Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

DM	Data Mining
FW	Fehlende Werte
MAR	Missing at random
MCAR	Missing completely at random
ML	Maschinelles Lernen
NMAR	Not missing at random

1 Einführung

Die fortschreitende Digitalisierung generiert eine immense Menge an Daten, die ein beträchtliches Potenzial bergen. Diese Daten können wertvolle Einblicke und Erkenntnisse liefern, die zu besseren Geschäftsentscheidungen, innovativen Entdeckungen und effizienteren Prozessen führen. Data Mining (DM), der Prozess der Extraktion nützlicher Informationen aus großen Datenmengen, stellt dabei ein bedeutsames Werkzeug dar. Zugleich gilt es bei der Umsetzung solcher Projekte jedoch zahlreiche Herausforderungen zu berücksichtigen, die den Erfolg maßgeblich beeinflussen können (vgl. Abdel-Karim, 2022: 66 f.; vgl. Baskar et al., 2013: 1).

Die Qualität der verfügbaren Daten stellt eine der größten Herausforderungen beim DM dar. Rohdaten sind oft unvollständig, inkonsistent und enthalten Fehler oder irrelevante Informationen. Solche Mängel können die Genauigkeit und Verlässlichkeit der Analyseergebnisse erheblich beeinträchtigen oder verfälschen. Darüber hinaus sind Daten häufig in verschiedenen Formaten gespeichert, was eine einheitliche Analyse erschwert. Um eine einheitliche Basis für die Analyse zu schaffen, müssen die Daten daher zunächst in ein einheitliches Format gebracht werden (vgl. Cleve/Lämmel, 2016: 202 ff.).

Ein weiteres Problem stellt die Größe der Datenmenge dar. Die Analyse von Daten aus vielen verschiedenen Quellen führt häufig dazu, dass nicht alle Attribute für die Auswertung relevant sind. Die Auswahl der richtigen Merkmale ist von entscheidender Bedeutung, um die Komplexität der Modelle zu reduzieren und die Effizienz der Analyse zu steigern (vgl. Han et al., 2011: 420 f.). Zudem ist eine Transformation der Daten in eine geeignete Struktur erforderlich, um eine effiziente Verarbeitung durch die Algorithmen des DM zu gewährleisten (Papp et al., 2022: 78 f.).

Die Datenvorbereitung ist daher ein unverzichtbarer Schritt im DM-Prozess. bringen, die für die Analyse geeignet ist (vgl. Han et al., 2011: 8).

Ziel dieser Arbeit ist es, verschiedene Ansätze und Techniken der Datenvorbereitung vorzustellen. Dabei werden Methoden beschrieben, die helfen, die zuvor genannten Herausforderungen zu bewältigen und die Datenqualität zu verbessern. Dazu gehören Techniken zur Datenbereinigung, wie das Erkennen und Beheben von Fehlern, Inkonsistenzen und widersprüchlicher Daten, sowie Methoden zur Diskretisierung kontinuierlicher Daten und die Zerlegung diskreter Werte in binäre Attribute. Diese Arbeit zeigt neben theoretischen Konzepten auch praktische Beispiele, um die Anwendung der verschiedenen Techniken zu veranschaulichen.

2 Fehlende Werte

In DM-Datensets kommt es sehr häufig vor, dass einige Werte fehlen. Dies passiert dann, wenn diese beispielsweise nie aufgenommen wurden oder während des Aufnahmeprozesses verloren gegangen sind. Manuelle Eintragungsprozesse, Geräte- oder Messfehler können die Ursache für das Auftreten von fehlenden Werten (FW). Das Fehlen von Werten der Attribute kann zu verschiedenen Problemen während des DM-Prozesses führen:

Unvollständige Mustererkennung: Daten können schlechter oder gar nicht mehr miteinander verglichen werden
Keine Möglichkeit zur Kategorisierung von Attributen mit fehlenden Werten
Komplikationen arithmetischer Algorithmen bei der Handhabung und Analyse des Datensets
Verzerrung der Ergebnisse (bias)

Alles in allem führen FW zu ineffizienteren und weniger zuverlässigen DM-Ergebnissen, weshalb die FW in der Datenvorbereitung gesondert behandelt werden müssen.
Um diesen Problemen gegenwirken zu können, bietet die Literatur viele verschiedene Ansätze zur Behandlung von FW. Dabei gibt es keine Optimallösung, sondern es kommt immer auf den Anwendungsfall an, welche Methode für welches Datenset am besten geeignet ist. Die Wahl der Methode kann dabei entscheidend für die Qualität der Datenbereinigung, da es bei der Erhebung fehlender Werte außerordentlich wichtig ist, dass der Anwender nicht selbst derjenige ist, der Verzerrungen in das Datenset einpflegt und die verfügbaren Informationen wahrheitsgetreu an das DM-Werkzeug übertragen werden können. Für die Wahl der korrekten Methode wird zwischen drei verschiedenen Fällen von FW differenziert (vgl. García et al., 2014: 59 f.; vgl. Pyle, 1999: 257; vgl. Alexandropoulos, 2019: 5 f.; vgl. Li, 2019: 3).

2.1 Fehlertypen

Missing completely at random (MCAR):
MCAR bedeutet, dass FW komplett zufällig sind und von keiner anderen Variable im Datensatz und auch nicht von der Variable selbst abhängen. Beispielsweise könnten bei einer Umfrage Werte zum Alter der befragten fehlen. Bei MCAR wären die FW nicht abhängig vom Alter der Person und auch nicht abhängig von anderen Angaben, welche die Personen getätigt haben (vgl. Enders, 2010: 7 f.; vgl. Donders, 2006; vgl. García et al., 2014: 61 ff.).

Missing at random (MAR):
MAR bedeutet, dass FW nicht von der Variable selbst abhängen aber sie von einer anderen Variable im Datensatz abhängig sind. Im Beispiel des Alters bei der Befragung wäre hier also das Fehlen der Angabe des Alters ebenfalls nicht abhängig

vom Alter der Person selbst, jedoch beispielsweise vom Geschlecht der Person (vgl. Enders, 2010: 6 f.; vgl. Donders, 2006; vgl. García et al., 2014: 61 ff.).

Not missing at random (NMAR):
NMAR bedeutet, dass FW von der Variable selbst und eventuell auch von anderen Variablen im Datensatz abhängen. Um erneut das obige Beispiel anzuwenden, würden die FW des Alters von dem eigenen Alter der Person und eventuell auch von dem Geschlecht der Person abhängen (vgl. Enders, 2010: 8 f.; vgl. Donders, 2006; vgl. García et al., 2014: 61 ff.).

2.2 Löschen von Einträgen

Die schnellste und unkomplizierteste Art FW zu behandeln, besteht darin, Einträge mit mindestens einem FW komplett aus dem Datenset zu Löschen oder zu ignorieren. Diese Methode sollte allerdings nur dann angewendet werden, wenn die FW MCAR sind, da das Datenset ansonsten verzerrt oder komplett unbrauchbar wird. Außerdem mindert diese Methode selbst im Optimalfall die Aussagekraft des DM-Prozesses durch das Fehlen eben dieser Attribute sinkt. Besonders dann, wenn viele Werte in verschiedenen Attributen fehlen und dadurch wegfallen (vgl. García et al., 2014: 63 ff.).

2.3 Mittelwert-Imputation

Eine weitverbreitete, simple Methode ist die Mittelwerts-Imputation Methode. Sie ist eine der einfachsten Optionen, um FW zu erheben. Dabei werden FW eines Attributs anhand des Mittelwerts der existierenden Werte ermittelt und für alle FW eingesetzt (siehe Abbildung 1). Ähnlich wie beim Löschen der Attribute wird diese Methode gern angewendet, wenn nur ein kleiner Prozentsatz der Daten fehlt oder davon ausgegangen werden kann, dass die FW MCAR sind. Allerdings würde sogar für FW welche MCAR sind die Varianz der Ergebnisse deutlich zu gering ausfallen, da alle FW mit demselben Wert, dem erhobenen Mittelwert ersetzt werden. Wird diese Methode für FW eingesetzt welche MAR oder gar NMAR sind würden sogar zweierlei Fehler auftreten. Einerseits wird wie bereits bei MCAR die Varianz deutlich zu gering sein und anderseits würde das berechnete Mittel vom tatsächlichen Mittel abweichen da die Abhängigkeit zwischen den FW und einer andere Variable nicht beachtetet werden würde. Dies wird anhand des Beispiels in Abbildung 1 deutlich, da hier die Job Performance in Abhängigkeit zum IQ steht, jedoch alle Y-Werte unterhalb von einem IQ unter 100 fehlen (vgl. Enders, 2010: 42 f.; vgl. García et al., 2014: 64; vgl. Han et al., 2011: 83).

