Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

Diplomarbeit

Untersuchung zur Qualität von Fertigungsdaten – Ein Beispiel für die Analyse großer Datenmengen

Andreas Laukart

Studiengang:	Informatik
Prüfer:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer:	M.Sc. Cornelia Kiefer
begonnen am:	20.12.16
beendet am:	31.03.17
CR-Klassifikation:	H.5.2, C.4, I.2.7

Kurzfassung

In dieser Arbeit wurde prototypisch ein Jupyter Notebook implementiert, dass beim Data Wrangling unterstützt. Hierzu wurde zunächst eine Literaturrecherche durchgeführt. Die Ergebnisse flossen in das Konzept ein. Hauptfokus ist der Aspekt der Datenqualität. Das Notebook versteht sich als flexible Toolbox. Es soll möglich sein, Metriken und Skripte bei Bedarf einzubinden. Hierzu wurde beispielhaft Funktionalität aus unterschiedlichen Quellen eingebunden. Verwendet werden die Sprachen Python, Java und R. Verschiedene Datenqualitätsmetriken ermöglichen es, die Datenqualität zu messen. Dabei werden neben strukturierte Daten auch Textdaten berücksichtigt. Anschließend wurde das Notebook auf Datensätze aus der Praxis angewendet. Hierzu wird ein Überblick über die Daten gegeben. Zusätzlich werden mögliche Datenqualitätsprobleme analysiert. Um die Verarbeitung großer Datenmengen zu unterstützen, wurde die Laufzeit der Metriken betrachtet. Hierzu wurden ausgewählte Metriken in Apache Spark implementiert. Anschließend wurde eine Evaluation durchgeführt. Die ersten Resultate sind vielversprechend. Für die Validierung von Datumsangaben wurde durch eine einfache Implementierung eine Verbesserung der Laufzeit um Faktor 3,6 erreicht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Die Verarbeitung elektronischer Daten spielt in der modernen Gesellschaft eine zentrale Rolle. Daten sind fundamentaler Bestandteil von operationalen Prozessessen von Unternehmen und anderen Organisationen. Zusätzlich bilden sie die Grundlage für Entscheidungen. Dabei ist die Qualität der Daten von entscheidender Bedeutung. Schätzungen zufolge führen Probleme mit der Qualität von Daten zu Kosten in Milliardenhöhe alleine in den USA. [3]

Dabei kommt dem Data Wrangling eine entscheidende Bedeutung zu. Dieser Begriff bezeichnet den Vorgang, Daten in eine nutzbare Form zu bringen. Hierzu zählt zum Beispiel das Säubern und Transformieren von Daten. Schätzungen zufolge muss in manchen Projekten bis zu 80% der Zeit für die Säuberung und Aufbereitung der Daten aufgewendet werden. [16]

In dieser Arbeit wird ein Jupyter Notebook entwickelt, das verschiedene Funktionen anbietet, die beim Data Wrangling unterstützen. Zusätzlich werden beispielhaft diverse Datenqualitätsmetriken angeboten, mit denen die Datenqualität gemessen und überwacht werden kann. Neben strukturierte Daten werden auch unstrukturierte Daten berücksichtigt. Es stehen drei Metriken zur Verfügung, die auf Textdaten arbeiten. Ziel ist eine flexible und erweiterbare Toolbox. Diese ist nicht für einen bestimmten Anwendungsfall oder eine Domäne gedacht. Anpassbarkeit und Erweiterbarkeit sind im Entwurf mit berücksichtigt. Um die Flexibilität des Ansatzes zu demonstrieren, werden viele Funktionen und Metriken aus Bibliotheken eingebunden. Dabei finden die Sprachen Java, Python und R Verwendung. Mit Hilfe dieses Notebooks werden zwei Datensätze aus der Praxis analysiert. Dabei handelt es sich zum einen um Daten aus der Produktions, zum anderen um den um den NHTSA Complaints Datensatz.

Die anfallenden Datenmengen werden immer größer, sollen aber trotzdem möglichst schnell verarbeitet werden [21]. Aus diesem Grund soll die Laufzeit der Datenqualitätsmetriken auch für große Datensätze berücksichtigt werden. Hierzu werden abschließend ausgewählte Metriken mit Apache Spark prototypisch optimiert. Durch Laufzeitmessungen wird evaluiert, ob sich dadurch die Laufzeit verbessern lässt..

1.1 Motivation

Bei einer Analyse hängt die Qualität der Erkenntnisse, und der Nutzen der daraus in Folge abgeleiteten Entscheidungen, stark von der Qualität der zu Grunde liegenden Daten ab [3].

Die zu verarbeitenden Datenmengen werden dabei immer größer und vielfältiger in ihrer Art und Struktur. Trotzdem sollen sie möglichst echtzeitnah verarbeitet werden. Diese Kombination aus Volumen, Vielfalt und Geschwindigkeit (im Englischen „Volume, Varierty, Velocity“) werden auch als die 3 V von Big Data bezeichnet. [21]

Dabei ist es wichtig, zu beachten, dass nicht nur Menschen, sondern auch Maschinen Daten konsumieren. Beispiele hierfür sind Algorithmen aus dem Bereich Machine Learning und Data Mining. Beispielsweise findet die automatische Verarbeitung von Texten in natürlichen Sprachen (im Englischen „natural language processing“, kurz NLP) häufig in Pipelines statt.

Dies spielt etwa bei der Analyse von Social Media Daten oder Freitextfeldern eine Rolle. Dabei werden verschiedene Machine Learning Algorithmen verwendet, die sukzessive aufeinander aufbauen. Dass Ergebnis eines Analyseschrittes dient dabei als Eingabe für einen weitergehenden Analyseschritt. Dadurch können sich Qualitätsprobleme verstärken und den Nutzen des Endresultats stark einschränken. Entscheidend ist neben klassischen Datenqualitätskriterien wie der Genauigkeit (im Englischen Accuracy) auch „fitness for use“. Die Daten müssen so bereitgestellt werden, wie sie die Datenkonsumenten erwarten, auch im Hinblick auf etwa Struktur oder Format. [18]

Damit eine Analyse überhaupt durchgeführt werden kann müssen Daten also in die entsprechende Form gebracht werden, ein Vorgang der auch als Data Wrangling bezeichnet wird. Dabei umfasst der Begriff neben dem Auffinden von Datenqualitätsproblemen und der Bereinigung der Daten auch die Integration heterogener Datenquellen. [16]

Data Wrangling ist ein sehr aufwendiger Schritt. Laut einer Studie von IBM wird bei Analyseprojekten bis zu 70% der Zeit dafür verwendet, Daten aufzufinden, zu säubern und zu integrieren. Hierfür werden drei Gründe angegeben: Daten sind über viele verschiedene Systeme und Applikationen verteilt. Zusätzlich müssen die Daten transformiert und in ein für die Analyse geeignetes Format gebracht werden. Außerdem müssen die Daten während des Analyseprozesses stets aktuell gehalten werden. [7]

Dies führt oft dazu, dass Domänenexperten, deren Fachexpertise für die eigentliche Analyse benötigt wird, sich stattdessen mit der Aufbereitung von Daten beschäftigen müssen. [16]

Die Analyse großer Datenmengen spielt in allen Phasen des Produktlebenszyklus eine Rolle. So zum Beispiel beim Marketing, Kundendienst, Logistik, aber auch bei der Abwicklung und beim Recycling. Hierzu können auch Daten aus den Sozialen Medien oder öffentlich verfügbare Datenquellen wie Wikipedia verwendet werden. [20].

Von zentraler Bedeutung für Industrieunternehmen, insbesondere in Ländern mit einem hohen Lohnniveau wie etwa Deutschland, ist die datengetriebene Optimierung von Fertigungsprozessen. Dadurch können Unternehmen Kosten senken und Produktqualität erhöhen und somit im globalen Wettbewerb bestehen. Im Kontext der Fertigung fallen große Mengen an strukturierten und unstrukturierten Daten an, etwa durch in Maschinen und Anlagen eingebettete Sensoren. Zusätzlich stehen oft Daten aus Manufacturing Execution Systems zur Verfügung, digitalen Systemen für das Fertigungsmanagement. Dabei ist es wichtig, nicht nur rückblickend zu analysieren, sondern den Prozess kontinuierlich zu überwachen um schon während der Ausführung mögliche Probleme vorauszusehen und präventiv eingreifen zu können. Deswegen ist eine echtzeitnahe Analyse entscheidend. [13]

1.2 Forschungsfrage und Aufgabenstellung

Zuerst wird im Grundlagenkapitel ein Überblick über die Themenfelder Datenqualität und Data Wrangling gegeben.

Basierend auf einer Literaturrecherche wird ein Konzept entwickelt, dabei sollen verschiedene Aspekte des Data Wranglings, die in der Literatur diskutiert werden, berücksichtigt werden.

Das Notebook soll diverse Datenqualitätsmetriken anbieten, und somit bei der Datenexploration und beim Data Wrangling unterstützten. Jupyter Notebooks basieren ursprünglich auf Python, sind aber nicht auf eine bestimmte Plattform oder Programmiersprache beschränkt. Spezielle Module, sogenannte Kernel, ermöglichen es, Notebooks mit unterschiedliche Programmiersprachen zu verwenden. Es wird eine Vielzahl von Programmiersprachen unterstützt, darunter zum Beispiel R. Diese Sprache wird im Bereich Datenanalyse und Data Wrangling häufig verwendet und bietet eine Vielzahl von Bibliotheken zur Auswertung und Visualisierung. Dies ermöglicht eine große Flexibilität. Es ist möglich, Werkzeuge und Datenqualitätsmetriken zu verwenden, die in unterschiedlichen Sprachen implementiert wurden. Im Rahmen dieser Arbeit wird untersucht, in wie weit dies möglich ist. Hierzu sollen verschiedene Metriken zur Messung der Datenqualität prototypisch implementiert sowie bereits verfügbare Metriken integriert werden.

Das Notebook soll neben diesen als ausführbarer Code bereitgestellten Metriken auch Textteile enthalten, etwa Checklisten, Erläuterungen und weitergehende Verweise. Ziel ist dabei kein fertiges Datenanalyseprodukt, sondern ein flexibel erweiterbarer Startpunkt. Das Notebook dient als Toolbox, die für verschiedene Situationen entsprechende Funktionen bereitstellt. Ist keine passende Funktion verfügbar, soll es möglich sein, vorhandene Implementierungen, etwa aus Bibliotheken, einzubinden.

Die Analyse soll auch für große Datenmengen möglich sein. Im Rahmen dieser Arbeit soll daher abschließend die Laufzeit von ausgewählten Metriken untersucht werden. Hierzu werden Experimente zur Laufzeitmessung durchgeführt. Verwendet wird dabei Beispielhaft die NHTSA Beschwerdedatenbank [41]. Als Evaluationsumgebung wird Openstack [40] genutzt. Die Optimierung erfolgt prototypisch durch Verwendung von Apache Spark [33].

1.3 Aufbau der Arbeit

Kapitel 1 enthält neben der Einführung auch die Motivation und die Forschungsfrage dieser Diplomarbeit. Kapitel 2 enthält einen kurzen Überblick über die theoretischen Grundlagen. Kapitel 3 bietet einen Überblick über verwandte Arbeiten und diskutiert Gemeinsamkeiten und Unterschiede zu dieser Diplomarbeit. Kapitel 4 gibt einen Überblick über ausgewählte Aspekte des Data Wranglings, die in der Literatur diskutiert werden. Zusätzlich wird dargestellt, was davon in dieser Arbeit berücksichtigt werden konnte. Kapitel 5 enthält darauf aufbauend das Konzept für das Jupyter Notebook. Kapitel 6 beschreibt die Implementierung des Notebooks und Kapitel 7 beschreibt die Implementierung der ausgewählten Metriken in Spark. Kapitel 8 enthält eine beispielhafte Analyse der Datenqualität zweier Datensätze durch Anwendung des Notebooks. Dieses Kapitel bietet auch praktische Beispiele für viele Funktionen des Notebooks. Kapitel 9 beschreibt die Evaluation der für Spark entwickelten Metriken. Kapitel 10 enthält eine Zusammenfassung und einen Überblick über mögliche weitergehende Forschungsansätze.

2 Grundlagen

2.1 Daten und Datenstrukturen

Die Onlineausgabe des Duden nennt für das Wort „Daten“ unter anderem folgende Bedeutungen [36]:

(1) „ (durch Beobachtungen, Messungen, statistische Erhebungen u. a. gewonnene) [Zahlen]werte“

(2) „(EDV) elektronisch gespeicherte Zeichen, Angaben, Informationen“.

Das Wort „Daten“ hat also eine starke Assoziation mit Zahlen und Messungen. Das Wort steht häufig im Zusammenhang mit Computern und bezieht sich dabei auf Fakten, die elektronisch übertragen und gespeichert werden. Ein Beispiel für eine solche Anwendung wäre etwa eine Datenbank. Bei diesen Fakten kann es sich etwa um Messungen handeln. Auch beschreibende Attribute von Entitäten sind möglich, etwa Namen und Orte. Daten können definiert werden als abstrakte Repräsentation ausgewählter Merkmale von Konzepten, Ereignissen und Objekten aus der realen Welt. Bei Daten handelt es sich immer um Repräsentationen. Das bedeutet auch, dass es oft mehrere Möglichkeiten gibt, das Gleiche zu repräsentieren. Wichtig ist auch, dass Daten nur ausgewählte Eigenschaften darstellen. Dabei handelt es sich gewissermaßen um ein Modell der Realität. Modelle spielen eine wichtige Rolle und helfen beim Verständnis verschiedener Aspekte der Realität. Allerdings handelt es sich bei Modellen um Vereinfachungen. Dies kann unter Umständen zu Problemen führen. Ein weiterer wichtiger Aspekt ist, dass Daten nicht an sich existieren, sondern geschaffen werden, etwa durch Prozesse. Der Kontext, in dem die Daten entstanden sind, spielt eine Entscheidende Rolle beim Verständnis der Bedeutung der Daten. [26]

Die Bedeutung des Kontextes kann an folgendem Beispiel gezeigt werden. Dieses Beispiel erweitert ein ähnlichen Beispiel in [26]:

Auf den ersten Blick erkennt man in der Zeichenfolge „10/03/2017“ eine Datumsangabe. In Deutschland versteht man diese im Allgemeinen als 10. März 2017. Allerdings ist es auch denkbar, dass die ersten beiden Zahlen den Monat repräsentieren anstelle des Tages. Dieses Format wird zum Beispiel in den Vereinigten Staaten verwendet. Damit würde sich die Datumsangabe auf den 3. Oktober 2017 beziehen. Es ist sogar theoretisch denkbar, dass es sich bei der Zeichenkette nicht um ein Datum handelt, sondern um eine Pfadangabe in einem Dateisystem. Abbildung 1 zeigt hierfür ein einfaches Beispiel für Linux. Ohne Kontext ist der Wert an sich nicht eindeutig.

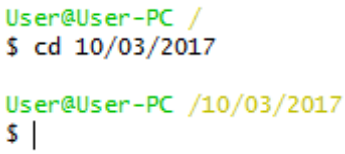

Abbildung 1: Beispiel für ein Linux-System. Hier steht 10/03/2017 nicht für ein Datum sondern eine Pfadangabe

Daten können auf verschiedene Weise organisiert sein. Zuerst wird allgemein zwischen strukturierten und unstrukturierten Daten unterschieden. Strukturierte Daten folgen einem festgelegtem Schema. Dieses erlaubt die Interpretation der entsprechenden Werte. Ein wichtiges Beispiel sind Datenbanken, die Informationen in relationalen Tabellen verwalten. Unstrukturierte Daten können aus einer beliebigen Folge von Zeichen bestehen. Beispiele hierfür sind Bilder und Texte. [3]

Es gibt verschiedene Datenstrukturen, mit denen Daten gespeichert und transportiert werden können, etwa XML oder CSV. CSV steht dabei für Comma Separated Values, also durch Kommas getrennte Datenwerte. Jede Zeile repräsentiert dabei einen Eintrag. Die Werte der jeweiligen Spalten werden durch Kommas getrennt. Diese Darstellung entspricht einer Tabelle. In der Praxis werden neben Kommas auch weitere Trennzeichen verwendet, etwa der Tabulator. [22]

Viele Tools, die solche Daten verarbeiten, erlauben es, beim Einlesen das Trennzeichen zu spezifizieren. Dies ist auch für das in der Arbeit entwickelte Jupyter Notebook der Fall. Aus diesem Grund wird im Kontext dieser Arbeit nicht strikt nach dem Trennzeichen unterschieden. Dem allgemeinen Sprachgebrauch folgend werden auch Dateien mit anderen Trennzeichen als dem Komma als CSV bezeichnet. Dies ist zum Beispiel für die in Kapitel 8 analysierten Datensätze der Fall. Der NHTSA Complaints Datensatz liegt als Textdatei vor, die als Trennzeichen den Tabulator verwenden. Die Daten des Industriepartners verwenden .csv als Dateiendung, nutzen aber das Semikolon als Trennzeichen.

2.2 Datenqualität

Daten spielen in der modernen Gesellschaft eine wichtige Rolle. Sie bilden oft die Grundlage für Entscheidungen, etwa durch Unternehmen und andere Organisationen. Dabei ist die Qualität der Daten entscheidend. Ein Konzept, das auf dem Begriff Datenqualität aufbaut und diesen ergänzt, ist die Informationsqualität. Dieser kommt bei Entscheidungen eine zentrale Bedeutung zu. Schlechte Informationsqualität kann Entscheidungen negativ beeinflussen und sich negativ auf die Resultate auswirken. [3]

Eine Sichtweise auf das Konzept Datenqualität ist die die Perspektive der Datenkonsumenten, also diejenigen Anwender, welche die Daten nutzen. Diese Sichtweise wird in der Literatur auch als „fitness for use“ bezeichnet. Die Daten müssen für die Verwendung durch die jeweiligen Datenkonsumenten geeignet sein. [29]

Datenqualitätsprobleme betreffen aber nicht nur menschliche Datenkonsumenten. Auch Systeme und Algorithmen verarbeiten Daten und können davon betroffen sein. Eine schlechte Qualität der Eingabedaten kann die verarbeitenden Algorithmen beeinträchtigen und zu einen schlechten Resultat führen. Dieses Problem verstärkt sich, wenn Daten in einer Pipeline verarbeitet werden. Dieser Englische Begriff bedeutet, dass eine Reihe von Schritten nacheinander ausgeführt werden. Die Ergebnisse des ersten Schrittes dienen dabei als Eingabe für den nächsten. Pipelines spielen zum Beispiel in der maschinellen Sprachverarbeitung eine wichtige Rolle. [18]

Ein Beispiel hierfür ist die Bestimmung der Wortarten[1]. Bevor ein Text mit den entsprechenden Tags versehen werden kann, muss dieser zuerst in Tokens zerlegt werden [4]. Siehe hierzu auch Abschnitt 2.5 über maschinelle Sprachverarbeitung.

Datenqualität hat mehrere verschiedene Aspekte, die auch als Datenqualitätsdimensionen bezeichnet werden können [3]. Ein in der Literatur häufig zitierte Qualitätsdimension ist Accuracy [28]. Im Deutschen kann für diesen Begriff zum Beispiel Genauigkeit verwendet werden. Dies bezeichnet, inwieweit ein Datenwert dem realen Phänomen gleicht, das er repräsentieren soll [3].

Es gibt in der Literatur eine Reihe von Ansätzen, Datenqualitätsdimensionen zu charakterisieren. Ein in [3] vorgestellter Ansatz gruppiert die Datenqualitätsdimensionen in 9 verschiedene Cluster. Diese Einteilung erfolgt anhand der Ähnlichkeit der einzelnen Dimensionen untereinander. So wird der oben erwähnte Begriff der Genauigkeit beziehungsweise Accuracy als Cluster aufgefasst, der unter anderem die Dimensionen Korrektheit und Präzision enthält. Weitere Cluster wären Vollständigkeit und Konsistenz. [3]

Ein weiterer Ansatz, vorgestellt in [29], ermittelte die Datenqualitätsdimensionen empirisch durch Fragebögen. Damit sollen die Dimensionen berücksichtigt werden, die für Datenkonsumenten eine Bedeutung haben. Basierend auf den Ergebnissen der Fragebögen wurde ein hierarchisches Framework mit 4 Kategorien entwickelt. Diese gruppieren ähnliche Datenqualitätsdimensionen. [29]

Ein Framework, das für die Überwachung und Messung der Datenqualität durch Metriken entwickelt wurde ist das Data Quality Assessment Framework (DQAF). Hier werden fünf Datenqualitätsdimensionen benutzt: Vollständigkeit, Rechtzeitigkeit, Validität, Konsistenz und Integrität. Diesen Dimensionen sind 48 Messarten zugeordnet. Hierdurch wird das wiederholbare Messen der Datenqualität möglich. Eine Messart ist dabei eine Verallgemeinerung und Zusammenfassung mehrerer spezieller Metriken. [26]

Siehe den folgenden Abschnitt 2.3 für eine nähere Betrachtung von Datenqualitätsmetriken.

1 Im Englischen „part-of-speech tagging, oder Abgekürzt POS tagging

2.3 Datenqualitätsmetriken

Datenqualität kann durch Metriken gemessen werden. Messungen helfen dabei, Beobachtungen durch Zahlen auszudrücken. Dadurch sind Vergleiche möglich. So können Objekte miteinander verglichen werden, oder die Entwicklung eines Objektes im Laufe der Zeit. Die ermittelten Messwerte können dabei als Grundlage von Entscheidungen dienen. Hierzu müssen die Messungen **verständlich**, **reproduzierbar** und **zielführend** sein. [26]

Diese drei Punkte sollen im folgenden näher erläutert werden:

Messungen müssen **verständlich** sein, damit sie ihren Zweck erfüllen. Die Ergebnisse können nicht bei der Entscheidungsfindung helfen, wenn niemand versteht, was gemessen wurde und was die Ergebnisse genau bedeuten. Dies unterstreicht die Bedeutung von Metadaten, welche die Messungen und die Ergebnisse dokumentieren. Dies hilft dem Datenkonsumenten, den Kontext zu verstehen und die Resultate zu interpretieren. [26]

Messungen müssen **reproduzierbar** sein. Inkonsistente Messungen führen dazu, dass die Ergebnisse wenig oder gar keine Bedeutung haben. Um zu zeigen, ob sich die Qualität in einem Datensatz verbessert oder verschlechtert, müssen die gleichen Daten mit den gleichen Methoden gemessen werden. Dadurch sind auch Vergleiche zwischen verschiedenen Objekten möglich. [26]

Messungen müssen **zielführend** sein. Es sollte das gemessen werden, was dabei hilft, die Unsicherheit bei einer Entscheidung zu reduzieren. Messungen dienen also einem Zweck und helfen bei konkreten Problemen. [26]

Mit Hilfe einer Datenqualitätsmetrik können bestimmte Aspekte der Qualität eines Datensatzes untersucht werden. So kann beispielsweise der Anteil an ungültigen Codes in einer bestimmten Spalte gemessen werden. Dabei spielt es keine Rolle, um welchen Code es sich genau handelt. Dies kann für jede Spalte ermittelt werden, die einen definierten Wertebereich hat. Dieser Wertebereich kann zum Beispiel durch eine zweite Tabelle definiert sein. Diese Tabelle enthält die zulässigen Werte, beispielsweise Diagnosecodes im medizinischen Kontext. Die Messung ungültiger Werte lässt sich als Validität verallgemeinern. Dabei handelt es sich um den prozentualen Anteil der Werte pro Spalte, die nicht in der für diese Spalte definierten Domäne existieren. Dies erlaubt Vergleiche von ähnlichen Datensätze. Enthalten zwei Datensätze etwa vergleichbare Spalten, so kann für diese die Validität ermittelt werden. Dadurch ist es möglich, die relative Datenqualität dieser Datensätze zu ermitteln. [26]

2.4 Wissensentdeckung

In vielen unterschiedlichen Bereichen fallen immer größere Mengen von Daten an. Die Wissensentdeckung in Datenbanken beschäftigt sich mit Werkzeugen und Methoden, mit deren Hilfe aus diesen wachsenden Datenmengen nützliche Informationen gewonnen werden können. Im Englischen wird dies als „knowledge discovery in databases“ bezeichnet, kurz KDD. Wissensentdeckung in Datenbanken bezeichnet den Prozess, mit dem aus Daten nützli-

ches Wissen extrahiert wird. Kern dieses Prozesses ist die Anwendung von Methoden des Data Minings. In Abgrenzung dazu kann Data Mining als ein einzelner Schritt im gesamten Prozess gesehen werden. Die Verarbeitung von großen Datensätzen aus der Praxis ist ein zentraler Fokus. Deshalb spielt der Aspekt der Skalierbarkeit von Algorithmen für große Datensätze eine wichtige Rolle. [10]

Der KDD Prozess ist iterativ und interaktiv, und besteht aus mehreren Schritten. Dabei soll zunächst ein Verständnis über die relevante Domäne erlangt werden. Zusätzlich muss auch das Ziel des Prozesses aus Sicht des Auftraggebers geklärt werden. Ein wichtiger Schritt ist die Bereinigung und Vorverarbeitung der Daten. Dabei geht es unter anderem auch darum eine Strategie festzulegen, wie mit fehlenden Werten umgegangen werden soll. Ein zentraler Schritt des Prozesses ist das Data Mining. Dabei geht es darum, nach interessanten Mustern in den Daten zu suchen, etwa durch Clustering. Danach müssen die gewonnenen Muster interpretiert werden. Hierbei können auch verschiedene Visualisierungen angewendet werden. Abschließend muss das gewonnene Wissen konsolidiert werden. Dies geschieht beispielsweise, indem die Ergebnisse dokumentiert und an die entsprechenden Interessenten weitergeleitet werden. [10]

Data Mining ist ein wichtiger Schritt des gesamten Prozesses. Dabei geht es um die wiederholte Anwendung verschiedener Data Mining Methoden. Dieser Prozess ist iterativ. Es können grob zwei Arten von Zielen unterschieden werden: die Verifikation und die Entdeckung. Bei der Verifikation geht es darum, vom Hypothesen des Benutzers zu überprüfen. Bei der Entdeckung sollen automatisch Muster gefunden werden. Diese kann in zwei weitere Unterkategorien unterteilt werden, Vorhersage und Beschreibung. Bei der Vorhersage geht es darum, zukünftiges Verhalten vorherzusagen. Bei der Beschreibung Muster in den Daten zu finden und dem Benutzer auf verständliche Weise zu präsentieren. Typische Data Mining Methoden sind dabei zum Beispiel Clustering und Klassifikation. [10]

2.5 Maschinelle Sprachverarbeitung

Die Verarbeitung von Texten in natürlicher Sprache hat zunehmende Bedeutung etwa in der Wissenschaft und der Wirtschaft [4]. Beispiel hierfür ist die Analyse von Social Media Daten, etwa für die Plattform Twitter [12].

Die Verarbeitung findet dabei häufig in Pipelines statt. Dies bedeutet, dass einzelne Schritte aufeinander aufbauen und hintereinander ausgeführt werden müssen. Die Ergebnisse eines Schrittes dienen dabei als Eingabe für folgenden Schritt. Dies kann Datenqualitätsprobleme verstärken. [18]

Die Annotation der Wortarten spielt in der maschinellen Sprachverarbeitung eine wichtige Rolle. Viele gängige Standardtools und Bibliotheken benutzen den englischen Namen „part-of-speech tagging“ oder die Abkürzung „POS tagging“ für entsprechende Funktionen. Dies gilt zum Beispiel OpenNLP [43], das in dieser Arbeit verwendet wird. Deshalb werden diese Begriffe im Folgenden auch synonym verwendet. Grundlage für die Annotation der Wortarten ist die Tokenisierung, die Zerlegung des Textes in Tokens.

