulationen während der Entwicklung von einzelnen Komponenten oder Systemen eingesetzt. Umgebungs- und Nutzungsparameter wie Temperatur, Druck, Luftfeuchtigkeit, etc. werden daher weitestgehend angenommen und entsprechen nicht den realen Umgebungsbedingungen. Darüber hinaus sind nicht für alle physikalischen Phänomene exakte Modelle verfügbar, sodass

die Genauigkeit bei einem realen Einsatz nicht hinreichend ist. Eine Anpassung von physikalischen Modellen an Realbedingungen ist darüber hinaus meist mit hohen Kosten verbunden. Mit steigender Komplexität der Anlagen und Maschinen steigt ebenso die Komplexität der physikalischen Modelle. Gerade in Anbetracht der zunehmenden Komponenten innerhalb eines Produktes im Rahmen der Industrie 4.0 steigt die Schwierigkeit der adäquaten Abbildung des Realzustandes des Systems [5], [24], [28].

Eine Alternative bieten datenbasierte Verfahren durch eine höhere Datenverfügbarkeit und bessere Datenverarbeitung bedingt durch die Verbesserungen in den Technologiefeldern der Industrie 4.0, siehe Kapitel 2.1.

Mittels datenbasierter Verfahren ist es möglich das Verhalten des Systems zu bestimmen, ohne den genauen physikalischen Zusammenhang zu kennen [24], [28]. Datenbasierte Verfahren lassen sich nach Abbildung 2.7 in Deep Learning (DL), Machine learning (ML) bzw. maschinelles Lernen, statistische Methoden und stochastische Methoden unterteilen. Aufgrund der sprachliche Gängigkeit der englischen Bezeichnungen werden diese im weiteren Verlauf der Arbeit weiterverwendet.

Statistische Verfahren bzw. Modelle zielen darauf ab das Verhalten von Zufallsvariablen auf Basis von historischen Daten zu analysieren. Beispielhaft sind hier die Regressionsanalyse, die Autoregression und bayessche Modelle zu nennen. Hauptnachteil von statistischen Verfahren ist, dass sie eine große Menge historischer Daten benötigen, um ein zuverlässiges Modell mit einer ausreichenden Gewissheit zu bilden [24].

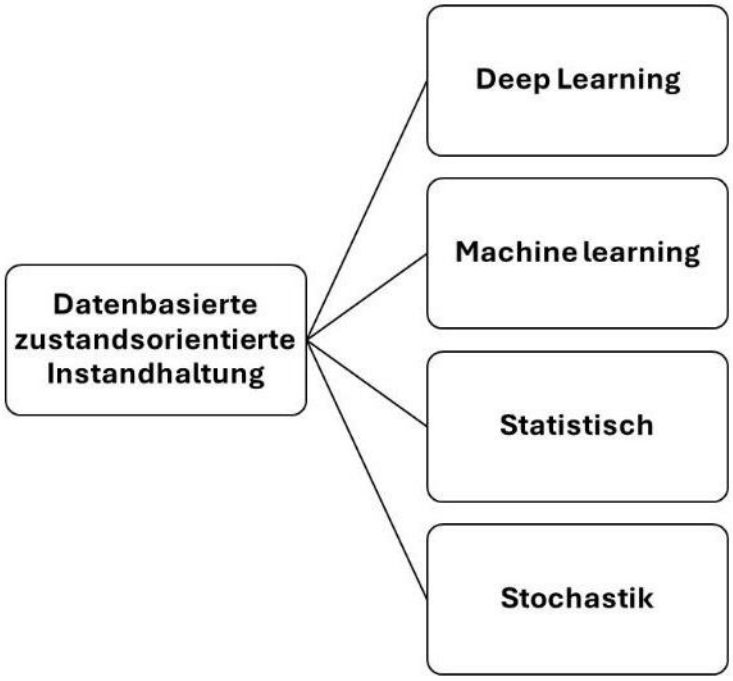


Abbildung 2.7 datenbasierte zustandsorientierte Instandhaltung nach [28]

Stochastische Modelle sind Wahrscheinlichkeitsmodelle, die auf die Untersuchung der Entwicklung von Zufallsvariablen im Verlauf der Zeit abzielen. Nach [24] wurden Gauß´sche Prozesse, Markov Ketten, und Levy Prozesse als Hauptmethoden der stochastischen Modelle identifiziert. Nachteilig bei diesen Modellen sind hohe Anforderungen an die Rechenleistung, erforderliche fortgeschrittene mathematische Kenntnisse und auftretende Unsicherheiten, die gesteuert werden müssen [24], [28].

Wie zuvor erwähnt werden neben statistischen und stochastischen Modellen ML und DL angewandt. ML umfasst dabei die Anwendung von Algorithmen menschlichen Lernes zu imitieren. Ziel dahinter ist die Entdeckung von Mustern in großen Datenbeständen. DL ist dabei ein Teilbereich des maschinellen Lernens, welches tiefe Neuronale Netze (NN) benutzt. NN transformieren Dateneingänge abhängig von einer Aktivierungsfunktion. Der Lerneffekt tritt durch eine rückgängige Berechnung des Fehlers des Ergebniswertes und eine Anpassung von Gewichtungen ein. Es existieren verschiedenen Arten von NN, abhängig von ihrem Aufbau. ML oder DL benötigen keine genauen Kenntnisse über die Funktionsweise von technischen Systemen wie Maschinen oder Anlagen, was sie zu einem starken Werkzeug für PdM macht [24], [28].

Gegenstand dieser Arbeit ist aufgrund der oben genannten Gründe die datenbasierte Zustandsüberwachung mittels ML resp. DL. Auf diese Arten wird in den folgenden Kapiteln näher eingegangen.

2.4 Künstliche Intelligenz, maschinelles Lernen und Deep Learning

Bei dem Schlagwort künstliche Intelligenz (KI) sind in der breiten Bevölkerung verschiedenste Vorstellung vorhanden. Bilder von menschenähnlichen Robotern aus Science-Fiction Romanen oder Filmen werden oft in Zusammenhang mit KI (engl. artificial intelligence) (AI), gebracht [12], [29]. Daher ist es sinnvoll eine Beschreibung von KI, DL und ML zu geben.

„Künstliche Intelligenz bezeichnet die Eigenschaft von technischen Systemen, Aufgaben zu lösen, deren Lösung durch einen Menschen Intelligenz erfordert“ [29]. Intelligenz setzt dabei die Fähigkeit des Lernens, der Schlussfolgerung, des Verstehens, des Überprüfens, des Vorausschauens und der Beurteilung voraus. Computer haben allerdings nur eine begrenzte Möglichkeit sich zu einem intelligenten System zu entwickeln. Auf Basis von Daten und mathematischen Prozessen werden diese manipuliert und ausgewertet, sodass ein echtes Begreifen oder Verstehen nicht möglich ist, jedoch aber die Abbildung von Mustern der zugeführten Daten. Daher ist es bei Entwicklungen bislang das Ziel menschliche Intelligenz bestmöglich nachzuahmen bzw. zu simulieren [32].

2.4.1 Machine Learning

Wie bereits zuvor erwähnt ist ML ein Teilbereich der KI. ML ist dabei der Vorgang der automatisierten Ableitung von möglichst allgemeingültigen Regeln aus einem Datensatz mit wenig oder keiner Unterstützung. Die Einsatzgebiete sind groß und gehen von Big Data, Robotik, Bilderkennung bis hin zu Spracherkennung. Für ML ist, wie bereits zuvor erwähnt, kein spezifisches Fachwissen über den zu untersuchenden Datensatz nötig. Darüber hinaus ist es für ML-Algorithmen möglich, verschiedene Dimensionen einzelner Sachverhalte zu berücksichtigen. Beispielsweise ist es für ML-Algorithmen relativ leicht Produkte anhand verschiedener Merkmale zu unterscheiden [9], [29], [45].

ML-Algorithmen lassen sich in die Kategorien „supervised learning“, „unsupervised learning“ und „reinforcement learning“ einteilen. Abbildung 2.8 veranschaulicht die Arten und gibt einige Beispiele von jeweiligen Algorithmen. Diese Aufstellung ist keineswegs abschließend und soll nur einen Auszug wesentlicher Methoden darstellen. Einige der aufgeführten Algorithmen sind für beide Arten, also unsupervised und supervised learning, einsetzbar. [9], [45]

Als unsupervised Learning werden Algorithmen verstanden, die allein auf Basis der Daten Muster erkennen oder Cluster bilden. Das Ergebnis (engl. Outcome) ist nicht bekannt. Das Hauptziel ist das Finden der unbekannten Cluster resp. das Clustering [9], [17], [45].

Beim supervised learning hingegen ist der Zielwert oder auch „Label“ bekannt. Daher spricht man auch von „gelabelten“ Daten. Untergebiete sind die Regression und Klassifikation. Der Fokus der Algorithmen liegt hier auf der Vorhersage zukünftiger Ereignisse. Auf Basis eines Trainingsdatensatzes mit gelabelten Daten, wird ein Modell trainiert, um später bei Verwendung von ungelabelten Daten eine Vorhersage zu treffen. Die mathematische Darstellung der gelabelten Daten ist in Formel 1 zu finden. x ist dabei die unabhängige Variable und y die abhängige. N steht für die Anzahl der Datenpaare. Der Lernprozess hat das Ziel optimale Werte für das spätere Modell zu finden, also jene, die eine Fehlerfunktion möglichst geringhalten. Die Auswahl der Fehlerfunktion hängt von der Problemstellung des Datensatzes ab [17], [29], [33], [45].

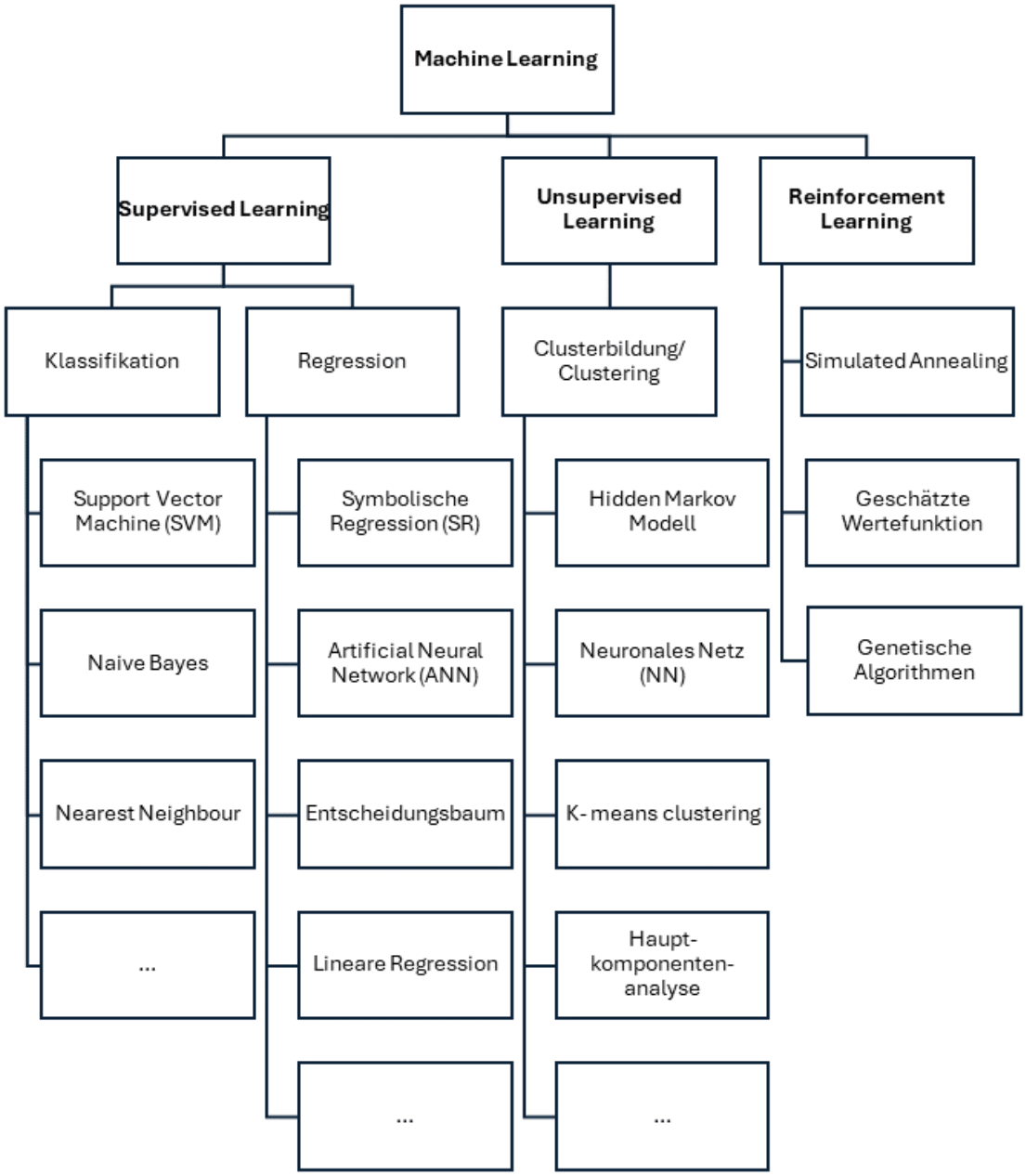


Abbildung 2.8 Arten Machine Learning nach [45]

$$D = \{(x_i, y_i)\}_{i=1}^{N} \quad \textbf{1}$$

Reinforcement learning oder auch bestärkendes Lernen genannt, beschreibt das Lernen durch ständiges Ausprobieren. Der Lernende muss eigenständig herausfinden, welche Parameter am besten geeignet sind, um den Datensatz zu beschreiben, anstatt es durch einen Algorithmus vermittelt zu bekommen. Das geschieht durch direktes Feedback von außerhalb bspw. bei einem Staubsaugerroboter durch seine Umgebung. Stößt er auf ein Hindernis, wird es nach und nach durch verschiedene Manöver umfahren [9], [17], [45].

Nach diesem Kapitel sind grundlegende Sachverhalte bezugnehmend auf KI und ML erläutert. In den folgenden Abschnitten werden bekannte ML-Algorithmen inkl. DL näher erläutert.

2.4.2 Deep Learning und Neuronale Netze

Eng mit dem ML verbunden, ist wie in Kapitel 2.3.3 beschrieben, das DL. DL ist eine spezielle Form des ML mittels Neuronalen Netzen. Genauer ist DL eine spezifische Form der Methode Neuronale Netze. Aus diesem Grund werden in diesem Kapitel zunächst die Funktionsweise von Neuronalen Netzen erklärt.

Neuronale Netze bzw. Artificial Neural Networks (ANN) ähneln vom Aufbau und der Funktionalität den menschlichen Neuronen, siehe Abbildung 2.9. ANN sind durch verschiedene, allerdings mit mindestens drei, siehe [45] Layer aufgebaut: den „Input Layer“, „Hidden Layer“ und den „Output Layer“. Jeder Layer bzw. Schicht oder Ebene, besteht aus einzelnen Neuronen. In der Abbildung 2.9 ist der Input Layer blau markiert, der Hidden Layer ist gelb und der Output Layer ist schwarz dargestellt. Je nach Aufbau, Typ, Problemstellung und Datenqualität ist es möglich, dass es mehrere Hidden Layer innerhalb eines ANN gibt. Der Input ist die unabhängige Variable x und y ist die Zielvariable. Im Rahmen von PdM sind x die Sensordaten und y die Zielvariable. Als Zielvariable wären die Zeit bis zum Ausfall oder einem Fehler, zukünftige Sensordaten oder auch das Auftreten einer Anomalie in den Daten möglich. Wenn die Information „feed-forward“ bewegt wird, bedeutet es, dass sie vom Input zum Output läuft. Jeder Output eines Neurons kann Input bei einem anderen sein. Die Neuronen werden, wie auch bei biologischen Neuronalen Netzen, durch einen Reiz aktiviert. Die Reizaktivierung bei einem künstlichen Neuron erfolgt über eine Aktivierungsfunktion und eine Gewichtung zwischen einzelnen Neuronen. Über ein vorheriges Training mit gelabelten Daten werden die Gewichtungen w_1 bis w_i durch das Modell erlernt. Ebenfalls durch das Training ermittelt wird der jeweiligen Schwellenwert. Dieser wird meist als b für engl. Bias angegeben. Sofern dann der Schwellenwert b überschritten wird, erfolgt die „Reizweitergabe“. Der Output für drei Inputwerte kann nach Formel 2, siehe [35], berechnet werden. Dabei ist f die Aktivierungsfunktion, w die Gewichte (engl. weights) und b der Schwellenwert als skalare Größe. Die Gewichte sowie der Bias können sowohl positiv als auch negativ sein. Deep Learning beschreibt Neuronale Netze, welche viele verdeckte Schichten aufweisen [27], [29], [33], [35], [45].

Spezifische DL- Algorithmen werden im weiteren Verlauf dieser Arbeit in Abhängigkeit der vorliegenden spezifischen Daten beschrieben.

Das grundsätzliche Vorgehen bei der Anwendung von Neuronalen Netzen für bzw. in Produktionssysteme gliedert sich in sechs Schritte. Diese sind in Abbildung 2.10 aufgeführt.

$$y = f(\sum_{i}^{3} w_i \times x_i + b) \quad \textbf{2}$$

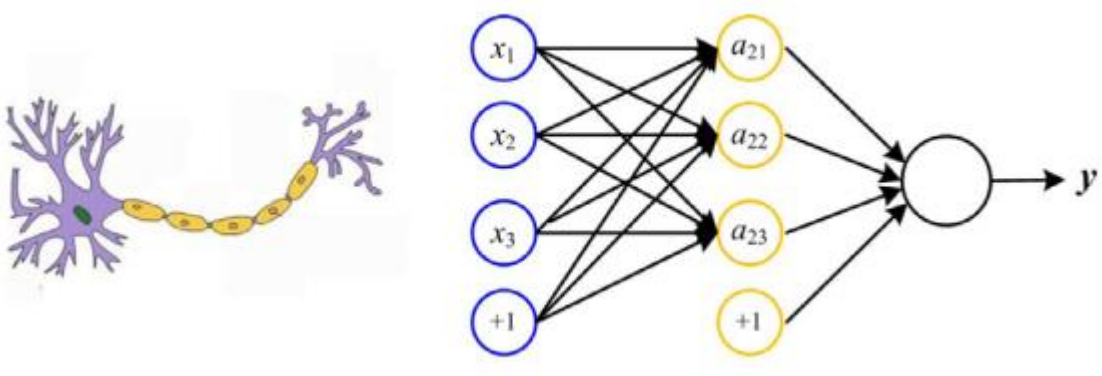


Abbildung 2.9 links: menschliches/ biologisches Neuron, rechts: ANN [35]

Der erste Schritt umfasst die Vorbereitung der Daten für das Training des NN. Es beinhaltet Datenaufbereitung. Fehlende Daten werden erzeugt, Ausreißer bereinigt und Parameter skaliert, um Stabilität im Lernprozess zu gewährleisten. Eine genaue Datenvorbereitung ist ein wichtiger Schritt, da die Qualität und Effektivität des Modells davon abhängt. In Schritt zwei, der Auswahl der Architektur, ist es notwendig auf Basis der Datenmerkmale und der Problemstellung die passende NN- Architektur bzw. den Aufbau des NN zu wählen. Danach ist das Training des gewählten NN durchzuführen. Das Ziel des Trainings ist es den Vorhersagefehler möglichst gering zu halten. Optimierungsalgorithmen werden verwendet, um die Parameter, also den Bias und die Gewichtungen, anzupassen. Für die Validierung werden separate, nicht im Training angewendete Daten verwendet. Die Effektivität und die Genauigkeit des Modells können so geprüft werden. Auf Grundlage der Ergebnisse in Schritt vier wird das Modell in Schritt 5 weiter optimiert und angepasst. Es ist nicht unüblich, dass z. B. die Anzahl der Layer, der Neuronen oder die Aktivierungsfunktionen weiter geändert werden. Das Ziel ist es die optimale Genauigkeit der Vorhersage zu erreichen und mögliche Fehler des Modells zu vermeiden. Nach der erfolgreichen Anpassung und Verifikation des Modells wird es dann in die Produktionsumgebung integriert. Nun sind automatische Prognosen auf Basis von z. B. Sensordaten möglich [27].

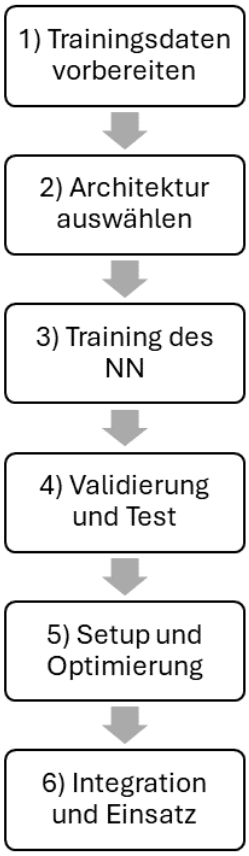


Abbildung 2.10 Prozessschritte Einsatz eines Neuronalen Netzes in Anlehnung an [27]

2.5 Zeitreihen

Erhobenen Daten von Produktions- oder Maschinenparametern liegen meist in Form von Sensormessungen abhängig von der Zeit vor. Daher ist es notwendig in diesem und den folgenden Kapiteln näher auf Zeitreihen und Zeitreihenanalysen einzugehen.

Eine Zeitreihe ist eine geordnete Menge von reellen Werten. Die Länge der Menge ist gleich der Anzahl der reellen Werte [14]. Dabei sind die Datenpunkt nach ihrem Auftreten im zeitlichen Verlauf geordnet. Anders ausgedrückt ist eine Zeitreihe eine geordnete Sammlung von Paaren von Messungen und Zeitpunkten. Dabei können Zeitreihen je nach Anzahl der gemessenen Variablen mehrere Dimensionen haben. Wenn z. B. die Temperatur und die Luftfeuchtigkeit gemessen wird, hat die Zeitreihe die Dimension zwei. Per Definition heißt eine Zeitreihe univariat, wenn nur eine Dimension vorliegt und multivariat, wenn mehrere Dimensionen vorliegen. Eine multivariante Zeitreihe setzt sich also aus mehreren univariaten Zeitreihen zusammen. Vorliegen können Zeitreihen als diskrete oder kontinuierliche Daten [13], [40]. Diskrete Zeitreihendaten beinhalten Beobachtungen oder Messungen, die auf bestimmte Kategorien oder Werte beschränkt sind. Im Vordergrund stehen einzelne Messungen zu einem bestimmten Zeitpunkt. In diese Kategorie würde z. B. die Anzahl von bestimmen Ereignissen abhängig von der Zeit fallen.

Im Gegensatz zu diskreten Daten stehen die kontinuierlichen Daten. Diese werden in regelmäßigen Zeitabständen aufgezeichnet und bilden dabei eine nahtlose und ununterbrochene Folge. Es werden also kontinuierliche Werte in einem bestimmten Bereich ermittelt. Als Beispiele könne der Aktienpreisverlauf über den Handelstag, Sensordaten oder Temperaturdaten innerhalb von Stunden, Tagen oder Minuten genannt werden [13], [31], [40].
Zeitreihen sind aufgrund ihrer natürlichen Ordnung so gut wie in jeder Aufgabe vertreten, die eine Art kognitiven menschlichen Prozess erfordert. Die Analyse von Zeitreihen kann also wertvolle Informationen liefern. Ebenso können durch eine breite Datengrundlage eine Basis für Prognosen geschaffen werden. Durch die Entwicklungen in der Industrie 4.0 ist die Datenverfügbarkeit stark gewachsen. Daher ist das industrielle Interesse an einer schnellen automatisierten Auswertung und Prognose von Zeitreihen naheliegend. Hierfür sind verschiedene Methoden des ML möglich. Diese werden in den nachfolgenden Kapiteln näher erläutert.

2.6 Regressionsmodelle

2.6.1 Lineare Regression

Das Ziel einer einfachen logistischen Regression ist es eine Gerade zu bilden, die möglichst vielen Datenpunkten möglichst nah kommt. Sofern der untersuchte Datensatz mehrere Einflussgrößen aufweist, ist eine Darstellung über eine mehrfache lineare Regression ebenfalls möglich. Bei einfachen Zusammenhängen, also z. B. zwischen zwei Variablen x und y, erfolgt die Modulation über eine Geradengleichung mit zusätzlicher Berücksichtigung eines zufälligen Fehlers u_i wie in Formel 3 dargestellt [15], [17].

$$y_i = a + bx_i + u_i, a, b \in \mathbb{R}, i = 1, 2, \dots, n \text{ [15]} \qquad \textbf{3}$$

Zentrale Annahmen für diese Verfahren ist, dass die Variable y von x linear abhängig ist sowie die Zufälligkeit des Fehlers u_i. b ist in dem Fall die Steigung der Geraden. Bei nicht linearen Zusammenhängen finden auch Funktionen mit Polynomen n.- Grades ihre Anwendung [15], [17].

2.6.2 Logistische Regression

Im Gegensatz zur linearen Regression sind bei der logistischen Regression als Label lediglich zwei mögliche Varianten vorhanden. Bei Klassifikationsproblemen ist es daher möglich einen metrischen Wert der Klasse A oder B zu zuordnen. Der Output ist immer binär auf Grundlage einer Wahrscheinlichkeit also null oder eins, sodass metrischer Input direkt transformiert werden kann. Für die Klassifikation wird keine lineare Funktion gewählt, sondern in der Regel eine spezielle Sigmoid- Funktion. In den meisten Fällen meint eine Sigmoid- Funktion einen

S- förmigen Graphen. Das Ziel der logistischen Regression ist es zukünftige auftretenden Ereignisse auf Basis vorliegender Daten vorherzusagen, also richtig zu klassifizieren. Einsatzgebiete sind z B. Die Klassifikation von Tumoren auf Basis ihrer Größe als gutartig oder nicht oder auch erhaltene E- Mails von unbekannten Absendern als Spam oder nicht [17], [45].

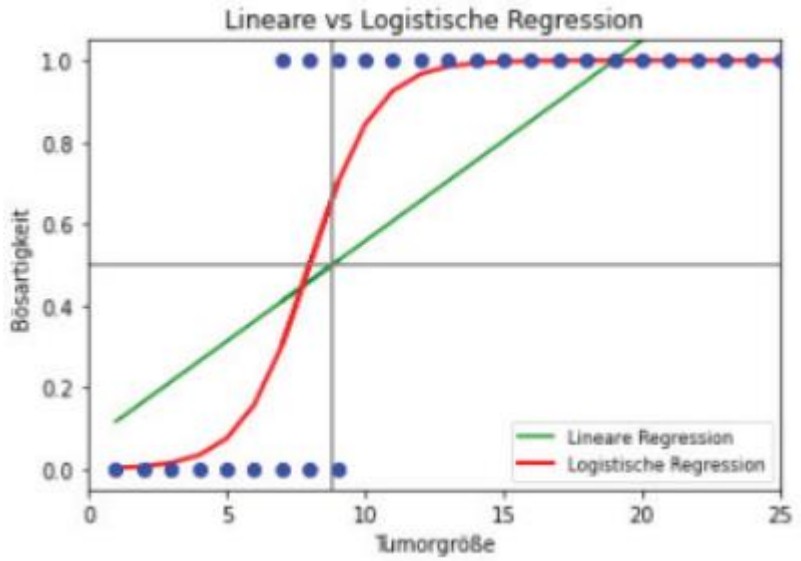


Abbildung 2.11 Lineare vs. Logistische Regression [17]

In Abbildung 2.11 ist ein Vergleich der logistischen und der linearen Regression am Beispiel der Tumorgröße und der Klassifizierung als gutartig oder bösartig dargestellt. Die lineare Regression (grüner Graph) klassifiziert in diesem Beispiel deutlich mehr Datenpunkte falsch als die logistische Regression (roter Graph). Grundsätzlich lässt sich bei binären Klassifikationsproblemen sagen, dass die logistische Regression bessere Klassifikationsergebnisse liefert als die lineare Regression [17], [45].

2.7 Machine und Deep Learning- Verfahren

Bislang wurden alle ML- Verfahren nur grob vorgestellt. In dem folgenden Kapitel sollen die verschiedenen und teilweise in Abbildung 2.8 aufgeführten Verfahren näher erläutert werden. Dabei wird der Fokus nur auf supervised ML- Verfahren gelegt. Wie zuvor in Kapitel 2.4.1 beschrieben können ML- Verfahren für Regression als auch für Klassifikation eingesetzt werden. Daher ist die Zuordnung in Abbildung 2.8 keineswegs unumstößlich, sondern eher als Übersicht zu verstehen. Über die bereits in Kapitel 2.4.2 im Zusammenhang mit DL vorgestellten NN werden in diesem Kapitel die Modelle Support Vector Machine (SVM), K- Nearest Neighbor, Entscheidungsbaum, Random Forest, Lineare Regression und Logistische Regression. Hinzu kommen spezifische DL- Mehtoden wie Convolutional Neural Network (CNN) und Recurrent

Neural Network (RNN). Die Auswahl der späteren Methode für den Anwendungsfall wird im späteren Verlauf dieser Arbeit getroffen.

2.7.1 Support Vector Machine

SVM gehört zu den überwachten Lernmethoden und ist für Mustererkennung, Klassifikationsprobleme und Regressionsanalysen einsetzbar. Das Verfahren basiert auf der Diskriminanzanalyse bei der durch eine Funktion, z. B. einer Gerade, die Datenpunkte in zwei Klassen eingeteilt werden. Die Grundlage von SVM beinhalten zwei Konzepte. Zum einen eine Maximierung des Abstandes der Trennfunktion zu den Datenpunkten und zu anderen eine Linearisierung des Raumes mit sog. Kernelfunktionen. Diese Konzepte dienen der Verbesserung der Generalisierung und Klassifikationszeit bei komplexen Mustern, allerdings geschieht dies auf Kosten der Trainingszeit. Veranschaulicht ist das Konzept der Maximierung des Abstandes in Abbildung 2.12. Abgebildet ist links eine Trenngerade mit kleinem Abstand zu den Datenpunkten, wobei die Klassen hier durch Dreiecke und Kreise dargestellt werden und rechts eine Trenngerade mit großem Abstand. Durch die Maximierung des Trennabstandes kann eine erhöhte Generalisierung bei der Klassifikation erreicht werden. Dies schließt zukünftige Zuordnungen von Datenpunkten in die richtige Klasse mit ein. Die Nutzung von Kerneln (mathematischen Operationen) hat das Ziel die ermittelte n-dimensionale Trennfunktion für komplexere Klassengrenzen zu transformieren, sodass die Trennung durch eine Gerade erfolgen kann. Die Suche nach einer optimalen Kernelfunktion wirkt sich verlängernd auf die Trainingszeit des Modells aus. Anhand der Geradenfunktion wird entschieden welcher Wert welche Klassen zu zuordnen ist [29], [45].

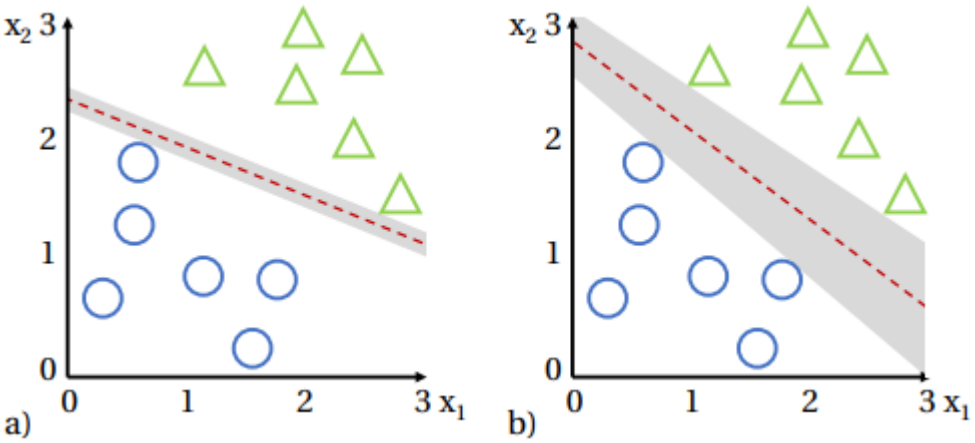


Abbildung 2.12 SVM Trenngerade a) mit geringem Abstand b) mit großem Abstand zu den Datenpunkten [29]

2.7.2 K- Nearest Neighbor

Der K- Nearest Neighbor Algorithmus wird bei Klassifikationsprobleme eingesetzt. Anhand von k benachbarten bekannten Datenpunkten wird bestimmt, ob ein unbekannter Datenpunkt zu einer bestimmten Klasse gehört. Der K- Nearest Neighbor Algorithmus kommt ohne klassische maschinelle Modellbildung aus, sondern versteht sich eher als Regel. Beispielhaft für k = 1 würde dann der nächste Nachbar die Klassifikation eines unbekannten Datenpunktes bestimmen. Wäre k = 4, würden die vier nächsten bekannten Datenpunkte die Klasse bestimmen. Sofern also für k ein kleiner Wert gewählt wird, ist das Ergebnis sehr stark von der direkten Umwelt des Datenpunktes und damit auch von möglichen Ausreißern abhängig. Auf der anderen Seite, wenn k groß bzw. zu groß gewählt wird, ist es möglich, dass sich ebenfalls eine negative Wirkung auf das Ergebnis eintritt. Bei nur einer kleinen Anzahl von einer Klasse und entsprechend großer Anzahl der anderen Klasse im bekannten Datensatz, würde dann bei der Klassifikation von unbekannten Datenpunkte bei einem großen k wahrscheinlich die Klasse mit der höheren Anzahl gewählt werden. Daher sollten in der Praxis verschiedene Werte für k getestet werden, um das bestmögliche Ergebnis zu erzielen. Für die Bestimmung der Distanz werden in der Regel die euklidische Distanz, siehe Formel 4 oder die Manhattan-Distanz, siehe Formel 5, verwendet. Durch die Berechnung jeder einzelnen Distanz ist die Berechnungszeit bei großen Datenmengen hoch [17], [29].

$$dist_{Euklid}(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2} \text{ [29]} \quad \textbf{4}$$

$$dist_{Manhattan}(p,q) = |p_1 - q_1| + |p_2 - q_2| + \cdots + |p_i - q_i| \text{ [29]} \quad \textbf{5}$$

2.7.3 Entscheidungsbaum/ decision tree

Das Verfahren Entscheidungsbaum basiert auf einer vorherigen Ermittlung von Schwellenwerten anhand dessen eine Klassifikation von Datenpunkten vorgenommen wird. In Abbildung 2.8 wurde das Verfahren der Regression zugeordnet, allerdings ist es auch für Klassifikationen einsetzbar. Der Entscheidungsbaum basiert grundsätzlich aus Knoten und Zweigen. Der Knoten repräsentiert den Zustand, in dem die Entscheidung zu treffen ist. Ein Zweig stellt ein mögliches Ergebnis der jeweiligen Entscheidung dar, sodass mehrere „wenn- dann"- Entscheidungen durchgeführt werden. Der erste Knoten wird Wurzelknoten genannt, die Knoten am Ende heißen Blattknoten und enthalten die Endergebnisse des Baumes. Sofern die geprüfte Eigenschaft auf einen Datenpunkt zutrifft, wird diese der entsprechenden Klasse zugeordnet. Im weiteren Verlauf wird dann dieser Datenpunkt nicht mehr berücksichtigt, sodass nur wenige Datenpunkte alle Stufen durchlaufen. Dieses Vorgehen ist als kaskadierte Klassifikation bekannt. In der Regel werden Trennfunktionen, im besten Fall Trenngeraden, als Entscheider bzw. Schwelle

eingesetzt. Wie bei allen Verfahren wird im Vorfeld ein Training des Algorithmus durchgeführt. Die grafische Veranschaulichung ist in der nachfolgenden Abbildung aufgeführt. Der Entscheidungsbaum ist rechts abgebildet [17], [29], [45].

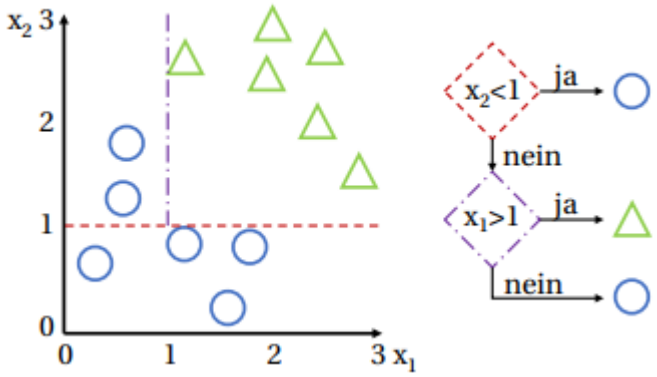


Abbildung 2.13 Entscheidungsbaum [29]

2.7.4 Random Forest

Der Random- Forest Algorithmus besteht aus mehreren Entscheidungsbäumen. Er wurde entwickelt, da bei Entscheidungsbäumen häufig das Problem der Überanpassung auftritt. Dadurch werden für andere Datensätze oder Anwendungsfälle keine validen Ergebnisse geliefert und der Einsatz bei anderen Szenarien liefert keine allzu genauen Ergebnisse. Random Forest gleicht diese durch die Schätzung von verschiedenen Entscheidungsbäumen und der Übernahme der am häufigsten auftretende Äste ins Zielmodell aus. Außerdem werden Random Forests nur mit einem Teil des vorliegenden Datensatzes trainiert, sodass dadurch auch eine Übergangspassung unwahrscheinlicher wird. Für Anwendungsfälle ist eine Anzahl von ca. zehn bis 100 Bäumen innerhalb eines Random Forest gängig. Zur Fehlervorbeugung existieren für Random Forest Algorithmen sog. „Bagging"- und „Boosting"- Verfahren. Von Bagging wird gesprochen, wenn selbständig erstellte Modelle gleichgewichtet mit in die Bewertung ein. Die häufigste Vorhersage wird als Endergebnis behalten. Beim Boosting wird jedes Modell. Also jeder Baum, nacheinander trainiert, sodass jedes Modell aus den Fehlern des vorherigen Modells lernt und sich dadurch anpasst. Ergo wird der erste Baum mehr Fehler und eine geringere Genauigkeit haben als der letzte [17], [29], [45].

Für das Boosting sind mehrere verschiedene Verfahren wie z. B. das AdaBoost-, das Gradient Boosting- oder das XGBoost- Verfahren. Diese seien an dieser Stelle der Vollständigkeit wegen erwähnt, werden aber im weiteren Verlauf dieser Arbeit nicht weiter berücksichtigt.

2.7.5 Recurrent Neural Networks

Bislang wurden NN allgemein vorgestellt. Dabei wurden Feed- Forward Netze und der allgemeine Aufbau von NN sowie DL präsentiert. Für spezielle Anwendungsfälle kann ein unterschiedlicher Aufbau bzw. Typ von NN bessere Ergebnisse liefern. Eine Form dieser Netze sind rekurrente neuronale Netze (engl. recurrent neural networks), kurz RNN. Im Verlauf dieser Arbeit werden sowohl der englische als auch der deutsche Ausdruck als Synonym verwendet.

RNN sind eine Erweiterung der eingangs erwähnten Feed- Forward Netze. Wie oben beschrieben weisen Feed- Forward Netze lediglich Vernetzungen der Vorgängerschicht mit der nächsten Schicht auf, wohingegen RNN weitere Arten von Rückkopplungen zulassen. Dies Arten sind in der Abbildung 2.14 dargestellt. Input sind x_1^1 und x_2^1. Die nachfolgende Schicht ist ein hidden layer und die darauf folgende Schicht sind Output layer, welche den Output A ausgeben. Unterschieden wird zwischen der seitlichen Rückkopplung (roter Pfeil), der indirekten Rückkopplung (grüner Pfeil) und der direkten Rückkopplung (blauer Pfeil). Bei der direkten Rückkopplung nutzt das Neuron den eigenen Ausgang als zusätzlichen Eingang. Von einer seitlichen Rückkopplung spricht man, wenn ein Neuron den Ausgang eines anderen Neurons derselben Schicht als Eingang nutzt [17], [29].

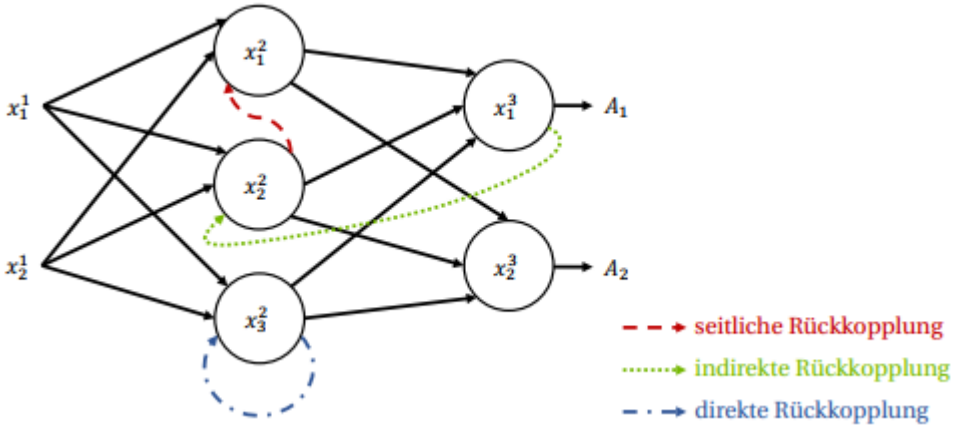


Abbildung 2.14 Arten der Rückkopplung bei RNN [29]

Die indirekte Rückkopplung liegt vor, wenn ein der Ausgang eines Neurons als Eingang eines anderen in einer beliebigen vorherigen Schicht genutzt wird. Nicht in Abbildung 2.14 dargestellt ist die vollständige Vernetzung. Hierbei wird jeder Ausgang jedes Neurons mit jedem Eingang jedes Neurons verbunden. Um keinen Zirkelbezug zu produzieren, werden für Rückkopplungen stets die Werte einer vorherigen Berechnung verwendet. Dadurch stehen teilweise ältere Ergebnisse bei einer zeitverzögerten Rückkopplung während der aktuellen Berechnung im Netzwerk als eine Art Gedächtnis zur Verfügung. Für Zeitreihenprognosen wird dies als sinnvolle Eigenschaft betrachtet. Einige NN weisen zur Nutzung der Speicherungseigenschaft

Speicherneuronen auf. Diese Zellen bzw. Neurone werden als „Recurrent Cell“ bezeichnet. Für die Recurrent Cells ist die Reihenfolge der Informationsübermittlung von erheblicher Bedeutung. Die Speicherung der Daten kann mit Lang- und Kurzzeitgedächtnis (engl. Long-/ short- term-memory) erfolgen. Die Optimierung der Parameter innerhalb des RNN erfolgen durch Rückpropagierung bzw. das Gradientenverfahren [17], [29], [33].

2.7.6 Convolutional Neural Networks

Convolutional Neural Networks sind eine weitere spezielle Art der von ANN. Diese Netzwerke sind dafür bekannt, Daten zu verarbeiten, die eine bekannte und tabellenähnliche Struktur aufweisen. Bilddateien in zwei Dimensionen durch Pixel als auch Zeitreihen in einer Dimension durch eine gitterähnliche Struktur weisen diesen Datentypen auf [17]. CNN werden im Bereich der Bilderkennung und der Bildverarbeitung bereits mit Erfolg eingesetzt und nun in der Forschung auch für Zeitreihen adaptiert [14]. Wie in Kapitel 2.4.2 beschrieben, ist zunächst ein Trainingsprozess notwendig, um das CNN auf die vorliegenden Daten anzupassen. Daher wird hier lediglich die grundsätzliche Funktionsweise eines CNN erläutert.
Aufgebaut ist ein CNN grundsätzlich aus einem Convolutional Block und einem Fully Connected Block. Dabei ist jeder dieser Blöcke aus verschiedenen Schichten zusammengesetzt. In Abbildung 2.15 ist der beispielhafte Aufbau eines CNN anhand der Erkennung einer Bilddatei eines Fahrzeugs dargestellt. Der Convolutional Block entspricht dem Teil, in dem das Feature Learning stattfindet. Hier sind die sog. Convolutional layer und Pooling layer zu finden, welche sich auch mehrfach hintereinander befinden können [17].

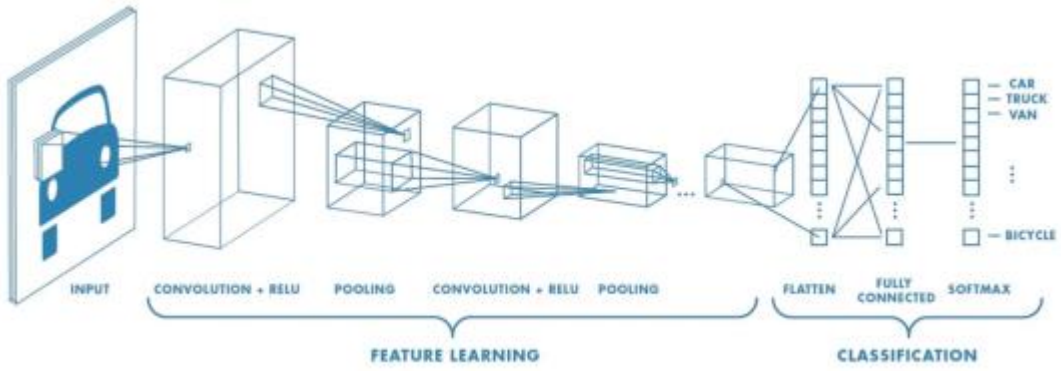


Abbildung 2.15 Beispielhafter Aufbau Convolutional Neural Network [29]

Der sog. Fully connected Block erfüllt die Aufgabe der Klassifizierung und besteht aus einem vollständig verbundenen einfachen Neuronalen Netz. Die Ausgabe des Convolutional Blocks stellt die Eingabe in den Fullly connected Block dar. Um die weitere Funktionsweise eines CNN zu erläutern, ist es sinnvoll die verschiedenen Layertypen näher zu erläutern. So dient der

Convolutional Layer dazu, sog. Features zu identifizieren. Hierfür werden sog. Filter, auch Kernel genannt, eingesetzt. Als Filter wird eine Matrix verstanden, die eine bestimmte Eigenschaft verkörpert. Bestimmte Sequenzen einer Zeitreihe oder Ausschnitte eines Bildes können so codiert werden, um in einer anderen Reihe oder Bild diese Sequenz bzw. diesen Ausschnitt wiederzuerkennen. In der Convolution wird die beschriebene Filtermatrix auf die Matrix der zu untersuchenden Zeitreihe resp. des Bildes angewendet. Durch eine Multiplikation aller Werte der Bild- und Filtermatrix gelingt es, eine Faltung resp. eine Reduktion der Matrixgröße durch eine Addition der Ergebnisse der Multiplikationen herbeizuführen. Am Ende bleibt so lediglich ein Zahlenwert übrig, was die Zeit des späteren Berechnungsprozesses verringert. Dieser Vorgang wird für alle Abschnitte wiederholt. Bei einer hohen Summe ist das gesuchte Feature sehr wahrscheinlich vorhanden, wohingegen bei einer kleinen Summe das Feature in dem untersuchten Abschnitt wahrscheinlich nicht vorhanden ist. Mögliche negative Werte werden durch die sog. ReLU (Rectified Linear Unit)- Aktivierungsfunktion gleich null gesetzt und positive Werte werden beibehalten. Der zweite Teil des Convolutional Bloks ist der sog. Pooling layer. Aus einem bestimmten Block der aus den Daten extrahierten Matrix werden z. B. die Maxima oder die Durchschnittswerte in eine kleine Matrix überführt. Das Ziel ist auch hier die Verkleinerung der Matrix, um Parameter zu verringern und dadurch Rechenleistung zu reduzieren. Der letzte Teil eines CNN ist der Fully Connected layer. Wie bereits erwähnt besteht dieser meist aus einem vollständig verbundenen einfachen NN. Die Ausgabe des NN ist ein n- dimensionaler Vektor, der die n- möglichen unterschiedlichen Klassen resp. Kategorien widerspiegelt [14], [17], [33].

2.8 Softwarevorstellung Python und Entwicklungsumgebung

Für die Analyse der vorliegenden Daten wird die in den späten 1980er Jahren entwickelte Programmiersprache Python gewählt. Diese ist industrieweit anerkannt und gilt als intuitiv. Darüber hinaus wurde Python bereits in diversen Analysen für PdM- Applikation eingesetzt. In Abbildung 3.16, einer Entscheidungshilfe für Analysetools für PdM, wird Python ebenfalls als präferierte Programmiersprache angeführt. Dazu sind für Python ebenfalls viele Analysepakete resp. Bibliotheken (engl. libraries) wie keras, tensorflow, numpy, pandas, scikitlearn oder SciPy vorhanden und können relativ einfach integriert werden. Diese Pakete beinhalten verschiedene, nutzbare vorgefertigte Funktionen oder Algorithmen, sodass nicht alles neu entwickelt werden muss [21], [24], [41], [45].

Des Weiteren gilt Python als einfach zu lernen und funktioniert so gut wie auf jedem Betriebssystem. Durch eine höhere Standardisierung in der Sprache, hat es bereits eine große Anwendung und Beliebtheit in der Data Science Community erlangt [21].

Zur einfacheren Nutzung von Python wird Anaconda.Navigator verwendet. Anaconda als Programmierumgebung bietet die Bereitstellung verschiedener Softwarepakete für Entwicklungsumgebungen [1], [32].

3 Durchführung

3.1 Vorstellung des Testaufbaus

Für ein breiteres Verständnis des später verwendeten Datensatzes, wird in diesem Kapitel der Ursprung der Messung vorgestellt. Gemessen wurden die Daten an einem Teststand der Hochschule für Technik und Wirtschaft Berlin und in [37] veröffentlicht. Der Testaufbau ist in Abbildung 3.1 dargestellt und besteht u. a. aus einem 230 V 50 Hz Wechselstrom Induktionsmotor mit einer abnehmbaren Ventilatorabdeckung. Der ursprüngliche Ventilator kann durch verschiedene 3D- gedruckte, gleichartige, modifizierte Ventilatoren ersetzt werden. Durch einen Metallschaft mit einem 3D-gedruckten Aufsatz ist der Motor mit einem Luftkompressor verbunden. Der Aufsatz ist mit einem Gewinde versehen, um später eine Unwucht mit einer Schraube zu simulieren. Der Aluminiumrahmen, der den Motor und den Kompressor hält, sind voneinander durch Polymerfedern entkoppelt. Die Messung erfolgt durch einen LSM9DS1-Sensor. Die Beschleunigung wird bei 400 Hz in x-, y- und z-Richtung gemessen. Verbunden ist der Sensor mit einem ESP32 Mikrocontroller, welcher die Messung mit einer Rate von ≤ 3 ms aufnimmt. Darüber hinaus ist eine Kamera mit einem Mikrophon vor dem Teststand aufgebaut. Über das Mikrophon wurde zusätzlich zu der Sensormessung, eine Messung des Audiosignals durchgeführt [37].

