# Kapitel 4
# Explorative Datenanalyse und Datenvorbereitung

*Wenn Sie in Ihrem Data-Science-Projekt mit Daten arbeiten, müssen Sie diese zuerst auf ihre Struktur und Inhalte hin analysieren und eventuell für die nächsten Data-Science-Schritte aufbereiten. Wie das gelingt, erfahren Sie in diesem Kapitel.*

Folgen Sie dem in Kapitel 1, »Einführung«, vorgestellten Prozess CRISP-DM, beginnen Sie jedes neue Data-Science-Projekt damit, zuerst die Anforderungen des Geschäftsprozesses zu analysieren und zu hinterfragen. Sobald Sie ein erstes gutes Verständnis der Anforderungen haben, können Sie sich auf die Suche nach eventuell relevanten Daten begeben. Die Analyse der Geschäftsanforderungen ist dennoch ein dynamischer Prozess und sollte niemals als abgeschlossen betrachtet werden. Im Projekt können sich Erkenntnisse ergeben, die die ursprünglichen Anforderungen beeinflussen.

Eventuell befinden sich die relevanten Daten bereits zentral in einer Datenbank. Oder aber die Daten sind über mehrere Systeme verteilt. Sobald Sie Zugriff auf die Daten haben, müssen Sie die Struktur und die Inhalte dieser Daten verstehen. Dieses Buch fokussiert sich größtenteils auf strukturierte Daten, die in SAP-HANA-Tabellen gespeichert sind. Dabei stellen Sie verschiedene Fragen: Wie heißen diese Tabellen, aus welchen Spalten bestehen diese und welche Daten sind letztlich in diesen Tabellen gespeichert? Sind die Daten relevant? Wie ist die Qualität dieser Daten? Sind für die Data-Science-Methoden ausreichend Daten vorhanden? Lassen sich die Daten miteinander verknüpfen? Müssen diese Daten für die Data-Science-Methoden noch weiter aufbereitet werden? In diesem Kapitel stellen wir verschiedene Methoden der *Datenanalyse und -vorbereitung* vor, unter anderem:

- Analyse der Datenstrukturen, wie Datentypen der Spalten
- Analyse der Inhalte einzelner numerischer und kategorischer Variablen
- Beziehung der Inhalte mehrerer Variablen zueinander
- Datenaufbereitung, wie Aggregation und Verknüpfung mehrerer Tabellen/Views

In den Materialien zu diesem Buch unter *www.sap-press.de/5539* finden Sie den ausführbaren Codes dieses Kapitels im Ordner **Kapitel 4**. In Abbildung 4.1 sehen, wie der von uns in diesem Kapitel genutzte Code in einem Notebook aussieht.

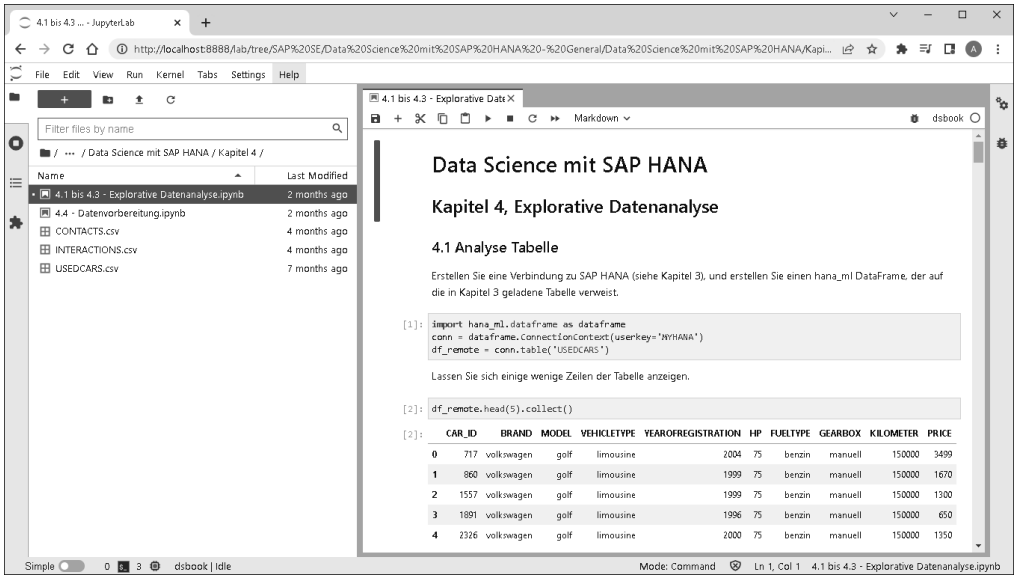


**Abbildung 4.1** Notebook mit ausführbarem Code

## 4.1 Analyse einer Tabelle

Der erste technische Schritt in einem Data-Science-Projekt ist das in Abschnitt 1.1.4, »Data Science«, erwähnte *Data Understanding*. Für diese explorative Datenanalyse nutzen wir die in Kapitel 3, »Erste Schritte«, erstellte Tabelle USEDCARS, die Details über zu verkaufende Gebrauchtfahrzeuge beinhaltet. Diese Daten werden in den folgenden Kapiteln häufig genutzt, z. B. um den Fahrzeugpreis zu schätzen, fehlende Informationen aufzufüllen oder um ein Clustering durchzuführen.

Wenn Sie diese Schritte selbst am System durchführen möchten, muss diese Tabelle vorhanden sein. Erstellen Sie einen `hana_ml`-DataFrame auf diese Tabelle:

```
import hana_ml.dataframe as dataframe
conn = dataframe.ConnectionContext(userkey='MYHANA')
df_remote = conn.table('USEDCARS')
```

Wie gehabt, bleiben die Daten hierbei in SAP HANA. Als schnellen Test, ob der `hana_ml`-DataFrame korrekt erstellt wurde, können Sie einige wenige Zeilen anzeigen las-

sen. Die `head()`-Methode schränkt die Daten auf fünf Zeilen ein. Die folgende `collect()`-Methode lädt die Daten herunter als Pandas-DataFrame in die Python-Umgebung. Wichtig ist die Reihenfolge der beiden Methoden. Die `head()`-Methode sollte zuerst erfolgen, damit Daten bereits in SAP HANA gefiltert werden und nur der bereits reduzierte Datensatz heruntergeladen wird.

```
df_remote.head(5).collect()
```

Sie sehen fünf zufällig ausgewählte Zeilen des Datensatzes (siehe Abbildung 4.2).

| CAR_ID | BRAND | MODEL | VEHICLETYPE | YEAROFREGISTRATION | HP | FUELTYPE | GEARBOX | KILOMETER | PRICE |
|---|---|---|---|---|---|---|---|---|---|
| 717 | volkswagen | golf | limousine | 2004 | 75 | benzin | manuell | 150000 | 3499 |
| 860 | volkswagen | golf | limousine | 1999 | 75 | benzin | manuell | 150000 | 1670 |
| 1557 | volkswagen | golf | limousine | 1999 | 75 | benzin | manuell | 150000 | 1300 |
| 1891 | volkswagen | golf | limousine | 1996 | 75 | benzin | manuell | 150000 | 650 |
| 2326 | volkswagen | golf | limousine | 2000 | 75 | benzin | manuell | 150000 | 1350 |

**Abbildung 4.2** Tabelle USEDCARS

**Sortierung der Daten**

Die Sortierung der Daten ist zufällig, da ein einfaches `SELECT`-Statement ohne weitere Angaben keine Sortierung garantiert. Wenn Sie eine Sortierung wünschen, können Sie diese explizit anfordern. Hier werden die fünf Fahrzeuge mit der kleinsten CAR_ID angezeigt:

```
df_remote.sort('CAR_ID', desc=False).head(5).collect()
```

Nun möchten wir die Struktur der Tabelle analysieren und herausfinden, aus wie vielen Spalten und Zeilen die Tabelle besteht. Dafür geben Sie die folgende Codezeile ein:

```
df_remote.shape
```

Ihnen wird der Wert **[263178, 10]** als Ergebnis angezeigt. Die erste Zahl gibt die Anzahl der Zeilen an. Die zweite Zahl gibt die Anzahl der Spalten an. Mit zehn Spalten ist die Tabellenstruktur überschaubar. Lassen Sie sich alle Spaltennamen und deren Datentypen anzeigen, indem Sie diesen Code eingeben:

```
df_remote.dtypes()
```

Wie in Abbildung 4.3 sichtbar, erhalten Sie diese Informationen als Python-Liste.

```
[('CAR_ID', 'INT', 10, 10, 10, 0),
 ('BRAND', 'NVARCHAR', 5000, 5000, 5000, 0),
 ('MODEL', 'NVARCHAR', 5000, 5000, 5000, 0),
 ('VEHICLETYPE', 'NVARCHAR', 5000, 5000, 5000, 0),
 ('YEAROFREGISTRATION', 'INT', 10, 10, 10, 0),
 ('HP', 'INT', 10, 10, 10, 0),
 ('FUELTYPE', 'NVARCHAR', 5000, 5000, 5000, 0),
 ('GEARBOX', 'NVARCHAR', 5000, 5000, 5000, 0),
 ('KILOMETER', 'INT', 10, 10, 10, 0),
 ('PRICE', 'INT', 10, 10, 10, 0)]
```

**Abbildung 4.3** Spaltennamen und Datentypen

Für Tabellen, die aus deutlich mehr Spalten bestehen, können Sie die Häufigkeit der Datentypen der Spalten mit diesem Code zählen lassen:

```
import pandas as pd
df_dtypes = pd.DataFrame(df_remote.dtypes())[[0, 1]]
df_dtypes.columns = ['COL_NAME', 'COL_TYPE']
df_dtypes.groupby(['COL_TYPE']).agg(['count'])
```

Die Tabelle USEDCARS besteht aus jeweils fünf Spalten der Typen INT und NVARCHAR (siehe Abbildung 4.4).