2.5.1 Tokenisierung

Bei der Tokenisierung handelt es sich um eine Fundamentale Aufgabe in der maschinellen Sprachverarbeitung. Sie dient als Grundlage für viele weitere Verarbeitungsschritte, beispielsweise die Annotation der Wortarten. Siehe hierzu Abschnitt 2.5.2. Bei der Tokenisierung soll ein Text in die Tokens zerlegt werden, Grundeinheiten eines Textes. In den meisten Fällen handelt es sich dabei um Wörter und Satzzeichen. Es gibt aber auch Ausnahmen. Beispielsweise wird im Englischen häufig die Kontraktion „don't“ für „do not“ verwendet. Je nach Anwendung kann diese in die Tokens „do“ und „n't“ beziehungsweise „not“ aufgeteilt werden. Aus technischer Sicht ist ein in einem Computer gespeicherter Text zunächst nur eine Folge von Zeichen. Durch die Tokenisierung wird dieser String in eine Liste von Tokens umgewandelt. Es gibt verschiedene Ansätze, die Tokenisierung durchzuführen. Die einfachste Methode ist es, Leerzeichen als Trennzeichen zu verwenden. Dies ist aber für viele Anwendungen ungeeignet, weil die Satzzeichen nicht als einzelne Tokens erkannt werden, weil sie in der Regel nicht durch ein Leerzeichen von einem Wort getrennt sind. Es gibt auch komplexere Implementierungen, die auf Machine Learning basieren, und teilweise Modelle benötigen. [4]

Ein Beispiel hierfür ist der TokenizerME aus dem Paket OpenNLP [43]. ME steht dabei für Maximum Entropy. Dieser Tokenizer muss ein Modell laden, bevor damit Texte verarbeitet werden können. [43]

2.5.2 Part-of-speech Tagging

Part-of-speech tagging dient als Grundlage für viele weitergehende Verarbeitungsschritte. Die Textdaten müssen dafür mit den entsprechenden Wortarten annotiert werden. Dies geschieht dadurch, das jedem Token die passende Wortart zugeordnet wird, beispielsweise Nomen oder Verb. Hierfür muss ein Text zuerst Tokenisiert werden. Das bedeutet, dass der Text in die einzelnen Tokens zerlegt wird. Bei Tokens handelt es sich in der Regel um Wörter. Siehe hierzu auch den vorgehenden Abschnitt 2.5.1. Dies ist ein Beispiel für eine einfache Pipeline. Als Eingabe für das Part-of-speech tagging dient dabei das Ergebnis der Tokenisierung. Part-of-speech tagging kann entweder manuell durch Experten durchgeführt werden, oder durch automatisierte Tools, sogenannte Part-of-speech tagger. Es gibt viele verschiedene Implementation. Es werden etwa trainierbare Machine Learning Algorithmen verwendet. Diese benötigen ein Modell, welches vor der Anwendung erstellt werden muss. Anhand dieses Modell wird dann im Anwendungsfall eine Entscheidung getroffen, welche Wortart dem jeweiligen Token zugeordnet wird. [4]

2.5.3 Textähnlichkeit

Textähnlichkeit beschreibt die Ähnlichkeit von zwei Texten. Dabei kann es sich um Sätze, Paragraphen oder ganze Bücher handeln. Ein Beispiel für Ähnlichkeit von Texten sind zwei Definitionen für das gleiche Konzept, die aus unterschiedlichen Wörterbüchern stammen. Textähnlichkeit lässt sich schwer durch Algorithmen berechnen. Es gibt viele Möglichkeiten, das gleiche zu sagen. Natürliche Sprache zeichnet sich durch eine große Vielfalt aus. Dies wird zusätzlich dadurch erschwert, das es mehrere Dimensionen gibt, nach denen die Ähnlichkeit von Texten beurteilt werden kann, etwa Inhalt, Struktur und Stil. [1]

In der Literatur werden eine Vielzahl von Metriken zur Messung von Textähnlichkeit diskutiert. Diese basieren auf verschiedenen Ansätzen. Beispielsweise gibt es Metriken, die auf Ebene der Buchstabendarstellung arbeiten. Diese sogenannten string distance metrics berücksichtigen also keine semantischen Informationen. Andere Metriken verwenden Vektordarstellungen der Texte, und berechnen darauf basierend die Textähnlichkeit durch Vektorähnlichkeitsfunktionen. Ein Beispiel hierfür ist die Cosine Metrik. [1]

Eine Textähnlichkeitsmetrik misst dabei die Ähnlichkeit von zwei Texten. Das Ergebnis wird als Zahl ausgedrückt. Diese Zahl wird oft normalisiert, damit sie im Intervall zwischen 0 und 1 liegt. 0 Bedeutet, dass sich die Texte nicht ähnlich sind. 1 bedeutet vollkommen ähnlich. [2]

2.6 Spark

Apache Spark ist ein generelles Framework für die Verarbeitung großer Datenmengen. Es kann bis zu 100 mal schneller sein als Hadoop MapReduce, falls die Datenverarbeitung im Hauptspeicher erfolgt. Für Festplatten ist eine Steigerung um bis zu Faktor 10 möglich. Für die Entwicklung können die Sprachen Java, Scala, Python und R verwendet werden. [33]

Das zugrundeliegende Programmiermodell basiert auf MapReduce. Zusätzlich bietet Spark „Resilient Distributed Datasets“, abgekürzt RDD. Dabei handelt es sich um eine Abstraktion, mit der Daten verteilt werden können. Durch diese Erweiterung ist es möglich, zusätzlich verschiedene Arbeitsbereiche abzudecken, wie etwa SQL oder Machine Learning. Abbildung 2 zeigt den Spark Software Stack. Die dargestellten Komponenten können in Spark genutzt werden. Die Implementierungen verwenden die gleichen Optimierungsmethoden, wie sie in spezialisierten Engines genutzt werden. Dabei basieren sie auf der gemeinsamen Grundlage von Spark. Durch die einheitliche Schnittstelle können neue Anwendungen leichter entwickelt werden. Zweitens können dadurch Arbeitsaufgaben effizienter kombiniert werden. In Spark können verschiedene Funktionen auf den Daten ausgeführt werden. Dies ist oft sogar im Hauptspeicher möglich. Durch die Einbindung von Hadoop kann Spark auf das Hadoop Distributed File System (HDFS) zugreifen. [31]

Das besondere an Spark sind RDDs. Hierbei handelt es sich um eine fehlertolerante Datenstruktur. Sie stellt eine Sammlung von Objekten dar. Diese werden über einen Cluster verteilt und können parallel bearbeitet werden. RDDs unterstützen dabei verschiedene Funktionen wie etwa das Filtern. Dieses werden verzögert ausgewertet. Dadurch kann Spark die Ausführung optimieren. [31]

Abbildung 2: Für Apache Spark verfügbare Komponenten. Entnommen aus [31]

2.7 Jupyter Notebooks

Bei Jupyter Notebooks handelt es sich um Webapplikationen. Diese ermöglichen das Erstellen und Teilen von Dokumenten, die etwa ausführbaren Code, Visualisierungen und erklärende Texte enthalten können. Bibliotheken, die zur Verarbeitung große Datenmengen geeignet sind, wie etwa Pandas [46], können leicht integriert werden. Es werden 40 verschiedene Programmiersprachen unterstützt, unter anderem Python und R. Python sofort nach der Installation genutzt werden. Die Einbindung anderer Programmiersprachen erfolgt durch Installation spezieller Module. Das Projekt Jupyter ist Open Source. Die Funktionen des Notebooks basieren alle auf dem IPython Kernel. Dabei handelt es sich um einen separaten Prozess, der unter anderem für die Ausführung des Codes verantwortlich ist. Verschiedene Frontends können sich dabei mit einem Kernel verbinden. Sie haben dadurch zugriff auf den Speicher und die darin enthaltenen Variablen. Dies war ursprünglich für die Entwicklung unterschiedlicher Ansichten für den gleichen Kernel gedacht. Dadurch ist es auch möglich, das mehrere Benutzer an einem Projekt arbeiten und den Datenstand teilen. Beim Jupyter Notebook handelt es sich genau genommen um ein solches Frontend. Dies hat die besondere Funktion, das Code und Texte in einem Dokument gesichert werden. Dieses verwendet das JSON Format und die Dateiendung .ipynb. Dadurch ist es möglich, Notebooks einfach zu teilen. Angeboten wird auch der Export in andere Formate, beispielsweise HTML oder LaTeX. [47]

3 Verwandte Arbeiten

Dieses Kapitel stellt verwandte Arbeiten vor, und beschreibt Gemeinsamkeiten und Unterschiede zu dieser Diplomarbeit.

Data Wrangler [17] ist ein interaktives Tool für Datentransformationen. Es ist online unter [38] in einer ursprünglichen Version verfügbar. Dieses Tool wird nicht mehr weiterentwickelt. Das Unternehmen TriFacta [50], das von Mitgliedern des Forschungsteams gegründet wurde, bietet nun unter anderem eine weiterentwickelte Version zum Download an. Hierfür wird eine Registrierung mit Angabe der vollständigen Kontaktdaten verlangt. Die folgende Beschreibung bezieht sich daher auf die ursprüngliche Version [17]. Das Tool bietet eine Vorschau für geplante Transformationsschritte, damit sich der Benutzer die genauen Auswirkungen an einem Beispiel veranschaulichen kann, so wie eine Liste von bereits vorgenommenen Schritten. Transformationen können auch rückgängig gemacht werden. In dieser Arbeit ist dies nicht direkt möglich. Eine Reihe von Transformationen können als Skript gespeichert und mit Kommentaren dokumentiert werden. Somit können Transformationen zu einem späteren Zeitpunkt wiederholt werden, etwa bei neuen Datensätzen. In dieser Arbeit ist dies nicht automatisch möglich. Ein besonderes Feature ist, dass das Tool versucht, Vorschläge für Transformationsschritte zu geben, etwa dem Auffüllen oder löschen leerer Zeilen. Außerdem kann das Tool anhand von Beispielen komplexere Transformationen erkennen, etwa für die Extraktion von Zahlen aus einem Freitextfeld. Machine Learning wird verwendet, um die Qualität der Vorschläge mit steigender Nutzung zu verbessern. In dieser Arbeit erfolgt die Unterstützung des Benutzers hauptsächlich durch Texte, die weitergehende Informationen zu Metriken und Transformationen bieten. Es werden keine automatischen Vorschläge für Transformationen angeboten. Das Tool ist Tabellenorientiert. Über jeder Spalte gibt es einen Datenqualitätsbalken der einfache Qualitätsmetriken darstellt, wie etwa fehlende Werte oder Werte vom falschen Datentyp. Die Datenqualitätsmetriken sind einfach und dienen vor allem zur Unterstützung der Datentransformation. Im Gegensatz dazu verwendet diese Arbeit auch zusätzlich komplexere Metriken, insbesondere für Textdaten. Das Thema Optimierung der Laufzeit wird nicht angesprochen. In dieser Arbeit werden ausgewählte Metriken mit Verwendung von Apache Spark implementiert und auf die Laufzeit hin evaluiert.

Ein frei verfügbares Tool ist OpenRefine [45]. Dieses Tool ist auch unter dem zuvor verwendeten Namen GoogleRefine bekannt. Damit lassen sich Daten säubern und transformieren. Das Tool bietet eine tabellenförmige Darstellung der Daten. Wie in dieser Arbeit auch ist es möglich, Daten zu filtern. Es wird eine Reihe von vorgegebenen Transformationsschritten angeboten. Bei dieser Arbeit ist es möglich, Funktionen bei Bedarf direkt im Notebook durch Code zu implementieren oder aus entsprechenden Skripten einzubinden. Ein Feature, dass in dieser Arbeit nicht betrachtet wird, ist die Möglichkeit, verschiedene Webservices zu verwenden, etwa um freie Datensätze einzubinden. Im Unterschied zu dieser Arbeit liegt der Fokus nicht auf Qualitätsmetriken.

Ein weiterer Ansatz für die Transformation, Integration und Analyse von Daten in der Literatur ist DataCommandr [25]. Das Hauptfeature ist die Spaltenorientierung, statt der ansonsten üblichen relationalen Ansatz mit der Betonung von Tabellen und Reihen. Als Grund für diesen Ansatz wird die Laufzeitoptimierung genannt. Allerdings bezieht sich eher auf weiter-

gehende Analysen. Im Gegensatz hierzu wird in dieser Diplomarbeit auch die Laufzeit von Datenqualitätsmetriken betrachtet, mit denen die Datenqualität ermittelt und überwacht werden kann. Das Tool verwendet eine domänenspezifische Sprache, COEL (concept-oriented expression language). Beziehungen zwischen Daten werden durch Links realisiert, die viel einfacher zu handhaben sind als Joins. Dadurch soll sich zusätzlich die Laufzeit von komplexen Analysen deutlich verbessern. In dieser Arbeit werden keine speziellen Optimierungen oder Darstellungsformen entwickelt. Stattdessen werden, wenn möglich, Standardbibliotheken verwendet. Im Gegensatz zu dieser Arbeit wird Datenqualität nicht diskutiert. Auch wird dem Benutzer hier keine Hilfestellung etwa durch weiterführende Texte oder Checklisten angeboten.

Es sind einige Jupyter Notebooks zum Thema Data Wrangling verfügbar. Ein Beispiel ist [37]. Dieses Notebook bietet Inspektion und Transformation von Daten. Wie in dieser Diplomarbeit werden Textfelder zur Beschreibung und Erläuterung genutzt. Allerdings ist es im Gegensatz zu dieser Arbeit als Einführung gedacht und arbeitet auf konkreten Beispielen. Die Themen Datenqualität und Laufzeit werde nicht angesprochen.

Ein weiteres Beispiel ist [27]. In dieser Arbeit wurde eine spezielle Fragestellung aus dem Bereich der Ozeanologie untersucht. Dazu wird ein Jupyter Notebook verwendet. Dabei werden Daten aufbereitet und gefiltert, um in der anschließenden Analyse verwendete werden zu können. Im Gegensatz zu dieser Arbeit ist das Notebook für einen speziellen Anwendungsfall in einer bestimmten Domäne gedacht. Auch werden keine Metriken angeboten, mit denen die Qualität der Daten gemessen werden kann.

Es gibt in der Literatur viele Arbeiten über Data Wrangling. Im folgenden werden drei wichtige Arbeiten exemplarisch herausgegriffen. Ein kurzer Überblick über das Thema Data Wrangling findet sich bei [9].

Die Arbeit von Kandel et al. [16] diskutiert verschiedenste Themen und Probleme aus dem Bereich Data Wrangling, insbesondere unter dem Aspekt der Visualisierung. So können zum Beispiel Schaubilder und Diagramme dabei helfen, Datenqualitätsprobleme zu erkennen, wie etwa fehlende Werte. Diese Diplomarbeit greift einige der in diesem Paper diskutierten Aspekte und Erkenntnisse auf. Der Hauptunterschied ist, dass in dieser Arbeit eine prototypische Implementierung durchgeführt wird. Zusätzlich werden Datenqualitätsmetriken bereitgestellt, insbesondere für Textdaten.

Das Buch „Python for Data Analysis“ [23] behandelt das Thema Data Wrangling als Teilaspekt der Datenanalyse unter Verwendung der Programmiersprache Python als wichtigstes Tool. Zusätzlich wird unter anderem IPython behandelt, der Vorläufer und die Basis von Jupyter Notebooks. Eine Ähnliche Einführung für die Programmiersprache R bietet [5]. Darüberhinausgehend soll das in dieser Arbeit prototypisch erstellte Notebook nicht auf Python, R oder eine andere bestimmte Programmiersprachen oder Plattformen beschränkt sein, sondern bei Bedarf verschiedene Implementierungen integrieren können.

Der Einsatz von Data Analytics in der Produktion wird in der Literatur oft diskutiert. So beschäftigt sich zum Beispiel [13] mit der Datengetriebenen Optimierung von Produktionsprozessen. Hierzu wurde die Plattform AdMA (Advanced-Manufacturing-Analytics) prototypisch implementiert und evaluiert, die Datenhaltung mit Wissen über den Produktionsprozes-

se sowie darauf aufbauende Analysen vereint. Zusätzlich können die Erkenntnisse über eine Mobile App auch Werkern und Fertigungsleitern präsentiert werden. Ähnlich wie diese Arbeit wird eine Mischung aus strukturierten und unstrukturierten Daten unterstützt und ganzheitlich betrachtet. Allerdings konzentriert sich diese Arbeit auf den Aspekt der Datenqualität im Kontext von Data Wrangling. Weitergehende Analysen wie etwa Data Mining sowie Aspekte der Präsentation stehen nicht im Fokus dieser Diplomarbeit. Zusätzlich ist das Toolkit flexibel gestaltet und nicht für eine bestimmte Domäne, wie etwa Produktionsprozesse, zugeschnitten. Jedoch wird das Notebook auch auf Daten angewendet, die aus der Produktion stammen.

Die Notwendigkeit, beim Thema Datenqualität nicht nur Menschen sondern auch Maschinenkonsumenten, wie etwa Data Mining Algorithmen, zu betrachten, wird in [18] betont. Dies spielt eine noch größere Rolle, wenn Analysen in einer Pipeline stattfinden, also wenn das Ergebnis eines Analyseschrittes direkt als Ausgangsbasis für den nächsten Schritt dient.

Datenqualität im allgemeinen spielt in der Literatur eine sehr große Rolle. Der in [29] präsentierte Ansatz betrachtet Datenqualität empirisch. Hierzu wurden Experten, die häufig mit Daten arbeiten, befragt. Daraus werden 4 Kategorien von Datenqualität abgeleitet, die wiederum insgesamt 15 Datenqualitätsdimensionen enthalten, zum Beispiel Accuracy oder Completeness.

Ein umfassender Überblick über das Thema und die Beschreibung verschiedener Datenqualitätsmetriken findet sich in [3]. Der Zusammenhang zwischen der Qualität der Daten und der Qualität der darauf aufbauenden Analysen spielt eine wichtige Rolle.

Ein Framework zur Messung von Datenqualität wird in [26] präsentiert. Es werden 48 konkrete Metriken besprochen, teilweise mit Implementierungsskizzen. Der Fokus liegt auf dem Aspekt der Messung von Datenqualität. Im Gegensatz zu dieser Arbeit werden Aspekte des Data Wrangling nicht betrachtet. Zusätzlich spielt die Laufzeit der Metriken eine untergeordnete Rolle. Diese Arbeit greift Ideen aus diesem Buch auf und implementiert zwei der beschriebenen Metriken.

Das „Bad Data Handbook“ [22] betrachtet das Thema Datenqualität sehr praktisch und vielfältig. Ein wichtiger Aspekt ist, das sehr viele Daten, etwa in Tabellenverarbeitungsprogrammen, in einem für menschliche Verwendung optimierten Form vorhanden sind. Diese ist für Maschinen schwer zu verarbeiten und muss aufwendig transformiert werden. Auch ungewöhnliche Datenqualitätsprobleme werden behandelt, etwa das von Menschen verfasste Freitextfelder (etwa Produktbewertungen) bewusste Lügen enthalten können.

4 Data Wrangling

Data Wrangling bezeichnet den iterativen Prozess, Daten in eine Form zu bringen, die für weitere Analysen geeignet ist. Dabei ist Data Wrangling ein sehr aufwendig und kostenintensiv. Laut Schätzungen wird dafür bis zu 80% der Entwicklungszeit aufgewendet. [16]

In diesem Kapitel soll ein Überblick über das Thema gegeben werden. Hierzu wird in Abschnitt 4.1 der Data Wrangling Prozess näher beschrieben. Abschnitt 4.2 diskutiert typische Datenqualitätsprobleme wie etwa fehlende Werte. Abschnitt 4.3 gibt einen kurzen Überblick über in der Literatur diskutierte Ansätze. Dieses Kapitel dient dabei als theoretische Grundlage für das Konzept des Jupyter Notebooks in Kapitel 5.

4.1 Data Wrangling Prozess

Data Wrangling ist ein aufwendiger und iterativer Prozess. Abbildung 3 zeigt eine Visualisierung des typischen Data Wrangling Prozesses. Ziel ist es, eine Menge roher Daten im Originalzustand so aufzubereiten dass sie für weitergehende Analyseschritte nutzbar sind. Erst dadurch lassen sich Erkenntnisse erzielen, die möglicherweise einen Mehrwert bringen. Dabei ist zu beachten, dass Data Wrangling und die anschließende Analyse zwar als getrennte Schritte betrachtet werden können. Allerdings lassen sie sich in der Praxis oft nicht scharf trennen und bilden einen wechselseitigen, iterativen Prozess. Dieser Vorgang ist in Abbildung 3 durch den zickzackförmigen Verlauf der Linie symbolisch dargestellt. So ist es möglich, dass beim Laden der Daten in das Analysetool festgestellt wird, dass die Daten nicht im richtigen Format für dieses Tool vorliegen, obwohl bereits mehrere Transformations- und Bereinigungsschritte erfolgt sind. Nachdem weitere Schritte durchgeführt wurden, können bei einem erneuten Analyseversuch weitere Probleme aufgedeckt werden. Zum Beispiel können nicht plausibel erscheinende Ergebnisse auf Datenqualitätsprobleme hindeuten. Diese müssen erst aufgespürt und behoben werden, bevor die Analyse fortgesetzt werden kann. [16]

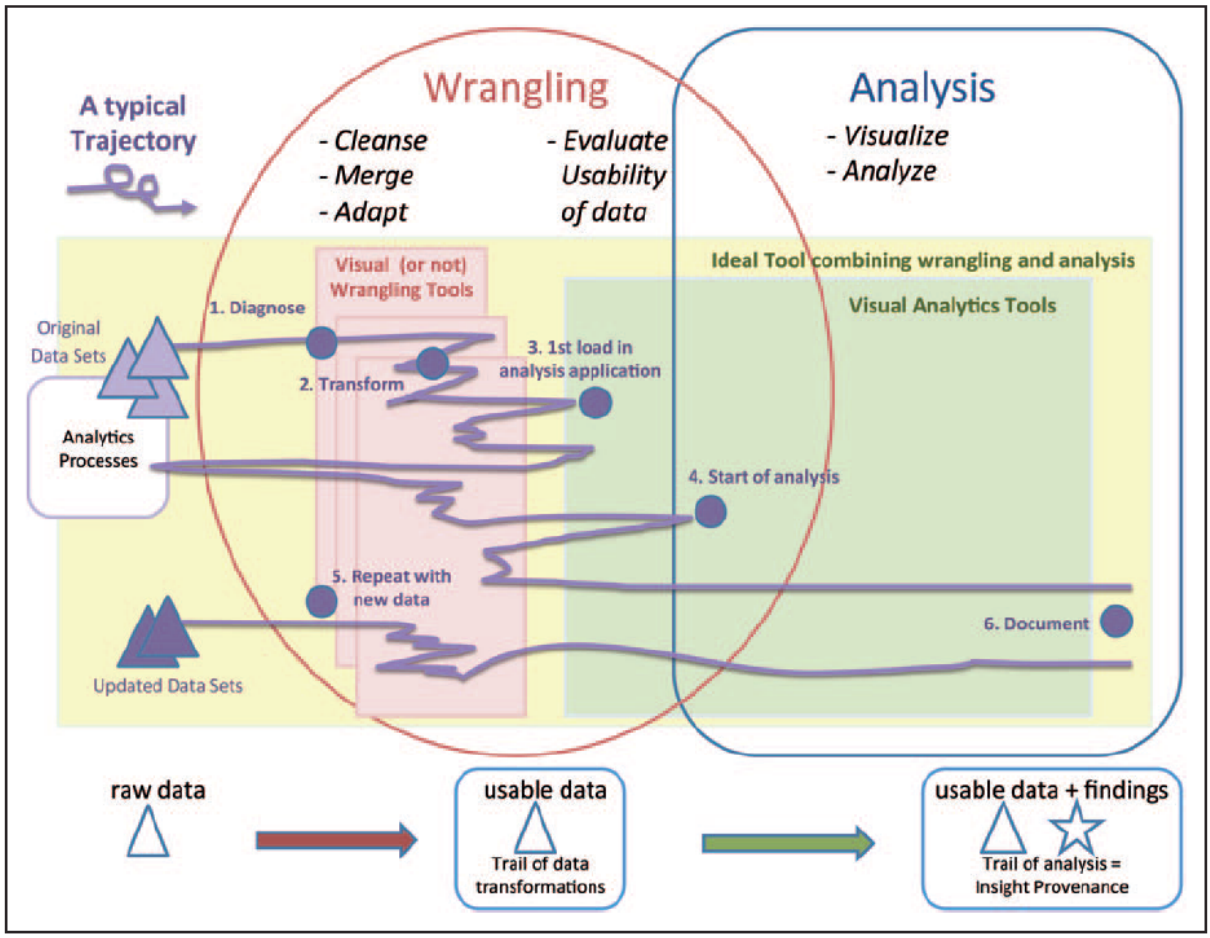

Abbildung 3: Visualisierung eines typischen Data Wrangling Prozesses. Entnommen aus [16]

Im Folgenden sollen die beiden wichtigen Aspekte des Data Wranglings etwas näher beschrieben werden, die Diagnose von möglichen Fehlern, und die Transformation der Daten in eine nutzbare Form. Bei der Diagnose geht es darum alle möglichen Probleme in den Daten zu identifizieren. Dabei gibt es verschiedene Arten von Problemen. Ein typisches Beispiel sind falsche oder fehlende Werte, verursacht etwa durch menschliche Fehler bei der Eingabe. Aber auch formal korrekte, aber irreführende Werte können die Analyse stark beeinträchtigen: So können bestimmte Informationen durch Codes dargestellt werden, etwa der Wert 999 für „unbekannt“ beim Alter. Da diese Werte aber entsprechend dem Datentyp zulässig sind, werden sie nicht direkt erkannt und führen zu falschen Ergebnissen bei der Analyse, etwa dem Durchschnittsalter. Ein weiteres Problemfeld ist das Zusammenführen von Daten aus unterschiedlichen Quellen. Diese können in unterschiedlichen Formaten vorliegen. Daneben können auch unterschiedliche Konventionen zu Datenqualitätsproblemen führen, etwa die Codierung der Altersangabe „unbekannt“ durch den Wert 0 statt wie im Beispiel zuvor durch 999. Eine weitergehende Diskussion möglicher Probleme findet sich in Sektion 4.2. Nachdem Datenqualitätsprobleme diagnostiziert wurden müssen diese behoben werden. Dies geschieht im Allgemeinen durch Transformation der Daten. Ähnlich wie beim gesamten Data Wrangling Prozess sollten diese beiden Schritte nicht strikt linear verstanden werden sondern wechselseitig und iterativ. Bei einer Transformation können zusätzliche Probleme aufgedeckt werden, die wiederum weitere Transformationsschritte auslösen und so weiter. Typische Trans-

formationsschritte sind dabei das (Um-)Formatieren der Daten, die Extraktion, die Umwandlung und die Korrektur fehlerhafter Werte. [16]

Beim Umformatieren der Daten geht es darum, den Datensatz in eine Form zu bringen, die für die weiteren Analysen nutzbar ist. Ein Beispiel hierfür wäre ein Datensatz, der in einer für die Verwendung durch Menschen gedachten Tabelle im CSV-Format bereitgestellt wird. Die Einbettung von wichtigen Informationen in Text, etwa den Namen des Bundesstaates bei einer Kriminalstatistik für die USA, stellt für Menschen kein Problem dar, ist aber nicht unmittelbar von Tools zu verarbeiten und muss umgewandelt werden. Tabelle 1 stellt ein vereinfachtes Beispiel dar. Auf der linken Seite ist das ursprüngliche Format zu sehen. Auf der rechten Seite das Zielformat, das leichter durch Tools zu verarbeiten ist. Diese Tabelle stellt auch gleichzeitig ein Beispiel für eine Extraktion dar. Die Information, auf welchen Bundesstaat sich die Werte beziehen, ist im Originaldatensatz nur in Form eines Textes vorhanden, der nicht ohne weiteres durch Tools verarbeitet werden kann. Hierzu muss der Name, im Beispiel „Alabama“, aus dem Text extrahiert und in einer zusätzlichen Spalte gespeichert werden. [17]

Ursprüngliches Format:		Zielformat:		
gemeldete Verbrechen für Alabama:	Eigentumsdelikte	Jahr	Staat	Eigentumsdelikte
2004	4000	2004	Alabama	4000
2005	5000	2005	Alabama	5000
2006	3900	2006	Alabama	3900

Tabelle 1: Beispiel für Umformatierung und Extraktion. Basierend auf [17]

Ein Beispiel für einen Umwandlungsschritt ist die Umrechnung von einer Maßeinheit in eine andere, etwa von Meilen zu Kilometern. Viele Umwandlungsschritte benötigen zusätzliche Informationen, beispielsweise zur Umrechnung von Maßeinheiten und zur Umrechnung von einer Währung in die andere. Hierfür muss der Wechselkurs bekannt sein. Zusätzlich muss festgelegt werden, zu welchem Zeitpunkt der Wechselkurs ermittelt wird. [16]

Typisch sind aber auch technisch bedingte Umwandlungen. Beispielsweise werden Zahlen häufig als Zeichenkette repräsentiert (String). Bevor diese Werte mathematisch verarbeitet werden können müssen sie zuerst in eine Zahlenrepräsentation umgewandelt werden (z.B. Integer, Float, etc.). [22]

4.2 Typische Datenqualitätsprobleme

In dieser Sektion soll ein kurzer Überblick über ausgewählte Datenqualitätsprobleme im Kontext von Data Wrangling gegeben werden. Die Darstellung ist keinesfalls vollständig oder erschöpfend. Hierbei wird der Begriff der Datenqualität im Sinne von „fitness for use“ benutzt [29]. Ein Datensatz, der inhaltlich und fachlich einwandfrei ist, aber im falschen Format für das genutzte Analysetool vorliegt, wird als Datenqualitätsproblem (im Kontext des Analysetools) aufgefasst [18].

Tabelle 2 zeigt eine Übersicht ausgewählter Datenqualitätsprobleme im Kontext von Data Wrangling. Die Spalte „mögliche Lösung“ enthält beispielhaft mögliche Lösungsansätze. Diese Lösungsansätze sind keinesfalls allgemeingültig oder erschöpfend. Die Spalte Referenz enthält den Verweis für die entsprechenden Literaturquelle, auf denen der Eintrag basiert. Der Verweis dient dabei als Referenz für die Herkunft und bedeutet nicht, dass dieses Problem ausschließlich nur in dieser Quelle diskutiert wird. Andere, auch hier zitierte Quellen können Aspekte dieses Problems beleuchten.

