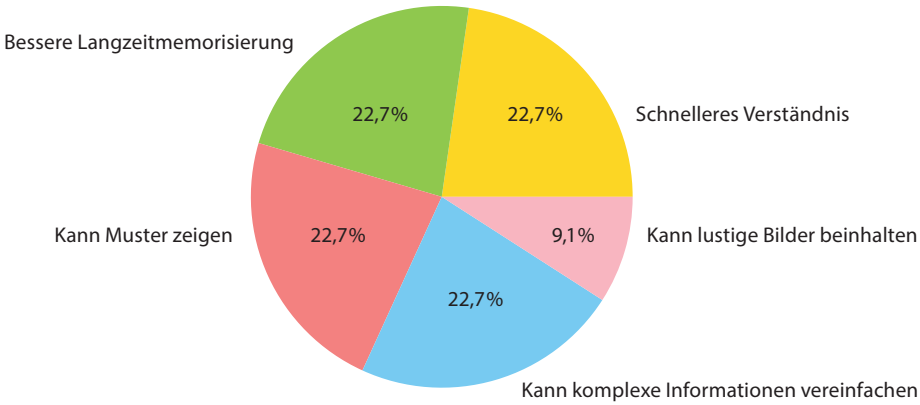
4.3 Visualisierung

Visualisierungen sind ein sehr mächtiges Werkzeug, um etwas über Daten zu lernen und diese zu verstehen. Hier sehen Sie ein Beispiel von einem Tortendiagramm (die Prozentzahlen sind ausgedacht und haben keine wissenschaftliche Bedeutung).

```
# we use matplotlib for the creation of visualizations
import matplotlib.pyplot as plt
# this is only required for Jupyter notebooks
%matplotlib inline

# data to plot
labels = ['Schnelleres Verständnis', 'Bessere Langzeitmemorierung',
          'Kann Muster zeigen', 'Kann komplexe Informationen vereinfachen',
          'Kann lustige Bilder beinhalten']
sizes = [25, 25, 25, 25, 10]
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'pink']

# plot
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%')
plt.show()
```



Es gibt viele Vorteile von Visualisierungen gegenüber anderen Arten, Daten darzustellen, zum Beispiel Tabellen oder deskriptive Statistik. Im Allgemeinen verarbeitet man visuelle Informationen schneller. Hierdurch kann man in der Regel schneller etwas über die Daten lernen. Hinzu kommt, dass viele Menschen sich visualisierte Informationen besser merken können.

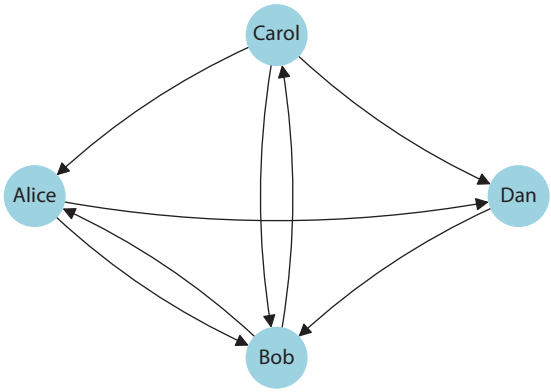
Die Vorteile von Visualisierungen gehen jedoch über die Verarbeitungsgeschwindigkeit und die Memorisierung hinaus. Visualisierungen können auch ein Verständnis der Daten ermöglichen, das man andernfalls nicht erlangen würde, zum Beispiel über Muster in Daten und komplexe Zusammenhänge, die man sonst nicht sehen könnte. Hierfür betrachten wir das folgende Beispiel. Alice kennt Bob und Dan, Dan kennt Bob, Bob kennt Carol und Alice, und Carol kennt Alice, Bob und

Dan. Diese textuelle Beschreibung ist sehr komplex und schwer zu verstehen. Außerdem bekommt man kein intuitives Verständnis der Beziehung zwischen Alice, Bob, Carol und Dan. Jetzt betrachten wir das Gleiche als gerichteten Graphen:

```
import networkx as nx  # networkx is a powerful library for working with graphs

# Create the graph by adding edges
# The vertices are implicitly defined through the endpoints of the edges
graph = nx.DiGraph()
graph.add_edge('Alice', 'Bob')
graph.add_edge('Alice', 'Dan')
graph.add_edge('Dan', 'Bob')
graph.add_edge('Bob', 'Carol')
graph.add_edge('Bob', 'Alice')
graph.add_edge('Carol', 'Alice')
graph.add_edge('Carol', 'Bob')
graph.add_edge('Carol', 'Dan')

# Plot the graph with a shell layout
nx.draw_shell(graph, with_labels=True, node_size=2000, node_color='lightblue',
              arrowsize=20, connectionstyle='arc3, rad = 0.1')
```



Dieser Graph ist einfach zu lesen und gibt uns ein intuitives Verständnis der Beziehungen.

Bitte beachten Sie, dass wir bei allen Grafiken in diesem Kapitel Legenden und andere Details bewusst weglassen. Stattdessen generieren wir die Grafiken mit möglichst wenig Quelltext, sodass sie trotzdem die gewünschten Erkenntnisse liefern. Der Grund hierfür ist, dass wir hier Visualisierungen als Werkzeug zur Erkundung von Daten betrachten. Wenn man ansonsten Visualisierungen für Texte (Bücher und sonstige Publikationen) oder Präsentationen erstellt, sind andere Aspekte ebenfalls relevant, zum Beispiel die konsistente Beschriftung, Farbwahl, Legenden und Titel.