| | COL_NAME |
|---|---|
| | count |
| COL_TYPE | |
| INT | 5 |
| NVARCHAR | 5 |

**Abbildung 4.4** Spaltentypen der Tabelle USEDCARS

Wenn Sie jetzt herausfinden möchten, wie die Spalten eines ausgewählten Datentyps heißen, können Sie sich alle Spaltennamen des jeweiligen Typs mit folgendem Code anzeigen lassen:

```
str_type = 'INT'
df_dtypes[df_dtypes['COL_TYPE']==str_type][['COL_NAME']]
```

Die Spalten von Typ INT der Tabelle USEDCARS sehen Sie in Abbildung 4.5.

| COL_NAME |
|---|
| CAR_ID |
| YEAROFREGISTRATION |
| HP |
| KILOMETER |
| PRICE |

**Abbildung 4.5** Spalten vom Typ »INT«

Eine weitere Möglichkeit, die Daten in der Tabelle zu analysieren, bietet die Methode `describe()`. Wenn Sie den folgenden Code ausführen, liefert die Methode Ihnen einen schnellen Überblick über die Spalteninhalte:

```
df_remote.describe().collect()
```

Ihnen wird nun angezeigt, wie viele unterschiedliche Werte und Nullwerte jede einzelne Spalte beinhaltet. Zusätzlich werden für numerische Spalten weitere Details angezeigt, wie etwa Durchschnitt, Minimum, Maximum sowie weitere Informationen über die Verteilung der Werte (siehe Abbildung 4.6). Beachten Sie die Spalten **25_percent_cont** und **75_percent_cont**. Diese geben die Grenzwerte an, unter denen 25 %, (1. Quartil) beziehungsweise 75 % der Werte (3. Quartil) der Spalte liegen. In Abschnitt 4.2.1, »Numerische Variablen«, erfahren Sie, wie Sie diese Daten auch grafisch auswerten können.

| column | count | unique | nulls | mean | std | min | max | median | 25_percent_cont | 25_percent_disc | 50_percent_cont | 50_percent_disc | 75_percent_cont | 75_percent_disc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CAR_ID | 263178 | 263178 | 0 | 185770.790461 | 107207.862523 | 3.0 | 371515.0 | 185584.0 | 92912.25 | 92912.0 | 185583.5 | 185583.0 | 278729.75 | 278730.0 |
| YEAROFREGISTRATION | 263178 | 107 | 0 | 2004.126971 | 33.809847 | 1000.0 | 9999.0 | 2004.0 | 2000.00 | 2000.0 | 2004.0 | 2004.0 | 2009.00 | 2009.0 |
| HP | 263178 | 668 | 0 | 125.365034 | 168.229849 | 0.0 | 20000.0 | 116.0 | 75.00 | 75.0 | 116.0 | 116.0 | 150.00 | 150.0 |
| KILOMETER | 263178 | 13 | 0 | 123037.601927 | 40559.039189 | 5000.0 | 150000.0 | 150000.0 | 100000.00 | 100000.0 | 150000.0 | 150000.0 | 150000.00 | 150000.0 |
| PRICE | 263178 | 5260 | 0 | 9130.201647 | 396101.314635 | 0.0 | 99999999.0 | 3999.0 | 1700.00 | 1700.0 | 3999.0 | 3999.0 | 8900.00 | 8900.0 |
| BRAND | 263178 | 40 | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| MODEL | 254228 | 250 | 8950 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| VEHICLETYPE | 249966 | 8 | 13212 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| FUELTYPE | 250495 | 7 | 12683 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| GEARBOX | 257435 | 2 | 5743 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

**Abbildung 4.6** Details über Spalteninhalte

Jede einzelne Zelle wurde in SAP HANA berechnet. Die `describe()`-Methode des `hana_ml`-Pakets hat die Anforderung in ein `SELECT`-Statement übersetzt, das in SAP HANA ausgeführt wurde. Die Berechnungen wurden entsprechend in SAP HANA ausgeführt, und nur die Ergebnisse wurden zurückgegeben und in Python ausgegeben. Das dafür automatisch erstellte `SELECT` lässt sich anzeigen. Die `print()`-Funktion stellt sicher, dass das SQL-Statement richtig angezeigt wird (siehe Tipp-Kasten zur Anzeige des SELECT-Statements).

```
print(df_remote.describe().select_statement)
```

**Anzeige des SELECT-Statements**

Das hier genutzte `SELECT`-Statement gibt den SQL-Code zurück, auf dem das Objekt basiert. Dieser Code kann direkt in SAP HANA ausgeführt werden. Sollte das SQL-Statement ein einzelnes Hochkommazeichen (`'`) beinhalten, stellt Python diesem bei der Anzeige einen Backslash (`\`) voran. Aus ' wird somit \' bei der Anzeige. Wenn Sie diesen Code kopieren und z. B. im SAP HANA Database Explorer ausführen möchten, erhalten Sie einen Fehler. Wenn Sie bei der Anzeige einen `print()`-Befehl nutzen, wird das Statement ohne Veränderung angezeigt und kann somit kopiert und aus einer

anderen Applikation heraus ausgeführt werden. Sie sehen das Verhalten z. B. bei diesem Statement `df_remote.describe('PRICE').select_statement` im Vergleich zum gültigen Statement mit `print()`-Befehl:

```
print(df_remote.describe('PRICE').select_statement)
```

Einige Spalten des Datensatzes weisen fehlende Werte auf, wie aus der **nulls**-Spalte der `describe()`-Methode ersichtlich. Wenn Sie nun herausfinden möchten, wie viele Zeilen vollständig sind, gibt es auch dafür eine passende Methode. Mit der `dropna()`-Methode werden alle Zeilen mit fehlenden Werten aus dem `hana_ml`-DataFrame herausgefiltert. Die `count()`-Methode zeigt an, dass noch 232.943 Zeilen im `hana_ml`-DataFrame verbleiben. Der Code für die beiden Methoden lautet:

```
df_remote.dropna().count()
```

Beachten Sie, dass die von der `dropna()`-Methode gefundenen Zeilen nicht aus der Tabelle gelöscht werden. Sie werden nur aus dem `hana_ml`-DataFrame herausgefiltert.

**Einschränkung auf weniger Spalten**

Bei einem Datensatz mit sehr vielen Spalten kann die Ausführung der `describe()`-Methode mehr Zeit in Anspruch nehmen. Schränken Sie in diesem Fall den Datensatz auf die Spalten ein, die Sie tatsächlich interessieren, indem Sie den Code entsprechend anpassen:

```
df_remote.describe(['HP', 'PRICE']).collect()
```

Des Weiteren bietet das `hana_ml`-Paket einen Dataset Report, der einen Überblick sowohl über die Tabellenstruktur als auch über die Dateninhalte gibt (siehe Abbildung 4.7). Erstellen Sie den Dataset Report für den `hana_ml`-DataFrame:

```
from hana_ml.visualizers.dataset_report import DatasetReportBuilder
datasetReportBuilder = DatasetReportBuilder()
datasetReportBuilder.build(df_remote, key="CAR_ID",
                           ignore_scatter_matrix=False,
                           ignore_correlation=False)
datasetReportBuilder.generate_notebook_iframe_report()
```

Der *Dataset Report* ist interaktiv. Zum Beispiel können Sie links im Menü über das Feld **Variables** eine Detailsicht für jede einzelne Variable erhalten. Bei einem sehr großen Datensatz kann die Erstellung des Reports mehr Zeit beanspruchen. Sie können den Bericht vereinfachen und die Wartezeit verkürzen, indem Sie die Parameter `ignore_scatter_matrix` und `ignore_correlation` auf **True** setzen.

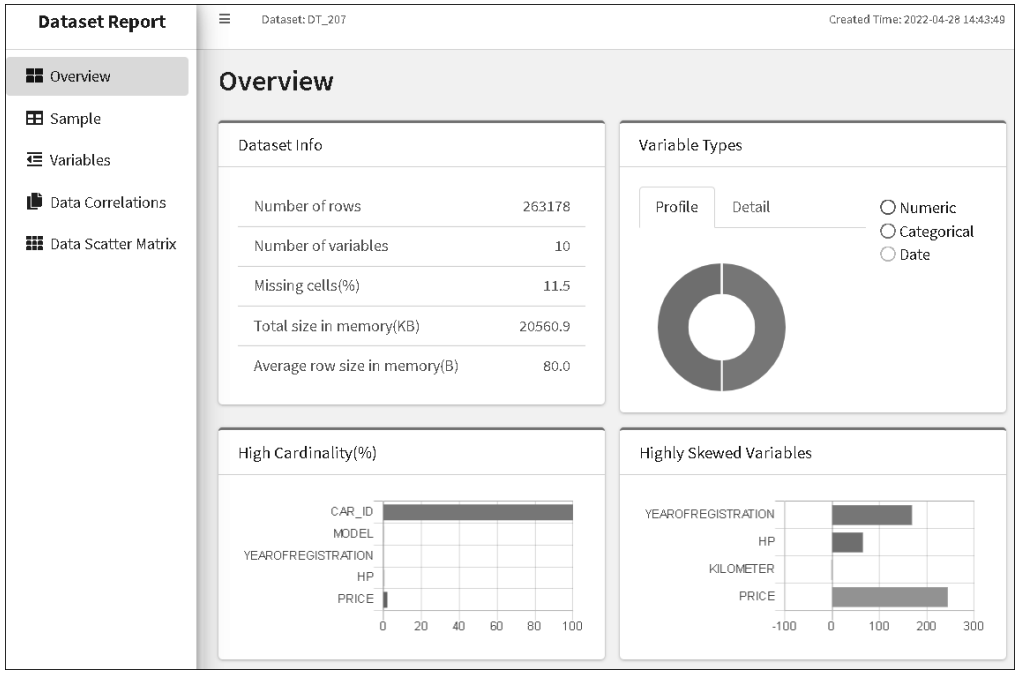


**Abbildung 4.7** Dataset Report

## 4.2 Analyse einzelner Variablen

Nun vertiefen wir die Analyse der Inhalte einzelner Variablen. In diesem Abschnitt werden zuerst numerische Variablen analysiert. Im Abschnitt darauf stehen kategorische Variablen im Fokus.

