Think BIGGER

DEEP DIVE FÜR FACHLEUTE

Datenbereinigung und ihre Bedeutung für maschinelles Lernen

Warum Datenqualität über Genauigkeit, Fairness, Stabilität und Wartbarkeit von ML-Systemen entscheidet

Autor	Erhard Rainer
Veröffentlicht	26.02.2026
Aktualisiert	06.07.2026
Umfang	ca. 39 Min. Lesezeit
Online	www.thinkbigger.at/wissen/datenbereinigung-maschinelles-lernen/

thinkbigger.at

Business Intelligence · Machine Learning · KI-Agenten · Schulungen

Erhard Rainer · Schönaugasse 106 · 8020 Graz · info@think-bigger.at

Inhalt

TL;DR 2
Warum Datenbereinigung bei Machine Learning so wichtig ist 3
Datenbereinigung, Datenqualität und Data Preparation 3
Typische Datenprobleme in ML-Projekten 4
Wie Datenfehler Modellarten unterschiedlich beeinflussen 9
Der Datenbereinigungsprozess als Pipeline 10
Methoden der Datenbereinigung im Detail 12
Feature Engineering und Datenbereinigung 16
Trainings-, Validierungs- und Testdaten sauber trennen 16
Datenqualität messen 17
Fairness, Bias und ethische Folgen schlechter Daten 18
Datenschutz und Datenminimierung 18
Automatisierung: hilfreich, aber nicht blind 19
Beispiel: Kundenabwanderung vorhersagen 19
Beispiel: Qualitätsprüfung in der Produktion 21
Beispiel: Dokumentensuche und generative KI 21
Checkliste: Datenbereinigung für ML-Projekte 22
Häufige Anti-Patterns 22
Datenbereinigung organisatorisch verankern 23
Praktisches Vorgehensmodell 24
Mini-Beispiel: Pipeline-Denken in Python 25
Qualitätsregeln als Tests 26
Wann Daten nicht bereinigt, sondern das Problem neu definiert werden muss 26
Verbindung zu Data Engineering und Datenarchitektur 27
Ein Entscheidungsrahmen für Cleaning-Strategien 27
Fazit 28
Weiterführende Literatur und Quellen 28

Datenbereinigung klingt nach einer lästigen Vorarbeit: ein paar fehlende Werte ergänzen, offensichtliche Tippfehler korrigieren, Duplikate entfernen, dann kann das eigentliche maschinelle Lernen beginnen. Diese Sicht ist verständlich, aber gefährlich. In realen ML-Projekten entscheidet die Datenbereinigung nicht nur darüber, ob ein Modell ein paar Prozentpunkte besser abschneidet. Sie entscheidet darüber, ob ein Modell überhaupt das richtige Problem lernt, ob es stabil bleibt, ob es fair eingesetzt werden kann und ob seine Ergebnisse fachlich erklärbar sind.

Maschinelles Lernen ist kein magischer Filter, der aus beliebig schlechten Daten zuverlässig Wahrheit gewinnt. Ein Modell erkennt statistische Muster in den Daten, die es bekommt. Sind diese Daten unvollständig, verzerrt, widersprüchlich, zeitlich falsch geschnitten oder fachlich falsch gelabelt, lernt das Modell genau diese Fehler mit. Manchmal zeigt sich das sofort in schlechten Kennzahlen. Häufiger ist es tückischer: Die Validierung sieht gut aus, aber das Modell versagt im Betrieb, weil während der Datenaufbereitung Information aus der Zukunft eingeflossen ist, weil Trainings- und Produktionsdaten unterschiedlich bereinigt wurden oder weil seltene, aber wichtige Fälle im Cleaning-Prozess weggefiltert wurden.

Dieser Artikel behandelt Datenbereinigung deshalb als eigenständige technische, fachliche und organisatorische Aufgabe. Er zeigt, welche Arten von Datenproblemen für Machine Learning besonders relevant sind, wie sie die Modellqualität beeinflussen, welche Bereinigungsstrategien sinnvoll sind, wo Automatisierung hilft und wo man sehr bewusst nicht automatisch korrigieren sollte. Der Schwerpunkt liegt auf tabellarischen Daten, weil dort viele typische ML-Projekte in Unternehmen beginnen. Viele Grundsätze gelten aber ebenso für Textdaten, Zeitreihen, Bilddaten, Logdaten, Sensordaten und Daten, die später in Vektordatenbanken oder generativen KI-Systemen verwendet werden.

TL;DR

Datenbereinigung ist im maschinellen Lernen kein kosmetischer Schritt, sondern Teil des Modellrisikos. Schlechte Daten führen nicht nur zu ungenaueren Modellen, sondern zu falschen Zielgrößen, unfairen Entscheidungen, instabilen Produktionssystemen, nicht reproduzierbaren Experimenten und irreführenden Erfolgskennzahlen.

Die wichtigsten Punkte:

- Datenbereinigung muss fachlich begründet werden. Nicht jeder ungewöhnliche Wert ist falsch, nicht jeder fehlende Wert ist zufällig und nicht jedes Duplikat darf gelöscht werden.
- Für ML ist die Reihenfolge entscheidend: Trainings-, Validierungs- und Testdaten müssen sauber getrennt bleiben. Cleaning darf keine Information aus dem Validierungs- oder Testset in das Training zurückspielen.
- Fehlende Werte, Ausreißer, Duplikate, Inkonsistenzen, falsche Datentypen, Label-Fehler, Datenlecks, unausgewogene Klassen und Drift betreffen unterschiedliche Ebenen des ML-Prozesses.
- Datenbereinigung und Feature Engineering gehören zusammen. Eine Imputation, Skalierung, Kodierung oder Normalisierung ist nicht neutral, sondern verändert den Lernraum des Modells.
- Gute Cleaning-Pipelines sind reproduzierbar, versioniert, testbar und erklärbar. Manuelle Excel-Korrekturen ohne Protokoll sind für produktive ML-Systeme ein Risiko.
- Automatisierte Profiling- und Validierungswerkzeuge helfen, ersetzen aber keine Domänenkenntnis. Die Fachfrage lautet immer: Was bedeutet dieser Wert in der Realität?
- Für produktive Systeme endet Datenbereinigung nicht beim Training. Datenqualität muss im Betrieb überwacht werden: Schemaänderungen, Verteilungsdrift, neue Kategorien, geänderte Prozesse und Label-Verzögerungen können ein ehemals gutes Modell beschädigen.

Die praktische Leitfrage lautet nicht: "Wie bekomme ich die Daten schnell sauber?" Besser ist: "Welche Datenfehler würden dazu führen, dass mein Modell das falsche Muster lernt, und wie kann ich diese Fehler reproduzierbar erkennen, behandeln und überwachen?"

Warum Datenbereinigung bei Machine Learning so wichtig ist

Klassische Software folgt expliziten Regeln. Wenn eine Eingabe ungültig ist, kann ein Programm sie ablehnen. Wenn ein Datumsfeld falsch formatiert ist, schlägt die Verarbeitung fehl. Bei Machine Learning ist die Lage subtiler. Ein ML-Modell muss nicht zwingend abstürzen, wenn Daten fehlerhaft sind. Es lernt trotzdem. Genau darin liegt das Risiko: Das Modell produziert ein Ergebnis, obwohl die Datenbasis fachlich falsch war.

Ein Regressionsmodell kann mit fehlerhaften Umsatzzahlen trainiert werden. Ein Klassifikationsmodell kann Kunden falsch als "gekündigt" oder "nicht gekündigt" gelabelt bekommen. Ein Fraud-Modell kann historische Prüfentscheidungen lernen, obwohl diese Prüfentscheidungen selbst verzerrt waren. Ein Forecasting-Modell kann eine Sonderaktion als wiederkehrendes Nachfrageverhalten interpretieren. Ein Sprachmodell-Workflow kann Dokumente indexieren, die veraltet, doppelt oder widersprüchlich sind. Das Modell liefert dann plausible, aber fachlich unzuverlässige Ergebnisse.

Datenbereinigung ist daher nicht nur ein Schritt vor dem Training. Sie ist die Übersetzung zwischen der realen Welt, den gespeicherten Daten und dem statistischen Lernproblem. In ihr wird entschieden:

- welche Beobachtungen überhaupt zum Problem gehören;
- welche Werte als gültig gelten;
- welche Fehler korrigiert, markiert oder ausgeschlossen werden;
- welche Information ein Modell sehen darf;
- wie fehlende oder unsichere Information repräsentiert wird;
- wie historische Daten zur späteren Produktionssituation passen;
- welche Fälle für Evaluation und Monitoring relevant bleiben.

Viele ML-Projekte scheitern nicht an einem zu einfachen Algorithmus, sondern an einem unklaren Datenverständnis. Ein Gradient-Boosting-Modell, ein neuronales Netz oder ein großer AutoML-Lauf kann keine saubere Problemdefinition ersetzen. Je leistungsfähiger die Modellklasse ist, desto stärker kann sie Artefakte aus schlechten Daten ausnutzen. Ein einfaches Modell scheitert sichtbar. Ein starkes Modell findet unter Umständen einen versteckten Datenfehler und macht daraus eine scheinbar hervorragende Validierungsleistung.

Ein bekanntes Beispiel ist Data Leakage: Das Modell bekommt während des Trainings ein Merkmal, das in der realen Vorhersagesituation noch gar nicht verfügbar wäre. Die Kennzahlen wirken exzellent, aber nur, weil die Datenbereinigung, Feature-Erzeugung oder Tabellenverknüpfung die zeitliche Logik verletzt hat. Solche Fehler sind keine Modellfehler im engeren Sinn. Sie entstehen in der Datenaufbereitung.

Der Artikel Machine Learning erklärt die Grundidee lernender Modelle. Die Datenbereinigung ist die vorgelagerte Disziplin, die dafür sorgt, dass diese Modelle nicht aus zufälligem Datenmüll, Prozessartefakten oder historischen Verzerrungen lernen.