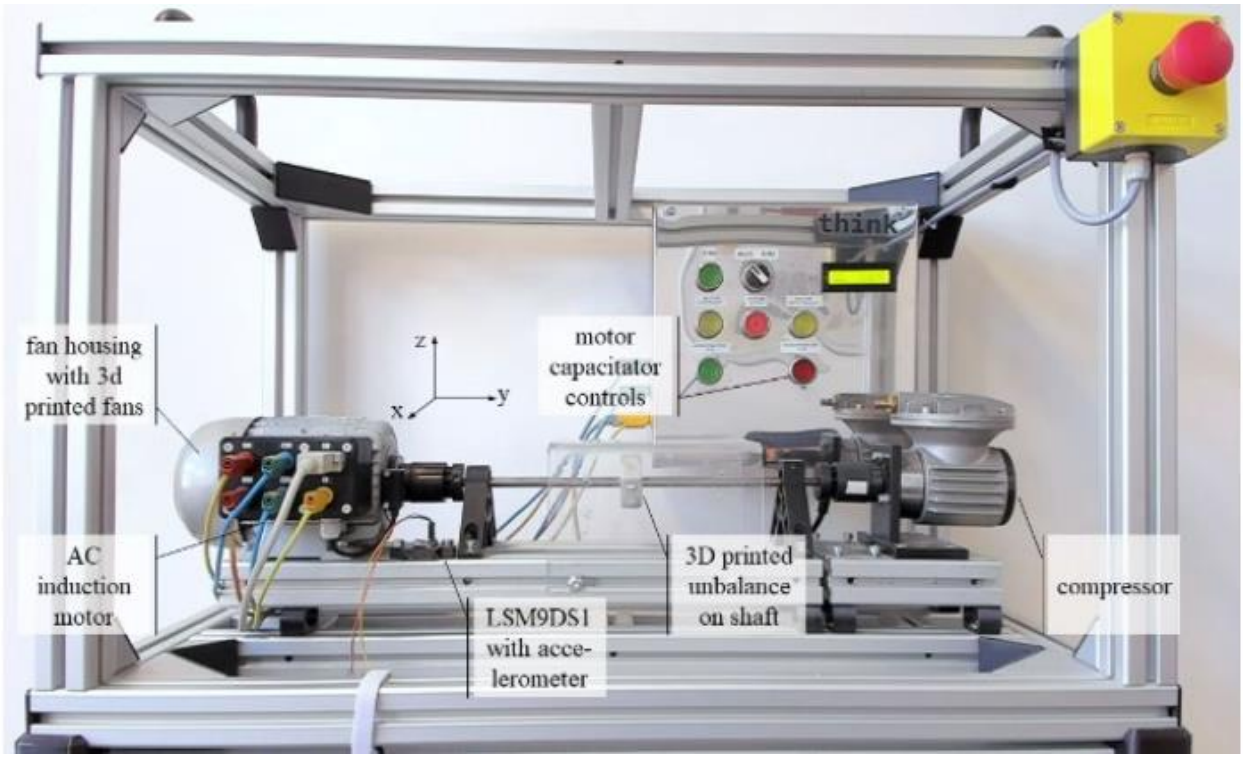


Abbildung 3.1 Teststand [37]

3.2 Vorstellung des Datensatzes

Der zu untersuchende Datensatz wurde über die online-big data Plattform kaggle abgerufen. Veröffentlicht wurde dieser in der Publikation „Structure-borne and Air-borne Sound Data for Condition Monitoring Applications“ [siehe 37]. Das Forschungsziel des Papers war es Realdaten für das Condition monitoring resp. für die Anwendung von ML- Methoden zu generieren, um den Mangel an frei verfügbaren Datensätzen entgegenzuwirken. Eine bebilderte Übersicht der Zustände ist in Abbildung 3.2 aufgeführt. Im Datensatz wird jedem Zustand eine ID sowie ein Label zugeordnet und eine kurze Beschreibung des Zustands gegeben, sodass hinterher eine eindeutige Rückverfolgbarkeit sichergestellt ist. Insgesamt wurden acht Zustände simuliert, welche eine Wartung wie z. B. ID6 oder ID7 oder eine Instandsetzung bspw. ID8 erfordern. Für jeden Betriebszustand wurden zehn Sekunden lang im Motorbereich die Beschleunigungen in x- , y- und z-Richtung sowie der Schall über das Kameramikrophon gemessen. Die Daten wurden danach mit einer Abtastrate von 300 Hz harmonisiert, sodass die gemessenen Daten in Roh- und harmonisierter Form vorliegen [37], [38].

Für die Generierung von Klassifikationsmerkmalen wurde eine „Short-Time-Fourier transform“ durchgeführt. Für den Körperschall, also die Vibrationsmessungen am Motor, wurde der Frequenzbereich zwischen 10 und 120 Hz in 5 Hz Intervalle komprimiert. Der Luftschall wurde in 25 Hz Intervalle im Bereich von 25 bis 2500 Hz zusammengefasst. Dabei überlappen die Fenster sich um 80 %, sodass pro Zustand 250 Beobachtungen in Zeitschritten von 0,04 s vorliegen. Als Zeitpunkt der Beobachtungen wird der Start der Fourier Transformation angegeben. Darüber hinaus zu erwähnen ist, dass die Autoren in [37] die Empfehlung für eine Verwendung des transformierten Datensatz für einen Condition monitoring Klassifizierungsalgorithmus geben [37]. Für eine weitere Betrachtung des Datensatzes werden grundsätzliche statistische Werte über Boxplots sowie die Graphen des „Frequency Feature“-Datensatzes bestimmt. Dieser Datensatz wird aufgrund der Empfehlung in [37] für den späteren Condition-Monitoring-Algorithmus verwendet. Für die Präsentation des Datensatzes wird die in Kapitel 2.8 vorgestellte Programmiersprache Python in der Version 3.12 mit der Anaconda.Navigator Version 2.5.2 und als Entwicklungsumgebung Spyder in der Version 5.4.3 verwendet.

Der Datensatz ist eine CSV-Datei und besteht aus 2000 Reihen und 172 Spalten. Ein Ausschnitt des in Python über die Bibliothek Pandas erstellten Dataframe ist in der nachstehenden Abbildung 3.3 aufgeführt. Die Bezeichnung „ID“ repräsentiert den entsprechenden Maschinenzustand ebenso wie „Label“. „Timestamp“ ist der Zeitpunkt zu dem die Fouriertransformation beginnt. Diese ist der Grund, warum in Zeile 1999 nicht 10 s stehen, sondern 9,96. Neben den für die Klassifikation notwendigen Spalten ID und Timestamp, sind die verschiedenen Frequenzbereiche als Spalten aufgeführt. xAcc, yAcc und zAcc stehen für die Beschleunigungen der jeweiligen Richtung für den jeweiligen Frequenzbereich, wie z. B. xAcc020Hz oder zAcc120Hz und werden

in mg, also 10^{-3} g angegeben [38]. „snd“ ist die Bezeichnung für die Audiodaten wie z. B. snd2500Hz und steht für Sound.

ID	Label	Description	Illustration
1	off	System is activated, but motor is turned off.	
2	on	Motor is running, powered with 50 Hz AC.	
3	cap	Motor capacitor is deactivated while motor is running.	
4	out	Compressor outlet valve is manually constricted.	
5	unb	A grub screw is inserted on one side of the shaft to create an unbalance.	
6	c25	Minor clogging of fan housing by attaching cover with 25 % reduced passage.	
7	c75	Major clogging of fan housing by attaching cover with 75 % reduced passage.	
8	vnt	Replacing the fan with defective fan that is missing 3 fan blades.	

Abbildung 3.2 Betriebszustände Teststand [37]

Es fällt auf, dass der Datensatz durch die große Anzahl an Spalten unübersichtlich ist. Daher wird der Datensatz über die pandas- Funktion „drop“ gekürzt, sodass nur die Daten des Vibrationssensors übrigbleiben.

	ID	Label	Timestamp	...	snd2450Hz	snd2475Hz	snd2500Hz
0	1	off	0.00	...	0.031688	0.007489	0.013386
1	1	off	0.04	...	0.018390	0.018145	0.007316
2	1	off	0.08	...	0.008647	0.034637	0.005606
3	1	off	0.12	...	0.006037	0.028448	0.014418
4	1	off	0.16	...	0.007217	0.014886	0.024236
...	..	...	...	...	...	...	...
1995	8	vnt	9.80	...	2.864246	4.875627	3.097777
1996	8	vnt	9.84	...	3.715002	7.698977	3.895627
1997	8	vnt	9.88	...	3.277185	9.806276	3.751800
1998	8	vnt	9.92	...	2.172725	7.631238	0.765262
1999	8	vnt	9.96	...	2.802488	0.846260	1.393357

[2000 rows x 172 columns]

Abbildung 3.3 Ausschnitt kompletter Datensatz

Dadurch wird eine Reduktion der Spalten auf 72 erreicht, sodass später Rechenkapazitäten reduziert werden können. Für eine erste Übersicht werden Boxplot- Diagramme sowie Verlaufs-Diagramme der jeweiligen Frequenzbereiche des jeweiligen Betriebszustandes erstellt. So ist es möglich einen Eindruck der vorliegenden Daten zu bekommen, um zu beurteilen, ob diese später noch bereinigt werden müssen, ob Extremwerte vorliegen oder Daten ggf. fehlen, ohne zu sehr in den Datensatz einzusteigen.

Für die Erstellung der Diagramme wurde die Python-Bibliothek „seaborn“ genutzt. Diese bietet ein umfassendes Angebot an Funktionen für die Visualisierung von statistischen Daten auf Basis von „matplotlib“. Vorteile dieser Bibliothek sind eine nachvollziehbare und übersichtliche Handhabung sowie eine enge Verbindung mit der pandas-Bibliothek [42].

Für eine erste Übersicht werden in den nachfolgenden Kapiteln die vorliegenden Daten in Form von Liniendiagrammen und Boxplots präsentiert. Dabei stellt jedes Diagramm den Verlauf der Beschleunigung über die Zeit für einen Frequenzbereich zu einem Betriebszustand (siehe Abbildung 3.2) dar. So soll überprüft werden, ob fehlende Werte vorhanden sind. Falls fehlende Werte auftauchen, ist es später für die später verwendeten Modelle schwierig eine Klassifikation durchzuführen.

Wie bereits oben beschrieben werden die Beschleunigungen auf die Erdbeschleunigung normiert in mg bzw. 10^{-3} g und die Zeit in Sekunden angegeben. Eine Darstellung von allen acht Betriebszuständen innerhalb eines Diagramms wird aus Gründen der Übersichtlichkeit als nicht zweckmäßig angesehen. Die Diagramme sind aufgrund der vorliegenden Datenstruktur nach Frequenzbereich geordnet. Dies resultiert pro Betriebszustand in 23 und in Summe in 69 Boxplots sowie 184 Liniendiagramme. Die farbliche Zuordnung der verschiedenen Richtungen sind der entsprechenden Legende zu entnehmen. Grundsätzlich sind die Diagramme in Anhang A zu finden. Bei Besonderheiten in den Betriebszuständen oder Frequenzbereichen ist das entsprechende Diagramm im Kapitel aufgeführt.

3.3 Frequenzbereiche 10 bis 120 Hz

Frequenzbereich 10 Hz

Wie bereits zuvor erwähnt soll am Beispiel der Beschleunigungswerte in x-, y- und z-Richtung der Datensatz kurz vorgestellt werden. Abbildung 7.1 in Anhang A zeigt die Liniendiagramme der Beschleunigungsrichtung des Frequenzbereiches 10 Hz. Bei Zustand ID1 fällt auf, dass die Werte im Vergleich zu den anderen Zuständen niedrig sind. Dies lässt sich durch den Betriebszustand des Teststandes erklären. Im Zustand mit ID1 ist der Motor nicht aktiv und das System lediglich aktiviert. Darüber hinaus stellen alle Verläufe Schwingungen dar, welche auf den ersten Blick keine Regelmäßigkeiten oder fehlende Werte aufweisen. Darüber hinaus ist die Schwingung in z-Richtung deutlich höher als in x oder y.

Die Beschleunigungen in y-Richtungen weisen höhere Werte auf als die der anderen beiden Richtungen. Es sind Werte über 3000 mg vorhanden. Verdeutlicht wird diese auch durch die Boxplots in der Abbildung 7.2. In y-Richtung sind deutlich größere Spannweiten vorhanden als bei den anderen Beschleunigungsrichtungen.

Das Boxplot-Diagramm xAcc10Hz in Abbildung 7.2 in Anhang A links oben, zeigt für in x-gerichtete Beschleunigung, bis auf den Zustand ID1, Mediane, repräsentiert durch den Strich innerhalb der in blau dargestellten Boxen, in einem ähnlichen Bereich um 500 mg auf. In z-Richtung sind die Mediane im Bereich 10 Hz über alle Betriebszustände mit Ausnahme von ID1 ähnlich.

Frequenzbereich 15 Hz

Die Verläufe der Beschleunigung im Bereich 15 Hz sind in Abbildung 7.3 in Anhang A aufgeführt. Sie weisen keine fehlenden Werte auf. Auffällig ist, dass die Werte in y-Richtung deutlich größer sind als in z- oder x-Richtung. Auch werden größere Maxima erreicht und die Spannweite ist deutlich größer. Verdeutlich wird diese auch durch die Boxplots in Abbildung 7.4 in Anhang A. Über alle Betriebszustände liegen alle Mediane in der jeweiligen Richtung mit Ausnahme von ID4 in x-Richtung, in einem ähnlichen Bereich.

Frequenzbereich 20 Hz

Die Verläufe der Beschleunigungen über die Zeit für den Frequenzbereich sind der Abbildung 7.5 im Anhang A zu finden. Anzeichen für fehlende Werte innerhalb dieses Bereiches sind nicht ersichtlich. Das Maximum der Beschleunigungswerte liegt bei ID7 bei über 5.000 mg. Darüber hinaus ist bei ID4 ein Anstieg der Beschleunigungen insbesondere in x-Richtung über die Zeit zu erkennen, siehe Abbildung 3.4. Bei den Zuständen ID4 bis ID8, sind die Beschleunigungen in x-Richtung deutlich höher als bei den anderen Richtungen.

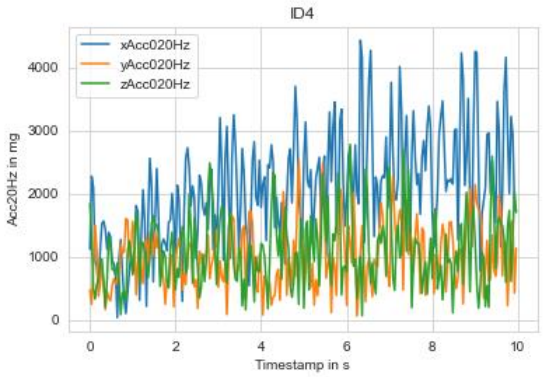


Abbildung 3.4 ID4 xAcc20Hz

In Zustand ID7 werden Beschleunigungen in x-Richtung über 5000 mg erreicht. Die Boxplots in Abbildung 7.6 verdeutlichen dies nochmal. Hier wird ersichtlich, dass nur ein Maximalwert über 5000 mg erreicht wird. Es fällt auf, dass der Frequenzbereich 20 Hz in x-Richtung bei Zustand ID4 eine größere Spannweite aufweist als bei den anderen Zuständen, siehe Abbildung 3.5. Außerdem sind die einzelnen Boxen, mit Ausnahme von ID4, schmal, was bedeutet, dass viele Werte zwischen dem unteren und oberen Quantil liegen.

In y-Richtung und z-Richtung in Abbildung 7.6 im Anhang A ist ersichtlich, dass im Vergleich zum normalen Betriebszustand ID2, die Mediane von ID3 bis ID8 leicht bis deutlich erhöht sind. Der Wertebereich von ID1 ist über alle Zustände und Richtungen signifikant niedriger.

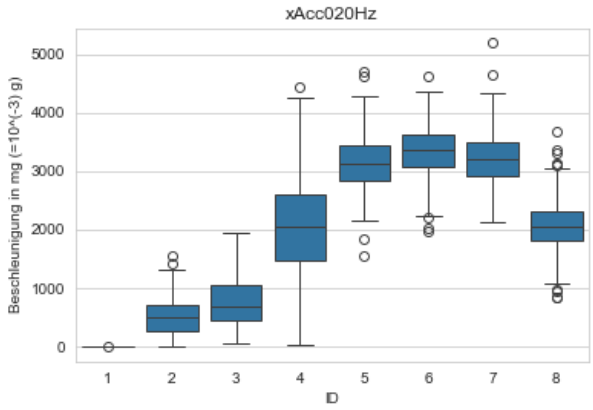


Abbildung 3.5 Boxplots xAcc20Hz

Frequenzbereich 25 Hz

In der Abbildung 7.7 im Anhang A sind die Verläufe der Beschleunigungen des Frequenzbereichs von 25 Hz aufgeführt. Die Verläufe geben keinen Aufschluss auf möglicherweise fehlende Werte. Bei den Zuständen ID4 bis ID8 sind die Beschleunigungen in x-Richtung deutlich höher als in y- und z.

Auffällig ist, dass bei ID4, siehe Abbildung 3.6, ein Anstieg der Werte in x-Richtung über die Zeit zu erkennen ist. Bei Zustand ID2 und ID3 sind die y-Werte im Vergleich zu x- und z- erhöht.

Darüber hinaus werden in x-Richtung Maximalwerte über 7.000 mg erreicht, wobei die Spannweite bei den Zuständen in x-Richtung gering ausfällt, siehe Abbildung 7.8 im Anhang A links oben.

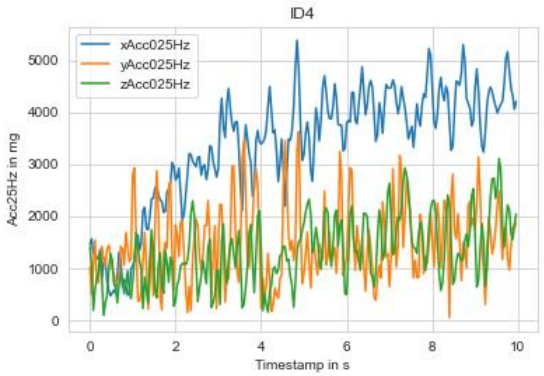


Abbildung 3.6 ID4 Acc25Hz

Eine Ausnahme bildet auch hier ID4. Die Spannweite als auch der Interquantilsabstand sind größer als bei den anderen Zuständen in x-Richtung. In y- und z-Richtung sind die Spannweiten grundsätzlich in den Frequenzbereich 25 Hz größer als in x-Richtung. Außerdem sind die Mediane höher als beim normalen Betriebszustand ID2, jedoch nicht in dem Maße erhöht wie in x-Richtung.

Frequenzbereich 30 Hz

Die für den Frequenzbereich 30 Hz zugehörigen Liniendiagramme der Beschleunigungen in x-, y- und z-Richtung sind in Abbildung 7.9 im Anhang A aufgeführt. Es fällt auf, dass außer für Zustand ID1 die Werte der Beschleunigungen in y- höher sind als in x- und z-Richtung. Es werden Maximalwerte bei den Zuständen ID4 bis ID6 von über 5.000 mg erreicht. Des Weiteren sind auch hier keine fehlenden Werte zu erkennen.

Außerdem sind die Spannweiten der y-Werte signifikant größer als in z- und x-Richtung, siehe Abbildung 7.10 in Anhang A. Lässt man Zustand ID1 außen vor, sind auch die Mediane aller weiteren Zustände im Vergleich zum Normalzustand ID2 erhöht.

Frequenzbereich 35 Hz

Abbildung 7.11 in Anhang A stellt die Verläufe der Beschleunigungen für den Frequenzbereich 35 Hz in Abhängigkeit von der Zeit dar. Es ist erkennbar, dass die Werte in y- deutlich höher liegen als in x- und y-Richtung. Auch der Wertebereich und Interquantilsabstand, veranschaulicht durch die Boxplots von yAcc35Hz in Abbildung 3.8 sind in y-Richtung deutlich größer. Dies schlägt sich auch in der Höhe der Mediane nieder. Der Maximalwert für diesen Frequenzbereich liegt bei über 7000 mg im Zustand ID5. Dieser weist in Abbildung 3.7 einen ansteigenden Verlauf in y-Richtung auf. Die weiteren Boxplots sind in Anhang A in Abbildung 7.12 zu finden.

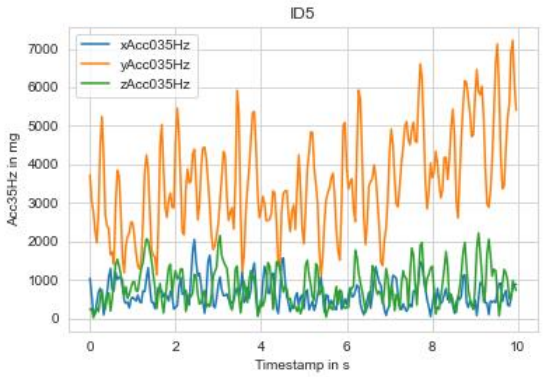


Abbildung 3.7 ID5 yAcc35Hz

Keine großen Schwankungen sind in diesem Frequenzbereich für die x-Richtung zu erkennen. Verdeutlicht wird diese durch den einen geringeren Interquantilsabstand, obwohl auch einige Ausreißer, verdeutlicht durch die kleinen Kreise, zu erkennen sind.

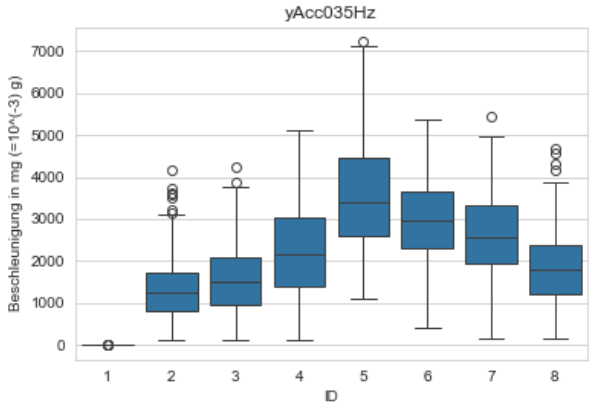


Abbildung 3.8 yAcc35Hz

Frequenzbereich 40 Hz

In Abbildung 7.13 Anhang A sind die Verläufe der Beschleunigungen über die Zeit für alle Zustände aufgeführt. Fehlende Daten sind nicht zu erkennen. Die Maximalwerte liegen bei über 4.000 mg bei Zustand ID5 in y-Richtung. In x-Richtung ist der große Wertebereich bei ID3 im Vergleich zu den anderen Zuständen auffällig, siehe auch Abbildung 3.9. Dieser erstreckt sich von ungefähr null bis größer als 3.000 mg. Die Mediane der einzelnen Beschleunigungen liegen bis auf Zustand ID3 in x-Richtung, in einem ähnlichen Bereich, siehe Abbildung 7.14, Anhang A. Darüber hinaus sind keine Auffälligkeiten zu erkennen.

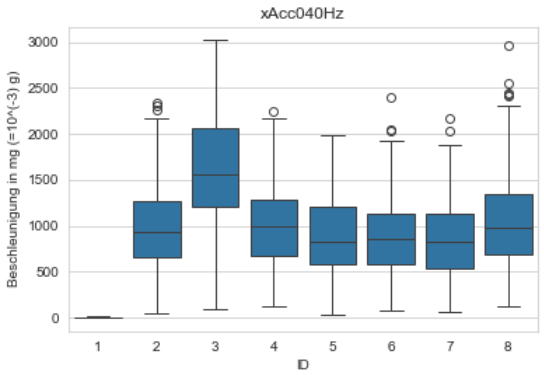


Abbildung 3.9 Boxplots xAcc40Hz

Frequenzbereich 45 Hz

Die Verläufe der Beschleunigungsrichtungen für den Frequenzbereich 45 Hz sind in Abbildung 7.15 in Anhang A dargestellt. Mit Ausnahme von ID1 verlaufen alle anderen Graphen, abhängig von der Beschleunigungsrichtung separat voneinander, sodass nahezu keine Überschneidungen auftreten. In x-Richtung werden bei ID3 Maximalwerte von über 12.000 mg erreicht. In y-Richtung liegt das Maximum bei über 4.000 mg und in z-Richtung bei über 8.000 mg bei ID8, siehe Abbildung 7.16 in Anhang A. Diese ist auch der Zustand bei dem in allen Beschleunigungsrichtungen die höchsten Werte erreicht werden.

Lässt man ID1 unbeachtet, liegt das Minimum in x-Richtung bei Zustand ID6 mit größer als 6.000 mg. In y-Richtung liegt das Minimum auch ohne Berücksichtigung von ID1 bei null und in z-Richtung bei ca. 2000 mg.

Frequenzbereich 50 Hz

Die Verläufe der Beschleunigungen für den Frequenzbereich 50 Hz sind durch die Liniendiagramme in Abbildung 7.17 in Anhang A dargestellt. Fehlende Werte in den Verläufen sind nicht zu erkennen. Mit Ausnahme von ID1, verlaufen die Beschleunigungsrichtungen weitestgehend ohne Überlappungen voneinander. Die Maxima in x-Richtung liegen bei über 14.000 mg, in y- bei über 4.000 mg und in z- bei über 12.000 mg. Über alle Zustände hat ID8 die höchsten Werte, siehe Abbildung 3.10. Verdeutlicht wird dies auch durch die Boxplots in Abbildung 7.18 in Anhang A. Auffällig ist, dass in x-Richtung lediglich bei ID4 ein Ausreißer zu erkennen ist. Aus den Boxplots geht des Weiteren hervor, dass die Mediane in x-Richtung, sofern ID1 unberücksichtigt gelassen wird, in einem nahezu identischen Wertebereich liegen. Für die z- und y-Richtung ist dies nicht zu erkennen, allerdings sind hier deutlichere Abweichungen zum Normalzustand ID2 ersichtlich.

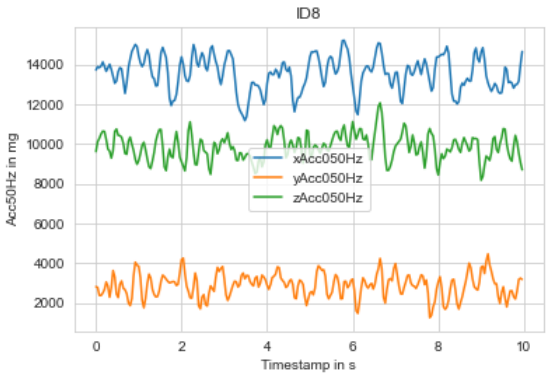


Abbildung 3.10 ID8 Acc50Hz

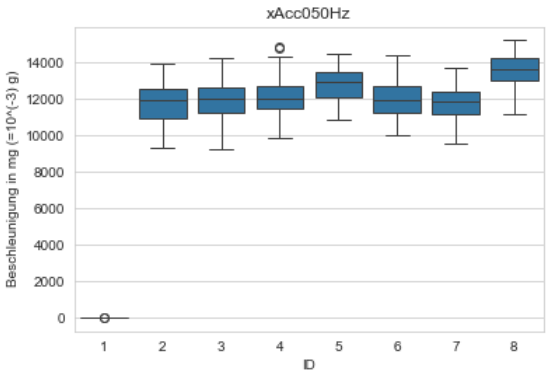
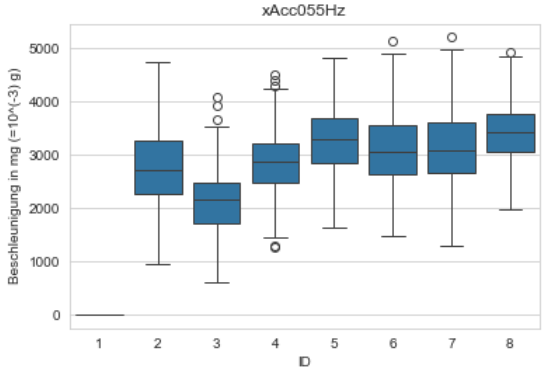
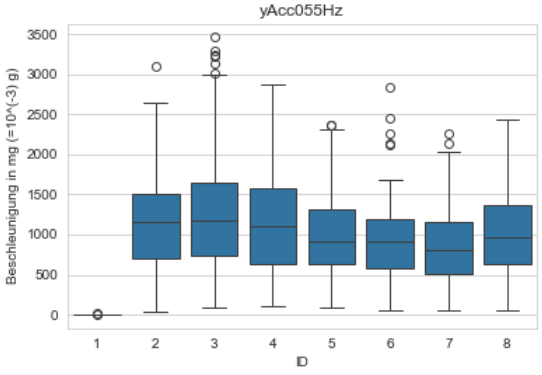


Abbildung 3.11 Boxplots xAcc50Hz

Frequenzbereich 55 Hz

Die Werteverläufe der Beschleunigungen im Frequenzbereich 55 Hz sind in der Abbildung 7.19 in Anhang A dargestellt. Es sind keine fehlenden Werte zu erkennen. Des Weiteren fällt auf, dass die Werte der drei Beschleunigungsrichtungen in einem ähnlichen Bereich liegen, wobei die Werte in x-Richtung bis auf ID1 durchweg höhere Beträge aufweisen. Verdeutlicht wird dies auch durch die Mediane in den Boxplot-Diagrammen in Abbildung 3.12. Darüber hinaus lassen sich keine Auffälligkeiten oder fehlende Werte erkennen.





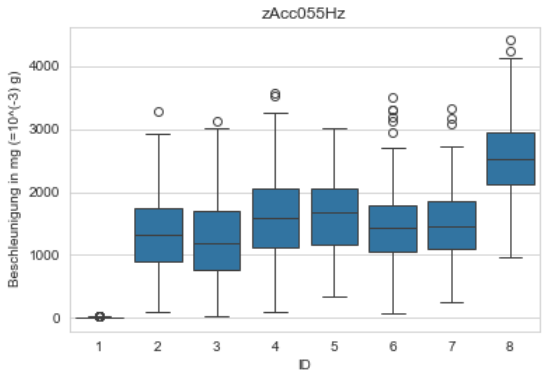


Abbildung 3.12 Boxplots Acc55Hz

Frequenzbereich 60 Hz

Abbildung 7.20 in Anhang A stellt die Verläufe der Beschleunigungen in x-, y- und z-Richtung für den Frequenzbereich 60 Hz je Betriebszustand dar. Die abgebildeten Graphen verlaufen in ähnlichen Wertebereichen. Verdeutlicht wird dies auch durch die Boxplots in Abbildung 7.21 in Anhang A. Die Mediane, also das 50 %-Quantil, ist für die jeweilige Beschleunigungsrichtung, bis auf ID1, in einem ähnlichen Bereich. Das Maximum in x-Richtung liegt knapp unterhalb von 2.000 mg in Zustand ID4, in y-Richtung mit über 3.000 mg bei ID3 und in z-Richtung mit über 2.500 mg bei ID8.

Frequenzbereich 65 Hz

Die für den Frequenzbereich 65 Hz Verläufe der Beschleunigungen sind in der Abbildung 7.22 in Anhang A als Liniendiagramme dargestellt. In Abbildung 7.23 ebenfalls in Anhang A zu finden sind die gemessenen Werte als Boxplots aufgeführt. In x-Richtung liegt das Maximum 3.000 mg bei ID4. Das Maximum in y-Richtung liegt bei mehr als 2.500 mg bei ID3 und in z-Richtung bei 3.000 mg bei ID7. Es fällt zudem auf, dass bei Zustand ID4 die Werte in x-Richtung im Vergleich zu den anderen beiden Richtungen deutlich höher sind. Darüber hinaus sind keine Besonderheiten auch hinsichtlich fehlender Werte erkennbar.

Frequenzbereich 70 Hz

In Abbildung 7.24 in Anhang A sind die Verläufe des Frequenzbereiches 70 Hz für alle gemessenen acht Zustände dargestellt. Der Maximalwert in diesem Frequenzbereich liegt mit ca. 4.500 mg in x-Richtung bei ID5, siehe auch Abbildung 7.25 in Anhang A. Der Verlauf des Graphen in x-Richtung bei Zustand ID3 zum Zeitpunkt von ungefähr 9 s und hebt sich von den anderen Betriebszuständen und Verläufen ab, siehe Abbildung 3.13. Grundsätzlich werden in x-Richtung, außer bei ID1, höhere Werte erreicht.

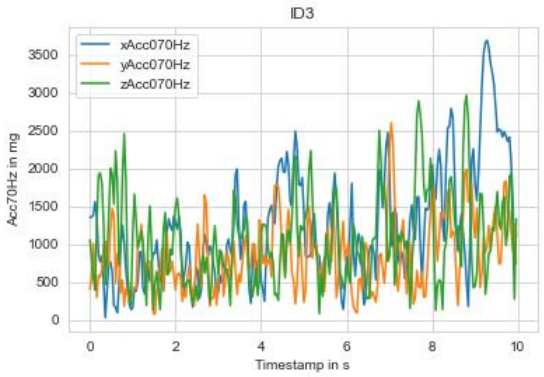


Abbildung 3.13 ID3 Acc70Hz

Des Weiteren sind Überschneidungen der Graphen bei fast allen Zuständen zu erkennen, wobei in y-Richtung im Vergleich zu z- und x- signifikant niedrigere Werte erreicht werden. Deutlich wird dies auch durch die Mediane in der Boxplot-Darstellung. Allerdings zeigt sich auch, dass im Vergleich zum Normalzustand ID2 auch in y-Richtung eine leichte Erhöhung, bis auf Zustand ID1, zu erkennen ist.

Frequenzbereich 75 Hz

Die Verläufe der Beschleunigungen für den Frequenzbereich 75 Hz sind in der Abbildung 7.26 in Anhang A dargestellt. Anzeichen auf fehlende oder unschlüssige Werte sind nicht zu erkennen. Weitere Besonderheiten sind in Abbildung 3.14 bei ID4 zu erkennen. Die Verläufe der Graphen für die Beschleunigung in x- und z-Richtung weisen über die Zeitspanen von 10 s ansteigende Werte auf.

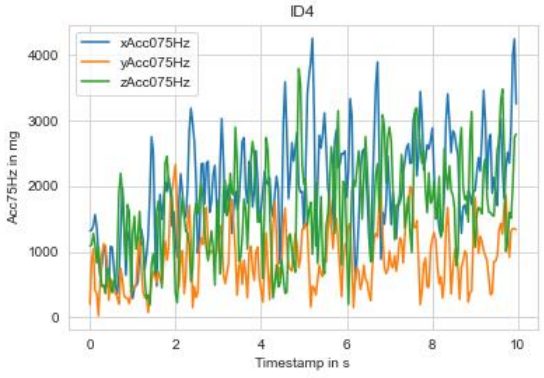


Abbildung 3.14 ID4 Acc75Hz

Abbildung 7.27 in Anhang A stellt die einzelnen Werte der Betriebszustände für den Frequenzbereich 75 Hz als Boxplots dar. Aus den vorliegenden Diagrammen sind statistische Werte der einzelnen Beschleunigungen ablesbar. Das Maximum in x-Richtung liegt bei über 5.000 mg in Zustand ID7, in y-Richtung mit ca. 2.500 mg bei ID5 und in z- Richtung mit über

4.000 mg ebenfalls bei Zustand ID5. Die Mediane der simulierten Fehlerzustände sind im Vergleich zum Normalzustand ID2 in x- und z-Richtung deutlich und in y-Richtung leicht erhöht.

Frequenzbereich 80 Hz

In Abbildung 7.28 in Anhang A sind die Verläufe der Beschleunigungen für den Frequenzbereich 80 Hz aufgeführt. Fehlende Werte sind nicht erkennbar. Ebenso sind die einzelnen Verläufe der Graphen überlappend. Das Maximum dieses Frequenzbereiches liegt in z-Richtung bei 3.500 mg. Des Weiteren fällt auch durch die Boxplots in Abbildung 7.29 in Anhang A auf, dass bei allen Zuständen das Maximum in z-Richtung zu finden ist. Die 50 %-Quantile liegen bei allen Zuständen in jeder Richtung bei einem ähnlichen Niveau. Eine Ausnahme bildet der Zustand ID1 mit dem Maximum von 30 Hz in z-Richtung.

Frequenzbereich 85 Hz

Die Werte des Frequenzbereichs 85 Hz sind in der Abbildung 7.30 in Anhang A durch Liniendiagramme veranschaulicht. Es sind keine fehlende Werte zu erkennen. Darüber hinaus sind keine Auffälligkeiten zu erkennen.

In Abbildung 7.31 in Anhang A wird der Frequenzbereich 85 Hz durch Boxplots veranschaulicht. Aus ihr geht hervor, dass die Maximalwerte in x-Richtung bei über 2.000 mg, in y-Richtung bei über 2.500 mg und in z-Richtung bei knapp unter 3.500 mg liegen. Des Weiteren liegen die Mediane der Fehlerzustände in x- und y-Richtung auf einem ähnlichen Niveau oder niedriger als der normale Betriebszustand ID2. In z-Richtung sind die Mediane im Vergleich zu ID2 leicht erhöht.

Frequenzbereich 90 Hz

Abbildung 7.32 in Anhang A stellt die Verläufe der drei Beschleunigungsrichtungen aller acht Betriebszustände für den Frequenzbereich 90 Hz als Liniendiagramme dar. Für fehlende Werte sind keine Anzeichen zu erkennen. Die Verläufe der Fehlerzustände unterscheiden sich zum Normalzustand ID2 deutlich. In Abbildung 7.33 in Anhang A sind zudem in x- sowie y-Richtung deutliche und in z-Richtung leichte Verschiebungen der Mediane zu erkennen. Das Maximum in x-Richtung ist bei Zustand ID3 über den gesamten Frequenzbereich bei nahe 5.000 mg. In y-Richtung liegt das Maximum in Zustand ID2 bei über 3.500 mg und in z-Richtung bei ID6 mit über 3.000 mg.

Frequenzbereich 95 Hz

Die Verläufe der Beschleunigungen in x- , y- und z-Richtung für den Frequenzbereich 95 Hz sind in der Abbildung 7.34 in Anhang A aufgeführt. Auffälligkeiten hinsichtlich fehlender Werte sind in keine der acht Diagramme zu erkennen. Die Verläufe an sich erscheinen optisch ebenfalls recht ähnlich. Jedoch zeigen sich, zu sehen in Abbildung 7.35 in Anhang A, deutliche Unterschiede im Wertebereich der einzelnen Beschleunigungsrichtungen.

Das Maximum für die x-Richtung liegt bei ID3 um 10.000 mg, das Maximum in y-Richtung ebenfalls bei Zustand ID3 ist größer als 5.000 mg und in z-Richtung in Zustand ID3 bei über

4.000 mg. Des Weiteren zeigen die Boxplots der y-Richtung in Abbildung 7.34 einen kleineren Interquantilsabstand als in x- oder z-Richtung. Im Vergleich zum Normalzustand ID2 liegen die Mediane in x-Richtung, mit Ausnahme von ID3, niedriger. In y-Richtung sind sie, ausgenommen ID1 und ID3, annähernd im gleichen Bereich. Die Mediane in z-Richtung liegen mit Ausnahme von Zustand ID1, in einem ähnlichen Bereich um 1.500 mg, wobei der Median von ID3 bei über 2.000 mg liegt.

Frequenzbereich 100 Hz

Abbildung 7.36 in Anhang A stellt die Verläufe der Beschleunigungen abhängig von ihrer Richtung für den Frequenzbereich 100 Hz dar. Fehlende Werte sind über alle Zustände hinweg nicht zu erkennen. Zu erkennen ist, dass sich die Fehlerzustände vom Normalzustand ID2 unterscheiden. Des Weiteren fällt auf, dass die Werte der Beschleunigungsrichtung x im Vergleich zu y und z über alle Zustände, mit Ausnahme von ID1 und ID3, deutlich erhöht sind. Darüber hinaus sind die Wertebereiche je nach Richtung, sofern ID1 außen vorgelassen wird, aller Zustände ähnlich. Verdeutlicht wird dies durch die Boxplotdarstellung in Abbildung 7.37 in Anhang A.

Die Mediane in x-Richtung liegen bis auf die Zustände ID1 und ID3 in einem Bereich von 4.000 mg bis 5.000 mg. In y-Richtung liegen die Mediane in einem Bereich um 1.000 mg und in z-Richtung zwischen 1.500 mg und 2.500 mg. Die Maxima der jeweiligen Beschleunigungen liegen für die x-Richtung bei ID3 mit über 7.000 mg, für die y-Richtung bei ebenfalls bei ID3 mit über 4.000 mg und für die z-Richtung bei ID3 und ID7 mit annähernd 6.000 mg.

Frequenzbereich 105 Hz

Die Verläufe der Beschleunigungen in Abbildung 7.38 in Anhang A für den Frequenzbereich 105 Hz weisen keine Auffälligkeiten hinsichtlich fehlender Werte auf. Für alle Betriebszustände, eine Ausnahme bildet ID1, befinden sie sich in einem ähnlichen Wertebereich. Zur Verdeutlichung ist auch Abbildung 7.39 in Anhang A heranzuziehen. Aus der Abbildung lassen sich auch die Mediane sowie die Maxima ablesen. In x-Richtung sowie in y liegen alle Mediane, eine Ausnahme bildet wieder ID1, um 1.000 mg. In z- Richtung unter Berücksichtigung derselben Ausnahme, leicht über 1.000 mg. Das absolute Maximum des Frequenzbereiches liegt in z-Richtung bei ID3 und ID mit über 4.000 mg. Die Maxima in x- und y-Richtung liegen jeweils bei über 3.500 mg.

Frequenzbereich 110 Hz

Abbildung 7.40 in Anhang A stellt die Verläufe der Beschleunigungen in x-, y-, und z-Richtung für den Frequenzbereich 110 Hz dar. Darüber hinaus zeigt Abbildung 7.41 in Anhang A die Beschleunigungswerte sortiert nach Richtung als Boxplots. Die Verläufe weisen keine Auffälligkeiten bezogen auf fehlende Werte auf. Trends sind aus der Grafik heraus ebenfalls nicht festzustellen. Die Werte reichen bei allen Beschleunigungsrichtungen bis 3.500 mg, in y-Richtung wird dieser Wert in Zustand ID3 und ID5 leicht überschritten. Des Weiteren liegen die Mediane, immer mit Ausnahme von ID1, in allen Richtungen um 1.000 mg.

Frequenzbereich 115 Hz

Die Verläufe der Beschleunigungen für den Frequenzbereich 115 Hz sind nach Zustand in der Abbildung 7.42 in Anhang A dargestellt. Zu erkennen sind keine Auffälligkeiten hinsichtlich fehlender Werte. Prägnant sind auftretende Spitzen in y- und z-Richtung. Der Wertebereich über alle Richtungen hinweg hat eine Range bis über 3.500 mg in x-Richtung bei Zustand ID3, siehe Abbildung 7.43 in Anhang A. Des Weiteren liegen die Graphen über den gesamten Frequenzbereich, mit Ausnahme von ID1, in einem mittleren Bereich um 1.000 mg, erkennbar an den Medianen. Darüber hinaus sind keine Besonderheiten ersichtlich.

Frequenzbereich 120 Hz

In Abbildung 7.44 in Anhang A sind die Beschleunigungsverläufe des Frequenzbereiches 120 Hz für alle acht Betriebszustände aufgeführt. Hinweise für fehlende Werte sind nicht zu erkennen. Der Maximalwert der einzelnen Beschleunigungsrichtungen liegen in x- Richtung bei 4.000 mg, in y-Richtung bei knapp unter 4.000 mg und in z-Richtung um 3.500 mg. Genauer geht dies auch durch die Darstellung der Werte als Boxplots in Abbildung 7.45 in Anhang A.

Bei den Verläufen ist bei ID4, siehe Abbildung 3.15, in x-Richtung ein Anstieg der Werte in x-Richtung über die Zeit zu erkennen. Des Weiteren ist bei ID3 um ca. 9 s eine Spitze bei allen drei Richtungen auffällig. Die Mediane in y- und z-Richtung liegen jeweils in einem ähnlichen Bereich, wobei sich eine leichte Erhöhung im Vergleich zum Normalzustand ID2 erkennen lässt. In x-Richtungen sind deutliche Unterschiede im Vergleich zum Normalzustand ersichtlich.

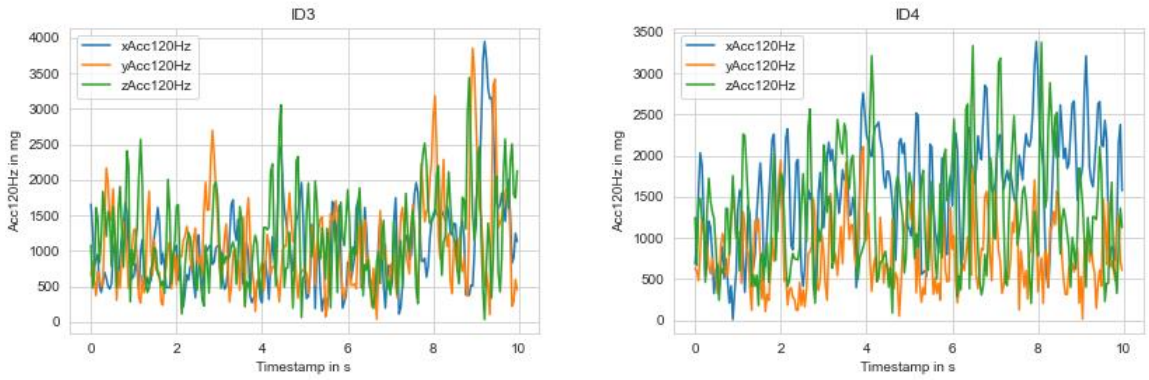


Abbildung 3.15 ID3 und ID4 Acc120Hz

3.4 Zwischenergebnis aus den Frequenzbereichen

Für eine bessere Zuordnung und Verständnis der in dem vorherigen Kapitel dargestellten Diagrammen sollen in diesem Kapitel kurz die wichtigsten übergeordneten Merkmale jedes Betriebszustands zusammengefasst werden. Die Ergebnisse sind in Tabelle 3.1 beschrieben. Darüber hinaus wurden alle Frequenzbereiche nach ID neugeordnet und als Boxplots in

Abbildung 7.46 bis Abbildung 7.53 in Anhang A dargestellt. Des Weiteren wurden keine fehlenden Werte im Datensatz festgestellt. Darüber hinaus sind die Wertebereiche mit Ausnahme von ID1 der einzelnen Betriebszustände in einem ähnlichen Bereich zwischen null und 14.000 mg. Auch die Verläufe lassen mit Ausnahme von ID1 keine offensichtlichen Differenzierungen zwischen den einzelnen Zuständen zu. Unterschiede lassen sich lediglich in den einzelnen Frequenzbereichen feststellen.

Tabelle 3.1 Merkmale der Betriebszustände

Zustand	Wertebereich [mg]	Merkmale
ID1	0 - 175	- Boxplot: Abbildung 7.46 - Besonders hoch bei zAcc10Hz im Vergleich zu den anderen Frequenzbereichen - In z-Richtung meist höhere Werte als x- oder y-Richtung - Kleine Interquantilsabstände - Meistens Werte zwischen 0 und 25 Hz - Lokales Maximum bei ca. 3 s
ID2	0 – 14.000	- Boxplot: Abbildung 7.47 - Besonders hoch bei xAcc45Hz und xAcc50Hz im Vergleich zu den anderen Frequenzbereichen - Variierende Interquantilsabstände; grundsätzlich, aber kleiner als bei den anderen Zuständen - Meistens Werte zwischen 0 und 4.000 mg - Überwiegend stationäre Verläufe - Acc45Hz, Acc50Hz, Acc55Hz, Acc95Hz und Acc100Hz: deutliche Erhöhung der Beschleunigung in x- im Vergleich zu y- und z-Richtung
ID3	0 - >14.000	- Boxplot: Abbildung 7.48 - Besonders hoch bei xAcc45Hz und xAcc50Hz im Vergleich zu den anderen Frequenzbereichen - Variierende Interquantilsabstände und Spannweiten. Diese sind größer als bei ID2 - Überwiegend stationäre Verläufe - Acc45Hz, Acc50Hz und Acc95Hz: deutliche Erhöhung der Beschleunigung in x- im Vergleich zu y- und z-Richtung

Zustand	Wertebereich [mg]	Merkmale
		- Signifikantes Maximum bei ca. 9 s bei xAcc70Hz, xAcc75Hz, xAcc115Hz und xAcc120Hz
ID4	0 - >14.000	- Boxplot: Abbildung 7.49 - Hohe Spannweiten und erhöhte Interquantilsabstände - Besonders hoch bei xAcc50Hz im Vergleich zu den anderen Frequenzbereichen - Deutliche Erhöhungen im mittleren Frequenzbereich im Vergleich zu ID2 - Ansteigende Werte sind insbesondere in x-Richtung bei Acc20Hz, Acc25Hz, Acc75Hz und Acc120Hz zu erkennen. Darüber hinaus scheinen die Verläufe stationär.
ID5	0 - >14.000	- Boxplot: Abbildung 7.50 - Auffällige Erhöhung bei xAcc20Hz und xAcc25Hz - Deutliche Erhöhung der Werte, der Spannweiten und Interquantilsabstände der unteren Frequenzbereiche gegenüber ID2 - Deutliche Erhöhung der Mediane besonders bei den niedrigeren und mittleren Frequenzen ca. bis 95 Hz im Vergleich zu ID2 - Überwiegend stationäre, mitunter zufällige wirkende Verläufe - Steigender Verlauf in y-Richtung bei Acc35Hz
ID6	0 - >14.000	- Boxplot: Abbildung 7.51 - Deutliche Erhöhung der Werte, der Spannweiten und Interquantilsabstände der unteren Frequenzbereiche gegenüber ID2 - Auffällige Erhöhung bei xAcc20Hz und xAcc25Hz - Stationäre Verläufe - Ähnliche Verläufe und Werte wir ID7 z B. bei Acc80Hz in Abbildung 7.28
ID7	0 - <14.000	- Boxplot: Abbildung 7.52 - Deutliche Erhöhung der Werte, der Spannweiten und Interquantilsabstände der unteren Frequenzbereiche gegenüber ID2

Zustand	Wertebereich [mg]	Merkmale
		- Auffällige Erhöhung bei xAcc20Hz und xAcc25Hz - Stationäre Verläufe - Ähnliche Verläufe und Werte wir ID6 z B. bei Acc80Hz in Abbildung 7.28
ID8	0 - >14.000	- Boxplot: Abbildung 7.53 - zAcc45Hz ist auffällig erhöht - grundsätzliche Erhöhung im niedrigen Frequenzbereich bis ca. 50 Hz und Abnahme bei höheren Frequenzbereichen im Vergleich zu ID2. - Stationäre Verläufe

3.5 Auswahl der Methode

In Kapitel 2.7 wurden verschiedene ML- und DL-Verfahren vorgestellt. Offen ist nun, welches Verfahren sich am besten für den vorliegenden Datensatz eignet. In diesem Kapitel soll eine Auswahl von Verfahren anhand von verschiedenen Quellen als auch der vorliegenden Datenstruktur getroffen werden. Verschiedene Reviews haben sich bereits mit der Auswahl von ML-Methoden beschäftigt und Vor- und Nachteile präsentiert.

Aufgrund des vorliegenden Datensatzes ergeben sich weitere Anforderungen. Es liegen Daten von acht Zuständen vor, daher muss die zu verwendende Methode eine Klassifikation von acht Klassen ermöglichen. Weiterhin sollte die Methode auftretende zeitliche Zusammenhänge berücksichtigen und als Inputs die verschiedenen Frequenzbereiche und Richtungen aufweisen. Des Weiteren kommt ein überwachter Lernalgorithmus in Frage, da die Daten über die Zustände bereits vorklassifiziert sind. Zudem soll später das Modell auf andere vergleichbare Maschinen übertragbar sein. Daher soll eine hohe Generalisierbarkeit vorliegen.

In [28] werden verschiedenste ML- als auch DL-Methoden aus verschiedensten Branchen miteinander verglichen, um den derzeitigen Stand der Technik im Bereich PdM wiederzugeben. Die dort vorgestellten Methoden wurden nach Industriesektoren unter Berücksichtigung der spezifischen Inputparameter mit dem Ziel eine Karte als Entscheidungshilfe für zukünftige PdM-Anwendungen zu geben ausgewählt. Als Ergebnis des Papers wird präsentiert, dass datenbasierte Methoden eine zukunftsweisende Lösung für die moderne Produktion darstellen. Bezugnehmend auf eine Übersicht von ML- und DL-Modellen wird angeführt, dass DL-Modelle als effizienter für die für die Erkennung von multidimensionalen Daten und Fehlererkennung [28]. Abbildung 3.16 zeigt die eben erwähnte entwickelte Entscheidungshilfe. Anhand des Einsatzgebiets, welches links dargestellt ist und der Inputfeatures, also der Art des Inputs resp.

der Art der Daten, können so passende Methoden für den spezifischen Anwendungsfall ausgewählt werden.