### 4.2.1 Numerische Variablen

Zur Analyse einer *numerischen Variablen* (man könnte auch von einer Kennzahl sprechen) ist nochmals die schon vorgestellte `describe()`-Methode zu erwähnen. Diese können Sie nutzen wie bisher und auf eine einzelne Variable einschränken, um diese zu analysieren. Hier sehen Sie beispielsweise die `describe()`-Methode mit der Einschränkung auf die Variable für den Preis:

```
df_remote.describe('PRICE').collect()
```

Einzelne Werte können über die `agg()`-Methode separat berechnet werden, wie z. B. das Minimum. Andere hier nutzbare Aggregate sind z. B. Maximum (`max`), Anzahl (`count`), Durchschnitt (`avg`), Summe (`sum`), Median (`median`) und weitere. Mit der folgenden Zeile wird z. B. der minimale Preis berechnet:

```
df_remote.agg([('min', 'PRICE', 'PRICE_MIN')]).collect()
```

Andere Ergebnisse aus der `describe()`-Methode hingegen führen weitere Schritte aus, die Sie aber auch nachstellen können. So wird, um den Durchschnitt der Preise zu berechnen, der Datentyp der Variablen von `INT` in `DOUBLE` umgewandelt, wie Sie folgend sehen. Das vermeidet einen Numeric-Overflow-Fehler.

```
df_remote.cast('PRICE', 'DOUBLE') \
         .agg([('avg', 'PRICE', 'PRICE_AVG')]).collect()
```

Neben der schon genutzten `describe()`-Methode können Sie auch einen *Box-Plot* erstellen lassen. Dieses Diagramm gibt einen schnellen Überblick der Verteilung der Daten. Führen Sie dazu diesen Code aus:

```
import matplotlib.pyplot as plt
from hana_ml.visualizers.eda import EDAVisualizer
f = plt.figure()
ax1 = f.add_subplot(111) # 111 refers to 1x1 grid, 1st subplot
eda = EDAVisualizer(ax1)
ax, cont = eda.box_plot(data=df_remote, column='PRICE')
```

Sie erhalten einen Box-Plot wie in Abbildung 4.8. Die gestrichelte Linie gibt den Median an. Eine Hälfte der Werte liegt unter diesem Punkt. Die andere Hälfte liegt darüber. Sie finden den Medianwert auch in der Ausgabe der `describe()`-Methode. Der blaue Balken gibt den Bereich vom ersten bis zum dritten Quartil an (siehe Abschnitt 4.1, »Analyse einer Tabelle«, für eine Erklärung der Quartile). Die blaue horizontale Linie (auch *Whisker* genannt) gibt an, wie sich die extremeren Werte unter dem ersten Quartil und über dem dritten Quartil verteilen. Diese werden über die sogenannte *Interquartile Range* (IQR) berechnet, der Differenz zwischen dem dritten und dem ersten Quartil. Von diesen IQR-Werten wird mit den folgenden Formeln das jeweilige Ende des erwähnten Whiskers berechnet:

Linkes Ende der Linie: erstes Quartil – 1.5 × IQR

Rechtes Ende der Linie: drittes Quartil + 1.5 × IQR

Falls das linke Ende unter dem Minimumwert beziehungsweise das rechte Ende über dem Maximumwert liegen sollte, wird der Wert entsprechend auf den Minimum- beziehungsweise Maximumwert eingerückt.

In Abschnitt 4.3, »Analyse mehrerer Variablen«, finden Sie eine verfeinerte Variante des Box-Plots, in dem die Verteilung je Ausprägung einer kategorischen Variablen angezeigt wird.

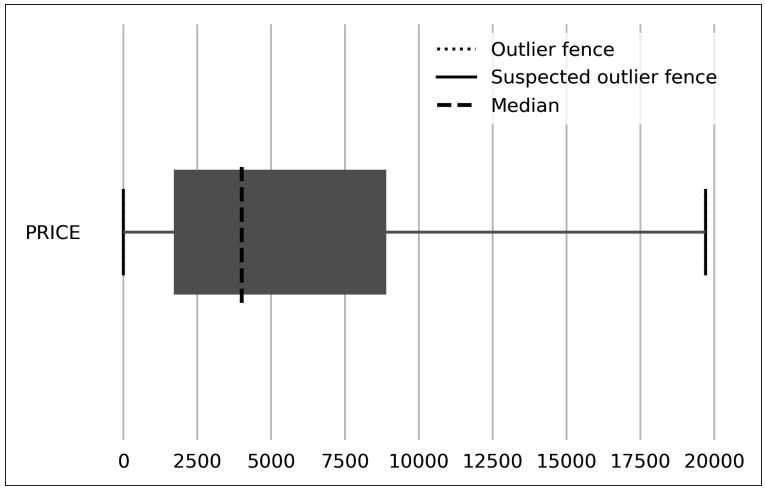


**Abbildung 4.8** Box-Plot der Variablen »PRICE« (ohne Ausreißer)

Beachten Sie, dass der Box-Plot in der Grundeinstellung ohne Ausreißer dargestellt wird. Sollten sich noch Werte über die Whisker hinaus im Datensatz befinden, werden diese nicht angezeigt. Sie können aber folgendermaßen angefordert werden:

```
f = plt.figure()
eda = EDAVisualizer(f.add_subplot(111))
ax, cont = eda.box_plot(data=df_remote, column='PRICE',
                        outliers=True)
```

Sollen Sie diesen Code ausführen, ist der Box-Plot kaum noch ersichtlich, da es einen Ausreißer mit dem Wert 99999999 gibt. Dieser führt dazu, dass der Bereich des ersten und dritten Quartils optisch auf eine kleine Linie gestaucht wird. Lassen Sie sich also die höchsten Preise anzeigen, indem Sie diesen Code ausführen:

```
df_remote.select('PRICE') \
         .sort('PRICE', desc=True) \
         .head(10).collect()
```

Abbildung 4.9 zeigt die zehn teuersten Fahrzeuge im Datensatz. Man kann davon ausgehen, dass diese Extremwerte nicht den tatsächlichen Wert des Fahrzeugs angeben. Eventuell mussten die Verkäufer*innen einen Preis angeben, und bei diesen Fahrzeugen wurde kein ernstgemeinter Wert eingetragen. Wir merken uns dies und werden später diese Ausreißer ausschließen, um die Data-Science-Methoden nicht in die Irre zu führen.

| PRICE |
|---|
| 99999999 |
| 99999999 |
| 99999999 |
| 99000000 |
| 14000500 |
| 12345678 |
| 12345678 |
| 12345678 |
| 11111111 |
| 11111111 |

**Abbildung 4.9** Die höchsten Preise im Datensatz

Wenn Sie nun analysieren wollen, wie häufig solch extreme Werte in dem Datensatz vorkommen, führen Sie den folgenden Code aus:

```
df_remote.filter('PRICE > 1000000').count()
```

Ihnen wird angezeigt, dass es in diesem Datensatz 26 Fahrzeuge gibt, die über 1 Million EUR kosten sollen. Aber auch die Anzahl der Fahrzeuge, die mit einem Preis über 100.000 EUR gespeichert sind, lässt sich so herausfinden. Ändern Sie den Befehl wie folgt ab:

```
df_remote.filter('PRICE > 100000').count()
```

326 Fahrzeuge werden für über 100.000 EUR angeboten. Fokussieren wir uns nun auf die Fahrzeuge, die weniger als 100.000 EUR kosten sollen. Eine sogenannte *Kerndichteschätzergrafik* gibt ein erstes Gefühl, wie die Daten in diesem Segment verteilt sind. Führen Sie diesen Code aus:

```
from hana_ml.visualizers.eda import kdeplot
import matplotlib.pyplot as plt
f = plt.figure(figsize=(10,6))
ax = kdeplot(df_remote.filter('PRICE < 100000'),
                              key='CAR_ID',
                              features=['PRICE'])
ax.grid()
plt.show()
```

In Abbildung 4.10 sehen Sie die entstandene Grafik. Daraus geht hervor, dass die meisten Fahrzeuge für einen sehr geringen Preis angeboten werden. Nach einem Peak bei den sehr günstigen Fahrzeugen fällt die Dichte schnell ab. Ab etwa 40.000 EUR gibt es nur noch sehr wenige Fahrzeuge.

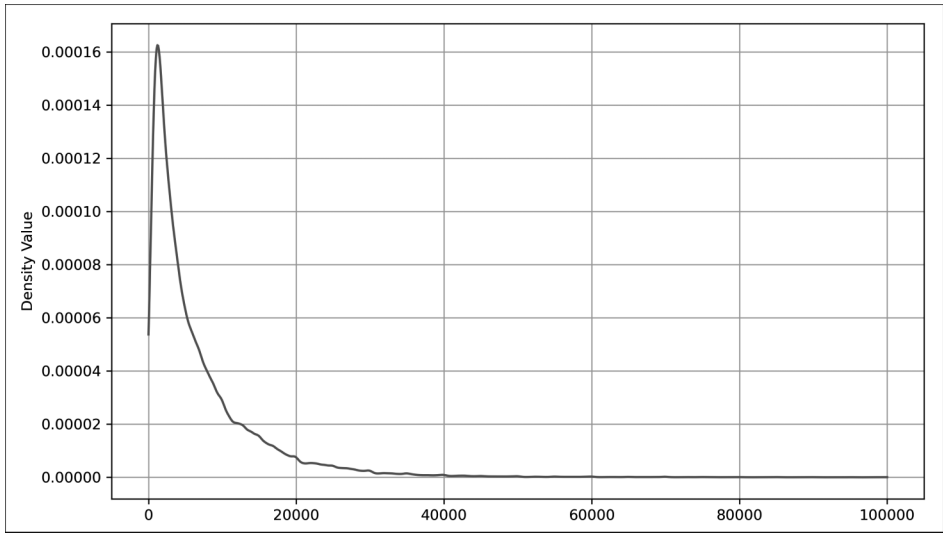


**Abbildung 4.10** Kerndichteschätzer für den Preis der Fahrzeuge

Berechnen wir abschließend das 99. Perzentil. Dies gibt den Grenzwert an, unter dem sich 99 % der Werte befinden. Nutzen Sie dazu diesen Code:

```
conn.sql('SELECT PERCENTILE_CONT(0.99) WITHIN GROUP(ORDER BY PRICE)\
         AS PERCENTILE FROM USEDCARS').collect()
```

Das Ergebnis zeigt: Diese Grenze liegt bei 39.900 EUR. 99 % der Fahrzeuge befinden sich darunter, das teuerste 1 % darüber.