Die Tabelle basiert auf einer Zusammenstellung von in der Literatur diskutierten Fällen. Die Auswahl erfolgte aus folgenden Gründen: Für Menschen gedachte Formate, die schwer maschinell verarbeitet werden können sind ein Beispiel für die Notwendigkeit von Data Wrangling, selbst wenn der Datensatz an sich keine fachlichen Fehler enthält. Fehlende und irreführende Werte sowie inkonsistente Semantik sind typische Beispiele die häufig in der Literatur zitiert werden. Fehlende und irreführende Werte werden auch im praktischen Teil dieser Diplomarbeit diskutiert. Siehe hierzu das Kapitel 8. Bewusste Täuschung als mögliche Quelle von Qualitätsproblemen kann je nach Domäne eine große Rolle spielen, etwa bei Online Daten wie Bewertungen, und sollte nicht unterschätzt werden. Das Encoding von Textdaten spielt bei der Analyse von Texten eine Rolle und kann zu Fehlern führen, die schwer zu entdecken sind. In dieser Arbeit wurden mehrere Metriken verwendet, die auf Textdaten arbeiten, siehe hierzu Kapitel 6. Im folgenden werden die einzelnen Qualitätsprobleme näher beschrieben, mit den dazu gehörenden Referenzen auf die Literaturquellen.

Datenqualitätsproblem:	mögliche Lösung:	Referenz
Für Menschen gedachtes Format	Umformatieren	17,22
Fehlende Werte	Imputieren, Filtern	16,22
Irreführende Werte, Codes	konvertieren	16,22
Bewusste Täuschung	Analysieren	22
Inkonsistente Semantik	Mapping	16
Encoding von Textdaten	konvertieren	22

Tabelle 2: Ausgewählte Datenqualitätsprobleme im Kontext von Data Wrangling mit möglichen Lösungsschritten

Häufig werden Datensätze in einem Format bereitgestellt, das **für die Benutzung durch Menschen gedacht** ist, beispielsweise formatierte Tabellen. Dies stellt ein Problem dar, falls die Daten maschinell verarbeitet werden sollen. [22]

Ein einfaches Beispiel hierfür zeigt Tabelle 1, basierend auf einem ausführlicheren Beispiel in [17]. Diese Tabelle zeigt (fiktive) Informationen über Verbrechen pro Jahr und Bundesstaat der USA. Auf welchen Bundesstaat sich die Informationen beziehen wird im Originaldatensatz durch einen Text beschrieben. Für Menschen stellt dies kein Problem dar, aber der einfache Text „gemeldete Verbrechen für [Bundesstaat]“ ist nicht ohne weiteres automatisch verarbeitbar. Die relevante Information, also der Name des Bundesstaates, muss hierfür extrahiert werden. Zusätzlich ist das Format zu beachten. Menschen können erkennen, dass sich der Text mit der Informationen über den Bundesstaat sich auf die folgenden drei Jahre 2006, 2007 und 2008 bezieht. Für automatische Verarbeitung ist es besser, den Namen des Bundesstaates zu jeder Zeile hinzuzufügen. [17]

Falls die Daten in einem Tabellenformat wie etwa CSV bereitgestellt werden, so ist es noch relativ einfach möglich, sie durch entsprechende Transformationen in ein für maschinelle Verarbeitung geeignetes Format zu konvertieren. Deutlich schwieriger wird dies etwa bei Daten, die in PDF Dokumenten oder Websites eingebunden sind. [22]

Fehlende Werte in Datensätzen können verschiedene Ursachen haben, etwa menschliche Fehler bei der Eingabe. Bei Datensätzen, die über längere Zeiträume erhoben werden, können bestimmte Spalten erst nachträglich in den Datensatz mit aufgenommen worden sein. Für ältere Werte wurden diese Daten nicht erhoben und stehen auch nicht zur Verfügung. Es ist auch möglich, das Daten ganz oder teilweise verlorengegangen sind. Eine Katastrophe, etwa ein Feuer in einem Archiv, kann bei historischen Daten zu Lücken im Datensatz oder zu einem Totalverlust bestimmter Werte führen. Aber auch technische Ursachen können eine Rolle spielen. Beispielsweise kann eine Schnittstelle, über die Daten abgerufen werden, auf eine bestimmte Anzahl an zurückgegeben Werten limitiert sein. [16]

Es gibt verschiedene Ansätze mit fehlenden Werten umzugehen. Die in Tabelle 2 aufgeführten Vorgehensweisen filtern und imputieren sind nicht die einzigen und wurden beispielhaft ausgewählt. Filtern bedeutet dabei Einträge zu entfernen, die bestimmten Kriterien entsprechen. In diesem Fall alle Einträge mit fehlenden Werten. In manchen Fällen ist dies die einzige Möglichkeit, mit fehlenden Werten umzugehen, etwa wenn die Werte nicht aus anderen Datenquellen ergänzt werden können. Es besteht auch die Möglichkeit, fehlende Werte zu imputieren. Imputieren bedeutet dabei, einen fehlenden Wert für einen Eintrag durch den entsprechend Wert eines ähnlichen Eintrags zu ersetzen. Allerdings kann dieses Vorgehen unter bestimmten Umständen die Qualität der Analyse verschlechtern und sollte nur mit Bedacht gewählt werden. [22]

Irreführende Werte können eine Analyse ernsthaft beeinträchtigen. Beispielsweise kann der Wert 999 bei einer Altersangabe bedeuten, dass das Alter unbekannt ist. Handelt es sich bei dem Datentyp des Feldes um „Zahl“, so ist 999 allerdings ein gültiger Wert. Einem Analysetool fehlt die semantische Information, dass 999 für einen Menschen kein gültiges Alter darstellt. Dadurch können Berechnungen auf den Daten zu falschen Ergebnissen führen, etwa die Berechnung des durchschnittlichen Alters. Eine Möglichkeit damit umzugehen, ist es diesen Wert zu konvertieren, etwa in einen expliziten Nullwert. [16]

Es ist möglich, dass Datenqualitätsprobleme in einem Datensatz durch **bewusste Täuschung** entstanden sind, beispielsweise bei Online-Bewertungen. Dabei kann es sich um fiktive Scherzeinträge handeln. Auch bewusste Manipulationen sind denkbar. Ein in Onlinebewer-

tungen tatsächlich vorkommendes Beispiel wäre etwa, wenn ein Nutzer einem von ihm eigentlich geschätzten Lokal eine schlechte Bewertung gibt, um andere davon abzuhalten, es zu besuchen und so eine Überfüllung zu verhindern. Solche Qualitätsprobleme lassen sich ohne aufwendige Analysen nicht leicht beziehungsweise möglicherweise gar nicht entdecken. [22]

Siehe hierzu auch [15].

Inkonsistente Semantik kann zum Beispiel dadurch entstehen, dass sich ein Klassifikationsschema im Laufe der Zeit verändert. Ändert sich beispielsweise zu einem Stichtag die Klassifikation für ein Feld in einem Datensatz, etwa die Todesursache in einer Kriminalstatistik, so kann dies zu Inkonsistenzen führen. Einige Werte können ab dem Stichtag eine andere Bedeutung haben als exakt die gleichen Werte bei Einträgen vor dem Stichtag. Manche Werte werden nicht mehr verwendet, neue Bezeichnungen tauchen auf die es im Datensatz zuvor nicht gab. Probleme mit inkonsistenter Semantik treffen auch häufig auf, wenn Datensätze aus unterschiedlichen Quellen integriert werden sollen, die auf unterschiedlichen Schemata und Definitionen basieren, etwa weil sie aus unterschiedlichen Organisationen erstellt wurden oder aus unterschiedlichen Ländern stammen. Eine Möglichkeit damit umzugehen ist eine semantische Transformation, ein Abbilden von Konzepten aufeinander, die eine vergleichbare Bedeutung haben. Dies kann zum Beispiel durch eine Mappingfunktion realisiert werden, die inkompatible Werte durch die jeweils passenden Werte im Zielschema ersetzt. In manchen Fällen sind die Unterschiede zwischen den Schemata aber so groß, dass keine korrekte Transformation durchgeführt werden kann. [16]

Falsches Encoding von Textdaten kann die Verarbeitung dieser Daten erschweren. In manchen Fällen ist es möglich, dass das fälschlicherweise gewählte Encoding sehr ähnlich zum tatsächlich vom Textdatensatz verwendeten Encoding ist. Dies kann dazu führen, dass Fehler entstehen die nur sehr schwer zu entdecken sind, weil bei oberflächlicher Inspektion der Textdaten keine offensichtlichen Probleme auffallen. Der Begriff Encoding oder Zeichenkodierung bezeichnet das technische Format, in dem Textdaten digital gespeichert werden. ASCII ist ein 7-bit Encoding. Das bedeutet, dass jedes Textzeichen durch eine Siebenstellige Binärzahl dargestellt wird, mit insgesamt 128 möglichen Zeichen, die durch das Format repräsentiert werden können. Der Zeichensatz umfasst unter anderem alle Buchstaben des englischen Alphabets in Groß- und Kleinschreibung, die Zahlen von 0 bis 9 sowie diverse Satzzeichen und mathematischen Symbole. ASCII ist aus historischen Gründen sehr weit verbreitet und spielt heute noch aus Kompatibilitätsgründen eine große Rolle. Viele in westeuropäischen Sprachen verwendete Symbole können nicht mit ASCII dargestellt werden, für die deutsche Sprache zum Beispiel die Umlaute. Es gibt viele andere Encoding-Schemata, die teilweise inkompatibel zu einander sind. Unicode ist ein universelles Encoding mit dem Ziel, alle Zeichen zu repräsentieren, die in naher Vergangenheit auf der Welt im Druck verwendet wurden. Unicode unterstützt Sprachen aus der ganzen Welt, darunter auch Sprachen mit einem komplexen Zeichensatz wie Chinesisch. Ein typisches Problem, dass durch die Verwendung unterschiedlicher Encodings entstehen kann, ist dass Werte nicht richtig verglichen werden können. Zwar stellen sie den gleichen Text dar und haben somit die gleiche Semantik. Aber wegen der unterschiedlichen Encodings werden sie technisch unterschiedlich repräsentiert. Dadurch werden sie etwa bei Vergleichen als unterschiedlich gewertet, was zum Beispiel bei JOIN Operationen zu Problem führen kann. Eine mögliche Lösung ist es, die Encodings vorliegende Textdaten zu vereinheitlichen. Hierzu werden alle Textdaten in ein einheitliches Standardformat

konvertiert, zum Beispiel UTF-8, eine Variante von Unicode. Dabei ist zu beachten, dass das Encoding der Ausgangsdaten korrekt angegeben wird. So sind die Encodings Code Page 1252 und ISO-8859-1 sehr ähnlich, unterscheiden sich aber in Details. Die Angabe des falschen Encodings könnte unter Umständen zu schwer diagnostizierbaren Datenqualitätsproblemen führen. [22]

Viele der in diesem Abschnitt diskutierten Aspekte sind beim Entwurf des Notebooks berücksichtigt worden. Siehe hierzu das Kapitel 5 über das Konzept. Im Folgenden soll dies noch etwas ausführlicher dargestellt und begründet werden:

Für Menschen gedachte Formate werden nicht direkt berücksichtigt. Dies liegt daran, dass es eine Vielzahl verschiedener Varianten und Darstellungsformen gibt. Eine automatische Lösung ist also extrem aufwendig und schwer zu verallgemeinern. Bei kleinen Dokumenten, etwa Tabellen die aus wenigen Seiten bestehen, kann eine manuelle Bearbeitung durch ein hierfür geeignetes Tool die einfachste Alternative darstellen. Beispiele für solche Tools sind Tabellenkalkulationsprogramme wie etwa OpenOffice Calc [42] oder spezialisierte Tools wie Data Wrangler [17]. Alternativ kann dieses Problem durch Code gelöst werden, wie im Bad Data Handbook erwähnt [22]. Soll eine Transformation zum Beispiel wiederholbar sein, etwa wenn der Datensatz regelmäßig aktualisiert wird, so kann eine automatische Transformation durch ein Skript realisiert werden. Dieses kann dann problemlos ins Notebook eingebunden werden. Im Notebook sind diverse Skripte und Bibliotheken integriert. Verwendet werden unter anderem die Programmiersprachen Java, Python und R. Die Implementierung und Einbindung kann als Starthilfe dienen, eigene Skripte einzubinden. Siehe hierzu auch das Kapitel 6.

Fehlende Werte spielen eine große Rolle und werden daher beim Entwurf des Notebooks berücksichtigt. Hierzu steht eine Metrik bereit, welche die Verteilung der Nullwerte im Datensatz berechnet. Zusätzlich ist es möglich, Zeilen mit Nullwerten zu filtern. Eine Imputation wird nicht angeboten. Zum einen ist die Implementierung aufwendig. Zum anderen können dadurch zusätzliche Datenqualitätsprobleme entstehen [22].

Beispiele für fehlende Werte in einem Datensatz bietet die Analyse des NHTSA Complaints Datensatzes in Kapitel 8.1. Auch bei der Betrachtung der Industriedaten in Kapitel 8.2 wird dieses Problem diskutiert.

Irreführende Werte spielen ebenfalls eine große Rolle und fließen ins Konzept ein. Das Notebook unterstützt durch Visualisierungen, etwa eines Histogramms, beim aufspüren möglicher Kandidaten. Zusätzlich wird eine Metrik zur Erkennung von Ausreißern beispielhaft angeboten. Werden irreführende Werte entdeckt, ist es durch die Ersetzungsfunktion möglich, sie zu ändern. Beispielsweise können sie durch explizite Nullwerte ersetzt werden. Siehe hierzu Kapitel 6.2.

Siehe die Analyse des NHTSA Complaints Datensatzes in Kapitel 8.1 für ein praktisches Beispiel. Die Spalte für das Modelljahr (YEARTXT) enthält Einträge mit dem Wert 9999. Dieser hat die Bedeutung, das der Wert unbekannt ist oder für den Eintrag nicht verfügbar ist.

Bewusste Täuschung ist ein sehr komplexes Problem. Das Notebook bietet hierfür keine Metriken. Es wird bewusst auf eine vereinfachte, beispielhafte Implementierung oder Einbindung verzichtet. Inkorrekte oder falsch interpretierte Ergebnisse könnten, je nach Domäne, zu ernst-

haften Konsequenzen führen. Die Entwicklung einer passenden Methode übersteigt bei weitem den Umfang der Arbeit.

Inkonsistente Semantik ist zwar ein wichtiger Aspekt, aber es ist schwer, automatische Lösungen zu entwerfen. Fachwissen über die Domäne und die Semantik ist meistens notwendig. Allerdings ist es durch die Funktionen des Notebooks möglich, den Datensatz zu untersuchen und zu verändern. Dies kann bei der Analyse und Behebung helfen. Das Notebook bietet keine Implementierung einer Mappingfunktion. Pandas [46], die Bibliothek, die als Grundlage für die Datenverarbeitung im Notebook dient, unterstützt Mappingfunktionen. So können diese relativ einfach entwickelt und direkt im Notebook verwendet werden.

Falsches Encoding von Textdaten spielt in dieser Arbeit eine wichtige Rolle. Es werden drei Metriken angeboten, mit denen die Qualität von Textdaten gemessen werden kann. Siehe hierzu Abschnitt 5.2.3. Damit diese korrekt funktionieren, muss das richtige Encoding verwendet werden. Das Encoding kann beim Import ausgewählt werden. Ein Textfeld fasst Aspekte dieses Themas kurz zusammen und nennt gängige Encodings, basierend auf dem Bad Data Handbook [22]. Die Checkliste enthält einen Punkt, der auf dieses Thema hinweist. Das Notebook erlaubt das Konvertieren von Encodings. Hierzu muss beim Speichern des Datensatz nur das gewünschte Encoding ausgewählt werden.

Siehe Kapitel 8 für praktische Beispiele.

4.3 Mögliche Ansätze

In diesem Abschnitt soll eine Auswahl von Data Wrangling Ansätzen aus der Literatur kurz vorgestellt werden. Diese Ansätze wurden exemplarisch ausgewählt.

Programmcode spielt im Data Wrangling nach wie vor eine große Rolle. Kandel et al. berichten, das Skripte in verschiedenen Programmiersprachen wie Python oder R eine große Rolle beim Data Wrangling spielen. Allerdings wird dies eher als Hindernis gesehen. Viele Menschen werden dadurch vom Umgang mit Daten abgehalten, weil ihnen die technische Expertise fehle. [16]

Das Bad Data Handbook [22] empfiehlt Code als flexible Möglichkeit zum Umgang mit Daten. So ist man unabhängig von Format der Daten. Das in Abschnitt 4.2 erwähnte Problem, dass Daten häufig in Formaten bereit gestellt werden, die für Menschen gedacht sind und maschinell schwer zu verarbeiten sind, bietet hierfür ein Beispiel. Falls ein Datensatz in einem ungewöhnlichen Format bereitgestellt wird, für das kein Standard-Tool vorhanden ist, so kann durch Programmieren ein Tool geschaffen werden, das die Umwandlung automatisch durchführt. [22]

Datenqualität spielt eine entscheidende Rolle beim Data Wrangling. Das Tool Data Wrangler implementiert einen Datenqualitätsmeter, einen Balken der einfache Datenqualitätsmetriken visualisiert. Ein Beispiel wäre der Prozentsatz der fehlenden Werte in einer Spalte, der während der Analyse ständig berechnet wird. Wird eine Transformation ausgeführt, wird der Wert aktualisiert. So kann der Anwender die Qualität des Datensatzes überblicken und die Auswirkungen der Änderungen verfolgen. [17]

Für eine etwas ausführlichere Beschreibung des Tools siehe Kapitel 3 über verwandte Arbeiten.

Visualisierungen können dabei helfen, Datenqualitätsprobleme in einem Datensatz zu entdecken. Typische Beispiele sind fehlende Werte und Ausreißer. Beispielsweise lassen sich Ausreißer in einem Diagramm oft als auffällige Extremwerte erkennen. Lücken im Schaubild deuten auf fehlende Werte hin. Dabei ist zu beachten, dass die Art der Darstellung einen großen Einfuß hat. Manche Datenqualitätsprobleme lassen sich gut durch Diagramme oder andere spezielle Darstellungen erkennen. Andere können nur durch Inspektion der Rohdaten erkannt werden. Eine guter Startpunkt ist die Visualisierung der Daten als einfache Textdarstellung in Tabellenform. Ein Histogramm ist eine weitergehende Darstellung. Hierbei wird dargestellt, wie oft ein Wert in einer Spalte vorkommt. [16]

Tabelle 3 enthält ein einfaches Beispiel für ein Histogramm. Der Datensatz (links) enthält die Werte A, B und C. Das Histogramm rechts enthält die Häufigkeiten für diese Spalte.

Datensatz	Histogram	
Wert	Wert	Anzahl
A	A	3
B	C	2
A	B	1
C		
C		
A		

Tabelle 3: Beispiel für ein Histogramm

Der Umgang mit „schmutzigen“ Daten, also Daten mit Qualitätsproblemen, ist ein wichtiges Thema. Es ist nicht immer möglich diagnostizierte Datenqualitätsprobleme zu beseitigen. Fehlende Werte können durch Verlust der Originaldaten bedingt sind, bei historischen Daten etwa durch einen Brand in einem Archiv. Dadurch sind die Daten unter Umständen verloren und können nicht wieder hergestellt werden. In anderen Fällen könnte die Beseitigung von Qualitätsproblemen einen unvertretbar hohen Aufwand erfordern. In diesen Fällen können Visualisierungen dabei helfen, die Tatsache, dass Werte fehlen, zu kommunizieren. Abbildung 4 zeigt an einem Beispiel 4 mögliche Visualisierungen für fehlende Daten. Das Schaubild zeigt Zensusdaten aus den USA. Dargestellt ist die Anzahl der Menschen, die zum jeweiligen Zeitpunkt als Landarbeiter tätig waren. Daten von 1890 sind bei einem Brand verlorengegangen. Bei Variante 1 werden die fehlenden Daten als ein Wert von 0 interpretiert. Bei Variante 2 wird das Fehlen des Wertes ignoriert. Die Linie im Schaubild stellt eine Interpolation dar. Bei Variante 3 wird der fehlende Wert explizit aus dem Schaubild entfernt. Bei Variante 4 wird der fehlende Wert interpoliert, allerdings ist dies durch eine andere Farbe explizit dargestellt. Es ist noch nicht klar, welche Formen der Darstellung optimal sind. Studien zeigen, dass Menschen nicht immer erkennen, wenn fehlende Werte durch Standardwerte ersetzt werden. [16]

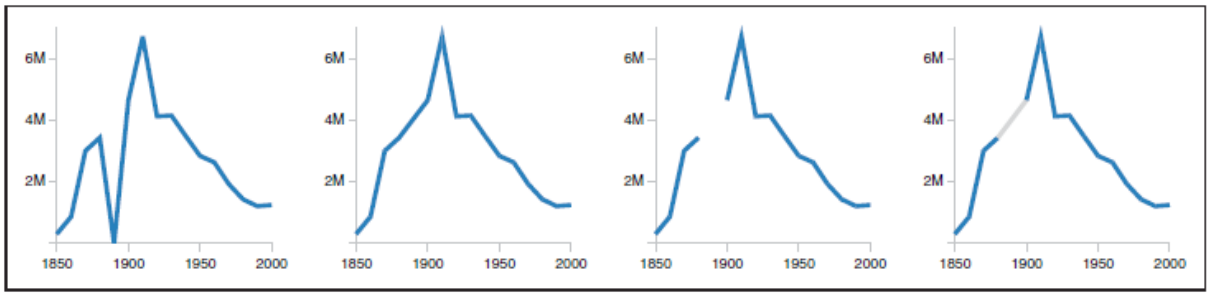

Abbildung 4: Vier mögliche Visualisierungen von fehlenden Werten. Entnommen aus [16]

Einige der hier diskutierten Aspekte wurden beim Entwurf des Notebooks berücksichtigt und spielen sich in der Umsetzung wieder. Im folgenden soll dies kurz dargestellt und begründet werden.

Code ist ein zentrales Element des Notebooks. Dadurch ist es möglich, flexible Lösungen bereitzustellen, die je nach Anwendungsfall und Domäne abgewandelt werden können. Siehe hierzu Kapitel 5.2.

Die Datenqualität ist ebenfalls ein zentraler Aspekt des Prototypen. Hierzu werden verschiedene Metriken bereitgestellt. Dabei werden sowohl strukturierte als auch unstrukturierte Daten berücksichtigt. Siehe hierzu Abschnitt 5.2.3.

Eine Anwendung der Metriken auf Datensätze aus der Praxis mit Berechnung konkreter Ergebnisse findet sich in Kapitel 8.

Ein Beispiel, dafür wie ein Diagramm dabei helfen kann, irreführende Werte zu erkennen, findet sich in der Analyse der NHTSA Complaints Daten in Sektion 8.1. Das Notebook bietet daher beispielhaft einfache Visualisierungen wie Diagramme und Histogramme. Siehe Kapitel 5.2.2. Komplexere Visualisierungen können nach Bedarf eingebunden werden.

Der Umgang mit „schmutzigen“ Daten ist ein wichtiger Aspekt. Wie in der Diskussion zuvor beschrieben können in manchen Fällen Qualitätsprobleme nicht gelöst werden, etwa wenn Daten unwiederbringlich verlorengegangen sind. Entsprechende Funktionen des Notebooks, die dabei helfen, diese Tatsachen zu kommunizieren, wären wünschenswert. Beispielsweise Visualisierungsvarianten, wie sie in Abbildung 4 dargestellt werden. Dies könnte dazu beitragen, Missverständnisse und Fehlinterpretationen bei der weiteren Verwendung der Daten zu reduzieren. Diese können aber im Rahmen dieser Arbeit nicht implementiert werden.

5 Konzept des Jupyter Notebooks

In diesem Kapitel soll das Konzept für eine prototypische Implementierung eines Jupyter Notebooks vorgestellt werden. Das Konzept berücksichtigt einige der in Kapitel 4 diskutierten Aspekte des Data Wranglings. Für Details zur Implementierung siehe Kapitel 6. Eine Anwendung des Notebooks auf Datensätze aus der Praxis findet sich in Kapitel 8.

Zunächst soll in Abschnitt 5.1 ein Überblick gegeben werden. Danach wird in Abschnitt 5.2 genauer auf den Aufbau eingegangen. Beim Entwurf des Notebooks sind Ideen und Ansätze aus verschiedenen Literaturquellen entnommen worden. Die entsprechenden Referenzen befinden sich in der Beschreibung des jeweiligen Abschnitts.

5.1 Überblick

Die Grundidee des Notebooks ist es ein flexibles Toolkit, das beim Data Wrangling unterstützt. Es ist als Prototyp gedacht, nicht als fertiges Produkt. Das Notebook ist nicht für einen bestimmten Anwendungsfall oder eine spezielle Domäne bestimmt. Vielmehr soll es als Startpunkt dienen. Für eine konkrete Analyse kann es angepasst und erweitert werden. Viele Metriken und Funktionen wurden nicht in dieser Arbeit entwickelt, sondern von anderen Quellen übernommen und eingebunden. Ein Literaturverzeichnis im Notebook soll diese Tatsache dokumentieren. Zusätzlich soll dadurch möglich sein, die originale Arbeit schnell aufzufinden. Dies kann hilfreich sein, wenn für die eigene Problemstellung Funktionen angepasst werden müssen. Außerdem kann dies auch als Startpunkt für eine Literaturrecherche dienen.

Beim Entwurf sollen die folgenden Punkte berücksichtigt werden:

In Kapitel 4.1 werden die Diagnose und die Datentransformation als wichtige Aspekte des Data Wrangling Prozesses beschrieben.

Bei der Diagnose sollen mögliche Fehler und Qualitätsprobleme im Datensatz aufgespürt werden. Das Notebook soll hierzu mehrere Funktionen bieten, mit denen die Daten betrachtet werden können. Es ist möglich, die Daten nach verschiedenen Kriterien zu filtern. Zusätzlich werden auch Visualisierungen angeboten. Das Notebook bietet exemplarisch mehrere Datentransformationen. Damit ist es möglich, die Daten zu bearbeiten und so mögliche Datenqualitätsprobleme zu beseitigen. Die Transformationsschritte werden nicht direkt auf den Originaldaten durchgeführt. Das Notebook lädt beim Import eine Kopie in den Hauptspeicher. Damit die Änderungen persistiert werden, müssen diese gespeichert werden. Hierzu steht unter **Save Data** eine Funktion bereit. Es soll technisch möglich sein, das Original zu überschreiben, allerdings muss dies explizit angegeben werden. Dies wird aber nicht empfohlen. Fehlerhafte Änderungen können so nicht mehr rückgängig gemacht werden, wodurch die Datenqualität negativ beeinträchtigt werden kann. Diese Empfehlung basiert auf der Funktion des Tools Data Wranglers, durchgeführte Transformationen rückgängig machen zu können [17]. Eine solche Funktion wäre auch für das Notebook wünschenswert, übersteigt aber wegen des damit verbundenen Aufwands den Umfang der Arbeit. Deswegen wird sie nicht in den Prototyp aufgenommen. Eine umständlichere, aber einfache Lösung ist es, Ergebnisse zwischenzuspeichern und bei bedarf auf eine ältere Version oder das Original zurückzugreifen.

Datenqualität spielt im Data Wrangling eine entscheidende Rolle, siehe hierzu Kapitel 4.3. Dies wird beim Entwurf des Notebooks berücksichtigt. Das Notebook soll Metriken sowohl für strukturierte als auch unstrukturierte Daten bieten, mit denen die Datenqualität gemessen werden kann. Hierzu zählen neben einfacheren Metriken wie der Verteilung von Nullwerten auch komplexere Berechnungen wie die Textähnlichkeit. Diese unterstützen bei der Diagnose von Qualitätsproblemen. Somit ist es möglich, den Ausgangszustand der Daten einschätzen zu können. Zusätzlich können so die Auswirkungen der Transformationsschritte auf die Qualität besser beurteilt werden.

Die Realisierung erfolgt durch einen Wechsel von Code und Texten. Diese Texte dienen als Dokumentation des Codes und bieten zusätzlich Hilfestellung beim Data Wrangling. So wird zum Beispiel eine Checkliste angeboten, damit wichtige Aspekte nicht untergehen. Diese basiert auf Kapitel 4.2. Als Sprache für die Texte wurde Englisch gewählt. Dies soll die Verwendung des Notebooks einem weiteren Personenkreis zugänglich machen.

5.2 Aufbau

Das Notebook soll in die folgenden Abschnitte gegliedert werden:

- **Checklist**
- **Import Data**
- **View Data**
- **Metrics**
- **Transform Data**
- **Save Data**

Diese Gliederung dient der Übersichtlichkeit. Hiermit sollen die verschiedenen Funktionen gruppiert werden, damit sie leichter zu finden sind. Die Gliederung versteht sich nicht als linearer Prozess, bei dem die einzelnen Punkte nacheinander abgearbeitet werden. Vielmehr handelt es sich um einen Werkzeugkasten, der verschiedene Funktionen bietet. Wie in Kapitel 4.1 beschrieben ist der Data Wrangling Prozess iterativ. Manche Funktionen werden in der Praxis mehrmals ausgeführt, etwa die Inspektion. Beispielsweise um nach einem Transformationsschritt zu überprüfen, ob dieser korrekt durchgeführt wurde. Es ist auch möglich, dass für manche Analysen Funktionen gar nicht verwendet werden. Die Verwendung von Metriken für Textdaten zum Beispiel macht keinen Sinn, wenn der Datensatz keine Textdaten enthält. Dieses Iterative Vorgehen wird durch das verlinkte Inhaltsverzeichnis unterstützt. Dadurch ist es möglich, direkt zum jeweiligen Abschnitt bzw. zur jeweiligen Metrik zu springen. Zusätzlich besteht bei jeder Funktion die Möglichkeit, durch einen Link zurück zum Inhaltsverzeichnis zu gelangen.