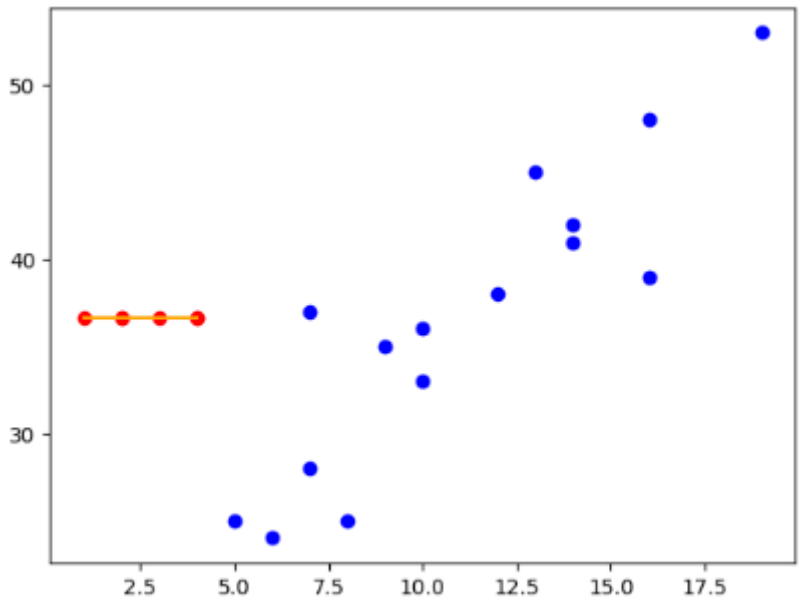


Abbildung 1: Mittelwerts-Imputation (Eigene Darstellung nach Enders, 2010: 43)

2.4 Regressions-Imputation

Für die Behandlung von FW die MAR sind eignet sich unter anderem die Regressions-Imputation Methode. Dabei werden die FW über eine Regressionsgerade durch die vorhandenen Daten des Attributs ermittelt. Mit Hilfe dieser Methode werden die Beziehungen zwischen den Variablen berücksichtigt, wodurch keine Verzerrung bei der Erhebung des Mittelwerts entsteht (siehe Abbildung 2) (vgl. Enders, 2010: 44 ff.; vgl. Alexandropoulos, 2019).

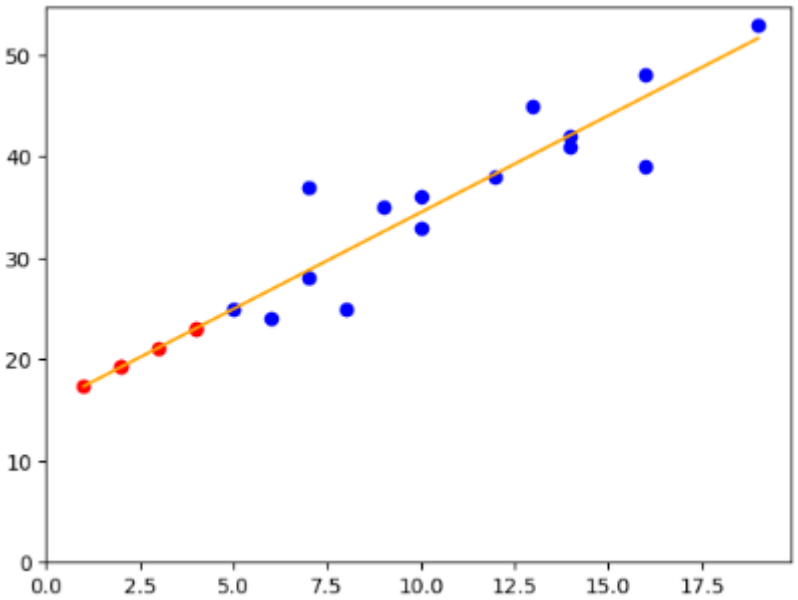


Abbildung 2: Regressions-Imputation (eigene Darstellung nach Enders, 2010: 46)

2.5 Stochastische Regressions-Imputation

Allerdings genügt es noch nicht um das Problem der fehlerhaften Varianz zu beheben. Jedoch kann dieses Problem behoben werden, indem man eine stochastische

Regressions-Imputation anwendet. Dabei wird auf den erhobenen Werten ein zufälliges Rauschen, welches auf der Verteilung der vorhandenen Werte basiert, angewendet. Dadurch können FW, welche MAR sind erhoben werden, ohne die Ergebnisse zu verzerren (siehe Abbildung 3) (vgl. Enders, 2010: 46 ff.).

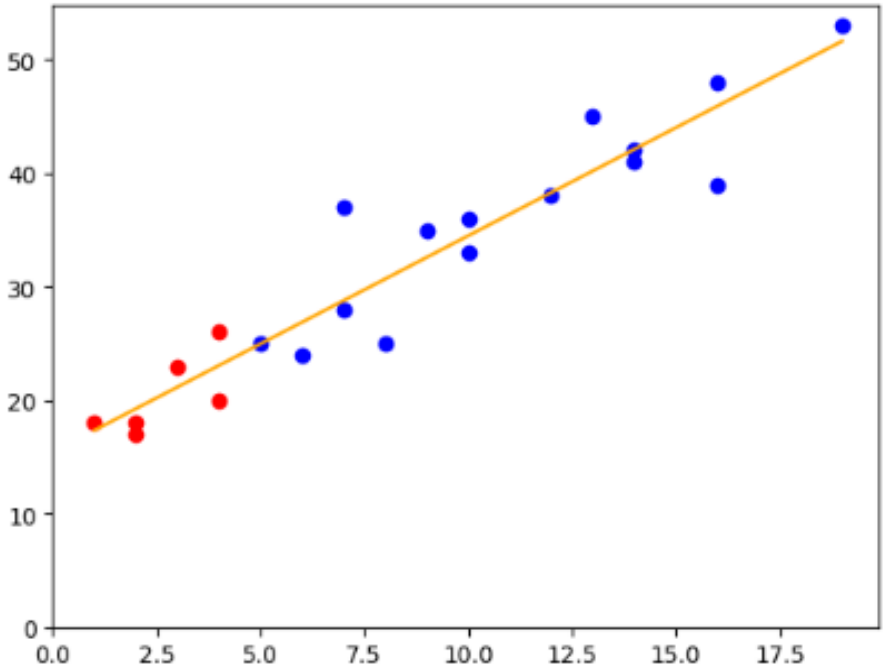


Abbildung 3: stochastische Regressions-Imputation (eigene Darstellung nach Enders, 2010: 48)

2.6 Multiple Imputation

Das Problem, welches bei Anwendung der stochastischen Regressions-Imputation auftritt, ist die Missachtung von Unsicherheiten oder Schätzfehlern bei der Erhebung der FW. Um dem entgegenzuwirken kann beispielsweise die Multiple Imputation herangezogen. Diese Methode besteht aus drei Phasen:

Imputation Phase:

Diese Phase unterscheidet sich kaum von einer Single Imputation Methode, heißt, dass auch jeder dieser Methoden angewendet werden kann. In diesem Beispiel nutzen wir erneut die stochastische Regressions-Methode, da diese die Ergebnisse mit der geringsten Verzerrung erheben konnte. Der einzige Unterschied besteht nun darin, dass diese Imputation, wie der Name schon sagt, mehrfach mit unterschiedlichen existierenden Werten durchgeführt wird. Beispielsweise werden für ein Datenset mit >1000 Werten für eine Imputation zufällige 100 Werte ausgewählt und für die nächste Imputation 100 andere Werte. Theoretisch genügen 5-10 Imputationen, um ein unverzerrtes Ergebnis zu erhalten, es ist jedoch empfehlenswert zwischen 20 und 100 Imputationen durchzuführen, um noch genauerer Werte zu erheben, was mit der heutigen Rechenleistung auch kein Problem darstellen sollte (vgl. Enders, 2010: 187 ff.; Pyle, 1999: 252 f.; García et al., 2014: 68-72).

Analysis Phase:
Im nächsten Schritt werden die erhobenen Werte für jede Imputation separat eingesetzt, wodurch n komplette Datensets entstehen, welche nun separat voneinander analysiert werden. In diesem Fall soll eine simple Mittelwerts-Analyse als Beispiel dienen. Dadurch werden für n Datensets ein Mittelwert erhoben (vgl. Enders, 2010: 187 ff.; Pyle, 1999: 252 f.; García et al., 2014: 68-72).

Pooling Phase:
Der letzte Schritt der Methode ist die Pooling Phase. Dabei wird aus n Mittelwerten ein Aggregat ermittelt (beispielsweise erneut ein Mittelwert), welches nun final als Wert für den FW des originalen Datensets verwendet wird. Weiterhin wird in diesem Schritt die Varianz bzw. der Standardfehler dieser Mittelwerte berechnet, wodurch mehr Unsicherheiten und Schätzfehler in der Erhebung der FW vermieden werden kann.

Für das Erheben von FW des Falles NMAR ist keine der behandelten Methoden ideal. Es existieren sogenannte Selection models, welche es potenziell ermöglichen Verzerrung, welche durch MAR-Methoden verursacht werden würden, zu verringern oder zu eliminieren. Jedoch treffen diese Modelle teilweise schwer zu rechtfertigende Annahmen, weshalb häufig auch MAR-Methoden zum Einsatz kommen, um FW des Falles NMAR zu erheben. Es wird demnach ein gewisser Grad an Verzerrung der Ergebnisse akzeptiert wird (vgl. Enders, 2010: 187 ff.; Pyle, 1999: 252 f.; García et al., 2014: 68-72).

Weitere Methoden zur Erhebung von FW sollen hier abschließend lediglich genannt werden, um den Umfang der Ausarbeitung zu begrenzen. Zur Erhebung von MCAR oder MAR FW sind die Hot-Deck imputation, similar response pattern imputation, averaging available items oder die last observation carried forward Methode. Für das Erheben von NMAR eignet sich neben der Mehrfachimputation die Maximum-Likelihood-Methode oder die besagten Selection models. Wobei erwähnt werden muss, dass keine dieser „NMAR-Methoden“ perfekt funktioniert und mehr oder weniger zur Verzerrung der Ergebnisse führen.