### 4.2.2 Kategorische Variablen

Für eine *kategorische Variable* (die typischerweise Text beinhaltet, man könnte auch von einer Dimension sprechen, z. B. der Name eines Landes, eines Kundensegments oder Produkts) können Sie die Häufigkeit der vorkommenden Werte berechnen lassen. Bei unserem vorliegenden Datensatz könnten wir so z. B. herausfinden, was die zehn am häufigsten vorkommenden Hersteller im Datensatz sind. Das möchten wir direkt einmal ausprobieren. Führen Sie dafür den Code in Listing 4.1 aus.

```
top_n = 10
variable_name = 'BRAND'
total_count = df_remote.count()
df_remote_freq = df_remote.agg([('count', variable_name, 'COUNT')],
                               group_by=variable_name)
df_freq = df_remote_freq.sort("COUNT", desc=True) \
                        .head(top_n).collect()
df_freq['PERCENT'] = round(df_freq['COUNT'] / total_count, 2)
df_freq.style.format({'COUNT':'{0:,.0f}', 'PERCENT':'{0:,.1%}'}) \
       .hide(axis="index")
```

**Listing 4.1** Zählen der zehn häufigsten Hersteller

Das Ergebnis wird Ihnen als Tabelle angezeigt (siehe Abbildung 4.11).

| BRAND | COUNT | PERCENT |
|---|---|---|
| volkswagen | 55,486 | 21.0% |
| bmw | 30,385 | 12.0% |
| mercedes_benz | 26,900 | 10.0% |
| opel | 26,319 | 10.0% |
| audi | 24,492 | 9.0% |
| ford | 16,883 | 6.0% |
| renault | 11,506 | 4.0% |
| peugeot | 7,728 | 3.0% |
| fiat | 6,173 | 2.0% |
| seat | 5,066 | 2.0% |

**Abbildung 4.11** Die zehn häufigsten Hersteller im Datensatz als Tabelle

Die in Listing 4.1 verwendete Variable `df_freq` ist ein Pandas-DataFrame. Das heißt, dass Sie dessen Daten auch als Grafik darstellen können. Sie erhalten das Ergebnis, das in Abbildung 4.12 zu sehen ist. Führen Sie dafür diesen Code aus:

```
%matplotlib inline
df_freq.plot.bar(x='BRAND', y='COUNT', title='Top ' + str(top_n));
```

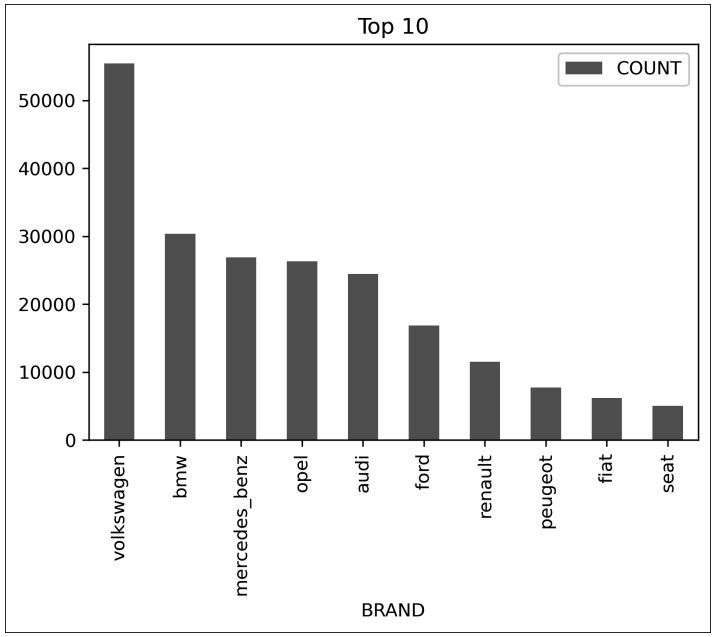


**Abbildung 4.12** Die zehn häufigsten Hersteller im Datensatz als Grafik

## 4.3 Analyse mehrerer Variablen

Nach der Analyse einzelner Variablen möchten wir nun das Verhältnis mehrerer Variablen zueinander analysieren. In Abschnitt 4.2.1, »Numerische Variablen«, haben wir die Verteilung der Fahrzeugpreise angesehen. Nun möchten wir die Verteilung der Fahrzeugpreise für die verschiedenen Kupplungstypen analysieren. Hierbei fokussieren wir uns nur auf die Fahrzeuge, die maximal 40.000 EUR kosten. Wie in Abschnitt 4.2.1, »Numerische Variablen«, gesehen, schließen wir damit die etwa 1 % der größten Ausreißer aus, die eine Analyse schnell ungewollt beeinflussen können. Führen Sie dafür zunächst diesen Code aus:

```
import matplotlib.pyplot as plt
from hana_ml.visualizers.eda import EDAVisualizer
f = plt.figure()
ax1 = f.add_subplot(111) # 111 refers to 1x1 grid, 1st subplot
eda = EDAVisualizer(ax1)
ax, cont = eda.box_plot(data=df_remote.filter('PRICE <= 40000'),
                        column='PRICE', groupby='GEARBOX', outliers=True)
ax.legend(bbox_to_anchor=(1, 1));
```

Das Ergebnis wird in Box-Plots dargestellt (siehe Abbildung 4.13). Diese zeigen, dass der Median der Fahrzeuge mit Automatikgetriebe am höchsten ist. Fahrzeuge mit Handschaltung sind hingegen günstiger. Beachten Sie auch den unteren Box-Plot mit der Beschriftung **MISSING**. Dieser Box-Plot gibt die Verteilung für Fahrzeuge an, für die keine Information über die Schaltung vorliegt. Die roten Punkte der Ausreißer (engl. Outlier) geben an, dass für alle drei Gruppen auch Fahrzeuge bis zu 40.000 EUR angeboten werden. Hierbei handelt es sich aber um Ausreißer.

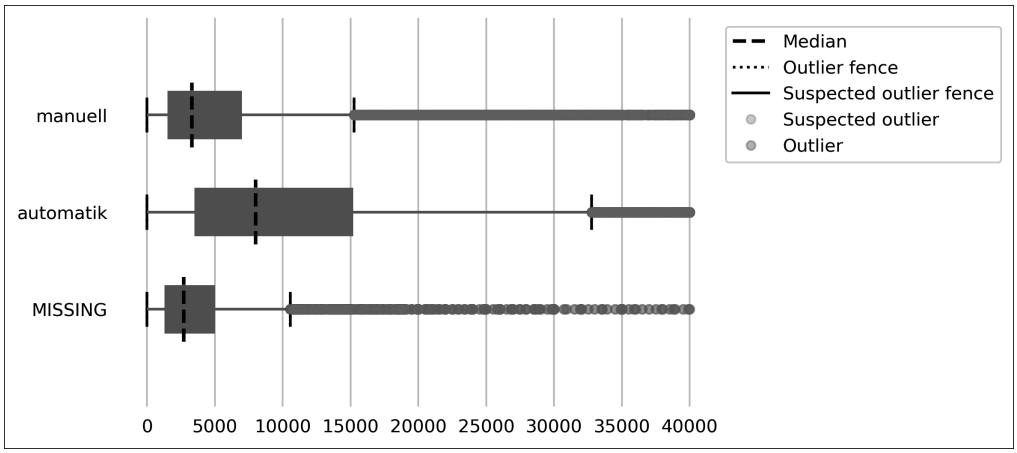


**Abbildung 4.13** Verteilung der Fahrzeugpreise nach Kupplungstypen

Über die `agg()`-Methode können Sie auch einzelne Werte wie den Median pro Kupplungstyp berechnen lassen. Führen Sie dazu den folgenden Code aus. Das Ergebnis sehen Sie in Abbildung 4.14.

```
df_remote.filter('PRICE <= 40000') \
   .agg([('median', 'PRICE', 'PRICE_MEDIAN')], group_by='GEARBOX') \
   .collect()
```

| GEARBOX | PRICE_MEDIAN |
|---|---|
| manuell | 3300 |
| automatik | 7999 |
| None | 2700 |

**Abbildung 4.14** Median des Fahrzeugpreises pro Kupplungstyp

Ebenso lässt sich der Median nach zwei kombinierten Dimensionen berechnen, hier Kupplungstyp und Motortyp (siehe Abbildung 4.15). Falls die Dimension der Spalten (hier FUELTYPE) fehlende Werte beinhalten sollte, werden diese automatisch durch den Wert **None** ersetzt. Der Code zur Berechnung lautet:

```
df_remote.filter('PRICE <= 40000') \
         .pivot_table(values='PRICE', aggfunc='median',
                      index='GEARBOX', columns='FUELTYPE').collect()
```

| GEARBOX | hybrid | elektro | andere | lpg | cng | benzin | diesel | None |
|---|---|---|---|---|---|---|---|---|
| manuell | 9660 | 3600 | 1700 | 2950 | 4400 | 2600 | 5550 | 1699 |
| automatik | 12500 | 6625 | 2650 | 5500 | 5800 | 5790 | 10900 | 3990 |
| None | 33000 | 6499 | 515 | 2700 | 7500 | 1999 | 3300 | 2500 |