Datenbereinigung, Datenqualität und Data Preparation

Die Begriffe werden oft durcheinander verwendet, meinen aber unterschiedliche Ebenen.

Begriff	Kernfrage	Typische Aktivitäten	ML-Relevanz
Datenqualität	Sind die Daten für einen Zweck geeignet?	Vollständigkeit, Korrektheit, Aktualität, Konsistenz, Eindeutigkeit prüfen	bestimmt, ob das Lernproblem belastbar abgebildet wird
Datenbereinigung	Wie behandeln wir erkannte Datenprobleme?	korrigieren, imputieren, ausschließen, markieren, zusammenführen	verändert Trainingsdaten und Modellverhalten direkt
Data Preparation	Wie machen wir Rohdaten modellierbar?	Join, Filter, Aggregation, Kodierung, Skalierung, Feature Engineering	erzeugt den finalen Lernraum
Data Validation	Wie stellen wir sicher, dass Annahmen eingehalten werden?	Schematests, Bereichsprüfungen, Verteilungschecks, referenzielle Regeln	verhindert stille Fehler in Training und Betrieb
MLOps Data Monitoring	Wie verändert sich Datenqualität nach Deployment?	Drift-Erkennung, neue Kategorien, Missingness, Qualitätsmetriken	schützt produktive Modelle vor schleichender Entwertung

Datenbereinigung ist also enger als Datenqualität, aber breiter als "Fehler löschen". Sie ist ein Set von Entscheidungen. Manche Entscheidungen sind technisch einfach, fachlich aber riskant. Ein negativer Lagerbestand kann ein Fehler sein, ein Hinweis auf verzögerte Buchungen oder ein reales Signal für Prozessprobleme. Ein Geburtsdatum im Jahr 1900 kann ein Defaultwert, eine echte Person, ein Datenschutzersatz oder ein historischer Datenimport sein. Ein Ausreißer im Umsatz kann ein Tippfehler oder der wichtigste Kunde des Jahres sein.

Für Machine Learning reicht es nicht, Daten "schön" zu machen. Ein zu aggressiver Cleaning-Prozess kann genau jene seltenen Ereignisse entfernen, die das Modell lernen soll. Das betrifft etwa Betrug, Maschinenausfälle, Kündigungen, Zahlungsausfälle, Reklamationen oder Sicherheitsvorfälle. In solchen Fällen ist die Minderheit der Daten fachlich besonders wichtig.

Der saubere Begriff wäre daher: **zweckgebundene Datenbereinigung**. Daten sind nicht abstrakt sauber oder unsauber. Sie sind für einen bestimmten Zweck ausreichend, problematisch oder ungeeignet.

Typische Datenprobleme in ML-Projekten

In der Praxis treten Datenprobleme selten isoliert auf. Ein Importfehler erzeugt fehlende Werte, falsche Datentypen und Duplikate. Eine Prozessänderung erzeugt neue Kategorien, geänderte Verteilungen und plötzlich schlechtere Labels. Eine fehlerhafte Tabellenverknüpfung erzeugt künstlich vervielfachte Datensätze. Trotzdem hilft eine systematische Typologie, weil jedes Problem andere Modellrisiken erzeugt.

Fehlende Werte

Fehlende Werte sind eines der häufigsten Datenprobleme. Sie entstehen durch nicht erfasste Angaben, technische Übertragungsfehler, unterschiedliche Pflichtfelder, manuelle Umgehungen, Datenschutzmaskierung, verspätete Datenlieferungen oder fachliche Nichtanwendbarkeit.

Für ML ist entscheidend, warum ein Wert fehlt:

Missingness-Typ	Beschreibung	Beispiel	Modellrisiko
zufällig fehlend	Fehlen hängt nicht systematisch mit dem Ziel zusammen	Sensor sendet sporadisch nicht	Datenverlust, höhere Unsicherheit
bedingt fehlend	Fehlen hängt von anderen Variablen ab	Einkommen fehlt häufiger bei Selbständigen	Verzerrung bestimmter Gruppen
informativ fehlend	Fehlen ist selbst ein Signal	Kunde verweigert bestimmte Angabe	Imputation kann wichtiges Signal zerstören
strukturell nicht anwendbar	Wert existiert fachlich nicht	Kündigungsdatum bei aktiven Kunden	falsche Nullwerte oder Scheinkorrelationen

Eine einfache Mittelwert-Imputation ist oft der erste Reflex. Sie kann sinnvoll sein, ist aber selten neutral. Sie reduziert Varianz, erzeugt künstlich typische Fälle und kann Gruppenunterschiede verwischen. Häufig besser ist eine Kombination aus Imputation und Missingness-Indikator: Das Modell bekommt einen Ersatzwert und zusätzlich die Information, dass der Originalwert fehlte. Bei Baumverfahren kann fehlende Information teilweise direkt verarbeitet werden; bei linearen Modellen, neuronalen Netzen oder Distanzverfahren muss man sie meist explizit behandeln.

Wichtig ist außerdem die Trennung der Datensplits. Der Imputationswert darf im Training nur aus dem Trainingsset gelernt werden. Wird der Mittelwert über den gesamten Datensatz berechnet, fließt Information aus Validierungs- oder Testdaten in das Training ein. Das klingt klein, ist aber genau die Sorte unsichtbarer Fehler, die Evaluationen beschönigt.

Falsche Datentypen und Formate

Viele Datenprobleme beginnen banal: Zahlen werden als Text importiert, Dezimaltrennzeichen wechseln zwischen Komma und Punkt, Datumsformate sind uneinheitlich, Währungen werden gemischt, Postleitzahlen verlieren führende Nullen, boolesche Werte erscheinen als `true`, `TRUE`, `1`, `Ja`, `Y` und `x`.

Für gewöhnliche Auswertungen fällt so etwas schnell auf. Für ML kann es subtiler sein. Wenn eine numerische Spalte als Kategorie behandelt wird, lernt das Modell eine völlig andere Struktur. Wenn eine Kategorie versehentlich numerisch kodiert wird, interpretiert ein lineares Modell möglicherweise eine künstliche Rangordnung. Wenn Datumswerte als Text sortiert werden, entstehen falsche Zeitfenster.

Besonders riskant sind semantische Typfehler: Eine Spalte sieht technisch korrekt aus, bedeutet aber fachlich nicht das, was ihr Name suggeriert. `created_at` kann das Erfassungsdatum, das Importdatum oder das Datum der ersten fachlichen Gültigkeit sein. `status` kann den aktuellen Status zeigen, nicht den historischen Status zum Vorhersagezeitpunkt. Solche Felder sind klassische Quellen für Data Leakage.

Inkonsistente Kategorien und Schreibweisen

Kategoriale Variablen sind in Unternehmensdaten selten sauber. Ein Produkt kann als `Notebook`, `Laptop`, `NB`, `Lap Top` oder über eine interne Produktgruppe erscheinen. Länder, Branchen, Abteilungen, Fehlercodes und Kundensegmente ändern sich über Jahre. Fusionen, Systemwechsel und manuelle Eingaben erzeugen Varianten.

Für ML führen solche Inkonsistenzen zu fragmentierten Signalen. Ein Modell lernt für jede Schreibweise getrennte Effekte, obwohl fachlich dasselbe gemeint ist. Bei seltenen Kategorien verschärft sich das Problem: Das Modell sieht zu wenig Beispiele je Variante. Für One-Hot-Encoding entstehen unnötig viele Spalten. Für Embeddings oder Zielkodierungen können scheinbar kleine Schreibweisen die Statistik verändern.

Bereinigung bedeutet hier nicht nur Kleinbuchstaben und Trim. Häufig braucht es Mapping-Tabellen, Referenzdaten, Synonyme, Gültigkeitszeiträume und klare Regeln für neue Kategorien. Eine gute Lösung trennt Rohwert und bereinigten Wert: Der Originalwert bleibt erhalten, die normalisierte Kategorie wird zusätzlich erzeugt.

Duplikate

Duplikate wirken harmlos, sind aber in ML-Projekten besonders gefährlich. Man unterscheidet technische Duplikate und fachliche Duplikate.

Technische Duplikate sind identische Zeilen, die etwa durch fehlerhafte Exporte oder Wiederholungen im ETL-Prozess entstehen. Sie können meist entfernt werden, wenn klar ist, dass keine Mehrfachbeobachtung gemeint ist.

Fachliche Duplikate sind schwieriger. Ein Kunde kann mehrfach vorkommen, weil es mehrere Verträge gibt. Eine Maschine kann mehrere Wartungen haben. Eine Rechnung kann mehrere Positionen enthalten. Ein Ereignis kann in verschiedenen Systemen gespiegelt werden. Löscht man hier "Duplikate" zu grob, verliert man reale Information.

Für ML ist zusätzlich relevant, ob Duplikate über Datensplits verteilt sind. Wenn dieselbe Beobachtung im Training und im Testset liegt, wirkt das Modell besser, als es ist. Bei Kundendaten, Dokumenten, Bildern oder Zeitreihen kann das auch in abgeschwächter Form auftreten: sehr ähnliche Fälle landen in verschiedenen Splits. Die Evaluation misst dann nicht Generalisierung, sondern Wiedererkennung.

Ein robustes Vorgehen prüft Duplikate daher vor dem Split und entscheidet, auf welcher Ebene gesplittet wird: Zeile, Kunde, Vertrag, Maschine, Dokument, Zeitraum oder Ereignis. Bei personenbezogenen oder entitätsbezogenen Daten ist ein gruppierter Split oft realistischer als ein zufälliger Zeilensplit.

Ausreißer

Ausreißer sind Beobachtungen, die deutlich vom üblichen Muster abweichen. Der bestehende Artikel zur Ausreißer-Erkennung im Forecasting behandelt das Thema bereits aus Zeitreihenperspektive. Für Datenbereinigung im ML-Kontext ist wichtig: Ein Ausreißer ist nicht automatisch ein Fehler.