3 Behandlung widersprüchlicher Daten

Eine Teilaufgabe der Datenvorbereitung besteht darin, widersprüchliche, fehlerhafte und ungenaue Daten zu korrigieren. Datenfehler können durch verschiedenste Faktoren entstehen: Änderungen der Datenbankstruktur, fehlerhafte Datenquellen, Fehler bei der Datenmigration, veraltete Daten oder auch menschliches Versagen – um nur wenige aufzuzählen (vgl. Cleve/Lämmel, 2016: 203).
Nachfolgende Abbildung veranschaulicht verschiedene Datenfehler.

Pers. Nr.	Name	Geschlecht	Alter	PLZ	Straße	Nr.
8	Fischer, Herbert	m	53	45895	Blumenstr.	25
45	Herbert Fischer	männl.	null	45895	Blumenstr.	0
8	Müller, Hilda	w	-5	47925	Flussstr.	9
12	Muster, Inge	fem	38	23575	Straußstr.	5

Abbildung 4: Tabelle mit Datenfehler (Eigene Abbildung in Anlehnung an Cleve/Lämmel, 2016: 203)

Die grau hinterlegten Felder verdeutlichen die vorliegenden Datenfehler in diesem Datensatz. So ist die Personalnummer 8 doppelt vergeben. Der Datensatz in Zeile 1 und 2 doppelt, mit Unterschieden im Namen, Alter und der Straßennummer. Die Einträge „fem“ und „männl.“ differenzieren sich von der eigentlichen Bezeichnung des Geschlechts („m“ oder „w“) und das Alter in Zeile 2 und 3 ist nicht vorhanden bzw. fehlerhaft.
Die verschiedenen Fehler können in syntaktische und semantische Datenfehler untergliedert werden (vgl. Cleve/Lämmel, 2016: 203; vgl. Han et al., 2011: 84 f.). Deren Unterschiede und Beispiele werden im Folgenden verdeutlicht.

3.1 Syntaktische Datenfehler

Syntaktische Fehler manifestieren sich in Form von Problemen im Format oder der Struktur der Daten. Diese können typischerweise durch Verstöße gegen vordefinierte Datentypen oder Formatregeln erkannt werden.
Datentypverstöße entstehen, wenn ein Wert nicht dem erwarteten Datentyp entspricht. Beispielsweise wenn ein Feld eine Zahl erwartet, aber einen Text enthält („elf“ statt „11“) oder ein Datumsfeld ein ungültiges Datum enthält („30.02.2024“).
Formatierungsfehler liegen vor, wenn Daten einem festgelegten Format nicht entsprechen. Klassische Beispiele hierfür sind das Format von Telefonnummern („1234567890“ statt „123-456-7890“), eine unzulässige Postleitzahl („587“ statt „58975“), oder verschiedene Formatierungen für Datumsangaben („23.08.2022“ und „2022/08/23“) oder Maßeinheiten („5,54m“ und „18,18ft“) (vgl. Cleve/Lämmel, 2016: 220; vgl. Han et al., 2011: 83 ff.).

3.2 Semantische Datenfehler

Semantische Fehler betreffen die Bedeutung und die Richtigkeit der Daten im Kontext. Da sie logische oder inhaltliche Widersprüche beinhalten, sind sie oft schwieriger zu identifizieren.

Inhaltliche Widersprüche beschreiben Daten, die nicht zueinander passen oder logisch inkonsistent sind. Beispielsweise wenn ein Geburtsdatum in der Zukunft, das Datum des Vertragsabschlusses vor dem Geburtsdatum des Vertragspartners liegt, das Alter einer Person nicht zum Geburtsjahr passt oder die Postleitzahl und der Wohnort eines Kunden sich widersprechen. Zusätzlich kann auch das Vorkommen von doppelten Datensätzen zu Problemen führen, da bei vielen DM-Algorithmen diese Datensätze aufgrund Mehrfachauftretens ein höheres Gewicht erhalten (vgl. Batini et al., 2009: 7 f.; vgl. Cleve/Lämmel, 2016: 206 und 212).
Unplausible Werte können ebenfalls auftreten. Diese Werte sind formal korrekt, erscheinen allerdings unplausibel und unwahrscheinlich. Beispielsweise bei einem Gehalt von 800.000€ für einen Einstiegsjob oder einer durchschnittlichen Raumtemperatur von über 100°C (vgl. Cleve/Lämmel, 2016: 211).
Das **Verletzen der referenziellen Integrität** ist ein weiterer klassischer Fehler. Ein solcher tritt auf, wenn unterschiedliche Datenquellen zusammengeführt werden und eine Tabelle auf einen nicht existierenden Referenzwert in einer anderen Tabelle verweist. Zum Beispiel wenn ein Bestellposten auf eine nicht existierende Bestellung verweist (vgl. Cleve/Lämmel, 2016: 206 f.).
Der Umgang mit fehlerhaften bzw. widersprüchlichen Daten stellt sich als aufwendig und kompliziert dar. Wenn möglich, ist es empfehlenswert im Vorfeld entsprechende Validierungen und Regeln für die Datengewinnung bzw. Datengenerierung zu definieren um Fehler, Widersprüche oder Ungenauigkeiten auf ein Minimum zu reduzieren (vgl. Papp et al., 2022: 153 ff.). Während der Datenhaltung helfen Konsistenzprüfungen bei der Sicherstellung der Datenqualität. Eine regelmäßige Überprüfung (in angemessenen Zeitintervallen) führt dazu, dass Inkonsistenzen, Fehler, Widersprüche etc. frühzeitig erkannt und entsprechende Maßnahmen zur Reduzierung dieser eingeleitet werden können, wodurch sich die Datenqualität nachhaltig verbessert (vgl. Cai/Yangyong, 2015: 3 ff.; vgl. Solanki, 2024).
Wenn dennoch Fehler auftreten, müssen diese häufig durch Fachpersonal (besonders bei komplexen Widersprüchen und Fehlern) behoben werden. Eine Möglichkeit bietet dabei die Zuhilfenahme von weiteren Datenquellen (externe Datenbanken o.ä.) und oder Datensätzen, welche als Referenzen für die fehlerhaften Datensätze dienen können. Eine weitere Option ist das explizite Löschen eines fehlerhaften bzw. widersprüchlichen Datensatzes. Je nach Anwendungsfall ist jedoch abzuwägen, ob dies als sinnvoll angesehen werden kann. Außerdem hat das Löschen den Nachteil, dass die Datenmenge reduziert wird. Sind zu wenige aussagekräftige Datensätze vorhanden, verschlechtert dies auch die Anwendungsmöglichkeiten von DM-Algorithmen (vgl. Cleve/Lämmel, 2016: 212).

4 Diskretisierung kontinuierlicher Daten

Hauptziel der Diskretisierung ist es, kontinuierliche Attribute in Diskrete umzuwandeln, indem diese in Intervalle gegliedert und mit kategorischen Werten assoziiert werden und somit quantitative Daten in qualitative Daten umgewandelt werden. Das ist dann erforderlich, wenn ein Mining-Algorithmus schlecht mit kontinuierlichen Werten umgehen kann oder die Komplexität des Datensets verringert werden soll, um die Effizienz des Prozesses zu erhöhen.
Was sind kontinuierliche Daten? Kontinuierliche Daten sind Attribute, dessen Werte jeden möglichen Wert innerhalb eines Intervalls annehmen können. Beispielsweise könnte eine Zahl zwischen 1 und 10 sowohl den Wert 1, 1.3, 2.45, 5.45321, usw. annehmen. Ein typisches Merkmal für kontinuierliche Daten ist die Messbarkeit (bspw. Temperaturen). In einem bestimmten Bereich können solche Daten unendlich viele Werte annehmen. Viele DM-Algorithmen sind allerdings so entworfen, dass sie nur diskrete Werte erwarten, weshalb eine Diskretisierung unumgänglich ist. Diskrete Daten können in einem Bereich nämlich nur bestimmte, vordefinierte Werte annehmen. Beispielsweise das Ergebnis eines Würfelwurfs oder festgelegte Altersgruppen (jung, mittelalt, alt). Der Unterschied zu den kontinuierlichen Daten besteht darin, dass die Werte endlich und zählbar statt unendlich und messbar sind (vgl. García et al., 2014: 249 f.; vgl. Hemada et al., 2013; vgl. Kotsiantis et al., 2006).

Für die Diskretisierung gibt es eine Menge verschiedener Methoden allerdings folgen sie alle einem ähnlichen Ablauf:

Sortieren: Kontinuierliche Werte werden auf- oder absteigend sortiert.
Auswahl eines Grenzwerts: Im nächsten Schritt bestimmt die Methode entweder die besten Grenzwerte damit die kontinuierlichen Werte an diesen Stellen aufgespalten werden können oder die besten angrenzenden Intervallpaare, um diese zusammenführen zu können (je nach Methode).
Aufteilung/Zusammenführung: Bei einer Aufteilung werden die kontinuierlichen Werte an den Grenzwerten aufgespalten und in separate Partitionen gegliedert. Dies wird so lange durchgeführt, bis ein Stoppkriterium erreicht wird. Bei der Zusammenführung werden statt optimaler Grenzwerte, die besten beieinander liegenden Intervalle gefunden und iterativ zusammengeführt, bis ein Stoppkriterium erreicht wird.
Stoppkriterium: Gibt den Endpunkt der Diskretisierung an. Dabei muss zwischen einer geringen Arithmetik, welche die Komplexität geringhält und einer hohen Genauigkeitskonsistenz abgewogen werden. Dies kann simpel, über eine eingangs festgelegte Anzahl an Intervallen oder komplexer, durch die Schätzung mithilfe einer Funktion passieren (vgl. García et al., 2014: 249 f.; vgl. Hemada et al., 2013; vgl. Kotsiantis et al., 2006).