**Abbildung 4.15** Median des Fahrzeugpreises pro Kupplungstyp und Motortyp

Abschließend für die Datenanalyse erstellen wir eine *Korrelationsmatrix*, die die Stärke und Richtung eines linearen Zusammenhangs zwischen zwei numerischen Variablen berechnet. Für jede Kombination zweier numerischer Variablen wird die Pearson-Korrelation angezeigt. Diese skaliert von –1 bis 1. Bei einem Wert von 0 gibt es keinerlei linearen Zusammenhang. Bei einem Wert von 1 gibt es einen perfekt positiven, linearen Zusammenhang. Dies bedeutet, dass sich die eine Variable in dieselbe Richtung bewegt, wenn sich eine andere Variable verändert. Erhöht sich z. B. eine Variable um einen Wert, erhöht sich die andere Variable ebenso in immer demselben Verhältnis. Bei einer linearen, negativen Korrelation bewegen sich die Variablen in unterschiedliche Richtungen. Erhöht sich die eine, reduziert sich die andere. An einem Beispiel lässt sich dies leicht verdeutlichen. Führen Sie den folgenden Code aus, um die Korrelationsmatrix für die Gebrauchtfahrzeuge zu berechnen. Wir arbeiten weiterhin mit den Fahrzeugen unter 40.000 EUR, da die positiven Ausreißer die Analyse schwer beeinflussen würden.

```
import matplotlib.pyplot as plt
from hana_ml.visualizers.eda import EDAVisualizer
f = plt.figure()
ax1 = f.add_subplot(111) # 111 refers to 1x1 grid, 1st subplot
eda = EDAVisualizer(ax1)
ax, corr_data = eda.correlation_plot(data=df_remote.drop('CAR_ID') \
                                                   .filter('PRICE < 40000'),
                                     cmap='coolwarm')
```

Die Korrelationsmatrix in Abbildung 4.16 zeigt z. B. eine Pearson-Korrelation von –0.46 für die Kombination von Kilometer und Preis. Diese vergleichsweise recht starke lineare Korrelation bedeutet, dass sich tendenziell bei einer Erhöhung der gefahrenen Kilometer der Preis reduziert. Die Pearson-Korrelation zwischen PS und Preis hingegen gibt an, dass sich tendenziell bei einer Erhöhung der Pferdestärke auch der

Preis erhöht. Werden in einer Korrelationsmatrix überraschende Werte angezeigt, sollten diese nachverfolgt werden. Eventuell existiert ein Problem mit der Datenqualität, oder die Daten sind korrekt und deuten auf eine neue Erkenntnis hin.

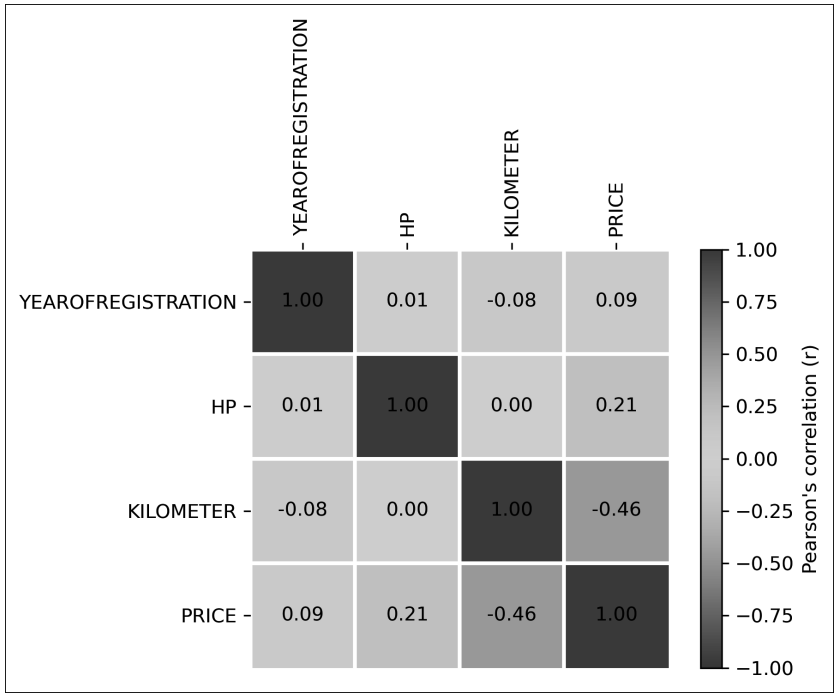


**Abbildung 4.16** Korrelationsmatrix der numerischen Spalten

## 4.4 Datenvorbereitung

Oftmals müssen die vorhandenen Daten noch weiter aufbereitet werden, um für Data-Science-Anforderungen geeignet zu sein. In diesem Abschnitt erläutern wir, wie Sie aus zwei Tabellen mit Rohdaten einen einheitlichen Datensatz zum Vorhersagen von Kundenverhalten erstellen. Beide Tabellen sind zur Erklärung der Datenvorbereitung bewusst einfach gehalten. Wir bereiten die Daten für eine Klassifizierung vor, mit der wir die Wahrscheinlichkeit schätzen möchten, dass ein Kontakt, der bislang noch kein Produkt erworben hat, in den nächsten 30 Tagen seinen ersten Kauf tätigen wird. Mit einer Klassifizierung können z. B. Personen einer von zwei Klassen zugewiesen werden, wie Käufer*innen und Nicht-Käufer*innen. In Abschnitt 5.2, »Klassifizierung mit der APL«, wird die Klassifizierung genauer erläutert. Wenn Sie so schnell wie möglich erste Algorithmen anwenden möchten, können Sie auch mit Kapitel 5 fortfahren und später zu dieser Datenvorbereitung zurückkommen.

### 4.4.1 Datenbeschreibung

Die Tabelle CONTACTS beinhaltet Stammdaten über Kund*innen und weitere Personen, die zumindest Interesse an den Produkten eines Unternehmens haben (siehe Tabelle 4.1). Folgende Spalten finden Sie in der Datei **CONTACTS.csv**, die Sie in den Materialen zu diesem Buch unter *www.sap-press.de/5539* herunterladen können.

| Spalte | Beschreibung |
|---|---|
| CUSTOMERID | ID der Person (Kund*in oder Interessent*in) |
| GENDER | Geschlecht |
| DATEOFBIRTH | Geburtsdatum |

**Tabelle 4.1** Spalten der Tabelle »CONTACTS.csv«

Die Tabelle INTERACTIONS beinhaltet Informationen über Aktivitäten/Interaktionen der Kontakte (siehe Tabelle 4.2). Folgende Spalten finden Sie Datei **INTERACTIONS.csv**, die Sie ebenfalls in den Download-Materialien zu diesem Buch finden.

| Spalte | Beschreibung |
|---|---|
| INTERACTIONDATE | Tag, an dem die Aktivität beziehungsweise Interaktion stattfand |
| CUSTOMERID | ID der Person (Kund*in oder Interessent*in) |
| TYPE | Art der Aktivität/Interaktion:<br>■ WEBSITE_VISIT: Die Person hat die Website besucht.<br>■ WEBSITE_SEARCH: Die Person hat auf der Webseite eine Suche durchgeführt.<br>■ PURCHASE: Die Person hat einen Kauf getätigt. |

**Tabelle 4.2** Spalten der Tabelle »INTERACTIONS.csv«

Laden Sie die beiden Dateien mit den Daten aus dem Downloadmaterial dieses Buchs herunter. Sie bilden die Grundlage für die nächsten Schritte.

### 4.4.2 Daten laden

Überführen Sie die Daten der Datei **CONTACTS.csv** in Ihr SAP-HANA-System. Laden Sie zuerst die CSV-Datei als Pandas-DataFrame in Ihrer lokalen Python-Umgebung und zeigen Sie zur Überprüfung, dass dies korrekt funktioniert hat, mit diesem Code die ersten Zeilen an:

```
import pandas as pd
df_contacts = pd.read_csv('CONTACTS.csv', sep=',')
df_contacts.DATEOFBIRTH = pd.to_datetime(df_contacts.DATEOFBIRTH)
df_contacts.head(5)
```

Erstellen Sie nun eine Verbindung zu SAP HANA. Dieses Beispiel nutzt die in Abschnitt 3.1.2, »Verbindung zu SAP HANA«, hinterlegten Login-Daten:

```
import hana_ml.dataframe as dataframe
conn = dataframe.ConnectionContext(userkey='MYHANA')
conn.connection.isconnected()
```

Bei einer erfolgreichen Verbindung wird Ihnen der Wert **True** angezeigt. Mit der Verbindung können die Daten nach SAP HANA geladen werden. Führen Sie dazu diesen Code aus:

```
df_rem_contacts = dataframe.create_dataframe_from_pandas(
                              connection_context=conn,
                              pandas_df=df_contacts,
                              table_name='CONTACTS',
                              force=True,
                              drop_exist_tab=True,
                              replace=False)
```

Lassen Sie sich einige Zeilen anzeigen (siehe Abbildung 4.17). Der Wert **NaT** (Abkürzung für »not a time«) gibt an, dass für diese Person kein Geburtsdatum bekannt ist.

```
df_rem_contacts.head(5).collect()
```

| CUSTOMERID | GENDER | DATEOFBIRTH |
|---|---|---|
| CXID000001 | Female | 1960-02-24 |
| CXID000002 | Female | NaT |
| CXID000003 | None | 1987-03-25 |
| CXID000004 | Female | NaT |
| CXID000005 | Male | 1964-08-11 |

**Abbildung 4.17** Einträge der CONTACTS-Tabelle

Entsprechend verfahren Sie mit der Datei **INTERACTIONS.csv**, um diese als Tabelle in SAP HANA anzulegen:

```
df_interactions = pd.read_csv('INTERACTIONS.csv', sep=',')
df_rem_interactions = dataframe.create_dataframe_from_pandas(
                              connection_context=conn,
```

```
                    pandas_df=df_interactions,
                    table_name='INTERACTIONS',
                    force=True,
                    drop_exist_tab=True,
                    replace=False)
```

Lassen Sie sich auch hier einige Zeilen dieser Tabelle mit dem folgenden Code anzeigen (siehe Abbildung 4.18).

```
df_rem_interactions.head(5).collect()
```

| INTERACTIONDATE | CUSTOMERID | TYPE |
|---|---|---|
| 2021-12-05 | CXID002873 | WEBSITE_VISIT |
| 2021-12-05 | CXID002486 | WEBSITE_VISIT |
| 2021-12-05 | CXID008417 | WEBSITE_VISIT |
| 2021-12-05 | CXID009515 | WEBSITE_VISIT |
| 2021-12-05 | CXID002817 | WEBSITE_VISIT |

**Abbildung 4.18** Einträge der INTERACTIONS-Tabelle

Die Daten der Tabellen CONTACTS und INTERACTIONS sind nun in Ihrem SAP-HANA-System geladen, im nächsten Schritt zeigen wir Ihnen, wie Sie diese Daten modifizieren.