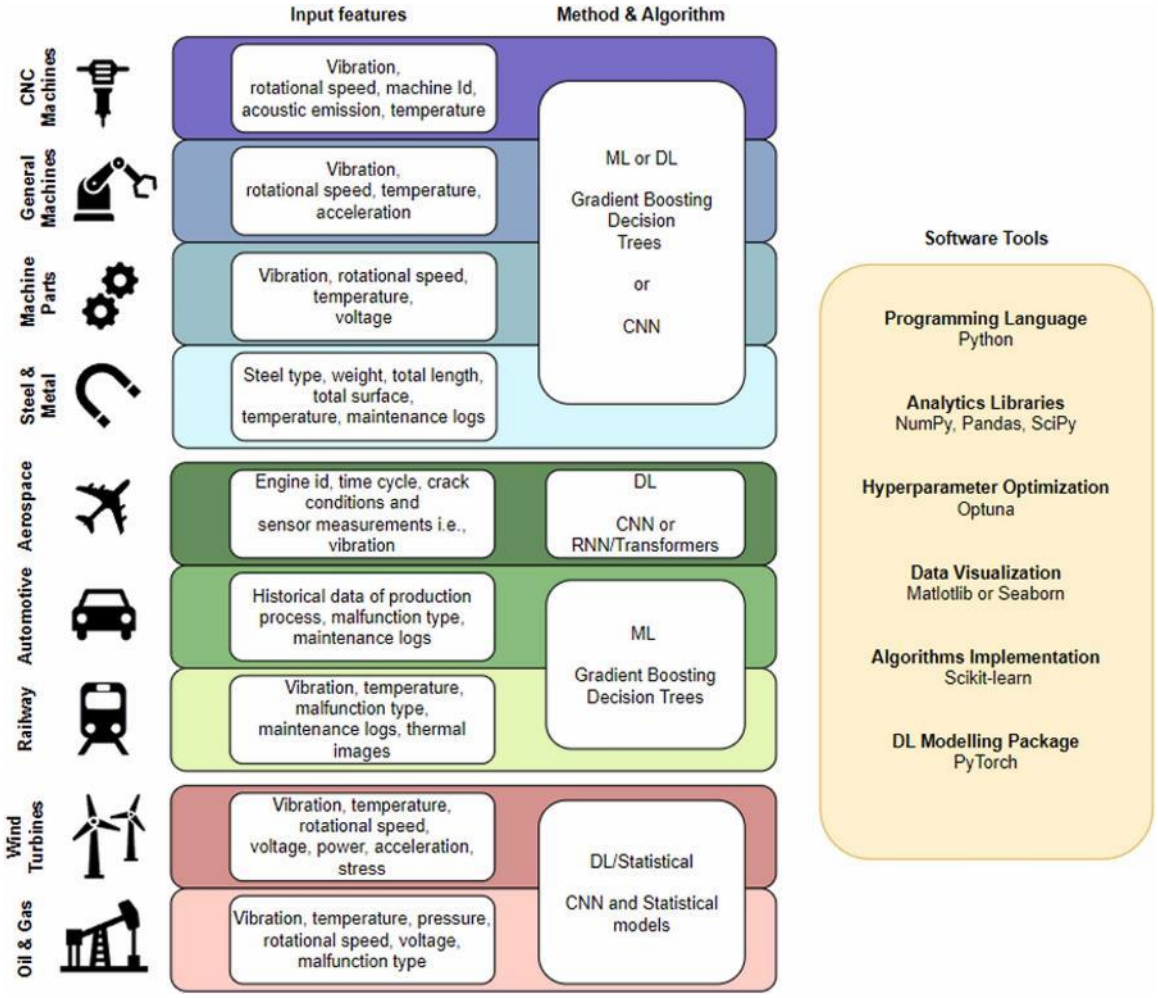


Abbildung 3.16 Karte Entscheidungshilfe PdM Anwendungen [28]

Wird nun die Karte auf die zuvor beschriebenen Daten angewendet, ergibt sich, dass die Daten am ehesten in die Kategorie „machine parts“, also Maschinenteile oder in die Kategorie „general machines“, Maschinen allgemein, einordnen lassen. Wie zuvor in Kapitel 3.1 beschrieben, liegt eine Testaufbau mit einem Motor und einen Ventilator als wesentliche Bestandteile vor. Die Messung erfolgt am Motor, was auf die Überwachung eines Teils der Gesamtmaschine „Teststand“ schließen lässt.

Darüber hinaus sind Vibrationsdaten vorhanden, sodass nach der Entscheidungshilfe in Abbildung 3.16 als Methoden und Algorithmen im Allgemeinen ML oder DL und im spezifischen die Verfahren „Gradient Boosting Decision Trees“ (Entscheidungsbaum), oder CNN in Frage kommen.

Eine qualitative Einschätzung zu den Algorithmen findet sich auch in [29] hinsichtlich Klassifikationszeit, Trainingszeit und Generalisierung. Generalisierung meint dabei eine

Übertragbarkeit des Algorithmus auf ähnliche Daten. Ein Auszug der für die zuvor vorgestellten Algorithmen ist in der Tabelle 3.2 aufgeführt. Es ist erkennbar, dass keine Methode in allen Kategorien optimale Werte aufweist. Darüber hinaus werden die Methoden allgemein bewertet und der Anwendungsfall bzw. der Datentyp nicht miteinbezogen. Dieser allerdings mit zu berücksichtigen. Zudem sind nicht alle zuvor vorgestellten Algorithmen aufgeführt. Die Einschätzung lässt allerdings eine erste Orientierung für die Auswahl einer Methode zu. Auf den ersten Blick scheine alle Methoden grundsätzlich geeignet.

Tabelle 3.2 qualitative Bewertung Algorithmen [29]

Methode	Klassifikationszeit	Trainingszeit	Generalisierung
KNN	Langsam	Sehr schnell	Gut für hohe k
SVM	Sehr schnell	Sehr langsam	Sehr gut
Random Forest	Schnell	Sehr langsam	Sehr gut
NN	Schnell	Langsam	gut

Kriterien für die Bewertung, Vor- und Nachteile einzelner ML- und DL-Verfahren werden auch in [35] in Bezug auf rotierende Maschinen herausgearbeitet. Bewertungskategorien sind zum einen die generelle Genauigkeit, die Klassifikationsgeschwindigkeit, die Robustheit gegenüber Rauschen, der Umgang mit Overfitting bzw. Überanpassung, die physikalische Erklärbarkeit und die Robustheit gegenüber Parametern. Jeder der in dieser wissenschaftlichen Publikation betrachteten Algorithmen wurde mit Sternen in den einzelnen Kategorien bewertet. Dabei bedeutet viele Sterne eine gute und wenige eine schlechte Performance. Ein Auszug der Bewertung ist in Tabelle 3.3 zu finden.

Tabelle 3.3 Performancevergleich verschiedener KI-Algorithmen nach [35]

Kategorie	SVM	KNN	DL
Allg. Genauigkeit	****	**	****
Klassifikationsgeschwindigkeit	****	*	**
Robustheit gegenüber Rauschen	**	*	****
Umgang mit Overfitting/ Überanpassung	**	***	***
Physikalische Erklärbarkeit	*	***	*
Robustheit gegenüber Parametern	*	***	**

Von den in Kapitel 2.7 beschriebenen Verfahren wird auf KNN, SVM und im Allgemeinen auf DL eingegangen. Von den drei Methoden erreichte in Summe DL mit 16 Sternen die höchste Sterneanzahl. SVM liegt bei 14 und KNN bei 13 Sternen. In der Kategorie Genauigkeit werden DL und SVM mit vier Sternen am höchsten bewertet. Im Bereich Klassifizierungsgeschwindigkeit

ist SVM mit ebenfalls vier Sternen am höchsten eingestuft. Des Weiteren liegt in der Kategorie Robustheit gegenüber Rauschen DL mit vier Sternen und in der Kategorie Umgang mit Overfitting mit drei Sternen vor SVM und KNN. In den anderen beiden Kategorien physikalische Erklärbarkeit und Robustheit gegenüber Parametern erzielt KNN mit jeweils drei Sternen den höchsten Wert der drei Algorithmen.

Neben der qualitativen Bewertung werden in verschiedenen Quellen allgemeine Vor- und Nachteile sowie spezifische in Bezug auf die Anwendung auf Zeitreihen der einzelnen Algorithmen genannt. Eine Übersicht von Vor- und Nachteilen ist in der Tabelle 3.4 zu finden. Maßgebend für die Auswahl des Algorithmus ist die Anwendbarkeit auf Zeitreihen. Daher sind die Methoden SVM, Entscheidungsbaum, Random Forest, Lineare Regression und Logistische Regression nicht geeignet. Aufgrund der Notwendigkeit der Verarbeitung von großen Datenmengen wird der KNN-Algorithmus ausgeschlossen. Zudem wurde mittels KNN eine Klassifizierung des vorliegenden Datensatzes im wissenschaftlichen Artikel „Structure-borne and Air-borne Sound Data for Condition Monitoring Applications“ angewandt und dabei eine Genauigkeit von 96,9 % bei der Verwendung der Vibrationsdaten erreicht [37]. Daher wird in dieser Arbeit der Fokus auf DL-Methoden gelegt. Ein klarer Vorteil gegenüber den anderen Methoden ist das automatisierte Lernen von Merkmalen und Fehlerszenarien. Des Weiteren spricht für DL die Berücksichtigung des Zeitbezugs und eine steigende Performance bei größer werdenden Datenmengen [35]. DL stellt des Weiteren den aktuellen Stand der Technik im Bereich PdM dar [33]. Die gängigsten verwendeten DL-Methoden sind CNN, RNN und ihre Varianten. Daher werden diese im weiteren Verlauf dieser Arbeit verwendet.

Tabelle 3.4 Vor- und Nachteile Algorithmen [13], [29], [35], [43]

Methode	Vorteil	Nachteil
KNN	- Leicht umsetzbar - Gute Effizienz	- Hohe Berechnungszeit - Benötigt viel Speicherplatz - Auswahl von k beeinflusst das Ergebnis sehr - Neue Durchführung bei neuen Daten notwendig - Hohes Risiko für Überanpassung
SVM	- Hohe Klassifikations-genauigkeit	- Geringe Effizienz für große Datenmengen - Keine physikalische Bedeutung

Methode	Vorteil	Nachteil
	- Gute Annäherung von komplexen nicht- linearen Funktionen	- zusätzliche Merkmalsextraktoren notwendig - Kein Zusammenhang von zeitlichen Beziehungen abbildbar
Entscheidungsbaum	- gute Trainingseffizienz	- Kein Zusammenhang von zeitlichen Beziehungen abbildbar - Nicht robust gegenüber Rauschen
Random Forest	- Geeignet für diskrete Daten	- Kein Zusammenhang von zeitlichen Beziehungen abbildbar - Schwierige Interpretation des Black Box Modell
Lineare Regression	- Hohe Vorhersagegenauigkeit	- Kein Zusammenhang von zeitlichen Beziehungen abbildbar - Nicht anpassbar für nicht-lineare Anwendungen
Logistische Regression	- Hohe Vorhersagegenauigkeit	- Kein Zusammenhang von zeitlichen Beziehungen abbildbar
DL (CNN, RNN)	- Automatisiertes Lernen von Merkmalen und Fehler-erkennung - Kein extra Merkmalsextraktor notwendig - Zeitbezug abbildbar - Robust gegenüber Rauschen	- Benötigt große Datenmengen - Keine physikalische Bedeutung - Lange Trainingszeit - Schwierige Interpretation des Black Box Modell

3.6 Anwendung von CNN und RNN

In dem vorherigen Kapitel wurden verschiedenen vorgestellte Methoden hinsichtlich der Anwendbarkeit auf den vorliegenden Datensatz unter Berücksichtigung von verschiedenen Quellen verglichen. Als vielversprechendste Methoden wurde DL-Algorithmen ausgewählt. Durch eine weite Verbreitung sowie aktuelle Trends im Bereich DL wird der Fokus auf RNN und CNN für den weiteren Verlauf dieser Arbeit gelegt.
Für eine bessere Vergleichbarkeit werden für jedes Modell als Kennzahlen die Genauigkeit, der „Mean Squared Error“ (MSE), die Präzision, die Sensitivität und der F1-Score berechnet. Der jeweilige Aufbau der Netze wird in den einzelnen Kapiteln beschrieben.
Die Genauigkeit (engl. accuracy) beschreibt, ob ein Modell richtig klassifiziert. Für die Berechnung der Genauigkeit wird dabei die Summe aus den richtig positiv (n_{RP}) als auch die richtig negativ (n_{RN}) klassifizierten Datenpunkten im Vergleich zu allen Datenpunkten (n_{ges}) gesetzt. Die Formel für die Berechnung ist nachstehend aufgeführt [29]:

$$Genauigkeit = \frac{n_{RP}+n_{NP}}{n_{ges}} \text{[29]} \quad \textbf{6}$$

Der MSE gibt den mittleren quadratischen Fehler an, siehe Formel 7. Es ist der durchschnittliche Vorhersagefehler über die gesamte Stichprobe. Es gilt, je kleiner der Wert, desto genauer die Vorhersage. y_i bezeichnet dabei einen Datenpunkt im Datensatz und $\hat{y}_i$ die Vorhersage, welche das Modell getroffen hat [9]:

$$MSE = \frac{1}{n_{ges}} \sum_{i=1}^{n_{ges}} (y_i - \hat{y}_i)^2 \text{[9]} \quad \textbf{7}$$

Die Sensitivität (engl. recall oder sensitivity) in Formel 8 gibt den Anteil der richtig positiven Werte im Vergleich zu allen als positiv klassifizierten Werten an. Die Gesamtheit der positiven Werte umfasst sowohl die richtig positiven Werte als auch die falsch negativen Werte (n_{FP}) [29].

$$Recall = \frac{n_{RP}}{n_{RP}+n_{FN}} \text{[29]} \quad \textbf{8}$$

Die Präzision (engl. precision) stellt den Anteil der richtig positiv klassifizierten Werte im Vergleich zu allen als positiv klassifizierten Werte dar, siehe Formel 9. Die Gesamtheit aller positiven Werte setzt sich aus den richtig positiven Werten als auch den falsch positiven Werten (n_{FP}) zusammen.

$$Precision = \frac{n_{RP}}{n_{RP}+n_{FP}} \text{[29]} \quad \textbf{9}$$

Der F1-Score ist ein Mittelmaß aus der Sensitivität und der Präzision. Dieser wird als sogenannter harmonischer Mittelwert nach Formel 10 berechnet [9], [12], [29].

$$F1 = \frac{2*Precision*Recall}{Precision+Recall} \text{ [12]} \qquad \textbf{10}$$

Für die angewendeten Modelle in den nachfolgenden Kapiteln werden die Werte über alle Klassen angegeben.
Bezugnehmend auf die Instandhaltung ist es im Allgemeinen schlimmer, wenn ein Fehler als solcher nicht erkannt wird bzw. ein Fehlerzustand als Normalzustand klassifiziert wird. So würde eine Maschine weiter betrieben werden, Ausschuss produzieren oder im schlimmsten Fall Schaden nehmen, was lange und schwierig abschätzbare Ausfallzeiten zur Folge hätte.
Zur Anwendung für die Erstellung der nachfolgenden Modelle kamen die Python-Bibliotheken pandas, numpy, tensorflow, sklearn sowie keras und für die Visualisierung seaborn und matplotlib. Als Verteilung für zwischen Trainings- und Testdatensatz wird die übliche Aufteilung 80 zu 20 gewählt.

3.6.1 Anwendung von CNN

In den nachfolgenden Kapiteln wird der in Kapitel 2.7.6 vorgestellte CNN-Algorithmus in unterschiedlichen Konfigurationen angewandt. Variiert werden zum Beispiel die Anzahl der Convolutional und Pooling Layer sowie deren Filter- und Kernelgröße.

3.6.1.1 CNN_1

Die erste Anwendung eines CNN erfolgte unter Nutzung der keras-Funktion „sequential“.Der Aufbau des Algorithmus gliedert sich durch anfänglich drei Convolutional Layer an denen jeweils ein Pooling Layer der Größe zwei anschließt. Für die Pooling-Layer wird MaxPooling verwendet. Eine andere Variante wäre AveragePooling, allerdings zeigt sich MaxPooling robuster gegenüber Störsignalen [9]. Die Fenstergröße des Pooling-Layers wird auf zwei gesetzt. Die Hyperparameter der einzelnen Layer sind in Tabelle 3.5 aufgeführt. Der Python Code ist in Anhang B abgebildet.
Alle Schichten haben die ReLU-Aktivierungsfunktion. Nach den Convolutional Layern folgt eine Flatten- Schicht, um die mehrdimensionale Ausgabe der letzten Faltung als Eingabe für den Dense Layer in einen eindimensionalen Vektor aufzubereiten. Der Dense Layer ist ein einfaches Neuronales Netz, bei dem alle Neuronen miteinander verknüpft sind. Hier erfolgt auch die eigentliche Klassifikation. Bei CNN_1 sind 128 Neuronen im Dense Layer vorhanden, siehe Tabelle 3.5. Um dem Problem der Überanpassung entgegenzuwirken, wird das Dropout- Verfahren angewendet. Dabei werden zufällig einzelnen Ausgabeverbindungen

während des Trainings aus dem Netzwerk entfernt. Der Parameter 0,5 bedeutet, dass 50 % dieser Verbindungen entfernt werden. Die Output-Schicht des Dense Layer umfasst acht Neuronen, um die acht Klassen des Datensatzes zu widerzuspiegeln. Als Aktivierungsfunktion für die Ausgabe-Schicht wird die Softmaxfunktion genutzt.

Der Vorteil der Softmax-Funktion ist, dass diese Wahrscheinlichkeiten repräsentiert und in Summe 100 % über alle Ausgabeneuronen nicht überschreitet. Die mathematische Darstellung ist in Formel 11 zu finden. Dabei stellt die Variable K die Anzahl der Klassen und z den Eingabevektor dar [9].

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{[9]} \qquad \textbf{11}$$

Tabelle 3.5 CNN_1 Layer- Spezifikationen

Layer	**Filtergröße**	**Kernelgröße**	**Aktivierungsfunktion**
Convolutional Layer 1	64	12	ReLU
Convolutional Layer 2	128	3	ReLU
Convolutional Layer 3	128	3	ReLU
	Anzahl Neuronen	**Dropout**	**Aktivierungsfunktion**
Dense Layer	128	0,5	ReLU
Dense Layer (Output)	8	/	Softmax

Als Optimierungsfunktion für die Anpassungen der einzelnen Gewichte innerhalb des Neuronalen Netzes wird der „Adam“-Algorithmus genutzt. Die Klassifikation erfolgt nach der jeweiligen im Datensatz zugewiesenen ID der Zeitreihen und ist für alle angewendeten Modelle identisch. Die Daten werden für eine spätere Evaluierung des Modells in einen Testdatensatze und einen Trainingsdatensatz aufgeteilt. Dabei entfallen auf den Testdatensatz 20 % und auf den Trainingsdatensatz 80 % der Daten.

In Kapitel 4 sind die Ergebnisse des CNN_1 dargestellt. Aus Ihnen geht ein notwendiger Optimierungsbedarf aufgrund von hohen Schwankungen und schlechten Ergebnissen bei den Kennzahlen hervor, sodass Anpassungen für eine bessere Klassifizierung notwendig sind. Die weiterentwickelten Modelle sind in den nachfolgenden Kapiteln beschrieben.

3.6.1.2 CNN_2

Die zweite Klassifizierung des Datensatzes durch ein CNN wurde mit vier Convolutional Layer, einem Average Pooling Layer und einem Dense Layer durchgeführt. Hierfür wurden die Python-Bibliotheken pandas, numpy, keras und sklearn zur Berechnung resp., Datenverarbeitung und die Bibliotheken matplotlib und seaborn für die Datenvisualisierung importiert.

Der Testdatensatz steht im Gegensatz zum vorherigen Modell für eine spätere manuelle Einsicht als externe CSV-Datei zur Verfügung.
Das Modell wird ebenfalls über die keras-Funktion „sequential“ erstellt und die einzelnen Layer hinzugefügt. Das Modell CNN_2 besteht aus vier Convolutional Layern, woran jeweils eine Batchnormalisierung-Schicht und eine MaxPooling-Schicht der Größe zwei folgen.
Die Batchnormalisierung dient zur Optimierung des Netzes hinsichtlich Stabilität und Performance. Für jeden Mini-Batch, eine Teilmenge des Trainingsdatensatzes, welche gleichzeitig durch das Neuronale Netz verarbeitet wird, wird während des Trainings der Mittelwert und die Standardabweichung berechnet. Daraufhin werden die Eingaben zwischen null und eins normalisiert, sodass sie einen Mittelwert von null und eine Standardabweichung von eins haben. Die Eingabewerte werden dann anhand von lernbaren Parametern verschoben [22].
Zur Vermeidung von Übergangspassung wird ein AveragePooling Layer anstatt des zuvor verwendeten Flatten-Layer vor der Eingabe in die Dense Layer eingefügt. Dadurch werden in die nächste Schicht nur ein Durchschnittswert pro Feature-Map in die nächste Schicht weitergegeben. Zwischen MaxPooling und der Batchnormalization wird ein Dropout zur Vermeidung von Übergangspassung durchgeführt.
Die Spezifikationen dieses Netzes sind in der Tabelle 3.6 zu finden. Für die eigentliche Klassifizierung werden drei Dense-Layer verwendet und in der letzten Schicht wie bei CNN_1 ein Output Layer mit acht Neuronen und der Softmax-Aktivierungsfunktion.

Tabelle 3.6 CNN_2 Layer-Spezifikationen

Layer	**Filtergröße**	**Kernelgröße**	**Dropout**	**Aktivierungsfunktion**
Convolutional Layer 1	128	5	0,2	ReLU
Convolutional Layer 2	256	7	0,3	ReLU
Convolutional Layer 3	128	3	0,3	ReLU
Convolutional Layer 4	64	3	0,3	ReLU
	Anzahl Neuronen			**Aktivierungsfunktion**
Dense Layer 1	128	/	0,3	ReLU
Dense Layer 2	256	/	0,2	ReLU
Dense Layer 3	128	/	0,2	ReLU
Dense Layer (Output)	8	/	/	Softmax

Darüber hinaus wurden zu Optimierung Callbacks eingefügt, was den Trainingsprozess hinsichtlich der Lernrate, der Durchlaufzeit und der Reproduzierbarkeit verbessern soll. Zum Beispiel wird der Trainingsprozess frühzeitig unterbrochen, sofern keine Steigerung hinsichtlich der Genauigkeit mehr eintritt. Das beste Modell hinsichtlich der Genauigkeit wird darüber hinaus gespeichert. Der Python Code ist in Anhang B zu finden.

3.6.1.3 CNN_3

Das dritte CNN-Modell ist im Kern aufgebaut wie CNN_2. Allerdings ist der Dropout nach den Convolutional Layern nicht mehr vorhanden, siehe Tabelle 3.7.
Durch die Verwendung der Batchnormalization ist ein zusätzlicher Dropout nicht erforderlich [22]. Hinzugefügt wurde ein Export der Testdaten für eine später manuelle Einsicht. Die Ergebnisse der Durchläufe für diese Modell sind ebenfalls in Kapitel 4 beschrieben.

Tabelle 3.7 CNN_3 Layer- Spezifikationen

Layer	**Filtergröße**	**Kernelgröße**	**Aktivierungsfunktion**
Convolutional Layer 1	128	5	ReLU
Convolutional Layer 2	256	7	ReLU
Convolutional Layer 3	128	3	ReLU
Convolutional Layer 4	64	3	ReLU
	Anzahl Neuronen	**Dropout**	
Dense Layer 1	128	0,3	ReLU
Dense Layer 2	256	0,2	ReLU
Dense Layer 3	128	0,2	ReLU
Dense Layer (Output)	8	/	Softmax

3.6.1.4 CNN_4 mit keras tuner

Das vierte CNN-Modell wurde unter Verwendung von keras tuner gestaltet. Keras Tuner ist ein vorgefertigtes Optimierungssystem für Hyperparameter. Es werden die Anzahl der Convolutional Layer, deren Kernel-Größen, deren Filter-Größen, die Anzahl der Dense-Layer, der Dropout der Dense-Layer, die Anzahl der Neuronen in jedem Dense-Layer und die Lernrate optimiert. Der letzte Output-Layer, welcher für die eigentliche Klassifikation sorgt, wird im Vorfeld auf eine Größe von acht Neuronen, bedingt durch die Anzahl der Klassen, festgelegt. Die variierten Hyperparameter mit ihren anfänglichen Intervallen sind in Tabelle 3.8 aufgeführt. Nach 50 Durchläufen wurden die in Tabelle 3.9 dargestellten Werte für das gesamte Netz als optimal bewertet und anschließend der Trainings- und Testprozess durchgeführt. Die optimale Lernrate wurde mit gerundet 0,0037 bestimmt. Nach jedem Convolutional Layer befindet sich wiederum eine Batchnormalisation und eine Max-Pooling-Schicht. Vor der Eingabe in den Dense-Layer ist eine Average-Pooling Schicht installiert.

Tabelle 3.8 CNN_4 Werte Hyperparameter

Hyperparameter	Minimum	Maximum	Schrittgröße
Anzahl Convolutional Layer	1	4	1
Filtergröße für Convolutional Layer	32	256	32
Kernelgröße für Convolutional Layer	3	11	2
Anzahl Dense Layer	1	3	1
Anzahl Neuronen	64	512	64
Dropout Dense Layer	0,1	0,5	0,1
Lernrate	$1e^{-4}$	$1e^{-2}$	/

Tabelle 3.9 CNN_4 keras Tuner Layer- Spezifikationen

Layer	Filtergröße	Kernelgröße	Aktivierungsfunktion
Convolutional Layer 1	64	11	ReLU
Convolutional Layer 2	192	5	ReLU
Convolutional Layer 3	224	7	ReLU
Convolutional Layer 4	256	7	ReLU
	Anzahl Neuronen	**Dropout**	
Dense Layer 1	320	0,1	ReLU
Dense Layer (Output)	8	/	Softmax

3.6.2 Anwendung von RNN

Nachstehend sind die für den vorgestellten Datensatz angewendeten RNN-Modelle beschrieben. Diese basieren im Kern auf dem in Kapitel 2.7.5 vorgestellten RNN-Algorithmus. Für die Klassifikation des vorgestellten Datensatzes werden die bereits angesprochenen LSTM verwendet. Der Python Code aller nachfolgend vorgestellter Modelle ist in Anhang B zu finden.
LSTM haben den Vorteil, dass sie aus sog. Memory Cells bestehen. Diese speziellen Zellen ermöglichen es einem LSTM über verschiedene sog. Gates Informationen gesteuert zu vergessen oder zu behalten. Es gibt Input-, Output- und Forget-Gates. Diese regulieren, welche Informationen in die Zelle eingegeben (Input-Gate), welche Informationen weitergegeben (Output-Gate) und welche Informationen nicht mehr beachtet (Forget-Gate) werden sollen [17].
Im Gegensatz zu den CNN muss der Datensatz in die Form [samples, time steps, features] gebracht werden. Für die Modellerstellungen wird die keras- Funktion „sequential“ verwendet.

3.6.2.1 RNN_1

Das erste Modell RNN_1 besteht aus zwei aufeinanderfolgenden LSTM mit jeweils 100 Neuronen und einem Dropout von 0,2. Darauf folgt ein einfaches Neuronales Netz (Dense Layer) mit 64

Neuronen und ReLU-Aktivierungsfunktion. Die eigentliche Klassifikation erfolgt genauso wie bei den CNN über einen Dense Layer mit acht Neuronen und Softmax-Aktivierungsfunktion. Die zusammengefassten Spezifikationen sind in der Tabelle 3.10 aufgeführt. Es wurden 50 Trainingsepochen und als Optimierungsfunktion die Adam-Funktion gewählt.

Tabelle 3.10 RNN_1 Layer- Spezifikationen

Layer	Anzahl Neuronen	Dropout	Aktivierungsfunktion
LSTM 1	100	0,2	/
LSTM 2	100	0,2	/
Dense Layer	64	/	ReLU
Dense Layer Output	8	/	Softmax

3.6.2.2 RNN_2

Im Gegensatz zu RNN_1 wurde im RNN_2 vor Eingabe der Daten in den Trainingsprozess zur Vermeidung von einer Überanpassung eine Kreuzvalidierung durchgeführt.
Bei der Kreuzvalidierung wird der Datensatz in k gleichgroße Unterdatensätze aufgeteilt und das Modell k-mal unabhängig voneinander trainiert sowie getestet. Dabei wird jedes Mal ein anderer Testsatz des Datensatzes verwendet. Die restlichen Teile des Satzes werden für das Training verwendet. Für RNN_2 wird k = 5 gewählt, was dem in der Praxis üblichen Wert entspricht. Sollte die Parameter der Leistungsmessungen im Anschluss an alle Durchläufe ähnlich sein, wird der übliche Trainingsprozess unter Berücksichtigung des gesamten Datensatzes durchgeführt [29].
Darüber hinaus wurden zwei bidirektionale LSTM gewählt. Diese Architektur ermöglicht es Informationen gleichzeitig von links nach rechts, sowie von rechts nach links zu lesen, sodass Werte der direkten Vergangenheit als auch der Zukunft in die Zelle einfließen können [17].
Die Layer-Spezifikationen des RNN-Netzes sind in Tabelle 3.11 aufgeführt. Im Vergleich zu RNN_1 wurde darüber hinaus zur Vermeidung einer Überanpassung der Dropout auf 0,3 erhöht. Zur weiteren Optimierung des Trainingsprozesses wurden wie zuvor bei den CNN-Modellen ebenfalls Callbacks zur Verbesserung der Lernrate und der Durchlaufzeit eingefügt.

Tabelle 3.11 RNN_2 Layer- Spezifikationen

Layer	Anzahl Neuronen	Dropout	Aktivierungsfunktion
Bidirektional LSTM 1	2- mal 128	0,3	/
Bidirektional LSTM 2	2- mal 64	0,3	/
Dense Layer	128	/	ReLU
Dense Layer Output	8	/	Softmax

3.6.2.3 RNN_3 mit keras tuner

Das dritte RNN-Modell wurde unter Verwendung von keras tuner gestaltet. Dabei wurden erneut Grenzen für die zu optimierenden Hyperparameter im Vorfeld festgelegt. Diese sind in Tabelle 3.12 aufgeführt.

Tabelle 3.12 RNN_3 Werte Hyperparameter

Hyperparameter	Minimum	Maximum	Schrittgröße
LSTM 1	32	256	32
Dropout 1	0,0	0,5	0,1
LSTM 2	32	256	32
Dropout 2	0,0	0,5	0,1
Dense Layer	32	256	32
Lernrate	$1e^{-4}$	$1e^{-2}$	/

Als optimale Parameter für RNN_3 wurden die in Tabelle 3.13 angegebene Werte für die jeweiligen Layer ermittelt. Demnach wird kein zweiter LSTM-Layer benötigt. Die nach der Verwendung von keras tuner angegebene Lernrate für die festgelegte Adam-Optimierungsfunktion beträgt 0,0030. Für Klassifizierung am Ende wird erneut ein Dense-Layer von acht Neuronen mit der Softmax-Aktivierungsfunktion verwendet.

Tabelle 3.13 RNN_3 keras Tuner Layer- Spezifikationen

Layer	Anzahl Neuronen	Dropout	Aktivierungsfunktion
LSTM 1	128	0,2	/
Dense Layer	128	/	ReLU
Dense Layer Output	8	/	Softmax

4 Ergebnisse

In diesem Kapitel sollen die aus der Durchführung hervorgegangenen Ergebnisse der Klassifikationen der einzelnen Modelle sowie der vorangegangenen deskriptiven Analyse in Kapitel 3.2 vorgestellt werden. Für jedes Modell wurden mehrere Durchläufe ausgeführt. Die dargestellten Konfusionsmatrizen und die Darstellung des Trainings- und Validierungsprozess sind beispielhaft. Zu beachten ist, dass die Achsen der Graphen des Trainings- und Validierungsprozesses zwischen den einzelnen Modellen unterschiedliche Einteilungen aufweisen.

4.1 Ergebnisse Vorstellung des Datensatzes

Die Vorstellung und grafische Aufbereitung des Datensatzes hatte das Ziel einen ersten Eindruck der verwendeten Daten zu vermitteln. Zusammenfassend wurde ermittelt, dass keine fehlenden Werte vorhanden waren. Ausreißer wurden als ggf. signifikant für die jeweilige Klasse angesehen. Die Verläufe zeigten sich mitunter stationär und ließen auf den ersten Blick keine sofortige Unterscheidung zwischen zwei Klassen resp. Betriebszuständen zu. Eine Ausnahme bildet in der Hinsicht ID1, da hier der Wertebereich sich deutlich von den anderen Klassen unterscheidet. Bei einigen Frequenzbereiche waren darüber hinaus auch Unterschiede im Vergleich zum Normalzustand ID2 erkennbar.

4.2 Ergebnisse CNN

Das Modell CNN 1 weist unstetige Ergebnisse hinsichtlich der Metriken auf. Dazu erreicht es nur einen akzeptablen Bereich hinsichtlich der Klassifikation der Ergebnisse und zeigt Schwankungen bei mehreren Durchläufen, sodass eine Reproduktion der Ergebnisse nicht erfolgreich war. Durch die Erhöhung der Epochen war es grundsätzlich möglich die Kennzahlen des Modells zu verbessern. Aufgrund der schlechten Reproduzierbarkeit und Fehlern wurde dieses Modell verworfen und weiter verbessert.
Durch die Anpassungen im CNN_2 konnten Schwankungen grundsätzlich reduziert werden, allerdings traten diese weiterhin auf. Des Weiteren wurde nur eine geringe Steigerung der Modellgenauigkeit erreicht. Gegenüber dem ersten Modell verschlechterte sich allerdings der MSE deutlich, wohingegen die anderen drei Kennzahlen stiegen. Um die Ergebnisse genauer zu beurteilen, wurde für das Modell CNN_2 eine Konfusionsmatrix erstellt, siehe Abbildung 4.1. In ihr ist dargestellt, ob die durch das Modell vorhergesagten Klassen auf der x-Achse den richtigen Klasse auf der y-Achse entsprechen. Eine dunklere Färbung weist eine höhere Häufigkeit auf als eine hellere. Die Legende ist bei der Konfusionsmatrix rechts abgebildet.

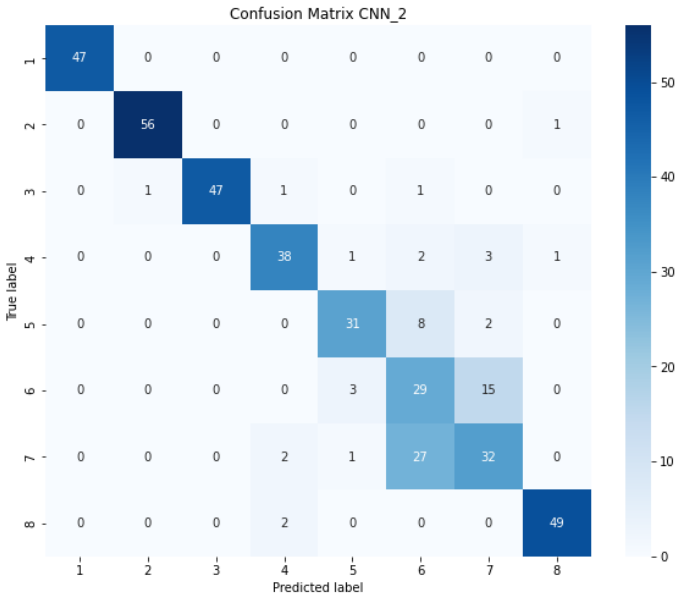


Abbildung 4.1 Konfusionsmatrix CNN_2

Es fällt auf, dass die hauptsächlichen Schwierigkeiten in der Unterscheidung der Klassen fünf, sechs und sieben liegen. Hervorzuheben sind darüber hinaus die falschen Klassifikationen als Klasse zwei, da dieser den gesunden Zustand des Teststands darstellt. Aufgetreten ist dies einmal, als die wahre Klasse ID3 war. Daher wurde auch dieses Modell weiteren Verbesserungen unterzogen.

Durch das Entfernen des Dropouts im Modell CNN_3 wurde eine höhere Genauigkeit in der Vorhersage der Klassen sowie bei den anderen Kennzahlen erzielt. Verdeutlicht wird dies auch durch die Konfusionsmatrix in Abbildung 4.2.

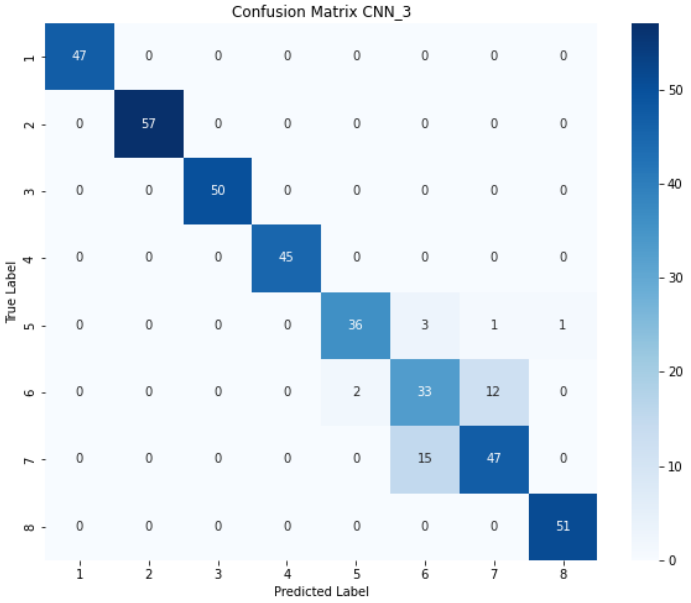


Abbildung 4.2 Konfusionsmatrix CNN_3

Aus ihr geht hervor, dass das Hauptproblem einer richtigen Klassifikation bei der Unterscheidung der Klassen fünf bis sieben liegt. Darüber hinaus werden in der Konfusionsmatrix für den abgebildeten Durchlauf alle fehlerhaften Betriebszustände auch als fehlerhaft klassifiziert.

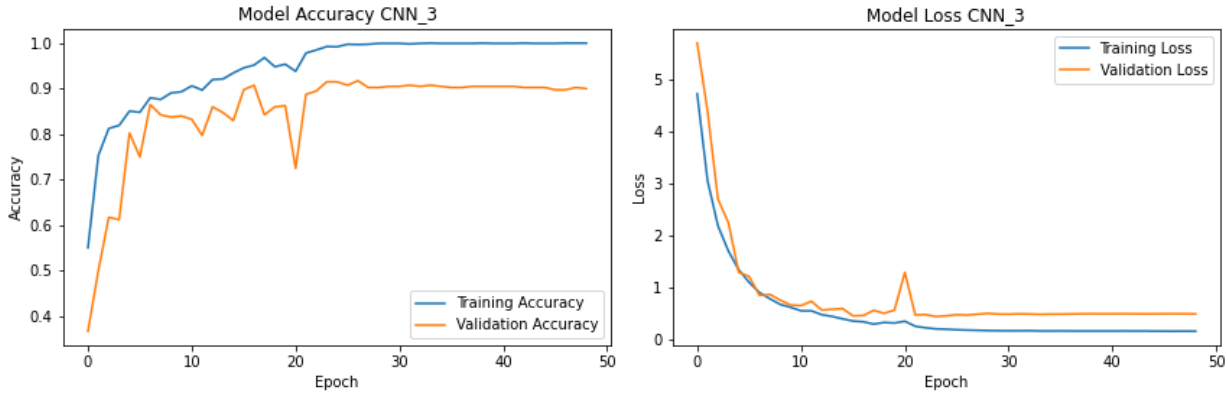


Abbildung 4.3 Modellgenauigkeit und Verlustfunktion CNN_3

Ebenfalls zeigte sich, dass die Schwankungen pro Durchlauf sich weiter reduzierten. In Abbildung 4.3 ist zudem die Modellgenauigkeit als auch die Verlustfunktion in Abhängigkeit der Epochen dargestellt. Aus Ihnen geht hervor, dass sich nach ca. 30 Epochen die Genauigkeit des Modells nicht signifikant erhöht. Des Weiteren sinkt der Fehler der Vorhersagen nach ca. 30 Epochen nicht weiter, sodass es nicht sinnvoll erscheint mehr als 30 Epochen für den Trainings- und Validierungsprozess zu verwenden.

Die Klassifikationsergebnisses des vierten CNN, welches mit keras tuner optimiert wurde, sind in Abbildung 4.4 über eine Konfusionsmatrix dargestellt. Hervorzuheben ist, dass im Kern falsche

Klassifikationen insbesondere zwischen den Klassen fünf bis sieben, auftreten. Darüber hinaus geht aus der Matrix hervor, dass kein wahrer Fehlerzustand als Klasse zwei erkannt wurde. Relevant ist dies, da der Zustand zwei den normalen Betriebszustand darstellt. Im Umkehrschluss lässt sich sagen, dass sofern ein Fehler vorliegt, dieser auch als Fehlerzustand klassifiziert wird.

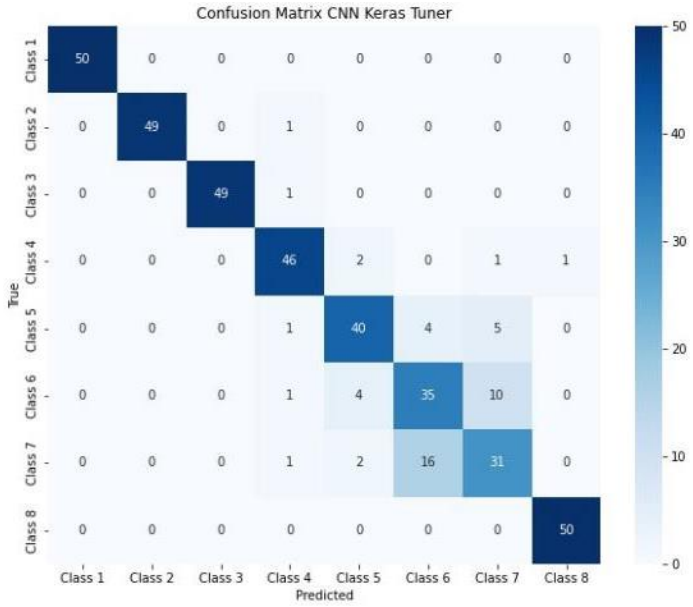


Abbildung 4.4 Konfusionsmatrix CNN_4 keras tuner

Die Metriken zur Beurteilung der Klassifikation weisen gegenüber den vorherigen Modellen mit Ausnahme von CNN_3 eine Verbesserung auf. Dem entgegen stehen die Verläufe der Modellgenauigkeit und der Verlustfunktion in Abbildung 4.5.

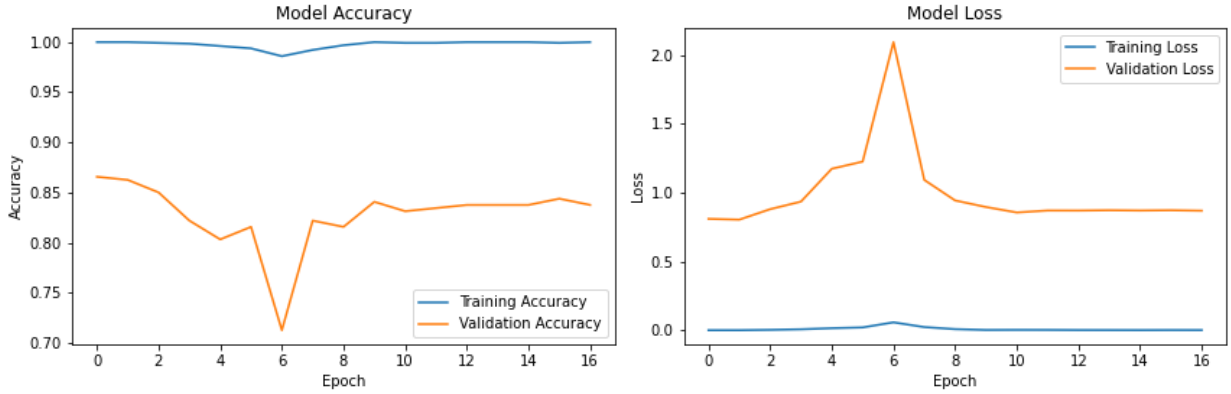


Abbildung 4.5 Modellgenauigkeit und Verlustfunktion CNN_4 keras tuner

Die Modellgenauigkeit während des Trainings verläuft in allen Epochen nahezu bei 100 % und weist so gut wie keine Fehlklassifikation auf. Im Validierungsprozess weist das Modell CNN_4

eine deutlich geringere Genauigkeit und höhere Fehlklassifikationen auf, was ein Hinweis auf eine Überanpassung des Modells ist.

4.3 Ergebnisse RNN LSTM

Die Klassifikationsergebnisse des Modells RNN_1 sind über die zugehörige Konfusionsmatrix in Abbildung 4.6 dargestellt. Aus der Matrix geht hervor, dass eine genaue Unterscheidung zwischen den Klassen fünf bis sieben schwierig erscheint. Darüber hinaus werden keine wahren Fehlerzustände als Normalzustand (Klasse 2) klassifiziert bzw. alle vorhergesagten Werte der Klasse zwei haben auch die wahre Klasse inne. Die weiteren Kennzahlen für die Bewertung des erstellten Modells sind in Tabelle 4.1 zu finden. Es fällt auf, dass der MSE vergleichsweise erhöht erscheint. Verdeutlicht wird dies auch in Abbildung 4.7 durch den ansteigenden Graphen der Verlustfunktion der Validierung bei zunehmender Epochenzahl, was ein Hinweis auf Überanpassung ist.

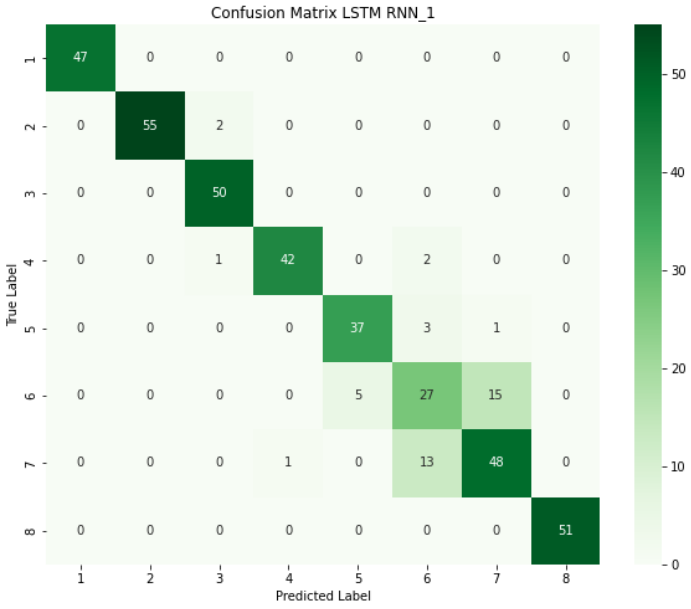


Abbildung 4.6 Konfusionsmatrix RNN_1

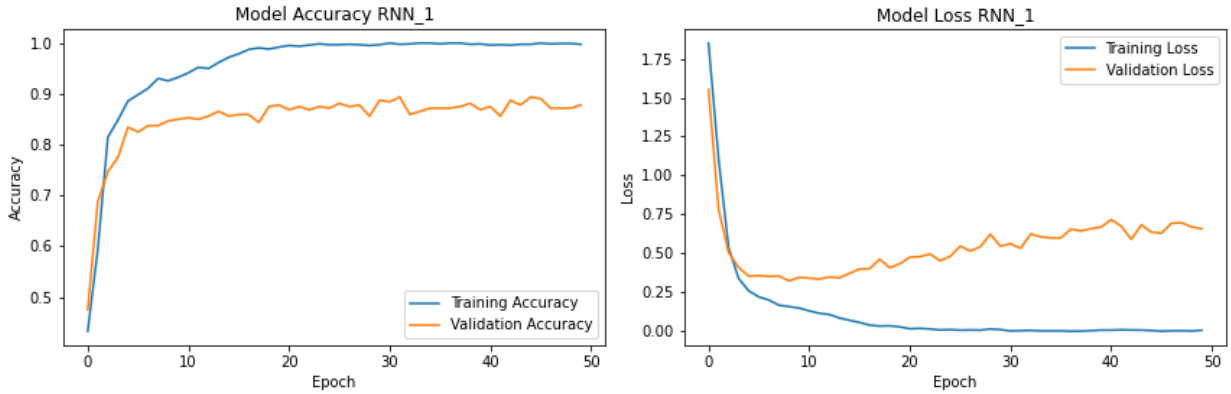


Abbildung 4.7 Modellgenauigkeit und Verlustfunktion RNN_1

Die Kennzahlen von RNN_2 in Tabelle 4.1 weisen gegenüber RNN_1 eine leichte Verschlechterung auf, liegen aber auf einem ähnlichen Niveau. Verdeutlich wird dies auch durch die Konfusionsmatrix in Abbildung 4.8. Eine richtige Klassifizierung der Klassen fünf bis sieben gelingt diesem Model nicht eindeutig. Positiv zu bewerten ist auch hier, dass das Modell, sofern ein Fehlerzustand vorliegt, es diesen auch als einen erkennt. Zudem werden alle Datenpunkte der Klasse zwei auch als solche eingeordnet. Ein Hinweis auf eine Übergangspassung des Modells liegt nicht vor. Verdeutlicht wird dies durch die Graphen des Trainings- und Validierungsprozesses in Abbildung 4.9. Die Verlustfunktionen nähern sich beide mit zunehmender Epochenanzahl null, was eine Verringerung des Klassifizierungsfehlers bedeutet. Ebenso steigt die Genauigkeit der Klassifizierung mit zunehmender Epochenzahl.

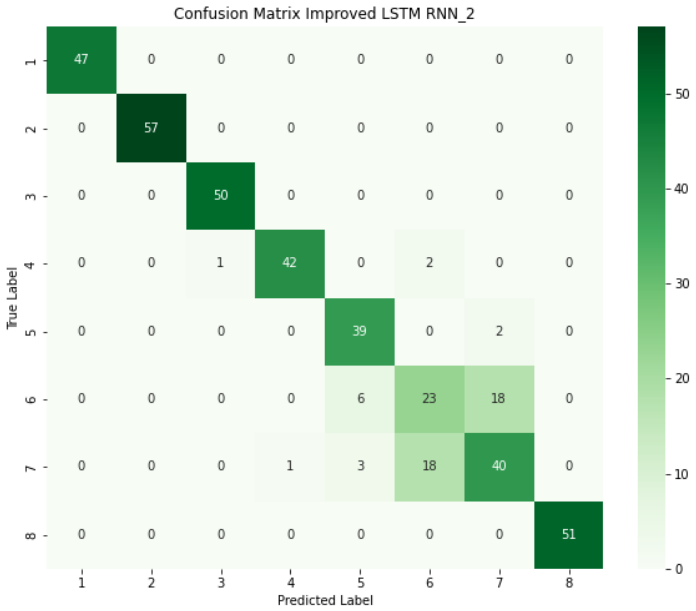


Abbildung 4.8 Konfusionsmatrix RNN_2

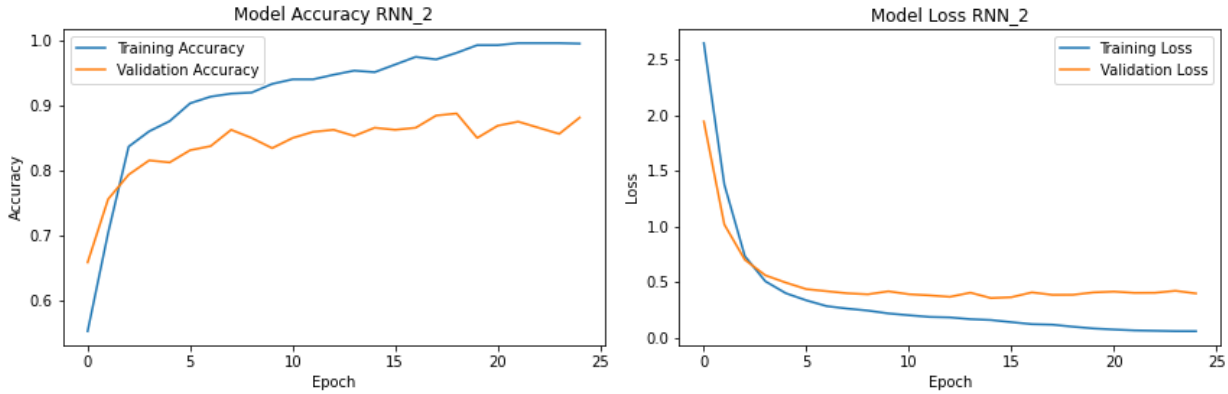


Abbildung 4.9 Modellgenauigkeit und Verlustfunktion RNN_2

Für das RNN resp. LSTM, was mit keras tuner erstellt worden ist, sind die Kennzahlen zur Beurteilung der Klassifikation ebenfalls in Tabelle 4.1 zu finden. RNN_3 weist die höchste Genauigkeit unter den RNN-Modellen auf, allerdings auch den zweitgrößten MSE. Auffällig ist, dass RNN_3 in Abbildung 4.10 eine sehr hohe Genauigkeit von über 96 % aufweist, aber die Testgenauigkeit bei 90,25 % liegt. Ebenso ist der Klassifikationsfehler bei Training und Validierung geringer als der final bestimmte MSE bei der Testung des Modells. Eine mögliche Erklärung ist, dass durch die Aufteilung des Datensatzes bei Verwendung der Kreuzvalidierung und der späteren Testung mit dem gesamten Testdatensatz diese Abweichung zu Stande kommt. Darüber hinaus wird das Training resp. die Validierung bereits nach ca. 12 Epochen abgebrochen, weil keine Verbesserung der Genauigkeit mehr eintritt.