Es gibt mehrere Ausreißerarten:

Art	Beispiel	Mögliche Behandlung
Messfehler	Temperaturwert von 900 Grad in einem Büro	korrigieren oder ausschließen
Eingabefehler	Umsatz wurde mit zwei Nullen zu viel erfasst	fachlich prüfen, korrigieren
seltenes reales Ereignis	Black-Friday-Umsatz, Maschinenausfall, Betrugsfall	behalten, markieren, separat modellieren
strukturelle Änderung	neues Produkt, neuer Markt, neue Regel	nicht als Fehler behandeln, Modellkonzept prüfen
Extremwert mit hoher Relevanz	Großkunde, Hochrisikofall, Störfall	explizit berücksichtigen

Aggressive Ausreißerentfernung kann Modelle beruhigen, aber auch blind machen. Ein Fraud-Modell, das extreme Transaktionen entfernt, lernt womöglich genau die Betrugsfälle nicht. Ein Wartungsmodell, das seltene Sensorwerte entfernt, verliert Hinweise auf bevorstehende Ausfälle. Ein Kreditrisikomodell, das wirtschaftliche Krisenjahre glättet, unterschätzt Risiko.

Statt reflexartig zu löschen, sollte man fragen:

- Ist der Wert technisch unmöglich oder nur ungewöhnlich?
- Gibt es eine plausible fachliche Erklärung?
- Betrifft der Ausreißer Eingangsmerkmal, Zielvariable oder beides?
- Würde der Wert in der Produktion wieder auftreten?
- Ist der Ausreißer Teil des Problems, das das Modell erkennen soll?

Häufig sind Markierungen besser als Löschungen: `is_outlier`, `is_promotional_period`, `is_sensor_error_candidate`, `was_capped`. So bleibt das Signal erhalten, aber das Modell kann es kontrollierter nutzen.

Fehlerhafte Labels

Label-Qualität wird unterschätzt. In überwachten Lernverfahren ist die Zielvariable die Antwort, aus der das Modell lernen soll. Ist diese Antwort falsch, lernt das Modell falsche Zusammenhänge.

Label-Fehler entstehen durch manuelle Fehlklassifikation, verzögerte Rückmeldungen, widersprüchliche Definitionen, historische Prozessänderungen, subjektive Einschätzungen oder Proxy-Labels. Ein Kunde gilt vielleicht als "nicht gekündigt", weil die Kündigung erst später im System erfasst wurde. Eine Reklamation gilt als "berechtigt", weil sie kulanzweise akzeptiert wurde. Ein Ticket gilt als "gelöst", obwohl der Kunde erneut Kontakt aufnimmt. Ein Bewerber gilt als "geeignet", weil ein historischer Auswahlprozess dies entschied, nicht weil das Label objektiv richtig ist.

Label-Fehler sind besonders kritisch, weil sie nicht durch bessere Modellarchitektur verschwinden. Ein Modell kann nicht konsistent besser werden als die Zieldefinition, auf der es trainiert. Bei sensiblen Anwendungen muss deshalb nicht nur die Feature-Qualität, sondern auch die Label-Herkunft geprüft werden.

Praktische Maßnahmen:

- Labeldefinition dokumentieren: Was genau bedeutet die Zielvariable?
- Labelzeitpunkt festlegen: Wann war das Label bekannt?
- Unsichere Labels markieren statt verschweigen.
- Stichproben fachlich nachprüfen.
- Mehrere Labelquellen vergleichen.
- Verzögerte Labels bei Monitoring und Retraining berücksichtigen.

Datenlecks

Data Leakage ist eines der gefährlichsten Probleme in ML-Projekten. Es bedeutet, dass das Modell im Training Information sieht, die in der echten Vorhersagesituation nicht verfügbar wäre oder die Zielvariable indirekt verrät.

Typische Beispiele:

- Ein Kündigungsmodell nutzt das Feld `cancellation_date`.
- Ein Kreditrisikomodell nutzt Mahnstufen, die erst nach dem Prognosezeitpunkt entstehen.
- Ein Diagnosemodell nutzt Abrechnungscodes, die nach der ärztlichen Entscheidung gesetzt wurden.
- Ein Forecasting-Modell verwendet aggregierte Monatswerte, obwohl am Monatsanfang vorhergesagt wird.
- Eine Imputation oder Skalierung wird auf dem gesamten Datensatz gelernt und nicht nur auf dem Trainingsset.
- Doppelte Kunden oder Dokumente landen in Training und Test.

Leakage führt zu überoptimistischen Kennzahlen. Das Modell scheint hervorragend, weil es die Lösung indirekt kennt. In Produktion bricht die Leistung ein. Deshalb muss Datenbereinigung immer zeitlich und prozessual denken: Welche Informationen waren zum Vorhersagezeitpunkt wirklich verfügbar?

Eine gute Regel lautet: Für jede Spalte muss beantwortet werden können, wann sie entsteht, wer sie erzeugt, ob sie nachträglich geändert wird und ob sie in der Zielumgebung rechtzeitig verfügbar ist.

Unausgewogene Klassen und seltene Ereignisse

Viele relevante ML-Probleme betreffen seltene Ereignisse: Betrug, Ausfall, Kündigung, Sicherheitsvorfall, Zahlungsstörung, Qualitätsmangel. Datenbereinigung kann solche Fälle unbeabsichtigt reduzieren. Wenn man etwa "unplausible" Extremwerte entfernt, fehlende Werte nur bei Mehrheitsfällen sauber imputiert oder Dubletten auf falscher Ebene löscht, verschwinden genau die Beispiele, die das Modell braucht.

Bei unausgewogenen Klassen ist Datenbereinigung mit Sampling, Gewichtung und Metrikwahl verbunden. Accuracy reicht nicht. Ein Modell, das bei 99 Prozent Nicht-Betrugsfällen immer "kein Betrug" sagt, erreicht 99 Prozent Accuracy und ist trotzdem nutzlos. Relevanter sind Precision, Recall, F1, PR-AUC, Kostenmatrizen und fachliche Schwellenwerte.

Cleaning-Fragen lauten hier:

- Werden Minderheitsfälle vollständig erhalten?
- Enthalten Minderheitsfälle häufiger fehlende Werte?
- Sind Labels bei seltenen Fällen verlässlicher oder unsicherer?
- Gibt es historische Untererfassung?
- Werden neue Fälle im Betrieb nach derselben Logik erkannt?

Zeitliche Drift und Prozessänderungen

Daten sind nicht statisch. Produkte, Märkte, Kundenverhalten, gesetzliche Regeln, interne Prozesse, Eingabemasken und IT-Systeme ändern sich. Ein Modell, das auf historischen Daten trainiert wurde, kann dadurch schleichend schlechter werden.

Man unterscheidet grob:

- **Data Drift**: Eingangsmerkmale verändern ihre Verteilung.
- **Concept Drift**: Der Zusammenhang zwischen Merkmalen und Ziel ändert sich.
- **Label Drift**: Die Zielvariable verändert ihre Definition oder Verteilung.
- **Schema Drift**: Spalten, Datentypen oder Kategorien ändern sich.
- **Process Drift**: Die Datenerzeugung selbst ändert sich, etwa durch neue Pflichtfelder.

Datenbereinigung muss Drift nicht nur rückwirkend reparieren, sondern sichtbar machen. Wenn plötzlich viel weniger fehlende Werte auftreten, kann das eine Qualitätsverbesserung sein oder eine neue Defaultbefüllung. Wenn eine Kategorie verschwindet, kann das fachlich korrekt sein oder ein Mappingfehler. Wenn ein Feld ab einem bestimmten Datum immer gefüllt ist, wurde vielleicht ein Prozess geändert.

Für historische Analysen ist der Artikel SQL-Historisierung anschlussfähig: Viele ML-Probleme benötigen zeitpunktgerechte Daten, nicht nur den aktuellen Datenstand.

Wie Datenfehler Modellarten unterschiedlich beeinflussen

Nicht jedes Modell reagiert gleich auf Datenprobleme. Datenbereinigung sollte deshalb die geplante Modellklasse berücksichtigen. Das bedeutet nicht, die Daten nur für ein bestimmtes Modell "hinzubiegen". Es bedeutet, die Wirkung von Cleaning-Entscheidungen zu verstehen.

Lineare Modelle

Lineare Regression, logistische Regression und verwandte Modelle sind interpretierbar, aber empfindlich gegenüber Skalierung, Ausreißern, Multikollinearität und Kodierung. Ein extremer Wert kann die Gerade stark beeinflussen. Kategoriale Variablen müssen sinnvoll kodiert werden. Fehlende Werte können nicht direkt verarbeitet werden.

Für lineare Modelle sind wichtig:

- robuste Behandlung von Ausreißern;
- Standardisierung oder Skalierung numerischer Merkmale;
- sinnvolle Dummy-Kodierung kategorialer Variablen;
- Prüfung auf starke Korrelationen;
- saubere Behandlung fehlender Werte;
- klare Trennung zwischen numerischen Größen und IDs.

IDs sind besonders tückisch. Eine Kundennummer ist numerisch, aber keine Zahl im mathematischen Sinn. Wird sie als kontinuierliches Merkmal verwendet, lernt das Modell künstliche Ordnung.

Baumverfahren und Ensembles

Entscheidungsbäume, Random Forests, Gradient Boosting und ähnliche Verfahren sind robuster gegenüber Skalierung und nichtlinearen Zusammenhängen. Sie können Ausreißer oft besser handhaben als lineare Modelle. Trotzdem lösen sie Datenprobleme nicht automatisch.

Baumverfahren können stark von Leakage-Features profitieren, seltene Kategorien überanpassen und bei fehlerhaften Labels komplexe falsche Regeln lernen. Sie sind außerdem empfindlich gegenüber systematischen Splittingfehlern. Wenn nahezu identische Fälle in Training und Test liegen, wirken sie beeindruckend gut.