4.1 Charakteristika

Die verschiedenen Methoden unterscheiden sich durch ihre unterschiedlichen Charakteristika, denen sie bei der Diskretisierung folgen.

Unbeaufsichtigt gegen Beaufsichtigt:
Eine unbeaufsichtigte Diskretisierung gliedert die Daten ungeachtet der Attributsinformationen in die entsprechenden Intervalle. Eine beaufsichtigte Methode hingegen nutzt diese Informationen, um den Diskretisierungsprozess bei der Einteilung der Werte in die entsprechenden Intervalle zu unterstützen. Beaufsichtigte Methoden haben einen höheren Berechnungsaufwand, dafür können in der Theorie die optimale Anzahl an Intervallen und besten Trennpunkte dieser Intervalle automatisch bestimmen (vgl. García et al., 2014: 252; vgl. Hemada et al., 2013; vgl. Kotsiantis et al., 2006).

Lokal gegen Global:
Es wird von einer lokalen Diskretisierung, wenn der Prozess nur auf einen Teil der Attributswerte angewendet, bzw. verschiedene Teile auf verschiedene Arten diskretisiert werden. Außerdem bieten diese Methoden die Möglichkeit den Prozess innerhalb des Lernprozesses des Modells angewendet werden zu können. Bei einer globalen Diskretisierung hingegen wird der Prozess einheitlich auf den kompletten Datensatz eines Attributs angewandt. Diese Methoden werden ausschließlich in der Datenvorbereitung angewendet, sind leichter zu implementieren, interpretieren und reproduzieren als lokale Methoden (vgl. García et al., 2014: 252 f.; vgl. Hemada et al., 2013; vgl. Kotsiantis et al., 2006).

Direkt gegen inkremental:
Direkte Methoden unterteilen den Bereich von k Intervallen gleichzeitig auf. Dafür benötigen die Methoden einen weiteren Input des Nutzers, welche die Anzahl an geforderten Intervallen übergibt. Inkrementale Methoden hingegen starten mit einer simplen Diskretisierung und verbessern diese schrittweise, bis ein bestimmtes Kriterium erreicht ist (vgl. García et al., 2014: 252 f.; vgl. Hemada et al., 2013; vgl. Kotsiantis et al., 2006).

Aufteilen (top-down) gegen Zusammenführen (bottom up):
Wie bereits im Ablauf ausgeführt gibt es Methoden, welche Intervalle anhand von Grenzwerten bestimmen und Methoden, welche die besten Nachbarintervalle ermittelt und zusammenführt.

Univariat gegen multivariat:
Multivariate Methoden berücksichtigen bei der Auswahl der Grenzwerte alle Attribute im Datenset. Zusätzlich können während des Prozesses bereits ausgewählte Grenzwerte für ein Attribut angepasst werden, sollten sich bei der Interaktion mit anderen Attributen Zusammenhänge zwischen den Attributen ergeben. Dem

entgegen stehen die univariaten Methoden, welche jeweils ein Attribut nach dem anderen behandeln. Die daraus resultierenden Ergebnisse können dabei zu keinem Zeitpunkt im Prozess mehr angepasst werden (vgl. García et al., 2014: 252; vgl. Hemada et al., 2013; vgl. Kotsiantis et al., 2006).

4.2 Equal-width Binning Methode

Beim Equal-width Binning wird eine Reihe Werte in gleichgroße Intervalle oder *bins* geteilt. Die Anzahl der *bins* muss hierbei vom Anwender selbst bestimmt werden. Reichen die Werte beispielsweise von 0 bis 100 und wir wollen diese auf insgesamt 10 *bins* aufteilen, wird jede Intervallbreite 10 betragen. Die Vorteile der Methode belaufen sich auf die simple Implementation und Interpretation. Außerdem wird durch die Anwendung dieser Methode die Verteilung der Daten bewahrt. Allerdings können durch diese Methode leere oder spärlich gefüllte *bins* entstehen, sollten die Daten verzerrt sein oder Ausreißer beinhalten. Dadurch wird der Informationsgehalt und die Genauigkeit der Analyse verringert. Equal-width Binning eignet sich nach den Ergebnissen von Putri dann, wenn Datensets eine hohe Datenkomplexität und viele Attribute beinhalten (vgl. García et al., 2014: 260; vgl. Putri et al., (2023).

4.3 Equal-frequency Binning Methode

Anstatt gleich großer Intervallbreiten werden beim Equal-frequency Binning die Werte so aufgeteilt, dass jeder *bin* dieselbe Anzahl an Werten beinhaltet. Wenn ein Datenset insgesamt 100 Einträgen fast und sich der Anwender für 10 *bins* entschieden hat werden demnach die Grenzen der Intervalle so gesetzt, dass in jedem *bin* 10 Einträge stehen. Dadurch können verzerrte Daten und Ausreißer besser gehandhabt werden, da immer balancierte *bins* generiert werden. Der Nachteil besteht darin, dass dadurch die Verteilung der Daten verzerrt und unregelmäßige Intervallbreiten entstehen können. Dadurch kann die Analyse komplexer und weniger intuitiv werden. Equal-frequency Binning eignet sich nach den Ergebnissen von Putri dann, wenn Datensets eine geringe Datenkomplexität oder weniger Attribute beinhalten (vgl. García et al., 2014: 260 f.; vgl. Putri et al., (2023).

5 Zerlegung diskreter Werte in binäre Attribute

Neben der Diskretisierung von kontinuierlichen Daten, spielt die Umwandlung von diskreten Merkmalen in binäre Attribute eine weitere Rolle in der Datenvorbereitung für anschließende Datenanalysen und maschinelles Lernen (ML). Wie im Kapitel 2.3 zuvor erklärt, sind diskrete Merkmale Daten, die endliche oder abzählbar unendliche Ausprägungen haben. Die Zerlegung diskreter Werte in binäre Attribute ist ein beliebtes Verfahren im Bereich der Datenvorbereitung. Das Ziel ist dabei die entsprechenden Variablen in ein numerisches bzw. binäres Format zu transformieren, ohne dass die eigentliche Bedeutung verloren geht (vgl. panData, 2024).

Diese Transformation ist für die spätere Anwendung von DM- und ML-Algorithmen (wie bspw. lineare und logistische Regression, Entscheidungsbäume, k-means Clustering und weitere) von Bedeutung welche auf mathematischen und statistischen Modellen basieren und entsprechend numerische Eingaben erfordern. Kategoriale Variablen, welche im Textformat vorliegen können in diesen Algorithmen und Modellen nicht direkt verarbeitet werden (vgl. Chancen, 2019: 4; vgl. panData, 2024). Um diskrete kategoriale Variablen wie bspw. Farben (rot, blau, grün, gelb), Wochentage (Montag, Dienstag, Mittwoch, …), Länder (Kanada, Mexiko, England, USA, …) in numerische bzw. binäre Formate umzuwandeln gibt es verschiedene Herangehensweisen:

5.1 Label Encoding

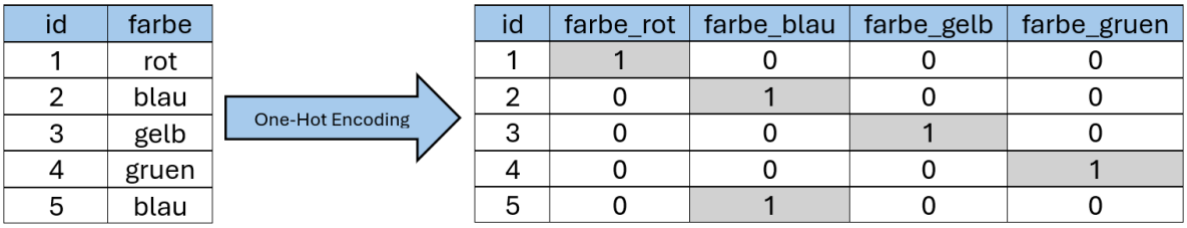
Beim Label Encoding erhält jede Ausprägung einer diskreten kategorialen Variable einen eindeutigen numerischen Wert (integer). Das bedeutet, dass die Ausprägungen der Variable durch 1 bis n (n entspricht Anzahl der Ausprägungen der Variable) aufeinanderfolgenden Zahlen ersetzt werden. Angenommen es liegt eine Variable Farbe mit vier Ausprägungen (rot, blau, gelb und grün) vor, so erhalten die Ausprägungen folgende Werte: rot = 1, blau = 2, gelb = 3 und grün = 4 (vgl. Liu et al., 2021: 4; vgl. Bilal et al., 2022: 3).

Vorteil von dieser Methode ist die Einfachheit und schnelle Ausführbarkeit. Problematisch ist allerdings, dass diese Wertezuordnung eine Reihenfolge oder Ordnung implizieren könnte. Die später angewendeten Algorithmen könnten fälschlicherweise annehmen, dass die Farbe rot = 1 weniger Wert oder kleiner ist als die Farbe gelb = 3. Diese Annahme soll vermieden werden, da die eigentliche Bedeutung der Ausprägungen nicht verloren gehen und die Daten nicht missinterpretiert werden sollen (vgl. panData, 2024).