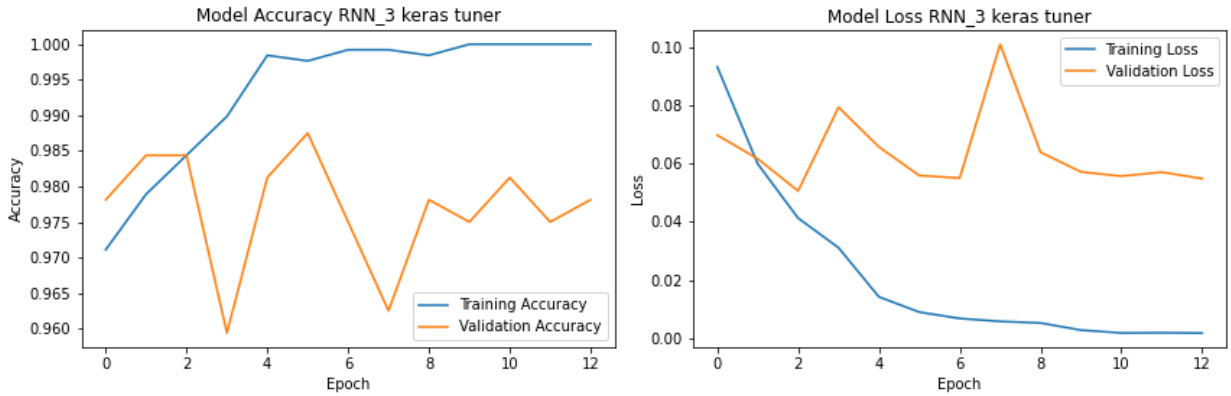


Abbildung 4.10 Modellgenauigkeit und Verlustfunktion RNN_3 keras tuner

Die Konfusionsmatrix für RNN_3 in Abbildung 4.11 stellt die Klassifikationsergebnisse des Modells dar. Über alle Klassen hinweg scheint das LSTM eine richtige Klassifikation vorzunehmen. Lediglich bei den Klassen fünf bis sieben treten in größere Mengen falsche Klassifizierungen unter diesen Klassen auf. Sofern ein Fehlerzustand der Klassen eins sowie drei bis acht vorliegt, weist das Modell auch eine Klasse aus, welche zu einem Fehlerzustand gehört.

Bezüglich der Klasse zwei, welches den Normalzustand des Teststands repräsentiert, ist zu sagen, dass bei einer Vorhersage von Klasse zwei, es auch die wahre Klasse ist.

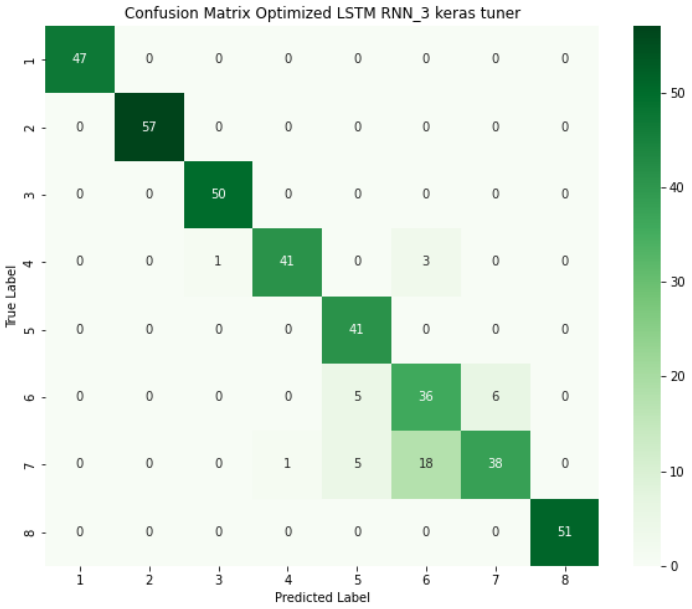


Abbildung 4.11 Konfusionsmatrix RNN_3 keras tuner

4.4 Gesamtergebnis CNN und RNN

Der Anwendung der DL-Algorithmen CNN und RNN erzielten für den untersuchten Datensatz grundsätzlich gute Klassifikationsergebnisse. Darüber hinaus wurden die Modelle über die einzelnen Entwicklungsstände hinweg weiterentwickelt und stabilisiert.

Auf Grundlage der zuvor vorgestellten Ergebnisse der einzelnen Modelle, geht die Tendenz für den Aufbau der Netze im Bereich CNN zu vier Convolutional Layer und Average Pooling vor dem Übergang in den Dense Layer. Im Bereich RNN resp. LSTM scheint ein einfaches LSTM mit einem Dense Layer ausreichend.

Tabelle 4.1 Vergleich der Modelle

Modell	Max. Epochen	Anzahl Parameter	Modell-genauig-keit	MSE	F1-Score	Präzision	Recall
CNN_1	20	157.896	~ 60,00 %	~ 0,0625	~ 0,5926	~ 0,6136	~ 0,6000
CNN_1	50	157.896	~ 70,00 %	~ 0,0455	~ 0,6765	~ 0,6716	~ 0,7125
CNN_1	100	157.896	~ 80,00 %	~ 0,0280	~ 0,7614	~ 0,8000	~ 0,7782
CNN_2	100	431.048	~ 82,25 %	~ 0,5350	~ 0,8276	~ 0,8389	~ 0,8225
CNN_3	100	431,048	**~ 91,50 %**	**~ 0,0175**	**~ 0,9157**	**~ 0,9171**	**~ 0,9150**
CNN_4_kt	100	485.544	~ 87,50 %	~ 0,0268	~ 0,8754	~ 0,8765	~ 0,8750

Modell	Max. Epochen	Anzahl Parameter	Modell-genauig-keit	MSE	F1-Score	Präzision	Recall
RNN_1	50	155.384	~ 89,25 %	~ 0,1500	~ 0,8921	~ 0,8925	~ 0,8924
RNN_2	100	384.648	~ 87,25%	~ 0,2000	~ 0,8702	~ 0,8725	~ 0,8694
RNN_3_kt	100	118.920	~ 90,25%	~ 0,1775	~ 0,9011	~ 0,9103	~ 0,9025

Aus Tabelle 4.1 geht hervor, dass im Vergleich aller Modell inkl. Variationen das Modell CNN_3 die besten Ergebnisse über alle Kennzahlen hinweg dicht gefolgt von RNN_3 liefert. Anhand der Kennzahlen fällt auf, dass die MSE-Werte der RNN-Netzwerke deutlich höher als die der CNN mit Ausnahme von CNN_2 sind.

Bei den RNN trat im Vergleich zu CNN Überanpassung auf, obwohl die Modelle deutlich weniger Parameter aufweisen als die CNN-Modelle, siehe Tabelle 4.1. Ursächlich kann der Modellaufbau selbst sein. Beispielsweise werden nicht genug Verbindungen durch Dropout gekürzt, sodass zu viele Parameter angepasst werden. Die Anzahl der Parameter wurde über die Funktion model.summary() bestimmt.

Des Weiteren zeigten alle Modelle Hauptprobleme bei der Unterscheidung zwischen den Klassen fünf bis sieben. Werden dazu die Verläufe der einzelnen Beschleunigungen herangezogen ist erkennbar, dass diese sich in nahezu jedem Frequenzbereich ähneln. Die verbleibenden Klassen wurden meist richtig erkannt. Wichtig ist außerdem zu erwähnen, dass bei Vorhersage des normalen Betriebszustands ID2, bis auf eine Klassifikation bei CNN_2, es immer auch der richtige Zustand war.

5 Diskussion

Die vorliegende Masterarbeit hatte das Ziel Methoden des Maschinellen Lernens zur automatisierten Klassifikation von Zeitreihen in Bezug auf Optimierungen der Instandhaltung im Kontext der Industrie 4.0 auszuwählen. Es wurde zu Beginn ein Einblick in die Industrie 4.0, ihre Entwicklung, die digitale Transformation sowie Technologiefelder gegeben. Verschiedene Instandhaltungsstrategien und -arten wurden vorgestellt sowie näher auf ML, KI und DL eingegangen.

Ein beispielhafter Datensatz wurde vorgestellt und auf Basis einer Literaturrecherche unter Berücksichtigung der Technologiefelder der Industrie 4.0 sowie den Eigenschaften des vorliegenden Datensatzes die Methoden CNN und RNN für die Klassifikation der Daten ausgewählt. Verschiedene Modelle der Algorithmen wurden entwickelt und anhand der Kennzahlen Genauigkeit, Präzision, MSE, Recall, F1-Score sowie einer Konfusionsmatrix beurteilt.

Die verwendeten Modelle lieferten im Maximum Genauigkeitswerte um 91 %. Im Vergleich zu einer Benchmark-Größe aus [37] schnitten diese allerdings schlechter ab. Hier erreichte ein KNN-Algorithmus eine Genauigkeit von 96,9 %. Zur Einordnung dieser Werte ist anzuführen, dass der KNN-Algorithmus mit den gesamten Vibrationsdaten getestet wurde. In dieser Arbeit wurden die Modelle mit im Vorfeld für das Modell unbekannten Daten getestet. Darüber hinaus wurden verschiedene Programmiersprachen verwendet, was unter Umständen durch die bereits abrufbaren Pakete zu Unterschieden führen kann. In [37] wurde das Programm MATLAB verwendet und in dieser Arbeit Python. Ein weiterer möglicher Grund für die geringere Kennzahlenwerte kann durch die Datenmenge bedingt sein. DL-Algorithmen benötigen für eine korrekte Klassifikation eine große Menge an Daten, sodass die vorhandenen Daten möglicherweise nicht ausreichend sind. Mitunter würde sich dann auch die generell festgestellte Problematik der richtigen Klassifikation der Klassen ID5, ID6 und ID7 verbessern.

Dazu muss allerdings erwähnt werden, dass die Klassen ID6 sowie ID7 einen ähnlichen Fehlerzustand, siehe Kapitel 3.1 und Kapitel 3.4, beschreiben, sodass in der Praxis wahrscheinlich kein Unterschied hinsichtlich der Vermeidungsmaßnahmen eintreten würde. Ein verstopftes Gehäuse würde eine Reinigung und eine kurze Unterbrechung nach sich ziehen. Eine Fehlklassifikation zwischen diesen beiden Zuständen hätte daher wahrscheinlich keine schwerwiegenden Auswirkungen. Ein Zusammenlegen der Werte innerhalb des Datensatzes könnte des Weiteren zu einer Verbesserung der Hauptfehlklassifikation von ID6 und ID7 sowie zu einer Erhöhung der Gesamtgenauigkeit führen. Weiter ist die vermehrte Fehlklassifikation von ID5 zu betrachten. Zunächst ist zu erwähnen, dass die verwendeten Modelle dennoch einen Fehlerzustand erkennen, sodass min. eine Prüfung der Maschine in der Praxis durchgeführt werden würde. Darüber hinaus ist anzunehmen, dass sich die Imbalance über die Zeit

verschlimmern würde, sodass das Gewicht zunimmt und die Sensoren größere Ausschläge registrieren würden, was eine zeitnahe Wartung bedingen kann. Daher scheinen die Modelle auf Basis der Ergebnisse für den Einsatz als Condition monitoring gut geeignet.

Eine weitere Ursache für die Fehlklassifikation kann zusätzlich ein nicht optimaler Aufbau der Modelle oder eine nicht optimale Wahl der einzelnen Parameter sein. Durch die Verwendung von keras tuner sollte dem entgegengewirkt werden, allerdings zeigt sich durch die Kennzahlen in Tabelle 4.1, dass z.B. das Modell CNN_3 bessere Werte liefert als das CNN-Modell, welches mit keras tuner erstellt wurde. Zudem zeigten sich in Abbildung 4.5 und Abbildung 4.10 Hinweise auf eine Überanpassung der mit keras tuner erstellten Modelle. Mögliche Lösungen könnten für das CNN die Anwendung der Kreuzvalidierung sein.

Überanpassungen zeigte sich eher bei RNN als bei den CNN, was auf eine bessere Eignung der CNN schließen lässt. Nicht betrachtet wurden auch Kombinationen von Modellen, welche z. B. in [24] beschrieben werden.

In [23] wird ein CNN-Algorithmus ebenfalls für die Klassifikation von Vibrationsdaten verwendet. Der angewandte Algorithmus bestand aus einem Convolutional Layer mit eine Filtergröße von 64 und einem Fully Connected Layer mit 200 Neuronen. Unterschieden wurden zwischen vier Klassen. Dabei erreichte das beschriebene CNN eine Genauigkeit von 93,61 %, eine Präzision von 94,52 %, einen Recall von 93,6 % und einen F1-Score von 94,06 %. Diese Werte sind nicht weit von den in dieser Masterarbeit ermittelten Werten des CNN_3 oder RNN_3 entfernt. Daher wird angenommen, dass die in dieser Arbeit ermittelten Werte plausibel sind.

Zusammenfassend zeigen CNN und RNN eine gute Eignung für die Klassifikation von Fehlerzuständen, wobei RNN eine leichte Tendenz zur Überanpassung aufweisen. Für alle Modelle sind dennoch weitere Steigerungen der Genauigkeit als auch eine Anpassung auf den Praxisfall je nach Anforderungen des Betreibers oder des Herstellers notwendig.

6 Ausblick

Aufbauend auf den in dieser Arbeit gewonnenen Erkenntnissen ergeben sich weitere Möglichkeiten zur Verbesserung des Maschinellen Lernens zur Klassifikation von Betriebszuständen von Maschinen und Anlagen.

Naheliegende Verbesserungen resp. Ansätze wären eine zusätzliche Einbindung der Roh- und Audiodaten oder eine andere Komprimierung als in [37] durchgeführt. Im Bereich LSTM wäre auch eine Kombination von keras tuner und bidirektionalen LSTM von Interesse. Für eine bessere Vergleichbarkeit der Modelle wäre es auch denkbar den selben Testdatensatz zu verwenden.

Als weiteres Modell wäre auch der Einsatz von Explainable AI denkbar, wodurch es möglich wird, anzuzeigen, welcher Teil der Zeitreihe für die Klassifikation signifikant ist. In eine ähnliche Richtung tendiert auch der Algorithmus der „Shapelets“. Shapelets sind Teilsequenzen einer Zeitreihe, welche repräsentativ für eine Klasse sind [44]. Vergleichbare Ergebnisse von Shapelets im Vergleich zu etablierten ML- oder DL-Algorithmen werden bspw. in [18] erreicht.

Neben der Variation der Daten können ebenso hybride Modelle aus DL und ML gute Ergebnisse erzielen. In [16] wurde ein CNN-Netzwerk einem LSTM vorgeschaltet. Ziel war es in diesem Paper einen Datensatz aus Vibrationsdaten zu analysieren. Dabei erreichte der verwendete Algorithmus eine Präzision von 1,0 und einen Recall von 1,0. Diese Möglichkeit wurden in dieser Masterarbeit nicht betrachtet und kann ein zukunftsfähiges Forschungsvorhaben bilden. Außerdem kann auf Grundlage dieser Arbeit eine weitere Betrachtung mit tatsächlichen realen anstatt im Labor erzeugten Daten erfolgen.

Auch, wenn nun gute Resultate vorliegen, „ist eine umfassende Erklärung des Verhaltens des trainierten Algorithmus noch schwieriger zu rechtfertigen, wenn es um Vorschriften für sicherheitskritische Systeme […] geht“ [24]. Sofern das Modell erst einmal trainiert ist, ist es schwierig zu erklären, wie es funktioniert. Für die Zukunft ist es daher notwendig weitere Forschung in diesem Bereich zu leisten [24].

Hinsichtlich der praktischen Anwendung eines solchen DL-Ansatzes sind auch die weiteren Ressourcen, die ein Betrieb oder Firma zur Verfügung hat zu beachten. Dabei sind nicht nur die technische Ausrüstung resp. die direkte Umwelt, die Datenaufnahme, -bereinigung,-transformation oder monetäre Stärke, sondern auch die Kompetenz und Verfügbarkeit von Mitarbeitenden, die Kultur eines Unternehmens sowie der Willen der Entscheidungsträger zur Umsetzung von neuen Ansätzen zu berücksichtigen.

7 Literaturverzeichnis

[1] ANACONDA, INC.: *Anaconda | The Operating System for AI | Anaconda Navigator*. URL https://docs.anaconda.com/navigator/?utm_source=anaconda_navigator&utm_medium=nav-docs. - abgerufen am 2024-09-24. — Anaconda

[2] ANDELFINGER, VOLKER P. ; HÄNISCH, TILL: *Industrie 4.0: Wie cyber-physische Systeme die Arbeitswelt verändern*. Wiesbaden : Springer Fachmedien Wiesbaden, Imprint: Springer Gabler, 2017. — Published: electronic; onlineDOI: 10.1007/978-3-658-15557-5 — ISBN 978-3-658-15557-5

[3] APEL, HARALD: *Instandhaltungs- und Servicemanagement: Systeme mit Industrie 4.0*. München : Carl Hanser Verlag GmbH & Co. KG, 2018 — ISBN 9783446456877

[4] BAUER, WILHELM ; SCHLUND, SEBASTIAN ; MARRENBACH, DIRK ; GANSCHAR, OLIVER: Industrie 4.0 - Volkswirtschaftliches Potenzial für Deutschland. In: BITKOM, B. I., TELEKOMMUNIKATION UND NEUE MEDIEN E. V. ; DAS FRAUNHOFER-INSTITUT FÜR ARBEITSWIRTSCHAFT UND ORGANISATION IAO (Hrsg.) (2014)

[5] BAUERNHANSL, THOMAS: *Handbuch Industrie 4.0: Band 1: Produktion*. 3rd ed. 2023. Berlin, Heidelberg : Springer Berlin Heidelberg, Imprint: Springer Vieweg, 2023. — Published: electronic; onlineDOI: 10.1007/978-3-662-58532-0 — ISBN 978-3-662-58532-0

[6] BLECHSCHMIDT, NILS ; MÄRZ, MARKUS: *Maintenance 4.0*. URL https://prozesstechnik.industrie.de/pharma/automation-pharma/maintenance-4-0/. - abgerufen am 2024-03-04. — prozesstechnik online

[7] BUNDESAMT FÜR SICHERHEIT IN DER INFORMATIONSTECHNIK (BSI): SYS.4.3: Eingebettete Systeme. In: *Eingebettete Systeme* (2021)

[8] BUNDESMINISTERIUM FÜR WIRTSCHAFT UND KLIMASCHUTZ ; BUNDESMINISTERIUM FÜR BILDUNG UND FORSCHUNG: *Was ist Industrie 4.0?* URL https://www.plattform-i40.de/IP/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html. - abgerufen am 2024-04-16

[9] CERULLI, GIOVANNI: *Fundamentals of Supervised Machine Learning: With Applications in Python, R, and Stata* : Springer International Publishing, 2023 — ISBN 978-3-031-41337-7

[10] *DIN 31051, Juni 2019. Grundlagen der Instandhaltung*. Berlin : DIN Deutsches Institut für Normung e. V., 2019

[11] *DIN EN 13306, Februar 2018. Begriffe der Instandhaltung*. Berlin : Beuth Verlag GmbH, 2018

[12] ERTEL, WOLFGANG: *Grundkurs Künstliche Intelligenz: eine praxisorientierte Einführung*, *Computational Intelligence*. 5. Auflage. Wiesbaden, Germany : Springer Vieweg, 2021 — ISBN 3-658-32074-5

[13] FARAHANI, MOJTABA A. ; MCCORMICK, M. R. ; GIANINNY, ROBERT ; HUDACHECK, FRANK ; HARIK, RAMY ; LIU, ZHICHAO ; WUEST, THORSTEN: Time-series pattern recognition in Smart Manufacturing Systems: A literature review and ontology. In: *Journal of Manufacturing Systems* Bd. 69 (2023), S. 208–241

[14] FAWAZ, HASSAN ISMAIL ; FORESTIER, GERMAIN ; WEBER, JONATHAN ; IDOUMGHAR, LHASSANE ; MULLER, PIERRE-ALAIN: Deep learning for time series classification: a review.

[15] FROST, IRASIANTY: *Einfache lineare Regression: Die Grundlage für komplexe Regressionsmodelle verstehen*, *essentials*. Wiesbaden : Springer Fachmedien Wiesbaden, Imprint: Springer VS, 2018. — Published: electronic; onlineDOI: 10.1007/978-3-658-19732-2 — ISBN 978-3-658-19732-2

[16] GONG, SEOKHYUN ; LEE, JAEHYEONG ; LEE, CHAE-GYU ; JEONG, JONGPIL ; CHOI, WONMIN ; KIM, CHANYOUNG ; PARK, JUNGSOO ; KIM, JONGHEON: CNN-LSTM-AE Based Predictive Maintenance Using STFT for Rotating Machinery. In: *2023 IEEE/ACIS 8th International Conference on Big Data, Cloud Computing, and Data Science (BCD)*, 2023, S. 80–85

[17] GRUNERT, PHILIPP: *Machine Learning und Neuronale Netze*, *Der verständliche Einstieg mit Python*. Landshut : BMU Verlag, 2021 — ISBN 978-3-96645-072-0

[18] GUILLAUME, ANTOINE AND VRAIN, CHRISTEL AND ELLOUMI, WAEL AND UNIVERSITÉ D'ORLÉANS AND FRANCE, WORLDLINE AND VRAIN, CHRISTEL: *Time series classification with Shapelets: Application to predictive maintenance on event logs ; Classification de séries temporelles avec les Shapelets : application à la maintenance prédictive via journaux d'événements*. HAL CCSD, thesis, 2023

[19] GUTSCHE, KATJA ; LEMMER, KARSTEN: *Integrierte Bewertung von Investitions- und Instandhaltungsstrategien für die Bahnsicherungstechnik ; Integrated evaluation of investment and maintenance strategies for rail operation control systems*, thesis, 2009

[20] HUBER, WALTER: *Industrie 4.0 in der Automobilproduktion: Ein Praxisbuch*. Wiesbaden : Springer Fachmedien Wiesbaden, Imprint: Springer Vieweg, 2016. — Published: electronic; onlineDOI: 10.1007/978-3-658-12732-9 — ISBN 978-3-658-12732-9

[21] HUNT, JOHN AND MACKIE, IAN, SERIES EDITOR AND ABRAMSKY, SAMSON, ADVISORY EDITOR AND HANKIN, CHRIS, ADVISORY EDITOR AND HINCHEY, MIKE, ADVISORY EDITOR AND KOZEN, DEXTER C., ADVISORY EDITOR AND PITTS, ANDREW, ADVISORY EDITOR AND RIIS NIELSON, HANNE, ADVISORY EDITOR AND SKIENA, STEVEN S., ADVISORY EDITOR AND STEWART, IAIN, ADVISORY EDITOR AND KIZZA, JOSEPH MIGGA, ADVISORY EDITOR: *A Beginners Guide to Python 3 Programming*, *Undergraduate Topics in Computer Science.*. Cham : Springer International Publishing, 2023 — ISBN 978-3-031-35122-8

[22] IOFFE, SERGEY ; SZEGEDY, CHRISTIAN: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: *CoRR* Bd. abs/1502.03167 (2015). — arXiv: 1502.03167

[23] JANSSENS, OLIVIER ; SLAVKOVIKJ, VIKTOR ; VERVISCH, BRAM ; STOCKMAN, KURT ; LOCCUFIER, MIA ; VERSTOCKT, STEVEN ; WALLE, RIK VAN DE ; HOECKE, SOFIE VAN: Convolutional Neural Network Based Fault Detection for Rotating Machinery. In: *Journal of Sound and Vibration* Bd. 377 (2016), S. 331–345

[24] JIMENEZ, JUAN JOSÉ MONTERO ; SCHWARTZ, SÉBASTIEN ; VINGERHOEDS, ROB ; GRABOT, BERNARD ; SALAÜN, MICHEL: Towards multi-model approaches to predictive maintenance: A systematic literature survey on diagnostics and prognostics. In: *Journal of Manufacturing Systems* Bd. 56 (2020), S. 539–557

[25] KAMISKE, GERD F.: *Handbuch QM-Methoden: Die richtige Methode auswählen und erfolgreich umsetzen*. 3., aktualisierte und erweiterte Auflage. München : Carl Hanser Verlag GmbH & Co. KG, 2015. — Published: electronic; onlineDOI: 10.3139/9783446444416 — ISBN 978-3-446-44441-6

[26] KRUPITZER, CHRISTIAN ; WAGENHALS, TIM ; ZÜFLE, MARWIN ; LESCH, VERONIKA ; SCHÄFER, DOMINIK ; MOZAFFARIN, AMIN ; EDINGER, JANICK ; BECKER, CHRISTIAN ; U. A.: A Survey on Predictive Maintenance for Industry 4.0.

[27] LAZOVSKIY, NIKOLAY V.: Neural Network System for Monitoring the Condition of Equipment and Predicting Malfunctions. In: *2024 Conference of Young Researchers in Electrical and Electronic Engineering (ElCon)*, 2024, S. 421–423

[28] MALLIORIS, PANAGIOTIS ; AIVAZIDOU, EIRINI ; BECHTSIS, DIMITRIOS: Predictive maintenance in Industry 4.0: A systematic multi-sector mapping. In: *CIRP Journal of Manufacturing Science and Technology* Bd. 50 (2024), S. 80–103

[29] MATZKA, STEPHAN: *Künstliche Intelligenz in den Ingenieurwissenschaften: Maschinelles Lernen verstehen und bewerten*. 1st ed. 2021. Wiesbaden : Springer Fachmedien Wiesbaden, Imprint: Springer Vieweg, 2021. — Published: electronic; onlineDOI: 10.1007/978-3-658-34641-6 — ISBN 978-3-658-34641-6

[30] MOBLEY, R. KEITH: 1 - Impact of Maintenance. In: MOBLEY, R. K. (Hrsg.): *An Introduction to Predictive Maintenance (Second Edition)*, *Plant Engineering*. Second Edition. Burlington : Butterworth-Heinemann, 2002 — ISBN 978-0-7506-7531-4, S. 1–22

[31] MOHAMMADI FOUMANI, NAVID ; MILLER, LYNN ; TAN, CHANG WEI ; WEBB, GEOFFREY I. ; FORESTIER, GERMAIN ; SALEHI, MAHSA: Deep Learning for Time Series Classification and Extrinsic Regression: A Current Survey. In: *ACM Comput. Surv.* Bd. 56. New York, NY, USA, Association for Computing Machinery (2024), Nr. 9

[32] MUELLER, JOHN PAUL ; MASSARON, LUCA ; LINKE, SIMONE: *Deep Learning kompakt für Dummies*, *... für Dummies; Lernen einfach gemacht; Erscheint auch als: Druck-Ausgabe:*

Mueller, John Paul, 1958-: Deep Learning kompakt für Dummies, ISBN 9783527716876. 1. Auflage. Weinheim : Wiley-VCH Verlag GmbH & Co. KGaA, 2020. — Published: electronic; online — ISBN 978-3-527-82597-4

[33] NAMUDURI, SRIKANTH ; NARAYANAN, BARATH NARAYANAN ; DAVULURU, VENKATA SALINI PRIYAMVADA ; BURTON, LAMAR ; BHANSALI, SHEKHAR: Review—Deep Learning Methods for Sensor Based Predictive Maintenance and Future Perspectives for Electrochemical Sensors. In: *Journal of The Electrochemical Society* Bd. 167, IOP Publishing (2020), Nr. 3, S. 037552

[34] REICHEL, JENS ; HAEFFS, JEAN ; MÜLLER, GERHARD: *Betriebliche Instandhaltung*, *VDI-Buch*. 2. Aufl. 2018. Berlin, Heidelberg : Springer Berlin Heidelberg, Imprint: Springer Vieweg, 2018. — Published: electronic; onlineDOI: 10.1007/978-3-662-53135-8 — ISBN 978-3-662-53135-8

[35] RUONAN, LIU ; XUEFENG, CHEN ; ENRICO, ZIO ; BOYUAN, YANG: Artificial intelligence for fault diagnosis of rotating machinery: A review. In: *Mechanical Systems and Signal Processing* Bd. 108. Elsevier Ltd (2018)

[36] SCHULZ, T. (Hrsg.): *Industrie 4.0 Potenziale erkennen und umsetzen*. 2. Aufl. : Vogel Communications Group, 2021 — ISBN 978-3-8343-6254-4

[37] STEPHAN MATZKA ; JOHANNES PILZ ; ANDREAS FRANKE: Structure-borne and Air-borne Sound Data for Condition Monitoring Applications. In: , 2021, S. 1–4

[38] STEPHAN MATZKA ; JOHANNES PILZ ; ANDREAS FRANKE: Condition Monitoring Dataset (AI4I 2021).

[39] SULLIVAN, GREG ; PUGH, RAY ; MELENDEZ, ALDO P ; HUNT, W D: Operations & Maintenance Best Practices - A Guide to Achieving Operational Efficiency (Release 3) (2010)

[40] *Time Series Analysis & Visualization in Python*. URL https://www.geeksforgeeks.org/time-series-data-visualization-in-python/. - abgerufen am 2024-07-02. — GeeksforGeeks

[41] WANG, JINJIANG ; MA, YULIN ; ZHANG, LAIBIN ; GAO, ROBERT X. ; WU, DAZHONG: Deep learning for smart manufacturing: Methods and applications. In: *Special Issue on Smart Manufacturing* Bd. 48 (2018), S. 144–156

[42] WASKOM, MICHAEL L.: seaborn: statistical data visualization. In: *Journal of Open Source Software* Bd. 6, The Open Journal (2021), Nr. 60, S. 3021

[43] XU, ZHAOYI ; SALEH, JOSEPH HOMER: Machine learning for reliability engineering and safety applications: Review of current status and future opportunities. In: *Reliability Engineering & System Safety* Bd. 211 (2021), S. 107530

[44] YE, LEXIANG AND KEOGH, EAMONN AND THE PENNSYLVANIA STATE UNIVERSITY CITESEERX ARCHIVES: Time series shapelets: a new primitive for data mining. In: *http://www.cs.ucr.edu/%7Elexiangy/Shapelet/kdd2009shapelet.pdf* (2009)

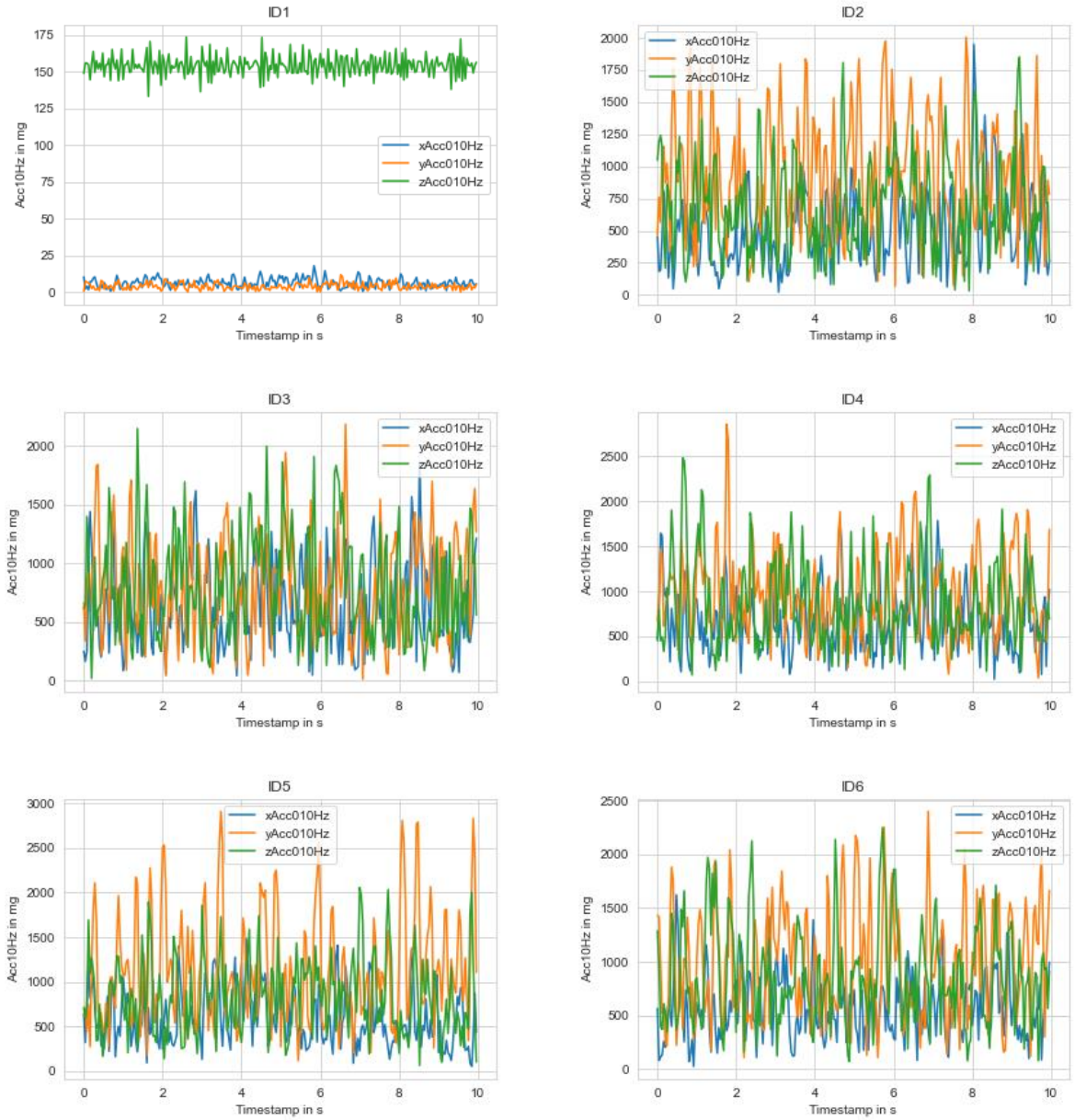
[45] ZEKI MURAT ÇINAR ; ABUBAKAR ABDUSSALAM NUHU ; ZEESHAN, QASIM ; KORHAN, ORHAN ; ASMAEL, MOHAMMED ; SAFAEI, BABAK: Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0. In: *Sustainability* Bd. 12. MDPI AG (2020), Nr. 19, S. 8211–8211

Anhang

A. Diagramme Frequenzbereiche

In den nachstehenden Kapiteln sind die Verläufe als auch die Boxplot- Diagramme der Beschleunigungen der Frequenzbereiche, auf die im Text verwiesen wurde, dargestellt.

A.1 Acc10Hz



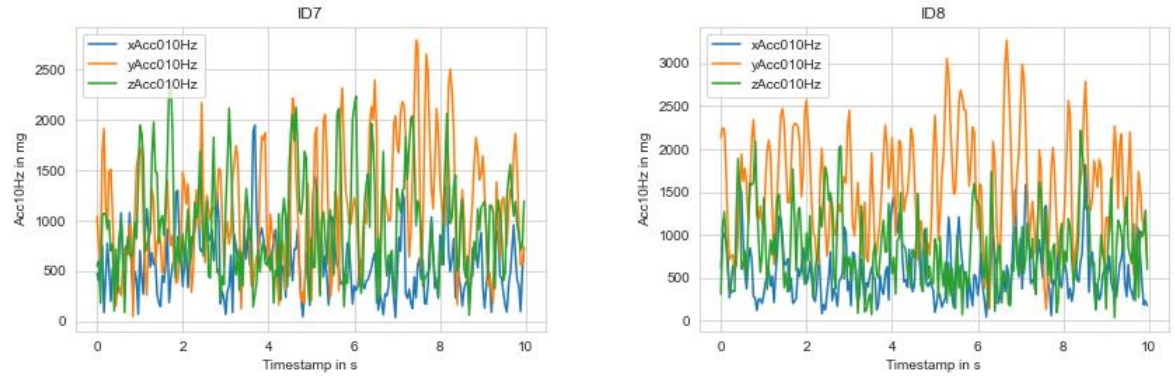


Abbildung 7.1 Liniendiagramme Acc10Hz

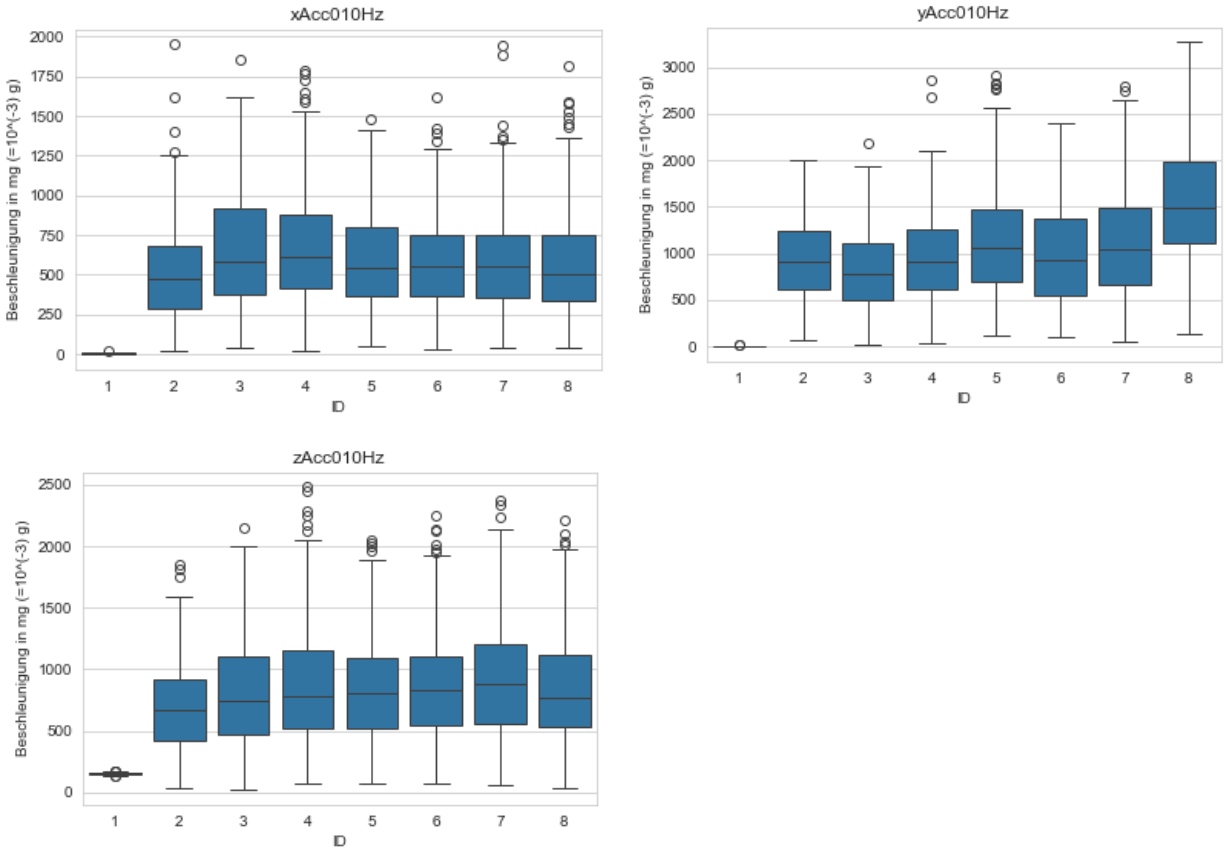


Abbildung 7.2 Boxplots Acc10Hz

A.2 Acc15Hz

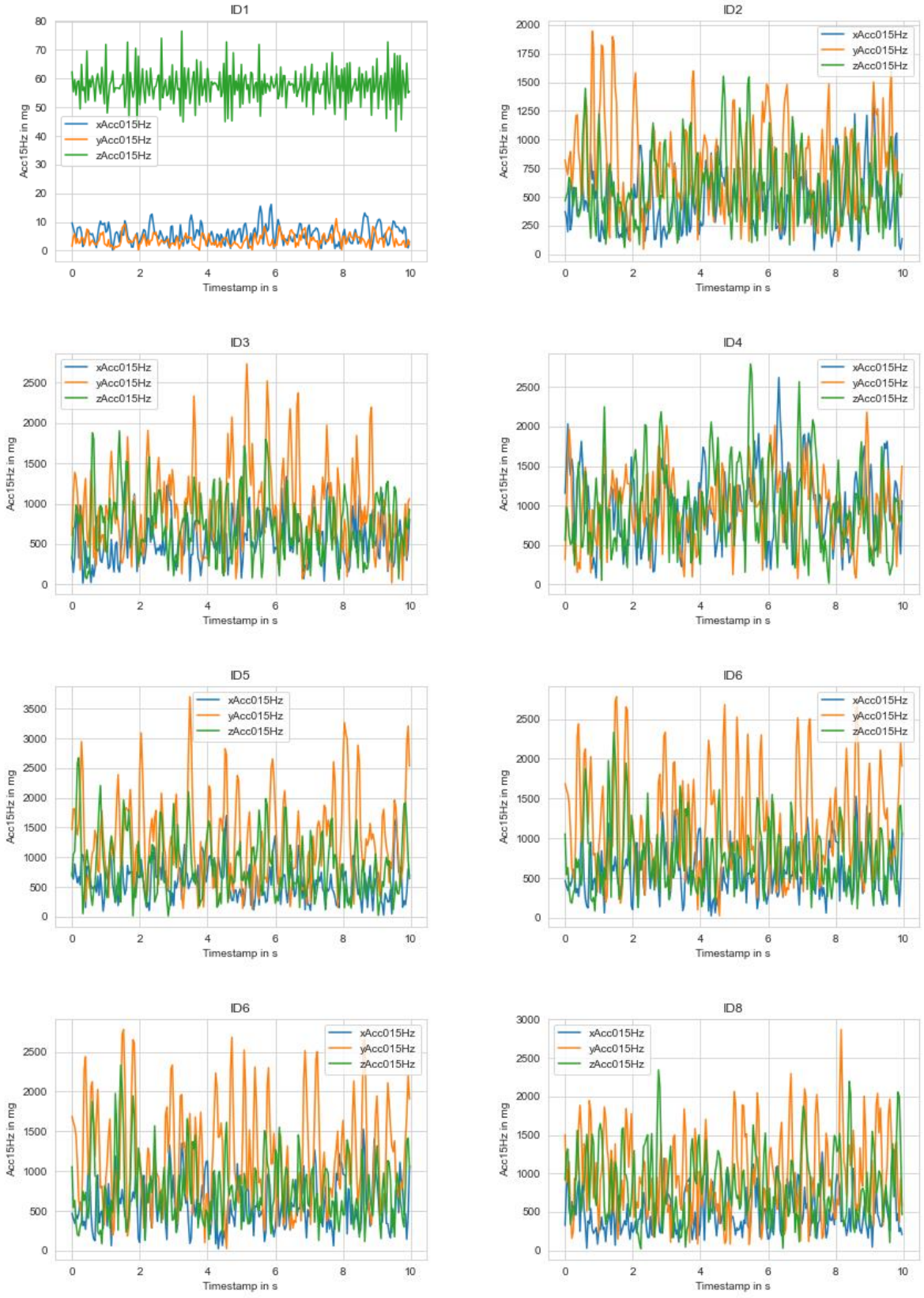


Abbildung 7.3 Liniendiagramme Acc15Hz

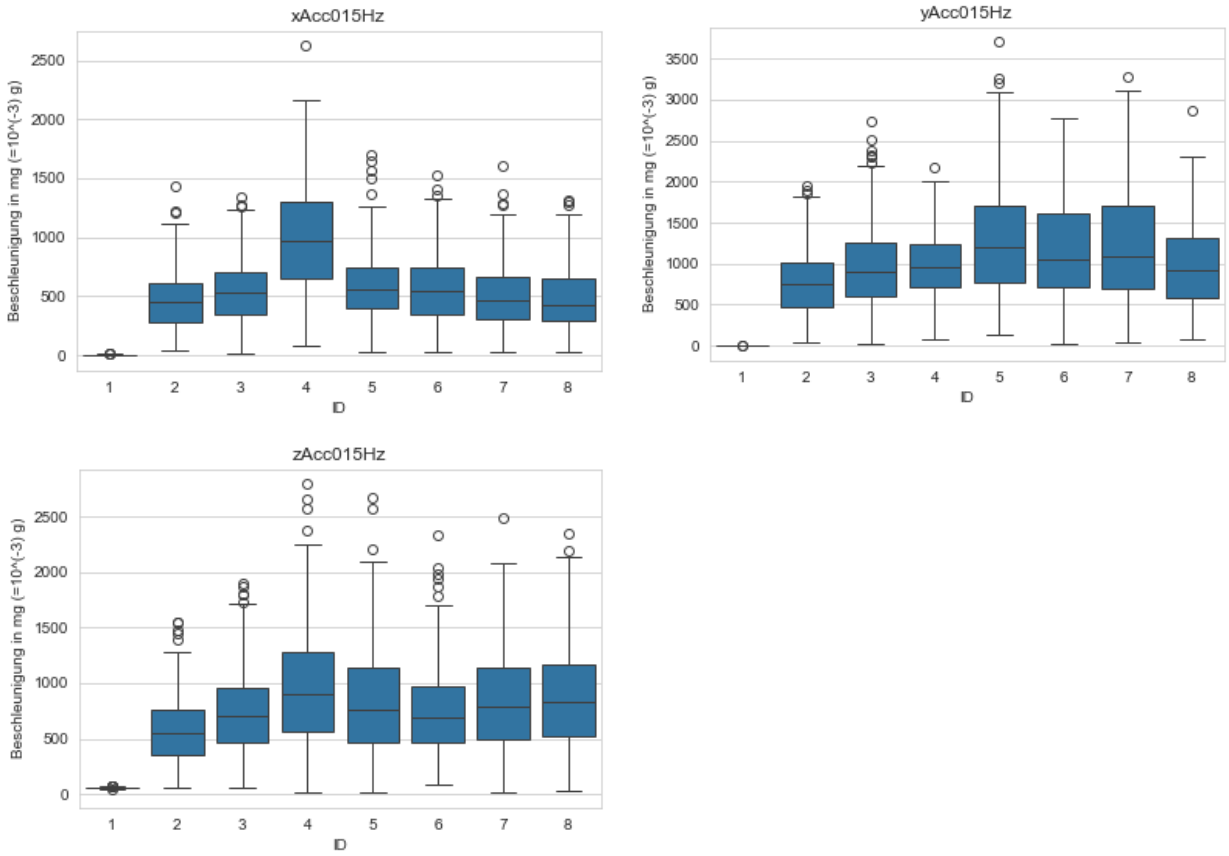
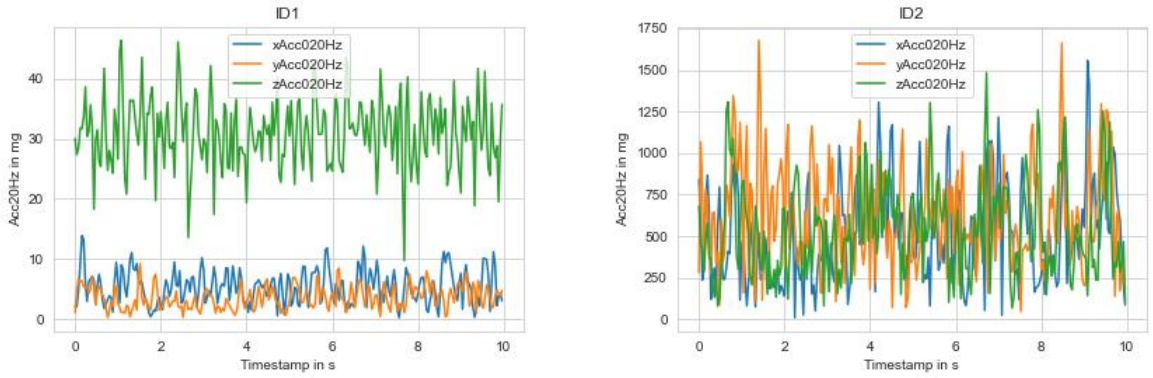


Abbildung 7.4 Boxplots Acc15Hz

A.3 Acc20Hz



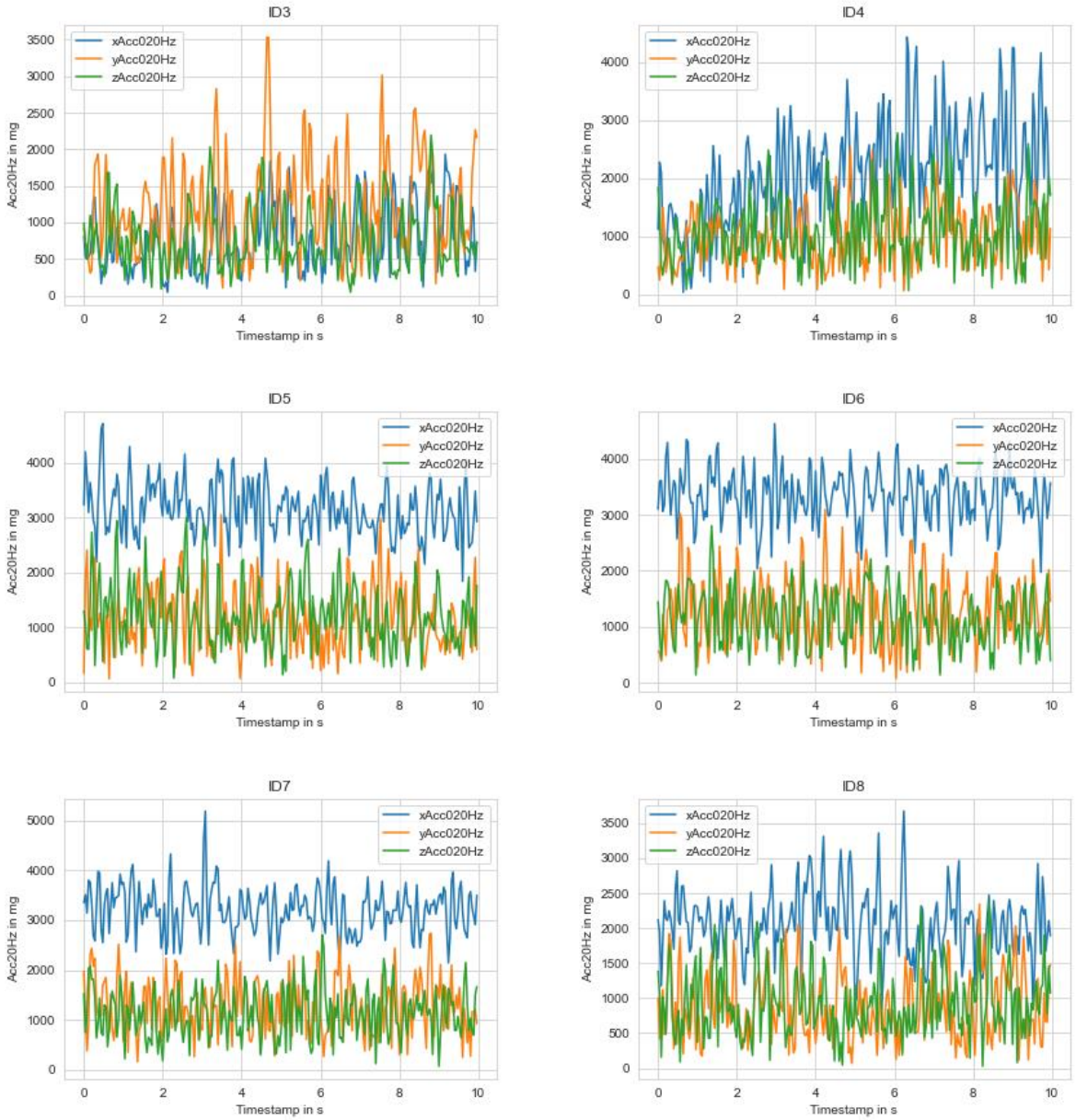
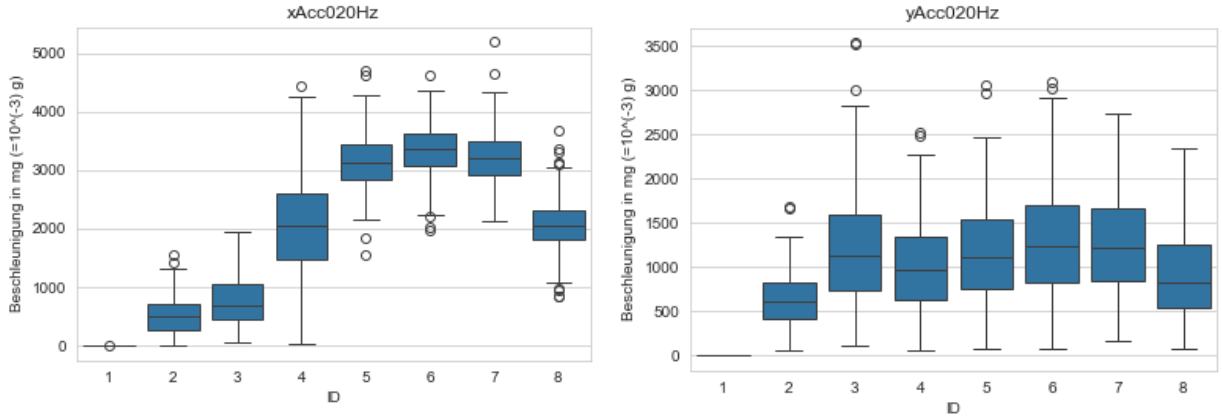