Bei Baumverfahren ist daher weniger Skalierung, aber mehr Validierungsdesign entscheidend: gruppierte Splits, zeitbasierte Splits, Leakage-Prüfung und Prüfung seltener Kategorien.

Distanzbasierte Verfahren

K-nearest neighbors, Clustering-Verfahren und viele Anomalieerkennungen hängen stark davon ab, wie Distanzen berechnet werden. Skalierung und Kodierung sind hier zentral. Ein Merkmal mit großer numerischer Spanne kann alle anderen dominieren. Fehlende Werte und Ausreißer verzerren Nachbarschaften.

Für solche Verfahren ist Datenbereinigung eng mit Feature-Transformation verbunden:

- Skalierung numerischer Merkmale;
- robuste Transformationen bei schiefen Verteilungen;
- sinnvolle Distanzmaße für gemischte Daten;
- Umgang mit kategorialen Variablen;
- Ausreißeranalyse vor Clustering.

Neuronale Netze

Neuronale Netze können komplexe Muster lernen, benötigen aber meist konsistente, gut skalierte und umfangreiche Daten. Fehlerhafte Labels, Leakage, unausgewogene Klassen und nicht repräsentative Trainingsdaten können besonders teuer werden, weil das Modell viele Freiheitsgrade hat.

Bei Bild-, Audio- und Textdaten erweitert sich Datenbereinigung zu Datenkuration. Es geht nicht nur um tabellarische Plausibilität, sondern um beschädigte Dateien, falsche Metadaten, Dubletten, Urheber- und Nutzungsrechte, Klassenbalance, Annotationen, Sprache, Qualität und Repräsentativität.

Zeitreihenmodelle

Bei Zeitreihen ist die zeitliche Logik zentral. Fehlende Werte, Ausreißer, Kalenderlogik, Feiertage, Saisonalität, Sonderaktionen, verspätete Buchungen und Aggregation beeinflussen das Modell stark. Zufällige Splits sind häufig falsch, weil die Zukunft nicht zur Vergangenheit werden darf.

Datenbereinigung bei Zeitreihen fragt:

- Welche Zeitauflösung ist fachlich sinnvoll?
- Sind Zeitstempel in derselben Zeitzone?
- Werden verspätete Ereignisse rückwirkend gebucht?
- Gibt es Lücken, Mehrfachmessungen oder unregelmäßige Intervalle?
- Sind Sonderereignisse markiert?
- Wird nur Information genutzt, die zum Prognosezeitpunkt verfügbar war?

Text- und RAG-Systeme

Bei Textdaten heißt Datenbereinigung häufig Dokumentenkuration. Für Retrieval-Augmented Generation, semantische Suche und Wissensdatenbanken reicht es nicht, Texte zu sammeln. Man muss Versionen, Dubletten, veraltete Dokumente, Zugriffsrechte, widersprüchliche Inhalte, Metadaten und Chunking-Regeln behandeln. Der Artikel Vektordatenbanken ist hier ein passender Anschluss.

Ein RAG-System mit schlechten Dokumenten wird nicht dadurch gut, dass das Sprachmodell stark ist. Es kann nur auf Basis des Materials antworten, das gefunden wird. Wenn die Wissensbasis veraltet, doppelt oder widersprüchlich ist, entstehen falsche Antworten mit scheinbar passenden Quellen.

Datenbereinigung für RAG bedeutet:

- veraltete Dokumente erkennen und aussteuern;
- Dubletten und Versionen bereinigen;
- Metadaten wie Gültigkeit, Dokumenttyp, Bereich und Vertraulichkeit pflegen;
- Zugriffsrechte berücksichtigen;
- Chunking nicht nur technisch, sondern semantisch planen;
- Retrieval-Ergebnisse regelmäßig testen.

Der Datenbereinigungsprozess als Pipeline

Eine robuste Datenbereinigung ist kein einmaliger Handgriff, sondern eine Pipeline. Sie sollte wiederholbar, dokumentiert und testbar sein. Das ist der Unterschied zwischen explorativer Datenanalyse und produktionsfähiger ML-Vorbereitung.

Eine typische Pipeline sieht so aus:

1. Rohdaten unverändert sichern.
2. Datenquellen und fachliche Bedeutung dokumentieren.
3. Datenprofiling durchführen.
4. Qualitätsregeln definieren.
5. Trainings-, Validierungs- und Testlogik festlegen.
6. Cleaning-Schritte als reproduzierbare Transformationen implementieren.
7. Feature Engineering vom Cleaning trennen, aber gemeinsam versionieren.
8. Qualitätsmetriken und Modellmetriken zusammen auswerten.
9. Entscheidungen dokumentieren.
10. Monitoring für produktive Daten einrichten.

Das klingt formal, spart aber später Zeit. Ohne Pipeline werden Fehler schwer nachvollziehbar. Niemand weiß mehr, ob eine Verbesserung vom Modell, von einer Datenkorrektur oder von einem unbeabsichtigten Leakage kam.

Rohdaten behalten

Rohdaten sollten nicht überschrieben werden. Jede Korrektur muss nachvollziehbar sein. Das gilt besonders, wenn spätere Audits, Reproduzierbarkeit oder fachliche Streitfälle möglich sind.

Eine gute Praxis ist die Trennung in Schichten:

Schicht	Zweck
Raw	unveränderte Quelldaten, nur technisch geladen
Staging	technische Typisierung, Strukturierung, Grundvalidierung
Clean	fachlich bereinigte, dokumentierte Daten
Feature	modellnahe Merkmale, Aggregationen, Kodierungen
Train/Test	eingefrorene Datensplits für Experimente
Serving	Transformationen für produktive Vorhersagen

Diese Logik passt zu Data-Warehouse- und Lakehouse-Architekturen. Der Artikel DWH - Data Warehouse und der Überblick zu Datenintegration - ETL vertiefen die infrastrukturelle Seite.

Datenprofiling

Datenprofiling ist die erste systematische Inspektion. Es beantwortet Fragen wie:

- Wie viele Zeilen und Spalten gibt es?
- Welche Datentypen liegen vor?
- Welche Werte fehlen?
- Welche Wertebereiche treten auf?
- Welche Kategorien sind häufig oder selten?
- Gibt es offensichtlich unmögliche Werte?
- Gibt es Dubletten?
- Wie verteilen sich Zielvariable und wichtige Merkmale?
- Gibt es zeitliche Brüche?
- Welche Spalten sind konstant oder fast konstant?

Profiling ist kein Ersatz für Fachwissen. Es zeigt Symptome. Ob ein Wert falsch ist, muss fachlich entschieden werden. Ein Alter von 105 Jahren ist selten, aber möglich. Eine Lieferzeit von 0 Tagen kann ein Fehler, ein Downloadprodukt oder eine Sofortabholung sein.

Qualitätsregeln definieren

Qualitätsregeln übersetzen Fachwissen in prüfbare Bedingungen. Beispiele:

- `order_date` darf nicht nach `delivery_date` liegen.
- `quantity` muss größer als 0 sein, außer bei Stornozeilen.
- `customer_id` muss in der Kundentabelle existieren.
- `country_code` muss einem gültigen ISO-Code entsprechen.
- `birth_date` darf nicht in der Zukunft liegen.
- `cancellation_date` darf nur bei gekündigten Verträgen gefüllt sein.
- `prediction_timestamp` muss vor allen verwendeten Ereignismerkmalen liegen.

Für ML kommen Modellregeln hinzu:

- Feature darf nicht nach dem Prognosezeitpunkt entstehen.
- Zielvariable darf nicht direkt oder indirekt als Feature enthalten sein.
- Imputer, Scaler und Encoder werden nur auf Trainingsdaten gelernt.
- Testdaten bleiben bis zur finalen Bewertung unberührt.
- Gruppierte Entitäten dürfen nicht über Splits verstreut werden, wenn Wiedererkennung droht.

Solche Regeln sollten nicht nur in Dokumenten stehen, sondern automatisiert geprüft werden. Ein roter Datenqualitätstest ist oft wertvoller als ein erfolgreiches Training mit falschen Daten.

Cleaning-Entscheidungen dokumentieren

Jeder relevante Cleaning-Schritt sollte eine Begründung haben:

Entscheidung	Dokumentationsfrage
Wert korrigiert	Welche Quelle oder Regel begründet die Korrektur?
Wert imputiert	Welche Methode, welcher Fit-Zeitraum, welche Annahme?
Zeile entfernt	Warum gehört die Beobachtung nicht zum Lernproblem?
Ausreißer gedeckelt	Warum ist Capping fachlich besser als Behalten?
Kategorie zusammengeführt	Welche fachliche Gleichsetzung steckt dahinter?
Feature ausgeschlossen	Leakage, Datenschutz, Verfügbarkeit oder fachliche Irrelevanz?
Label angepasst	Welche Definition gilt, und ab wann?

Dokumentation ist kein Selbstzweck. Sie schützt vor späteren Missverständnissen. Wenn ein Modell in Produktion schlechter wird, muss man wissen, ob sich die Daten verändert haben oder ob alte Cleaning-Annahmen nicht mehr gelten.

Methoden der Datenbereinigung im Detail

Es gibt keine universelle Methode, die alle Datenprobleme löst. Gute Datenbereinigung kombiniert einfache Regeln, statistische Verfahren, Domänenwissen und technische Tests.

Standardisierung und Normalisierung von Formaten

Viele Cleaning-Projekte beginnen mit Standardisierung:

- Leerzeichen entfernen;
- Groß-/Kleinschreibung vereinheitlichen;
- Unicode- und Sonderzeichen normalisieren;
- Datumsformate vereinheitlichen;
- Zeitzonen festlegen;
- Dezimaltrennzeichen korrigieren;
- Einheiten umrechnen;
- Währungen normalisieren;
- Codes auf Referenztabellen abbilden.