Als nächstes folgt eine Checkliste, die typische Arbeitsschritte und Probleme zusammenfasst, die allgemein beim Data Wrangling beachtet werden sollen. Sie soll als möglicher Startpunkt

dienen. Die Checkliste ist keinesfalls vollständig oder allgemeingültig. Auch ist sie nicht als Anleitung oder Handbuch zu verstehen. Beispielsweise spielt bei Textdaten das Encoding eine große Rolle, siehe hierzu Kapitel 4.2. In Kapitel 8.1 wird an einem praktischen Beispiel beschrieben, wie einzelne Anführungszeichen in Freitextfeldern die Importfunktion beeinträchtigen können. Diese Aspekte spielen bei der Verarbeitung von Textdaten eine Rolle. Deshalb wird ein entsprechender Punkt in die Checkliste aufgenommen. Bei Datensätzen ohne Textdaten hingegen wäre dieser Punkt überflüssig. Andere Formen von Daten können ebenfalls typische Probleme haben, die hier aus Gründen des Umfangs nicht berücksichtigt werden können.

5.2.1 Import Data

Damit Daten im Notebook verarbeitet werden können, müssen diese zunächst Importiert werden. Hierfür soll eine Funktion im Abschnitt **Import Data** bereitgestellt werden. Die interne Darstellung des Notebooks soll auf Pandas [46] basieren, einer Python Bibliothek für Datenverarbeitung und Analyse. Der Import soll beispielhaft für Textdateien implementiert werden. Dadurch soll es möglich sein, sowohl Textdaten als auch strukturierte Daten zu verarbeiten. Dazu müssen die strukturierten Daten in einem geeigneten Format vorliegen, beispielsweise CSV. Andere Datenquellen, wie etwa Datenbanken, können bei Bedarf eingebunden werden. Das Notebook ist im allgemeinen nicht dafür gedacht, Daten in ihrer rohen Form zu manipulieren. Liegt beispielsweise eine fehlerhafte CSV Datei vor, so sollte es mit dem Notebook möglich sein, eine Textdarstellung der Datei zu importieren und zu betrachten. Eine Bearbeitung im Notebook wird aber hierfür nicht vorgesehen. Kapitel 8.1 bietet ein Beispiel für ein solches Problem.

5.2.2 View Data

Der Abschnitt **View Data** soll verschiedene Funktionen bieten, mit denen die Daten inspiziert und visualisiert werden können. Siehe hierzu auch Kapitel 4.3.

Als Einstieg dient eine Darstellung der Daten in Tabellenform. Somit kann der Benutzer sich schnell einen Überblick über die Daten verschaffen. Diese Funktion basiert auf einer Erwähnung bei Kandel et al. [16]. Es ist möglich, sich nur einen Ausschnitt der Daten anzeigen zu lassen. Hierzu können Zeilen und Spalten angegeben werden, die dargestellt werden sollen.

Eine weitere Funktion, basierend auf einem Beispiel im Bad Data Handbook [22], ist das Histogramm. Dieses berechnet, wie oft jeder Wert in einer Spalte vorkommt. Dies kann dabei unterstützen, sich einen Überblick über die Daten zu verschaffen. Zusätzlich kann ein Histogramm dabei helfen, Anomalien und mögliche Datenqualitätsprobleme zu entdecken. Das Histogramm kann auch als Schaubild visualisiert werden. Hierzu soll eine entsprechende Funktion bereitstehen.

Eine weitere wichtige Funktion ist das Filtern. Diese ermöglicht, einen Ausschnitt der Daten auszuwählen, der bestimmten Kriterien entspricht. Dies basiert auf Erwähnungen in der Literatur, zum Beispiel im Bad Data Handbook [22]. Beispielsweise lassen sich so Zeilen untersu-

chen, die Nullwerte enthalten. Dadurch ist es möglich, sich eine Detailansicht zu verschaffen, und Datenqualitätsprobleme genauer zu untersuchen.

Für ein praktisches Beispiel dieser Funktionen, siehe die Analyse des NHTSA Complaints Datensatzes in Kapitel 8.1.

Eine einfache Funktion soll es ermöglichen, zu überprüfen, ob im Datensatz Sonderzeichen vorkommen. Dies hilft zum Beispiel dabei, zu prüfen, ob Umlaute oder andere spezielle Zeichen in einem Text vorkommen. Falls ja, kann dadurch auch sichergestellt werden dass sie korrekt dargestellt werden. Damit soll das Arbeiten mit Texten erleichtert werden. Ein falsches Encoding führt oft dazu, das Zeichen falsch dargestellt werden. Siehe hierzu Kapitel 4.2.

5.2.3 Metrics

Im Abschnitt **Metrics** werden sollen diverse Metriken zur Berechnung der Datenqualität angeboten werden. Der Entwurf berücksichtigt mehrere der von in Kandel et al. in [16] und [17] diskutierten Aspekte: Zum einen wird es so möglich Datenqualität während des Data Wrangling Prozesses zu messen. Neben einfachen Metriken wie der Aufschlüsselung der Nullwerte pro Spalte stehen auch komplexere Metriken wie die Textähnlichkeit zur Verfügung. Dabei werden auch unstrukturierte Daten berücksichtigt. Es sollen Metriken angeboten werden, die auf Textdaten arbeiten.

Ein wichtiger Grund bei der Auswahl der Metriken war, die Flexibilität des Notebooks zu demonstrieren. So sollen Metriken integriert werden, die in Python, Java sowie R implementiert sind. Diese Implementierungen können als Startpunkt dienen, um weitere Metriken zum Notebook hinzuzufügen. Kandel et al. [16] diskutieren die Tatsache, dass beim Data Wranging oft eine große Anzahl verschiedener Skripte in verschiedenen Programmiersprachen verwendet wird. Das Notebook bietet die Möglichkeit, diese an einer zentralen Stelle zu bündeln. So können sie leicht angepasst werden. Neue Skripte können problemlos hinzugefügt werden. Dies ermöglicht einen schnelleren Überblick und erleichtert den Einstieg für neue Benutzer. Das soll einen Beitrag dazu darstellen, einem größeren Personenkreis die Verarbeitung von Daten zu ermöglichen. Dies wird ebenfalls von Kandel et al. [16] gefordert.

Siehe Kapitel 6.3 für die Details der Implementierung.

Folgende Metriken sollen in der Standardversion verfügbar sein und sollen im folgenden kurz beschrieben werden. Die Metriken Nullwerte, Validierung und Part-of-Speech-Tagging werden dabei selbst implementiert. Die Metriken Local Outlier Facor, Noisy Data und Textähnlichkeit werden eingebunden.

Nullwerte:

Diese Funktion liefert für jede Spalte des Datensatzes die Anzahl und den prozentualen Anteil der Nullwerte. Dabei werden nur explizite Nullwerte berücksichtigt, also das Fehlen eines Wertes in einer Spalte.

Diese Metrik basiert auf Anregungen aus [26] und soll für das Notebook implementiert werden. Dabei sollen Funktionen von Pandas [46] benutzt werden, die Informationen über Nullwerte berechnen.

Codierte Nullwerte (siehe Kapitel 4.2) können dadurch nicht direkt ermittelt werden. Es ist möglich, diese Werte mit zu berücksichtigen, wenn sie zuvor in explizite Nullwerte konvertiert wurden. Siehe Kapitel 8.1 für ein praktisches Beispiel. Die Implementierungsdetails werden in Abschnitt 6.3.1 beschrieben.

Validierung:

Mit der Validierungsfunktion ist es möglich, zu überprüfen, ob Werte bestimmten Kriterien entsprechen. Beispielsweise kann geprüft werden, ob Datumsfelder auch Werte enthalten, die zulässige Datumsangaben in dem spezifizierten Datumsformat darstellen.

Diese Metrik basiert auf [26]. Diese Metrik soll implementiert werden.

Siehe Abschnitt 6.3.2 für Implementierungsdetails.

Local Outlier Factor:

Diese Metrik hilft beim aufspüren von Ausreißern. Sie basiert auf dem Local Outlier Factor [6]. Kandel et al. [16] erwähnen, das Ausreißern beim diagnostizieren von Datenqualitätsproblemen eine Rolle spielen. Bei dieser Metrik ist zu beachten, dass sie ein Indiz für mögliche Ausreißer liefert. Diese müssen dann manuell inspiziert werden. Für eine automatische Erkennung von Ausreißern ist sie eher nicht geeignet.

Diese Metrik wird in einem Paket bereitgestellt und ist in R implementiert. Durch das spezielles Paket rpy2 [48] ist es möglich, die Metrik in Python zu nutzen. Siehe hierzu auch 6.1. Das Notebook soll prototypisch eine Einbindung bieten. Hierzu soll die Funktionalität des Paketes durch Objekte in Python direkt verwendet werden. In Jupyter Notebooks ist es durch Verwendung sogenannter magic functions möglich, Code in R direkt zu verwenden. Allerdings funktioniert dies bei dieser Metrik aus technischen Gründen nicht so leicht. Deshalb soll diese Metrik durch Verwendung von rpy2 implementiert werden.

Siehe auch Abschnitt 6.3.3.

Textähnlichkeit

Diese Metrik wurde in einer vorhergehenden Arbeit entwickelt [19] und basiert auf [18]. Für das Notebook soll der dabei entwickelte Prototyp eingebunden werden.

Die Metrik berechnet die Textähnlichkeit zwischen einem Eingabetext und einer Reihe von hinterlegten Trainingsdatensätzen. Der Anwendungsfall ist folgender: soll ein Textdatensatz mit trainierbaren Machine Learning Algorithmen verarbeitet werden, so stellt sich die Frage, welcher Trainingsdatensatz das beste Resultat liefert. Die Metrik unterstützt bei der Auswahl, indem sie den ähnlichsten Datensatz empfiehlt, unter der Annahme, dass die Qualität der Analyse umso besser wird, je mehr sich die Trainingsdaten den Eingabedaten ähneln. Diese Annahme wird in [19] mit ersten experimentellen Ergebnissen bestätigt.

Diese Metrik benötigt Datensätze, mit denen die Eingabedaten verglichen werden sollen. Für die Standardversion des Notebooks werden zu Demonstrationszwecken 6 Datensätze bereitgestellt. Diese werden in Abschnitt 9.3 genannt und kurz beschrieben. Für den konkreten Anwendungsfall sollen diejenigen Datensätze verwendet werden, die auch für die Analyse zur Verfügung stehen.

In Abschnitt 2.5.3 des Grundlagenkapitels wird eine kurze Einführung zu Textähnlichkeit geboten.

Die Details der Implementierung werden in 6.3.4 erläutert. Hier wird auch das Interface beschrieben, also wie genau Trainingsdatensätze hinterlegt werden können.

Noisy Data

Dies ist eine weitere Metrik, mit der die Qualität von Textdaten gemessen werden kann. Sie soll nicht selbst entwickelt werden, sondern beruht konzeptionell auf [18]. Damit wird der Anteil der Rechtschreibfehler in einem Text ermittelt.

Das besondere an der Einbindung im Notebook ist, dass diese Metrik eine Bibliothek benutzt, die nur mit einer 32Bit Version von Python 2.7. funktioniert. Sie dient auch als komplexeres Beispiel dafür, wie im Notebook flexibel Implementierungen aus verschiedenen Quellen und mit verschiedenen Anforderungen eingebunden werden können. Die Metrik steht nur zur Verfügung, wenn auf dem System eine entsprechende Version von Python installiert ist.

Die Implementierungsdetails werden in Abschnitt 6.3.5 beschrieben.

Part-of-speech

Diese Metrik basiert auf einem Part-of-speech tagger aus OpenNLP [43]. Dieser liefert für jedes Token einen Wahrscheinlichkeitswert für den gewählten Tag. Die Metrik mittelt die Wahrscheinlichkeiten aller Tokens im Text.

Diese Metrik basiert konzeptionell auf [18]. Für das Notebook soll sie implementiert werden.

Für weitergehende Informationen zum Part-of-Speach-Tagging, siehe Abschnitt 2.5.2 im Grundlagenkapitel.

Details zur Implementierung dieser Metrik finden sich in 6.3.6.

5.2.4 Transform Data

Datentransformationen sind ein entscheidender Teil des Data Wrangling Prozesses. Wurden im Diagnoseschritt Qualitätsprobleme im Datensatz festgestellt, so können diese eventuell durch die richtigen Transformationsschritte behoben werden. Siehe hierzu Kapitel 4.1.

Eine wichtige Funktion ist das Ersetzten von Werten. Diese soll für das Notebook implementiert werden. Dadurch ist es zum Beispiel möglich, codierte Nullwerte durch explizite zu ersetzten. Siehe hierzu 4.2.

Dieser Abschnitt soll eine Filterfunktion bieten, ähnlich der Filterfunktion im Abschnitt **View Data**. Auch hier werden Einträge ausgewählt, die bestimmten Kriterien entsprechen. Anstatt die Daten anzuzeigen, werden diese in eine neue Datenstruktur geladen. Dadurch ist es möglich, diese zu sichern oder auf diesem Teil des Datensatzes weiterzuarbeiten.

Diese Funktion kann auch dazu benutzt werden, Einträge zu entfernen. Typischer Anwendungsfall ist das entfernen von Nullwerten. Sollen beispielsweise alle Einträge mit Nullwerten entfernt werden, so wird als Kriterium „enthält keinen Nullwert“ gewählt.

Diese Funktionen sind in Anlehnung an das Bad Data Handbook [22] ins Notebook aufgenommen worden.

5.2.5 Save Data

Während der Arbeit mit dem Notebook werden die importierten Daten im Hauptspeicher gehalten. Damit die Änderungen gesichert werden, müssen sie persistiert werden. Hierzu soll das Notebook beispielhaft im Abschnitt **Save Data** eine Funktion anbieten, den Datensatz in eine CSV-Datei zu schreiben. Andere Datenformate sollen standardmäßig nicht angeboten werden, können aber eingebunden werden.

Es stehen verschiedene Parameter zur Verfügung. Neben dem Dateinamen kann unter anderem auch das Trennzeichen und das Encoding gewählt werden. Wie zu Beginn in 5.1 erwähnt steht keine Funktion bereit, mit der Änderungen leicht rückgängig gemacht werden können. Es wird daher empfohlen, Ergebnisse zwischenzuspeichern. So kann das Notebook auch zur Dokumentation der Transformationsschritte verwendet werden. Hierzu kann das gespeicherte Ergebnis mit den Ausgangsdaten verglichen werden.

Diese Funktion kann auch dazu genutzt werden, Textdaten in ein einheitliches Encoding zu konvertieren. Wird beim Import der Datei das korrekte Encoding angegeben, so reicht es beim Speichern das gewünschte Encoding anzugeben. Die Konversion soll automatisch erfolgen. Siehe hierzu die Diskussion in 4.2.

6 Implementierung

In diesem Abschnitt soll die Implementierung des Jupyter Notebook Prototypen beschrieben werden. Abschnitt 6.1 listet die verwendete Software. Abschnitt 6.2 beschreibt die Implementierung des Notebooks selbst. Die verschiedenen zur Verfügung stehenden Metriken werden in 6.3 beschrieben. Abschnitt 7 beschreibt die Implementierung ausgewählter Metriken in Spark. Siehe Kapitel 9 für eine Evaluation.

6.1 Verwendete Software

Im folgenden Abschnitt sollen alle verwendeten Tools und Softwarebibliotheken aufgezählt und kurz beschrieben werden.

Bei **DKPro Similarity** handelt es sich um ein Framework für Textähnlichkeit. Es bietet dabei unter anderem eine umfangreiche Sammlung verschiedenster Ähnlichkeitsmetriken mit standardisiertem Interface. Das Framework ist als Ergänzung zu DKPro Core [8] entworfen, eines Frameworks Sprachverarbeitung. Beide Frameworks sind open source. [2]

Dieses Framework wird zur Umsetzung der Textähnlichkeitsmetrik für Spark genutzt. Siehe hierzu Abschnitt 7.1.2. Der für die Textähnlichkeitsmetrik des Notebooks eingebundene Prototyp nutzt ebenfalls DKPro Similarity. Beide basieren konzeptionell auf [18]. Der Prototyp wurde in einer vorhergehenden Arbeit [19] entwickelt. Für eine Auflistung der hierzu verwendeten Software wird auf diese Arbeit verwiesen.

Pandas ist eine Open Source Bibliothek für Python. Sie bietet Datenstrukturen und Tools für die Datenanalyse [46].

Das Notebook verwendetet Pandas für viele Funktionen, unter anderem für den Import und die interne Verwaltung der Daten. Dies basiert auf Dataframes, einer von Pandas bereitgestellten Datenstruktur. Diese ist wie eine Tabelle organisiert ermöglicht verschiedene Funktionen wie etwa das Filtern der Daten nach bestimmten Kriterien. Siehe hierzu die Beschreibung des Notebooks in Abschnitt 6.2. Zusätzlich wird Pandas in diversen Metriken für den Import von Daten eingesetzt.

RloF ist ein Paket für die Programmiersprache R. Dieses bietet eine Implementierung der Local Outlier Factor Metrik an. [30]

Dieses Paket wird im Notebook eingebunden und für die Berechnung des Local Outlier Factors verwendet, siehe hierzu Abschnitt 6.3.3

NLTK ist ein Open Source Toolkit für die maschinelle Sprachverarbeitung [4].

Die Noisy Data Metrik verwendet für die Implementierung Funktionen aus NLTK. Siehe hierzu Abschnitt 6.3.5.

OpenNLP [43] ist ein Toolkit für Sprachverarbeitung. Es unterstützt viele gängige Aufgaben wie etwa die Tokenisierung oder das Part-of-speech tagging.

OpenNLP wird für die Implementierung der Part-of-speech Metrik verwendet. Siehe hierzu Abschnitt 6.3.6. Auch die Implementierung der Textähnlichkeitsmetrik für Spark greift auf OpenNLP zurück.

Apache Spark ist ein generelles Framework für die Verarbeitung großer Datenmengen [33]. Siehe Abschnitt 2.6 im Grundlagenkapitel für eine kurze Einführung zu Spark.

In dieser Studienarbeit wird Spark für die Implementierung und Evaluierung ausgewählter Metriken genutzt. Siehe hierzu Abschnitt 7 für die Details der Implementierung. Die Evaluation wird in Kapitel 9 beschrieben.

Anaconda ist eine offene Plattform für Data Science, basierend auf Python. [32]

Anaconda wird als Plattform für das Jupyter Notebook verwendet. Dies wird in der Installationsanleitung von Jupyter [39] empfohlen.

PyEnchant ist eine Open Source Bibliothek für Python. Sie bietet Funktionalität für die Rechtschreibprüfung. [24]

PyEnchant wird intern von der Noisy Data Metrik verwendet. Siehe hierzu Abschnitt 6.3.5.

Rpy2 ist ein Interface, mit dem es Möglich ist, aus Python heraus die Programmiersprache R zu verwenden. [48]

Dieses Paket wird für die Einbindung des in R implementierten RLoF Paketes genutzt. Siehe hierzu Abschnitt 6.3.3.

6.2 Notebook Prototyp

Die Implementierung des Notebooks basiert auf Python 3. Implementiert und getestet wurde auf Python 3.6 aus Anaconda 4.3.0.

Der allgemeine Aufbau ist, dass jeder Funktion ein Textfeld[2] vorausgeht, das Informationen zu dieser Funktion bereitstellt. Diese können Erläuterungen sein, oder Verweise auf weitergehende Ressourcen wie die Dokumentation der eingebunden Metriken. Auch finden sich hier die Referenzen auf die Literaturquellen. Hierbei handelt es sich um eine Stärke des Jupyter Notebooks. Durch eine Mischung von Text und Code können dem Benutzer zusätzliche Informationen angeboten werden. Der Code selbst ist mit Kommentaren versehen. Dadurch, dass fachliche Informationen in die Beschreibung ausgelagert werden, können sich die Kommentare aber stärker auf technische Aspekte beschränken und der Code bleibt übersichtlich. Dadurch erhöht sich auch die Benutzerfreundlichkeit. So werden zum Beispiel Links durch Verwendung von HTML klickbar, und müssen nicht erst aus dem Quellcode herauskopiert werden. Der Benutzer kann die Textfelder bearbeiten. Die Informationen werden gespeichert

2 auch als Text-Zelle bezeichnet

und stehen beim Öffnen des Notebooks zur Verfügung. Zusätzlich ist es dem Benutzer möglich, beliebig neue Textfelder anzulegen. Dies erlaubt, Erkenntnisse festzuhalten. Auch kann dies dazu beitragen, den konkreten Data Wrangling Prozess zu dokumentieren und anderen zugänglich zu machen. Eine weitere Stärke des Notebooks ist die Kollaboration. Das Notebook wird durch ein Webinterface aufgerufen. Dadurch können mehrere Benutzer mit dem gleichen Notebook arbeiten. In der Literatur werden Dokumentation und Kollaboration als wichtige Aspekte des Data Wrangling herausgestellt, beispielsweise durch Kandel et al. [16].

Die Funktionen selbst sind durch ausführbaren Code implementiert, der in Code-Zellen gegliedert ist. Der Normalfall ist eine Zelle pro Funktion. Im Allgemeinen muss der Code für ein konkretes Beispiel angepasst werden, bevor er ausgeführt werden kann. Beispielsweise muss beim Import der Name der Datei eingetragen werden, die importiert werden soll. Zusätzlich müssen eventuell weitere Parameter angegeben werden, etwa das verwendete Trennzeichen oder das Encoding. Wenn möglich, wurden sinnvolle Standardwerte gewählt. Allerdings sollte vor Ausführung der Funktion darauf geachtet werden, dass alles korrekt konfiguriert wurde.

Um eine Funktion des Notebooks durchzuführen, muss die entsprechende Code-Zelle ausgeführt werden. Hierzu gibt es im Notebook einen Button. Alternativ kann die Tastenkombination „SHIFT + ENTER“ verwendet werden. Das Notebook ist so entworfen, dass jede Code-Zelle einzeln ausgeführt werden soll, nachdem der Code gegebenenfalls für den konkreten Anwendungsfall angepasst wurde. Eine automatische Ausführung aller Zellen hintereinander macht im Normalfall keinen Sinn und wird standardmäßig nicht unterstützt. Es wäre aber durchaus möglich, das Notebook so anzupassen, dass ein vordefinierter Transformationsprozess in einem Arbeitsgang durchgeführt wird. Somit wäre eine Wiederverwendbarkeit gewährleistet. Dies wird in der Literatur gefordert, zum Beispiel durch Kandel et al. [16].

Manche Code-Zellen setzen voraus, das zuvor andere Code-Zellen ausgeführt wurden. Dies wird in der Beschreibung und auch in den Kommentaren dokumentiert. Beispielsweise kann die Textähnlichkeit nur berechnet werden, wenn zuvor im vorhergehenden Feld ein Text aus den Daten generiert wurde. Diese Trennung wurde vorgenommen, um die Modularität zu erhöhen. Bei manchen Datensätzen sind zusätzliche Vorverarbeitungsschritte notwendig. Diese können nicht durch eine allgemeingültige Lösung abgedeckt werden, sondern müssen gegebenenfalls für den jeweiligen Einzelfall implementiert werden. Siehe hierzu ein praktisches Beispiel in Kapitel 8.2.

Die Implementierung der Checkliste erfolgt als einfaches Textfeld. Soll ein Punkt als erledigt markiert werden, so muss dies durch Bearbeitung des Textfeldes erfolgen. Beispielsweise durch das Eintragen eines „X“. Diese Implementierung wurde wegen der extremen Einfachheit gewählt. Zwar ist dies nicht sehr benutzerfreundlich, allerdings wird so automatisch sichergestellt, dass die Informationen erhalten bleiben. Die Textfelder des Notebooks werden beim Schließen des Notebooks gespeichert. Außerdem wird durch das System regelmäßig automatisch zwischengespeichert. So gehen Änderungen auch bei einem Absturz oder einem anderen Problem nicht verloren. Dadurch ist es auch möglich, dass mehrere Personen an einem Notebook arbeiten. Alternativ wäre es denkbar, die Benutzerfreundlichkeit zu verbessern, indem klickbare Checkboxen verwendet werden. Allerdings muss hierfür zusätzlich eine Lösung implementiert werden, die den Zustand der Checkboxen speichert und beim Öffnen des Notebooks automatisch lädt.

6.3 Metriken

Dieser Abschnitt beschreibt die Implementierung der im Notebook verwendeten Metriken.

6.3.1 Nullwerte

Diese Metrik berechnet für alle Spalten des Datensatzes den prozentualen Anteil der Nullwerte, sowie die absolute Anzahl. Wie im Konzept in Abschnitt 5.2.3 erwähnt werden hierfür nur tatsächlich fehlende Werte berücksichtigt. Werte, welche die Semantische Bedeutung eines Nullwertes haben, werden nicht mitgezählt. Für die Implementierung werden Funktionalitäten von Pandas [46] genutzt. Siehe auch Abschnitt 6.1.

Falls diese Werte mit berücksichtigt werden sollen, so können sie durch die Ersetzungsfunktion des Notebooks in Nullwerte umgewandelt werden.

6.3.2 Validierung

Diese Metrik berechnet den prozentualen Anteil der ungültigen Werte für die angegebene Spalte. Zusätzlich wird noch die absolute Zahl ausgegeben. Die Überprüfung erfolgt durch eine Funktion, die für jeden Wert bestimmt, ob es sich um einen gültigen Wert handelt. Im allgemeinen Fall muss diese Funktion für die gewünschte Analyse jeweils implementiert werden.

Im Notebook steht eine Funktion bereit, mit der Datumsfelder validiert werden können. Dafür wird eine gängige Funktion aus Python benutzt. Hierzu muss das Format als String angegeben werden. Es wird das für Python gängige Format benutzt. Der begleitende Text enthält einige Beispiele. Siehe auch Abschnitt 7.1.1 für ein Beispiel.

Die Implementierung verwendet Funktionen von Pandas [46].

6.3.3 Local Outlier Factor

Diese Metrik basiert auf dem Paket RLoF [30]. Sie berechnet für eine Spalte den Local Outlier Factor. Siehe hierzu auch 5.2.3.

Für die Berechnung muss die entsprechende Spalte angegeben werden.

Für die Einbindung ins Notebook wird das Paket rpy2 [48] verwendet. Dieses ermöglicht es, Funktionalität aus R durch Objekte in Python zu verwenden. Siehe hierzu auch Abschnitt 6.1.

6.3.4 Textähnlichkeit

Diese Metrik ermöglicht es, Textdaten miteinander zu vergleichen. Hierzu wird die Textähnlichkeit zwischen den Eingabedaten und einer Reihe von Trainingsdatensätzen ermittelt. Die Implementierung verwendet eine in einer vorhergehenden Arbeit prototypisch entwickelten Metrik [19]. Diese basiert auf dem in [18] vorgestellten Ansatz. Siehe auch Abschnitt 5.2.3.

Die Einbindung in das Notebook erfolgt über ein Skript. Die Berechnung selbst erfolgt über den Prototypen, der als ausführbare JAR Datei vorliegt. Hierfür muss auf dem System Java installiert sein. Für die Berechnung werden die Textdaten in eine Datei geschrieben, damit der Prototyp sie verarbeiten kann. Für das korrekte Funktionieren dieser Metrik sind also Schreibrechte notwendig.

Als Eingabe erwartet die Metrik einen Eingabetext als String. Hierzu ist eine Vorverarbeitungszelle vorhanden. In der Standardvariante muss in dieser Zelle eine Spalte ausgewählt werden, die Textdaten enthält. Diese wird dann in einen Eingabetext konvertiert. Es ist an dieser Stelle auch möglich, weitere Vorverarbeitungsschritte zu berücksichtigen. Beispielsweise wird im gesamten NHTSA Complaints Datensatz durchgehend für alle Zeichen die Großschreibung verwendet. Dies gilt auch für Spalte 20, welche ein Freitextfeld enthält. Die Code-Zelle enthält hierfür exemplarisch eine Funktion, mit der man den gesamten Text in Kleinschreibung umwandeln kann. Diese ist auskommentiert, weil dadurch im Normalfall Informationen zur Groß-und Kleinschreibung verlorengehen. Für die Analyse des NHTSA Complaints Datensatzes kann diese Funktion aber nützlich sein. Im Textfeld wird darauf hingewiesen. Dies dient auch als praktisches Beispiel, wie das Notebook für die jeweilige Analyse angepasst werden kann. Für die Vorverarbeitung auch Skripte importiert werden. Siehe Abschnitt 8.2 für ein Beispiel.