Abbildung 7.5 Liniendiagramme Acc20Hz



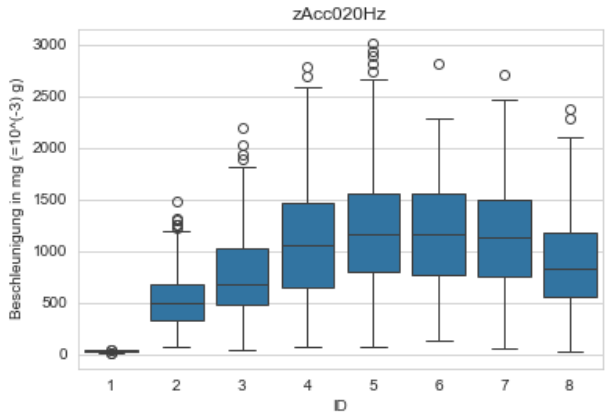
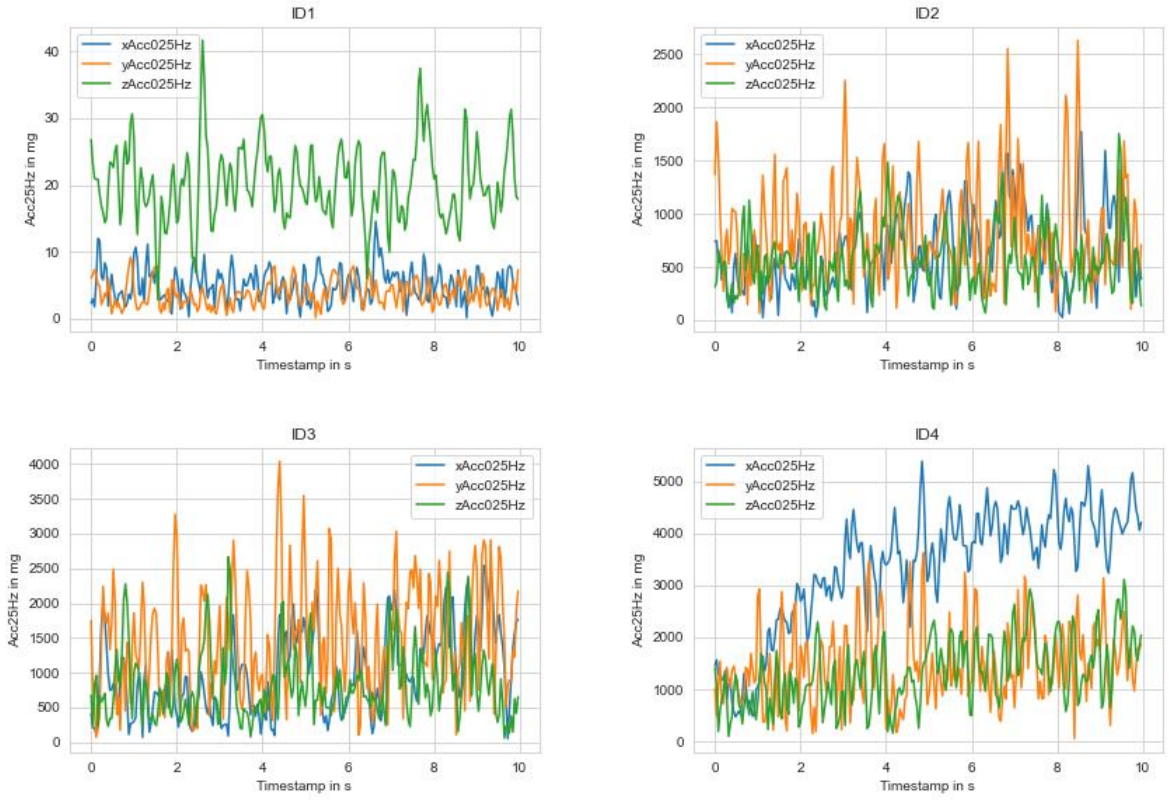


Abbildung 7.6 Boxplots Acc20Hz

A.4 Acc25Hz



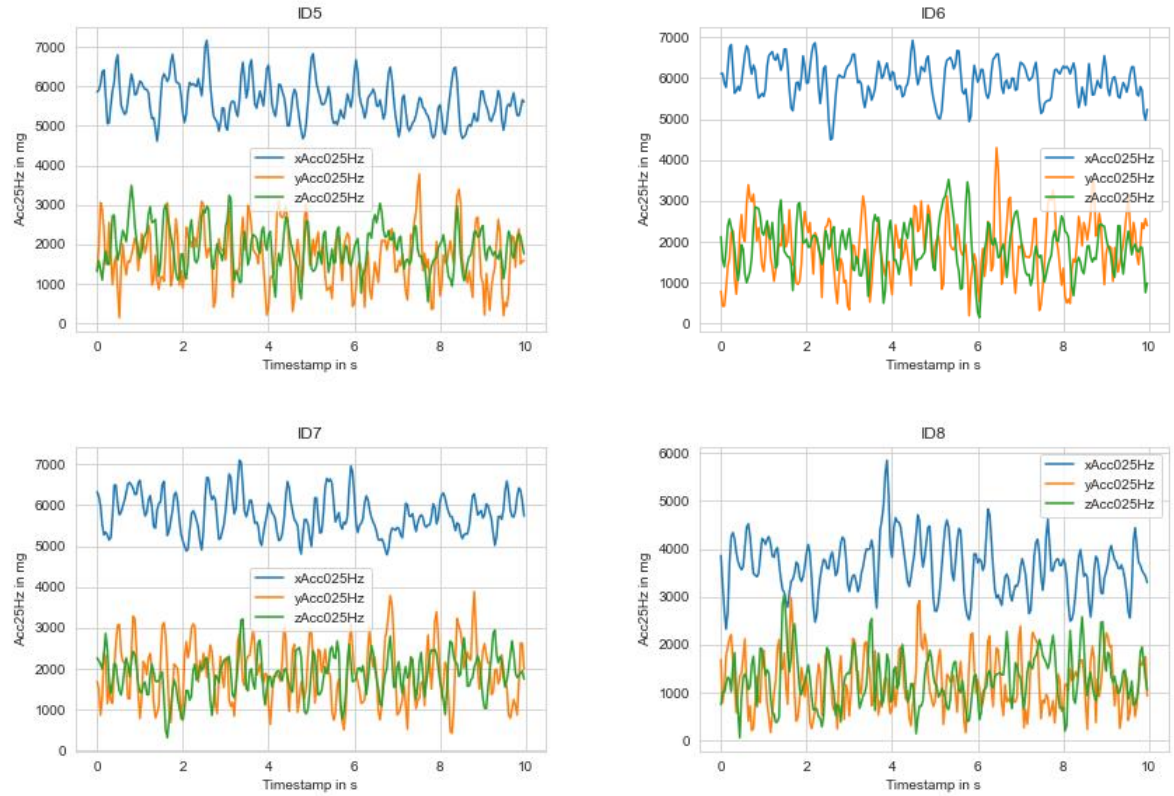


Abbildung 7.7 Liniendiagramme Acc25Hz

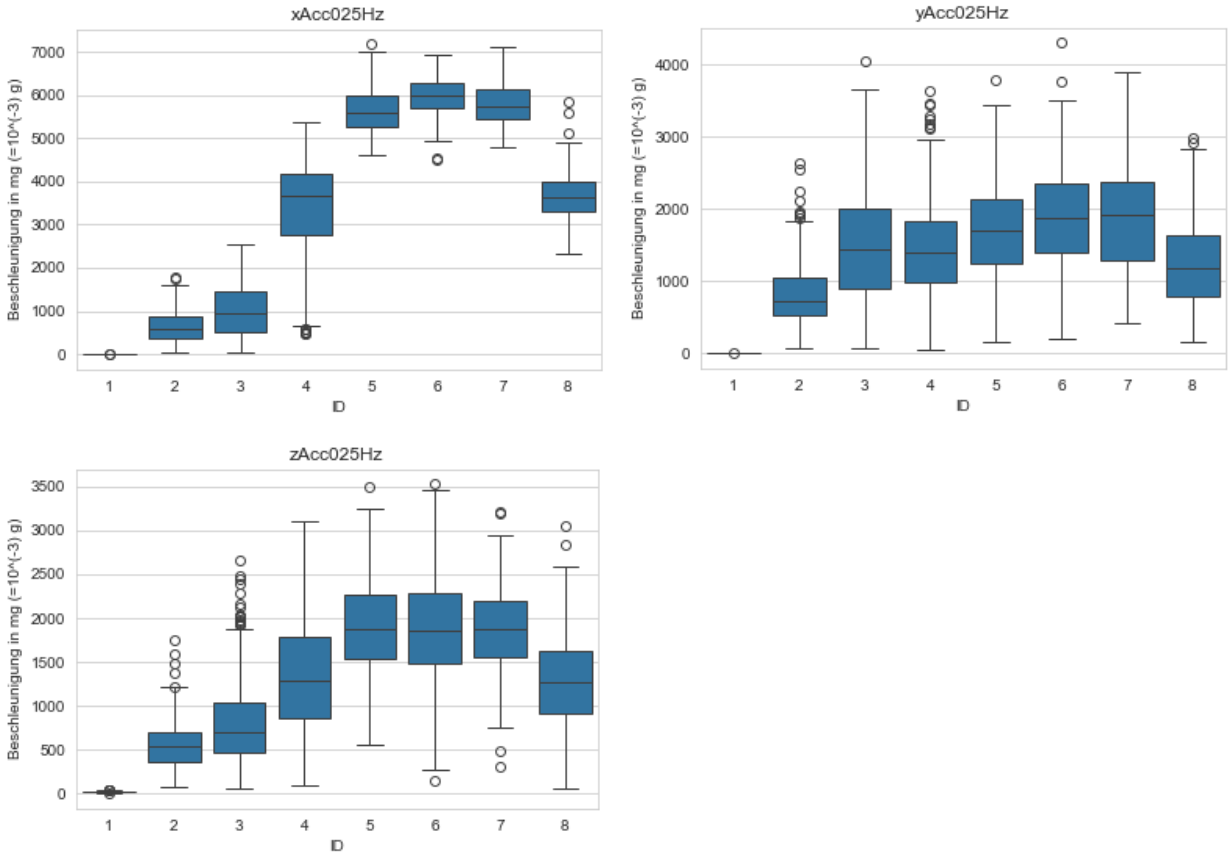


Abbildung 7.8 Boxplots Acc25Hz

A.5 Acc30Hz

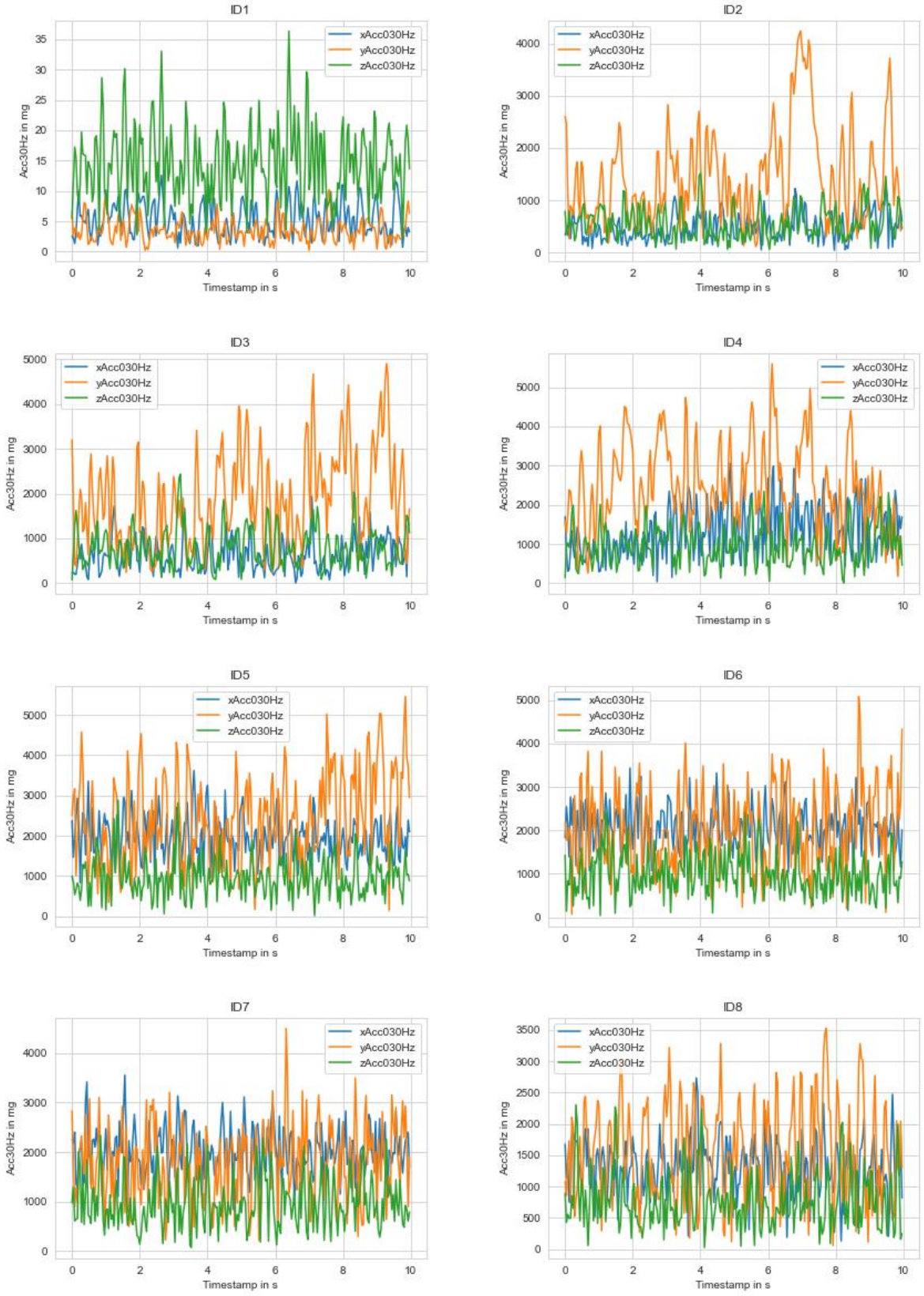

ID1
ID2
ID3
ID4
ID5
ID6
ID7
ID8
xAcc030Hz
yAcc030Hz
zAcc030Hz
Acc30Hz in mg
Timestamp in s

Abbildung 7.9 Liniendiagramme Acc30Hz

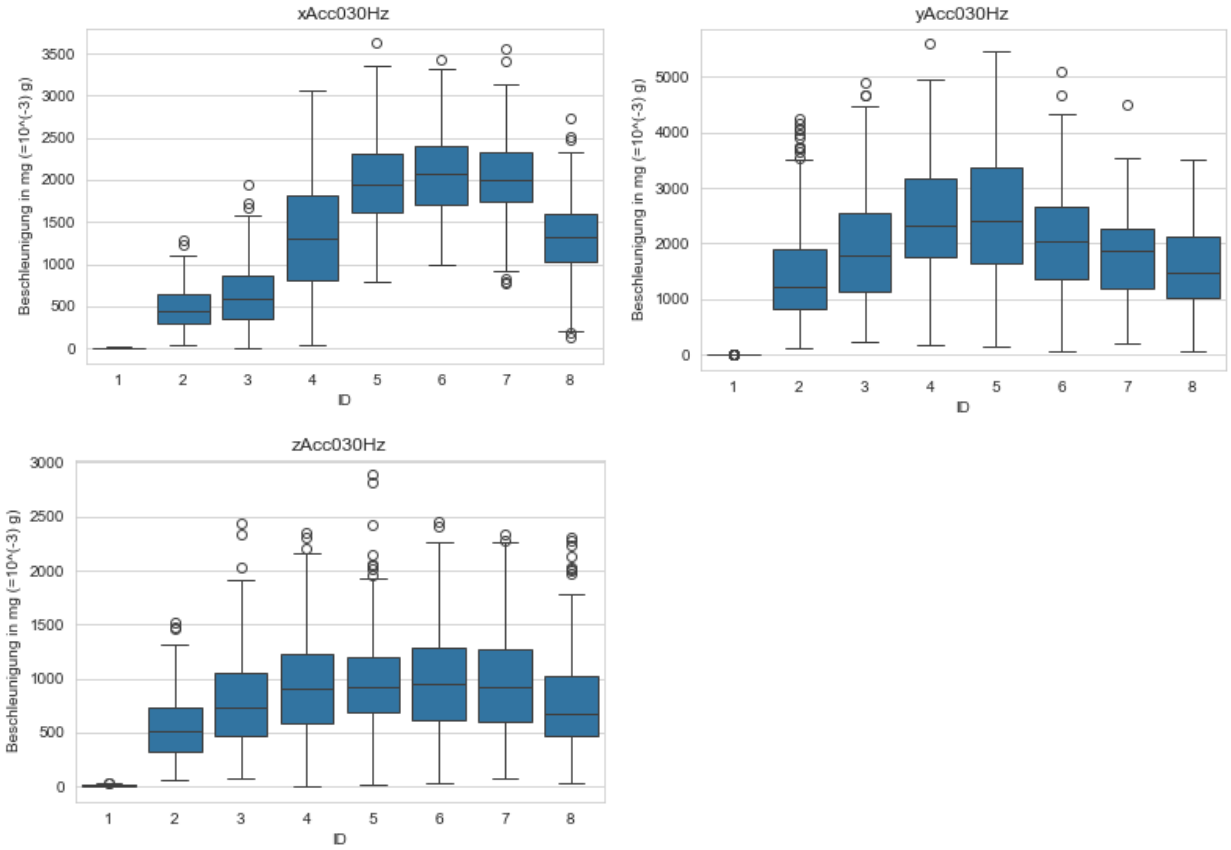
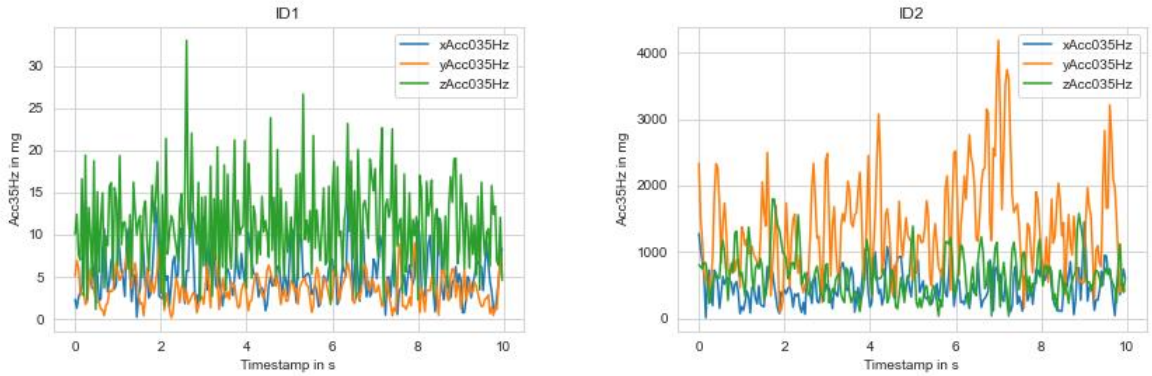


Abbildung 7.10 Boxplots Acc30Hz

A.6 Acc35Hz



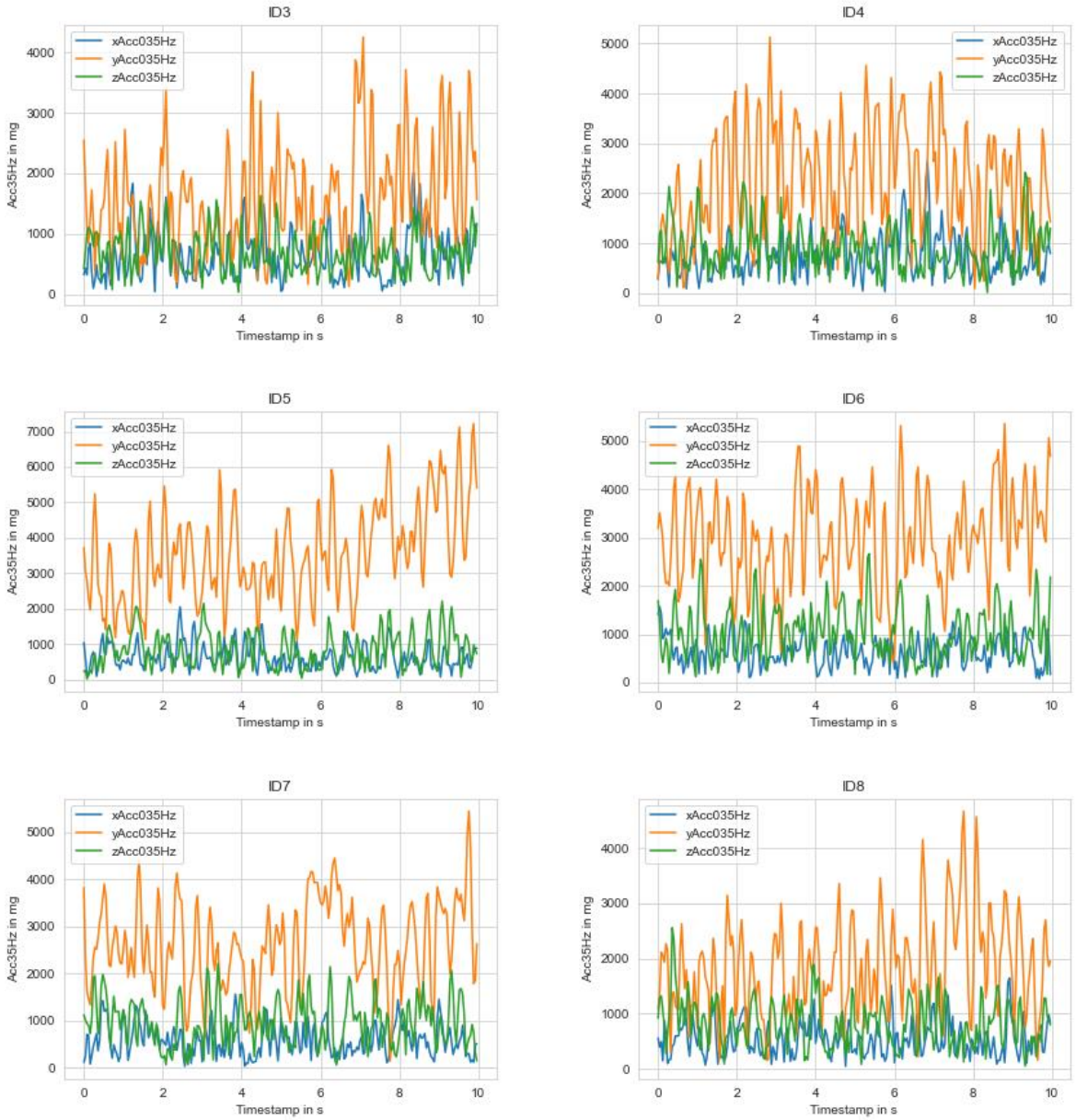
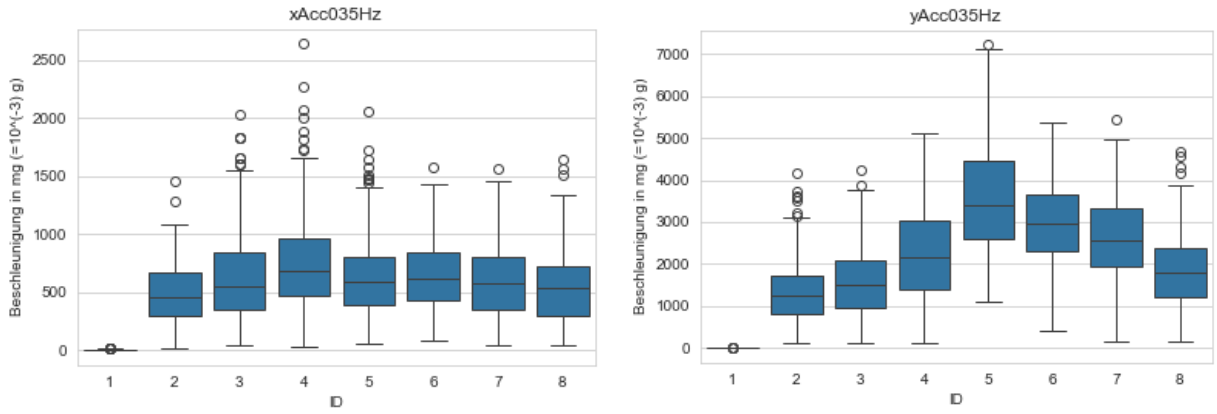


Abbildung 7.11 Liniendiagramme Acc35Hz



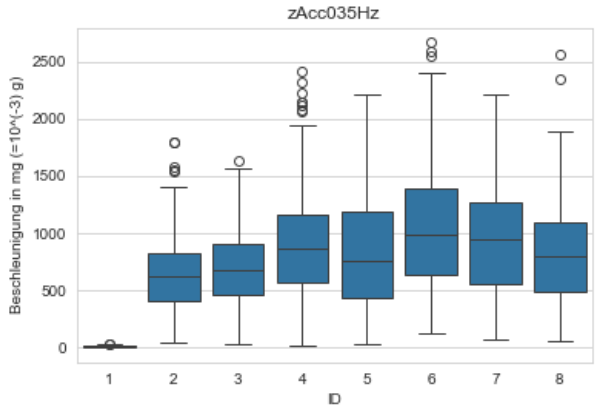
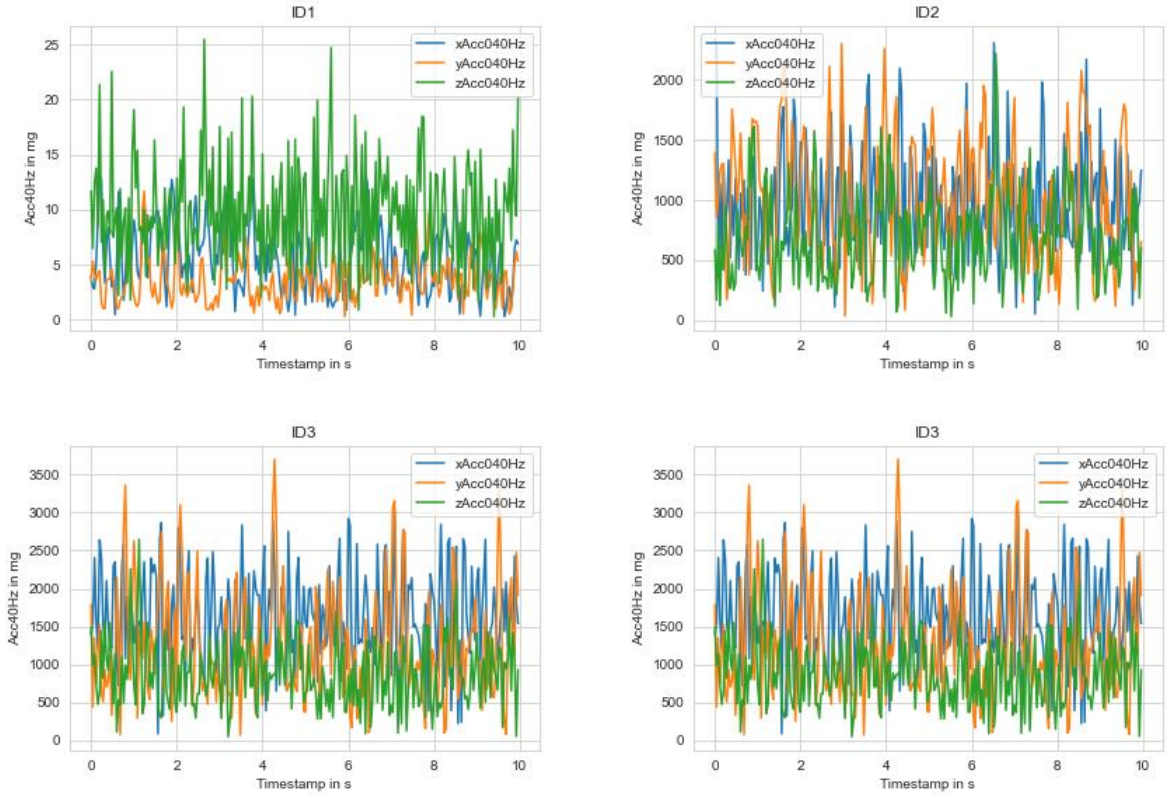


Abbildung 7.12 Boxplots Acc35Hz

A.7 Acc40Hz



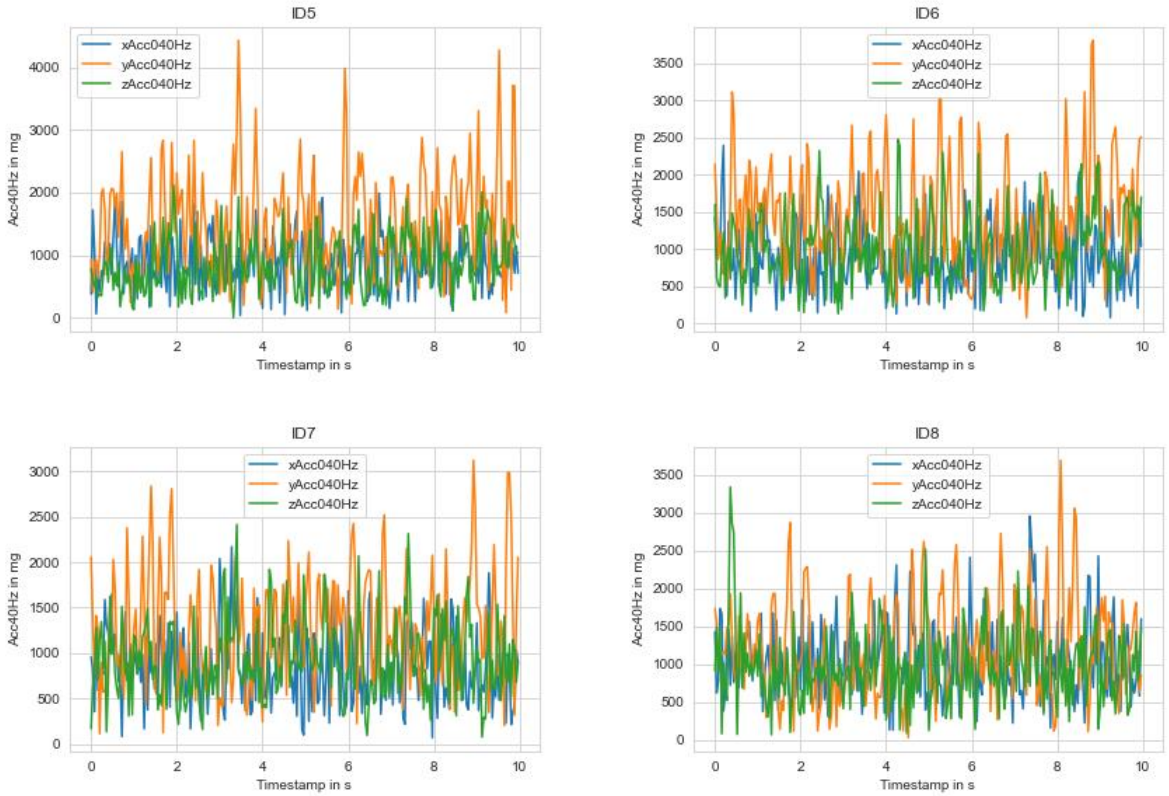


Abbildung 7.13 Liniendiagramme Acc40Hz

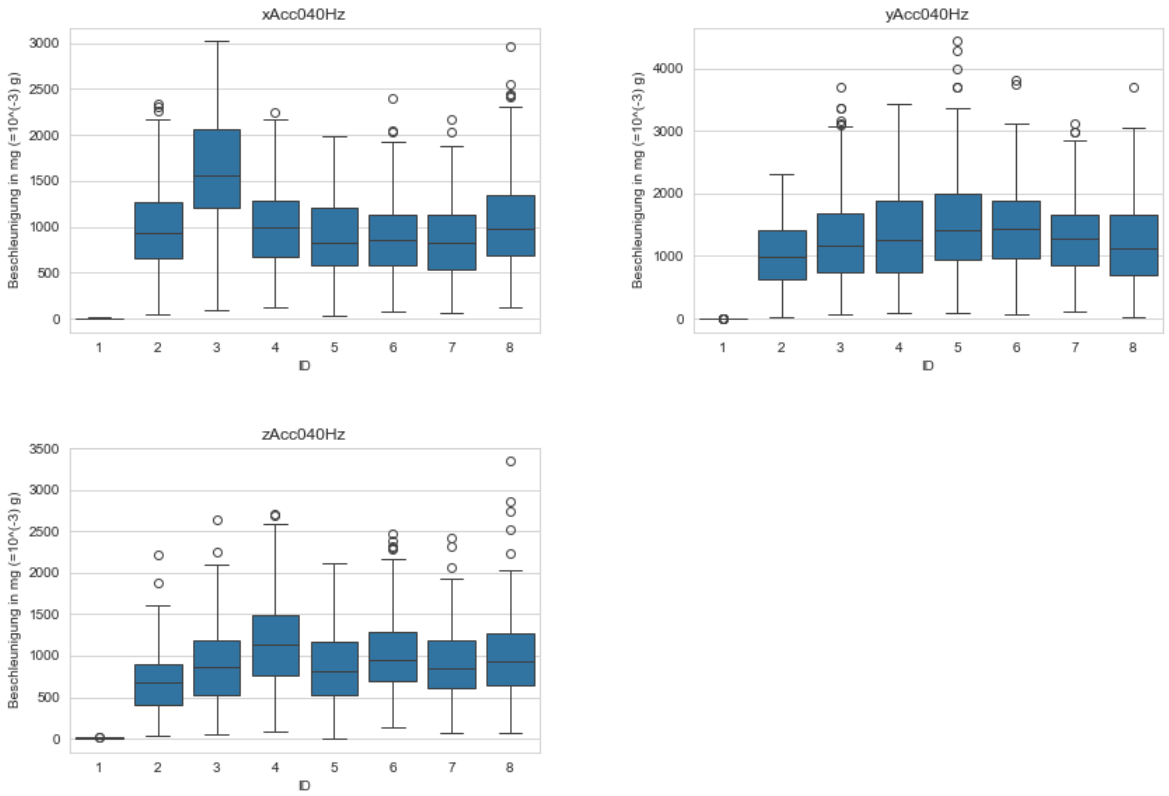


Abbildung 7.14 Boxplots Acc40Hz

A.8 Acc45Hz

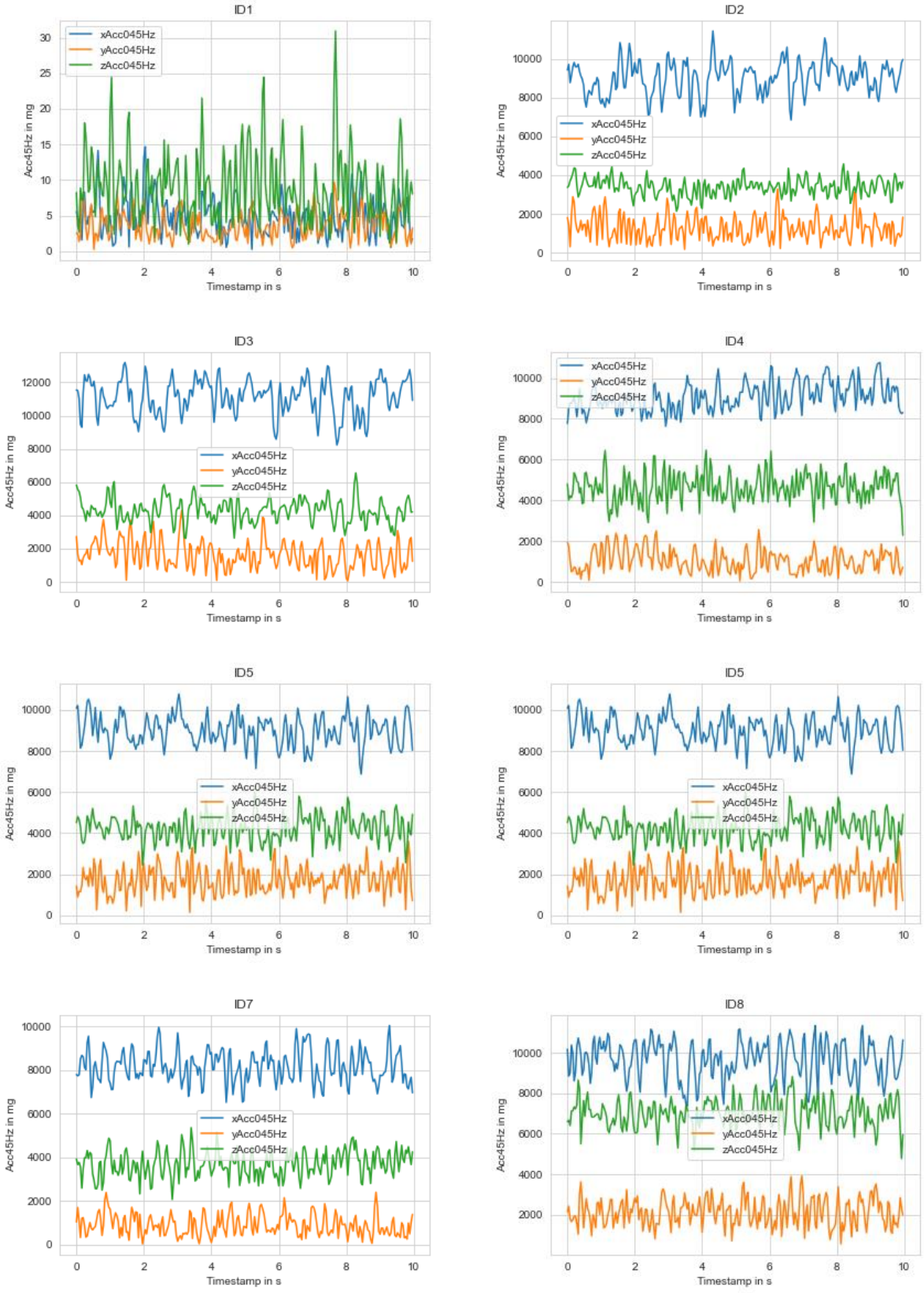


Abbildung 7.15 Liniendiagramme Acc45Hz

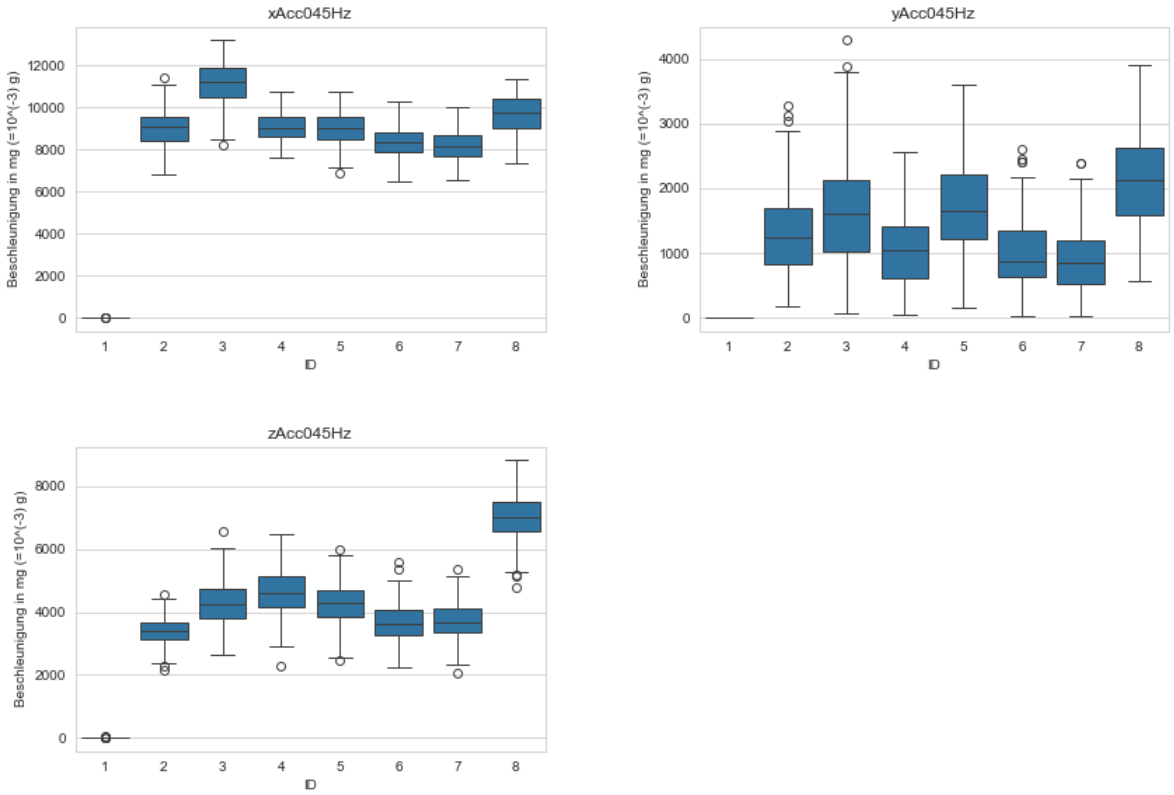
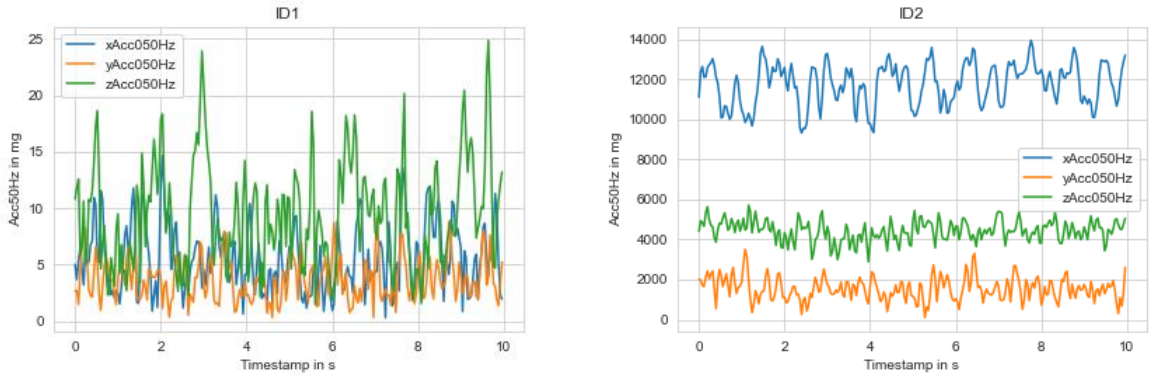


Abbildung 7.16 Boxplots Acc45Hz

A.9 Acc50Hz



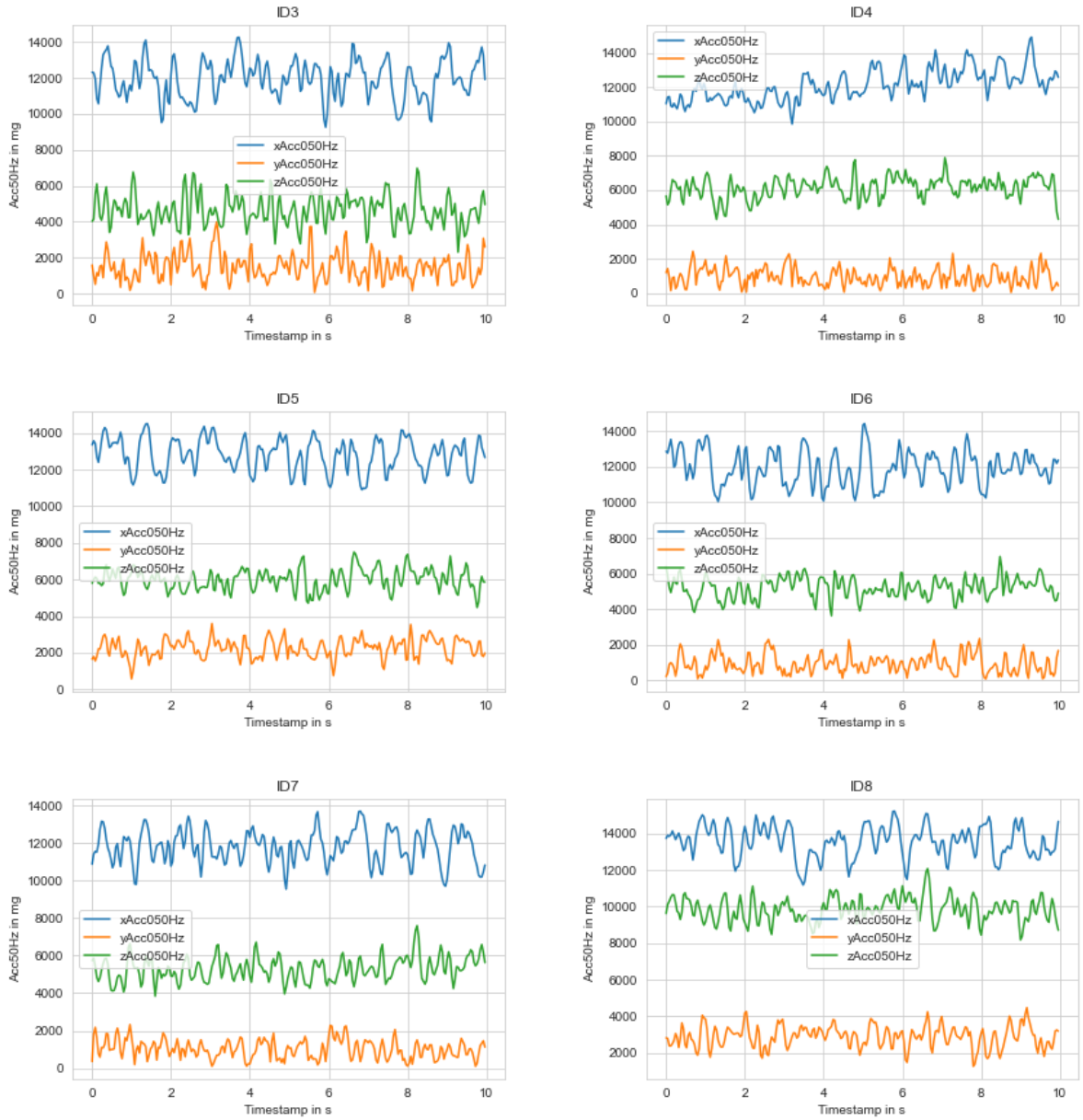
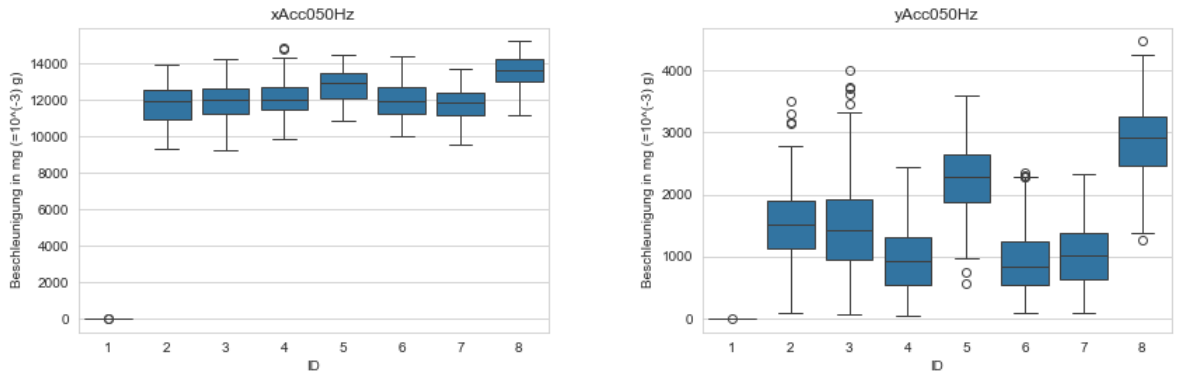


Abbildung 7.17 Liniendiagramme Acc50Hz



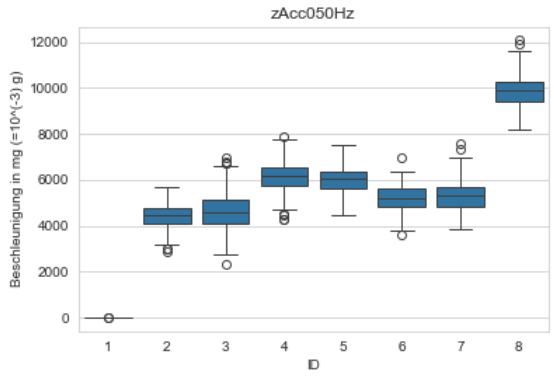
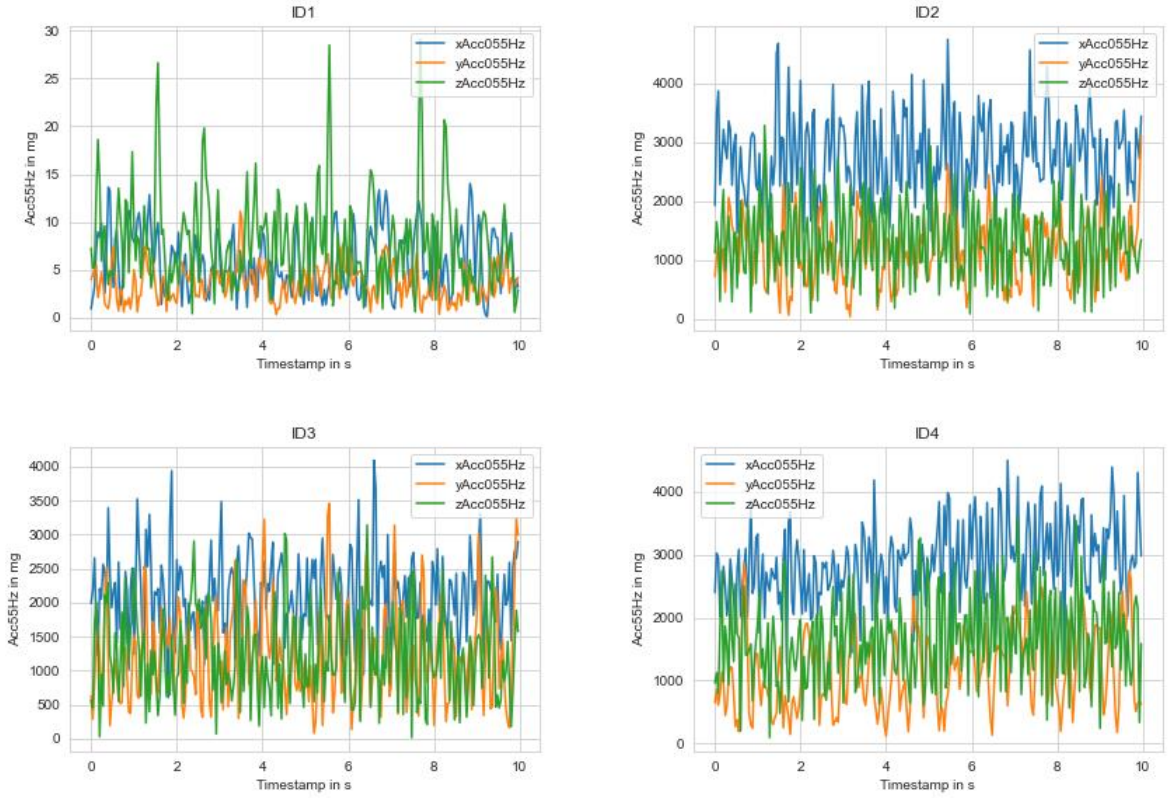