Diese Schritte wirken banal, sind aber Grundlage für spätere Prüfungen. Wenn `AT`, `Austria`, `Österreich`, `Oesterreich` und `AUT` nicht zusammengeführt werden, entstehen künstliche Kategorien. Wenn Kilometer und Meilen in derselben Spalte stehen, ist jedes Modell falsch.

Bei Standardisierung sollte der Originalwert erhalten bleiben. Das ermöglicht Nachvollziehbarkeit und spätere Korrekturen, falls das Mapping falsch war.

Fehlende Werte behandeln

Der richtige Umgang mit fehlenden Werten hängt von Ursache, Modellklasse und fachlichem Ziel ab.

Mögliche Strategien:

Strategie	Vorteil	Risiko
Zeilen löschen	einfach, klar	Datenverlust, Verzerrung bei systematischem Fehlen
Spalten löschen	reduziert Komplexität	Verlust wichtiger Signale
einfacher Ersatzwert	schnell, reproduzierbar	verzerrt Verteilungen
Gruppenimputation	berücksichtigt Segmente	kann Gruppenmuster verstärken
modellbasierte Imputation	nutzt Zusammenhänge	komplex, Leakage-Gefahr
Missingness-Indikator	erhält Fehlen als Signal	erhöht Feature-Anzahl
separate Kategorie	sinnvoll bei kategorialen Werten	kann technische Fehler legitimieren

Ein Beispiel: In einem Kündigungsmodell fehlt das Feld `last_support_contact_days` für Kunden, die nie den Support kontaktiert haben. Das ist nicht dasselbe wie ein unbekannter Wert. Ein Ersatz durch den Mittelwert wäre fachlich falsch. Besser wäre eine Kombination aus `had_support_contact` und einem numerischen Wert nur für relevante Fälle.

Bei Zeitreihen ist zusätzlich wichtig, ob fehlende Werte interpoliert werden dürfen. Eine lineare Interpolation kann bei Sensordaten sinnvoll sein, bei Bestellungen oder Ereigniszählungen aber völlig falsche Daten erzeugen.

Ausreißer behandeln

Ausreißerbehandlung reicht von einfacher Plausibilitätsprüfung bis zu statistischen Verfahren.

Mögliche Methoden:

- harte fachliche Grenzen;
- z-Score oder robuste z-Scores;
- Interquartilsabstand;
- Perzentil-Capping;
- Isolation Forest;
- Local Outlier Factor;
- Zeitreihenbasierte Anomalieerkennung;
- manuelle Stichprobenprüfung;
- Segmentvergleich.

Fachliche Grenzen sind oft besser als rein statistische. Ein negativer Preis ist meist unzulässig. Ein 99,9-Perzentil-Umsatz kann dagegen ein echter Großauftrag sein. Statistische Methoden zeigen Auffälligkeit, nicht Fehlerhaftigkeit.

Bei ML sollte man außerdem die Wirkung messen: Wie verändern sich Modellmetriken, Fairnessmetriken und Fehlerfälle, wenn Ausreißer behalten, markiert, gedeckelt oder entfernt werden? Eine Cleaning-Entscheidung ist dann nicht nur ästhetisch, sondern empirisch und fachlich begründet.

Duplikate und Entity Resolution

Duplikaterkennung ist einfach, wenn alle Spalten identisch sind. In der Praxis braucht man häufig Entity Resolution: Gehören mehrere Datensätze zur gleichen realen Entität?

Beispiele:

- `Erhard Rainer`, `Rainer Erhard`, `E. Rainer`;
- gleiche Firma mit unterschiedlicher Rechtsformschreibung;
- Produktnummern aus alten und neuen Systemen;
- Geräte mit Seriennummer und Inventarnummer;
- Dokumentversionen mit leicht unterschiedlichem Titel.

Methoden:

- exakte Schlüssel;
- zusammengesetzte Schlüssel;
- Hashes;
- Fuzzy Matching;
- phonetische Ähnlichkeit;
- Adressnormalisierung;
- Referenzdaten;
- manuelle Prüfung bei unsicheren Treffern.

Für ML ist wichtig, ob die Zusammenführung vor oder nach dem Split passiert. Entity Resolution nach dem Split kann Training und Test verunreinigen. Ebenso müssen Aggregationen auf Entitätsebene zeitlich korrekt sein.

Kategoriale Werte bereinigen

Kategorien sollten nicht blind numerisch gemacht werden. Vor der Kodierung steht die fachliche Bereinigung:

- Synonyme zusammenführen;

- veraltete Codes mappen;
- Hierarchien prüfen;
- seltene Kategorien sinnvoll gruppieren;
- unbekannte Kategorien für Produktion einplanen;
- Gültigkeitszeiträume berücksichtigen.

Ein Modell, das nur historische Kategorien kennt, muss im Betrieb neue Werte behandeln können. Deshalb braucht jede produktive Pipeline eine Strategie für "unknown" oder "other". Diese Strategie muss bereits im Training simuliert werden, sonst schlägt das Modell später bei neuen Kategorien fehl oder erzeugt unvorhersehbare Kodierungen.

Numerische Werte transformieren

Numerische Daten können schief verteilt, stark skaliert oder heterogen sein. Transformationen helfen, Modelle stabiler zu machen:

- logarithmische Transformation;
- Box-Cox- oder Yeo-Johnson-Transformation;
- Standardisierung;
- Min-Max-Skalierung;
- robuste Skalierung über Median und Interquartilsabstand;
- Winsorizing oder Capping;
- Bucketisierung.

Nicht jede Transformation ist Datenbereinigung. Manche gehören eher zum Feature Engineering. Die Grenze ist fließend. Entscheidend ist, dass Transformationen reproduzierbar sind und nur auf Trainingsdaten gelernt werden.

Datensätze zusammenführen

Viele ML-Projekte kombinieren Daten aus CRM, ERP, Webtracking, Support, Produktion, Finance oder externen Quellen. Hier entstehen besonders viele Fehler:

- Join vervielfacht Zeilen;
- Schlüssel sind nicht eindeutig;
- Historientabellen werden falsch aufgelöst;
- aktuelle Stammdaten werden auf historische Ereignisse angewendet;
- Zeitzonen passen nicht;
- Aggregationen verwenden Information aus der Zukunft;
- veraltete Mappingtabellen erzeugen falsche Kategorien.

Ein Join ist daher kein rein technischer Schritt. Für ML muss man die Körnung der Daten kennen: Eine Zeile pro Kunde? Pro Vertrag? Pro Monat? Pro Ereignis? Pro Produktposition? Ohne klare Körnung erzeugt man Datensätze, die statistisch aussehen, aber fachlich unbrauchbar sind.

Der Artikel Datenbankmodellierung ist hier relevant, weil viele ML-Probleme aus unscharfen Entitäten, Schlüsseln und Beziehungen entstehen.

Feature Engineering und Datenbereinigung

Feature Engineering baut aus Rohdaten lernbare Merkmale. Datenbereinigung sorgt dafür, dass diese Merkmale verlässlich sind. In der Praxis sind beide Schritte eng verbunden.

Ein Beispiel: Aus Bestellungen sollen Kundenmerkmale entstehen.

Rohdatenproblem	Feature-Risiko
doppelte Bestellungen	Umsatzfeature zu hoch
Stornos nicht korrekt markiert	Kaufhäufigkeit falsch
Währung gemischt	Warenkorbwert nicht vergleichbar
verspätete Buchungen	Prognosezeitpunkt verletzt
neue Produktgruppen nicht gemappt	Kategorien fehlen
Testkunden im System	Modell lernt interne Artefakte

Feature Engineering muss zeitpunktgerecht sein. Für ein Kündigungsmodell zum Monatsanfang darf ein Feature nicht Informationen aus späteren Supportfällen im selben Monat enthalten. Für ein Kreditrisikomodell darf eine nachträgliche Mahnung nicht als Eingangsmerkmal verwendet werden. Für Forecasting darf eine Monatsaggregation nicht den ganzen Monat kennen, wenn am ersten Tag prognostiziert wird.

Eine gute Praxis ist eine Feature-Spezifikation:

Feature	Beschreibung	Datenquelle	Zeitpunkt	Cleaning-Regel	Leakage-Risiko
`orders_90d`	Anzahl Bestellungen in letzten 90 Tagen	ERP	vor Prognosezeitpunkt	Stornos ausschließen	niedrig
`support_cases_30d`	Supportfälle in letzten 30 Tagen	Ticketing	vor Prognosezeitpunkt	Duplikate pro Ticket-ID entfernen	mittel
`last_invoice_status`	letzter Rechnungsstatus	Finance	vor Prognosezeitpunkt	nur historischer Stand	hoch, wenn aktueller Status genutzt wird
`customer_segment`	Kundensegment	CRM	gültig zum Stichtag	Mapping nach Gültigkeit	mittel

Ohne solche Spezifikation wird Feature Engineering schnell zur Quelle versteckter Fehler.

Trainings-, Validierungs- und Testdaten sauber trennen

Die wichtigste technische Regel in der ML-Datenbereinigung lautet: Alles, was aus Daten gelernt wird, muss innerhalb des Trainingssets gelernt werden. Das gilt für Imputation, Skalierung, Encoding, Feature-Auswahl, Ausreißergrenzen, Zielkodierung und Textvektorisierung.

Der typische Fehler:

1. Gesamten Datensatz bereinigen.
2. Gesamten Datensatz skalieren.
3. Gesamten Datensatz in Training und Test splitten.

4. Modell trainieren.
5. Gute Testmetriken feiern.

Das Problem: Der Testdatensatz hat bereits Cleaning-Parameter beeinflusst. Manchmal ist der Effekt klein, manchmal massiv. Bei Zielkodierung, Feature-Auswahl oder zeitabhängigen Aggregationen kann es die Evaluation vollständig verfälschen.

Korrekt ist:

1. Split-Logik festlegen.
2. Trainingsdaten verwenden, um Cleaning- und Transformationsparameter zu lernen.
3. Dieselben gelernten Transformationen auf Validierungs- und Testdaten anwenden.
4. Testdaten erst für die finalere Bewertung nutzen.