5.2 One-Hot Encoding

Die zuvor beschriebene Problematik löst das One-Hot Encoding. Bei dieser Methode wird für jede mögliche Ausprägung der Variable eine eigene binäre Spalte erstellt. Jede Ausprägung erhält demnach für die zutreffende binäre Spalte den Wert 1 und für jede nichtzutreffende Spalte den Wert 0. Dadurch wird sichergestellt, dass eine Fehlinterpretation der Daten und jegliche Beziehung zwischen den vorhandenen Ausprägungen eliminiert werden (vgl. Bilal et al., 2022: 107766).

Für die zuvor beschriebene Variable Farbe mit den Ausprägungen rot, blau, gelb und grün entspricht das Ergebnis nach der Anwendung des One-Hot Encodings demnach: rot = [1, 0, 0, 0]; blau = [0, 1, 0, 0]; gelb = [0, 0, 1, 0]; grün = [0, 0, 0, 1] (vgl. panData, 2024). Dieser Prozess wird durch folgende Abbildung veranschaulicht.



id	farbe
1	rot
2	blau
3	gelb
4	gruen
5	blau

id	farbe_rot	farbe_blau	farbe_gelb	farbe_gruen
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	1	0	0

Abbildung 5: Prozess One-Hot Encoding (Eigene Darstellung, in Anlehnung an panData, 2024)

Nachteilig an dieser Methode ist, dass sie zu einer hohen Anzahl an Spalten führen kann, je nachdem wie viele Ausprägungen die Variable hat und durch diese Kardinalität die Dimensionalität, Speicherressourcen und Berechnungslaufzeit des Datensatzes erhöht wird (vgl. Bilal et al., 2022: 107766; vgl. Cleve/Lämmel, 2016: 204).
Dieser Problematik können andere Methoden wie das Targed-based Encoding oder das Feature-Hashing entgegenwirken. Aufgrund des begrenzten Umfangs dieser Arbeit werden diese allerdings nicht weiter betrachtet.

5.3 Binarization

Eine weitere Methode, um diskrete Werte in binäre Attribute zu transformieren bietet Binarization. Dabei werden Variablen vereinfacht, welche lediglich zwei Ausprägungen haben wie bspw.:

- Geschlecht = männlich oder weiblich
- Brillenträger = Ja oder Nein
- Diabeteserkrankung = Wahr oder Falsch

Durch die begrenzte Anzahl an Ausprägungen werden eine einfache Interpretation und Manipulation der Daten bereitgestellt. Bei der Anwendung werden die vorliegenden Ausprägungen entsprechend binär zugeordnet. Somit wird bspw. bei der Variable Geschlecht, einem Mann der Wert 1 und einer Frau der Wert 0 zugeordnet (vgl. panData, 2024).

6 Fazit

Mit dieser Hausarbeit sollten Möglichkeiten zur Datenvorbereitung von Rohdaten fürs Data-Mining behandelt werden. Hierfür sollten Erkenntnisse zu den Szenarien: fehlende Werte, die Behandlung widersprüchlicher Daten und sowohl das Diskretisieren kontinuierlicher Daten als auch die Zerlegung diskreter Werte in binäre Attribute gesammelt werden.

Durch die Erklärungen der Szenarien, deren negative Auswirkungen auf den Data-Mining Prozess, bzw. den Vorteilen, welche die Eliminierung dieser Probleme bieten können, konnte das theoretische Verständnis und die Wichtigkeit der Datenvorbereitung rübergebracht werden. Die dazu beigefügten Methoden samt praktischen Beispielen bieten einen kompakten Überblick der bekanntesten Anwendungsfälle und können somit den Einstieg in die Datenvorbereitung fürs Data-Mining ermöglichen.

Literaturverzeichnis

Abdel-Karim, Benjamin M. (2022): Data Science: Best Practices mit Python, Springer Vieweg

Alexandropoulos, Stamatios-Aggelos N./Sotiris B. Kotsiantis/Michael N. Vrahatis (2019): Data preprocessing in predictive data mining, in: Knowledge Engineering Review, Bd. 34, [online] doi:10.1017/s026988891800036x.

Baskar, S. S./Dr. Arockiam, L./Charles, S. (2013): A systematic Approach on Data Preprocessing in Data Mining, in: COMPUSOFT, An International journal of advanced computer technology,f Volume 2 (Issue 11), S. 335-339

Batini, Carlo/Cappiello, Cinzia/Francalanci, Chiara/Maurino, Andrea (2009): Methodologies for Data Quality Assessment and Improvement, in: ACM Computing Surveys, Volume 41 (Nr. 3) Artikel 16

Bilal, Mehwish/Ali, Ghulam/Iqbal, Muhammad Waseem/Anwar, Muhammad/Malik, Muhammad Sheraz Arshad/Kadir, Rabiah Abdul (2022): Auto-Prep: Efficient and Automated Data Preprocessing Pipeline, in: IEEE Access, Vol. 10, S. 107764-107784, 2022, [online] https://ieeexplore.ieee.org/document/9856663

Cai, Li/Zhu, Yangyong (2015): The Challenges of Data Quality and Data Quality Assessment in the Big Data Era, in: Data Science Journal, Volume 14, Artikel 2, S. 1-10, [online] http://dx.doi.org/10.5334/dsj-2015-002

Chancen, Li (2019): Preprocessing Methods and Pipelines of Data Minnig: An Overview [online] https://arxiv.org/abs/1906.08510

Cleve, Jürgen/Lämmel, Uwe (2016): Data Mining, 2. Auflage, De Gruyter

Donders, A. Rogier T./Geert J.M.G. Van der Heijden/Theo Stijnen/Karel G.M. Moons (2006): Review: A gentle introduction to imputation of missing values, in: Journal Of Clinical Epidemiology, Bd. 59, Nr. 10, S. 1087–1091, [online] doi:10.1016/j.jclinepi.2006.01.014.

Enders, Craig K. (2010): Applied missing data analysis, Guilford Press.

Garciá Salvador/Julián Luengo/Francisco Herrera (2014): Data Preprocessing in Data Mining, Springer

Han, Jiawei/Micheline Kamber/Jian Pei (2011): Data Mining: Concepts and Techniques, Elsevier.

Hemada B./K.S.Vijaya Lakshmi (2013): A Study On Discretization Techniques, in International Journal of Engineering Research & Technology (IJERT), Volume 02, Issue 08 [online] doi:10.17577/IJERTV2IS80599

Kotsiantis, S. and Kanellopoulos, D. (2006): Discretization Techniques: A Recent Survey, GESTS International Transactions on Computer Science and Engineering, 32, 47-58.

Liu, Chang/Yang, Liu/Qu, Jingyi (2021): A structured data preprocessing method based on hybrid encoding, in: Journal of Physics: Conference Series, 1738(2021) 012060, [online] doi:10.1088/1742-6596/1738/1/012060

panData (2024): Mastering Fundamental Binarization in SQL, [online] https://levelup.gitconnected.com/mastering-fundamental-binarization-in-sql-fd887a9e0eda (zuletzt aufgerufen am 24.05.2024)

Papp, Stefan/Weidinger, Wolfgang/Munro, Katherine/Ortner, Bernhard/Cadonna, Annalisa/Langs, Georg/Licandro, Roxane/Meir-Huber, Mario/Nikolic, Danko/Toth, Zoltan/Vesela, Barbora/Wazir, Rania/Zauner, Günther (2022): Handbuch Data Science und KI: Mit Machine Learning und Datenanalyse – Wert aus Daten generieren, 2. Auflage, Hanser

Putri, Pramaishella Ardiani Regita/Sri Suryani Prasetiyowati/Yuliant Sibaroni (2023): The Performance of the Equal-Width and Equal-Frequency Discretization Methods on Data Features in Classification Process, in: Sinkron, Bd. 8, Nr. 4, S. 2082–2098, [online] doi:10.33395/sinkron.v8i4.12730.

Pyle, Dorian (1999): Data preparation for data mining, [online] https://dl.acm.org/citation.cfm?id=299577 (zuletzt aufgerufen am 24.05.2024)

Solanki, Jatin (2024): What is Data Consistency? Definition, examples and best practice, [online] https://www.decube.io/post/what-is-data-consistency-definition-examples-and-best-practice (zuletzt aufgerufen am 23.05.2024)

Anhang
Anwendung „Fehlende Werte“

In [13]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as mno
from sklearn import linear_model
%matplotlib inline
```

Aufrufen des Datensets

In [14]:
```
df = pd.read_csv('C:/Users/maxtr/Desktop/BI_Mitarbeiterdaten_missing_values.csv', sep =',', index_col=0)
df[["Vorname", "Nachname","Geschlecht","Alter","Gehalt"]].head()
```

Out[14]:

MitarbeiterID	Vorname	Nachname	Geschlecht	Alter	Gehalt
101	Anna	Müller	weiblich	53	45000.0
102	Peter	Schmidt	männlich	33	NaN
103	Julia	Becker	weiblich	35	47000.0
104	NaN	Weber	männlich	49	0.0
105	Sandra	Fischer	weiblich	31	52000.0

Ausgeben der bestehenden NaN Werte im Datensatz

In [15]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 50 entries, 101 to 150
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Vorname       42 non-null     object
 1   Nachname      50 non-null     object
 2   Geschlecht    50 non-null     object
 3   Geburtsdatum  50 non-null     object
 4   Alter         50 non-null     int64
 5   Land          50 non-null     object
 6   PLZ           50 non-null     int64
 7   Wohnort       50 non-null     object
 8   Straße        50 non-null     object
 9   Hausnr.       50 non-null     int64
 10  Abteilung     50 non-null     object
 11  Gehalt        45 non-null     float64
 12  Bonus         50 non-null     object
dtypes: float64(1), int64(3), object(9)
memory usage: 5.5+ KB
```