Damit eine Berechnung erfolgen kann, muss mindestens ein Trainingsdatensatz hinterlegt sein. Diese müssen sich als Textdateien im Ordner training_data befinden. Jede Textdatei repräsentiert dabei einen Datensatz. Der Name der Datei wird als Name für den Datensatz verwendet. Die Textdateien sollen eine reine Textrepräsentation des Datensatzes enthalten, ohne Annotation. Als Format muss UTF-8 verwendet werden.

Für die Berechnungen in dieser Arbeit wurden exemplarisch 6 Datensätze verwendet. Siehe Kapitel 9.3 für eine kurze Beschreibung.

6.3.5 Noisy Data

Die Metrik berechnet den Anteil an Rechtschreibfehlern in einem Text und basiert auf [18]. Für die Implementierung wurde vom Betreuer bereitgestellter Code leicht angepasst. Zur Berechnung wird die Bibliothek PyEnchant [24] verwendet.

Die Metrik ist auf zwei Code-Zellen aufgeteilt. Die erste Zelle regelt die Vorverarbeitung. Die zweite Zelle erwartet als Eingebe einen Text als String. Sie bindet die eigentliche Metrik via Skript ein, und gibt das Ergebnis aus. Die beispielhaft implementierte Vorverarbeitung erzeugt den Eingabestring aus einem Textfeld. Hierzu muss die entsprechende Spalte im Datensatz angegeben werden. Sind komplexere Vorverarbeitungsschritte nötig, können diese direkt in der Zelle implementiert oder zum Beispiel via Skript eingebunden werden. Siehe Kapitel 8

für praktische Beispiele. Damit diese Metrik korrekt funktioniert, muss auf dem System Python 2.7 in der 32 Bit Version installiert sein. Die Einbindung erfolgt im Notebook über ein Skript. Hierzu muss im Skript der Pfad für Python 2.7 angegeben werden.

Das Notebook verwendet Python 3. Implementiert und getestet wurde auf Python 3.6 aus Anaconda 4.3.0. Es muss daher zusätzlich Python 2.7 installiert werden, die zur Ausführung genutzt wird. Die Implementierung verwendet dann diese Version zur Durchführung der Berechnungen. Dies geschieht durch ein Skript. Diese Metrik demonstriert, dass auch eine komplexere Konfiguration durch das Notebook berücksichtigt werden kann.

6.3.6 Part-of-speech

Diese Metrik basiert auf [18]. Siehe hierzu auch Abschnitt 5.2.3.

Als Eingabe erwartet die Metrik eine Spalte, die Textdaten enthält. Jeder Wert dieser Spalte wird durch den Tokenizer in Tokens zerlegt und anschließend an den Part-of-speech tagger weitergeleitet. Der Part-of-speech Tagger wird dabei auf jeden Eintrag angewendet und ermittelt exemplarisch die Wortarten. Dabei liefert der Tagger für jeden Tag, den er einem Token zuweist einen Wahrscheinlichkeitswert. Diese Metrik ermittelt hierfür den Durchschnittlichen Zahlenwert für alle Tokens.

Die Metrik ist in Java implementiert. Das Paket OpenNLP [43] wird für den Tokenizer und den Part-of-speech-tagger verwendet. Siehe Abschnitt 6.1. Beide benötigen entsprechende Modelle, die aus Dateien geladen werden müssen. Diese werden durch OpenNLP zur Verfügung gestellt [43]. Für die Implementierung werden sie beim kompilieren zur .jar Datei hinzugefügt und stehen somit direkt zur Verfügung. Die Einbindung in das Notebook erfolgt über ein Skript. Dies funktioniert auf ähnliche Weise, wie die Einbindung der Textähnlichkeitsmetrik.

7 Implementierung der ausgewählten Metriken in Spark

Als Ausgangspunkt wird die Implementierung der Metriken im Jupyter Notebook verwendet. Die Validierungsmetrik basiert im Notebook auf Python. Die Textähnlichkeitsmetrik wurde in einer vorhergehenden Arbeit [19] in Java implementiert, konzeptionell basierend auf [18].

Für die Umsetzung in Spark wird als Programmiersprache Java verwendet. Die beiden Metriken sind prototypisch als eigenständige Applikationen implementiert, die als .jar Dateien vorliegen. Damit diese ausgeführt werden können, müssen die .jar Dateien in HDFS kopiert werden. Danach können sie mit einem entsprechenden Befehl in Spark ausgeführt werden. Die Ergebnisse werden in HDFS im einem Ordner als Textdateien bereitgestellt.

7.1.1 Validierung von Datumsangaben

Mit dieser Metrik lässt sich überprüfen, ob Werte in einem Datensatz in einem gültigen Datumsformat vorliegen. Die Überprüfung basiert auf einer Methode von Java.

Hierzu muss via Kommandozeilenparameter ein Datumsformat spezifiziert werden. Es wird das in Java genutzte Format verwendet. Beispielsweise verwendet der NHTSA Complaints Datensatz [41] für alle Spalten, die Datumswerte enthalten, das Format „YYYYMMDD“. Dies bedeutet, dass alle Zeitangaben durch eine 8-stellige Zeichenkette dargestellt werden. Die ersten vier Zeichen enthalten das Jahr, Zeichen 5 und 6 den Monat und die letzten beiden Zeichen den Tag. Bei diesen 8 Zeichen muss es sich dabei entsprechend der üblichen Zeitangaben um Zahlen von 0 bis 9 handeln.

Der Wert 20170308 wird als gültiges Datum für dieses Format erkannt.

Der Wert 2017/03/03 enthält zusätzliche Zeichen, die nicht im obigen Format spezifiziert sind. Die Validierung würde für dieses Beispiel also fehlschlagen.

Wird hingegen zum Beispiel das Format „YYYY/MM/DD“ verwendet, so schlägt die Validierung für den ersten Wert fehl, weil er keine Schrägstriche enthält. Der Zweite Wert hingegen entspricht dem Format und wird als korrekt angesehen.

Die Implementierung dieser Metrik orientiert sich direkt an der Version dieser Metrik im Notebook. Unterschied ist, das hier Java verwendet wird. Die Metrik im Notebook basiert auf Python.

Die Applikation lädt Daten aus einer Datei, die in HDFS vorliegen muss. Die Datei soll dabei die zu analysierenden Werte enthalten. Jede Zeile stellt dabei einen Wert dar. Als Ergebnis liefert die Metrik den prozentualen Anteil der ungültigen Werte. Dieser wird in eine Datei in HDFS geschrieben.

7.1.2 Textähnlichkeit

Die Textähnlichkeitsmetrik wurde in einer früheren Arbeit entwickelt [19] und basiert konzeptionell auf [18]. Der Prototyp ist nicht für eine verteilte Architektur entwickelt worden. Deshalb ist es nicht ohne weiteres möglich, diesen im Kontext von Spark zu verwenden. Aus diesem Grund wurde die Metrik in einer abgeänderten Form neu implementiert. Beide Implementierungen basieren auf dem gleichen Konzept. Es in wird beiden Fällen zur eigentlichen Berechnung der Ähnlichkeit die Cosine Similarity Metrik aus DKPro Similarity [2] verwendet. In der für Spark implementierten Variante wird die gleiche Metrik genutzt. Der Hauptunterschied liegt darin, das nicht mehr das DKPro Core Framework für die Vorverarbeitung genutzt wird. Stattdessen werden die Textdaten direkt verarbeitet. Hierzu wird der OpenNLP Tokenizer direkt aus dem Paket OpenNLP [43] verwendet. Siehe hierzu Abschnitt 6.1 über die verwendete Software. Die ursprüngliche Implementierung benutzt den OpenNLP Tokenizer, der vom DKPro Core Framework bereitgestellt wird. Zusätzlich erfolgt die Entfernung der Stoppwörter nicht mehr durch vom Framework bereitgestellte Methoden.

Für die Implementierung in Spark wurde die Vorverarbeitung der Textdaten vereinfacht. Es kann nicht garantiert werden, dass die Metrik stets die gleichen Ergebnisse wie die Originalmetrik liefert. Hierzu wäre eine Evaluation notwendig, wie sie in [19] für den Prototypen durchgeführt wurde.

Der verwendete Tokenizer (TokenizerME) benötigt ein Modell, dass aus einer Datei geladen werden muss. Diese Datei wurde im Projekt hinterlegt und wird beim kompilieren automatisch zur .jar Datei hinzugefügt. Das bedeutet, dass sie für die Applikation automatisch sichtbar ist, und nicht aus HDFS geladen werden muss. Allerdings muss sie trotzdem eingelesen werden. Entsprechende Modelle stehen unter [43] zur Verfügung.

Damit die Metrik in Spark ausgeführt werden kann, müssen alle Eingabedaten in dem verteilten Dateisystem HDFS bereitgestellt werden. Als Textencoding soll für alle Dateien UTF-8 verwendet werden.

Der Ordner training_data soll alle verfügbaren Trainingsdatensätze in ihrer Textrepräsentation enthalten. Jede Textdatei repräsentiert dabei einen Trainingsdatensatz.

Die Datei input.txt im Ordner input soll die Eingabedaten enthalten, ebenfalls als Textdarstellung.

Zusätzlich muss die Datei stopwords.txt vorhanden sein. Jede Zeile repräsentiert dabei ein Stoppwort. Alle in dieser Datei vorkommenden Tokens werden aus den Textdatensätzen entfernt, bevor die Berechnung der Ähnlichkeit stattfindet. Ist diese Datei leer, werden keine Stoppwörter entfernt.

Wie bei diesem Modus üblich schreibt die Metrik die Ergebnisse nicht mehr in eine lokale Datei, sondern ebenfalls auf HDFS. Das Ergebnis wird in einer Textdatei geschrieben, ähnlich wie in der ursprünglichen Implementierung. Der Unterschied ist, dass für jede Berechnung ein neuer Ordner angelegt wird. Der Name dieser Ordner folgt dem Muster „Textsimilarity_output_[Timestamp]“ wobei [Timestamp] die aktuelle Systemzeit zum Zeitpunkt der Ausführung darstellt. Somit sind alle Ergebnisse im Dateisystem hinterlegt und werden nicht überschrieben.

Die oben beschriebene Implementierung wird in dieser Arbeit auch als Standardvariante bezeichnet. Zusätzlich wurde eine zweite, leicht abgewandelte Version entwickelt: Variante 2. Diese Metrik unterscheidet sich von der Standardvariante nur durch den Tokenizer. In allen anderen Punkten ist sie identisch mit der Standardvariante. Der Grund hierfür ist, dass der verwendete Tokenizer nicht serialisierbar ist. Somit muss er bei jedem Aufruf neu instanziiert werden. Hierzu wird allerdings im Gegensatz zur Standardvariante kein Modell benötigt. Dadurch ist die Instanziierung deutlich schneller. Dieses Problem wird bei der Diskussion der Ergebnisse in Abschnitt 9.5 näher erläutert. Die Metrik liefert zwar Vergleichbare, aber leicht abweichende Ergebnisse wie das Original aus [19] und die Standardvariante. Ob diese Abweichungen signifikant sind, muss in einer Evaluation geklärt werden. Dies übersteigt aber den Umfang dieser Arbeit. Aus diesem Grund wurde darauf verzichtet, eine solche Variante ebenfalls für das Notebook zu implementieren. Sie wird nur für die Evaluation der Laufzeit verwendet, und dient als Beispiel, wie unterschiedliche Implementierungen die Laufzeit beeinflussen. Siehe hierzu auch die Analyse der Evaluationsergebnisse in Abschnitt [9.5].

8 Anwendung des Notebooks auf Datensätze aus der Praxis

Im folgenden Kapitel wird das in dieser Arbeit implementierte Jupyter Notebook exemplarisch auf reale Datensätze angewendet. Dabei soll auch ein Überblick über die Datenqualität der jeweiligen Datensätze gegeben werden. Bei den verwendeten Datensätzen handelt es sich zum einen um den NHTSA Complaints Datensatz, der in Abschnitt 8.1 betrachtet wird. Zum anderen liegen Daten aus der Industrie vor. Diese werden in Abschnitt 8.2 untersucht.

8.1 NHTSA

Der NHTSA Complaints Datensatz enthält Informationen über sicherheitsrelevante Defekte bei Fahrzeugen. Die NHTSA, National Highway Traffic Safety Administration, ist eine US-amerikanische Regierungsbehörde die unter anderem für die Verkehrssicherheit zuständig ist. Der Datensatz kann auf der Webseite der NHTSA in komprimierter Form als .zip Datei heruntergeladen[3] werden. Das .zip Archiv enthält eine knapp 850 MB große Textdatei, FLAT_CMPL.txt. Zusätzlich steht eine kurze Beschreibung des Datensatzes in einer Textdatei (CMPL.txt) sowie eine Anleitung für den Import des Datensatzes in eine Access-Datenbank zum Download bereit. [41]

Im folgenden wird nur die Textdatei betrachtet, ein Import in ein Access-System wurde nicht durchgeführt. Dies hat zwei Gründe. Zum einen unterstützt der Notebook Prototyp direkt Textdateien, während eine Anbindung an MS-Access erst implementiert werden müsste. Zum Anderen werden die Daten als Textdatei zur Verfügung gestellt. Die Analyse des Datensatzes erfolgt also direkt ohne Zwischenschritte. Der Datensatz wurde am 18.03.2017 heruntergeladen, die folgende Analyse bezieht sich also auf diesen Stand.

Der Datensatz soll unter Verwendung des Jupyter Notebook Prototyps näher betrachtet werden. Als ersten Schritt müssen die Daten in das Notebook importiert werden. Hierzu wird die Importfunktion unter **Save Data** verwendet. Die Dokumentation gibt an, dass als Trennzeichen ein Tabulator verwendet wird. Dies muss beim Import im Notebook angegeben werden. Da es sich bei dem Datensatz um eine Textdatei handelt, spielt das Encoding eine Rolle. Siehe hierzu Abschnitt 4.2. Falls beim Import kein Encoding angegeben wird, verwendet das Notebook als Standardformat utf-8. Ein Import des Datensatzes mit diesem Encoding schlägt fehl, weil Zeichen nicht verarbeitet werden konnten. Dies bedeutet, dass es sich um das falsche Encoding für diese Textdatei handelt. In der Dokumentation (CMPL.txt) wird kein Encoding spezifiziert. Andere typisch vorkommende Encodings sind Code Page 1252 und ISO-8859-1. Wie in Abschnitt 4.2 erwähnt, besteht hier Verwechslungsgefahr [22]. Allerdings schlägt der Import für Code Page 1252 fehl und erfolgt nur für ISO-8859-1. Die Importfunktion ist standardmäßig so konfiguriert, dass fehlerhafte Zeilen ignoriert werden. Stattdessen wird eine Warnung angezeigt. Alternativ wäre möglich, den Import komplett abzubrechen. In diesem Beispiel meldet die Importfunktion, dass 12 Zeilen fehlerhaft sind und nicht importiert werden konnten. Hierbei handelt es sich also um ein potentielles Datenqualitätsproblem. Da der

3 https://www-odi.nhtsa.dot.gov/downloads/

Datensatz aber insgesamt aus über 1,3 Millionen Einträgen besteht, erscheint das Problem verhältnismäßig zunächst vernachlässigbar. Das Problem wird später genauer betrachtet.

Eine erste Inspektion der Daten erfolgt mit einer Funktion unter **View Data**, die es ermöglicht, einen Ausschnitt der Daten zu betrachten. Angezeigt werden sollen die ersten 5 Einträge, mit den Spalten 1-8 (Spalte 3 wurde aus Gründen der Übersichtlichkeit weggelassen). In Python beginnt das Zählen bei 0, für die erste Spalte muss also 0 angegeben werden. Abbildung 5 zeigt einen Screenshot des Resultat. Man erkennt schnell, das etwas nicht stimmt. Die oberste Zeile (fettgedruckt) soll eigentlich die Namen der Spalten enthalten. Stattdessen scheint es sich um einen Eintrag zu handeln. Offenbar gibt es in der Datei keine Kopfzeile, also eine Zeile, welche die Namen der Spalten enthält. Der erste Eintrag des Datensatzes wurde also fälschlicherweise als Kopfzeile verwendet. Dies bedeutet, dass der Import erneut ausgeführt werden muss, unter Angabe eines besonderen Parameters, der spezifiziert, das keine Kopfzeile vorhanden ist.

In [8]:
```
#Shows a Slice of the Data

#List of rows to be displayed
rows=range(5)
#List of columns to be displayed
columns=[0,3,4,5,6,7]

data.ix[rows,columns]
```

Out[8]:

	1	NISSAN	MAXIMA	1994	Unnamed: 6	19950103
0	2	NISSAN	PATHFINDER	1994	NaN	NaN
1	3	TOYOTA	LAND CRUISER	1994	NaN	19941223
2	4	TOYOTA	PASEO	1994	Y	19941226
3	5	TOYOTA	COROLLA	1994	Y	19941128
4	6	GERRY	CHILD SAFETY SEAT	1993	NaN	19941231

Abbildung 5: Screenshot mit den ersten 5 Einträge des NHTSA Complaints Datensatzes. Der erste Eintrag wird fälschlicherweise als Kopfzeile interpretiert.

Der Screenshot in Abbildung 6 zeigt das Resultat der erneuten Inspektion der ersten 5 Einträge. Da keine Spaltennamen vorhanden sind, werden diese im Notebook durchnummeriert. Der erste Eintrag wurde nun korrekt erkannt. Dieses kleine Beispiel zeigt die in Kapitel 4 beschriebene iterative Natur des Data Wrangling Prozesses. Hier musste bereits der erste Schritt, der Import, wiederholt werden, nachdem ein Problem entdeckt wurde.

In [12]:

```
#Shows a Slice of the Data

#List of rows to be displayed
rows=range(5)
#List of columns to be displayed
columns=[0,3,4,5,6,7]

data.ix[rows,columns]
```

Out[12]:

	0	3	4	5	6	7
0	1	NISSAN	MAXIMA	1994	NaN	19950103
1	2	NISSAN	PATHFINDER	1994	NaN	NaN
2	3	TOYOTA	LAND CRUISER	1994	NaN	19941223
3	4	TOYOTA	PASEO	1994	Y	19941226
4	5	TOYOTA	COROLLA	1994	Y	19941128

Abbildung 6: Screenshot mit den ersten 5 Einträge des NHTSA Complaints Datensatzes mit Parameter Header = none

Im Folgenden soll zunächst ein Überblick über die Struktur der Daten gegeben werden. Der Datensatz besteht aus 49 Spalten und hat insgesamt 1,36 Millionen Einträge. Die Einträge repräsentieren dabei Beschwerden über sicherheitsrelevante Probleme bei Fahrzeugen. Die Dokumentation gibt an, dass Daten berücksichtigt werden, die seit dem 1. Januar 1995 bei der NHTSA eingegangen sind.

Tabelle 4 gibt eine Übersicht über ausgewählte Spalten mit dazugehörigem Datentyp. Die Zahl in Klammern gibt die maximale Länge des Feldes an. Diese Informationen stammt aus der Dokumentation (CMPL.txt), die Datei selbst (FLAT_CMPL.txt) hat, wie oben erwähnt, keine Kopfzeile und somit auch keine Informationen über die Semantik der Spalten. Der gesamte Datensatz verwendet Großschreibung. Dies trifft auch für die Namen der Spalten zu, die in der Dokumentation aufgeschlüsselt werden (CMPL.txt).

Spalte	Name	Datentyp
1	CMPLID	CHAR(9)
3	MFR_NAME	CHAR(40)
4	MAKETXT	CHAR(25)
5	MODELTXT	CHAR(256)
6	YEARTXT	CHAR(4)
7	CRASH	CHAR(1)
9	FIRE	CHAR(1)
10	INURED	NUMBER(2)
11	DEATHS	NUMBER(2)
13	CITY	CHAR(30)
14	STATE	CHAR(2)
15	VIN	CHAR(11)
16	DATEA	CHAR(8)
20	CDESCR	CHAR(2048)
22	POLICE_RPT_YN	CHAR(1)

Tabelle 4: Ausgewählte Spalten des NHTSA Complaints Datensatzes

Zu Beginn der Analyse wurde beim Einlesen des Datensatzes festgestellt, dass 12 Spalten nicht eingelesen werden konnten. An dieser Stelle soll dies jetzt näher untersucht werden. Beim Import meldet das Notebook, falls Zeilen nicht eingelesen werden konnten, mit der Angabe der Zeile und des Grundes. Die Fehlermeldung besagt in allen 12 Fällen, dass zu viele Spalten vorhanden sind. Beispielsweise lautet die Fehlermeldung für die erste fehlerhafte Spalte: „Skipping line 562494: expected 49 fields, saw 70“. Um diese Zeilen inspizieren zu können, muss die gesamte Textdatei ins Notebook geladen werden. Hierzu kann als Trennzeichen der Zeilenumbruch[4] angegeben werden. Das Notebook interpretiert somit jede Zeile als einen einzigen Wert, der als String dargestellt wird. Das bedeutet, dass die Struktur des Datensatzes ignoriert wird. Nur die Textrepräsentation wird betrachtet. Dies ermöglicht den gesamten Datensatz ohne Fehler zu laden. Allerdings ist eine Verarbeitung der Daten in diesem Modus extrem aufwendig. Dies wird nur durchgeführt, um die fehlerhaften Zeilen betrachten zu können. Alternativ wäre auch ein anderes Tool, etwa ein Texteditor, denkbar.

Die Inspektion der Zeile 562494, sowie Zeilen in der Nähe dieser Zeile, liefert keine Anhaltspunkte. Allerdings fällt auf, dass die Anzahl der Zeilen in der Textdatei 1363351 beträgt, es im importierten Datensatz aber nur 1362838 Einträge gibt. 513 Zeilen fehlen, nicht nur 12. Nach aufwendiger manueller Inspektion stellt sich heraus, dass manche Zeilen ein extrem langes Textfeld haben. Dieses Textfeld enthält neben dem eigentlichen Text für diese Zeile auch Daten von einer oder mehr Zeilen als Text. Beim Import ist also ein Problem aufgetreten, und die Zeilen wurden nicht richtig erkannt. Dies führt auch dazu, dass eine irreführende Meldung ausgegeben wird, dass nur 12 Spalten betroffen sind statt 513. Dieses Problem entsteht dadurch, dass manche Textfelder Anführungszeichen enthalten („“). Die Importfunktion des Notebooks geht davon aus, das Texte unter Umständen in Anführungszeichen gesetzt werden. Dies wäre zum Beispiel notwendig, wenn der Text ein Trennzeichen enthalten würde, der

4 In diesem Fall: \n

aber Teil des Textes ist. Wird Beispielsweise ein Komma als Trennzeichen verwendet, so müssen Texte, die ein Komma enthalten, in Anführungszeichen[5] gesetzt werden. Ansonsten wäre nicht erkennbar, dass das Komma Teil des Textes ist und nicht eine neue Spalte anzeigt. Der NHTSA Datensatz verwendet keine Anführungszeichen zum markieren von Texten. Allerdings tauchen in manchen Freitextfeldern Anführungszeichen auf, so zum Beispiel in Zeile 402673. In diesem Fall enthält das Textfeld aber nur ein einzelnes Anführungszeichen. Die Importfunktion interpretiert nun alles bis zum nächsten Anführungszeichen als Bestandteil des Textes. In diesem Fall ist das nächste Anführungszeichen in Zeile 402688. Alle Einträge bis inklusive dieser Zeile werden also als Text des Textfelds von Zeile 402673 betrachtet. Tabelle 5 zeigt den Eintrag 402673 für ausgewählte Spalten. Das Textfeld wurde gekürzt, symbolisiert durch „...“, enthält aber keine weiteren Anführungszeichen mehr.

1	4	5	6	20
CMPLID	MAKETXT	MODELTXT	YEARTXT	CDESCR
402673	NISSAN	MAXIMA	2002	"MY 2002 MAXIMA HAS CAUSED ME SOOOO ...

Tabelle 5: Eintrag 402673 aus NHTSA Complaints. Hier führt ein einzelnes Anführungszeichen zu einem Problem beim Import.

Tabelle 6 verdeutlicht das Problem allgemein. Im Originaldatensatz enthalten die Textfelder von Zeile 2 und 4 jeweils ein einzelnes Anführungszeichen. Dies führt dazu, dass beim Import standardmäßig davon ausgegangen wird, das alles dazwischen einen Text darstellt, der zum Textfeld von Zeile 2 gehört. Nach dem Import gibt es statt 4 Zeilen nur noch 2, die Daten von Zeile 3 und 4 wurden dabei ins Textfeld von Zeile 2 aufgenommen.

Original:

ID	Text
1	A
2	„B
3	C
4	D“

Import:

ID	Text
1	A
2	„B 3 C 4 D“

Tabelle 6: Symbolisches Beispiel, wie einzelne Anführungszeichen zu Problemen beim Import führen

Dieses Problem lässt sich lösen, wenn beim Laden des Notebooks mit Hilfe eines Parameters[6] angegeben wird, dass Anführungszeichen ignoriert werden sollen. Nachdem der Datensatz mit diesem Parameter erneut importiert wurde, meldet die Importfunktion nur noch Fehler für

5 Oder andere vorher definierte Zeichen

6 quoting = 3

11 Einträge. Ein Vergleich mit der Zeilenanzahl ergibt, dass tatsächlich nur noch 11 Zeilen fehlen. Das Problem, dass einzelne Anführungszeichen den Import stören, hat in diesem Fall verhältnismäßig geringe Auswirkungen. Es sind nur etwa 500 Zeilen von über 1,36 Millionen betroffen, also weniger als 0,04%. Allerdings ist es denkbar, dass auch viel mehr Zeilen betroffen sein können. Taucht zum Beispiel im Textfeld der ersten und der letzten Zeile ein Anführungszeichen auf, während alle anderen Zeilen keine oder nur eine gerade Anzahl an Anführungszeichen haben, so wird der gesamte Datensatz von der Importfunktion als ein riesiges Textfeld interpretiert. Dies betrifft nicht nur die Importfunktion des Notebooks sondern auch viele weitere Tools, die Textdaten in Tabellenformat (CSV) verarbeiten. Beispielsweise tritt das gleiche Problem für das Tabellenverarbeitungsprogramm Calc von OpenOffice auf [42]. Auch hier muss beim Import die Standardmäßig ausgewählte Verwendung von Anführungszeichen als Texttrenner deaktiviert werden.

Für den NHTSA Complaints Datensatz kann dieses Problem wie oben beschrieben einfach gelöst werden. Beim Import wird via Parameter angegeben, dass Anführungszeichen ignoriert werden sollen. Dies funktioniert, weil der Datensatz diese nicht zum Markieren von Text nutzt. Das bedeutet aber auch, dass in keinem der Felder ein Trennzeichen, hier ein Tabulator, vorkommen darf. Die Importfunktion meldet, dass 11 Zeilen nicht importiert werden konnten, weil zu viele Spalten vorhanden sind. Dies deutet allerdings darauf hin, dass doch extra Tabulatoren in diesen 11 Zeilen vorhanden sind. Die erste Fehlermeldung nennt Zeile 1032430. Auch hier muss zur manuellen Inspektion der Datensatz wie oben beschrieben zeilenweise ins Notebook geladen werden. Tabelle 7 zeigt den Eintrag. Das Feld STATE enthält nach dem Kürzel TX für Texas einen Tabulator. Dieser wird als Trennzeichen interpretiert, was dazu führt, das die Importfunktion von einer extra Spalte zwischen Spalte 14 und 15 ausgeht. Da dadurch aber zu viele Spalten in der Zeile vorhanden sind, wird diese Zeile übersprungen und stattdessen eine Fehlermeldung erzeugt. Eine Untersuchung der übrigen 10 Zeilen ergibt, das bei allen das gleiche Problem auftritt. Das Feld State enthält neben dem Kürzel jeweils einen Tabulator.

1	4	5	13	14		15
CMPLID	MAKETXT	MODELTXT	CITY	STATE		VIN
1032430	FORD	FIESTA	AUSTIN	TX	[TAB]	3FADP4BJ0B

Tabelle 7: Eintrag 1032430 aus NHTSA Complaints. Ein zusätzlicher Tabulator im Feld STATE verhindert den Import.

Hierbei handelt es sich also um ein Qualitätsproblem des Datensatzes. Um diese Zeilen verarbeiten zu können, müssten die unnötigen Tabulator-Zeichen entfernt werden. Allerdings ist das Notebook nicht dazu gedacht, solche Änderungen an „rohen“ Daten vorzunehmen. Entsprechend wird keine passende Funktion bereitgestellt.

Eine mögliche Lösung könnte folgendermaßen aussehen: Der Datensatz wird zeilenweise in das Notebook geladen. Für die 11 fehlerhaften Zeilen wird per Skript der unnötige Tabulator entfernt. Das Ergebnis wird in eine Datei geschrieben. Dann kann der Datensatz mit der nor-

malen Importfunktion des Notebooks importiert und weiter bearbeitet werden. An dieser Stelle wird wegen des geringen Umfangs des Problems auf eine Implementierung verzichtet.

Das Notebook bietet die Möglichkeit, Histogramme zu berechnen und mit Hilfe eines Diagramms zu visualisieren. Ein Histogramm gibt dabei an, wie oft ein Wert in einer Spalte vorkommt. Dies soll exemplarisch für die Spalte 6 (YEARTXT) geschehen. Diese Spalte enthält als Werte das Modelljahr des betroffenen Fahrzeugs. Das Ergebnis ist in Abbildung 7 zu sehen. Dabei repräsentiert die x-Achse die in der Spalte vorkommenden Werte – in diesem Fall die Jahreszahlen. Die y-Achse repräsentiert die Häufigkeit der Werte in der Spalte. Diese Darstellung ist möglich, weil es sich bei den Werten um Jahresangaben handelt, die problemlos als Zahlen aufgefasst werden können. Dadurch ist es möglich, sie sinnvoll auf der x-Achse eines Schaubildes anzuordnen.