Abbildung 7.18 Boxplots Acc50Hz

A.10 Acc55Hz



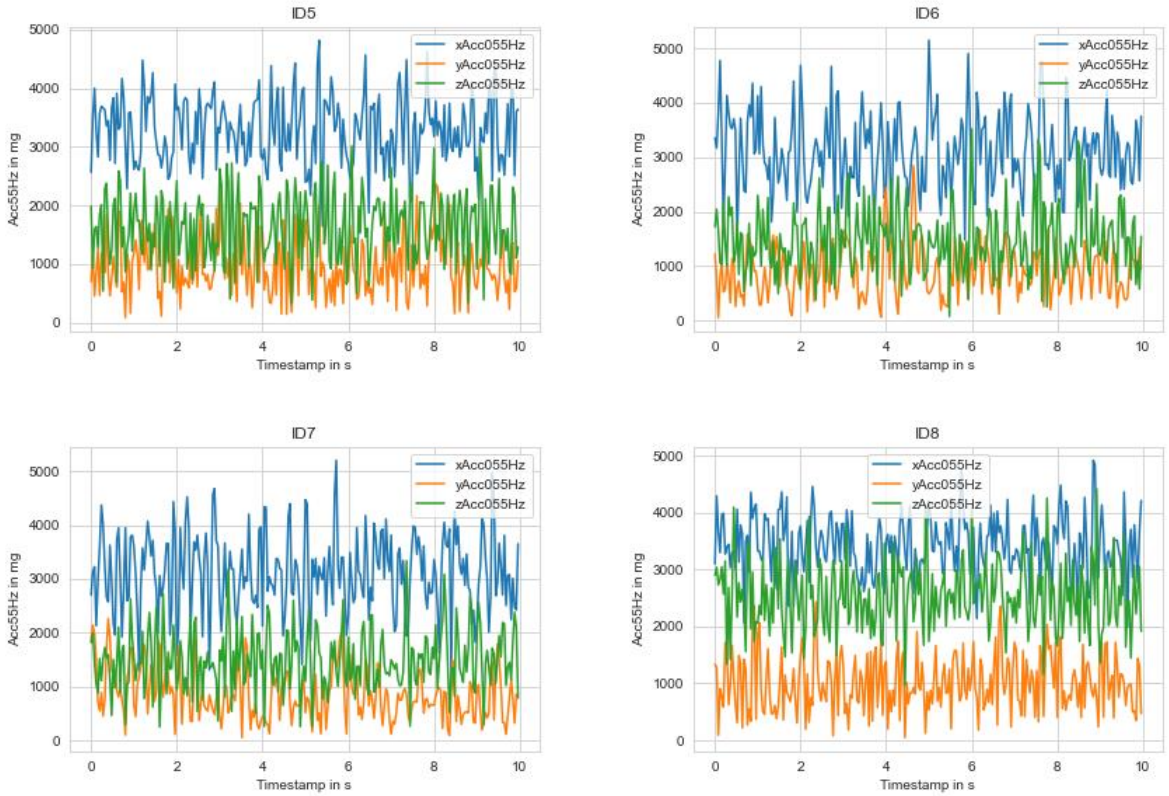
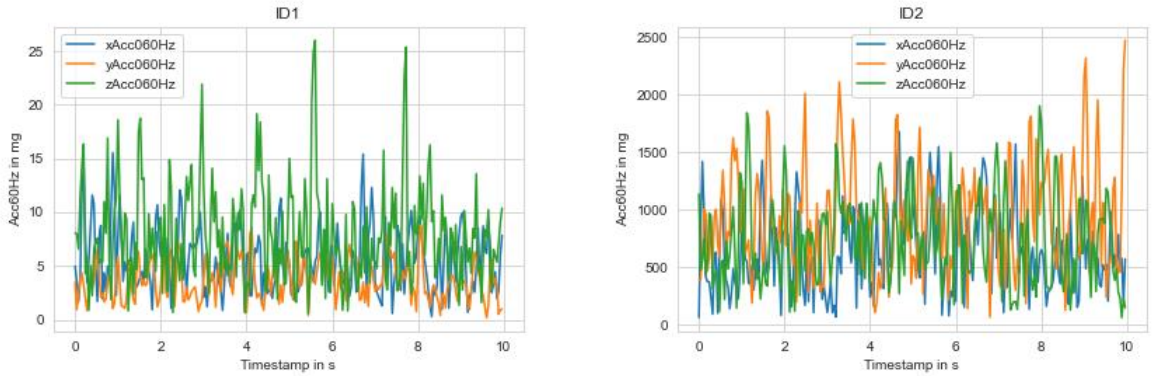


Abbildung 7.19 Liniendiagramme Acc55Hz

A.11 Acc60Hz



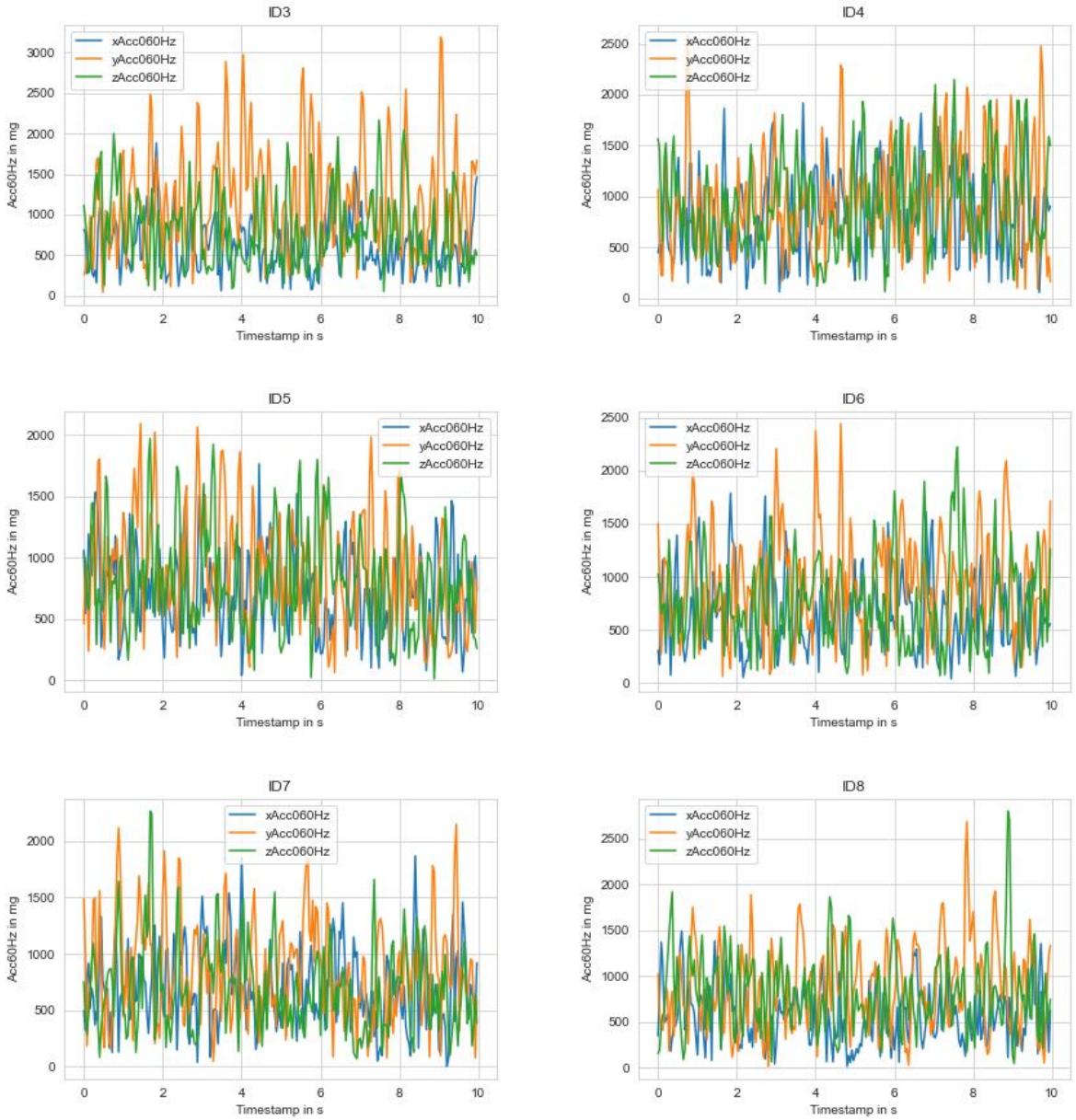
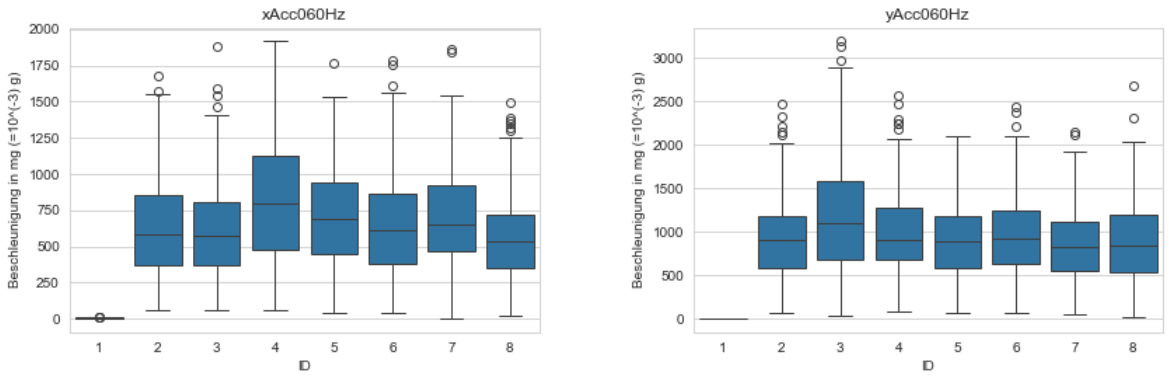


Abbildung 7.20 Liniendiagramme Acc60Hz



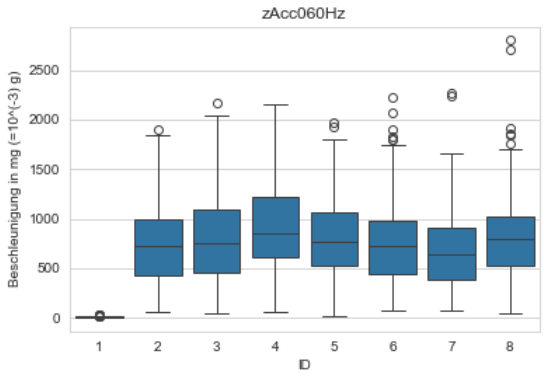
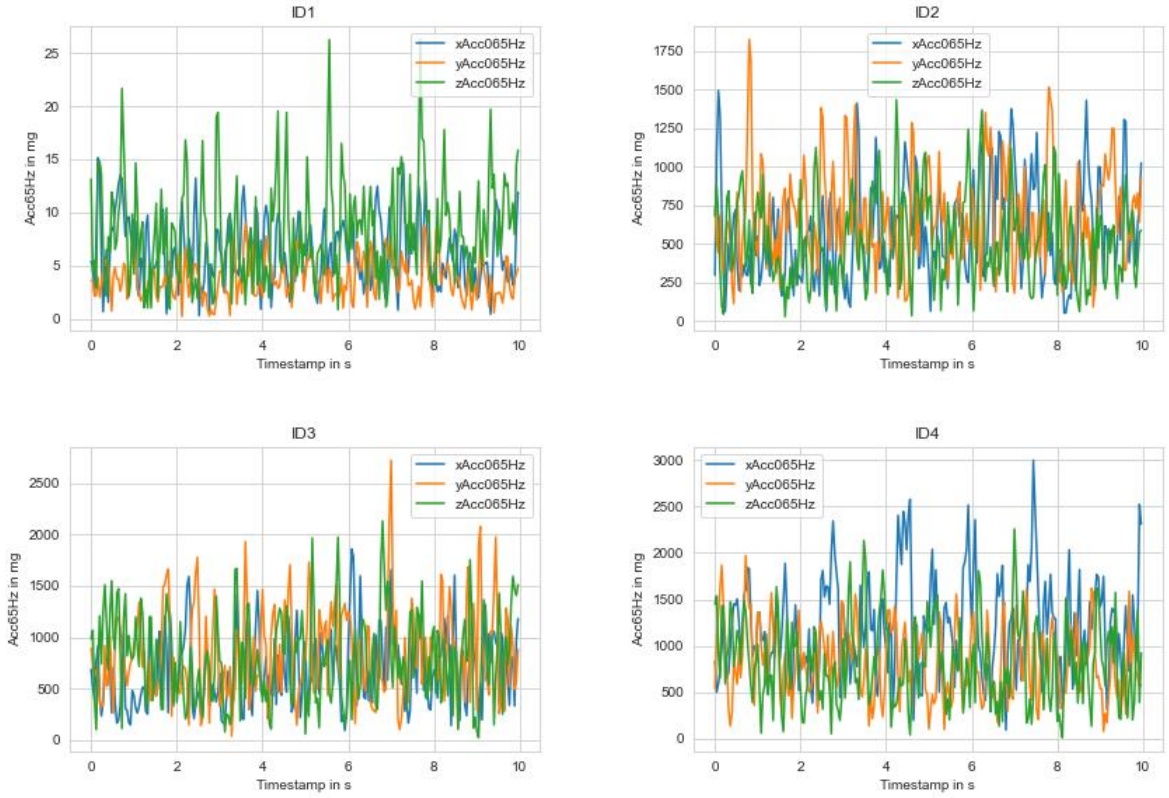


Abbildung 7.21 Boxplots Acc60Hz

A.12 Acc65Hz



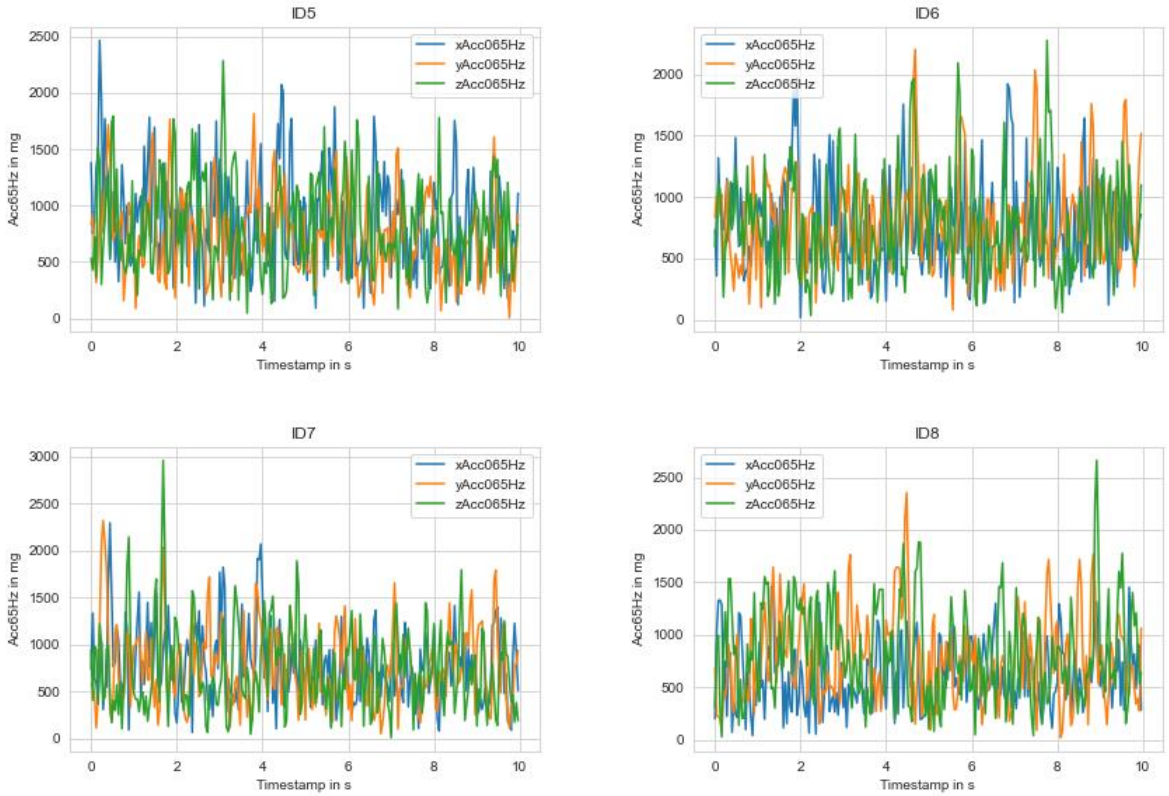


Abbildung 7.22 Liniendiagramme Acc65Hz

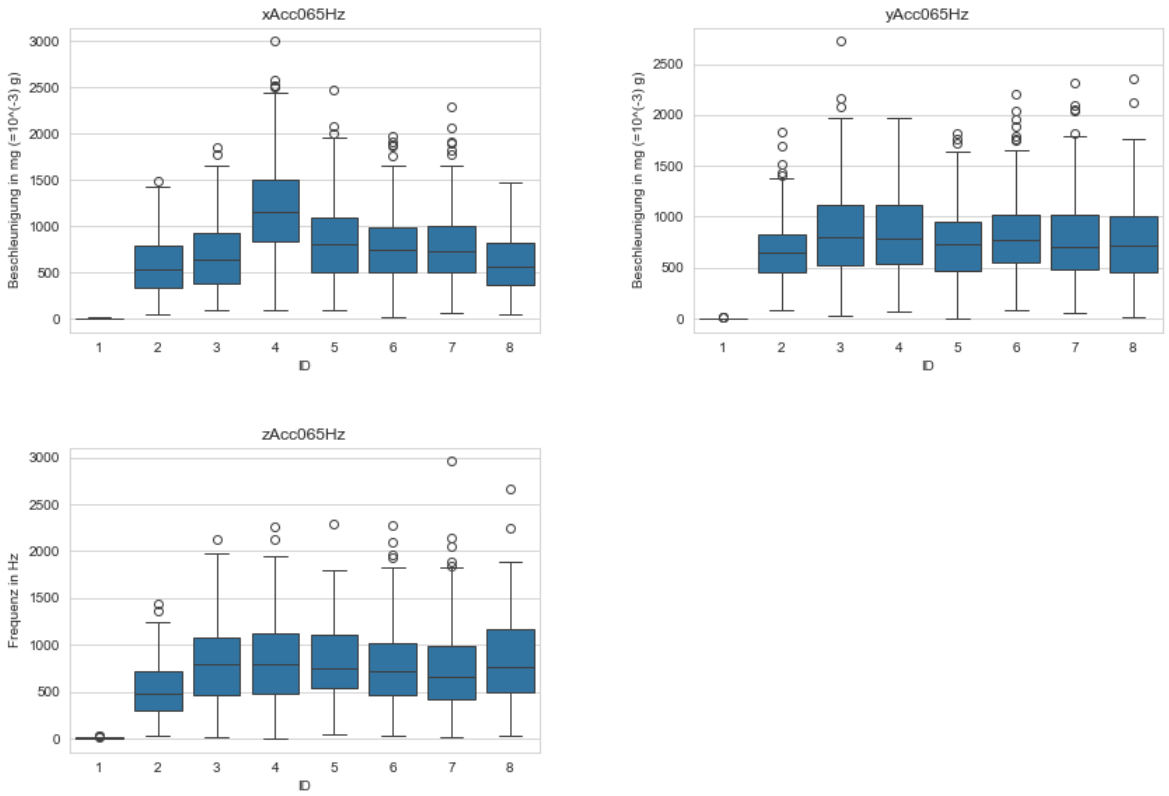


Abbildung 7.23 Boxplots Acc65Hz

A.13 Acc70Hz

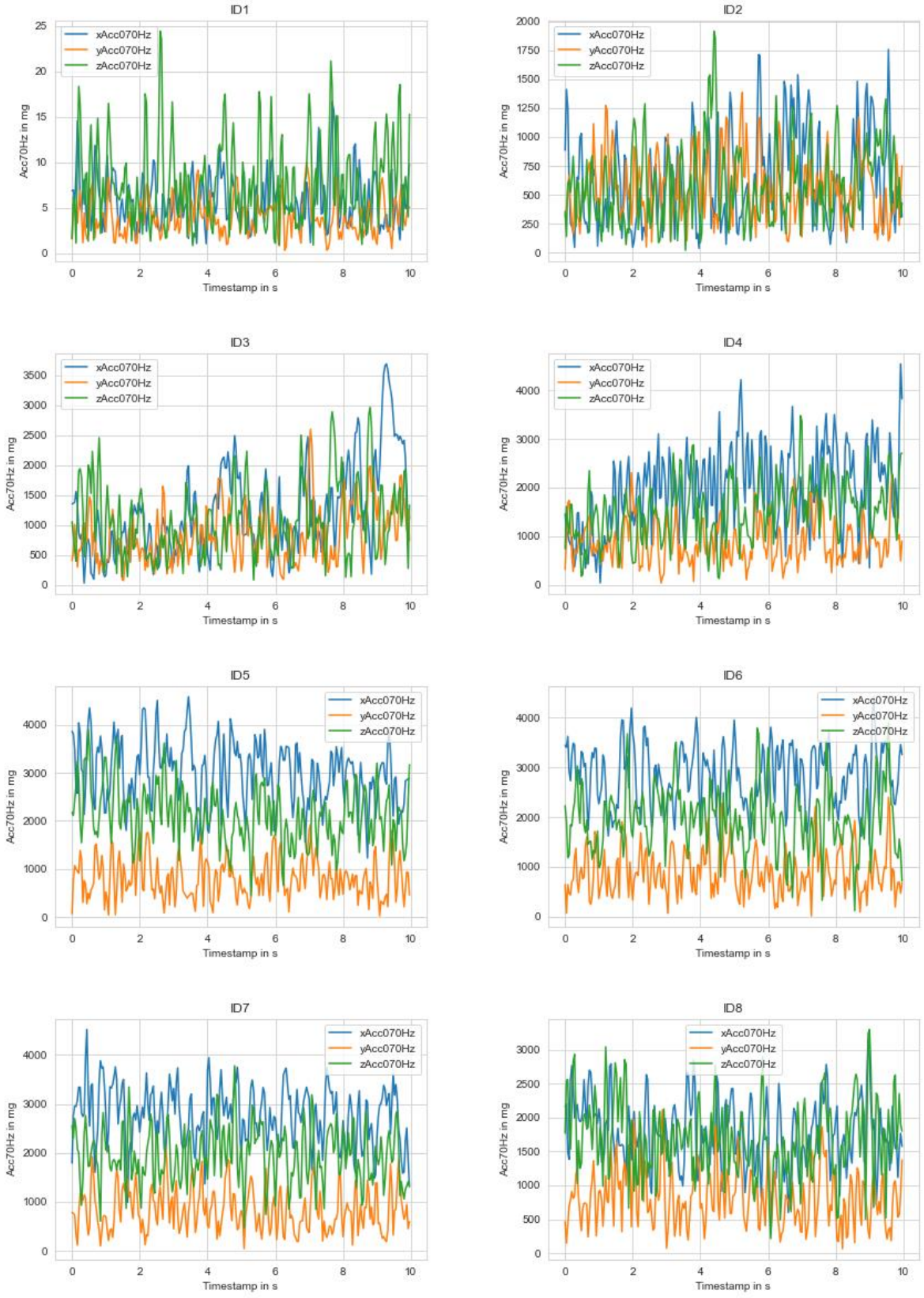


Abbildung 7.24 Liniendiagramme Acc70Hz

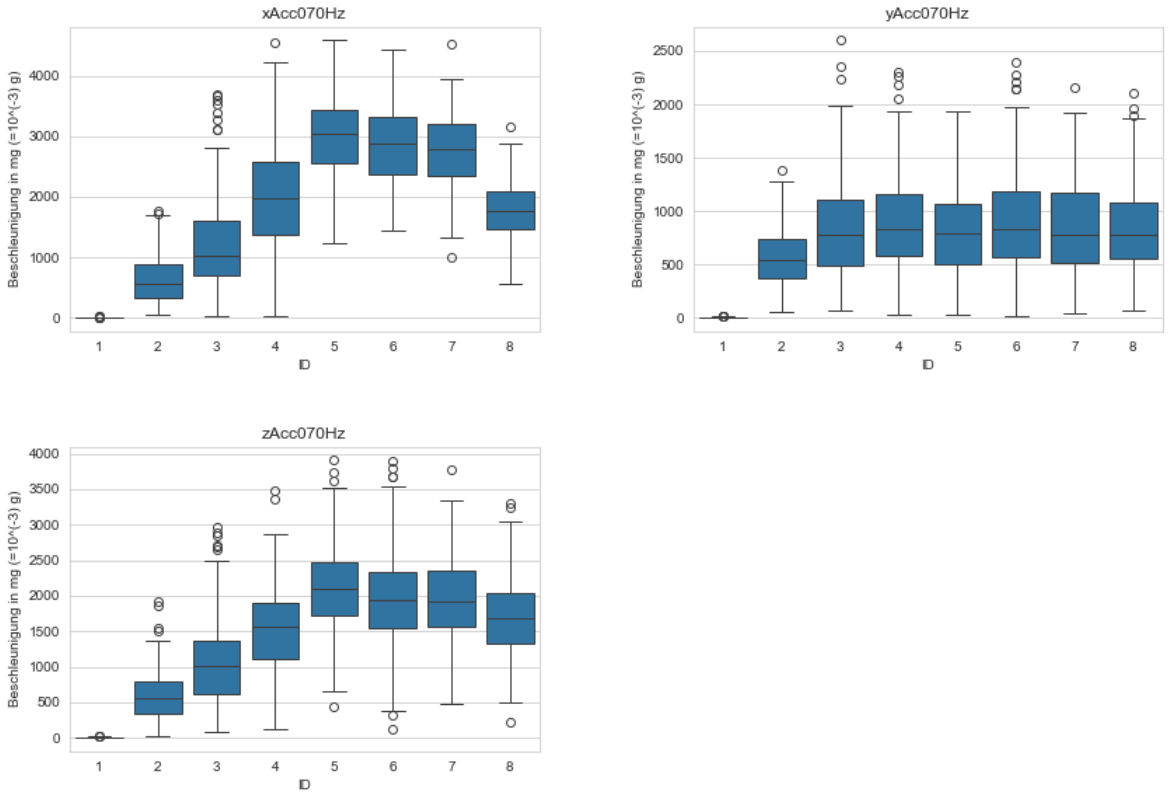
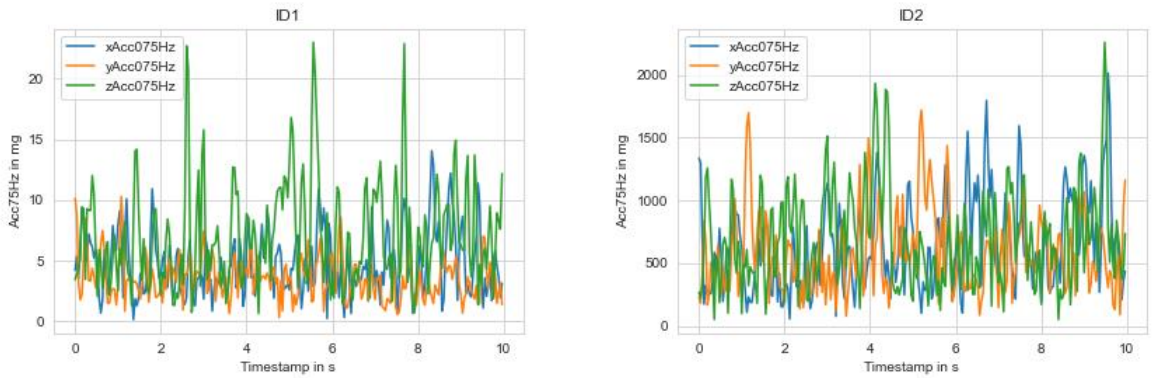


Abbildung 7.25 Boxplots Acc70Hz

A.14 Acc75Hz



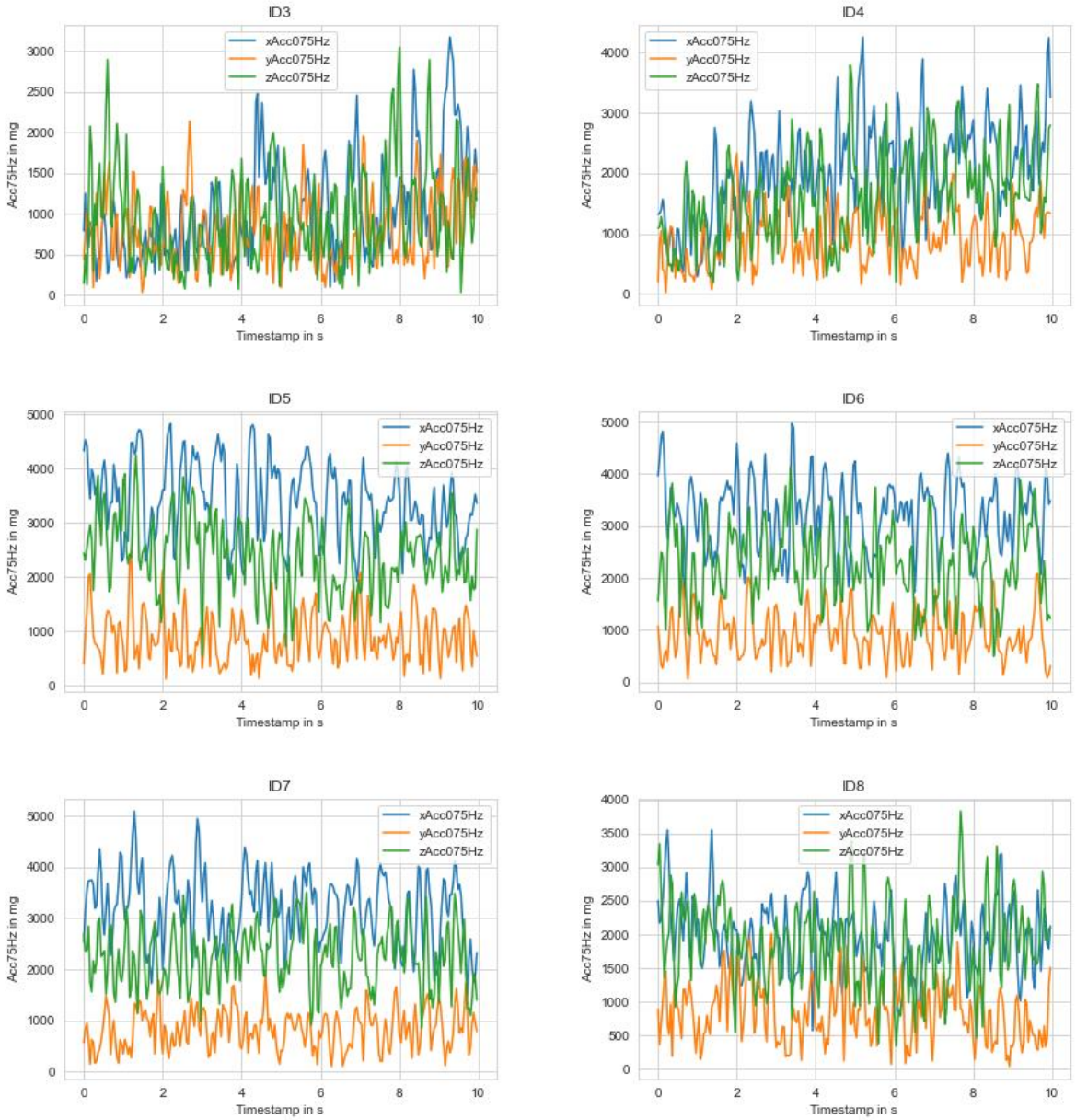
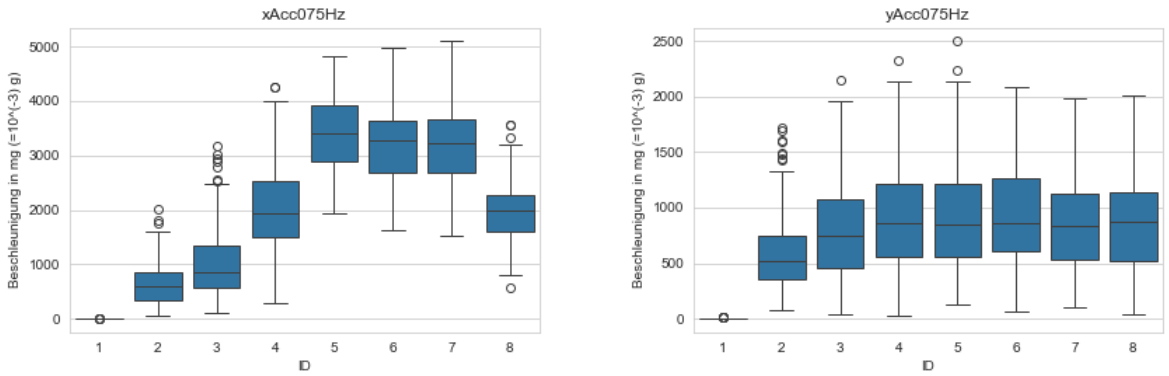


Abbildung 7.26 Liniendiagramme Acc75Hz



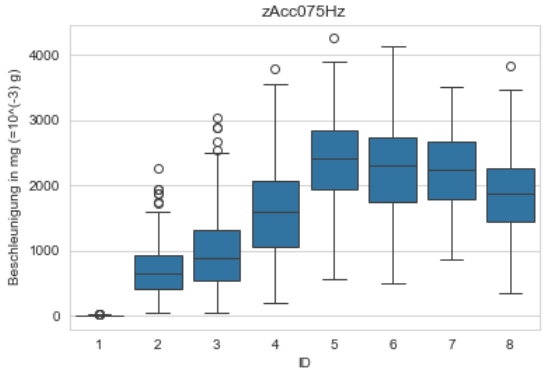
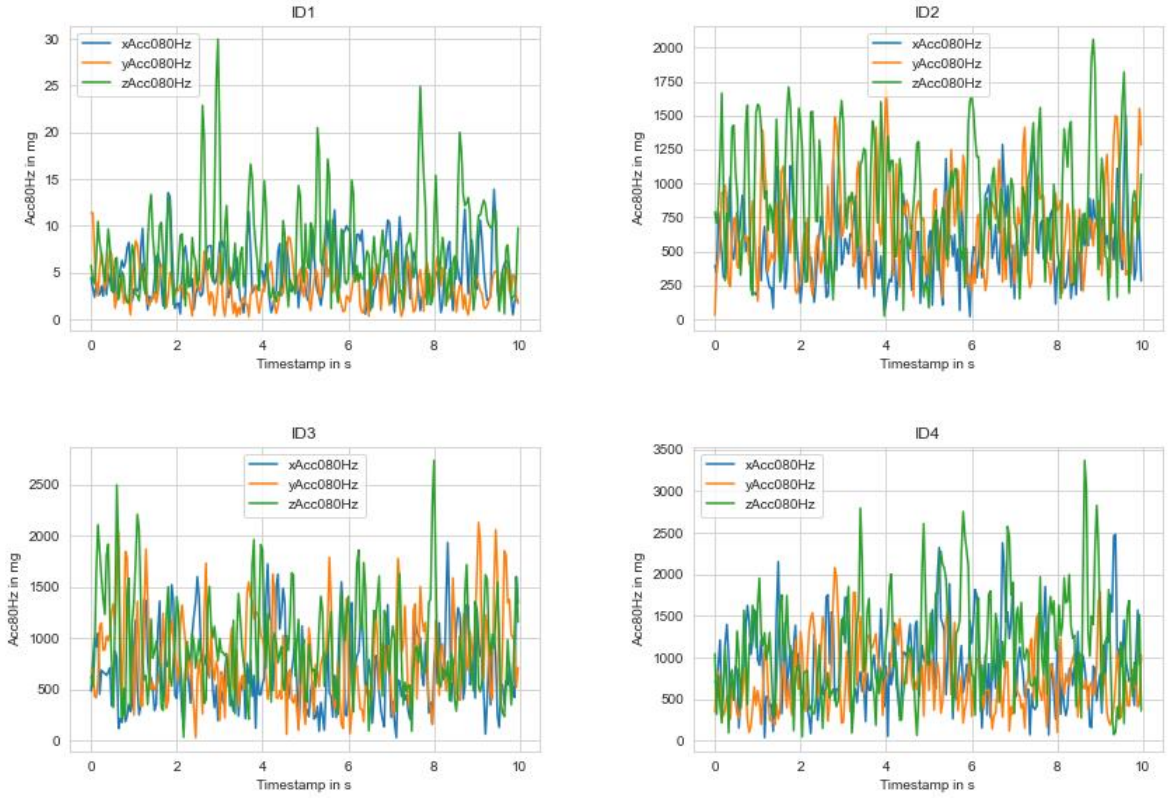


Abbildung 7.27 Boxplots Acc75Hz

A.15 Acc80Hz



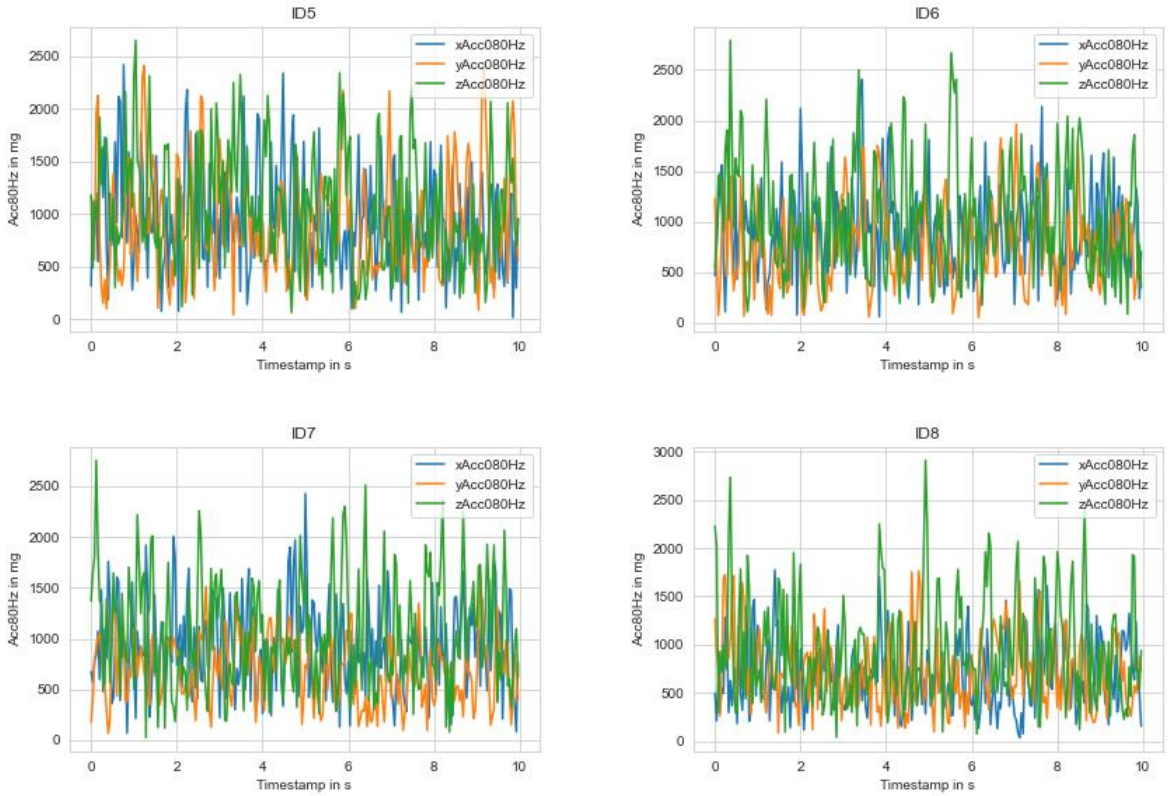


Abbildung 7.28 Liniendiagramme Acc80Hz

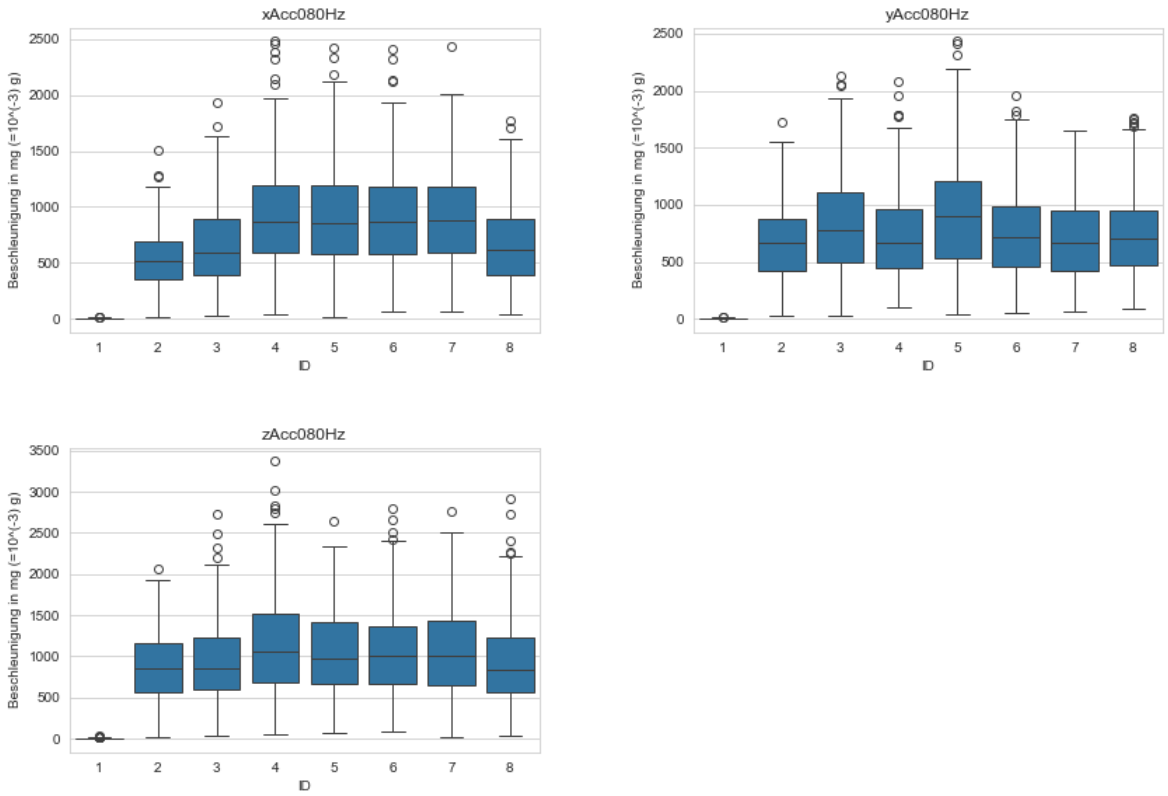


Abbildung 7.29 Boxplots Acc80Hz

A.16 Acc85Hz

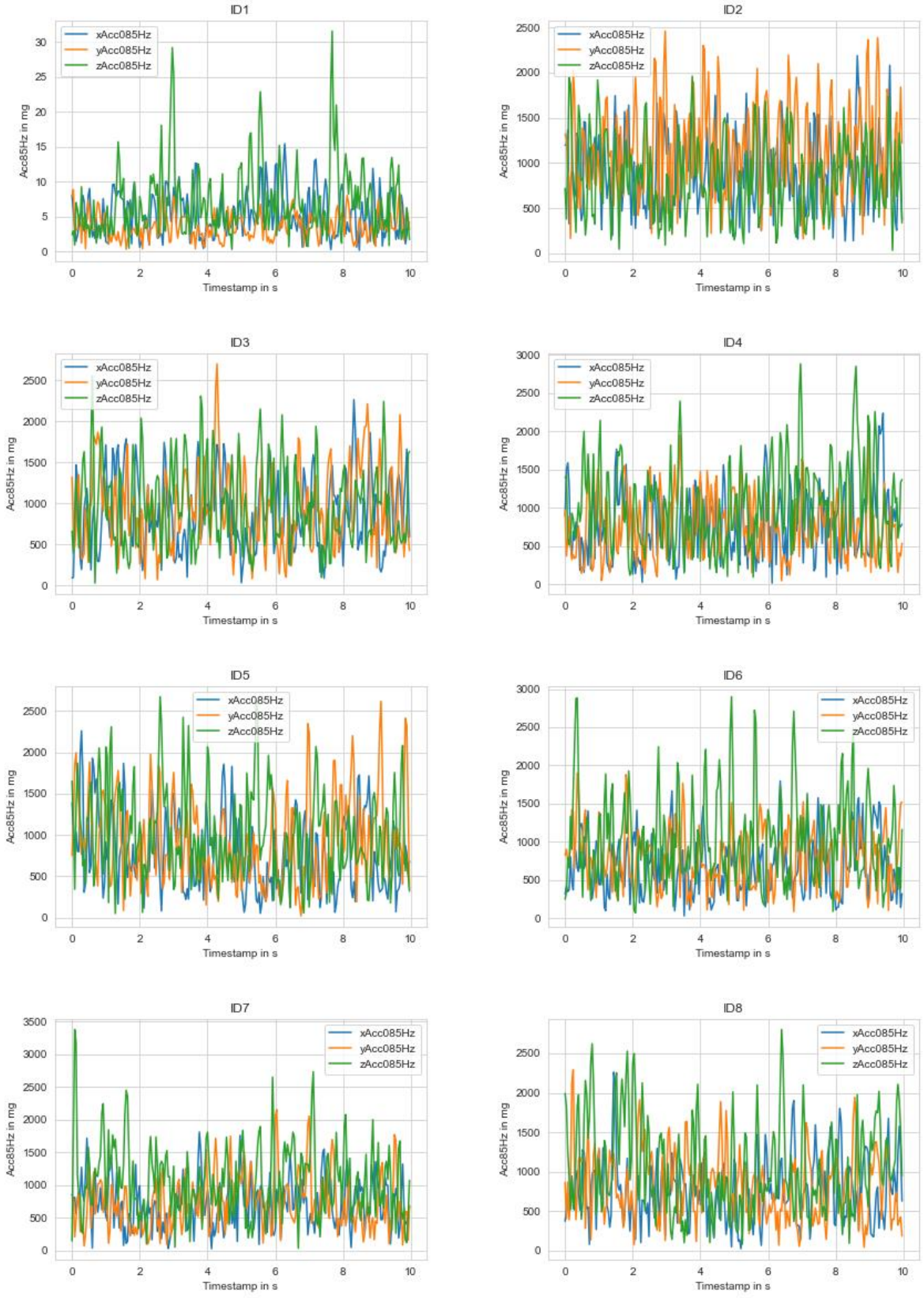


Abbildung 7.30 Liniendiagramme Acc85Hz

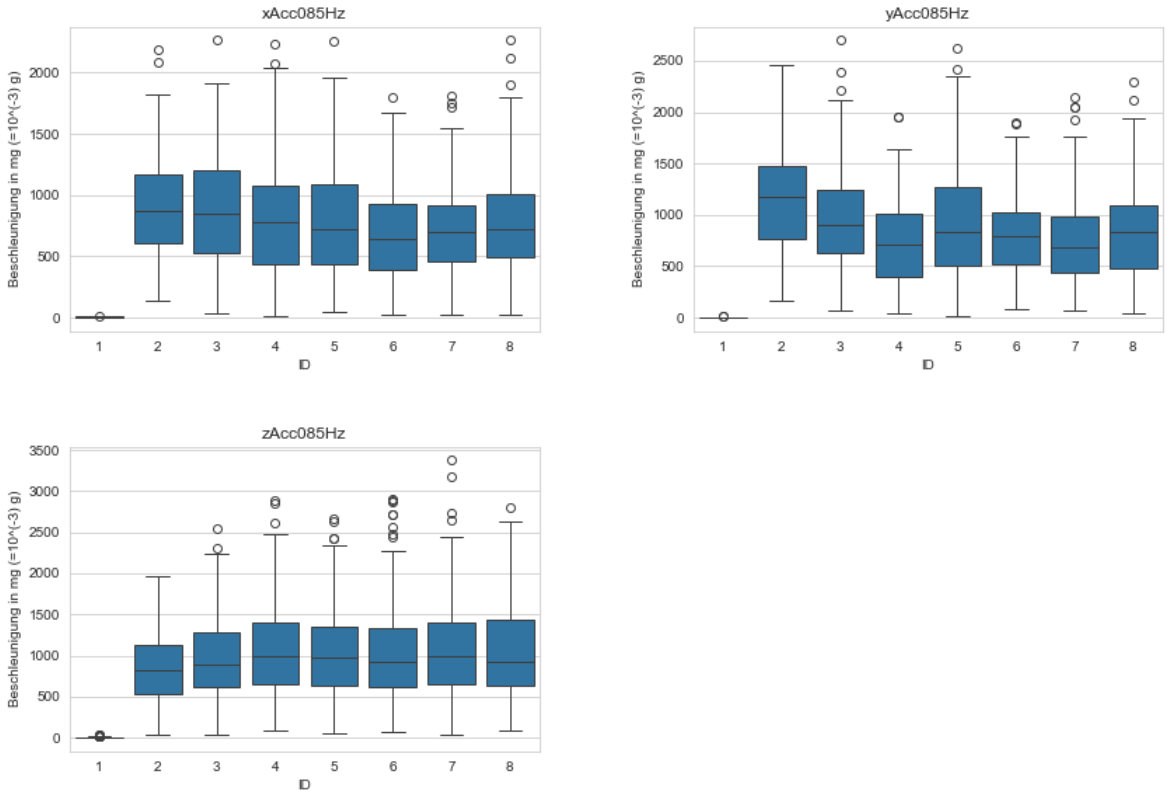
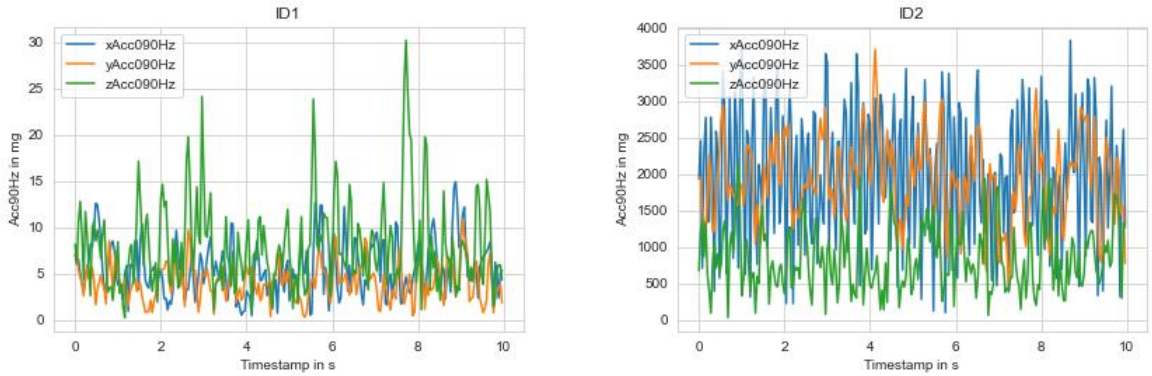