Ausgeben der numerischen Datensatzstatistiken zur Überprüfung von fehlerhaften Einträgen

In [16]:
```
df.describe()
```

Out[16]:

	Alter	PLZ	Hausnr.	Gehalt
count	50.000000	50.000000	50.000000	45.000000
mean	38.460000	34501.420000	26.800000	43964.444444
std	6.794986	34453.690286	35.491304	19553.413508
min	27.000000	1010.000000	1.000000	0.000000
25%	34.000000	5020.000000	8.000000	46100.000000
50%	38.500000	10147.000000	14.500000	52000.000000
75%	42.750000	66200.500000	34.250000	53100.000000
max	60.000000	99084.000000	222.000000	64100.000000

Gehälter, welche fälschlicherweise mit 0 eingetragen wurden, als NAN markieren Fehlende Vornamen sind für uns irrelevant, weshalb diesen der Wert "Unbekannt" zugewiesen wird

In [17]:
```
df.loc[df["Gehalt"] == 0.0, "Gehalt"] = np.NAN
df["Vorname"] = df["Vorname"].fillna("Unbekannt")
df.isnull().sum()[1:13]
```

Out[17]:
```
Nachname         0
Geschlecht       0
Geburtsdatum     0
Alter            0
Land             0
PLZ              0
Wohnort          0
Straße           0
Hausnr.          0
Abteilung        0
Gehalt          12
Bonus            0
dtype: int64
```

In [17]:
```
df.loc[df["Gehalt"] == 0.0, "Gehalt"] = np.NAN
df["Vorname"] = df["Vorname"].fillna("Unbekannt")
df.isnull().sum()[1:13]
```

Out[17]:
```
Nachname         0
Geschlecht       0
Geburtsdatum     0
Alter            0
Land             0
PLZ              0
Wohnort          0
Straße           0
Hausnr.          0
Abteilung        0
Gehalt          12
Bonus            0
dtype: int64
```

Ausgeben der bestehenden NaN Werte im Datensatz

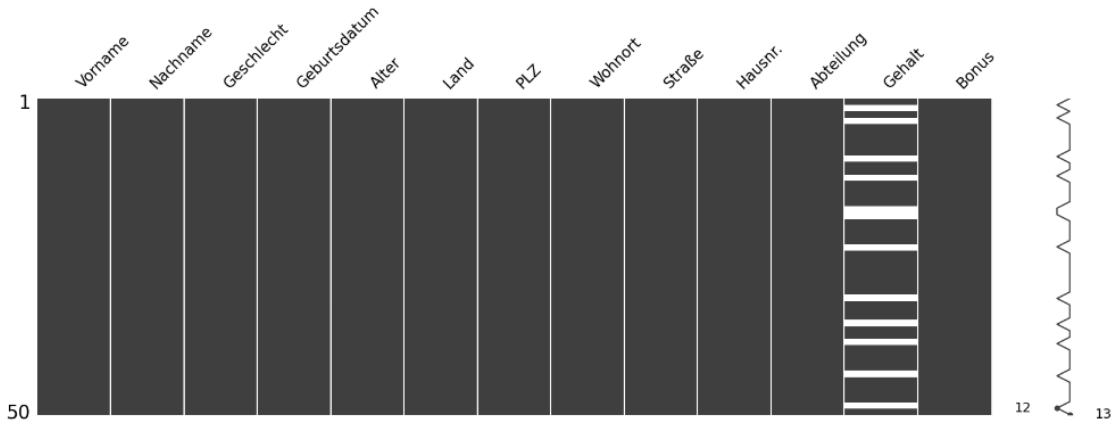
In [18]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 50 entries, 101 to 150
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Vorname       50 non-null     object
 1   Nachname      50 non-null     object
 2   Geschlecht    50 non-null     object
 3   Geburtsdatum  50 non-null     object
 4   Alter         50 non-null     int64
 5   Land          50 non-null     object
 6   PLZ           50 non-null     int64
 7   Wohnort       50 non-null     object
 8   Straße        50 non-null     object
 9   Hausnr.       50 non-null     int64
 10  Abteilung     50 non-null     object
 11  Gehalt        38 non-null     float64
 12  Bonus         50 non-null     object
dtypes: float64(1), int64(3), object(9)
memory usage: 5.5+ KB
```

Visualisierung der fehlenden Gehälter

In [19]:
```
mno.matrix(df, figsize = (20, 6))
```

Out[19]:
```
<Axes: >
```



Erstellen eines Subsets, in welchem keine Werte fehlen

In [20]:
```
# Entfernen der Zeilen, mit fehldenen Werten in der Spalte Gehalt
df_alter_gehalt = df.dropna(axis = 0, subset = ["Gehalt"])
# Auswahl der benötigten Spalten des Dataframe
df_alter_gehalt = df_alter_gehalt.loc[:, ["Gehalt", "Alter"]]
# Erstellen eines Dataframes für fehlende Gehälter
missing_gehalt = df["Gehalt"].isnull()
# Extrahieren der Altersdaten für fehlende Gehälter
alter_misgehalt = pd.DataFrame(df["Alter"][missing_gehalt])
```

Durchführung der Regression

[35]:
```
# Aufteilen der Daten in Feature- (X) und Zielvariablen (Y)
X = df_alter_gehalt[["Alter"]]
Y = df_alter_gehalt["Gehalt"]


from sklearn.linear_model import LinearRegression

# Erstellung des linearen Regressionmosells und Anpassung an die Trainingsdaten
lm = LinearRegression().fit(X, Y)

# fehlende Gehälter werden basierend auf dem Alter geschätzt
gehalt_pred = lm.predict(alter_misgehalt)

# Standardabeichung wird berechnet
std_error = np.std(Y - lm.predict(X))
random_error = np.random.normal(0, std_error, gehalt_pred.shape[0])

# Standardabweichung wird auf Gehaltsschätzungen angerechnet und Ergebnis wird gerundet
gehalt_pred_with_error = gehalt_pred + random_error
gehalt_pred_with_error_rounded = np.round(gehalt_pred_with_error, 2)

for alter, gehalt in zip(alter_misgehalt['Alter'], gehalt_pred_with_error_rounded):
    print(f"Alter: {alter}, Geschätztes Gehalt: {gehalt}")
```

```
Alter: 33, Geschätztes Gehalt: 53498.95
Alter: 49, Geschätztes Gehalt: 50245.68
Alter: 29, Geschätztes Gehalt: 48877.38
Alter: 43, Geschätztes Gehalt: 46220.74
Alter: 48, Geschätztes Gehalt: 54622.12
Alter: 41, Geschätztes Gehalt: 58732.17
Alter: 48, Geschätztes Gehalt: 57735.96
Alter: 38, Geschätztes Gehalt: 42870.7
Alter: 39, Geschätztes Gehalt: 48851.84
Alter: 41, Geschätztes Gehalt: 56193.76
Alter: 43, Geschätztes Gehalt: 44183.68
Alter: 44, Geschätztes Gehalt: 50253.18
```

Ersetzen der fehlenden Werten mit den erhobenen Werten

In [22]:
```
df.loc[df["Gehalt"].isnull(), "Gehalt"] = gehalt_pred_with_error_rounded

```

Speicherung der neuen Werte in die CSV-Datei

In [23]:
```
df.to_csv("C:/Users/maxtr/Desktop/BI_Mitarbeiterdaten_filled_values.csv")
df[["Vorname", "Nachname","Geschlecht","Alter","Gehalt"]].head()
```

Out[23]:

MitarbeiterID	Vorname	Nachname	Geschlecht	Alter	Gehalt
101	Anna	Müller	weiblich	53	45000.00
102	Peter	Schmidt	männlich	33	46451.52
103	Julia	Becker	weiblich	35	47000.00
104	Unbekannt	Weber	männlich	49	53524.54
105	Sandra	Fischer	weiblich	31	52000.00

Anwendung „Behandlung widersprüchlicher Daten“

Wir importieren die Daten

In [91]:
```
import pandas as pd
import matplotlib.pyplot as plt
from datetime import date
df = pd.read_excel(r"C:/Users/maxtr/Desktop/BI_Mitarbeiterdaten_widerspruch.xlsx")
#df
df.head()
```

Out[91]:

	MitarbeiterID	Vorname	Nachname	Geschlecht	Geburtsdatum	Alter	Land	PLZ	Wohnort	Straße	Hausnr.	Abteilung	Gehalt	Bonus
0	101	Anna	Müller	weiblich	1985-03-15	39.0	Deutschland	10115	Berlin	Hauptstraße	12	Vertrieb	45000	ja
1	102	Peter	Schmidt	männlich	1990-07-22	33.0	Deutschland	20095	Hamburg	Bahnhofstraße	5	IT	55000	nein
2	103	Julia	Becker	w	1988-11-08	350.0	Österreich	1010	Wien	Kärntner Straße	20	Marketing	47000	ja
3	104	Michael	Weber	männlich	1975-02-25	49.0	Schweiz	8001	Zürich	Bahnhofstrasse	45	Finanzen	60000	ja
4	105	Sandra	Fischer	w	1992-05-30	31.0	Deutschland	80331	München	Sonnenstraße	8	Personalwesen	52000	ja