Das Schaubild erscheint auf den ersten Blick sehr plausibel. Der Datensatz berücksichtigt Meldungen ab 1995. Sehr alte Fahrzeuge sind im Datensatz selten. Die Anzahl der Einträge nimmt dabei mit steigender Jahreszahl immer stärker zu. Die meisten Einträge beziehen sich auf Fahrzeuge mit einem Modelljahr aus den letzten beiden Jahrzehnten. In Richtung Gegenwart nimmt die Anzahl der Einträge wieder ab. Auffallend dabei ist eine sehr deutliche Spitze ganz am Rand des Schaubilds. Es erscheint wenig plausibel, dass es extrem viele Einträge für Fahrzeuge mit Modelljahr 2017 gibt. Eine weitere Analyse ergibt, dass es sich hierbei um den Wert 9999 handelt. Die Dokumentation enthält die Information, dass dieser Wert die Bedeutung „unbekannt“ oder „nicht zutreffend“ hat. Es handelt sich also um einen codierten Nullwert. Dies ist irreführend und kann zu Datenqualitätsproblemen führen. Siehe hierzu Kapitel 4.2. Abbildung 7 ist hierfür bereits ein Beispiel. Der Datentyp der Spalte 6 ist als vierstellige Zahl festgelegt. 9999 ist also ein gültiger Wert. Die Visualisierungsfunktion hat keine Information, dass 9999 kein gültiges Jahr darstellt. Durch die Skalierung ist im Schaubild nicht sofort erkennbar, dass es sich um einen nicht plausiblen Wert handelt. Eine einfache Berechnung des Durchschnitts für diese Spalte ergibt 2358.

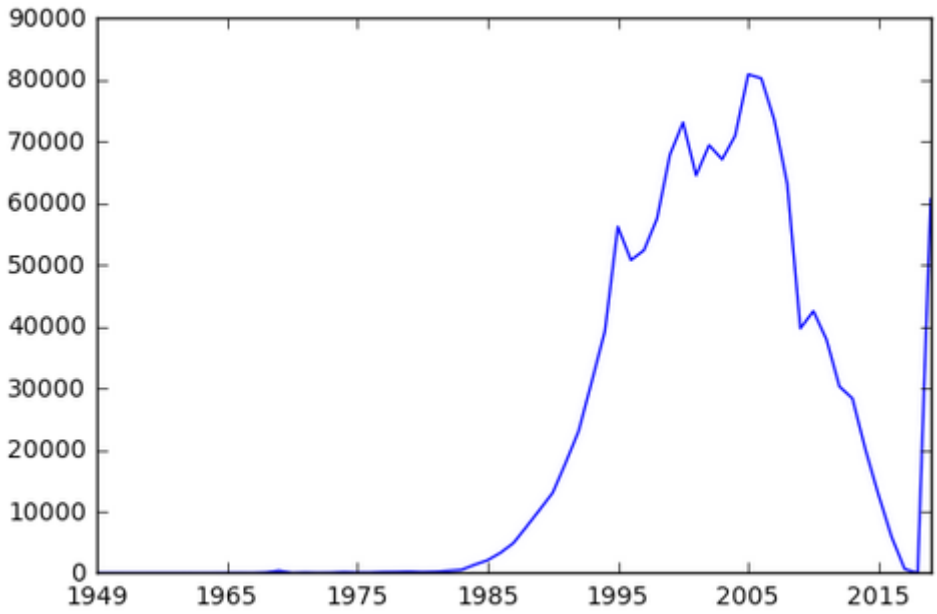

Abbildung 7: Schaubild des Histogramms für die Spalte 6

Um solche Probleme zu vermeiden, können die codierten Nullwerte in explizite Nullwerte konvertiert werden. Das Notebook bietet hierfür eine Ersetzungsfunktion. Nachdem die Ersetzung vorgenommen wurde, ergibt die Berechnung des Durchschnitts für die Spalte 6 den Wert 2002. Abbildung 8 zeigt erneut das Histogramm für Spalte 5 für den angepassten Datensatz. Die irreführende Spitze ist nicht mehr zu sehen. Zu beachten ist, dass die Ersetzung der codierten Werte nicht in jedem Fall durchgeführt werden muss. Für manche Analysen kann es durchaus interessant sein, zwischen codierten und expliziten Nullwerten zu unterscheiden. Für das konkrete Beispiel in Abbildung 8, nämlich ein korrektes Histogramm von Spalte 6, macht die Ersetzung Sinn und wurde durchgeführt.

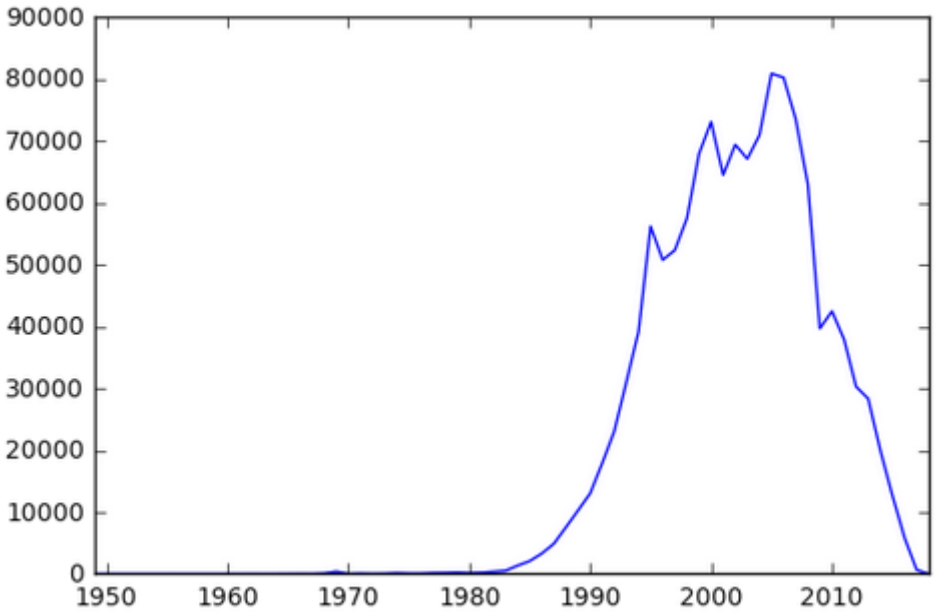

Abbildung 8: Schaubild des Histogramms für Spalte 6. Der Code 9999 wurde durch explizite Nullwerte ersetzt

Als nächsten Schritt sollen allgemein die Nullwerte im Datensatz betrachtet werden. Fehlende Werte spielen beim Data Wrangling eine wichtige Rolle, siehe hierzu Abschnitt 4.2. Das Notebook bietet eine Funktion, mit der für den Datensatz die Nullwerte ermittelt werden können. Hierbei sollen zunächst nur explizite Nullwerte betrachtet werden, also Spalten, die keinen Wert enthalten. Das zuvor durchgeführte Ersetzung des Wertes 9999 durch explizite Nullwerte in Spalte 5 fließt nicht in diese Betrachtung ein. Dies hat zwei Gründe. Zum einen müsste für ein vergleichbares Ergebnis in jeder betrachteten Spalte analysiert werden, ob codierte Nullwerte vorliegen. Dies wird aus Gründen des Umfangs für dieses Beispiel nicht durchgeführt. Zum anderen ist es durchaus interessant, die expliziten und die codierten Nullwerte getrennt zu betrachten. Bei einem codierten Wert ist bekannt, dass die Information, dass dieser Wert unbekannt ist, im Originaldatensatz vorhanden ist. Bei einem expliziten Nullwert, also dem fehlen eines Wertes, wäre auch denkbar, dass diese Information beim Transport der Daten verlorengegangen ist. Alternativ könnte auch ein fehlerhafter Transformationsschritt die Information zerstört haben. Tabelle 8 zeigt die Anzahl und den Prozentsatz der Nullwerte pro jeweiliger Spalte. Das Notebook berechnet die Werte automatisch für alle Spalten. Aus Gründen des Umfangs werden aber nur die ausgewählten Spalten betrachtet.

Spalte	Name	Absolut	Prozentual
1	CMPLID	0	0,00%
3	MFR_NAME	193	0,00%
4	MAKETXT	193	0,00%
5	MODELTXT	193	0,00%
6	YEARTXT	193	0,00%
7	CRASH	256181	19,00%
9	FIRE	80818	6,00%
10	INURED	712207	52,00%
11	DEATHS	728720	53,00%
20	CDESCR	51	0,00%
22	POLICE_RPT_YN	135529	10,00%

Tabelle 8: Explizite Nullwerte für ausgewählte Spalten

Spalte 1 enthält keine Nullwerte. Bei dieser Spalte handelt es sich laut Dokumentation um eine fortlaufende, eindeutige ID. Fehlende Werte hier würden auf ein massives Problem hindeuten, entweder im Datensatz selbst, oder beim Import. Dies ist nicht der Fall. Die Spalten 3, 4, 5 und 6 beinhalten Details zum betroffenen Fahrzeug. Beispielsweise den Hersteller (Spalte 3) oder das Modell (Spalte 5). In jeder dieser 4 Spalten gibt es 193 Nullwerte. Eine weitere Analyse zeigt, dass es sich jeweils um die gleichen Zeilen handelt. Außerhalb dieser Zeilen kommen in diesen Spalten keine expliziten Nullwerte vor. Tabelle 9 zeigt ein Beispiel für einen solchen Eintrag. Zusätzlich zu den aufgeführten Spalten fehlen Werte für die meisten anderen Felder. Der Eintrag enthält fast keine Informationen. Der Wert für Spalte 20 ist „NOTHING“ - also Nichts. Hierbei handelt es sich also eigentlich um einen Nullwert – dieser ist aber nicht explizit sondern als englisches Wort. Dies ähnelt dem oben diskutierten Fall für Spalte 5. Auch hier werden Nullwerte codiert dargestellt. Allerdings handelte es sich beim Wert 9999 um einen eindeutigen und durch die Dokumentation spezifizierten Zahlenwert. In diesem Fall sind die Nullwerte durch Texte dargestellt. Solche Nullwerte sind sehr schwer zu entdecken, weil es viele Varianten gibt, ein Fehlen von Information durch Text auszudrücken. Beispielsweise lautet die Beschreibung (Feld 20) für einen ähnlichen Eintrag 113745: „NO SUMMARY“ - also keine Zusammenfassung. Insgesamt kommt „NOTHING“ 17 mal im Datensatz vor. „NO SUMMARY“ insgesamt 1444 mal. Eine weitere Variante, die nur ein einziges mal vorkommt, ist zum Beispiel „NO FAILURES GIVEN.“. Prozentual spielen diese Einträge keine Rolle. Es gibt aber viele unterschiedliche Möglichkeiten, die theoretisch vorkommen könnten. Spalte 20 enthält nur 51 explizite Nullwerte. Allerdings kann ohne eine aufwendige Analyse nicht exakt angegeben werden, wie viele Einträge wirklich keine Informationen enthalten. Diese Beispiele stellen Fälle von irreführenden Werten da, die in Kapitel 4.2 beschrieben werden.

1	4	5	6	13	14	17	20
CMPLID	MAKETXT	MODELTXT	YEARTXT	CITY	STATE	LDATE	CDESCR
175887				BURBANK	CA	19990908	NOTHING

Tabelle 9: Beispiel für Eintrag mit Nullwerten in den Spalten 3 bis 6. Spalte 3 wurde in der Darstellung wegen der Übersichtlichkeit weggelassen.

Spalte 7 (CRASH) enthält zu 19% Nullwerte. Diese Spalte hat laut Dokumentation nur die zulässigen Werte „Y“ und „N“ für Ja und Nein. Hierbei handelt es sich eindeutig um ein Datenqualitätsproblem. Es ist nicht ersichtlich, welche Semantik ein Nullwert hier hat. Naheliegend ist die Bedeutung „unbekannt“, es ist allerdings möglich dass die Daten aus technischen oder sonstigen Gründen fehlen. Das gleiche gilt für Spalte 9 (FIRE) in abgeschwächter Form. Diese besteht zu 6% aus Nullwerten. Auch hier sind die zulässigen Werte nur „Y“ und „N“.

Die Spalten 10 (INJURED) und 11 (DEATH) haben einen Nullwertanteil von jeweils 52% bzw. 53%. Spalte 10 enthält die Anzahl der verletzten im Zusammenhang mit dem in der Beschwerde beschriebenen Defekt, Spalte 11 die Anzahl der Toten. Laut Dokumentation sollen beide Spalten Zahlen enthalten. Der sehr hohe Anteil an Nullwerten stellt ein Datenqualitätsproblem dar. Für die Spalte 10 enthalten insgesamt 65206 Zeilen einen Wert größer 0. Alle anderen Einträge sind entweder 0 oder ein Nullwert. Für die Zeile 11 sind 3500 Werte größer 0, insgesamt gibt es 634620 Werte. Bezogen darauf ist der Prozentuale Anteil von Einträgen mit Todesfällen 0,55%. Basierend darauf kann eventuell vermutet werden, dass es bei der überwiegenden Anzahl der Einträge mit Nullwert in Spalte 11 keine Todesfälle gab. Allerdings ist dies nur eine Mutmaßung. Sollen zum Beispiel bei einer Analyse nur Beschwerden betrachtet werden, bei denen es keine Toten gab, müssten theoretisch mehr als die Hälfte der Einträge verworfen werden, weil Daten fehlen.

Spalte 22 (POLICE_RPT_YN) gibt an, ob der Vorfall der Polizei gemeldet wurde, und hat als gültige Werte „Y“ für Ja und „N“ für Nein. Die Spalte enthält zu 10% Nullwerte. Laut Dokumentation wurde diese Spalte am 14.09.2007 zum Datensatz hinzugefügt. Spalte 16 (DATEA) gibt den Zeitpunkt an, wann der Eintrag in die Datenbank aufgenommen wurde. Die meisten Einträge, die vor diesem Datum zum Datensatz hinzugefügt wurden, haben einen Wert für Spalte 22. Dabei kommen sowohl die Werte „Y“ als auch „N“ vor. Tabelle 10 enthält ein Beispiel für einen Eintrag, der 1995 in die Datenbank aufgenommen wurde, deutlich vor 2007. Es ist an dieser Stelle nicht klar, was dies genau bedeutet. Waren die Werte bereits vor 2007 intern vorhanden, und wurden erst danach veröffentlicht?

1	5	6		16	22
CMPLID	MODELTXT	YEARTXT	INURED	DATEA	POLICE_RPT_YN
25499	RANGER	1993	2	19951010	Y

Tabelle 10: Beispiel für einen Eintrag mit "Y" für Spalte 22, der vor 2007 zum Datensatz hinzugefügt wurde

Spalte 20 enthält einen Freitext, der die Beschwerde näher beschreibt. Das Notebook bietet mehrere Metriken für Textdaten. Im folgenden soll die Textähnlichkeitsmetrik berechnet werden. Diese Metrik ermittelt die Ähnlichkeit des Eingabetextes mit einer Reihe von hinterlegten Korpora. Soll ein Text mit einem Machine Learning Algorithmus analysiert werden, so kann diese Metrik dabei helfen, den besten Trainingsdatensatz auszuwählen. Für eine Beschreibung der Metrik siehe Abschnitt 2.5.3. Die konkrete Implementierung der Metrik wird in Kapitel 6.3 beschrieben.

Als Eingabe benötigt die Metrik einen Text als String. Hierzu sollen alle Texte in Spalte 20 verwendet werden. Im Notebook genügt es, Spalte 20 auszuwählen. Der NHTSA Complaints Datensatz verwendet ausschließlich Großschreibung. Zur Berechnung der Metrik wird der gesamte Datensatz in Kleinbuchstaben umgewandelt. Es werden keine weiteren Vorverarbeitungsschritte durchgeführt. Der entstehende Datensatz ist sehr groß, knapp 600 MB. Zusätzlich benötigt die Metrik Trainingsdatensätze, damit ein Vergleich durchgeführt werden kann. Für die Berechnung wurden Beispielhaft 6 Textdatensätze genommen. Siehe Abschnitt 9.3 für eine Beschreibung.

Der Datensatz ist seht groß. Auf dem Demonstrationssystem benötige die Berechnung 3 Stunden. Tabelle 11 zeigt die Resultate. Die Tabelle zeigt den Namen des jeweiligen Trainingsdatensatzes und den dazu ermittelten Textähnlichkeitswert. Die Tabelle ist absteigend nach Textähnlichkeit sortiert. Der Brown Reviews Corpus erzielt mit 0,73 den höchsten Wert. Stehen nur diese 6 Datensätze zur Verfügung, so empfiehlt diese Metrik die Verwendung von Brown Reviews als Trainingsdaten. Die ermittelten Werte sind relativ niedrig. Die vorliegenden Texte enthalten Beschreibungen zu Beschwerden über Fahrzeugdefekte. Diese unterscheiden sich deutlich von Zeitungstexten, wie etwa dem CoNLL Korpus, der auf Artikeln des Wall Street Journal beruht. Die Werte sind generell etwas höher, als die für die Industriedaten ermittelten Ergebnisse. Siehe hierzu Tabelle 20 in Kapitel 8.2. Hier erzielt der Brown Fiction Korpus mit 0,62 den höchsten Wert. Die Metrik deutet also auf eine größere Ähnlichkeit von NHTSA Complaints Texten und den verwendeten Standardkorpora. Die Texte der Industriedaten unterscheiden sich laut Metrik stärker von diesen. Dies erscheint plausibel, da die Texte in den Industriedaten deutlich technischer sind und sich stark auf die Fertigung beziehen. Die Textähnlichkeit für den Twitterdatensatz beträgt 0,38. Hierbei handelt es sich um Texte der Social Media Plattform Twitter. NPS Chat erzielte einen Wert von 0,36. Dieser Datensatz basiert auf Texten aus einem Online-Chat. Beide Werte liegen deutlich unter denen der anderen 4 Korpora. Auch hier erscheint die Reihenfolge plausibel, weil sich die beiden Datensätze deutlich von den NHTSA Complaints Textdaten unterscheiden.

Trainingsdatensatz	Textähnlichkeit
brown_reviews	0,73
conll	0,69
brown_fiction	0,69
treebank	0,66
twitter	0,38
nps	0,36

Tabelle 11: Textähnlichkeit für das Freitextfeld von NHTSA Complaints

Tabelle 12 zeigt 4 Beispiele für Texte des Freitextfeldes in Spalte 20. Wie bereits zu Beginn der Analyse erwähnt wird für den gesamten Text Großschreibung verwendet. Dadurch geht Information über Groß- und Kleinschreibung verloren. Im Beispiel fallen 2 Rechtschreibfehler auf. Im ersten Eintrag aus Tabelle 12 steht „MY ITSELF“ statt „BY ITSELF“. In Eintrag 3 ist mit „ACCIENT“ wohl „ACCIDENT“ gemeint, es fehlt ein „D“.

20 CDESCR
GAS PEDEL ACCELERATOR MY ITSELF.
DRIVER DOOR LATCH FAILED AND FLEW OPEN WHILE DRIVING. DOOR WOULD NOT STAY CLOSED.
REVERSE TRNSMISSION NOT WORKING WENT OUT WHILE DRIVING. ALMOST CAUSED AN ACCIENT.
MY AIRBAG LIGHT WOULD GO ON AND OFF AT TIMES AND NOW IT STAYS ON ALL THE TIME...SAFETY CONCERN!

Tabelle 12: Vier Beispieltexte aus Spalte 20

Das Notebook bietet eine Metrik, mit der der Anteil an Rechtschreibfehlern in einem Text gemessen werden kann. Diese Metrik wurde aus [18] integriert. Für eine nähere Beschreibung siehe 6.3.5. Die Berechnung ist relativ aufwendig, so dass sie nicht für den kompletten Textdatensatz berechnet werden kann. Stattdessen wird ein Ausschnitt verwendet, für den die Metrik beispielhaft berechnet werden soll. Hierzu werden exemplarisch nur Textfelder der Einträge vom Januar 2005 ausgewählt, ca. 4500 Zeilen. Auch hier wird alles in Kleinbuchstaben umgewandelt. Dieser Datensatz ist ca. 2MB groß. Die Metrik liefert als Ergebnis 11%. Dieser Wert liegt deutlich unter den 23%, die für das Freitextfeld der Industriedaten ermittelt wurde. Allerdings sind beide Textfelder nicht vergleichbar. Siehe dazu die Ausführungen in Kapitel 8.2. Die Metrik deutet darauf hin, dass sich die NHTSA Complaints Daten eher Standard-Textdaten ähneln als die Industriedaten.

8.2 Industriedaten

Bei dem Industriedatensatz handelt es sich um Produktionsdaten im Kontext einer Fertigungslinie. Alle im Folgenden verwendeten Beispiele sind an den realen Datensatz angelehnt und anonymisiert. Dazu wurden in den Daten vorkommende Werte, wie zum Beispiel IDs, Prozesse oder Personennamen, soweit wie nötig durch allgemeine Platzhalter ersetzt. Das folgende Beispiel soll das Vorgehen verdeutlichen:

Fiktives Beispiel für einen Datensatz im Klartext:

ID	Process_ID	Process	Remark
12345	54A	GlühbirneTauschen	Glühbirne getauscht -Max

Datensatz anonymisiert:

ID	Process_ID	Process	Remark
10000	A10	[Prozess1]	[Objekt1] getauscht -[Person1]

Tabelle 13: Veranschaulichung des Vorgehens zur Anonymisierung

Tabelle 13 gibt ein Beispiel für die Anonymisierung. Eine Zeile (zur Illustration aus einem fiktiven Datensatz) ist zuerst im Klartext zu sehen. Nach der Anonymisierung sind die konkreten Entitäten durch abstrakte Platzhalter ersetzt. Zusätzlich wurden auch technische Daten, wie etwa IDs, abgewandelt.

Der Datensatz basiert auf Daten, die in einer relationalen Datenbank verwaltet werden. Für diese Analyse liegt ein Auszug in Form von .csv Dateien vor, die jeweils die entsprechenden Tabellen repräsentieren.

Als ersten Schritt werden die Daten in das Jupyter Notebook geladen. Da die Daten als eine Reihe von .csv Dateien vorliegen, muss hierzu neben dem Pfad der entsprechenden Datei auch das verwendete Trennzeichen angegeben werden. In diesem Fall handelt es sich um ein Semikolon. Zum Einstieg soll eine Tabelle betrachtet werden – im folgenden Tabelle A genannt, die Prozessnamen enthält. Diese hat 6 Spalten mit ungefähr 1200 Einträgen. Tabelle 14 zeigt exemplarisch zwei Einträge.

DB_Param	Process_ID	AG_ID	Description	EquipmentNo	Type
L100	2000	1234	[Prozess1]		2
L200	1000	5678	[Prozess2]		2

Tabelle 14: Beispielhafte Einträge für Tabelle A

Die hier anonymisiert dargestellten Beschreibungen (Spalte Description) [Prozess1] und [Prozess2] sind auf Deutsch und enthalten Umlaute. Das Notebook bietet eine Funktion an, mit

der Sonderzeichen in Textfeldern erkannt werden können. Siehe Kapitel 6. Die Texte wurden vom Jupyter Notebook Prototypen korrekt geladen und werden inklusive Umlaute richtig angezeigt. Allerdings ist es wichtig, bei der weiteren Verarbeitung der Daten auf das korrekte Encoding zu achten. Theoretisch wäre es möglich, das für die weitere Analyse ein Tool verwendet wird, das keine Umlaute oder andere Sonderzeichen unterstützt. In diesem Fall könnten die Umlaute zum Beispiel umgewandelt werden (ü → ue). Das Notebook bietet hierfür standardmäßig keine Funktion an. Falls dies für einen Analyseschritt tatsächlich erforderlich wäre, könnte man diese Funktion problemlos hinzufügen, etwa durch Einbindung eines Skriptes.

Ein Histogram der ersten Spalte (DB_Param) zeigt, das 56 Einträge den Wert „Test“ enthalten. Hier muss geklärt werden, ob es sich bei diesen Einträgen um Testeinträge handelt, also um Einträge die erstellt wurden, um die Funktion des Systems zu überprüfen. In diesem Fall enthalten diese Einträge keine realen, fachlichen Daten und sollten bei der Analyse nicht berücksichtigt werden. Hierzu bietet das Notebook eine Filterfunktion, mit der Zeilen entfernt werden können, die bestimmten Kriterien entsprechen. In diesem Fall: der Wert der Spalte DB_Param ist gleich „Test“. Diese Funktion des Notebooks wird in Abschnitt 5.2.4 näher beschrieben.

Die Prozessbeschreibungen (Spalte Description) sind relativ kurz, im Mittel etwa 26 Zeichen lang. Allerdings enthalten einige Beschreibungen als letztes Zeichen einen Zeilenumbruch (\r\n). Dies könnte eventuell bei weiteren Analysen zu Problemen führen. Da der Zeilenumbruch das letzte Zeichen ist, enthält er auch keine zusätzlichen Informationen und sollte entfernt werden. Das Notebook stellt für diesen Fall keine fertig implementierte Funktion bereit. Allerdings ist es problemlos möglich, im Notebook eine neue ausführbare Zelle hinzuzufügen. Damit kann das Problem direkt durch Code gelöst werden, oder ein passendes Skript importiert werden. In diesem Fall reicht eine Zeile Code in Python[7]. Zeilenumbrüche werden durch das Notebook nicht speziell erkannt. Im Normalfall ist es durchaus denkbar, das ein Freitextfeld Zeilenumbrüche enthält, etwa bei längeren Texten. Dies stellt allgemein kein Qualitätsproblem dar. In diesem Fall sind die unnötigen Zeilenumbrüche durch manuelle Inspektion eines Ausschnitts der Daten entdeckt worden. Falls Zeilenumbrüche beim Analyseprojekt eine Rolle spielen sollten, können entsprechende Funktionen leicht ins Notebook integriert werden.

Als nächsten Schritt werden die Nullwerte in den jeweiligen Spalten berechnet. Abbildung 9 zeigt das Ergebnis. Die Spalten DB_Param, Process_ID und Type enthalten überhaupt keine Nullwerte, die Spalte Description nur 2. Auffällig ist, das die Spalte AG_ID zu 68% Nullwerte enthält. An dieser Stelle sollte die Semantik dieser Spalte geklärt werden. Wahrscheinlich ist diese Spalte optional. Falls nicht, handelt es sich um ein sehr großes Datenqualitätsproblem. Die Spalte EquipmentNo besteht zu 100% aus Nullwerten und enthält keinen Einzigen Eintrag. Auch an dieser Stelle sollte geklärt werden, aus welchem Grund überhaupt keine Werte vorhanden sind. Eventuell ist es möglich, das diese Spalte nicht benötigt wird. In diesem Fall kann sie entfernt werden. Beim Speichern der Ergebnisse können im Notebook Spalten ausgewählt werden. Spalten die entfernt werden sollen, können einfach weggelassen werden. Diese Funktion wird in Kapitel 5.2.5 näher beschrieben.

7 data['Description'] = data['Description'].str.replace('\r\n', '')

Out[33]:

	Count	Percent
DB_Param	0	0%
Process_ID	0	0%
AG_ID	811	68%
Description	2	0%
EquipmentNo	1194	100%
Type	0	0%

Abbildung 9: Nullwerte für Tabelle A

Das Notebook bietet die Möglichkeit, eine Qualitätsmetrik zu berechnen, die den Anteil der Rechtschreibfehler in einem Textdatensatz misst. Siehe Kapitel 6.3.5 für eine Beschreibung. Diese Metrik soll für das Feld Description beispielhaft berechnet werden: Hierzu genügt es, die entsprechende Spalte anzugeben, sowie die Sprache Festzulegen (hier Deutsch). Als Ergebnis liefert die Berechnung einen Wert von 25%. Dieser Wert erscheint überraschend hoch, schließlich handelt es sich um vordefinierte Beschreibungen, nicht um Freitexte. Allerdings ist zu bedenken, dass die Metrik ein Standard-Wörterbuch für die Deutsche Sprache verwendet. Fachbegriffe und fachspezifische Abkürzungen sind nicht im Wörterbuch gelistet und werden daher als Fehler gewertet. Für ein aussagekräftiges Ergebnis müsste eine Metrik verwendet werden, die den domänenspezifischen Sprachgebrauch berücksichtigt. Allerdings kann diese Metrik auch als Indikator dafür benutzt werden, ob ein Text von Standardtexten abweicht, wie dieses Beispiel zeigt.