Bei Zeitreihen lautet die Regel noch strenger: Vergangenheit trainiert, Zukunft validiert. Zufällige Splits sind nur geeignet, wenn die Daten wirklich austauschbare, unabhängige Beobachtungen sind.

Bei Kunden-, Patienten-, Maschinen- oder Dokumentdaten ist ein gruppierter Split oft sinnvoll. Alle Beobachtungen derselben Entität gehören in denselben Split, damit das Modell nicht dieselbe Entität wiedererkennt.

Datenqualität messen

Datenqualität muss messbar werden, sonst bleibt sie Bauchgefühl. Typische Qualitätsdimensionen sind:

Dimension	Frage	Mögliche Metrik
Vollständigkeit	Sind notwendige Werte vorhanden?	Anteil fehlender Werte je Spalte
Gültigkeit	Entsprechen Werte erlaubten Regeln?	Anteil Regelverletzungen
Konsistenz	Passen Werte zueinander?	Widerspruchsquote zwischen Feldern
Eindeutigkeit	Gibt es ungewollte Duplikate?	Duplicate Rate
Aktualität	Sind Daten zeitgerecht?	Alter der letzten Lieferung
Genauigkeit	Stimmen Daten mit Referenz oder Realität überein?	Stichprobenfehlerquote
Repräsentativität	Decken Daten relevante Fälle ab?	Segmentabdeckung, Klassenverteilung
Stabilität	Bleiben Verteilungen im erwarteten Bereich?	Drift-Metriken

Für ML sollte man Datenqualitätsmetriken neben Modellmetriken betrachten. Eine Modellverbesserung ist verdächtig, wenn gleichzeitig die Duplicate Rate steigt oder ein Leakage-verdächtiges Feature auftaucht. Umgekehrt kann eine scheinbare Modellverschlechterung durch strengere Datenbereinigung ehrlicher sein.

Ein gutes Dashboard zeigt nicht nur Accuracy oder AUC, sondern auch:

- Anzahl Trainingsbeobachtungen;
- Missingness je Feature;
- Verteilung der Zielvariable;
- neue Kategorien;
- Anteil ausgeschlossener Zeilen;
- Ausreißerquote;

- Label-Verzögerung;
- Datenalter;
- Schemaänderungen;
- Drift gegenüber Trainingsdaten.

Fairness, Bias und ethische Folgen schlechter Daten

Datenbereinigung hat eine Fairness-Dimension. Verzerrte Daten führen zu verzerrten Modellen. Das betrifft nicht nur sensible Merkmale wie Geschlecht, Alter, Herkunft oder Behinderung, sondern auch Proxy-Merkmale: Wohnort, Beschäftigungsart, Bildung, Kaufhistorie, Sprache, Gerätetyp, Arbeitszeitmuster oder Interaktionskanal.

Cleaning kann Bias reduzieren, aber auch verstärken.

Beispiele:

- Fehlende Werte werden bei einer Gruppe häufiger imputiert und verlieren dadurch Signal.
- Seltene Kategorien werden zu "Sonstige" zusammengeführt und machen Minderheiten unsichtbar.
- Historische Ablehnungsentscheidungen werden als objektive Labels behandelt.
- Ausreißer einer Gruppe werden häufiger entfernt.
- Daten aus digitalen Kanälen überrepräsentieren technisch affine Nutzer.
- Qualitätsregeln wurden für Mehrheitsfälle definiert und markieren andere Fälle als "unplausibel".

Fairness beginnt daher nicht erst beim Modell. Sie beginnt bei Datenherkunft, Labeldefinition, Sampling, Cleaning und Evaluation. Eine faire Datenbereinigung fragt:

- Welche Gruppen fehlen oder sind unterrepräsentiert?
- Sind fehlende Werte gleich verteilt?
- Sind Labels für alle Gruppen gleich zuverlässig?
- Werden Proxy-Merkmale verwendet?
- Haben Cleaning-Regeln unterschiedliche Auswirkungen auf Gruppen?
- Werden Fehlerfälle nach Segmenten analysiert?

Bei KI-Systemen mit rechtlicher oder wirtschaftlicher Wirkung sollte diese Analyse dokumentiert werden. Der Artikel DSGVO bei der Anwendung von KI-Chatbots und der EU AI Act Deep Dive sind sinnvolle Anschlussstellen für regulatorische Fragen.

Datenschutz und Datenminimierung

Datenbereinigung kann personenbezogene Daten reduzieren, pseudonymisieren oder strukturieren. Gleichzeitig kann sie neue Risiken erzeugen, wenn zusätzliche Datenquellen verknüpft werden.

Für ML gilt:

- Nicht jedes verfügbare Merkmal darf oder sollte verwendet werden.
- Personenbezogene Daten brauchen Rechtsgrundlage, Zweckbindung und Schutzmaßnahmen.
- Pseudonymisierung ersetzt keine Datenschutzprüfung.
- Aggregationen können Re-Identifikation erschweren, aber nicht immer verhindern.

- Sensible Merkmale können für Fairnessprüfung nötig sein, dürfen aber nicht unkontrolliert in das Modell einfließen.
- Lösch- und Berichtigungspflichten müssen mit Trainingsdaten und Modellversionen zusammengedacht werden.

Datenminimierung ist nicht gleichbedeutend mit "so wenig Daten, dass das Modell schlecht wird". Sie bedeutet: nur jene Daten verwenden, die für den Zweck erforderlich, begründbar und geschützt sind. Eine gute Cleaning-Pipeline kann sensible Felder früh entfernen, Rollen trennen und Auditspuren erzeugen.

Automatisierung: hilfreich, aber nicht blind

Automatisierte Tools können Datenprofiling, Schema-Validierung, Missingness-Analysen, Drift-Erkennung und Pipeline-Tests stark erleichtern. In Python-Umgebungen sind Bibliotheken wie pandas, scikit-learn Pipelines, Great Expectations, Pandera, TensorFlow Data Validation oder ähnliche Werkzeuge verbreitet. In Spark- und Lakehouse-Umgebungen kommen andere Validierungs- und Qualitätsframeworks zum Einsatz.

Automatisierung ist besonders nützlich für:

- wiederkehrende Schema-Prüfungen;
- Datentyp- und Wertebereichstests;
- Missingness-Monitoring;
- Duplikaterkennung;
- Referenzdatenabgleich;
- Drift-Metriken;
- Pipeline-Reproduzierbarkeit;
- Trainings-/Serving-Konsistenz.

Aber Automatisierung beantwortet nicht:

- ob ein ungewöhnlicher Wert fachlich wichtig ist;
- ob ein Label gerechtfertigt ist;
- ob ein Proxy-Merkmal unfair wirkt;
- ob eine historische Prozessänderung das Lernproblem verändert;
- ob ein Feature im Prognosezeitpunkt verfügbar war;
- ob ein Modellzweck legitim ist.

Die beste Architektur verbindet automatisierte Tests mit fachlicher Review. Data Scientists, Data Engineers und Fachbereiche müssen dieselbe Sprache finden. Fachbereiche wissen, was ein Wert bedeutet. Data Engineers wissen, wie er entsteht. Data Scientists wissen, wie er das Modell beeinflusst.

Beispiel: Kundenabwanderung vorhersagen

Ein Unternehmen möchte vorhersagen, welche Kunden in den nächsten 60 Tagen kündigen könnten. Der Rohdatensatz enthält Kundendaten, Verträge, Rechnungen, Supporttickets und Kündigungen.

Eine naive Datenaufbereitung könnte so aussehen:

- aktuelle Kundentabelle exportieren;

- alle Rechnungen aggregieren;
- alle Supporttickets zählen;
- Kündigungsstatus als Zielvariable setzen;
- fehlende Werte mit 0 ersetzen;
- Modell trainieren.

Das wirkt plausibel, ist aber voller Risiken.

Problem 1: aktueller Datenstand statt historischer Sicht

Wenn die aktuelle Kundentabelle verwendet wird, enthält sie Informationen, die zum damaligen Prognosezeitpunkt nicht bekannt waren. Das heutige Segment, die heutige Adresse oder der heutige Vertragsstatus können nach der Kündigung geändert worden sein.

Korrektur: Für jeden Stichtag muss der historische Stand rekonstruiert werden. Features dürfen nur Informationen enthalten, die vor dem Stichtag verfügbar waren.

Problem 2: Kündigungsnahe Informationen als Feature

Supporttickets kurz vor der Kündigung können relevant sein. Aber Tickets, die nach der Kündigung entstehen, dürfen nicht verwendet werden. Auch ein Feld wie `retention_offer_sent` kann Leakage sein, wenn es erst als Reaktion auf eine Kündigungsabsicht entsteht.

Korrektur: Jede Ereignistabelle braucht einen Zeitfilter relativ zum Prognosezeitpunkt.

Problem 3: fehlende Werte falsch interpretiert

Ein fehlendes Supportticket ist nicht dasselbe wie ein unbekannter Wert. Es kann bedeuten, dass der Kunde keinen Kontakt hatte. Ein fehlendes Geburtsdatum ist dagegen unbekannt oder nicht erhoben. Beide Fälle gleich mit 0 zu ersetzen wäre falsch.

Korrektur: Fehlende Werte nach Bedeutung klassifizieren: nicht vorhanden, unbekannt, nicht anwendbar, verspätet, technisch fehlerhaft.

Problem 4: unausgewogene Klassen

Wenn nur zwei Prozent der Kunden kündigen, ist ein Modell mit hoher Accuracy wenig aussagekräftig. Datenbereinigung muss sicherstellen, dass Kündigungsfälle nicht durch Filterregeln verschwinden und dass Evaluation geeignete Metriken nutzt.

Korrektur: PR-AUC, Recall bei definiertem Precision-Niveau, Kostenmatrix und segmentierte Fehleranalyse verwenden.