### 4.4.3 Datenmodifikation

Nutzen Sie die in den vorherigen Abschnitten vorgestellte `describe()`-Methode, für eine erste Analyse der CONTACTS-Tabelle:

```
df_rem_contacts.describe().collect()
```

Das Ergebnis in Abbildung 4.19 zeigt z. B., dass die Zeile des Geschlechts insgesamt vier verschiedene wie auch fehlende Werte beinhaltet.

| column | count | unique | nulls | mean | std | min | max | median | 25_percent_cont | 25_percent_disc | 50_percent_cont | 50_percent_disc | 75_percent_cont | 75_percent_disc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CUSTOMERID | 10000 | 10000 | 0 | None | None | None | None | None | None | None | None | None | None | None |
| GENDER | 8495 | 4 | 1505 | None | None | None | None | None | None | None | None | None | None | None |
| DATEOFBIRTH | 6052 | 5204 | 3948 | None | None | None | None | None | None | None | None | None | None | None |

**Abbildung 4.19** CONTACTS-Tabelle

Sehen wir uns an, welche Werte in der GENDER-Zeile vorkommen. Da auch fehlende Werte enthalten sind, werden diese im folgenden Code mit dem String `NONE` ersetzt. Andernfalls könnte die Häufigkeit der fehlenden Werte nicht gezählt werden.

```
df_rem_contacts.fillna('NONE', subset=['GENDER']) \
               .agg([('count', 'GENDER', 'COUNT')],
                    group_by='GENDER').collect()
```

Das Ergebnis, in Abbildung 4.20 zu sehen, zeigt, dass neben den Werten **Female** und **Male** auch die Werte **M** und **F** in der Spalte vorkommen.

| GENDER | COUNT |
|---|---|
| F | 52 |
| Male | 2314 |
| M | 38 |
| NONE | 1505 |
| Female | 6091 |

**Abbildung 4.20** Häufigkeit der Einträge der GENDER-Spalte

Es handelt sich hierbei um ein Datenqualitätsproblem. Die Werte **Male** und **M** sowie **Female** und **F** sollten zusammengelegt werden. Aber auch dieses Problem können wir lösen. Erstellen Sie dazu einen neuen `hana_ml`-DataFrame, in dem die Einträge auf **Female** und **Male** standardisiert werden. Gleichzeitig werden fehlende Werte in der GENDER-Spalte durch den String `Unknown` ersetzt. Den vollständigen Code sehen Sie hier:

```
df_rem_prepped = \
     df_rem_contacts.select(['CUSTOMERID', 'DATEOFBIRTH', \
                             ("""CASE WHEN GENDER = 'M' THEN 'Male'
                                 WHEN GENDER = 'F' THEN 'Female'
                                 WHEN GENDER IS NULL THEN 'Unknown'
                                 ELSE GENDER END""", 'GENDER')])
```

Damit werden die Daten in der Tabelle CONTACTS nicht verändert. Die Anpassung befindet sich momentan lediglich in der Logik des `hana_ml`-DataFrames. Diesen DataFrame werden wir in diesem Abschnitt noch weiter anreichern und später als View in SAP HANA speichern.

Lassen Sie sich die Häufigkeit der GENDER-Einträge des modifizierten `hana_ml`-DataFrames `df_rem_prepped` anzeigen:

```
df_rem_prepped.fillna('NONE', subset=['GENDER']) \
               .agg([('count', 'GENDER', 'COUNT')],
                    group_by='GENDER').collect()
```

Abbildung 4.21 zeigt, wie die Einträge der GENDER-Spalte bereinigt wurden.

| GENDER | COUNT |
|---|---|
| Male | 2352 |
| Unknown | 1505 |
| Female | 6143 |

**Abbildung 4.21** Häufigkeit der Einträge der modifizierten GENDER-Spalte

Nun wenden wir uns der Zeile DATEOFBIRTH zu, in dem das Geburtsdatum der Personen vermerkt ist. Für ein Machine-Learning-Modell ist solch ein exaktes Datum nicht besonders hilfreich. Um ein Muster in den Daten erkennen zu können, das sich auch in die Zukunft fortschreiben lässt, ist es zu empfehlen, das Geburtsdatum in das Alter der Person umzurechnen. Mit dieser kontinuierlichen Variablen kann ein Machine-Learning-Modell nach einem Muster in den Daten suchen, dass z. B. Personen mit ähnlichem Alter auch ein ähnliches Verhalten zeigen.

Passen Sie dazu den `hana_ml`-DataFrame `df_rem_prepped` weiter an, indem Sie das Geburtsdatum durch das Alter in der Zeile AGE ersetzen:

```
df_rem_prepped =  df_rem_prepped \
     .select('CUSTOMERID', 'GENDER',
            ('YEARS_BETWEEN(DATEOFBIRTH, CURRENT_DATE)', 'AGE'))
```

Des Weiteren ist für mehrere Tausend Personen das Geburtsdatum und damit auch das Alter nicht bekannt. Viele Data-Science-Methoden in SAP HANA, wie die Automated Predictive Library, können mit solch fehlenden Werten umgehen. Falls Sie sich jedoch selbst um die fehlenden Werte kümmern möchten, können Sie z. B. das durchschnittliche Alter pro Geschlecht berechnen und die fehlenden Werte durch den Durchschnitt des jeweiligen Geschlechts ersetzen.

Berechnen Sie das durchschnittliche Alter pro Geschlecht und weisen Sie die Logik einem weiteren `hana_ml`-DataFrame zu, hier `df_rem_agg`:

```
df_rem_agg =  df_rem_prepped \
   .agg([('median', 'AGE', 'AGE_MEDIAN')], group_by='GENDER')
df_rem_agg.head(5).collect()
```

Verknüpfen Sie beide `hana_ml`-DataFrames über die GENDER-Spalte mit dem folgenden Code:

```
df_rem_prepped.set_index('GENDER')
df_rem_agg.set_index('GENDER')
df_rem_prepped = df_rem_prepped.join(df_rem_agg)
```

Geben Sie nun an, dass fehlende Werte in der Altersspalte mit dem Durchschnitt des jeweiligen Geschlechts ersetzt werden.

```
df_rem_prepped = df_rem_prepped.select('CUSTOMERID', 'GENDER',
     ("CASE WHEN AGE IS NULL THEN AGE_MEDIAN ELSE AGE END", 'AGE'))
```

Führen Sie nun die `describe()`-Methode aus, erkennt diese, dass sich über 200-jährige Personen im Datensatz befinden:

```
df_rem_prepped.describe().collect()
```

Dies ist eindeutig ein weiteres Datenqualitätsproblem. Wir entscheiden uns dazu, alle Personen, deren Alter mit über 130 Jahre angegeben ist, aus dem Datensatz zu entfernen. Führen Sie dazu diesen Befehl aus:

```
df_rem_prepped = df_rem_prepped.filter('AGE <= 130')
```

Wenn Sie nun erneut die `describe()`-Methode aufrufen, sehen Sie die nun aufbereiteten Daten der Tabelle (siehe Abbildung 4.22).

| column | count | unique | nulls | mean | std | min | max | median |
|---|---|---|---|---|---|---|---|---|
| AGE | 9998 | 58 | 0 | 44.729046 | 11.718254 | 18.0 | 75.0 | 45.0 |
| CUSTOMERID | 9998 | 9998 | 0 | NaN | NaN | NaN | NaN | NaN |
| GENDER | 9998 | 3 | 0 | NaN | NaN | NaN | NaN | NaN |

**Abbildung 4.22** Übersicht der angepassten Daten der CONTACTS-Tabelle

Wir sind mit dieser Aufbereitung zufrieden und speichern sie als View in SAP HANA:

```
df_rem_prepped.save('V_CONTACTS_PREPPED',
                    table_type='VIEW', force=True)
```

Der View beinhaltet die über das `hana_ml`-Paket definierten Transformationen. Beim Aufruf des View werden diese Transformationen auf der Tabelle ausgeführt. Somit werden die Transformationen auf allen Daten ausgeführt, auch auf Daten, die eventuell erst nach dem Erstellen des View in die Tabelle hinzukamen. Die ursprüngliche CONTACTS-Tabelle bleibt unverändert. Andere Prozesse, die mit diesen Daten bereits arbeiten, finden weiterhin die erwartete Tabellenstruktur und Inhalte. Für unsere Anforderungen hingegen können wir künftig mit diesem View arbeiten, der auf den gesamten Datenbestand die definierten Transformationen durchführt.

Alle Transformationen, die in diesem Abschnitt durchgeführt wurden, sind über das `hana_ml`-Paket in SQL-Logik übertragen worden, weswegen die Datensicht auch als View gespeichert werden könnte. Den zugrunde liegenden SQL-Befehl können Sie sich hiermit anzeigen lassen:

```
print(df_rem_prepped.select_statement)
```

Der erstellte View kann wie jeder andere View auch benutzt werden. Sie sehen ihn z. B. im Datenbank-Explorer. Oder Sie können ihn direkt als Grundlage für einen hana_ml-DataFrame verwenden, indem Sie diesen Code nutzen:

```
df_rem_prepped = conn.table('V_CONTACTS_PREPPED')
```

**Speichern eines »hana_ml«-DataFrames als Tabelle**

Im obigen Beispiel wird der hana_ml-DataFrame als View gespeichert, um die Transformationslogik einfach auf neue Daten anwenden zu können. Sollten Sie die transformierten Daten aber als Tabelle speichern möchten, müssen Sie beim Speichern dem Parameter table_type nur den Wert TABLE zuweisen. Da TABLE der Default-Wert ist, können Sie den Parameter auch weglassen, wie Sie hier sehen:

```
df_rem_prepped.save('T_CONTACTS_PREPPED', force=True)
```

Die Anpassung der Stammdaten der CONTACTS-Tabelle ist hiermit beendet. Nun nutzen wir die detaillierten transaktionalen Daten der INTERACTIONS-Tabelle. Wir können diese Tabelle für unterschiedliche Zwecke verwenden. Beginnen wir mit der Definition der Population. Unser Machine-Learning-Modell soll die Wahrscheinlichkeit schätzen, dass eine Person in den nächsten 30 Tagen einen ersten Kauf tätigen wird. Hierzu lernen wir von den Personen, die in den 30 letzten Tagen ihren ersten Kauf getätigt haben, und wie sich die Personen vor diesen 30 Tagen verhalten haben. Ergo interessieren uns nur die Personen, die vor 30 Tagen noch kein Produkt von uns erworben haben.