An dieser Stelle soll die exemplarische Analyse für diese Tabelle beendet werden. Im Kontext dieser Analyse wurden zwei mögliche Änderungen an den Daten beschrieben: Testeinträge entfernen und Bereinigen der Texten von unnötigen Zeilenumbruchzeichen. Diese Änderungen müssen noch durch einen Domänenexperten geprüft werden. Falls sie aus fachlicher Sicht Sinn machen, müssen sie gespeichert werden. Wenn die entsprechende Funktionen des Notebooks durchgeführt wurden, sind die Änderungen im Arbeitsspeicher, aber noch nicht dauerhaft gesichert. Hierzu ist es im Notebook möglich, die Daten in eine .csv Datei zu schreiben. Zusätzlich soll die Spalte EquipmentNo entfernt werden, weil sie nur Nullwerte enthält. Im Notebook wird unter Save Data die zu speichernden Daten bestimmt: der Datensatz mit allen Spalten außer EquipmentNo. Zusätzlich muss der Dateiname angegeben werden. Optional können noch weitere Parameter wie das Trennzeichen und das verwendete Encoding (als default-Wert wird UTF-8 genommen) gewählt werden. Damit die Datei gespeichert wird, muss die Code-Zelle ausgeführt werden.

Als zweites betrachten wir eine weitere Tabelle mit 7 Spalten und ca. 150.000 Einträgen. Diese wird im folgenden als Tabelle B bezeichnet. Tabelle 15 enthält einen Beispiel-Eintrag.

ID	LFDNR	DB_Param	ProductionStart	ProductionEnd	Shift	TTN	Parts
90000	1234567	L100	[Timestamp1]	[Timestamp2]	1	123456789	500

Tabelle 15: Beispielhafter Eintrag für Tabelle B

Diese Tabelle enthält keine Textfelder oder andere unstrukturierte Daten. Die Berechnung der Nullwerte liefert folgendes Ergebnis (Abbildung 10): Die Spalten ID, DB_Param, ProductionStart, Shift und Parts enthalten keine Nullwerte.

Out[42]:

	Count	Percent
ID	0	0%
LFDNR	2830	2%
DB_Param	0	0%
ProductionStart	0	0%
ProductionEnd	444	0%
Shift	0	0%
TTN	5241	4%
Parts	0	0%

Abbildung 10: Nullwerte für Tabelle B

Die Spalte LFDNR enthält 2830 Nullwerte, etwa 2% der Einträge. Tabelle 16 enthält beispielhaft einen Eintrag, bei dem es für LFDNR keinen Wert gibt. Auffällig bei diesem Eintrag ist außerdem, das für ProductionStart und ProductionEnd der gleiche Timestamp eingetragen ist. Dies ist für insgesamt 19 Datensätze der Fall, bei denen LFDNR Null ist. In Tabelle B kommt diese Konstellation (ProductionStart gleich ProductionEnd) ansonsten nicht mehr vor. Dies deutet stark auf Probleme mit diesen Einträgen hin.

ID	LFDNR	DB_Param	ProductionStart	ProductionEnd	Shift	TTN	Parts
50000		L123	[Timestamp1]	[Timestamp1]	0	1112223334	100

Tabelle 16: Beispiel für einen Eintrag mit LFDNR = Null

Die Spalte ProductionEnd enthält 444 Nullwerte. Eine weitere Analyse zeigt, dass alle diese Einträge auch gleichzeitig einen Nullwert in der Spalte LFDNR haben.

Die Spalte TTN enthält 5241 Nullwerte, etwa 4% der gesamten Einträge. Ein Histogram dieser Einträge zeigt, dass alle diese Einträge einen von 8 Werten für DB_Param haben, wobei mit 3939 Einträgen 75% der Nullwert-Einträge auf 2 DB_Param-Werte entfallen. Diese Spalte enthält die Fertigungswerkstatt des zugehörigen Eintrags. In der gesamten Tabelle kommen 45 verschiedene DB_Param Werte vor.

Um im weiteren Vorgehen entscheiden zu können, ob es sich bei den zuvor beschriebenen Anomalien in Tabelle B um relevante Datenqualitätsprobleme handelt, ist zusätzliches Domänenwissen erforderlich, zum Beispiel über die Semantik der Einträge. Eine allgemeine Lösung wäre es, alle Einträge mit Nullwerten auszufiltern. Allerdings könnten gerade diese Einträge Wertvolle Informationen über Fehler und Probleme darstellen.

Als nächstes soll eine Tabelle betrachtet werden, die ein Schichtbuch darstellt, in dem Probleme bei der Produktion eingetragen werden. Die Tabelle hat insgesamt 15 Spalten und etwa 150.000 Einträge. Tabelle 17 bietet eine Übersicht aller Spalten sowie die dazu ermittelten Nullwerte. Tabelle 18 zeigt einen beispielhaften Eintrag. Aus Gründen der Lesbarkeit wurden mehrere Spalten weggelassen.

Spalte	Nullwerte
ID	0%
EREIGNISLISTE_ID	96%
DB_Param	0%
Process_ID	0%
Code	0%
Begin	0%
End	0%
TTN	24%
EquipmentNo	100%
Cause	68%
Remark	66%
Person_ID	99%
Duration_in_Seconds	0%
Affected_Parts	0%
isCreatedBySystem	0%

Tabelle 17: Spalten der Tabelle "Schichtbuch"

DB_Param	Process_ID	Code	TTN	Cause	Remark
L321	1000	Y1234Z	1234567890	[Objekt] verklemmt	

Tabelle 18: Beispiel für einen Eintrag in der Tabelle "Schichtbuch" für ausgewählte Spalten

Ein Histogramm der Spalte FtedBySystem zeigt, das alle Einträge den Wert 0 haben. Die für diese Spalte vorhandene Dokumentation besagt, das dies immer der Fall ist. Genauere Informationen liegen nicht vor. Diese Spalte kann also für die weiteren Analysen entfernt werden, da sie keine Informationen enthält. Der Prozentsatz der Nullwerte pro jeweiliger Spalte

kann Tabelle 17 entnommen werden. In 9 Spalten kommen keine Nullwerte vor. Ähnlich wie in Tabelle B enthält die Spalte EquipmentNo nur Nullwerte. Auch die Spalte EREIGNISLISTE_ID hat einen sehr hohen Nullwertanteil von 96%. In diesen beiden Fällen sollte geklärt werden, warum dies so ist. Etwa 24% der Einträge der Spalte TTN sind Nullwerte. Auch hier stellt sich die Frage, welche Semantik ein Nullwert hat. Die Spalte Person_ID besteht zu 99% aus Nullwerten. Eine weitergehende Analyse ergibt in diesem Fall, das diese Spalte optional ist und von den Werkern in der Regel nicht eingetragen wird. Bei den Spalten Cause und Remark handelt es sich um Freitextfelder. Diese Enthalten Kommentare und Anmerkungen, die von den Werkern zu den entsprechenden Einträgen verfasst wurden. Die Felder haben einen Nullwert-Anteil von 68% beziehungsweise 66%. Betrachtet man nur die Spalten, in denen sowohl Cause als auch Remark keine Einträge besitzen, so reduziert sich der Anteil der Nullwerte auf etwa 45%. Nur etwa 11% der Einträge haben einen Wert für beide Spalten, sowohl Cause als auch Remark. Von diesen Einträgen haben allerdings über 61% einen identischen Wert, jeweils den gleichen für Cause und Remark. Insgesamt haben nur knapp 4% der Einträge, bezogen auf die gesamte Tabelle, unterschiedliche Texte. Abbildung 11 illustriert diesen Zusammenhang. Dies deutet auf eine Redundanz dieser beiden Felder hin. Die Einbeziehung domänenspezifischen Wissens zeigt dass dieses Problem bereits bekannt ist und diese Felder möglicherweise zusammengelegt werden sollen.

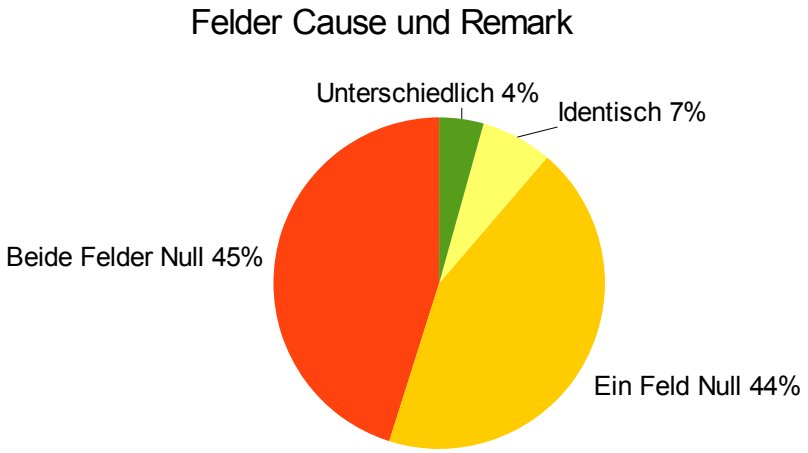

Abbildung 11: Prozentuale Verteilung für die Felder Cause und Remark

Das Feld Duration_in_Seconds gibt die Dauer der Störung in Sekunden an. Ein Histogramm dieser Spalte zeigt, das die 7 häufigsten Werte alles Vielfache von 60 sind, also runde Minutenangaben darstellen. Insgesamt sind etwa 26% der Werte direkte Vielfache von 60. Knapp 0,7% der Einträge haben den Wert 0. An dieser Stelle stellt sich die Frage, ob es sich hierbei um Fehler in den Daten handelt, oder ob eine Dauer von 0 Sekunden eine besondere fachliche Bedeutung hat und deshalb plausibel ist. Das Notebook integriert eine Metrik zum Erkennen von Ausreißern, die hier exemplarisch für die Spalte Duration_in_Seconds angewendet werden soll. Hierzu werden für die ersten hundert Einträge der LoF-Wert ermittelt. Siehe hierzu Kapitel 6.3.3. Abbildung 12 zeigt einen Plot des Ergebnisses, der auch im Notebook angezeigt wird. Die x-Achse repräsentiert dabei den Eintrag, die y-Achse den ermittelten LoF-Wert. Werte deutlich über 2 deuten bei dieser Metrik auf mögliche Ausreißer hin. Laut Plot gibt es nach dieser Metrik 3 Kandidaten für Ausreißer, zum Beispiel die erste Spitze in der Grafik mit einem Wert von 4,75. Im folgenden soll nur der erste Eintrag näher betrachtet wer-

den. Tabelle 19 zeigt den Eintrag mit ausgewählten Spalten. Wie alle anderen Beispiele auch ist die Darstellung am echten Eintrag angelehnt aber vollständig anonymisiert. Der Wert der Spalte Durations_in_Seconds ist mit 13 deutlich niedriger als der Durchschnitt für diese Spalte, der bei 480 liegt. Allerdings kann keine Aussage darüber getroffen werden, inwieweit der Wert einen Ausreißer darstellt. Falls ein Ausreißer vermutet wird, muss eine detailliertere Analyse erfolgen.

DB_Param	Process_ID	Code	TTN	Cause	Remark	Duration_in_Seconds
B123	2000	A1234Z	9876543210	[TTN1]->[TTN2]	[TTN1]->[TTN2]	13

Tabelle 19: Eintrag mit LoF-Wert von 4,75

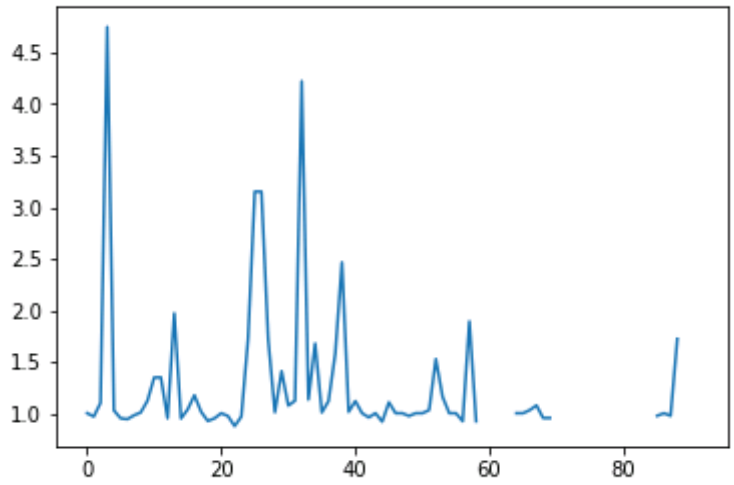

Abbildung 12: LoF-Werte für die ersten 100 Einträge der Spalte Duration_in_Seconds

An dieser Stelle soll die Validierungsfunktion demonstriert werden. Hierzu wird das Feld Process_ID überprüft. Die zuvor beschriebene Tabelle A schlüsselt die in dieser Spalte enthaltenen Codes weiter auf. Für dieses Beispiel wird vereinfachend überprüft, ob ein Code aus der Tabelle Schichtbuch auch in der Tabelle A vorkommt. Die Metrik liefert als Resultat, dass dies für jeden einzelnen Eintrag der Fall ist. Dies überrascht nicht, da die Daten aus einer relationalen Datenbank stammen, die solche Konsistenzkriterien automatisch sicherstellt. Eine Überprüfung wäre aber sinnvoll, falls die Daten zum Einlesen komplexeren Vorverarbeitungsschritten unterzogen werden müssten. Auch nach dem Ausführen von Transformationen auf den Daten kann eine Überprüfung Sinn machen, um potentielle Fehler aufzuspüren.

Die Spalten Cause und Remark sind Freitextfelder, enthalten also unstrukturierte Daten in Form von Texten, die das Problem näher beschreiben. Eine weitergehende Analyse der in den Texten enthaltenen Informationen mit Algorithmen, die natürliche Sprache verarbeiten, wäre möglicherweise interessant. Dabei stellt sich die Frage nach der Qualität der Textdaten. Der Notebook Prototyp bietet eine Metrik, mit der die Textähnlichkeit zwischen dem zu untersuchenden Datensatz, und einer Reihe von vorhandenen Trainingsdatensätzen ermittelt werden

kann. Die Idee ist, das die Ergebnisse eines Machine Learning Algorithmus umso besser sind, je mehr sich die Trainings- und die Eingabedaten gleichen [18]. Für Details siehe die Beschreibung der Metrik in Abschnitt 6.3.4. Für diese Beispielhafte Anwendung der Textähnlichkeitsmetrik werden die Textdaten aus den Spalten Cause und Remark entnommen. Wegen der zuvor beschriebenen Situation, das die beiden Spalten nur in 4% der Fälle unterschiedliche Einträge enthalten, werden beide Spalten zusammen als ein Textdatensatz betrachtet. Dieser Textdatensatz besteht also aus Einträgen der Spalten Cause und Remark, jeweils ohne Nullwerte. Die einzelnen Einträge werden mit einem Zeilenumbruch getrennt. Für den oben beschriebenen Fall, dass Cause und Remark einen identischen Text enthalten, wird der Text nur ein mal verwendet, das Duplikat wird nicht berücksichtigt. Es werden keine weiteren Vorverarbeitungsschritte durchgeführt. Das Notebook bindet die Textähnlichkeitsmetrik ein, allerdings erfordert das Zusammenlegen von Spalten, sowie die Bereinigung von Duplikaten extra Code. Dies kann problemlos durch Hinzufügen einer zusätzlichen ausführbaren Zelle erreicht werden. Abbildung 13 zeigt den Code, der für dieses Beispiel benutzt wurde.

Da dieses Fragment Domänenspezifisch ist, wird es nicht in das Standard-Notebook integriert, sondern als Skript bereit gestellt. An dieser Stelle zeigt sich die Flexibilität des Notebooks als großer Vorteil. Durch einfaches einbinden von Code können auch Einzelfall-spezifische Verarbeitungsschritte unterstützt werden.

```
In [ ]: # Select all values from Cause that are not null
        cause_df = data[data['Cause'].notnull()]
        # Remove duplicates
        cause_df = cause_df[cause_df['Remark'] != cause_df['Cause']]
        # Select all values from Remark that are not null
        remark_df = data[data['Remark'].notnull()]

        # Convert to lists
        cause = cause_df['Cause'].tolist()
        remark = remark_df['Remark'].tolist()
        data_sample = cause + remark

        # Convert to String, using a linebreak as separator
        text = "\n".join(data_sample)

        # Convert entire String to lower case
        text = text.lower()
```

Abbildung 13: Code-Zelle im Jupyter Notebook. Der Code erzeugt den Textdatensatz aus den Spalten Cause und Remark

Nachdem die Textdaten in das von der Textähnlichkeitsmetrik erwartete Format gebracht wurden, kann die Metrik einfach durch Ausführen der Code-Zelle berechnet werden. Tabelle 20 zeigt die Ergebnisse. Die erste Spalte zeigt die vorhandenen Trainingsdatensätze. Die 6 verwendeten Datensätze wurden beispielhaft ausgewählt. Für eine Beschreibung der Datensätze siehe Abschnitt 9.3. Die zweite Spalte zeigt die ermittelte Textähnlichkeit, ein Wert zwischen 0 und 1. Je höher der Wert, desto ähnlicher sind sich die entsprechenden Textdaten. Die Tabelle ist nach der Textähnlichkeit sortiert.

Trainingsdatensatz	Text Similarity
brown_fiction	0,62
conll	0,58
brown_reviews	0,55
treebank	0,54
nps	0,51
twitter	0,47

Tabelle 20: Die Ergebnisse der Textähnlichkeitsmetrik für den Textdatensatz aus den Spalten Cause und Remark

Der Brown Fiction Datensatz erzielt die höchste Ähnlichkeit, mit einem Wert von 0,62. Stehen nur diese 6 Trainingsdatensätze zur Verfügung, so wäre die Empfehlung des Systems, die Brown Fiction Daten als Trainingsdaten zu verwenden. Allerdings ist der erzielte Wert sehr niedrig. Es ist sehr wahrscheinlich, dass die Qualität der Analyse dadurch stark beeinträchtigt werden würde. Dies liegt daran, dass die für dieses Beispiel verwendeten Trainingsdatensätze hauptsächlich aus literarischen und journalistischen Texten bestehen. Daneben handelt es sich bei den Datensätzen NPS und Twitter um Daten aus einem Onlinechat beziehungsweise um Daten der Social Media Plattform Twitter. Diese Datensätze unterscheiden sich stark von technischen Anmerkungen im Produktionskontext. Die Kommentare enthalten viel fachspezifisches Vokabular und Abkürzungen, wie etwa „AEG“ für „Ausschalten, Einschalten, Geht“. Tabelle 21 zeigt hierfür ein Beispiel.

DB_Param	Process_ID	Code	TTN	Cause	Remark
L123	2000	A1234Y	1234567890	Bleibt beim [Vorgang] stehen. AEG	

Tabelle 21: Beispiel für Domänenspezifische Abkürzungen

Ein weiterer Aspekt ist das die Textfelder oft Kommentare der Form TTN1 → TTN2 enthalten, was sich auf bestimmte domänenspezifische Prozesse bezieht. Eine TTN ist dabei ein technische Bezeichner, zum Beispiel in Form einer Zehnstelligen Zahl. Tabelle 22 bietet hierfür ein Beispiel. Das Beispiel zeigt auch die zuvor beschriebene Konstellation, das Cause und Remark den gleichen Inhalt haben, und dadurch Informationen unnötig dupliziert werden.

In einem zweiten Schritt wurde die Textähnlichkeitsmetrik nochmal ohne diese Kommentare berechnet. Hierzu wurde vereinfachend alle Kommentare gefiltert, die die Zeichenkette „->“ enthalten. Allerdings änderte sich das Ergebnis der Metrik hierdurch nicht.

DB_Param	Process_ID	Code	TTN	Cause	Remark
L345	3000	Y4321X		1234567890 → 9876543210	1234567890 → 9876543210

Tabelle 22: Beispiel für einen Kommentar der Form TTN->TTN

Diese Kommentare stellen eigentlich technische, strukturierte Information dar, und kommen so in natürlicher Sprache nicht vor. An dieser Stelle stellt sich die Frage, ob solche Informationen nicht besser in einer Strukturierten Form repräsentiert werden sollten. So ist es zum Beispiel bei einem Textfeld nicht möglich die TTN zu validieren, um etwa Fehlern vorzubeugen. Anstatt die Informationen direkt aus den strukturierten Daten zu übernehmen müssten sie bei Analysen erst aufwendig durch Verarbeitung von Strings extrahiert werden, etwa durch reguläre Ausdrücke.

Das Notebook bietet eine weitere Qualitätsmetrik für Textdaten: es ist möglich, den Anteil an Rechtschreibfehlern in einem Text zu messen. Hierzu werden als Eingabedaten die Spalten Cause und Remark verwendet. Die Vorverarbeitung erfolgt genauso wie im obigen Beispiel für die Textähnlichkeit. Nullwerte werden entfernt, zusätzlich wird ein Text nur einmal benutzt, falls Cause und Remark den gleichen Inhalt haben. Als Sprache wird Deutsch ausgewählt. Die Metrik liefert als Ergebnis einen Wert von 23%. Die Kommentare enthalten in der Tat Rechtschreibfehler. Tabelle 23 zeigt als Beispiel einen solchen Eintrag. Die Felder enthalten Kommentare über Probleme, die vom Personal unter Produktionsbedingungen eingetragen wurden. Neben Rechtschreibfehlern kommen auch für normalen Sprachgebrauch ungewöhnliche Abkürzungen vor, etwa um Zeit zu sparen. Tabelle 24 bietet hierfür ein Beispiel. Bei „Q.“ handelt es sich wahrscheinlich um eine Abkürzung für Qualität.

DB_Param	Process_ID	Code	TTN	Cause	Remark
A123	1000	Y1234Z	1234567890		[Vorgang1]. Danach geht niochts mehr.

Tabelle 23: Beispiel für einen Rechtschreibfehler in den Spalten Cause und Remark

DB_Param	Process_ID	Code	TTN	Cause	Remark
A456	2000	Y1234Z	9876543210		[Objekt] schlechte Q.

Tabelle 24: Beispiel für eine Abkürzungen in den Spalten Cause und Remark

Die Metrik basiert auf einem Standardwörterbuch der Deutschen Sprache. Fachbegriffe sind nicht bekannt und werden als Fehler gewertet. Der Wert von 23% ist daher wahrscheinlich höher als der reale Anteil an Rechtschreibfehlern. In diesem Fall könnte ein genaueres Ergebnis erzielt werden, wenn eine Metrik verwendet wird, die Fachwissen mit einbezieht. Bei Fachausdrücken und Abkürzungen handelt es sich zwar nicht um Rechtschreibfehler im eigentlichen Sinne, trotzdem bereiten sie Probleme bei der Verarbeitung durch Standardtools. Diese Metrik selbst ist ein gutes Beispiel dafür. So wurde am Anfang dieser Analyse in der Tabelle A für die Spalte Description ein Rechtschreibfehleranteil von 25% berechnet. Dies ist allerdings unwahrscheinlich, weil es sich um statische Beschreibungstexte handelt. Der Grund wird eher in einem hohen Anteil an Fachvokabular liegen.

Insgesamt deuten die Ergebnisse darauf hin, das eine Analyse der Textfelder Cause und Remark durch Standardtools wahrscheinlich keine guten Resultate liefern würde, weil die Art

der vorliegenden Textdaten sich stark sauberen Textdaten wie etwa Zeitungstexten unterscheidet.

9 Evaluation

In dieser Arbeit wurde ein Jupyter Notebook entwickelt, das verschiedene Datenqualitätsmetriken anbietet. Dabei stellt sich die Frage, ob diese Metriken sich auch für große Datensätze skalieren lassen. In diesem Kapitel soll daher exemplarisch eine Evaluation durchgeführt werden. Hierzu wurden zwei Metriken beispielhaft ausgewählt und in Spark implementiert. Siehe hierzu Kapitel 7. Diese Metriken entsprechen den bereits in das Notebook eingebundenen Metriken, nutzen aber Apache Spark. Die Evaluation soll klären, in wie weit sich dadurch Vorteile im Hinblick auf die Laufzeit bei Anwendung auf große Datensätze ergeben. Abschnitt 9.1 beschreibt dabei die verwendete Methodik. Der Aufbau der Experimente, insbesondere die Konfiguration des Clusters, wird in Abschnitt 9.2 näher beschrieben. In Abschnitt 9.4 werden die Ergebnisse präsentiert und in Abschnitt 9.5 analysiert. Abschnitt 9.6 enthält eine kurze Zusammenfassung.

9.1 Methodik

Die Evaluation der für Spark implementierten Metriken erfolgt durch Laufzeitmessungen mit Hilfe einer Cloud-basierten Infrastruktur, basierend auf Openstack [40]. Hierzu wurde ein beispielhaftes Spark-Cluster eingerichtet. Die Metriken werden als Applikationen implementiert und auf dem Cluster ausgeführt. Siehe Kapitel 7 für die Beschreibung der Implementierung. Zur Messung der Laufzeit wird die von Spark bereitgestellte Web-API verwendet. Diese bietet Informationen zum Cluster und den darauf ausgeführten Applikationen. Unter anderem gibt es zu jeder beendeten Applikation die Information zur Laufzeit in der Spalte Duration. Für die Konfiguration des Clusters wurde dabei so weit wie möglich eine Standardkonfiguration verwendet. Siehe hierzu das Kapitel 9.2.

Zum Vergleich werden die ausgewählten Metriken verwendet, wie sie im Jupyter Notebook Prototypen angeboten werden. Um möglichst vergleichbare Ergebnisse zu erhalten, werden hierzu spezielle Skripte verwendet, welche die jeweiligen Metriken automatisch auf Vergleichbare Weise berechnen, wie dies im Notebook durch Interaktion des Benutzers geschehen würde. Diese Skripte sind nicht für eine konkrete, interaktive Analyse gedacht sondern nur strikt für Evaluationszwecke. Jedes Skript stellt dabei ein Experiment dar, dass Daten einliest, die Metrik berechnet und die Ergebnisse in eine Datei schreibt. Dies ist notwendig, um eine Vergleichbarkeit mit Spark zu erreichen. Eine Spark-Applikation arbeitet im allgemeinen nach folgendem Muster: Nachdem ein Job beim Spark Master eingereicht wurde, wird dieser an Worker im Cluster verteilt. Diese lesen die Daten ein und verarbeiten sie. Die Ergebnisse stehen nach Fertigstellung im HDFS-Dateisystem in Form von Dateien zur Verfügung. Es sollte also auch das Einlesen und Schreiben der Daten in der Zeitmessung berücksichtigt werden. Für die Evaluation wurde die Zeit gemessen, die die Metriken bis zur Bereitstellung des Ergebnisses benötigen.

9.2 Experimentaufbau

Für die Experimente wurde folgender Aufbau verwendet:

In der Weboberfläche der bereitgestellten Openstack Cloud-Plattform ist es unter dem Punkt *Data Processing* möglich, ein Cluster einzurichten. Hierzu müssen Vorlagen konfiguriert werden. Unter dem Unterpunkt *Node Group Templates* wurden hierzu über den Button *Create Template* zwei Vorlagen erstellt, jeweils für den Master und die Worker. Zuerst muss das Plugin und die Version ausgewählt werden, hier Apache Spark in der Version 1.3.1. Danach muss im Konfigurationsfenster das entsprechende Flavor ausgewählt werden. Dieses Flavor legt fest, welche Spezifikation die einzelnen Knoten des Clusters haben sollen, also die Anzahl der CPUs, RAM, Speicherplatz etc. Für die Evaluation wurde das Flavor *m1.large* gewählt, und zwar sowohl für den Master als auch für die Worker. Dieses Flavor bietet 4 CPUs, 8 GB Ram und 80 GB Speicherplatz. Unter dem Reiter *Node Processes* müssen bei der Vorlage für den Worker die Prozesse *datanode* und *slave* ausgewählt werden. Für den Master sind es *namenode* und *master*. Im *Configure Node Group Template* wurde die Option *Auto-configure* verwendet, ansonsten wurden keine weiteren Konfigurationsschritte vorgenommen.

Danach wird unter dem Unterpunkt *Cluster Templates* eine Vorlage für das Cluster erstellt. Auch hier wird im entsprechenden Konfigurationsdialog die Option *Auto-configure* verwendet[8]. Unter dem Reiter *Node Groups* können die im Schritt zuvor erstellten Vorlagen hinzugefügt werden. Für die Experimente werden ein Master und 4 Worker verwendet. Ansonsten wurden keine weiteren Änderungen an den Standardwerten vorgenommen. Nachdem die Vorlage erstellt wurde, kann das Cluster mit *Launch Cluster* gestartet werden. Hierzu muss das Image ausgewählt werden[9]. Unter Keypair muss ein Schlüsselpaar angegeben werden, mit dem man Zugriff auf die Maschinen im Cluster erhält. Unter Compute->Access & Security, Reiter Key Pairs können Schlüssselpaare verwaltet werden. Um sich auf den Maschinen via SSH einloggen zu können werden diese Schlüssel benötigt, eine Authentifizierung über ein Passwort ist zu Beginn nicht möglich.

Die Zeitmessung für die regulären Metriken erfolgte auf dem Demonstrationssystem. Hierbei handelt es sich um eine Virtuelle Maschine mit dem gleichen Flavor (*m1.large*) wie die Knoten das Spark-Clusters. Das Betriebssystem ist Windows Server 2008[10].

Die Testdaten werden dem NHTSA Complaints Datensatz entnommen. Für die Validierungsmetrik wurden exemplarisch alle Einträge der Spalte 8 verwendet. Diese Spalte heißt FAILDATE und enthält das Datum, an dem der Vorfall passiert ist. Im Datensatz ist diese Spalte ein String mit 8 Zeichen. Dieser repräsentiert das Datum im Format „YYYYMMDD“. Dies bedeutet, dass die ersten vier Zeichen das Jahr angeben. Zeichen 5 und 6 stellen den Monat dar, und die letzten beiden Zeichen den Tag. Mit dieser Metrik wird überprüft, ob dieses Format eingehalten wird. Dieser Datensatz hat eine Größe von ca. 12MB.