Problem 5: Fairness und Proxy-Merkmale

Wohnort, Zahlungsverhalten, Kanalnutzung oder Gerätetyp können sensible Gruppen indirekt abbilden. Ein Modell zur Kundenbindung ist weniger heikel als ein Kreditmodell, aber auch hier können unerwünschte Schlechterstellungen entstehen.

Korrektur: Segmentierte Evaluation und fachliche Prüfung der Features.

Dieses Beispiel zeigt: Datenbereinigung ist nicht nur "Nullwerte füllen". Sie ist die Konstruktion einer historischen, fairen und prognosetauglichen Sicht auf die Realität.

Beispiel: Qualitätsprüfung in der Produktion

Ein Produktionsunternehmen möchte Maschinenausfälle vorhersagen. Es nutzt Sensordaten, Wartungsprotokolle, Schichtdaten und Fehlermeldungen.

Typische Datenprobleme:

- Sensoren senden unregelmäßig;
- Werte fehlen bei Netzwerkproblemen;
- Sensoren werden kalibriert oder ausgetauscht;
- Wartungen werden verspätet dokumentiert;
- Fehlercodes ändern sich nach Softwareupdates;
- Ausfälle sind selten;
- Extremwerte können Messfehler oder frühe Warnzeichen sein.

Eine zu aggressive Bereinigung würde viele seltene Extremwerte entfernen. Das Modell wird dadurch ruhiger, aber schlechter im Erkennen echter Ausfälle. Eine bessere Strategie:

- technische Sensorfehler markieren;
- Kalibrierungszeiträume dokumentieren;
- Ausreißer nicht pauschal löschen;
- Wartungen zeitlich korrekt einbeziehen;
- Feature-Fenster sauber definieren;
- nach Maschinen, Linien und Schichten evaluieren;
- Drift nach Sensorwechsel überwachen.

Bei Sensordaten ist außerdem wichtig, Trainings- und Produktionspipeline identisch zu halten. Wenn im Training geglättete Daten verwendet werden, in Produktion aber Rohwerte ankommen, entsteht ein Serving-Skew.

Beispiel: Dokumentensuche und generative KI

Ein Unternehmen baut eine interne Wissenssuche mit generativer Antwortfunktion. Dokumente werden extrahiert, bereinigt, in Chunks zerlegt, eingebettet und in einer Vektordatenbank gespeichert.

Hier sieht Datenbereinigung anders aus:

- PDF-Extraktion kann Tabellen zerstören;
- Kopf- und Fußzeilen erzeugen Rauschen;
- alte Versionen bleiben im Index;
- Dokumente haben fehlende Gültigkeitsdaten;
- vertrauliche Dokumente werden falsch berechtigt;
- ähnliche Dokumente erzeugen widersprüchliche Antworten;
- OCR-Fehler verändern Begriffe;
- Chunking trennt Definition und Kontext.

Ein starkes Sprachmodell kann diese Probleme nicht zuverlässig reparieren. Es formuliert aus gefundenem Material eine Antwort. Wenn der Index schlechte oder falsche Dokumente liefert, entsteht eine gute Formulierung mit schlechtem Fundament.

Cleaning-Maßnahmen:

- Dokumentversionen eindeutig regeln;
- Metadaten erzwingen;
- veraltete Inhalte aussteuern;
- Dubletten entfernen oder priorisieren;
- Zugriffsklassen in Retrieval einbauen;
- Extraktion stichprobenartig prüfen;
- Testfragen mit erwarteten Quellen definieren;
- Antworten gegen Quellen validieren.

Der Artikel Prompt Engineering ergänzt diese Perspektive: Gute Prompts helfen, aber sie ersetzen keine saubere Datenbasis.

Checkliste: Datenbereinigung für ML-Projekte

Eine kompakte Checkliste für den Projektstart:

Bereich	Prüffragen
Zieldefinition	Ist klar, was vorhergesagt, klassifiziert oder optimiert werden soll?
Zielvariable	Ist das Label fachlich korrekt, zeitlich passend und stabil definiert?
Datenquellen	Sind Herkunft, Aktualität, Verantwortliche und Erfassungslogik bekannt?
Körnung	Ist klar, wofür eine Zeile steht?
Zeitlogik	Sind Prognosezeitpunkt und Verfügbarkeitszeitpunkt je Feature geklärt?
Fehlende Werte	Ist bekannt, warum Werte fehlen?
Ausreißer	Werden technische Fehler von seltenen realen Ereignissen unterschieden?
Duplikate	Ist die Entitätsebene geklärt, und werden Splits nicht kontaminiert?
Kategorien	Gibt es Mapping, Referenzdaten und Unknown-Strategie?
Datenschutz	Sind personenbezogene und sensible Merkmale begründet und geschützt?
Fairness	Werden Gruppenwirkungen und Proxy-Merkmale geprüft?
Pipeline	Sind Cleaning-Schritte reproduzierbar und versioniert?
Evaluation	Sind Splits, Metriken und Fehleranalysen passend zum Einsatzfall?
Monitoring	Werden Schema, Missingness, Drift und neue Kategorien nach Deployment überwacht?

Häufige Anti-Patterns

Alles in Excel bereinigen

Excel ist für Exploration und kleine Korrekturen praktisch. Für produktive ML-Pipelines ist es gefährlich, wenn Bereinigungsschritte manuell und nicht reproduzierbar passieren. Niemand weiß später, welche Zeilen gelöscht, welche Werte geändert und welche Filter gesetzt wurden.

Wenn Excel genutzt wird, sollten Korrekturen in strukturierten Tabellen oder Regeln dokumentiert und anschließend automatisiert übernommen werden.

Erst alles bereinigen, dann splitten

Das ist eine häufige Leakage-Quelle. Transformationen, die aus Daten lernen, gehören in eine Pipeline und werden nur auf Trainingsdaten gefittet.

Ausreißer automatisch löschen

Ausreißer können Fehler sein, aber auch die wichtigsten Signale. Erst fachlich prüfen, dann entscheiden.

Missing Values mit 0 verwechseln

Null bedeutet "nichts" oder "Wert 0". Fehlend bedeutet "nicht bekannt", "nicht erhoben", "nicht anwendbar" oder "technisch verloren". Diese Bedeutungen dürfen nicht vermischt werden.

IDs als numerische Merkmale verwenden

IDs sind meist Schlüssel, keine Zahlen. Sie können Gruppierung, Verknüpfung oder Split-Logik steuern, sollten aber selten direkt als numerische Features ins Modell.

Aktuelle Stammdaten für historische Vorhersagen nutzen

Das erzeugt oft Leakage. Historische Modelle brauchen historische Stammdatenstände oder klar definierte Stichtagslogik.

Nur globale Metriken betrachten

Ein Modell kann global gut sein und in wichtigen Segmenten schlecht. Datenbereinigung muss Fehler segmentiert betrachten: Gruppen, Produkte, Regionen, Zeiträume, Kanäle, Maschinen oder Kundentypen.

Trainings- und Produktionspipeline auseinanderlaufen lassen

Wenn Training anders bereinigt als Produktion, entsteht Serving-Skew. Die Transformationen müssen als Code, Konfiguration oder Pipeline-Artefakt wiederverwendbar sein.

Datenbereinigung organisatorisch verankern

Datenbereinigung ist Teamarbeit. Die Rollen sind unterschiedlich:

Rolle	Beitrag
Fachbereich	Bedeutung der Daten, Plausibilitätsregeln, Labeldefinition
Data Engineering	Datenquellen, ETL, Historisierung, Qualitätstests
Data Science	Modellwirkung, Splits, Features, Evaluation
Datenschutz und Security	Zulässigkeit, Schutzbedarf, Zugriff, Löschkonzepte
Betrieb / MLOps	Monitoring, Retraining, Pipeline-Stabilität
Management	Priorisierung, Verantwortlichkeit, Risikotoleranz

Ein häufiger Fehler ist, Datenbereinigung als Junior-Aufgabe zu behandeln. Natürlich gibt es technische Routinearbeit. Aber die entscheidenden Cleaning-Fragen sind oft Senior-Fragen: Was bedeutet dieser Wert? Darf das Modell dieses Merkmal kennen? Ist das Label stabil? Welche Fälle dürfen nicht verloren gehen? Welche Gruppen werden benachteiligt? Welche Datenänderung würde das Modell im Betrieb beschädigen?

Organisationen, die ML ernsthaft einsetzen wollen, brauchen daher Datenqualitätsverantwortung. Nicht als bürokratische Zusatzrolle, sondern als Bestandteil jeder produktiven Daten- und KI-Initiative.

Praktisches Vorgehensmodell

Für ein reales Projekt kann ein schlankes Vorgehensmodell so aussehen:

Phase 1: Datenverständnis

- Ziel und Vorhersagezeitpunkt definieren.
- Datenquellen inventarisieren.
- Körnung je Tabelle klären.
- Labeldefinition festlegen.
- erste Profiling-Auswertung erstellen.
- offensichtliche Qualitätsprobleme sammeln.

Ergebnis: Datensteckbrief und Liste offener fachlicher Fragen.

Phase 2: Qualitätsregeln

- Pflichtfelder festlegen.
- Wertebereiche und Referenzdaten definieren.
- zeitliche Regeln formulieren.
- Leakage-verdächtige Spalten markieren.
- Datenschutz- und Fairnessfragen klären.
- Split-Strategie bestimmen.

Ergebnis: prüfbare Qualitätsregeln und Split-Konzept.

Phase 3: Reproduzierbare Cleaning-Pipeline

- Rohdaten unverändert speichern.
- Cleaning-Schritte als Code oder deklarative Regeln umsetzen.
- Imputer, Scaler und Encoder in ML-Pipeline integrieren.
- Entscheidungen dokumentieren.
- Testdatensatz unangetastet lassen.

Ergebnis: wiederholbare Datenaufbereitung.