Abbildung 7.31 Boxplots Acc85Hz

A.17 Acc90Hz



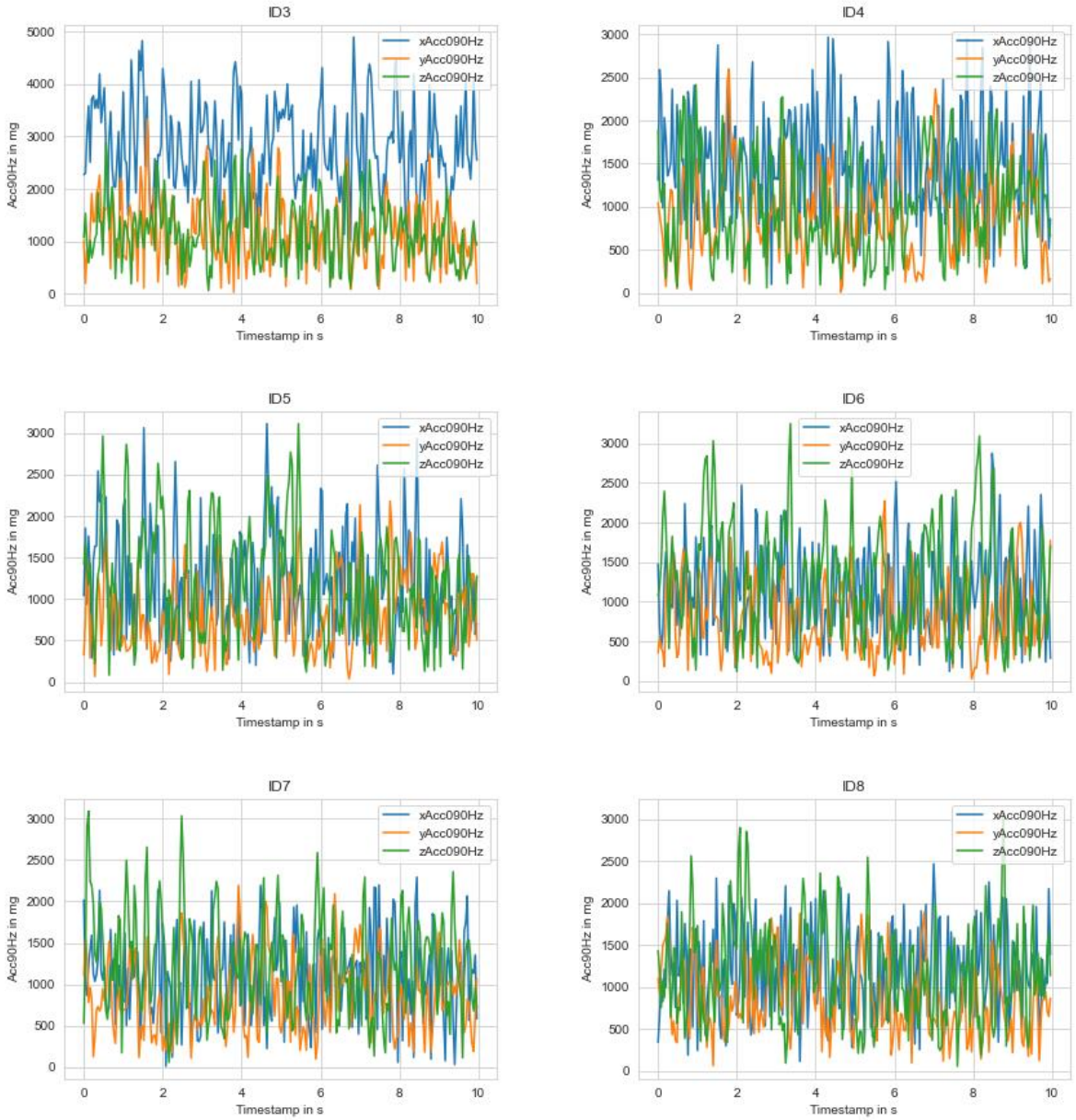
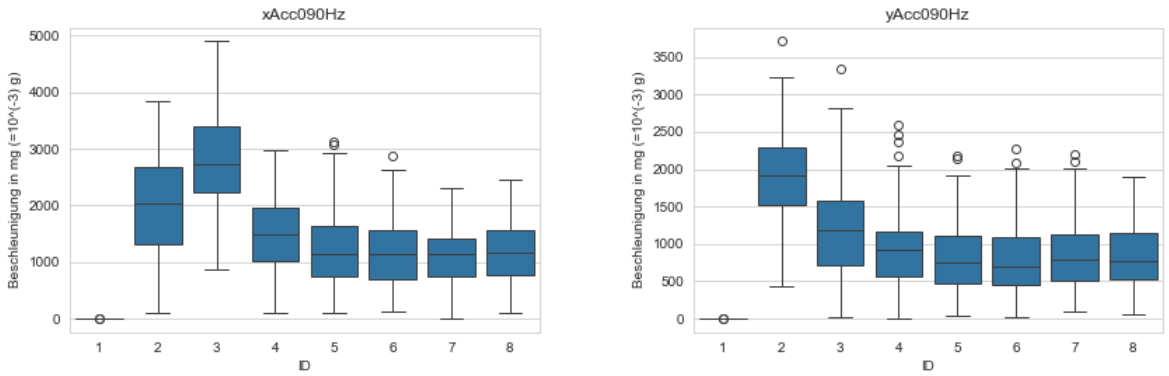


Abbildung 7.32 Liniendiagramme Acc90Hz



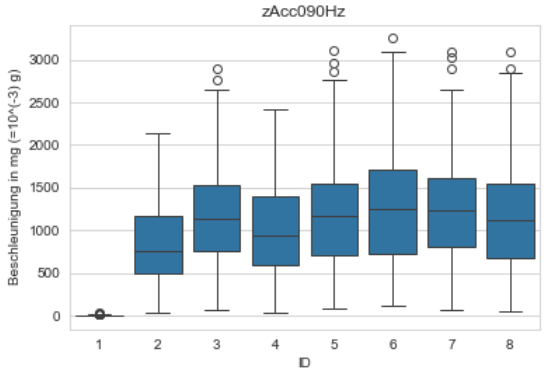
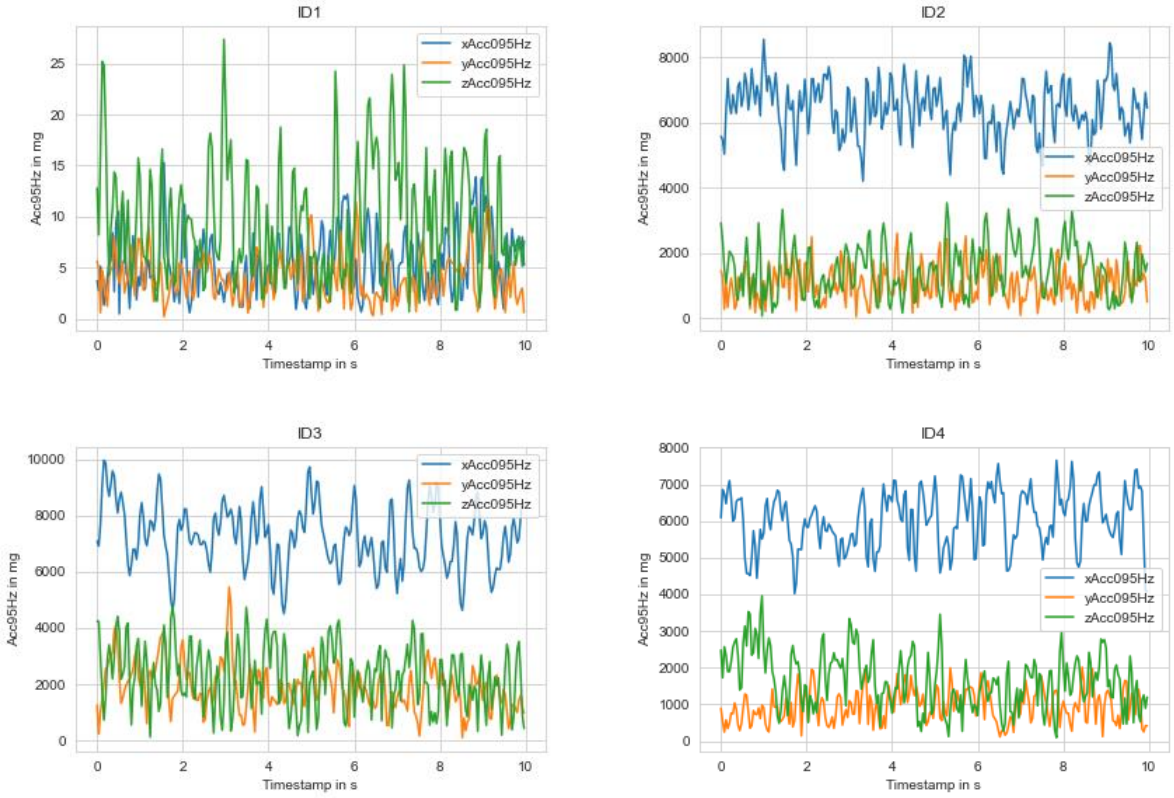


Abbildung 7.33 Boxplots Acc90Hz

A.18 Acc95Hz



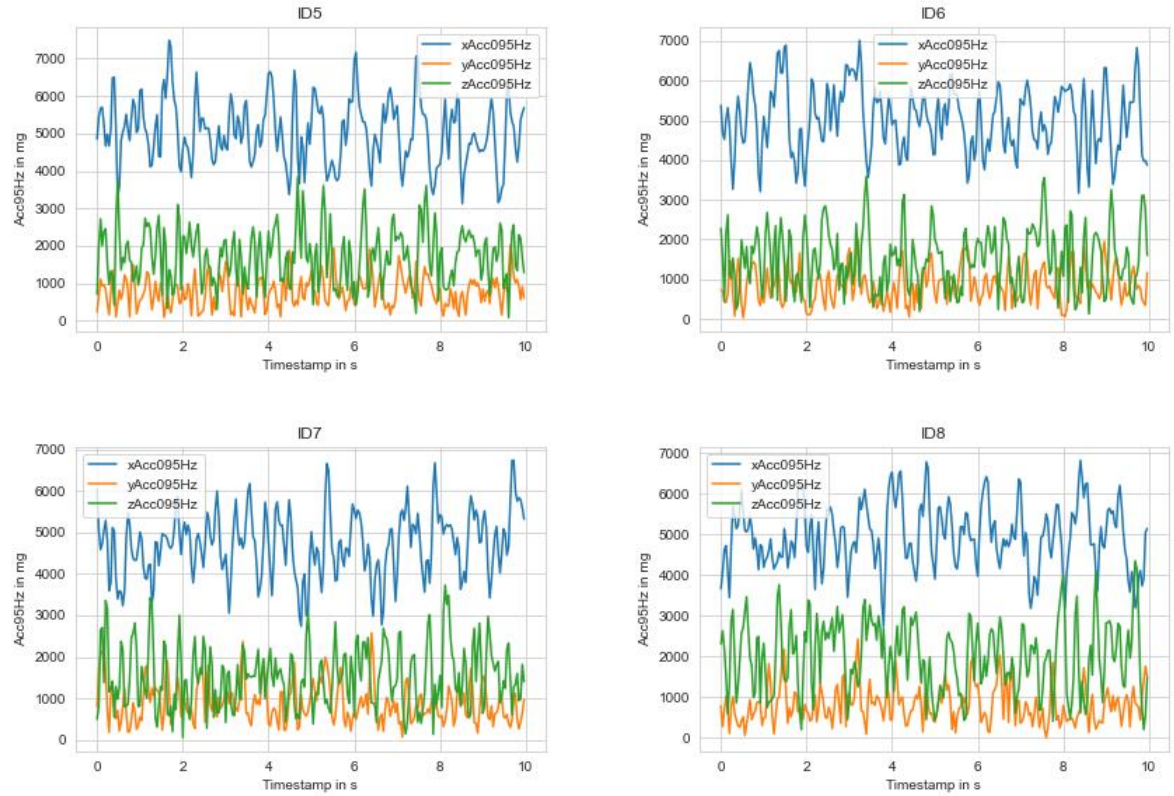


Abbildung 7.34 Liniendiagramme Acc95Hz

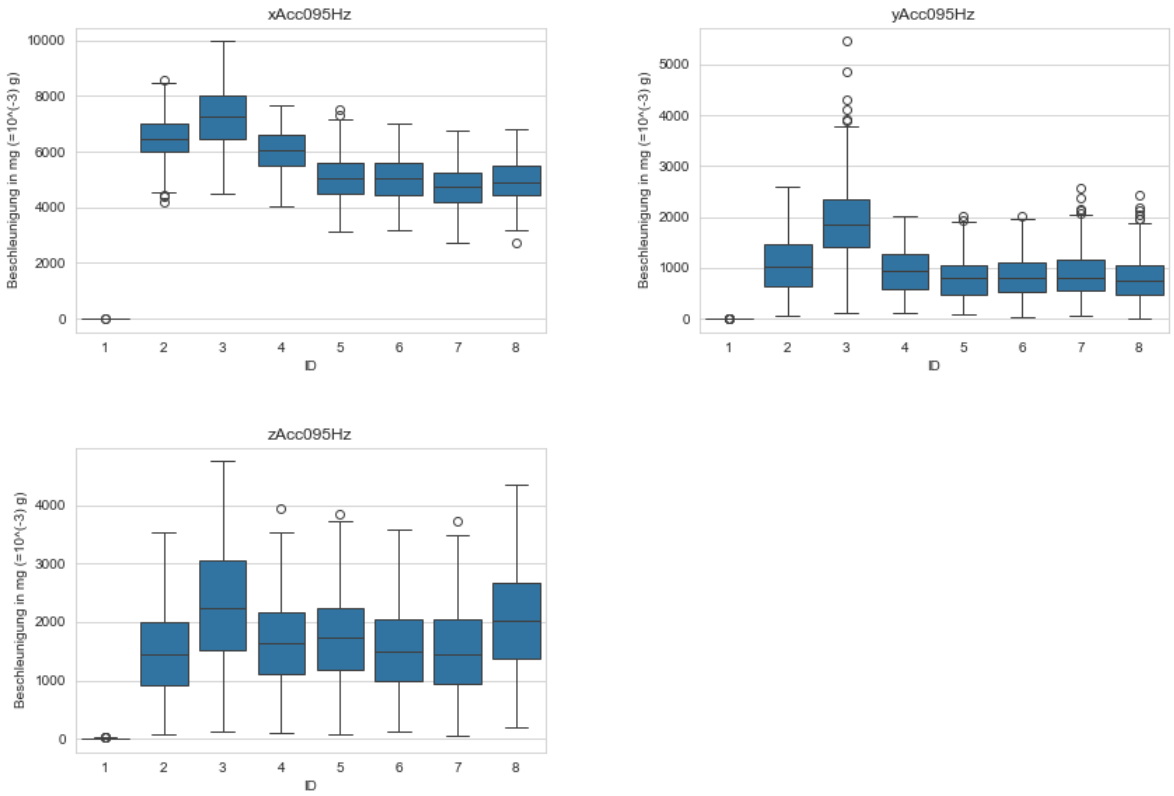


Abbildung 7.35 Boxplots Acc95Hz

A.19 Acc100Hz

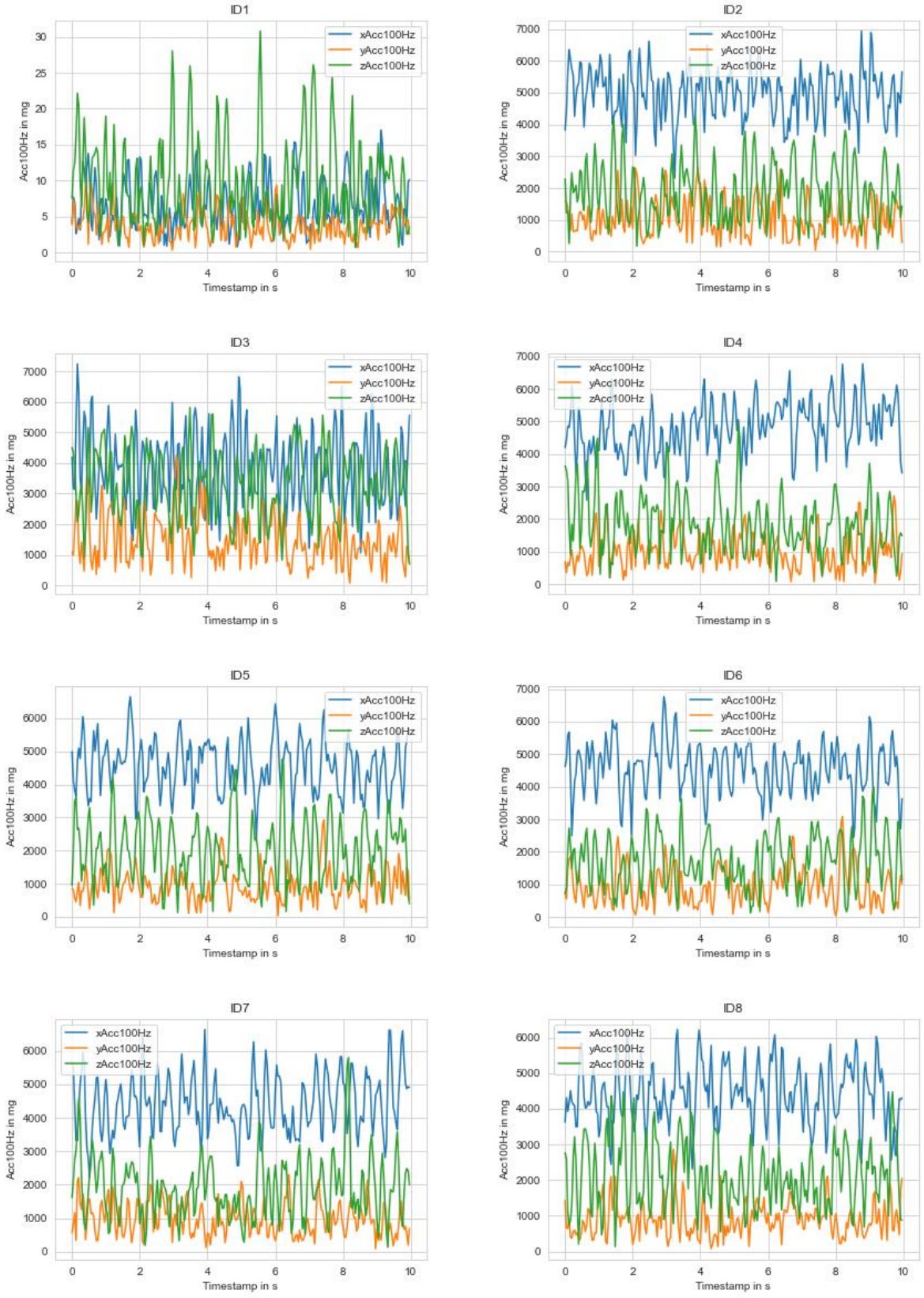


Abbildung 7.36 Liniendiagramme Acc100Hz

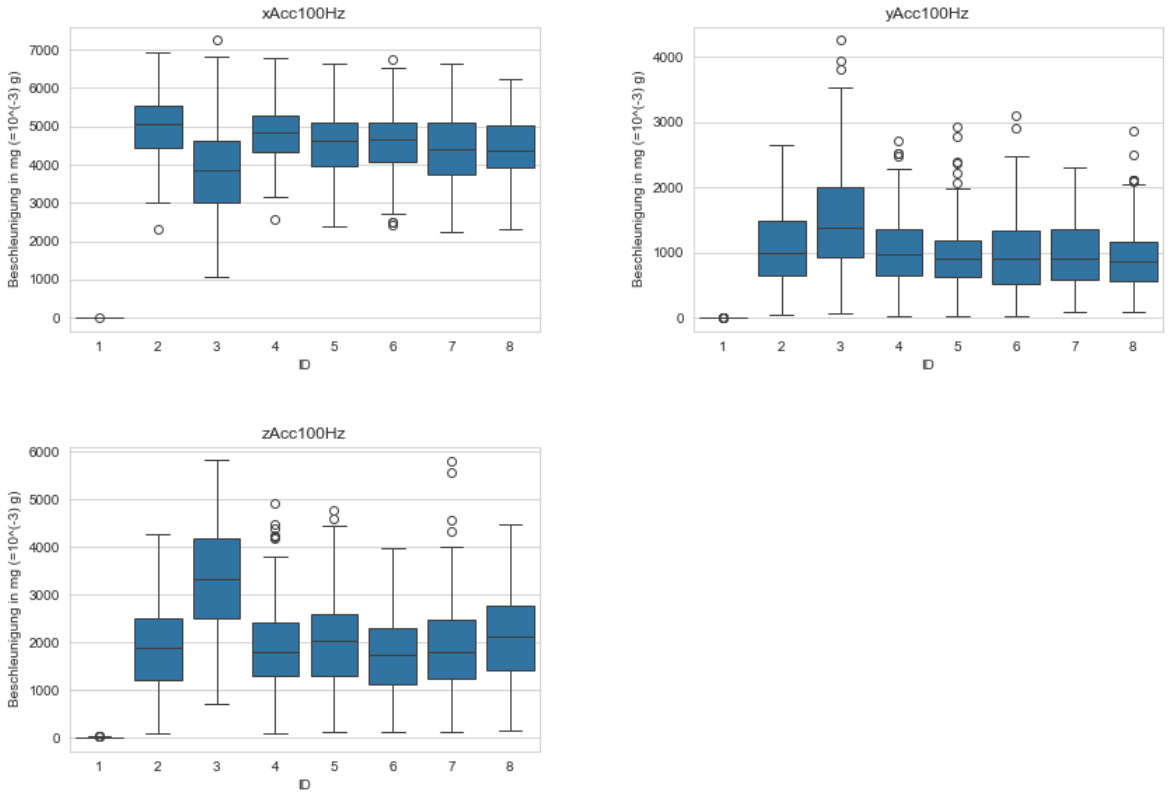
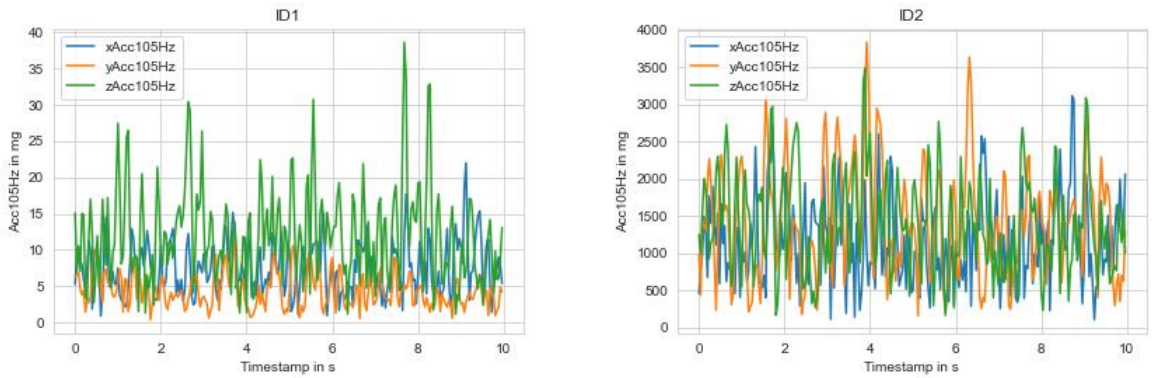


Abbildung 7.37 Boxplots Acc100Hz

A.20 Acc105Hz



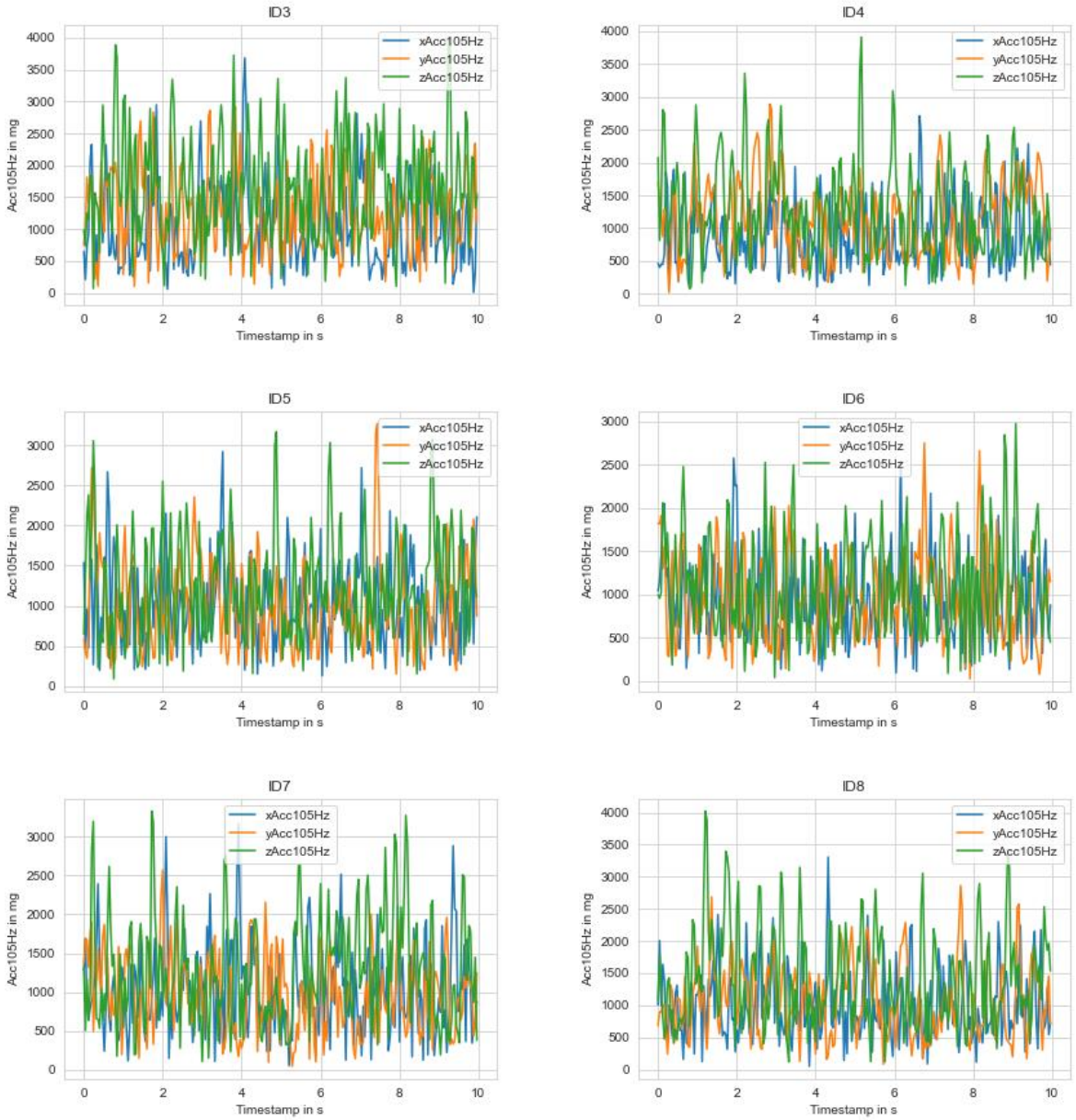
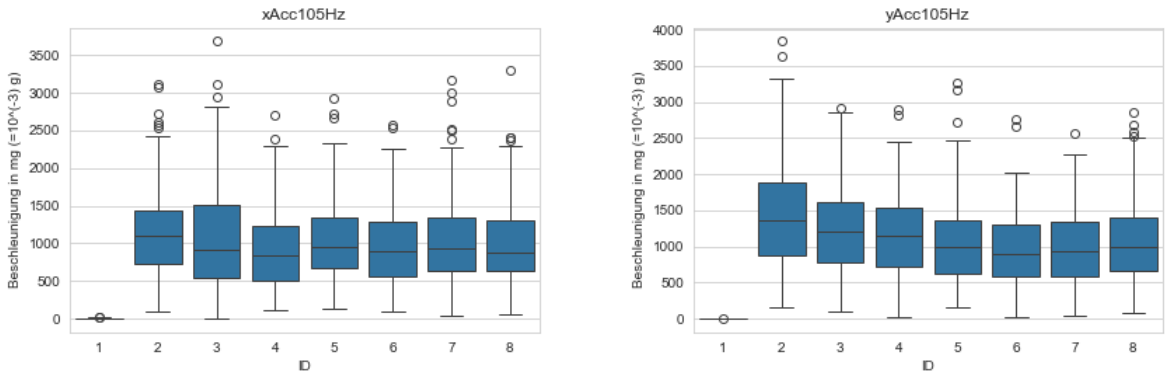


Abbildung 7.38 Liniendiagramme Acc105Hz



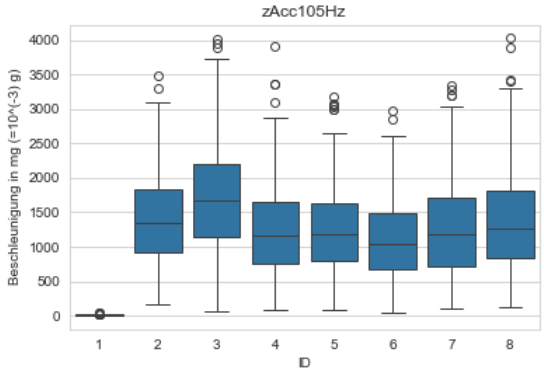
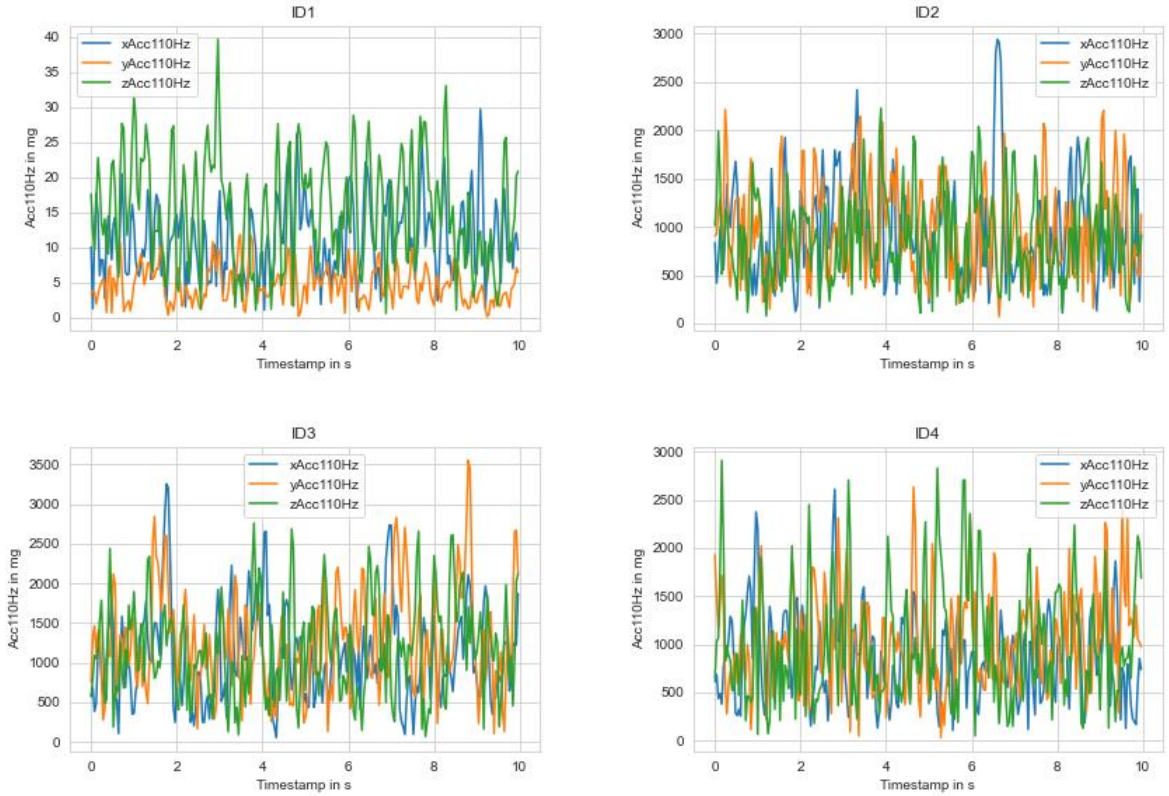


Abbildung 7.39 Boxplots Acc105 Hz

A.21 Acc110Hz



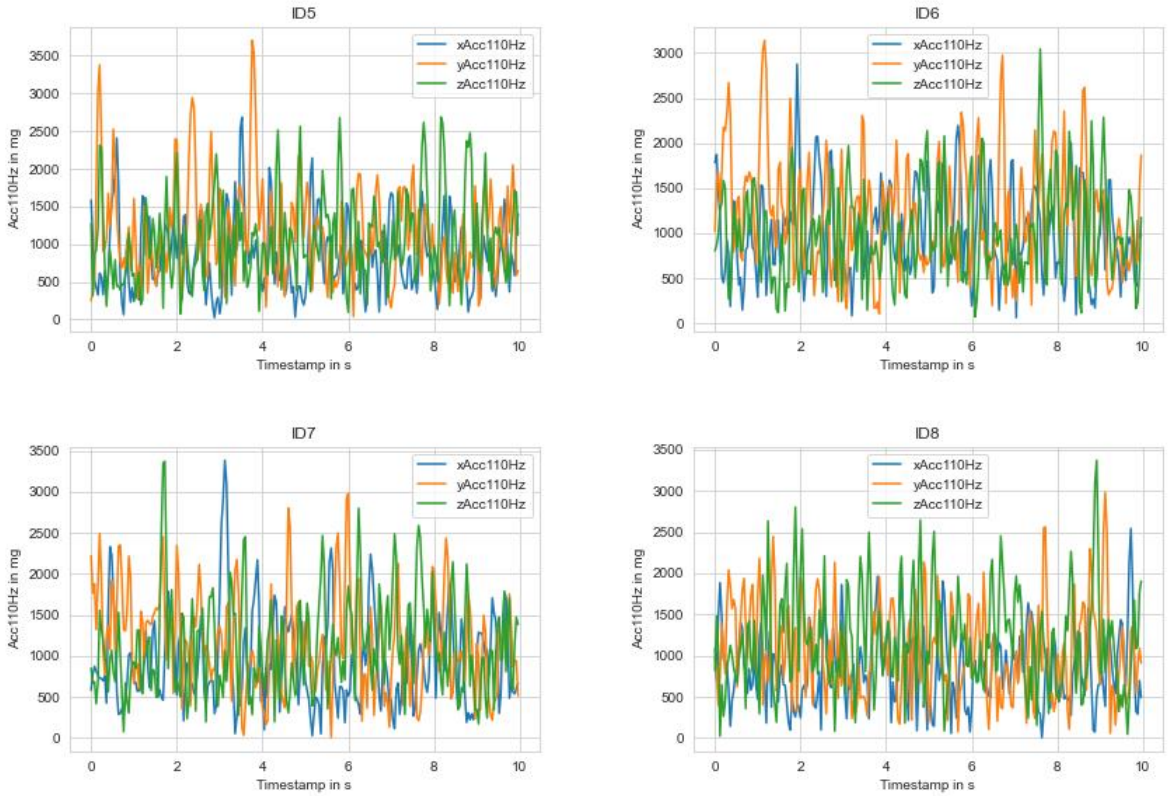


Abbildung 7.40 Liniendiagramme Acc110Hz

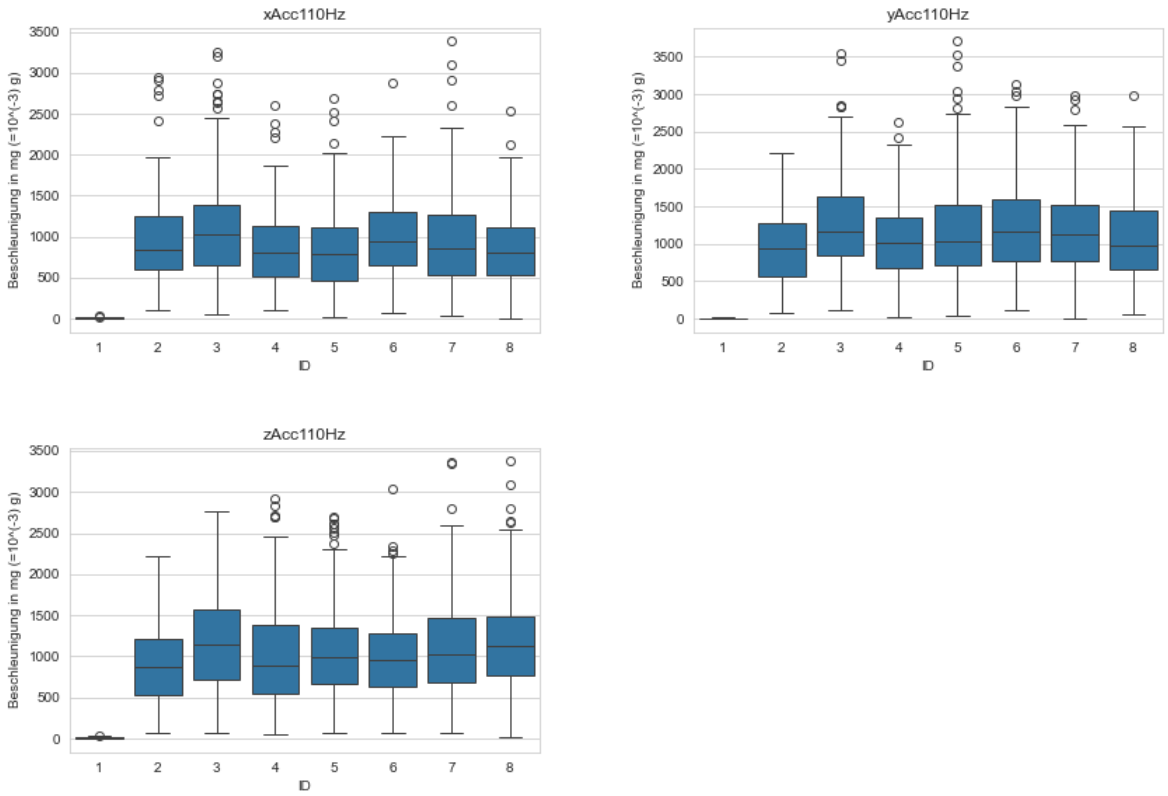


Abbildung 7.41 Boxplots Acc110Hz

A.22 Acc115Hz

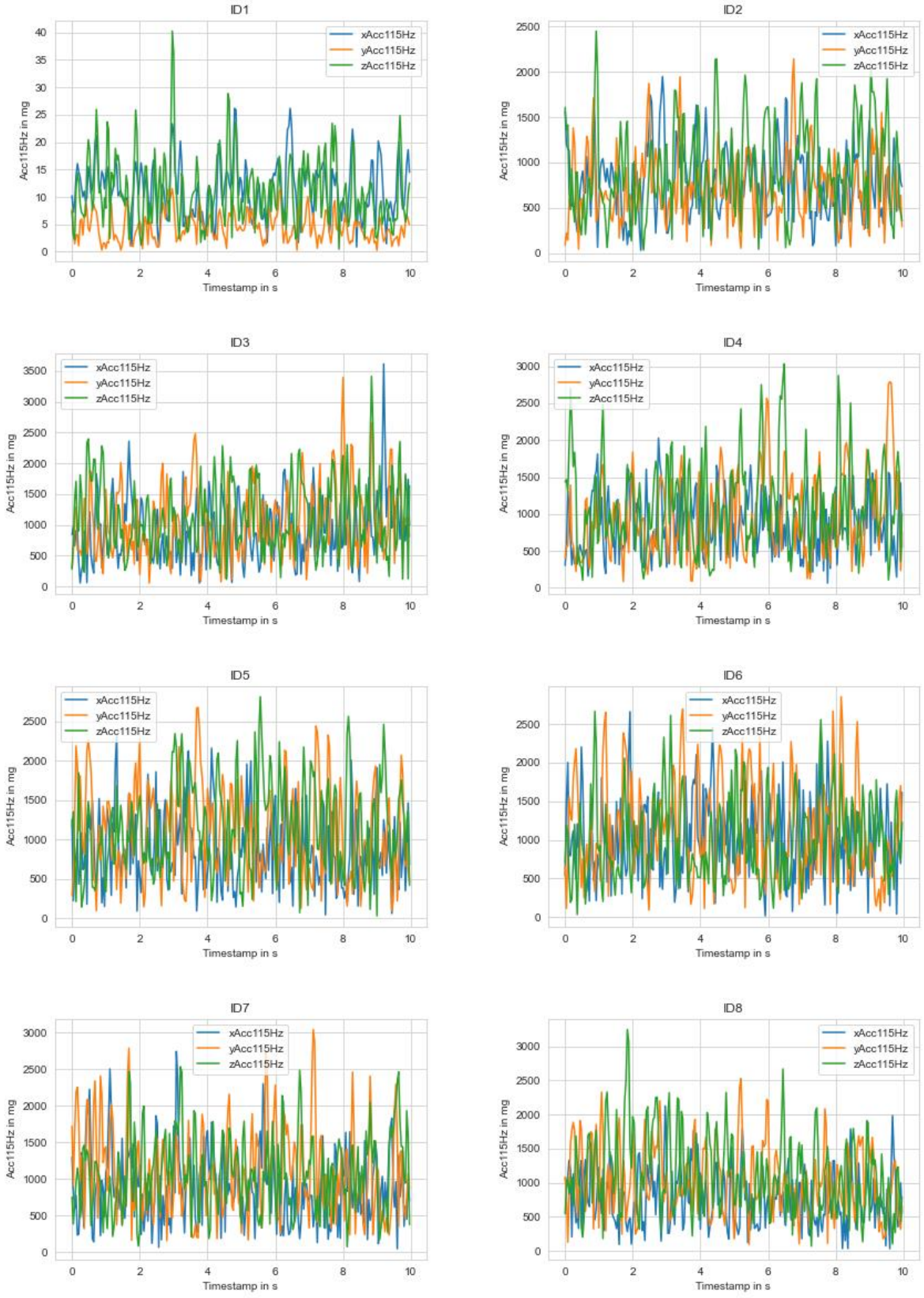


Abbildung 7.42 Liniendiagramme Acc115Hz

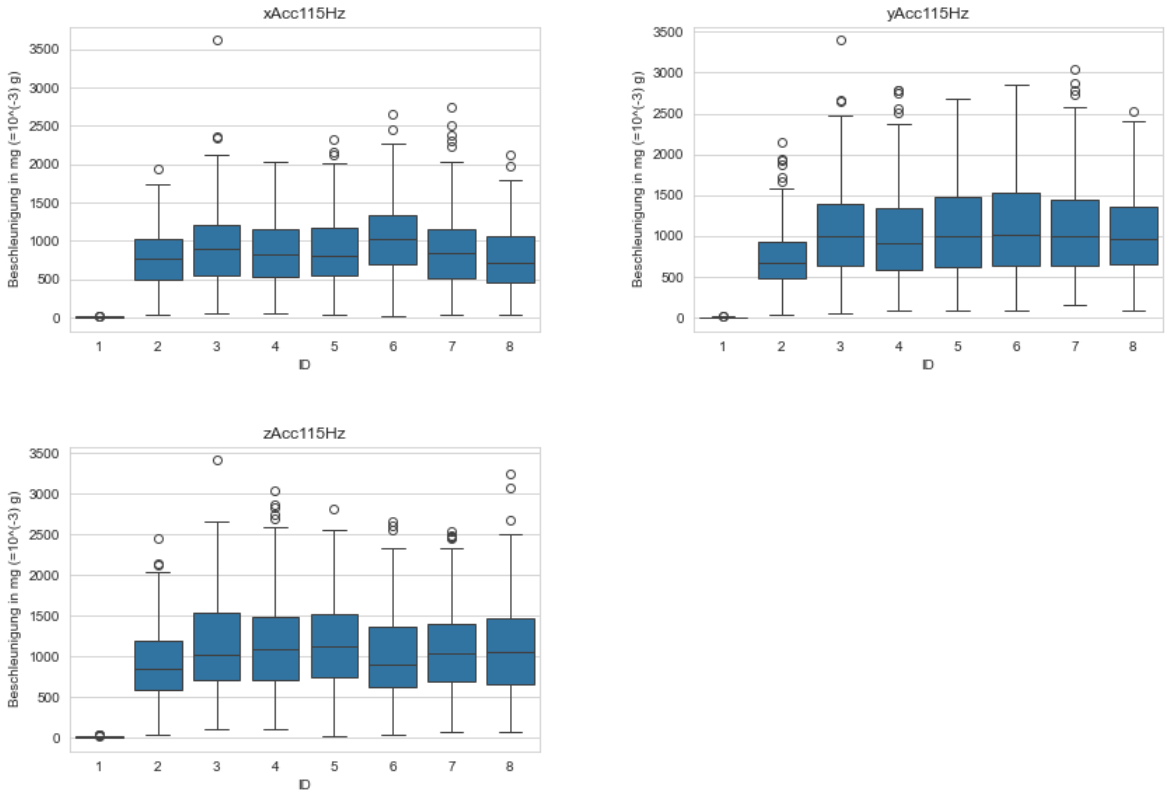
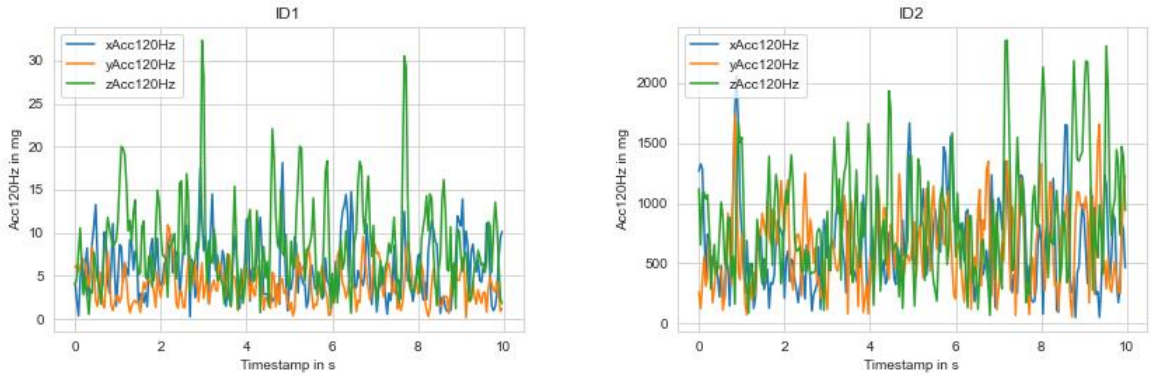


Abbildung 7.43 Boxplots Acc115Hz

A.23 Acc120 Hz



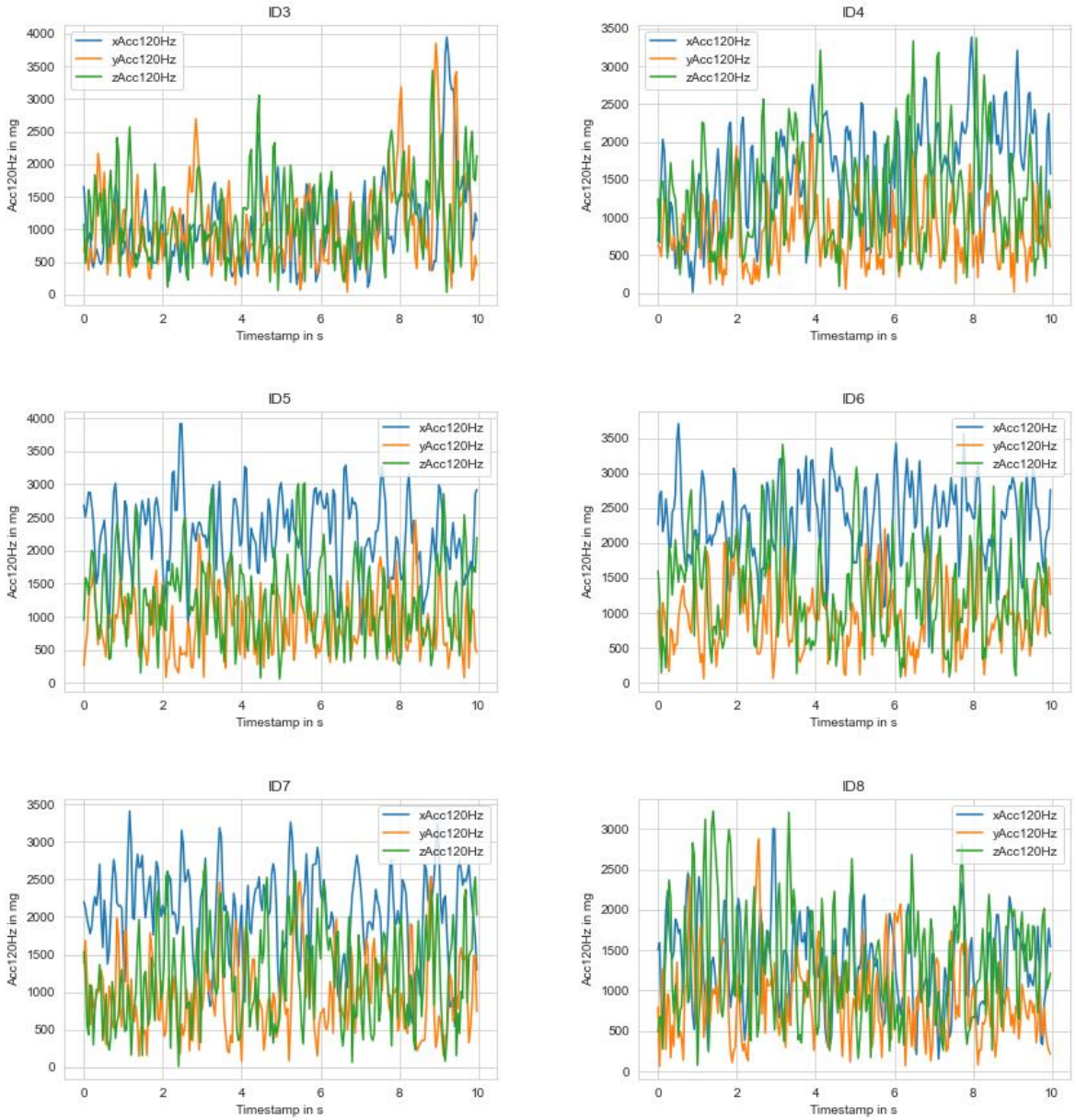
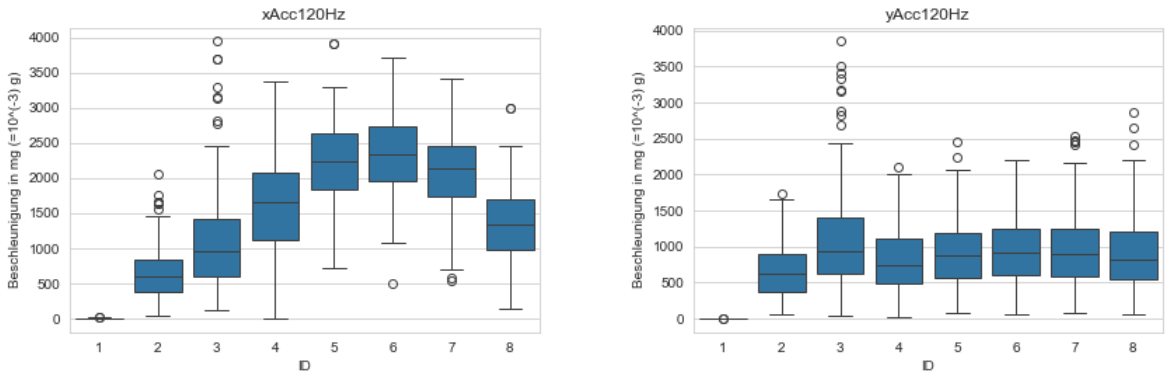