8 Zusätzlich wurden unter dem Reiter General Parameters die Optionen *Enable NTP Service* und *Enable Swift* deaktiviert, weil sich das Cluster ansonsten nicht starten lies.

9 Verwendet wurde für die Experimente das Image sahara-liberty-spark-1.3.1-ubuntu-14.04.

10 Das verwendete Image ist windows_2008_r2

Für die Textähnlichkeit wird dabei Spalte 20 verwendet. Bei dieser Spalte handelt es sich um ein Freitextfeld, das einen Kommentar zur jeweiligen Beschwerde enthält. Für die Textähnlichkeit werden drei Experimente durchgeführt. Einmal wird die gesamte Spalte als Textdatensatz verwendet. Dieser Text ist sehr groß, knapp 600MB. Für das zweite Experiment wurde exemplarisch ein kleiner Datensatz verwendet. Dieser besteht aus knapp 4500 Einträgen. Hierzu wurden beispielhaft nur die Einträge vom Januar 2005 verwendet. Dieser hat eine Größe von etwa 2MB.

9.3 Verwendete Textkorpora

Für die Berechnung der Textähnlichkeitsmetrik muss mindestens ein Trainingsdatensatz vorliegen. Siehe hierzu Abschnitt 6.3.4. Für die Anwendung der Metrik in dieser Arbeit werden deshalb 6 Trainingskorpora beispielhaft verwendet. Diese sollen im folgenden genannt und kurz beschrieben werden:

Der **Brown** Korpus besteht aus einer großen Sammlung von Texten aus den 60er Jahren, die in den Vereinigten Staaten im Druck erschienen sind. Der Korpus ist in verschiedene Kategorien unterteilt. In dieser Arbeit werden die Kategorien **Fiction** und **Review** verwendet. Diese bilden jeweils einen Datensatz. [34]

Der **CoNLL** Korpus enthält Texte aus dem Wall Street Journal. Enstanden ist er im Rahmen der Conference on Computational Natural Language Learning im Jahre 2000. [35]

Der **NPS Chat** Datensatz besteht aus Nachrichten eines Onlinechats. Diese wurden im Jahre 2006 gesammelt und aus Datenschutzgründen anonymisiert. [11]

Der **Twitter** Korpus basiert auf Daten der Social Media Platform Twitter[11]. Hierzu wurden Tweets gesammelt und aufbereitet [12]. Die Daten sind unter [51] verfügbar.

Siehe hierzu auch die Anwendung des Notebooks auf Daten aus der Praxis in Kapitel 8.

9.4 Ergebnisse

Tabelle 25 zeigt die Ergebnisse für die Validierungsmetrik. Die Spalten Notebook und Spark enthalten die Laufzeit für die jeweilige Version. Die Spalte Faktor gibt an, um welchen Faktor die Implementierung in Spark schneller ist als die Standardversion.

	Laufzeit		
	Notebook	Spark	Faktor
Validierung	18s	5s	3,6

Tabelle 25: Laufzeiten für die Validierungsmetrik für den kompletten Datensatz aus Spalte 8

11 https://twitter.com/

Tabelle 26 zeigt die Ergebnisse für die Textähnlichkeitsmetrik. Die Spalte Datensatz gibt an, welcher Datensatz verwendet wurde. Siehe hierzu den vorgehenden Abschnitt 9.2 für Details. Die Spalte Notebook enthält die Ergebnisse für die reguläre Implementierung, wie sie im Notebook verwendet wird. Die Spalte Spark enthält die Werte für die Implementierung mit Spark. Bei den Werten handelt es sich um die Laufzeit mit Angabe der Einheit. Die Textähnlichkeitsmetrik für Spark liegt in zwei Varianten vor, Standard und Variante 2. Variante 2 verwendet einen anderen Tokenizer. Der Tokenizer der Standard-Variante benötigt ein Modell, das aus einer Datei geladen werden muss. Siehe hierzu die Implementierungsdetails in Abschnitt 7.1.2. Die reguläre Metrik gibt es nur in einer Variante, siehe hierzu Abschnitt 6.3.4. Diese verwendet einen vergleichbaren Tokenizer wie die Standard-Variante für Spark. Es gibt für Variante 2 keine direkt vergleichbare Metrik im Notebook, deshalb wird kein Wert angegeben. Siehe hierzu auch die Diskussion in Abschnitt 9.5. Die Spalte Faktor gibt an, um welchen Faktor die Laufzeit durch die Implementierung in Spark verbessert werden konnte. Für die vereinfachte Metrik bezieht sich diese Spalte auf die Laufzeit der Standardimplementierung für Spark.

Textähnlichkeitsmetrik		Laufzeit		
	Datensatz	Notebook	Spark	Faktor
Standard	2MB	24s	23s	1,04
Variante 2	2MB	-	11s	2,1
Standard	600MB	180min	Fehler	
Variante 2	600MB	-	Fehler	

Tabelle 26: Laufzeiten für die Textähnlichkeitsmetrik

Beide Textähnlichkeitsmetriken konnten für den großen Textdatensatz kein Ergebnis berechnen. Die Spark Weboberfläche gab den Zustand des Jobs als „gescheitert“ an. Eine Analyse ergab, dass dies durch eine Exception verursacht wurde, die auf Speicherprobleme hindeutet. Siehe hierzu auch Abschnitt 9.5.

9.5 Analyse

In diesem Abschnitt sollen die zuvor in 9.4 präsentierten Ergebnisse analysiert und diskutiert werden.

Die Ergebnisse für die Validierungsfunktion können Tabelle 25 entnommen werden. Die Variante im Notebook benötige 18s. Die Implementierung in Spark lieferte das Ergebnis in 5s, eine Verbesserung um Faktor 3,6. Eine mögliche Erklärung für das sehr gute Ergebnis ist, das sich dieses Problem leicht verteilt bearbeiten lässt. Der Datensatz enthält über 1 Million Einträge. Die Metrik muss für jeden Wert prüfen, ob dieser dem Format entspricht. Allerdings kann dies unabhängig von den anderen Werten geschehen. So kann das Problem leicht auf mehrere Worker verteilt werden. Dieses Beispiel zeigt, dass die Verwendung von Spark bei großen Datensätzen zu einem deutlichen Gewinn an Performance führen kann. Zu beachten

ist, dass der Cluster in Standard-Konfiguration verwendet wurde. Auch an der Metrik selbst wurden keine komplexen Optimierungen durchgeführt, etwa durch Anpassung diverser von Spark angebotener Parameter. Es wäre denkbar, dass noch deutlich bessere Laufzeiten erzielt werden können, falls entsprechende Schritte unternommen werden.

Die Ergebnisse für die Textähnlichkeitsmetrik werden in Tabelle 26 dargestellt. Die Ergebnisse für den kleinen Datensatz sind vielversprechend. Die Standardvariante benötigt in Spark 23s und ist damit etwas schneller als die Standardimplementierung im Notebook mit 24s. Variante 2 ist mit 11s deutlich schneller, und zwar um Faktor 2,1. Dies liegt daran, dass ein anderer Tokenizer verwendet wird. Siehe Abschnitt 2.5.1 im Grundlagenkapitel für eine kurze Einführung zu Tokenizern. Dieser benötigt zur Instanziierung kein Modell aus einer externen Ressource. Im Gegensatz dazu muss der Maximum Entropy Tokenizer (TokenizerME) ein Modell aus einer Datei laden, bevor er verwendet werden kann. Diese Datei ist mehrere Megabyte groß. Durch die Verwendung eines anderen Tokenizers unterscheidet sich das Ergebnis der Metrik teilweise von der Standardimplementierung. Dadurch ist es möglich, dass die Qualität beeinträchtigt wird. Aus diesem Grund wird diese Implementierung nur für Spark zu Demonstrationszwecken verwendet. Es gibt keine vergleichbare Implementierung für das Notebook. Beide Tokenizer stellen kein Serializable Interface zur Verfügung. Dieses wird von Spark aber benötigt, falls das Objekt nur ein mal instanziiert werden soll. Eine Spark Applikation im Cluster-Modus wird verteilt auf mehreren Workern bearbeitet. Durch dieses Interface kann das Objekt allen Workern zur Verfügung gestellt werden. Eine Alternative ist es, eine statische Funktion zu verwenden. Innerhalb dieser Funktion wird das Objekt instanziiert. Somit ist es möglich, auch Objekte ohne Implementierung des Serializable Interfaces zu verwenden. Allerdings wird das Objekt dann bei jedem Aufruf der Funktion neu erzeugt. Bei Objekten, deren Instanziierung aufwendig sind, etwa beim TokenizerME, der eine externe Datei laden muss, führt dies zu Performanzproblemen. Im Fall der Textähnlichkeitsmetrik wird der Tokenizer einmal pro Textdatensatz aufgerufen. In diesem Fall also insgesamt 7 mal. Es wurden für diese Evaluation beispielhaft 6 Trainingssätze verwendet, hinzu kommen die Eingabedaten. Dadurch ist die Implementierung insgesamt trotzdem etwas schneller also die Standardversion im Notebook. Variante 2 verwendet einen Tokenizer, der viel schneller instanziiert werden kann. Dadurch ist die Laufzeit auch deutlich besser.

Dieses Problem ist auch der Grund, warum die Part-Of-speech Metrik nicht in Spark implementiert werden konnte. Siehe für diese Metrik Abschnitt 6.3.6. Hierzu wird ein Part-of-Speech Tagger benötigt, der ähnlich wie der TokenizerME ein Modell aus einer externen Datei lädt. Bei dieser Metrik müsste allerdings die Funktion für jede Zeile des Datensatzes aufgerufen werden. Beim NHTSA Complaints Datensatz sind das mehr als 1,3 Millionen Zeilen. Die Modelldatei für den Part-of-speech tagger müsste dann 1,3 Millionen mal geladen werden. In diesem Fall wäre eine solche Implementierung nicht sinnvoll.

Es stellt sich die Frage, ob es möglich wäre, das Serializable Interface nachträglich zu implementieren. Dabei wäre auch zu prüfen, mit welchem Aufwand dies verbunden wäre. Zusätzlich muss geklärt werden, wie sich das auf die Performance auswirken könnte. Die Spark Dokumentation zum Tuning [49] erwähnt, dass die Serialisierung in Java oft langsam sein kann.

Für den kompletten Textdatensatz konnte die Textähnlichkeitsmetrik in Spark kein Ergebnis erzielen. Dies gilt auch für Variante 2. Der Job wurde durch Spark abgebrochen, und der Zustand auf „gescheitert“ gesetzt. In den Logs wird als Ursache die folgende Exception genannt:

java.lang.OutOfMemoryError: Java heap space

Offenbar gibt es Probleme mit dem Speicher. Die Dokumentation von Spark bietet eine Seite, die Informationen zum Tuning enthält [49]. Entsprechend dieser Dokumentation können Java-Objekte deutlich mehr Speicherplatz verbrauchen, als die reinen Werte, die sie repräsentieren. Dies gilt insbesondere für Strings. Es wird empfohlen, Arrays anstelle der Java Collection Classes zu verwenden. Zusätzlich sollen möglichst primitive Datentypen verwendet werden. Die aus DKPro Similarity eingebundene Textähnlichkeitsmetrik erwartet als Eingabe einer Liste von Strings für jeden der beiden Texte, die miteinander verglichen werden sollen. Jedes Element dieser Liste stellt einen Token dar. Ein Token ist dabei ein Wort oder eine kleine Gruppe von Worten, die zusammengehören. Bei einem großen Text muss also eine Liste mit sehr vielen Elementen verwaltet werden. Zusätzlich handelt es sich bei jedem dieser Elemente um einen String. Jeder der Spark Worker verfügt über 8 GB Arbeitsspeicher. Die Standard-Variante des Notebooks wurde auf einem Rechner mit gleichen Ressourcen ausgeführt und war in der Lage, ein Ergebnis zu berechnen. Allerdings dauerte dies für den gesamten Datensatz 3 Stunden. Offenbar erschweren Besonderheiten von Spark die Berechnung dieser Metrik für große Datensätze.

Eine mögliche Lösung des Problems könnte es sein, die Textähnlickeitsmetrik neu zu implementieren. Dabei müsste statt einer Liste von Strings eine Darstellung gewählt werden, welche die verteilte Ausführung von Spark-Applikationen berücksichtigt. Eine solche Implementierung übersteigt jedoch bei weitem den Umfang der Arbeit.

9.6 Zusammenfassung

In diesem Kapitel wurden Experimente durchgeführt, mit der die Laufzeit ausgewählter Metriken ermittelt wurde. Dabei wurde verglichen, in wie weit sich die Laufzeit durch eine Verwendung von Spark reduzieren lässt. Für die Validierungsmetrik konnte bereits durch eine einfache Implementation in Spark in ein deutlicher Performancegewinn erzielt werden. Die Version im Notebook, die auf Python basiert, benötigte 18s, die Variante in Spark nur 5. Eine Verbesserung um den Faktor 3,6.

Für die Textähnlichkeit konnte für einen kleinen Beispieldatensatz eine leichte Verbesserung erzielt werden. Hier ist die Standardimplementierung mit 24s etwas langsamer als die Variante in Spark mit 23s. Um diese Metrik berechnen zu können, ist ein Tokenizer nötig, ein Machine Learning Algorithmus der ein Modell aus einer externen Datei laden muss. Die verwendete Bibliothek bietet für diesen Tokenizer kein Serializeable Interface an. Dadurch muss er in Spark für jeden Textdatensatz neu instanziiert werden. In diesem Experiment 7 mal. Eine Alternative Implementierung verwendet einen Tokenizer, der deutlich einfacher instanziiert werden kann. Diese Implementierung ist mit 11s deutlich schneller als die Standard-Vairante in Spark. Da der Tokenizer ein anderes Ergebnis liefert, kann diese Implementierung aber nicht direkt übernommen werden. Es sollte geprüft werden, ob die Qualität den Anforderungen genügt. Dieses Beispiel zeigt aber, dass eine deutliche Performancesteigerung möglich sein könnte. Hierzu muss das zuvor erwähnte Problem der Serialisierbarkeit gelöst werden. Dieses Problem betrifft auch viele weitere Objekte, etwa den Part-of-Speech tagger. Auch dieser muss ein Modell aus einer externen Quelle laden. Kann das Objekt nicht serialisiert und

an die Worker geschickt werden, so muss es bei jedem Aufruf neu instanziiert werden, was zu massiven Performance-Verlusten führt. Als mögliche Alternative wäre denkbar, diese Metriken in Anlehnung an die Implementierungen in den jeweiligen Bibliotheken neu zu entwickeln. Dabei könnten die Besonderheiten von Spark beim Entwurf von Anfang an berücksichtigt werden. Eventuell wäre es dadurch möglich, auch das Problem der Textähnlichkeitsmetrik mit sehr großen Texten zu lösen. Die Verwendung des vollen Textdatensatzes aus NHTSA Complaints führte bei beiden Varianten zu Fehlern wegen Speicherproblemen. Dies hängt vermutlich mit der internen Implementierung der verwendeten Textähnlichkeitsmetrik in DK-Pro Similarity [2] zusammen. Diese verwendet Datentypen, Listen und Strings, die in Spark zu Problemen führen können [49].

10 Fazit und Ausblick

In dieser Arbeit wurde ein Jupyter Notebook für das Data Wrangling entworfen und prototypisch implementiert. In der Literatur wird eine Vielzahl von Problemen und möglichen Ansätzen im Kontext des Data Wranglings diskutiert. Einen Überblick darüber findet sich in Kapitel 4. Darauf aufbauend wird in Kapitel 5 das Konzept präsentiert.

Das entwickelte Notebook erlaubt es, Daten zu inspizieren und zu transformieren. Verschiedene Datenqualitätsmetriken ermöglichen es die Datenqualität zu messen. Dies kann zusätzlich dazu beitragen, potentielle Datenqualitätsprobleme zu diagnostizieren. Einige Metriken wurden dabei speziell für das Notebook implementiert. Bereits existierende Metriken in den Sprachen Python, Java und R wurden in das Notebook eingebunden. Siehe hierzu das Kapitel 6. Dies zeigt die große Flexibilität des Ansatzes.

Das Notebook wurde auf Datensätze aus der Praxis angewendet. Diese sind der NHTSA Complaints Datensatz und ein Datensatz aus der Produktion. Hierzu wird in Kapitel 8 zunächst ein Überblick über die Struktur und, soweit wie möglich, die Semantik der Daten gegeben. Dies geschieht durch die Anwendung des Notebooks auf die Datensätze. Dabei wird auch die Datenqualität analysiert. Hierzu wurden passende Metriken berechnet. Das Notebook ist als flexibler Werkzeugkasten gedacht, der für verschiedene Aufgabenstellungen eingesetzt werden kann. Nicht immer ist jede Metrik anwendbar. Beispielsweise enthielt nicht jede Tabelle der Industriedaten Textfelder. Eine Berechnung der Textähnlichkeitsmetriken war deshalb für diese Tabellen nicht notwendig. Verschiedene Beispiele zeigen mögliche Datenqualitätsprobleme. Das Notebook erlaubt es, jederzeit, auch während der Analyse neue Funktionalitäten zu integrieren. Beispielsweise erlaubt die Standardimplementierung die Analyse von Textdaten, die einer Spalte entnommen wurden. Für die Industriedaten sollten aber aus zwei Spalten ein Textdatensatz gebildet werden. Zusätzlich mussten Duplikate entfernt werden, weil diese Spalten gegebenenfalls die gleichen Werte enthielten. Diese Funktion konnte per Skript implementiert und die Analyse durchgeführt werden. Siehe hierzu Abschnitt 8.2 für eine weitergehende Beschreibung und für die Ergebnisse. Insgesamt zeigt die Anwendung auf Datensätzen aus der Praxis, dass das Konzept des Notebooks vielversprechend ist. Die Flexibilität des Ansatzes konnte bestätigt werden.

Wegen des begrenzten Umfangs konnten viele in der Literatur diskutierte Ansätze nicht berücksichtigt werden. Beispielsweise wäre es wünschenswert, durchgeführte Transformationen sofort rückgängig machen zu können. Dies wird zum Beispiel in [17] diskutiert und implementiert. Ein weitergehender Aspekt, der nicht berücksichtigt werden konnte, ist der Umgang mit „dirty“ Data. Siehe hierzu die Diskussion in 4.3. Für manche diagnostizierten Datenqualitätsprobleme gibt es keine Lösung. Fehlende Werte etwa können darauf zurückzuführen sein, das Teile der Daten für immer verloren sind. Beispiel hierfür wäre ein Feuer in einem Archiv. Die Tatsache, das Daten fehlen, stellt ein Qualitätsproblem dar. Eventuell ist es möglich, Fehlinterpretationen entgegenzuwirken, wenn dem Konsumenten diese Tatsache auf passende Weise kommuniziert wird. Zum Beispiel durch entsprechende Visualisierungen, die deutlich machen, das Daten fehlen. Siehe hierzu die Diskussion in Abschnitt 4.3. Entsprechende Funktionen in das Notebook zu integrieren könnte ein vielversprechender Ansatz sein.

Die Anwendung des Notebooks soll auch auf große Datensätze möglich sein. Dabei stellt sich die Frage nach der Performance und der Skalierbarkeit der verwendeten Metriken. Hierzu wurde exemplarisch geprüft, ob sich die Laufzeit der Metriken durch eine Implementierung mit Spark verbessern lässt.

Hierzu wurden die Metriken Validierung und Textähnlichkeit ausgewählt. Die Validierungsmetrik arbeitet auf strukturierten Daten. Die Textähnlichkeitsmetrik ist ein Beispiel für die Qualitätsmessung von unstrukturierten Daten. Die Ergebnisse für die Validierungsmetrik sind sehr gut. Die Laufzeit konnte bereits durch eine einfache Implementierung in Spark um den Faktor 3,6 verbessert werden. Dies liegt vermutlich daran, dass sich diese Problemstellung gut in Spark abbilden lässt.

Die Ergebnisse für die Textähnlichkeitsmetrik sind vielversprechend. Die Textähnlichkeitsmetrik war für einen kleineren Datensatz mit 23s leicht schneller als die im Notebook verwendete Version mit 24s. Eine abgewandelte Variante konnte ein Ergebnis in 11s berechnen. Allerdings muss eine Reihe von Problemen gelöst werden. Beispielsweise konnte die Metrik für den kompletten Datensatz im Cluster-Modus nicht berechnet werden. Die Analyse deutet auf Speicherprobleme hin. Die reguläre Version im Notebook war in der Lage, ein Ergebnis zu berechnen. Die Ursache ist möglicherweise, dass die zur Berechnung der Textähnlichkeit genutzte Bibliothek DKPro Similarity [2] auf Listen von Strings arbeitet. Laut der Dokumentation von Spark ist dies ungünstig [49]. Eine Mögliche Lösung könnte es sein, die Berechnung der Textähnlichkeit speziell für Spark neu zu implementieren. Siehe Kapitel 9.5 für eine detaillierte Analyse der Ergebnisse.

Insgesamt ist die Verwendung eines Jupyter Notebooks für Data Wrangling vielversprechend. Der entwickelte Prototyp hat sich bei der Anwendung auf Datensätze aus der Praxis bewährt. Durch die Flexibilität des Ansatzes können Anpassungen und Erweiterungen leicht durchgeführt werden.

Literaturverzeichnis

[1] Bär, Daniel; Gurevych, Iryna; Dagan, Ido; Zesch, Torsten (2013): A Composite Model for Computing Similarity Between Texts. Darmstadt: Universitäts- und Landesbibliothek Darmstadt.

[2] Bär, D., Zesch, T. and Gurevych, I. (August 2013). DKPro Similarity: An Open Source Framework for Text Similarity. In ACL (Conference System Demonstrations), 121-126

[3] Batini, Carlo; Scannapieco, Monica (2016): Data and Information Quality. Dimensions, Principles and Techniques, Seiten 1-72, 113-134, 337-352. 1st ed. 2016. s.l.: Springer-Verlag (Data-Centric Systems and Applications).

[4] Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc.

[5] Boehmke, B. (2016). Data Wrangling with R. Springer.

[6] Breunig, M.M., Kriegel, H.P., Ng, R.T. and Sander, J.(Mai 2000). LOF: identifying density-based local outliers. In *ACM sigmod record* (Vol. 29, No. 2, 93-104). ACM.

[7] Chessell, M., Scheepers, F., Nguyen, N., van Kessel, R. and van der Starre, R. (2014). Governing and managing big data for analytics and decision makers. IBM Redguides for Business Leaders.

[8] de Castilho, R.E. and Gurevych, I. (August 2014). A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT (OIAF4HLT) at COLING, 1-11.

[9] Endel, F. and Piringer, H. (2015). Data Wrangling: Making data useful again. IFAC-PapersOnLine, 48(1), 111-112.

[10] Fayyad, U.M., Piatetsky-Shapiro, G. and Smyth, P. (August 1996). Knowledge Discovery and Data Mining: Towards a Unifying Framework. In KDD (Vol. 96, 82-88).

[11] Forsyth, Eric N. and Martell, Craig H. (September 2007). "Lexical and Discourse Analysis of Online Chat Dialog," Proceedings of the First IEEE International Conference on Semantic Computing (ICSC 2007), pp. 19-26

[12] Gimpel, K., Schneider, N., O'Connor, B., Das, D., Mills, D., Eisenstein, J., Heilman, M., Yogatama, D., Flanigan, J. and Smith, N.A. (Juni 2011). Part-of-speech tagging for twitter: Annotation, features, and experiments. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2, 42-47. Association for Computational Linguistics.

[13] Gröger, C. (2015). Advanced Manufacturing Analytics. Lohmar, Köln: Eul-Verl.

[15] Jindal, N. and Liu, B., (Februar 2008). Opinion spam and analysis. In Proceedings of the 2008 International Conference on Web Search and Data Mining, 219-230. ACM.

[16] Kandel, S., Heer, J., Plaisant, C., Kennedy, J., van Ham, F., Riche, N.H., Weaver, C., Lee, B., Brodbeck, D. and Buono, P. (2011). Research directions in data wrangling: Visualizations and transformations for usable and credible data. Information Visualization, 10(4), 271-288.

[17] Kandel, S., Paepcke, A., Hellerstein, J. and Heer, J. (Mai 2011). Wrangler: Interactive visual specification of data transformation scripts. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 3363-3372. ACM.

[18] Kiefer, Cornelia (2016): Assessing the Quality of Unstructured Data: An Initial Overview. In: Ralf Krestel, Davide Mottin und Emmanuel Müller (Hg.): Proceedings of the LWDA 2016 Proceedings (LWDA). Aachen (CEUR Workshop Proceedings), 62–73.

[19] Laukart, Andreas (2016): Fit of training data and text data – automatic identification of the best fitting training data. Studienarbeit. Universität Stuttgart. Institut für Parallele und Verteilte Systeme.

[20] Li, J., Tao, F., Cheng, Y. and Zhao, L. (2015). Big data in product lifecycle management. The International Journal of Advanced Manufacturing Technology, 81(1-4), 667-684.

[21] McAfee, A., Brynjolfsson, E., Davenport, T.H., Patil, D.J. and Barton, D. (2012). Big data. The management revolution. Harvard Bus Rev, 90(10), 61-67.

[22] McCallum, Q. (2012). Bad Data Handbook, Seiten 31-51, 53-68, 83-93, 129-140. O'Reilly Media, Inc.

[23] McKinney, W. (2012). Python for Data Analysis. O'Reilly Media, Inc.

[24] Kelly, Ryan, PyEnchant Website, http://pythonhosted.org/pyenchant/, Zugriff am 23.03.2017

[25] Savinov, A. (2016). DataCommandr: Column-Oriented Data Integration, Transformation and Analysis. In Proc. International Conference on Internet of Things and Big Data (IoTBD 2016), 339-347.

[26] Sebastian-Coleman, L. (2012). Measuring data quality for ongoing improvement: a data quality assessment framework. Seiten S 3-15, 39-53. Newnes.

[27] Signell, R.P., Fernandes, F. and Wilcox, K. (2016). Dynamic reusable workflows for ocean science. Journal of Marine Science and Engineering, 4(4), 68.

[28] Wand, Y. and Wang, R.Y. (1996). Anchoring data quality dimensions in ontological foundations. Communications of the ACM, 39(11), 86-95.

[29] Wang, R.Y. and Strong, D.M. (1996). Beyond accuracy: What data quality means to data consumers. Journal of management information systems, 12(4), 5-33.

[30] Yingsong Hu, Wayne Murray and Yin Shan, RLOF package, https://CRAN.R-project.org/package=Rlof, Zugriff am 23.03.2017

[31] Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J. and Ghodsi, A. (2016). Apache Spark: a unified engine for big data processing. Communications of the ACM, 59(11), 56-65.

[32] Anaconda Webseite, https://www.continuum.io/anaconda-overview, Zugriff am 23.03.2017

[33] Apache Spark Homepage, http://spark.apache.org/, Zugriff am 23.03.2017

[34] Brown Korpus Handbuch, http://www.hit.uib.no/icame/brown/bcm.html, Zugriff am 23.03.2017

[35] CoNLL2000 Korpus, http://www.cnts.ua.ac.be/conll2000/chunking/, Zugriff am 23.03.2017

[36] „Daten“ auf Duden online, www.duden.de/rechtschreibung/Daten, Zugriff am 23.03.2017

[37] Data Wrangling with Python Repository, https://github.com/aepton/python-data-wrangling, Zugriff 23.03.2017

[38] Data Wrangler App, http://vis.stanford.edu/wrangler/app/, Zugriff am 23.03.2017

[39] Installationsanleitung für Jupyter, ,http://jupyter.org/install.html, Zugriff am 23.03.2017

[40] Openstack Homepage, http://www.openstack.org/ , Zugriff 23.03.2017

[41] NHTSA Complaints Datensatz, http://www-odi.nhtsa.dot.gov/downloads/, Zugriff am 18.03.2017

[42] Open Office Homepage, https://www.openoffice.org/, Zugriff am 23.03.2017

[43] OpenNLP Homepage, https://opennlp.apache.org/, Zugriff am 23.03.2017

[44] OpenNLP Models, http://opennlp.sourceforge.net/models-1.5/, Zugriff am 23.03.2017

[45] OpenRefine Homepage, http://openrefine.org/index.html, Zugriff am 23.03.2017

[46] Pandas Homepage, http://pandas.pydata.org/, Zugriff am 23.03.2017

[47] Projet Jupyter Homepage, http://jupyter.org/, Zugriff am 23.03.2017

[48] rpy2 Homepage, https://rpy2.bitbucket.io/, Zugriff am 23.03.2017

[49] Spark Tuning Page für Version 1.3.1, https://spark.apache.org/docs/1.3.1/tuning.html, Zugriff am 23.03.2017

[50] Trifacta homepage, https://www.trifacta.com/, Zugriff am 19.03.2017

[51] Twitter corpus repository, https://github.com/brendano/ark-tweet-nlp/, Zugriff am 23.03.2017

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Stuttgart, den _____