```
Übersicht über die Daten verschaffen & ggf. unnötige Spalten entfernen
```

In [82]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   MitarbeiterID  20 non-null     int64
 1   Vorname        20 non-null     object
 2   Nachname       20 non-null     object
 3   Geschlecht     20 non-null     object
 4   Geburtsdatum   20 non-null     datetime64[ns]
 5   Alter          18 non-null     float64
 6   Land           20 non-null     object
 7   PLZ            20 non-null     int64
 8   Wohnort        20 non-null     object
 9   Straße         20 non-null     object
 10  Hausnr.        20 non-null     int64
 11  Abteilung      20 non-null     object
 12  Gehalt         20 non-null     int64
 13  Bonus          20 non-null     object
dtypes: datetime64[ns](1), float64(1), int64(4), object(8)
memory usage: 2.3+ KB
```

```
Entsprechende Spalten mit Fehlern, Widersprüchen, Ungenauigkeiten etc. identifizieren

-> Geschlecht, Geburtsdatum, Alter
```

```
Spalte "Geschlecht" bereinigen
```

In [83]:
```
df["Geschlecht"].unique()
#plt.hist(df["Geschlecht"])
```

Out[83]:
```
array(['weiblich', 'männlich', 'w', 'm'], dtype=object)
```

In [92]:
```
df["Geschlecht"] = df["Geschlecht"].replace({"männlich": "m", "weiblich": "w"})
#df
df.head()
```

Out[92]:

	MitarbeiterID	Vorname	Nachname	Geschlecht	Geburtsdatum	Alter	Land	PLZ	Wohnort	Straße	Hausnr.	Abteilung	Gehalt	Bonus
0	101	Anna	Müller	w	1985-03-15	39.0	Deutschland	10115	Berlin	Hauptstraße	12	Vertrieb	45000	ja
1	102	Peter	Schmidt	m	1990-07-22	33.0	Deutschland	20095	Hamburg	Bahnhofstraße	5	IT	55000	nein
2	103	Julia	Becker	w	1988-11-08	350.0	Österreich	1010	Wien	Kärntner Straße	20	Marketing	47000	ja
3	104	Michael	Weber	m	1975-02-25	49.0	Schweiz	8001	Zürich	Bahnhofstrasse	45	Finanzen	60000	ja
4	105	Sandra	Fischer	w	1992-05-30	31.0	Deutschland	80331	München	Sonnenstraße	8	Personalwesen	52000	ja

In [85]:
```
df["Geschlecht"].unique()
```

Out[85]:
```
array(['w', 'm'], dtype=object)
```

Spalte "Alter" bereinigen

In [93]:
```
df["Geburtsdatum"].info()
#df["Geburtsdatum"]
df["Geburtsdatum"].head()

# -> Pandas Serie
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 20 entries, 0 to 19
Series name: Geburtsdatum
Non-Null Count  Dtype
--------------  -----
20 non-null     datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 292.0 bytes
```

Out[93]:
```
0   1985-03-15
1   1990-07-22
2   1988-11-08
3   1975-02-25
4   1992-05-30
Name: Geburtsdatum, dtype: datetime64[ns]
```

Neue Spalte für das aktuelle Alter auf Basis des angegebenen Geburtsdatums

In [94]:
```
geburtstage = []   # Leer Liste für datetime Objekte

for geburtstag in df["Geburtsdatum"]:
    geburtstage.append(pd.to_datetime(geburtstag)) # jeder Eintrag wird zum datetime Objekt in der Liste

def berechne_alter(geburtstag): # hier wird einzelnes Objekt benötigt
    heute = date.today()
    return heute.year - geburtstag.year - ((heute.month, heute.day) < (geburtstag.month, geburtstag.day))

df["Alter_neu"] = df["Geburtsdatum"].apply(berechne_alter)
#df
df.head()
```

Out[94]:

beiterID	Vorname	Nachname	Geschlecht	Geburtsdatum	Alter	Land	PLZ	Wohnort	Straße	Hausnr.	Abteilung	Gehalt	Bonus	Alter_neu
101	Anna	Müller	w	1985-03-15	39.0	Deutschland	10115	Berlin	Hauptstraße	12	Vertrieb	45000	ja	39
102	Peter	Schmidt	m	1990-07-22	33.0	Deutschland	20095	Hamburg	Bahnhofstraße	5	IT	55000	nein	33
103	Julia	Becker	w	1988-11-08	350.0	Österreich	1010	Wien	Kärntner Straße	20	Marketing	47000	ja	35
104	Michael	Weber	m	1975-02-25	49.0	Schweiz	8001	Zürich	Bahnhofstrasse	45	Finanzen	60000	ja	49
105	Sandra	Fischer	w	1992-05-30	31.0	Deutschland	80331	München	Sonnenstraße	8	Personalwesen	52000	ja	32

In [95]:
```
check = ["Geburtsdatum", "Alter", "Alter_neu"]
df_check = df[check]
#df_check
df_check.head()
```

Out[95]:

	Geburtsdatum	Alter	Alter_neu
0	1985-03-15	39.0	39
1	1990-07-22	33.0	33
2	1988-11-08	350.0	35
3	1975-02-25	49.0	49
4	1992-05-30	31.0	32

X

In [89]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   MitarbeiterID  20 non-null     int64
 1   Vorname        20 non-null     object
 2   Nachname       20 non-null     object
 3   Geschlecht     20 non-null     object
 4   Geburtsdatum   20 non-null     datetime64[ns]
 5   Alter          18 non-null     float64
 6   Land           20 non-null     object
 7   PLZ            20 non-null     int64
 8   Wohnort        20 non-null     object
 9   Straße         20 non-null     object
 10  Hausnr.        20 non-null     int64
 11  Abteilung      20 non-null     object
 12  Gehalt         20 non-null     int64
 13  Bonus          20 non-null     object
 14  Alter_neu      20 non-null     int64
dtypes: datetime64[ns](1), float64(1), int64(5), object(8)
memory usage: 2.5+ KB
```

Spalte "Geburtsdatum" bereinigen

In [96]:
```
# pd.to_datetime() -> Datentyp festlegen / ändern
# .dt.strftime() -> Formatierung

df["Geburtsdatum"] = pd.to_datetime(df["Geburtsdatum"], format='%Y/%m/%d')
df["Geburtsdatum"] = df["Geburtsdatum"].dt.strftime('%d.%m.%Y')
#df
df.head()
```

Out[96]:

beiterID	Vorname	Nachname	Geschlecht	Geburtsdatum	Alter	Land	PLZ	Wohnort	Straße	Hausnr.	Abteilung	Gehalt	Bonus	Alter_neu
101	Anna	Müller	w	15.03.1985	39.0	Deutschland	10115	Berlin	Hauptstraße	12	Vertrieb	45000	ja	39
102	Peter	Schmidt	m	22.07.1990	33.0	Deutschland	20095	Hamburg	Bahnhofstraße	5	IT	55000	nein	33
103	Julia	Becker	w	08.11.1988	350.0	Österreich	1010	Wien	Kärntner Straße	20	Marketing	47000	ja	35
104	Michael	Weber	m	25.02.1975	49.0	Schweiz	8001	Zürich	Bahnhofstrasse	45	Finanzen	60000	ja	49
105	Sandra	Fischer	w	30.05.1992	31.0	Deutschland	80331	München	Sonnenstraße	8	Personalwesen	52000	ja	32

ggf. "Alter" löschen (und andere unnötige Spalten), "Alter_neu" in "Alter" umbenennen, ggf. Spalten anordnen / sortieren

Anwendung „Diskreditierung kontinuierlicher Daten“

In [1]:
```
import seaborn
import pandas as pd
```

Einlesen der CSV-Datei

In [2]:
```
df = pd.read_csv('C:/Users/maxtr/Desktop/BI_Mitarbeiterdaten_filled_values.csv', sep =',', index_col=0)
df[["Vorname", "Nachname","Geschlecht","Alter","Gehalt"]].head()
```

Out[2]:

MitarbeiterID	Vorname	Nachname	Geschlecht	Alter	Gehalt
101	Anna	Müller	weiblich	53	45000.00
102	Peter	Schmidt	männlich	33	46451.52
103	Julia	Becker	weiblich	35	47000.00
104	Unbekannt	Weber	männlich	49	53524.54
105	Sandra	Fischer	weiblich	31	52000.00

Übergeben der Altersspalte in die equal-frequenzy Funktion

In [3]:
```
df["Alter"] = pd.qcut(df["Alter"], q=5, labels=["sehr jung","jung","mittelalt","alt","sehr alt"])
df["Alter"].value_counts().sort_index()
```

Out[3]:
```
Alter
sehr jung    10
jung         10
mittelalt    10
alt          10
sehr alt     10
Name: count, dtype: int64
```

Speicherung der neuen Werte in die CSV-Datei

In [4]:
```
df.to_csv("C:/Users/maxtr/Desktop/BI_Mitarbeiterdaten_binned.csv")
df[["Vorname", "Nachname","Geschlecht","Alter","Gehalt"]].head()
```

Out[4]:

MitarbeiterID	Vorname	Nachname	Geschlecht	Alter	Gehalt
101	Anna	Müller	weiblich	sehr alt	45000.00
102	Peter	Schmidt	männlich	jung	46451.52
103	Julia	Becker	weiblich	jung	47000.00
104	Unbekannt	Weber	männlich	sehr alt	53524.54
105	Sandra	Fischer	weiblich	sehr jung	52000.00

Anwendung „Zerlegung diskreter Werte in binäre Attribute“

Importieren / Einlesen der Daten

In [52]:
```
import pandas as pd
df = pd.read_excel(r"C:/Users/maxtr/Desktop/BI_Mitarbeiterdaten_diskret_binär.xlsx")
#df
df.head()
```

Out[52]:

	MitarbeiterID	Vorname	Nachname	Geschlecht	Geburtsdatum	Alter	Land	PLZ	Wohnort	Straße	Hausnr.	Abteilung	Gehalt	Bonus
0	101	Anna	Müller	weiblich	1985-03-15	39	Deutschland	10115	Berlin	Hauptstraße	12	Vertrieb	45000	ja
1	102	Peter	Schmidt	männlich	1990-07-22	33	Deutschland	20095	Hamburg	Bahnhofstraße	5	IT	55000	nein
2	103	Julia	Becker	weiblich	1988-11-08	35	Österreich	1010	Wien	Kärntner Straße	20	Marketing	47000	ja
3	104	Michael	Weber	männlich	1975-02-25	49	Schweiz	8001	Zürich	Bahnhofstrasse	45	Finanzen	60000	ja
4	105	Sandra	Fischer	weiblich	1992-05-30	31	Deutschland	80331	München	Sonnenstraße	8	Personalwesen	52000	ja

Übersicht über die Daten verschaffen & ggf. unnötige Spalten entfernen

In [42]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   MitarbeiterID  20 non-null     int64
 1   Vorname        20 non-null     object
 2   Nachname       20 non-null     object
 3   Geschlecht     20 non-null     object
 4   Geburtsdatum   20 non-null     datetime64[ns]
 5   Alter          20 non-null     int64
 6   Land           20 non-null     object
 7   PLZ            20 non-null     int64
 8   Wohnort        20 non-null     object
 9   Straße         20 non-null     object
 10  Hausnr.        20 non-null     int64
 11  Abteilung      20 non-null     object
 12  Gehalt         20 non-null     int64
 13  Bonus          20 non-null     object
dtypes: datetime64[ns](1), int64(5), object(8)
memory usage: 2.3+ KB
```