Abbildung 7.44 Liniendiagramme Acc120Hz



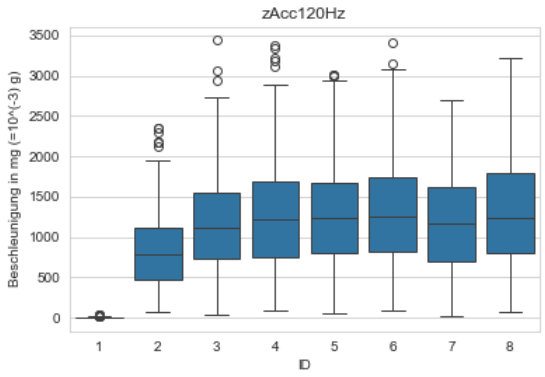


Abbildung 7.45 Boxplots Acc120Hz

A.24 Boxplots von jedem Betriebszustand

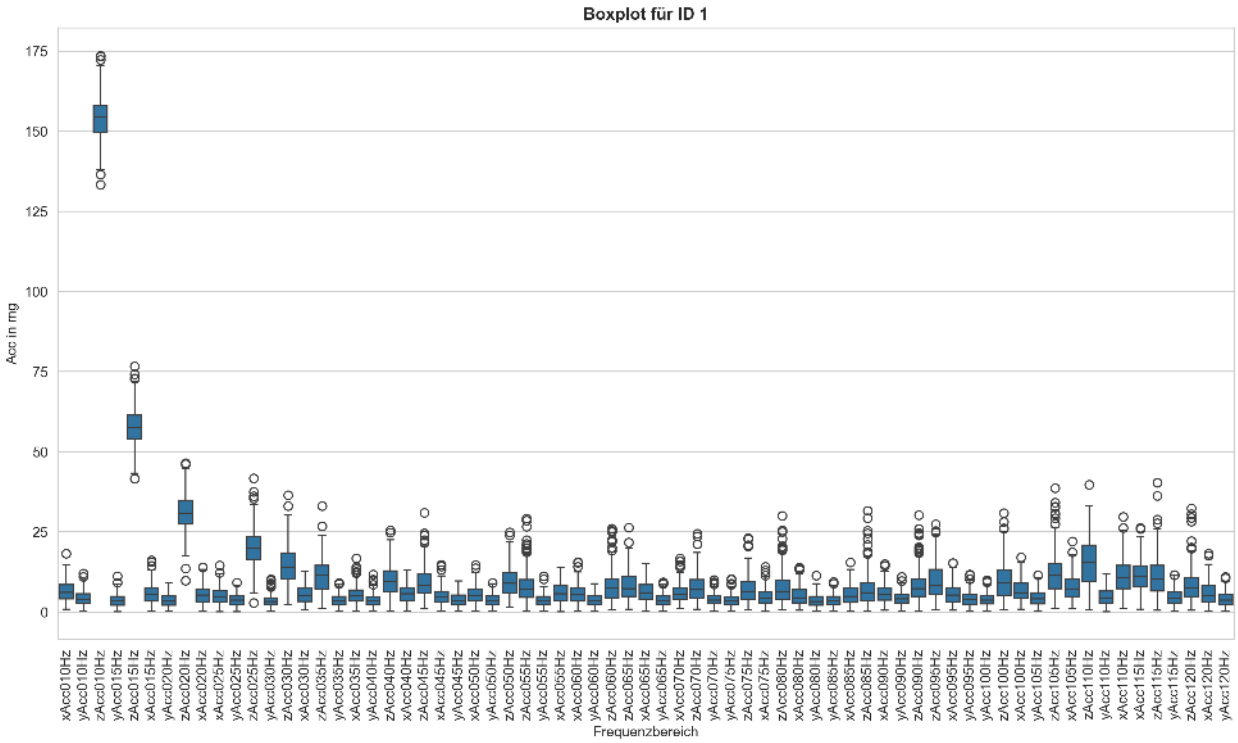


Abbildung 7.46 Boxplots für ID1

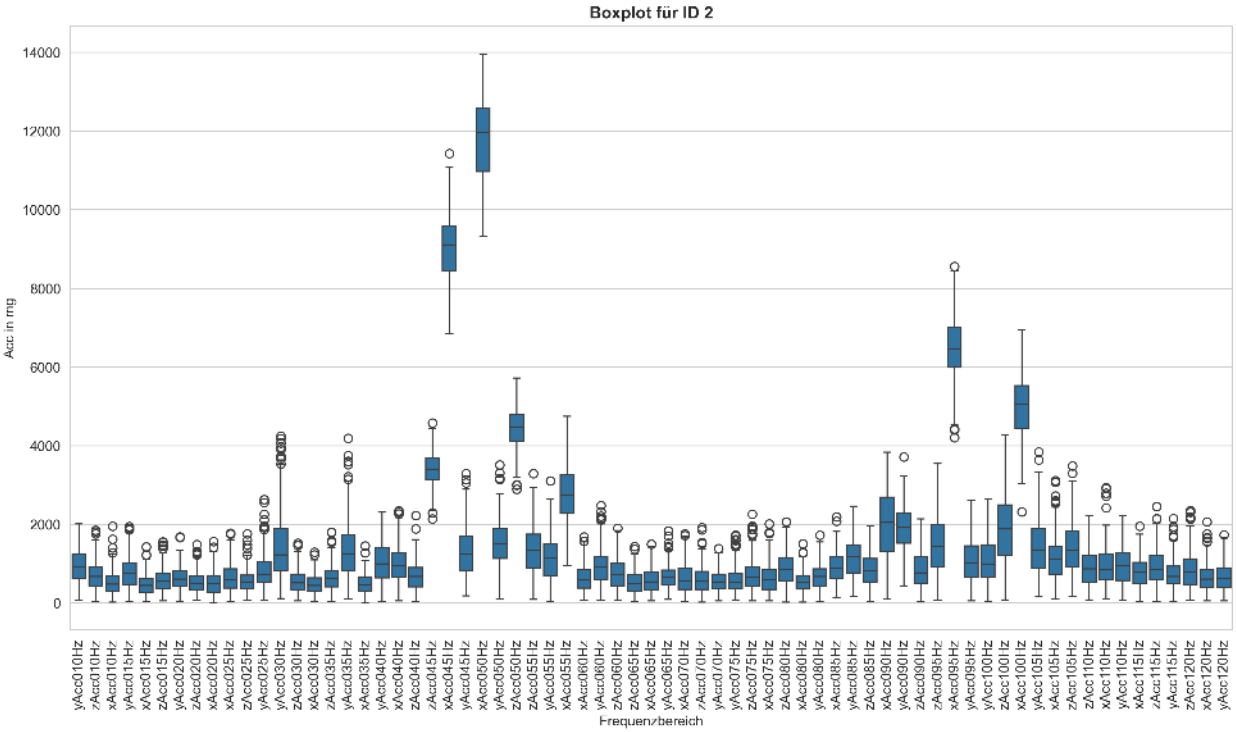


Abbildung 7.47 Boxplots für ID2

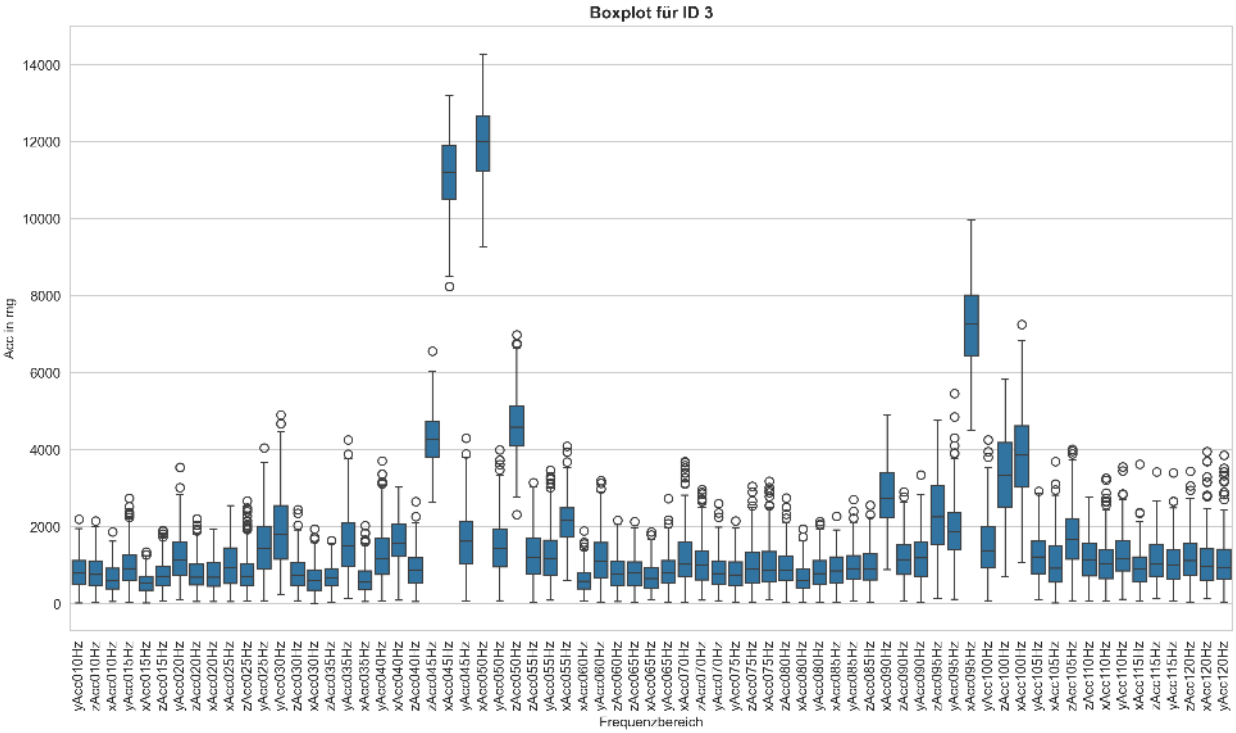


Abbildung 7.48 Boxplots für ID3

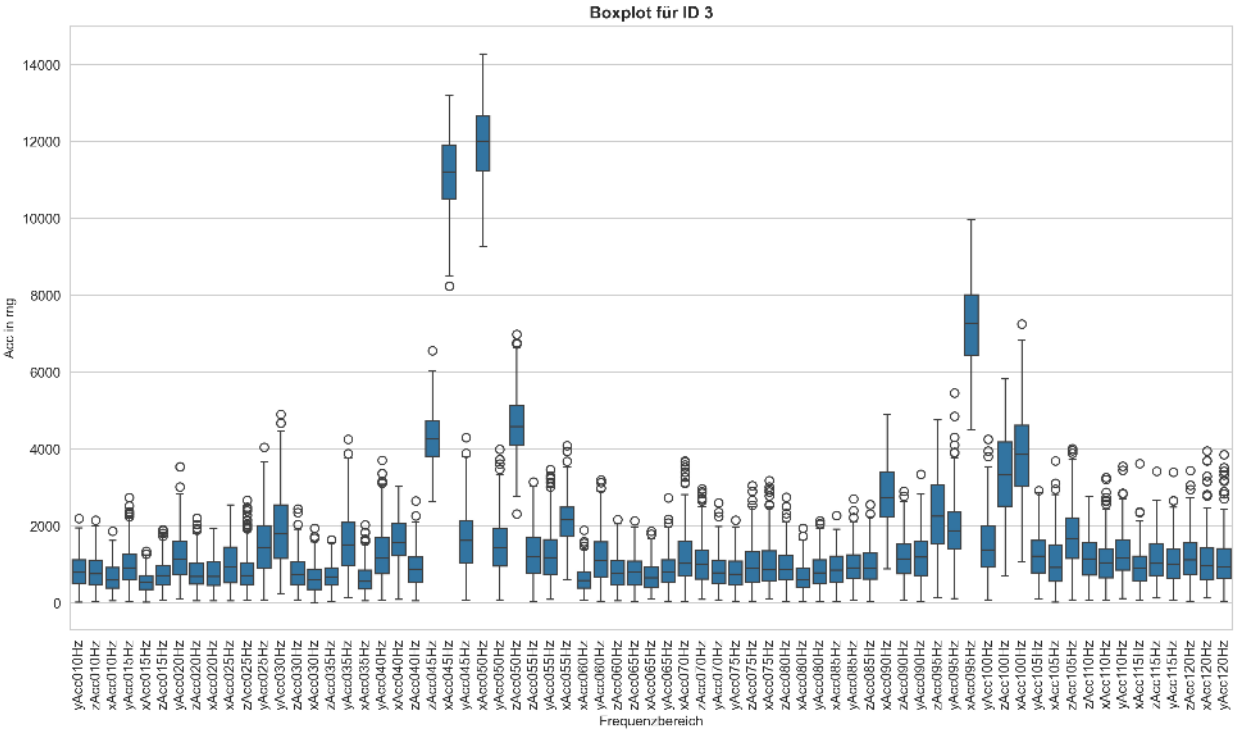


Abbildung 7.49 Boxplots für ID4

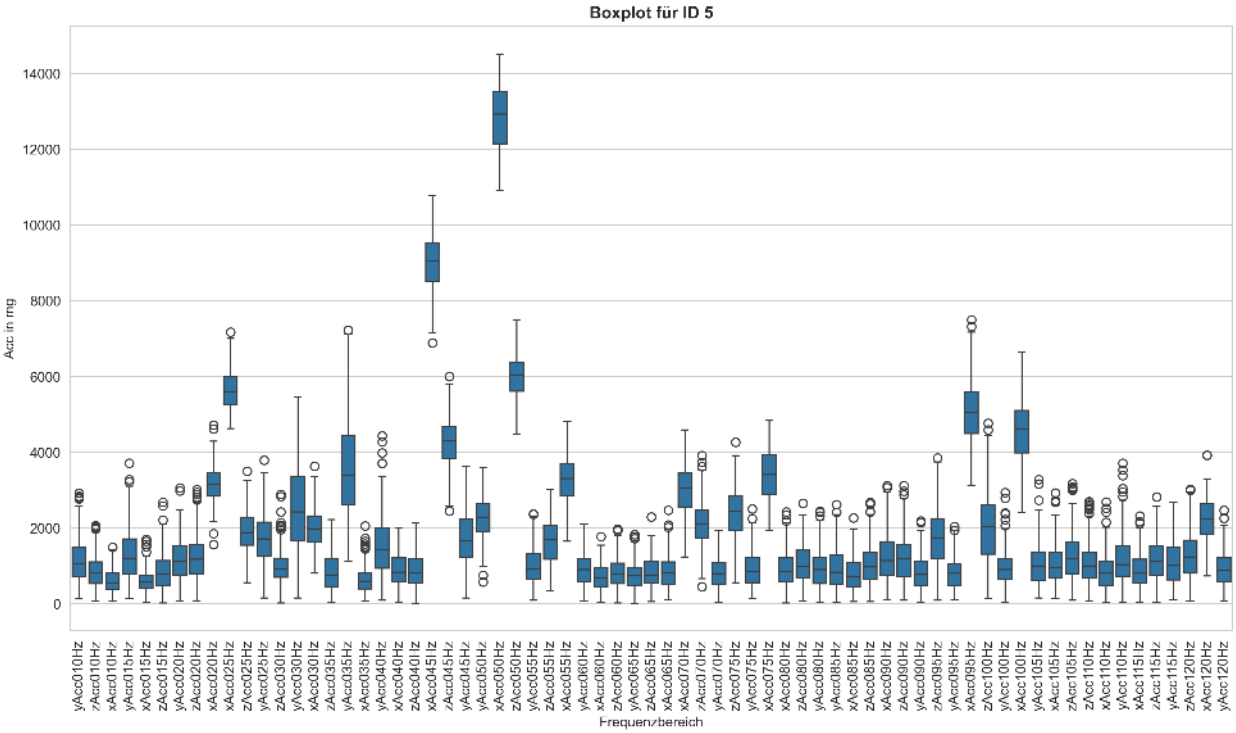


Abbildung 7.50 Boxplots für ID5

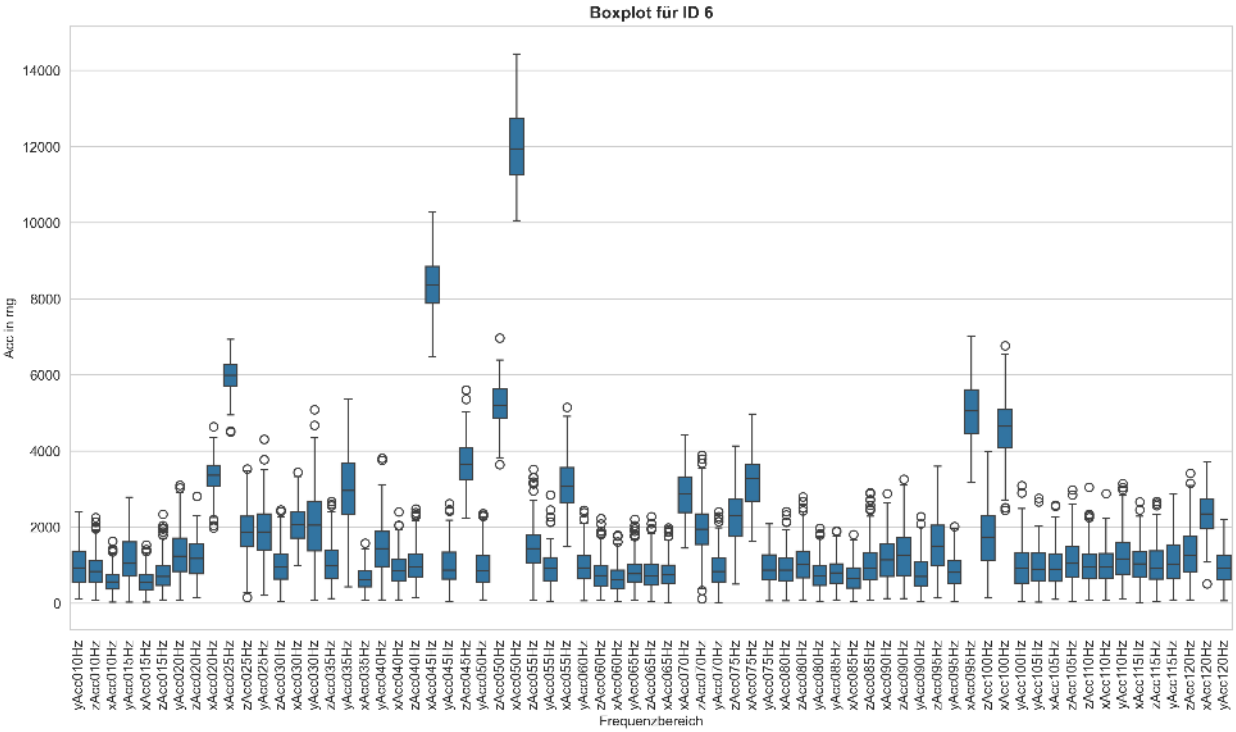


Abbildung 7.51 Boxplots für ID6

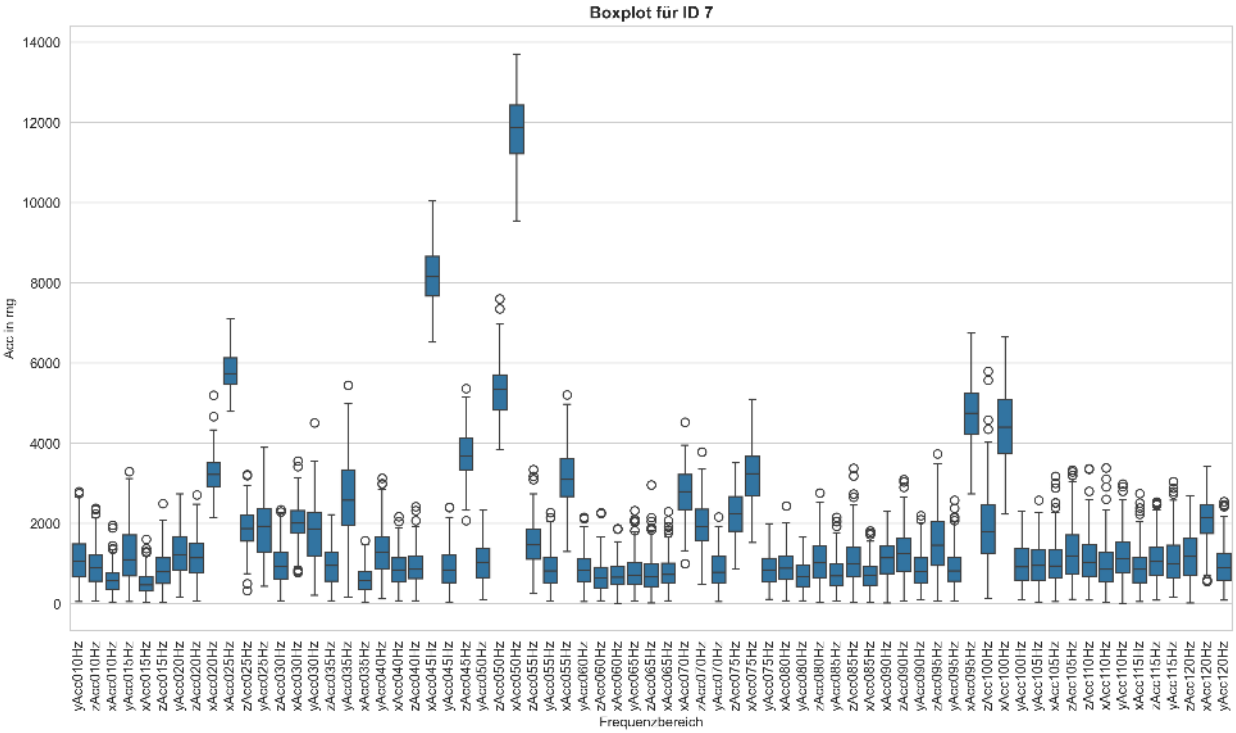


Abbildung 7.52 Boxplots für ID7

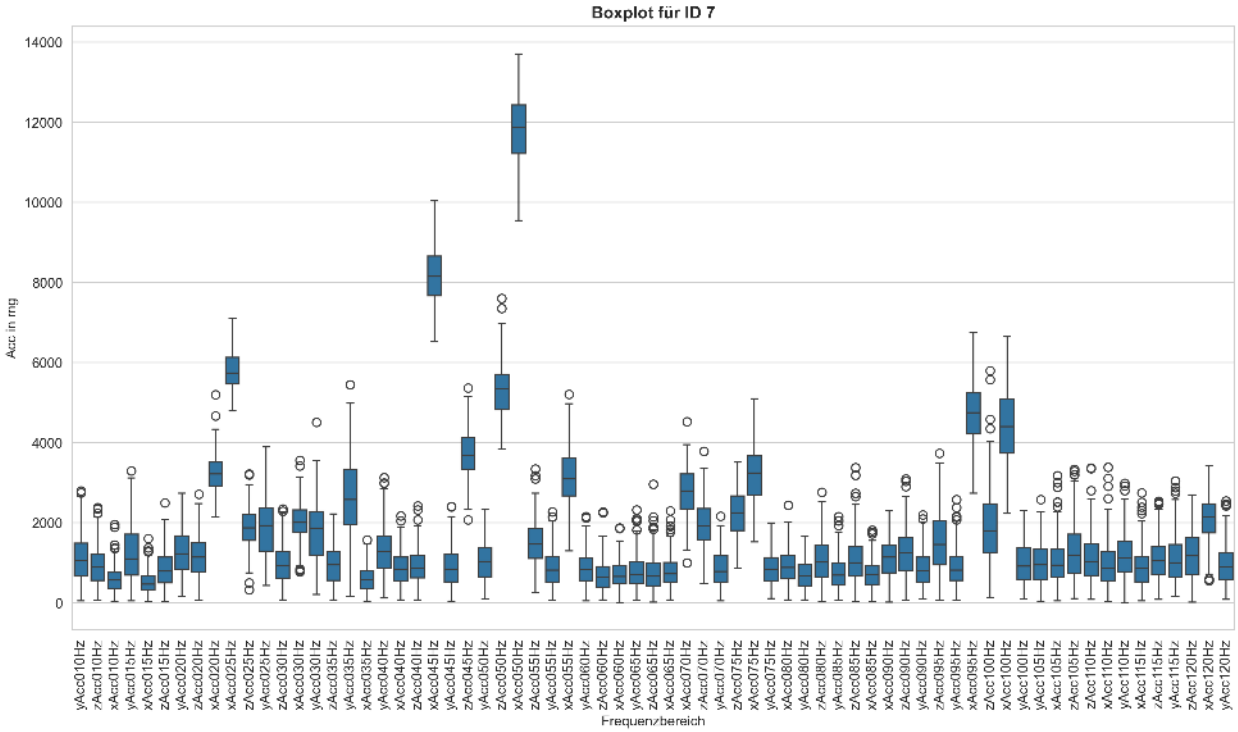


Abbildung 7.53 Boxplots für ID8

B. Python- Skript

B.1 Code CNN_1

```
#CNN_1

import pandas as pd
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, f1_score, recall_score, precision_score
from tensorflow.keras.metrics import MeanSquaredError

# Laden des Datensatzes
df = pd.read_csv("C:\\Users\Raphael
Willeke\\Documents\\Uni\Masterthesis\\Datensatz\\dropped_werte.csv")  # Pfad zur CSV-Datei

# Extrahieren der relevanten Spalten
classes = df['ID']  # Klassen in der ersten Spalte
time_series = df.iloc[:, 4:]  # Zeitreihen ab der vierten Spalte
print("Dataframe")
print(time_series)

# Umwandeln der Klassen in numerische Werte
label_encoder = LabelEncoder()
classes_encoded = label_encoder.fit_transform(classes)
classes_one_hot = to_categorical(classes_encoded)

# Umwandeln des DataFrame in ein numpy-Array und Reshape für CNN
time_series_array = np.array(time_series)
time_series_array = time_series_array.reshape((time_series_array.shape[0],
time_series_array.shape[1], 1))

# Aufteilen der Daten in Trainings- und Testdaten
```

```
X_train, X_test, y_train, y_test = train_test_split(time_series_array, classes_one_hot, test_size=0.2, random_state=42)

# Modell erstellen
model = Sequential()
# Erster Convolutional Layer
model.add(Conv1D(filters=64, kernel_size=12, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
# Zweiter Convolutional Layer
model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
# Dritter Convolutional layer
model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
# Flatten Layer
model.add(Flatten())
# Dense Layer
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
# Ausgabe Layer
model.add(Dense(8, activation='softmax'))

# Modell kompilieren
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy', MeanSquaredError()])

# Modell trainieren
history = model.fit(
    X_train, y_train,
    epochs=50, batch_size=32,
    validation_data=(X_test, y_test))

# Modell evaluieren
loss, accuracy, mse = model.evaluate(X_test, y_test)
print(f'Modellgenauigkeit: {accuracy * 100:.2f}%')
print(f'Mean Squared Error: {mse:.4f}')
```

```
# Vorhersagen für Testdaten
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

# Berechnung zusätzlicher Metriken
f1 = f1_score(y_true_classes, y_pred_classes, average='weighted')
recall = recall_score(y_true_classes, y_pred_classes, average='weighted')
precision = precision_score(y_true_classes, y_pred_classes, average='weighted')

print(f'F1 Score: {f1:.4f}')
print(f'Recall: {recall:.4f}')
print(f'Precision: {precision:.4f}')

model.summary()
```

B.2 Code CNN_2

```
# CNN_2

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import f1_score, mean_squared_error, accuracy_score, precision_score, recall_score, confusion_matrix
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, GlobalAveragePooling1D, Dense, Dropout, BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt
import seaborn as sns

# Load and preprocess data
```

```
def load_and_preprocess_data(filepath):
    df = pd.read_csv(filepath)

    # Drop irrelevant column
    df = df.drop(columns=['Unnamed: 0', 'Label'])

    # Handle missing values (if any)
    df.fillna(df.mean(), inplace=True)

    # Extract time series data and target labels
    classes = df['ID']  # Target labels
    time_series = df.iloc[:, 4:]  # Time-series data from 4th column onwards
    print("Dataframe TimeSeries")
    print(time_series)
    # Normalize time-series data
    scaler = StandardScaler()
    time_series_scaled = scaler.fit_transform(time_series)

    # Encode target classes
    label_encoder = LabelEncoder()
    classes_encoded = label_encoder.fit_transform(classes)
    classes_one_hot = to_categorical(classes_encoded)

    # Reshape for CNN (samples, timesteps, features)
    time_series_array = time_series_scaled.reshape((time_series_scaled.shape[0],
time_series_scaled.shape[1], 1))

    return time_series_array, classes_one_hot

# Split the data
def split_data(X, y, test_size=0.2, random_state=42):
    return train_test_split(X, y, test_size=test_size, random_state=random_state)

# Build CNN model
def build_model(input_shape, num_classes):
    model = Sequential()
```

```
    # Convolutional layers with Dropout and Batch Normalization
    model.add(Conv1D(filters=128, kernel_size=5, activation='relu', input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))
    model.add(MaxPooling1D(pool_size=2))

    model.add(Conv1D(filters=256, kernel_size=7, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.3))
    model.add(MaxPooling1D(pool_size=2))

    model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.3))
    model.add(MaxPooling1D(pool_size=2))

    model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.3))
    model.add(MaxPooling1D(pool_size=2))

    # Global Average Pooling and Dense layers
    model.add(GlobalAveragePooling1D())
    model.add(Dense(128, activation='relu', kernel_regularizer='l2')) # Dense 1
    model.add(Dropout(0.3))
    model.add(Dense(256, activation='relu', kernel_regularizer='l2')) # Dense 2
    model.add(Dropout(0.2))
    model.add(Dense(128, activation='relu', kernel_regularizer='l2')) # Dense 3
    model.add(Dropout(0.2))

    # Output layer
    model.add(Dense(num_classes, activation='softmax')) #Dense Output Layer

    return model

# Compile the model
def compile_model(model):
```

```
    model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Callbacks
def get_callbacks():
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.00001, verbose=1)
    early_stop = EarlyStopping(monitor='val_loss', patience=25, restore_best_weights=True, verbose=1)
    checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy', save_best_only=True, verbose=1)
    return [reduce_lr, early_stop, checkpoint]

# Train the model
def train_model(model, X_train, y_train, callbacks, validation_split=0.2, batch_size=32, epochs=100):
    history = model.fit(
        X_train,
        y_train,
        validation_split=validation_split,
        epochs=epochs,
        batch_size=batch_size,
        callbacks=callbacks,
        verbose=1
    )
    return history

# Evaluate the model
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    y_test_argmax = np.argmax(y_test, axis=1)
    y_pred_argmax = np.argmax(y_pred, axis=1)

    # Calculate metrics
    accuracy = accuracy_score(y_test_argmax, y_pred_argmax)
    f1 = f1_score(y_test_argmax, y_pred_argmax, average='weighted')
```

```
    mse = mean_squared_error(y_test_argmax, y_pred_argmax)
    precision = precision_score(y_test_argmax, y_pred_argmax, average='weighted')
    recall = recall_score(y_test_argmax, y_pred_argmax, average='weighted')

    # Print metrics
    print(f"Accuracy: {accuracy * 100:.2f}%")
    print(f"F1 Score: {f1:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"MSE: {mse:.4f}")

    # Compute confusion matrix
    cm = confusion_matrix(y_test_argmax, y_pred_argmax)

    # Plot confusion matrix
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title('Confusion Matrix CNN_2')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

    # Set x and y axis labels
    class_labels = [str(i) for i in range(1, 9)]  # Creates labels from 1 to 8
    plt.xticks(np.arange(8) + 0.5, class_labels)
    plt.yticks(np.arange(8) + 0.5, class_labels)

    plt.show()

    return accuracy, f1, precision, recall, mse

# Main function to execute the workflow
def main():
    # Load and preprocess data
    filepath = 'C:\\Users\Raphael Willeke\\Documents\\Uni\Masterthesis\\Datensatz\\dropped_werte.csv'
    X, y = load_and_preprocess_data(filepath)
```

```
    # Split data into training and test sets
    X_train, X_test, y_train, y_test = split_data(X, y)

    # Build and compile the model
    model = build_model(input_shape=(X_train.shape[1],  1), num_classes=y_train.shape[1])
    model = compile_model(model)

    # Get callbacks
    callbacks = get_callbacks()

    # Train the model
    train_model(model, X_train, y_train, callbacks)

    # Evaluate the model
    evaluate_model(model, X_test, y_test)
model.summary()

# Run the main function
if __name__ == "__main__":
    main()
```

B.3 Code CNN_3

```
# CNN_3

import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout, BatchNormalization,
GlobalAveragePooling1D
from keras.optimizers import Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.metrics import confusion_matrix, mean_squared_error, f1_score, precision_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns

# Laden des Datensatzes
df = pd.read_csv("C:\\Users\Raphael Willeke\\Documents\\Uni\Masterthesis\\Datensatz\\droppe d_werte.csv")  # Dateipfad zur CSV-Datei

print("kompletter Datensatz")
print(df)

# Extrahieren der relevanten Spalten
classes = df['ID']  # Klassen in der ersten Spalte
time_series = df.iloc[:, 4:]  # Zeitreihen ab der vierten Spalte
print("Dataframe time Series")
print(time_series)



# Umwandeln der Klassen in numerische Werte
label_encoder = LabelEncoder()
classes_encoded = label_encoder.fit_transform(classes)
classes_one_hot = to_categorical(classes_encoded)

# Umwandeln des DataFrame in ein numpy-Array und Reshape für CNN
time_series_array = np.array(time_series)
time_series_array = time_series_array.reshape((time_series_array.shape[0], time_series_array.shape[1], 1))

# Aufteilen der Daten in Trainings- und Testdaten
X_train, X_test, y_train, y_test = train_test_split(time_series_array, classes_one_hot, test_size=0.2, random_state=42)

# Export test data to CSV
def export_test_data(X_test, y_test, label_encoder, filepath):
    # Reshape X_test back to 2D
    X_test_2d = X_test.reshape(X_test.shape[0], -1)
```

```
    # Convert one-hot encoded y_test back to original labels
    y_test_labels = label_encoder.inverse_transform(np.argmax(y_test, axis=1))

    # Create a DataFrame
    test_df = pd.DataFrame(X_test_2d, columns=[f'feature_{i}' for i in range(X_test_2d.shape[1])])
    test_df['label'] = y_test_labels

    # Export to CSV
    test_df.to_csv(filepath, index=False)
    print(f"Test data exported to {filepath}")

# Call the function to export test data
export_test_data(X_test, y_test, label_encoder, "C:\\Users\\Raphael Willeke\\Documents\\Uni\\Masterthesis\\Datensatz\\test_data_CNN3.csv")

# Model Definition
model = Sequential()

# First Convolutional Layer
model.add(Conv1D(filters=128, kernel_size=5, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2))

# Second Convolutional Layer
model.add(Conv1D(filters=256, kernel_size=7, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2))

# Third Convolutional Layer
model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
model.add(BatchNormalization())  # Added Batch Normalization
model.add(MaxPooling1D(pool_size=2))

# Fourth Layer to Improve Learning
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
```

```
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2))

# Global Average Pooling instead of Flatten
model.add(GlobalAveragePooling1D())

# Dense Layers
model.add(Dense(128, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.3))
model.add(Dense(256, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.2))

# Output Layer
model.add(Dense(8, activation='softmax'))

# Compile the Model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])

# Callbacks
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.00001,
verbose=1)
early_stop = EarlyStopping(monitor='val_loss', patience=25, restore_best_weights=True,
verbose=1)
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy', save_best_only=True,
verbose=1)

# Modell trainieren
history = model.fit(
    X_train,
    y_train,
    validation_split=0.2,
    epochs=50,
    batch_size=32,
    callbacks=[reduce_lr, early_stop, checkpoint],validation_data=(X_test, y_test)
```

```
)

# Modell evaluieren
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Modellgenauigkeit: {accuracy * 100:.2f}%')

model.summary()

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy CNN_3')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss CNN_3')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.savefig('training_history_CNN_3.png')
plt.close()

# Model Evaluation
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Confusion Matrix
cm = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
```

```
plt.title('Confusion Matrix CNN_3')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
num_classes = 8
tick_labels = list(range(1, num_classes + 1))
plt.xticks(np.arange(num_classes) + 0.5, tick_labels)
plt.yticks(np.arange(num_classes) + 0.5, tick_labels)
plt.savefig('confusion_matrix_CNN_3.png')
plt.close()

# Calculate additional metrics
mse = mean_squared_error(y_test, y_pred)
f1 = f1_score(y_true, y_pred_classes, average='weighted')
precision = precision_score(y_true, y_pred_classes, average='weighted')
recall = recall_score(y_true, y_pred_classes, average='weighted')

# Print all metrics
print(f'Model Accuracy: {accuracy * 100:.2f}%')
print(f'Mean Squared Error: {mse:.4f}')
print(f'F1 Score: {f1:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
```

B.3 Code CNN_4

```
#CNN_4 keras tuner

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, mean_squared_error, accuracy_score, precision_score,
recall_score, classification_report, confusion_matrix
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GlobalAveragePooling1D, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
from keras_tuner import RandomSearch
import matplotlib.pyplot as plt
import seaborn as sns


def load_and_preprocess_data(filepath):
    df = pd.read_csv(filepath)
    df = df.drop(columns=['Unnamed: 0', 'Label'])
    df.fillna(df.mean(), inplace=True)
    print("Complete DataFrame:")
    print(df.head())
    classes = df['ID']
    time_series = df.iloc[:, 2:]  # Timestamp column is not considered
    print("Time series DataFrame:")
    print(time_series.head())

    scaler = StandardScaler()
    time_series_scaled = scaler.fit_transform(time_series)

    label_encoder = LabelEncoder()
    classes_encoded = label_encoder.fit_transform(classes)
    classes_one_hot = to_categorical(classes_encoded)

    time_series_array = time_series_scaled.reshape((time_series_scaled.shape[0], time_series_scaled.shape[1], 1))

    return time_series_array, classes_one_hot, label_encoder

def split_data(X, y, test_size=0.2, random_state=42):
    return train_test_split(X, y, test_size=test_size, random_state=random_state, stratify=np.argmax(y, axis=1))

def save_test_data(X_test, y_test, filename='dropped_werte_test_data_CNN_4_kt.csv'):
```

```
    test_df = pd.DataFrame(X_test.reshape(X_test.shape[0], -1))
    test_df['label'] = np.argmax(y_test, axis=1)
    test_df.to_csv(filename, index=False)
    print(f"Test data saved to {filename}")

def build_model(hp, input_shape, num_classes):
    model = Sequential()

    # Initial convolutional layer
    model.add(Conv1D(filters=hp.Int('filters_1', 32, 256, step=32),
                     kernel_size=hp.Int('kernel_size_1', 3, 11, step=2),
                     activation='relu', input_shape=input_shape,
                     padding='same'))                    # Use 'same' padding to maintain input size
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=2))

    # Dynamically add convolutional layers based on input size
    current_size = input_shape[0] // 2                   # Size after first MaxPooling
    for i in range(hp.Int('n_conv_layers', 1, 4)):
        if current_size > 1:
            model.add(Conv1D(filters=hp.Int(f'filters_{i+1}', 32, 256, step=32),
                             kernel_size=hp.Int(f'kernel_size_{i+1}', 3, 11, step=2),
                             activation='relu',
                             padding='same'))  # Use 'same' padding
            model.add(BatchNormalization())
            model.add(MaxPooling1D(pool_size=2))
            current_size //= 2
        else:
            break                                        # Stop adding layers if size becomes too
small

    model.add(GlobalAveragePooling1D())

    # Dense layers
    for i in range(hp.Int('n_dense_layers', 1, 3)):
        model.add(Dense(units=hp.Int(f'dense_units_{i+1}', 64, 512, step=64), activation='relu'))
        model.add(Dropout(hp.Float(f'dropout_{i+1}', 0.1, 0.5, step=0.1)))
```

```
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', 1e-4, 1e-2,
sampling='LOG')),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

def run_hyperparameter_tuning(X_train, y_train, num_classes):
    input_shape = (X_train.shape[1], 1)
    tuner = RandomSearch(
        lambda hp: build_model(hp, input_shape, num_classes),
        objective='val_accuracy',
        max_trials=30,                    # Reduced from 100 to speed up the process
        executions_per_trial=2,
        directory='tuner_dir',
        project_name='cnn_tuning'
    )

    early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

    tuner.search(X_train, y_train, epochs=50, validation_split=0.2, callbacks=[early_stop])

    best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
    print("Best Hyperparameters:", best_hps.values)

    return tuner.get_best_models(num_models=1)[0]

def evaluate_model(model, X_test, y_test, label_encoder):
    y_pred = model.predict(X_test)
    y_test_argmax = np.argmax(y_test, axis=1)
    y_pred_argmax = np.argmax(y_pred, axis=1)

    metrics = {
        'Accuracy': accuracy_score(y_test_argmax, y_pred_argmax),
        'F1 Score': f1_score(y_test_argmax, y_pred_argmax, average='weighted'),
```

```
        'Precision': precision_score(y_test_argmax, y_pred_argmax, average='weighted'),
        'Recall': recall_score(y_test_argmax, y_pred_argmax, average='weighted'),
        'Mean Squared Error': mean_squared_error(y_test, y_pred)
    }

    for metric, value in metrics.items():
        print(f"{metric}: {value:.4f}")

    target_names = [f'Class {i+1}' for i in range(len(label_encoder.classes_))]

    print("\nClassification Report:")
    print(classification_report(y_test_argmax, y_pred_argmax, target_names=target_names))

    cm = confusion_matrix(y_test_argmax, y_pred_argmax)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=target_names,
yticklabels=target_names)
    plt.title('Confusion Matrix CNN Keras Tuner 50 Epochs')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.savefig('confusion_matrix_kt 50 Epochs.png')
    plt.close()

    return y_pred_argmax

def plot_training_history(history):
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy 50 Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
```

```
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss 50 Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.savefig('training_history_kt 50 Epochs.png')
    plt.close()

def main():
    filepath = 'dropped_werte.csv'
    print("CNN Keras Tuner")
    X, y, label_encoder = load_and_preprocess_data(filepath)

    X_train, X_test, y_train, y_test = split_data(X, y)
    save_test_data(X_test, y_test)

    best_model = run_hyperparameter_tuning(X_train, y_train, y.shape[1])

    best_model.summary()

    callbacks = [
        EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True),
        ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-6),
        ModelCheckpoint('best_model_kt.h5', save_best_only=True, monitor='val_loss',
mode='min')
    ]

    history = best_model.fit(X_train, y_train, validation_split=0.2, epochs=100,
                             callbacks=callbacks, batch_size=32)

    plot_training_history(history)

    best_model.load_weights('best_model_kt.h5')
    best_model.summary()
```

```
    predicted_classes = evaluate_model(best_model, X_test, y_test, label_encoder)

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

B.4 Code RNN_1

```
#RNN 1 LSTM

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, SimpleRNN
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import mean_squared_error, f1_score, recall_score, precision_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Daten laden und vorbereiten
df = pd.read_csv("C:\\Users\Raphael Willeke\\Documents\\Uni\Masterthesis\\Datensatz\\dropped_werte.csv")  # Dateipfad

# Nicht benötigte Spalten entfernen
df = df.drop(['Unnamed: 0', 'Label'], axis=1)

# Features und Zielwerte extrahieren
X = df.drop(['ID', 'Timestamp'], axis=1)
y = df['ID']

# Label encoding für die Zielwerte
le = LabelEncoder()
y = le.fit_transform(y)
```

```
# Daten in Trainings- und Testsets aufteilen
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Daten normalisieren
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Daten für LSTM umformen (samples, time steps, features)
X_train_reshaped = X_train_scaled.reshape((X_train_scaled.shape[0], 1,
X_train_scaled.shape[1]))
X_test_reshaped = X_test_scaled.reshape((X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))

# One-hot encoding für die Zielwerte
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)

# 2. Modell erstellen
model = Sequential([
    LSTM(100, input_shape=(1, X_train_reshaped.shape[2]), return_sequences=True),
    Dropout(0.2),
    LSTM(100),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dense(y_train_cat.shape[1], activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 3. Modell trainieren
history = model.fit(X_train_reshaped, y_train_cat, epochs=50, batch_size=32,
                    validation_split=0.2, verbose=1)

# 4. Modell evaluieren
loss, accuracy = model.evaluate(X_test_reshaped, y_test_cat, verbose=0)
print(f'Test accuracy: {accuracy*100:.2f}%')
```

```
# 5. Vorhersagen machen
predictions = model.predict(X_test_reshaped)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(y_test_cat, axis=1)

model.summary()

# 6. Erweiterte Metriken berechnen
mse = mean_squared_error(true_classes, predicted_classes)
f1 = f1_score(true_classes, predicted_classes, average='weighted')
recall = recall_score(true_classes, predicted_classes, average='weighted')
precision = precision_score(true_classes, predicted_classes, average='weighted')

print(f'Mean Squared Error: {mse:.4f}')
print(f'F1 Score: {f1:.4f}')
print(f'Recall: {recall:.4f}')
print(f'Precision: {precision:.4f}')

# 7. Konfusionsmatrix erstellen und visualisieren
cm = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens')
plt.title('Confusion Matrix LSTM RNN_1')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
num_classes = 8
tick_labels = list(range(1, num_classes + 1))
plt.xticks(np.arange(num_classes) + 0.5, tick_labels)
plt.yticks(np.arange(num_classes) + 0.5, tick_labels)
plt.savefig('Confusion Matrix LSTM RNN_1.png')

# 8. Trainings- und Validierungsverlauf visualisieren
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy RNN_1')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss RNN_1')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.savefig('training history LSTM RNN_1.png')
plt.show()
```

B.4 Code RNN_2

```
# improved RNN LSTM

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.regularizers import l2
from sklearn.metrics import mean_squared_error, f1_score, recall_score, precision_score,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Data loading and preprocessing (same as before)
df = pd.read_csv("C:\\Users\Raphael
Willeke\\Documents\\Uni\Masterthesis\\Datensatz\\dropped_werte.csv") # Dateipfad
```

```
df = df.drop(['Unnamed: 0', 'Label'], axis=1)
X = df.drop(['ID', 'Timestamp'], axis=1)
y = df['ID']
le = LabelEncoder()
y = le.fit_transform(y)

# Splitting and scaling
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshaping
X_train_reshaped = X_train_scaled.reshape((X_train_scaled.shape[0], 1,
X_train_scaled.shape[1]))
X_test_reshaped = X_test_scaled.reshape((X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))

# One-hot encoding
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)

# Improved model
def create_model():
    model = Sequential([
        Bidirectional(LSTM(128, return_sequences=True), input_shape=(1,
X_train_reshaped.shape[2])),
        Dropout(0.3),
        Bidirectional(LSTM(64)),
        Dropout(0.3),
        Dense(128, activation='relu', kernel_regularizer=l2(0.01)),
        Dense(y_train_cat.shape[1], activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```

```
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)

# K-fold cross-validation
n_splits = 5
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
cv_scores = []

for fold, (train_index, val_index) in enumerate(kf.split(X_train_reshaped)):
    print(f"Fold {fold+1}/{n_splits}")

    X_train_fold, X_val_fold = X_train_reshaped[train_index], X_train_reshaped[val_index]
    y_train_fold, y_val_fold = y_train_cat[train_index], y_train_cat[val_index]

    model = create_model()
    history = model.fit(X_train_fold, y_train_fold, epochs=100, batch_size=32,
                        validation_data=(X_val_fold, y_val_fold),
                        callbacks=[early_stopping, lr_scheduler], verbose=1)

    scores = model.evaluate(X_val_fold, y_val_fold, verbose=0)
    cv_scores.append(scores[1])
    print(f"Fold {fold+1} accuracy: {scores[1]*100:.2f}%")

print(f"Mean cross-validation accuracy: {np.mean(cv_scores)*100:.2f}% (+/-
{np.std(cv_scores)*100:.2f}%)")

# Final evaluation and metrics calculation
final_model = create_model()
final_history = final_model.fit(X_train_reshaped, y_train_cat, epochs=100, batch_size=32,
                                validation_split=0.2, callbacks=[early_stopping, lr_scheduler], verbose=1)

loss, accuracy = final_model.evaluate(X_test_reshaped, y_test_cat, verbose=0)

final_model.summary()

print(f'Test accuracy: {accuracy*100:.2f}%')

predictions = final_model.predict(X_test_reshaped)
```

```
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(y_test_cat, axis=1)

mse = mean_squared_error(true_classes, predicted_classes)
f1 = f1_score(true_classes, predicted_classes, average='weighted')
recall = recall_score(true_classes, predicted_classes, average='weighted')
precision = precision_score(true_classes, predicted_classes, average='weighted')

print(f'Mean Squared Error: {mse:.4f}')
print(f'F1 Score: {f1:.4f}')
print(f'Recall: {recall:.4f}')
print(f'Precision: {precision:.4f}')

# Confusion matrix visualization
cm = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens')
plt.title('Confusion Matrix Improved LSTM RNN_2_72')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
num_classes = 8
tick_labels = list(range(1, num_classes + 1))
plt.xticks(np.arange(num_classes) + 0.5, tick_labels)
plt.yticks(np.arange(num_classes) + 0.5, tick_labels)
plt.savefig('Confusion Matrix LSTM RNN_2 72.png')
plt.show()

# Training history visualization
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(final_history.history['accuracy'], label='Training Accuracy')
plt.plot(final_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy RNN_2 72')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(final_history.history['loss'], label='Training Loss')
plt.plot(final_history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss RNN_2 72')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.savefig('training history LSTM RNN_2 72.png')
plt.show()
```

B.5 Code RNN_3 keras tuner

```
#RNN LSTM keras tuner

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.metrics import mean_squared_error, f1_score, recall_score, precision_score,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from keras_tuner import RandomSearch
import tensorflow as tf

# 1. Daten laden und vorbereiten
df = pd.read_csv("C:\\Users\\Raphael
Willeke\\Documents\\Uni\\Masterthesis\\Datensatz\\dropped_werte.csv") #Dateipfad
df = df.drop(['Unnamed: 0', 'Label'], axis=1)
```

```
X = df.drop(['ID', 'Timestamp'], axis=1)
y = df['ID']
le = LabelEncoder()
y = le.fit_transform(y)

# Daten in Trainings- und Testsets aufteilen
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Daten normalisieren
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Daten für LSTM umformen (samples, time steps, features)
X_train_reshaped = X_train_scaled.reshape((X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
X_test_reshaped = X_test_scaled.reshape((X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))

# One-hot encoding für die Zielwerte
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)

# 2. Modell-Builder-Funktion für Keras Tuner definieren
def build_model(hp):
    model = Sequential()
    model.add(LSTM(units=hp.Int('lstm_units', min_value=32, max_value=256, step=32),
                   return_sequences=hp.Boolean('return_sequences'),
                   input_shape=(1, X_train_reshaped.shape[2])))
    model.add(Dropout(hp.Float('dropout_1', min_value=0.0, max_value=0.5, step=0.1)))

    if hp.Boolean('second_lstm_layer'):
        model.add(LSTM(units=hp.Int('lstm_units_2', min_value=32, max_value=256, step=32)))
        model.add(Dropout(hp.Float('dropout_2', min_value=0.0, max_value=0.5, step=0.1)))

    model.add(Dense(hp.Int('dense_units', min_value=32, max_value=256, step=32),
activation='relu'))
    model.add(Dense(y_train_cat.shape[1], activation='softmax'))
```

```
    model.compile(
        optimizer=tf.keras.optimizers.Adam(
            hp.Float('learning_rate', min_value=1e-4, max_value=1e-2, sampling='log')
        ),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

# 3. Keras Tuner initialisieren und ausführen
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=10,
    executions_per_trial=2,
    directory='keras_tuner',
    project_name='rnn_timeseries'
)

# Early Stopping und Learning Rate Schedule definieren
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
lr_schedule = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)

# 4. K-Fold Cross-Validation
n_splits = 5
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

for fold, (train_index, val_index) in enumerate(kf.split(X_train_reshaped)):
    print(f"Fold {fold+1}/{n_splits}")
    X_train_fold, X_val_fold = X_train_reshaped[train_index], X_train_reshaped[val_index]
    y_train_fold, y_val_fold = y_train_cat[train_index], y_train_cat[val_index]

    tuner.search(X_train_fold, y_train_fold,
                 epochs=60,
                 validation_data=(X_val_fold, y_val_fold),
                 callbacks=[early_stopping, lr_schedule])
```

```
# Bestes Modell abrufen und zusammenfassen
best_model = tuner.get_best_models(num_models=1)[0]
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print("Best Hyperparameters:", best_hps.values)
best_model.summary()

# 5. Bestes Modell auf gesamten Trainingsdaten trainieren
history = best_model.fit(X_train_reshaped, y_train_cat,
                         epochs=100,                # möglich zu variieren
                         batch_size=32,
                         validation_split=0.2,
                         callbacks=[early_stopping, lr_schedule],
                         verbose=1)

# 6. Modell evaluieren
loss, accuracy = best_model.evaluate(X_test_reshaped, y_test_cat, verbose=0)
print(f'Test accuracy: {accuracy*100:.2f}%')

# 7. Vorhersagen machen
train_predictions = best_model.predict(X_train_reshaped)
test_predictions = best_model.predict(X_test_reshaped)

train_predicted_classes = np.argmax(train_predictions, axis=1)
test_predicted_classes = np.argmax(test_predictions, axis=1)

train_true_classes = np.argmax(y_train_cat, axis=1)
test_true_classes = np.argmax(y_test_cat, axis=1)

# 8. Gesamtgenauigkeit berechnen
train_accuracy = accuracy_score(train_true_classes, train_predicted_classes)
test_accuracy = accuracy_score(test_true_classes, test_predicted_classes)
overall_accuracy = accuracy_score(
    np.concatenate([train_true_classes, test_true_classes]),
    np.concatenate([train_predicted_classes, test_predicted_classes])
)
```

```
print(f'Training Accuracy: {train_accuracy*100:.2f}%')
print(f'Test Accuracy: {test_accuracy*100:.2f}%')
print(f'Overall Accuracy: {overall_accuracy*100:.2f}%')

# 9. Erweiterte Metriken berechnen
mse = mean_squared_error(test_true_classes, test_predicted_classes)
f1 = f1_score(test_true_classes, test_predicted_classes, average='weighted')
recall = recall_score(test_true_classes, test_predicted_classes, average='weighted')
precision = precision_score(test_true_classes, test_predicted_classes, average='weighted')

print(f'Mean Squared Error: {mse:.4f}')
print(f'F1 Score: {f1:.4f}')
print(f'Recall: {recall:.4f}')
print(f'Precision: {precision:.4f}')

# 10. Konfusionsmatrix erstellen und visualisieren
cm = confusion_matrix(test_true_classes, test_predicted_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens')
plt.title('Confusion Matrix Optimized LSTM RNN_3 keras tuner')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
num_classes = 8
tick_labels = list(range(1, num_classes + 1))
plt.xticks(np.arange(num_classes) + 0.5, tick_labels)
plt.yticks(np.arange(num_classes) + 0.5, tick_labels)
plt.savefig('confusionmatrix RNN_3_kt.png')
plt.show()

# 11. Trainings- und Validierungsverlauf visualisieren
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy RNN_3 keras tuner')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
```

```
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss RNN_3 keras tuner')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.savefig('training history RNN_3_kt.png')
plt.show()
```