Nehmen wir an, heute ist der 30. März 2022. Wir fokussieren uns also auf die Kontakte, die bis zum 28. Februar 2022 kein Produkt bei uns erworben haben. Käufe sind in der INTERACTIONS-Tabelle mit Einträgen des Typs PURCHASE festgehalten. Welche Kontakte haben also bis zum 28. Februar keinen entsprechenden Eintrag des Typs PURCHASE?

Um dies herauszufinden, erstellen Sie einen hana_ml-DataFrame, der auf die Tabelle INTERACTIONS verweist:

```
df_rem_interactions = conn.table('INTERACTIONS')
```

Über diesen DataFrame erstellen wir einen weiteren DataFrame, der nur Kontakte beinhaltet, die bis zum Stichtag einen Kauf getätigt haben:

```
df_rem_customers = df_rem_interactions.filter("TYPE = 'PURCHASE' AND
INTERACTIONDATE <= '2022-02-28'").distinct(['CUSTOMERID', 'TYPE'])
```

Dieser DataFrame mit den Käufer*innen hilft uns, eine Liste der Personen zu erstellen, die nichts gekauft haben. Verknüpfen Sie einen `hana_ml`-DataFrame aller Kontakte über einen Left-Join mit dem `hana_ml`-DataFrame der Käufer*innen:

```
df_rem_target = conn.table('V_CONTACTS_PREPPED')
df_rem_target.set_index('CUSTOMERID')
df_rem_customers.set_index('CUSTOMERID')
df_rem_target = df_rem_target.join(df_rem_customers, how='left')
```

Der Left-Join stellt sicher, dass alle Kontakte des Views `V_CONTACTS_PREPPED` im Datensatz verbleiben. Da nur die Käufer*innen einen Eintrag in der TYPE-Spalte der INTERACTIONS-Tabelle haben, müssen wir nur auf die Kontakte ohne Eintrag in der TYPE-Spalte filtern, und wir haben die Nicht-Käufer zum Zeitpunkt des Stichtags identifiziert. Dafür geben Sie diesen Code ein:

```
df_rem_target = df_rem_target.filter('TYPE IS NULL')
df_rem_target = df_rem_target.drop('TYPE')
df_rem_target.count()
```

Das Ergebnis zeigt, dass bis zum Stichtag des 28. Februar 2022 7.586 Kontakte noch kein Produkt erworben hatten. Damit haben wir die Population gefunden, auf der wir das Machine-Learning-Modell erstellen können.

Im nächsten Schritt erstellen wir die Zielvariable, auf der ein Machine-Learning-Modell trainiert werden kann. Dafür müssen wir zunächst in einer Variablen beschreiben, ob die 7.586 Kontakte, die uns interessieren, in den 30 Tagen nach dem Stichtag einen Kauf getätigt haben. Zuerst erstellen wir dafür den `hana_ml`-DataFrame `df_rem_purchase`, der alle Kontakte beinhaltet, die in dem Zeitraum dieser 30 Tage nach dem Stichtag einen Kauf getätigt haben. Die Spalte FUTUREPURCHASE bekommt den Wert `Yes` für all diese Kontakte.

```
df_rem_purchase = df_rem_interactions.filter("TYPE = 'PURCHASE' \
     AND INTERACTIONDATE BETWEEN ADD_DAYS('2022-02-28', 1) \
     and ADD_DAYS('2022-02-28', 30)").distinct(['CUSTOMERID'])
df_rem_purchase = df_rem_purchase. \
     select('CUSTOMERID', ("'Yes'", 'FUTUREPURCHASE'))
```

Mit dem folgenden Code finden wir heraus, dass es sich um 680 Kontakte handelt, die in den 30 Tagen ein Produkt gekauft haben.

```
df_rem_purchase.count()
```

Wir reichern den bereits bestehenden `hana_ml`-DataFrame `df_rem_purchase`, der die Population beinhaltet, mit der Zielvariablen `FUTUREPURCHASE` an. Mit einem Left-Join ist sichergestellt, dass nach dem Join weiterhin nur die Kontakte ohne Kauf bis zum

Stichtag im Datensatz enthalten sind. Das Ergebnis des Join ist in dem `hana_ml`-DataFrame `df_rem_360` festgehalten.

```
df_rem_target.set_index('CUSTOMERID')
df_rem_purchase.set_index('CUSTOMERID')
df_rem_360 = df_rem_target.join(df_rem_purchase, how='left')
```

Wir werden diesen DataFrame im Laufe des Kapitels mit weiteren Details anreichern.

Kontakte aus der Population, die in den darauffolgenden 30 Tagen den ersten Kauf tätigten, haben in der Spalte FUTUREPURCHASE wie gewollt den Eintrag `Yes`. Für Kontakte in dem DataFrame, die auch in den folgenden 30 Tagen keinen Kauf getätigt haben, weist die Spalte FUTUREPURCHASE noch keinen Wert auf. Wir ersetzen diesen fehlenden Wert mit `No`:

```
df_rem_360 = df_rem_360 \
     .fillna(value='No', subset=['FUTUREPURCHASE'])
```

Die Zielvariable ist erstellt. Wir lassen uns die Struktur der Daten anzeigen (siehe Abbildung 4.23):

```
df_rem_360.describe().collect()
```

| column | count | unique | nulls |
|---|---|---|---|
| AGE | 7586 | 53 | 0 |
| CUSTOMERID | 7586 | 7586 | 0 |
| GENDER | 7586 | 3 | 0 |
| FUTUREPURCHASE | 7586 | 2 | 0 |

**Abbildung 4.23** Datenstruktur mit Zielvariable

Nun nutzen wir die Interaktionen, um das Verhalten der Kontakte vor der Zielperiode zu beschreiben. Wir weisen dem `hana_ml`-DataFrame alle Interaktionen zu, die in den 30 Tagen vor dem Stichtag festgehalten wurden.

```
df_rem_interactions = df_rem_interactions.filter \
     ("INTERACTIONDATE BETWEEN ADD_DAYS('2022-02-28', -30) \
       AND '2022-02-28'")
```

Aus dieser Liste zählen wir, wie oft der Kontakt unsere Webseite besucht hat, indem wir diesen Code ausführen:

```
df_rem_webvisits = df_rem_interactions \
     .filter("TYPE = 'WEBSITE_VISIT'") \
```

```
    .agg([('count', 'CUSTOMERID', 'WEBSITE_VISITS_30DAYS')], \
        group_by='CUSTOMERID')
```

Ebenso halten wir fest, wie oft der Kontakt die Suchfunktion unserer Webseite in diesem Zeitraum (30 Tage vor dem Stichtag) genutzt hat:

```
df_rem_websearches = df_rem_interactions \
    .filter("TYPE = 'WEBSITE_SEARCH'") \
    .agg([('count', 'CUSTOMERID', 'WEBSITE_SEARCHES_30DAYS')], \
        group_by='CUSTOMERID')
```

Nun hängen wir die neuen Spalten an den bestehenden `hana_ml`-DataFrame `df_rem_360`. Für Kontakte, die die Webseite nicht besucht oder keine Suche durchgeführt haben, sind die jeweiligen Aggregate/Spalten noch leer. Wir ersetzen diesen fehlenden Wert in folgendem Listing mit einer numerischen 0:

```
df_rem_360 = df_rem_360.set_index('CUSTOMERID').join([
    df_rem_webvisits .set_index('CUSTOMERID'),
    df_rem_websearches.set_index('CUSTOMERID')], how='left')
df_rem_360 = df_rem_360.fillna(0, \
    ['WEBSITE_VISITS_30DAYS', 'WEBSITE_SEARCHES_30DAYS'])
```

Damit haben wir den Datensatz in diesem Beispiel für unseren Stichtag fertiggestellt (siehe Abbildung 4.24). Verwenden Sie diesen Code, um sich das Ergebnis anzeigen zu lassen:

```
df_rem_360.describe().collect()
```

| column | count | unique | nulls | mean |
|---|---|---|---|---|
| AGE | 7586 | 53 | 0 | 44.529133 |
| WEBSITE_VISITS_30DAYS | 7586 | 8 | 0 | 0.521751 |
| WEBSITE_SEARCHES_30DAYS | 7586 | 11 | 0 | 0.424730 |
| CUSTOMERID | 7586 | 7586 | 0 | NaN |
| GENDER | 7586 | 3 | 0 | NaN |
| FUTUREPURCHASE | 7586 | 2 | 0 | NaN |

**Abbildung 4.24** Finale Datenstruktur

**Anreichern eines Datensatzes für das Machine Learning**

In einem Projekt würden Sie an diesen Punkt wahrscheinlich versuchen, das Verhalten der Personen über weitere Spalten zu beschreiben. Hier können Sie kreativ werden und versuchen, potenziell relevante Spalten zur Verfügung zu stellen. Dies können weitere verhaltensbezogene Spalten sein, wie das Verhalten in länger zu-

rückliegenden Zeiträumen oder eine Veränderung des Verhaltens. Wir wissen aktuell, wie sich die Person in den letzten 30 Tagen verhalten hat. Aber wie war das Verhalten z. B. in den 30 Tagen davor? Dies kann eine weitere Variable sein. Hat sich das Verhalten verändert und wenn ja, wie? Auch dies kann eine weitere Variable sein, die die Differenz der einzelnen Zeiträume berechnet. Aber auch ganz andere Daten, wie der Zeitraum, seitdem die Person bei uns registriert ist, sind hier denkbar. Wie in Abschnitt 1.1.5 zum Thema CRISP-DM-Prozessmodell beschrieben, hilft das zunehmende Verständnis bei der Aufbereitung des Datensatzes, z. B. bei der Identifizierung und Erstellung weiterer Variablen. Bei der Erstellung der Variablen ist der in Kapitel 1, »Einführung«, erwähnte Subject Matter Expert (SME) eine wichtige Unterstützung, z. B. für weitere Ideen oder zur Verifizierung Ihrer Vorstellungen.