Entsprechende Spalten für das One-Hot Encoding identifizieren (Zerlegung diskreter Werte in binäre Attribute) --> Land & Abteilung (ggf. Alter nach der Diskretisierung) unters. Einträge der Spalten untersuchen

In [43]:
```
df["Land"].unique()
```

Out[43]:
```
array(['Deutschland', 'Österreich', 'Schweiz'], dtype=object)
```

In [44]:
```
df["Abteilung"].unique()
```

Out[44]:
```
array(['Vertrieb', 'IT', 'Marketing', 'Finanzen', 'Personalwesen'],
      dtype=object)
```

Anwendung One-Hot Encoding auf die Spalte "Land" und deren Ausprägungen

In [53]:
```
encoded_df = pd.get_dummies(df, columns=["Land"])
#encoded_df
encoded_df.head()
```

Out[53]:

e	Geschlecht	Geburtsdatum	Alter	PLZ	Wohnort	Straße	Hausnr.	Abteilung	Gehalt	Bonus	Land_Deutschland	Land_Schweiz	Land_Österreich
er	weiblich	1985-03-15	39	10115	Berlin	Hauptstraße	12	Vertrieb	45000	ja	True	False	False
lt	männlich	1990-07-22	33	20095	Hamburg	Bahnhofstraße	5	IT	55000	nein	True	False	False
er	weiblich	1988-11-08	35	1010	Wien	Kärntner Straße	20	Marketing	47000	ja	False	False	True
er	männlich	1975-02-25	49	8001	Zürich	Bahnhofstrasse	45	Finanzen	60000	ja	False	True	False
er	weiblich	1992-05-30	31	80331	München	Sonnenstraße	8	Personalwesen	52000	ja	True	False	False

Ursprüngliche Spalte "Land" verschwindet Dummys dafür werden erstellt

ggf. True == 1; False == 0; -> für die Algorithmen irrelevant

ggf. Datentabelle speichern

In [40]:
```
#encoded_df["Land_Deutschland"] = encoded_df["Land_Deutschland"].replace({True: 1, False: 0})
#encoded_df.head()
```

hname	Geschlecht	Geburtsdatum	Alter	PLZ	Wohnort	Straße	Hausnr.	Abteilung	Gehalt	Bonus	Land_Deutschland	Land_Schweiz	Land_Österreich
Müller	weiblich	1985-03-15	39	10115	Berlin	Hauptstraße	12	Vertrieb	45000	ja	1	False	False
chmidt	männlich	1990-07-22	33	20095	Hamburg	Bahnhofstraße	5	IT	55000	nein	1	False	False
Becker	weiblich	1988-11-08	35	1010	Wien	Kärntner Straße	20	Marketing	47000	ja	0	False	True
Weber	männlich	1975-02-25	49	8001	Zürich	Bahnhofstrasse	45	Finanzen	60000	ja	0	True	False
Fischer	weiblich	1992-05-30	31	80331	München	Sonnenstraße	8	Personalwesen	52000	ja	1	False	False
Meyer	männlich	1981-09-14	42	50667	Köln	Hohe Straße	14	Vertrieb	48000	nein	1	False	False

In [46]:
```
encoded_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   MitarbeiterID     20 non-null     int64
 1   Vorname           20 non-null     object
 2   Nachname          20 non-null     object
 3   Geschlecht        20 non-null     object
 4   Geburtsdatum      20 non-null     datetime64[ns]
 5   Alter             20 non-null     int64
 6   PLZ               20 non-null     int64
 7   Wohnort           20 non-null     object
 8   Straße            20 non-null     object
 9   Hausnr.           20 non-null     int64
 10  Abteilung         20 non-null     object
 11  Gehalt            20 non-null     int64
 12  Bonus             20 non-null     object
 13  Land_Deutschland  20 non-null     bool
 14  Land_Schweiz      20 non-null     bool
 15  Land_Österreich   20 non-null     bool
dtypes: bool(3), datetime64[ns](1), int64(5), object(7)
memory usage: 2.2+ KB
```

Methode Binarization (2 Ausprägungen)

In [47]:
```
encoded_df.head()
```

Out[47]:

e	Geschlecht	Geburtsdatum	Alter	PLZ	Wohnort	Straße	Hausnr.	Abteilung	Gehalt	Bonus	Land_Deutschland	Land_Schweiz	Land_Österreich
er	weiblich	1985-03-15	39	10115	Berlin	Hauptstraße	12	Vertrieb	45000	ja	True	False	False
dt	männlich	1990-07-22	33	20095	Hamburg	Bahnhofstraße	5	IT	55000	nein	True	False	False
er	weiblich	1988-11-08	35	1010	Wien	Kärntner Straße	20	Marketing	47000	ja	False	False	True
er	männlich	1975-02-25	49	8001	Zürich	Bahnhofstrasse	45	Finanzen	60000	ja	False	True	False
er	weiblich	1992-05-30	31	80331	München	Sonnenstraße	8	Personalwesen	52000	ja	True	False	False

Entsprechende Spalten identifizieren --> Geschlecht & Bonus unters. Einträge der Spalten untersuchen

In [48]:
```
df["Geschlecht"].unique()
```

Out[48]:
```
array(['weiblich', 'männlich'], dtype=object)
```

In [49]:
```
df["Bonus"].unique()
```

Out[49]:
```
array(['ja', 'nein'], dtype=object)
```

Anwendung Binarization

In [54]:
```
encoded_df["Geschlecht_binär"] = encoded_df["Geschlecht"].replace({"männlich": 1, "weiblich": 0})
encoded_df["Bonus_binär"] = encoded_df["Bonus"].replace({"ja": 1, "nein": 0})
#encoded_df
encoded_df.head()
```

Out[54]:

Iter	PLZ	Wohnort	Straße	Hausnr.	Abteilung	Gehalt	Bonus	Land_Deutschland	Land_Schweiz	Land_Österreich	Geschlecht_binär	Bonus_binär
39	10115	Berlin	Hauptstraße	12	Vertrieb	45000	ja	True	False	False	0	1
33	20095	Hamburg	Bahnhofstraße	5	IT	55000	nein	True	False	False	1	0
35	1010	Wien	Kärntner Straße	20	Marketing	47000	ja	False	False	True	0	1
49	8001	Zürich	Bahnhofstrasse	45	Finanzen	60000	ja	False	True	False	1	1
31	80331	München	Sonnenstraße	8	Personalwesen	52000	ja	True	False	False	0	1

Ursprüngliche Spalten (Geschlecht und Bonus) entfernen

In [55]:

```
encoded_df.drop("Geschlecht", axis=1, inplace=True)   # axis=1 -> Spaltenachse,
encoded_df.drop("Bonus", axis=1, inplace=True)        # inplace=True -> in df integrieren keine Kopie von df erstellen
#encoded_df
encoded_df.head()
```

Out[55]:

MitarbeiterID	Vorname	Nachname	Geburtsdatum	Alter	PLZ	Wohnort	Straße	Hausnr.	Abteilung	Gehalt	Land_Deutschland	Land_Schweiz	Lan
101	Anna	Müller	1985-03-15	39	10115	Berlin	Hauptstraße	12	Vertrieb	45000	True	False	
102	Peter	Schmidt	1990-07-22	33	20095	Hamburg	Bahnhofstraße	5	IT	55000	True	False	
103	Julia	Becker	1988-11-08	35	1010	Wien	Kärntner Straße	20	Marketing	47000	False	False	
104	Michael	Weber	1975-02-25	49	8001	Zürich	Bahnhofstrasse	45	Finanzen	60000	False	True	
105	Sandra	Fischer	1992-05-30	31	80331	München	Sonnenstraße	8	Personalwesen	52000	True	False	