Phase 4: Modell und Fehleranalyse

- Baseline-Modell trainieren.
- Fehlerfälle fachlich analysieren.
- Datenprobleme aus Modellfehlern ableiten.

- Cleaning-Regeln iterativ verbessern.
- Segmentmetriken prüfen.

Ergebnis: bessere Daten und realistischere Modellbewertung.

Phase 5: Betrieb und Monitoring

- Datenqualität im Serving überwachen.
- Schema- und Verteilungsänderungen melden.
- neue Kategorien behandeln.
- Label-Verzögerungen berücksichtigen.
- Retraining an Qualitätskriterien koppeln.
- fachliche Reviews einplanen.

Ergebnis: kontrollierter ML-Lebenszyklus statt einmaligem Experiment.

Mini-Beispiel: Pipeline-Denken in Python

Das folgende Beispiel ist bewusst klein. Es zeigt nicht "die perfekte Bereinigung", sondern die Grundidee: Transformationen gehören in eine Pipeline, damit Training und spätere Anwendung dieselbe Logik nutzen.

```
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression

numeric_features = ["age", "orders_90d", "support_cases_30d"]
categorical_features = ["customer_segment", "country"]

numeric_pipeline = Pipeline(
    steps=[
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler()),
    ]
)

categorical_pipeline = Pipeline(
    steps=[
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("encoder", OneHotEncoder(handle_unknown="ignore")),
    ]
)

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_pipeline, numeric_features),
        ("cat", categorical_pipeline, categorical_features),
    ]
)

model = Pipeline(
    steps=[
        ("preprocess", preprocess),
        ("classifier", LogisticRegression(max_iter=1000)),
    ]
)
```

```
model.fit(X_train, y_train)
score = model.score(X_valid, y_valid)
```

Der wichtige Punkt: Median, Skalierungsparameter und Kategorien werden beim `fit` aus den Trainingsdaten gelernt. Bei `score` werden dieselben gelernten Transformationen auf Validierungsdaten angewendet. So wird verhindert, dass Validierungsdaten die Datenbereinigung beeinflussen.

In echten Projekten kommen weitere Schritte hinzu: zeitbasierte Splits, gruppierte Splits, Custom Transformer, Feature Stores, Modellversionierung, Datenvalidierung und Monitoring. Das Prinzip bleibt gleich.

Qualitätsregeln als Tests

Datenqualität sollte automatisiert geprüft werden. Ein einfaches Schema für Tests:

```
def test_no_future_dates(df):
    invalid = df["order_date"] > df["prediction_date"]
    assert invalid.sum() == 0

def test_required_columns_present(df):
    required = {"customer_id", "prediction_date", "target"}
    assert required.issubset(set(df.columns))

def test_target_is_binary(df):
    assert set(df["target"].dropna().unique()).issubset({0, 1})

def test_customer_not_in_multiple_splits(train_ids, test_ids):
    assert set(train_ids).isdisjoint(set(test_ids))
```

Solche Tests wirken unspektakulär. Gerade deshalb sind sie wirksam. Sie verhindern, dass sich einfache Fehler in komplexen Modellen verstecken.

Wann Daten nicht bereinigt, sondern das Problem neu definiert werden muss

Manchmal zeigt Datenbereinigung nicht nur kleine Fehler, sondern ein falsches Projektverständnis. Dann sollte man nicht weiter putzen, sondern die Problemdefinition ändern.

Warnsignale:

- Zielvariable ist nicht zuverlässig messbar.
- Daten entstehen erst nach der gewünschten Entscheidung.
- relevante Fälle sind historisch kaum erfasst.
- Labels bilden menschliche Vorurteile oder Prozessfehler ab.
- Datenquellen wurden mehrfach umgestellt und sind nicht vergleichbar.
- Fachbereich kann zentrale Felder nicht erklären.
- Modell soll etwas vorhersagen, das im Datensatz nur als Proxy existiert.
- notwendige Daten dürfen aus Datenschutzgründen nicht verwendet werden.

In solchen Fällen ist "mehr Datenbereinigung" keine Lösung. Besser ist ein Schritt zurück: anderes Ziel, andere Datenquelle, einfachere Regel, manuelle Entscheidungshilfe, Pilot mit neuer Datenerhebung oder kein ML.

Gute ML-Arbeit erkennt auch, wann ein Modell nicht verantwortbar ist.

Verbindung zu Data Engineering und Datenarchitektur

Datenbereinigung wird einfacher, wenn die Datenarchitektur sauber ist. Schlechte Datenmodelle, unklare Schlüssel, fehlende Historisierung, unkontrollierte Freitextfelder und Schattenlisten erzeugen dauerhafte Cleaning-Kosten.

Langfristig sollte man daher nicht nur die Symptome bereinigen, sondern die Ursachen verbessern:

- Pflichtfelder dort einführen, wo sie fachlich nötig sind;
- Referenzdaten zentral pflegen;
- Eingabevalidierung in Quellsystemen verbessern;
- Stammdatenmanagement etablieren;
- Historisierung korrekt modellieren;
- Datenverträge zwischen Systemen definieren;
- Datenkatalog und Verantwortliche pflegen;
- ETL-Pipelines testen;
- Änderungsprozesse für Schema und Semantik einführen.

Der Artikel Begriffsbestimmungen rund um Data hilft bei der sprachlichen Einordnung, wenn Teams zwischen Data Engineering, BI, Analytics und ML unterscheiden müssen.

Ein Entscheidungsrahmen für Cleaning-Strategien

Für jede Datenauffälligkeit kann man fünf Fragen stellen:

1. **Ist der Wert technisch möglich?** Wenn nein, korrigieren, markieren oder ausschließen.
2. **Ist der Wert fachlich plausibel?** Wenn nein, Fachbereich einbinden.
3. **Ist der Wert für das Ziel relevant?** Wenn ja, nicht vorschnell entfernen.
4. **Wäre der Wert in Produktion verfügbar?** Wenn nein, Feature ausschließen oder zeitlich neu bauen.
5. **Kann die Behandlung reproduziert und überwacht werden?** Wenn nein, Pipeline verbessern.

Daraus entstehen vier Grundentscheidungen:

Entscheidung	Wann sinnvoll?	Beispiel
behalten	Wert ist ungewöhnlich, aber real und relevant	Großauftrag, seltener Ausfall
korrigieren	Fehler ist eindeutig und Korrektur begründet	vertauschte Datumsformate
markieren	Unsicherheit oder Spezialfall ist selbst ein Signal	fehlende Angabe, Ausreißerverdacht
ausschließen	Beobachtung gehört nicht zum Lernproblem oder ist unrettbar fehlerhaft	Testkunde, synthetischer Importfehler

Diese Logik verhindert zwei Extreme: naive Datengläubigkeit und überaggressive Reinigung.

Fazit

Datenbereinigung ist im maschinellen Lernen kein Vorzimmer der eigentlichen Arbeit. Sie ist ein zentraler Teil der Modellierung. In ihr wird entschieden, welche Realität das Modell sieht, welche Fehler es ignorieren soll, welche Signale erhalten bleiben und welche Annahmen später im Betrieb überwacht werden müssen.

Ein gutes ML-Modell entsteht nicht durch einen starken Algorithmus allein. Es entsteht durch eine saubere Zieldefinition, belastbare Datenquellen, reproduzierbare Cleaning-Pipelines, zeitlich korrekte Features, faire Evaluation, fachliche Fehleranalyse und laufendes Monitoring. Datenbereinigung verbindet all diese Ebenen.

Die wichtigste Haltung lautet: Daten werden nicht gereinigt, damit sie ordentlich aussehen. Sie werden so aufbereitet, dass ein Modell aus ihnen das richtige Problem lernen kann.

Weiterführende Literatur und Quellen

- D. Sculley et al.: *Hidden Technical Debt in Machine Learning Systems*. Ein grundlegender Text darüber, warum ML-Systeme durch Datenabhängigkeiten, Pipeline-Komplexität und versteckte Kopplungen technisch riskant werden.
- Eric Breck et al.: *The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction*. Praktischer Rahmen zur Bewertung produktiver ML-Systeme, einschließlich Daten- und Pipeline-Tests.
- Foster Provost, Tom Fawcett: *Data Science for Business*. Gute Einführung in die Verbindung zwischen Geschäftsproblem, Daten, Modell und Evaluation.
- Trevor Hastie, Robert Tibshirani, Jerome Friedman: *The Elements of Statistical Learning*. Mathematisch anspruchsvoller Klassiker zu statistischem Lernen.
- Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani: *An Introduction to Statistical Learning*. Zugänglichere Einführung in viele ML-Grundlagen.
- scikit-learn Dokumentation zu Pipelines, Preprocessing und Model Evaluation: nützlich für reproduzierbare Transformationen und saubere Trennung von Training und Validierung.
- pandas Dokumentation zu Missing Data, Datentypen und Joins: praktisch für explorative Datenbereinigung und tabellarische Datenaufbereitung.
- Great Expectations, Pandera und TensorFlow Data Validation: Beispiele für Frameworks, mit denen Datenqualitätsregeln automatisiert und wiederholbar geprüft werden können.

Über den Autor

Erhard Rainer berät kleine und mittlere Unternehmen in Österreich zu Business Intelligence, Machine Learning und KI-Agenten. Er hat 15 Jahre Erfahrung in Data Analytics, davon 10 Jahre in der Leitung von Teams und Departments in Konzernen, und ist Dozent für betriebswirtschaftliche und Data-Analytics-Fächer.

Sprechen Sie mit mir über Ihre Daten: Das Erstgespräch ist kostenlos.

https://www.thinkbigger.at/contact.html#anfrage

© 2026 Think BIGGER – Erhard Rainer. Dieses PDF gibt den Stand des Online-Artikels vom 06.07.2026 wieder; maßgeblich ist die Online-Fassung unter https://www.thinkbigger.at/wissen/datenbereinigung-maschinelles-lernen/