Um solch einen Datensatz dauerhaft einsetzen zu können, ist es hilfreich, wenn Sie flexibel einen Stichtag benennen können und der View sich darauf entsprechend anpasst. Nur so lässt sich das Trainieren und Anwenden von Data- Science-Methoden durchweg automatisieren.

Im obigen Beispiel sind wir davon ausgegangen, dass heute der 30 März ist. Um aus der Vergangenheit lernen zu können, ist die Sicht der Daten auf den 28. Februar gelegt. So kann eine Klassifizierung lernen, welches Kundenverhalten auf einen ersten Kauf hindeutet. Das trainierte Modell kann dann auf einen Datensatz aus heutiger Sicht angewendet werden. Dazu muss die Datenlogik mit dem heutigen Datum reproduziert werden. Ähnlich verhält es sich, wenn Sie in der Folgewoche ein neues Machine-Learning-Modell erstellen und anwenden möchten. Auch hier müssen die Datumstempel entsprechend angepasst werden. Die Logik der Daten bleibt bestehen, aber die Berechnungen beziehen sich immer auf unterschiedliche Datumstempel.

Für jede Anforderung, die Logik mit einem unterschiedlichen Datumstempel jedes Mal neu aufzusetzen, wäre zu aufwendig. Für einen effizienten und flexiblen Einsatz können Sie die Datenlogik als parametrisierten View aufrufen, und dabei lediglich das entsprechende Datum als Parameter übergeben.

Der parametrisierte View ist einfach zu erstellen. Der `hana_ml`-DataFrame `df_rem_360` repräsentiert bereits das SQL-Statement für das fix gesetzte Datum vom 28. Februar 2022. Dieses Datum muss nur durch einen Parameter ersetzt werden, wie Sie hier sehen:

```
reference_date = '2022-02-28'
sql_createview = df_rem_360.select_statement
sql_createview = sql_createview \
     .replace("'" + reference_date + "'", ":P_REFERENCEDATE")
sql_createview = "CREATE VIEW V_FIRSTPURCHASEIN30DAYS \
     (IN P_REFERENCEDATE NVARCHAR(10)) AS " + sql_createview + ";"
```

Führen Sie dieses angepasste SQL-Statement aus. Sie können hierfür eine beliebige Applikation wie den SAP HANA Database Explorer nutzen oder aber einfach direkt aus Python heraus den Code ausführen. Das hier genutzte `hdbcli`-Paket wurde bei der Installation des `hana_ml`-Pakets mit installiert und hilft in diesem Fall, beliebigen SQL-Code auf SAP HANA auszuführen.

```
from hdbcli import dbapi
dbapi_cursor = conn.connection.cursor()
dbapi_cursor.execute(sql_createview)
```

Der View ist angelegt. Dieser kann nun über Python oder aus anderen Umgebungen heraus angesprochen werden, und das gewünschte Datum wird als Parameter übergeben. Zum Trainieren des Machine-Learning-Modells nutzen Sie das Datum von vor 30 Tagen.

```
df_rem_train = conn \
    .sql("SELECT * FROM V_FIRSTPURCHASEIN30DAYS('2022-02-28')")
df_rem_train.describe().collect()
```

Lassen Sie einige Zeilen mit dem folgenden Code anzeigen, das Ergebnis sehen Sie in Abbildung 4.25:

```
df_rem_train.head(5).collect()
```

| CUSTOMERID | GENDER | AGE | FUTUREPURCHASE | WEBSITE_VISITS_30DAYS | WEBSITE_SEARCHES_30DAYS |
|---|---|---|---|---|---|
| CXID000001 | Female | 62 | No | 1 | 0 |
| CXID000002 | Female | 45 | No | 0 | 0 |
| CXID000003 | Unknown | 35 | No | 1 | 0 |
| CXID000004 | Female | 45 | No | 0 | 0 |
| CXID000005 | Male | 57 | No | 0 | 0 |

**Abbildung 4.25** Trainingsdaten

**Weitere Datenvorbereitungen**

Es gibt viele weitere Möglichkeiten der Datenvorbereitung. Einige weitere Beispiele hierzu finden Sie in Kapitel 9, »Tipps und Tricks«.

Nachdem wir die Daten erfolgreich vorbereitet haben, möchten wir sie in einem konkreten Beispiel nutzen. Das Erstellen einer hier notwendigen binären Klassifizierung zum Schätzen der Kaufwahrscheinlichkeit einer Person wird zwar erst in Abschnitt 5.2, »Klassifizierung mit der APL«, detailliert erläutert, dennoch möchten wir kurz zeigen, wie die beiden Sichten der Daten zum Trainieren und Anwenden eines

Machine-Learning-Modells genutzt werden können. Spätestens, wenn Sie sich mit der Klassifizierung in Abschnitt 5.2, »Klassifizierung mit der APL«, vertraut gemacht haben, werden Sie die folgenden Schritte selbst vornehmen können.

Im ersten Schritt trainieren wir ein binäres Klassifizierungsmodell mit der Automated Predictive Library (APL). Führen Sie dafür den folgenden Code aus:

```
col_target = 'FUTUREPURCHASE'
col_id = 'CUSTOMERID'
from hana_ml.algorithms.apl.gradient_boosting_classification \
import GradientBoostingBinaryClassifier
classapl = GradientBoostingBinaryClassifier()
classapl.fit(data=df_rem_train.sort(col_id, desc=False),
               key=col_id,
               label=col_target)
```

Lassen Sie sich nun die Eigenschaften des trainierten Modells anzeigen:

```
from hana_ml.visualizers.unified_report import UnifiedReport
UnifiedReport(classapl).build().display()
```

In Abbildung 4.26 sehen Sie das trainierte ML-Modell im Model Report.

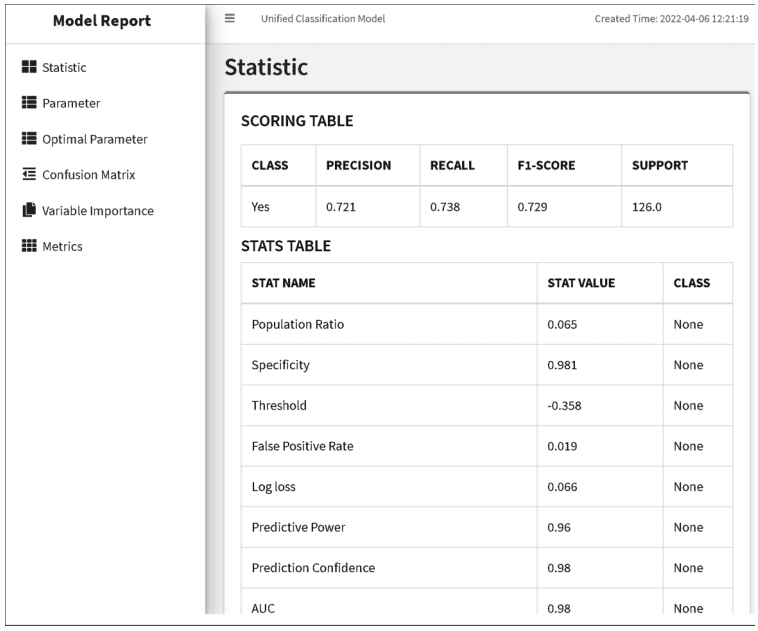


**Abbildung 4.26** Trainiertes Machine-Learning-Modell

Vorhersagen erstellen Sie auf den Datenbestand aus heutiger Sicht, mit dem heutigen Zeitstempel des 30. März 2022. Das trainierte Modell können Sie nun auf den ak-

tuellen Datensatz anwenden, der die Kontakte beinhaltet, die bis heute noch kein Produkt erworben haben, und die Informationen dazu, wie sich diese Personen in der Vergangenheit, aus heutiger Sicht, verhalten haben.

Wir verwenden hierfür denselben View. Nur die Zielvariable FUTUREPURCHASE sollte herausgenommen werden, die für alle Kontakte ein No zeigt, da diese das Verhalten in der Zukunft mit einbezieht, das wir letztlich mit dem Modell vorhersagen möchten. Es entsteht der folgende Code:

```
df_rem_apply = conn \
     .sql("SELECT * FROM V_FIRSTPURCHASEIN30DAYS('2022-03-30')") \
     .drop('FUTUREPURCHASE')
```

Wenden Sie das Modell auf den Datensatz an und erstellen Sie die Vorhersagen:

```
classapl.set_params(extra_applyout_settings =
           {'APL/ApplyExtraMode': 'Advanced Apply Settings',
            'APL/ApplyDecision': 'true',
            'APL/ApplyProbability': 'true',
            'APL/ApplyPredictedValue': 'false'
             })
df_rem_pred = classapl.predict(df_rem_apply)
```

Nun können Sie mit diesem Code die Kontakte mit der höchsten Kaufwahrscheinlichkeit anzeigen lassen:

```
df_rem_pred.sort('gb_proba_FUTUREPURCHASE', desc=True) \
           .head(5).collect()
```

Das Ergebnis wird wieder in einer Tabelle ausgegeben, die Sie in Abbildung 4.27 sehen. Hier zeigt sich, dass die Kundin oder der Kunde mit der **CustomerID CXID001858** die höchste Kaufwahrscheinlichkeit aufweist.

| CUSTOMERID | PREDICTED | gb_proba_FUTUREPURCHASE |
|---|---|---|
| CXID001858 | Yes | 0.952355 |
| CXID005369 | Yes | 0.830857 |
| CXID002911 | Yes | 0.830857 |
| CXID004988 | Yes | 0.830857 |
| CXID003822 | Yes | 0.764115 |

**Abbildung 4.27** Vorhersagen erstellt auf den transformierten Daten

Bereits mit vergleichsweise wenig Aufwand haben wir den vorhandenen Daten wichtige Informationen entnehmen können. Diese Kaufwahrscheinlichkeiten können nun zur Personalisierung der Kundenkommunikation genutzt werden.