4.3.1 Anscombes Quartett

Anscombes Quartett ist ein berühmtes Beispiel für Daten, in dem deskriptive Statistiken irreführend sind. Das Beispiel basiert auf vier Datensätzen $(x_1,y_1),...,(x_4,y_4)$ mit je elf Paaren aus zwei Variablen x_i und y_i, $i=1,...,4$. Tabelle 4–1 zeigt die Werte für jedes der Paare.

$x_{1,2,3}$	y_1	y_2	y_3	x_4	y_4
10	8,04	9,14	7,46	8	6,58
8	6,95	8,14	6,77	8	5,76
13	7,58	8,74	12,74	8	7,71
9	8,81	8,77	7,11	8	8,84
11	8,33	9,26	7,81	8	8,47
14	9,96	8,10	8,84	8	7,04
6	7,24	6,13	6,08	8	5,25
4	4,26	3,10	5,39	19	12,50
12	10,84	9,13	8,15	8	5,56
7	4,82	7,26	6,42	8	7,91
5	5,68	4,74	5,73	8	6,89

Tab. 4–1 *Anscombes Quartett*

Wenn wir das arithmetische Mittel und die Standardabweichung betrachten, sehen wir, dass die Werte für alle x_i und für alle y_i gleich sind.

```
import numpy as np  # we now also need numpy, the commonly used numerics
                                                    library for Python

x = np.array([10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5]) # same for x1, x2, and x3
y1 = np.array([8.04, 6.95, 7.58, 8.81, 8.33, 9.96,
               7.24, 4.26, 10.84, 4.82, 5.68])
y2 = np.array([9.14, 8.14, 8.74, 8.77, 9.26,
               8.10, 6.13, 3.10, 9.13, 7.26, 4.74])
y3 = np.array([7.46, 6.77, 12.74, 7.11, 7.81,
               8.84, 6.08, 5.39, 8.15, 6.42, 5.73])
x4 = np.array([8, 8, 8, 8, 8, 8, 8, 19, 8, 8, 8])
y4 = np.array[(6.58, 5.76, 7.71, 8.84, 8.47, 7.04,
               5.25, 12.50, 5.56, 7.91, 6.89])

print('Arithmetisches Mittel')
print('x1, x2, x3 = ', statistics.mean(x))
print('x4         = ', statistics.mean(x4))
print('y1         = ', statistics.mean(y1))
print('y1         = ', statistics.mean(y2))
print('y1         = ', statistics.mean(y3))
print('y1         = ', statistics.mean(y4))
↓
```

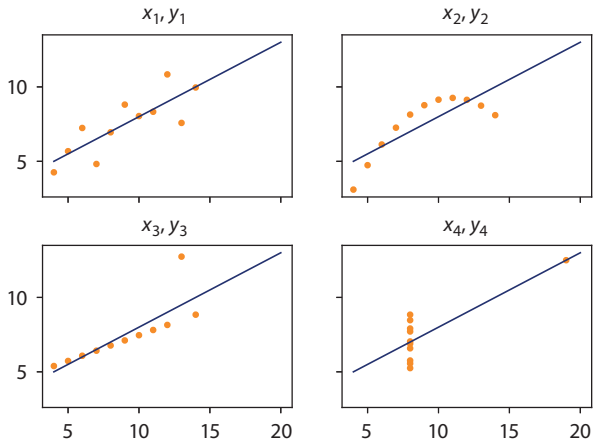
```
print('Standardabweichung')
print('x1, x2, x3 = ', statistics.stdev(x))
print('x4         = ', statistics.stdev(x4))
print('y1         = ', statistics.stdev(y1))
print('y1         = ', statistics.stdev(y2))
print('y1         = ', statistics.stdev(y3))
print('y1         = ', statistics.stdev(y4))
```

```
Arithmetisches Mittel
x1, x2, x3 =  9
x4         =  9
y1         =  7.500909090909091
y1         =  7.500909090909091
y1         =  7.5
y1         =  7.500909090909091
Standardabweichung
x1, x2, x3 =  3.3166247903554
x4         =  3.3166247903554
y1         =  2.031568135925815
y1         =  2.0316567355016177
y1         =  2.030423601123667
y1         =  2.0305785113876023
```

Es gibt sogar noch mehr Gemeinsamkeiten. Wenn wir eine lineare Regression von y_i durch x_i bestimmen würden (siehe Kap. 8), würden wir jedes Mal die gleiche Regressionsgerade $y = 3 + 0{,}5 \cdot x$ finden. Statistisch sind sich diese vier Datensätze also sehr ähnlich. Wenn wir uns die Daten mit einem einfachen *Scatterplot* visualisieren, sehen wir, dass die Datensätze eigentlich sehr unterschiedlich sind.

```
xfit = np.array([4, 20])
yfit = 3+0.5*xfit

f, axes = plt.subplots(2, 2, sharey=True, sharex=True)
axes[0, 0].plot(x, y1, color='darkorange', marker='.', linestyle='none')
axes[0, 0].plot(xfit, yfit, color='navy', lw=1)
axes[0, 0].set_title('$x_1, y_1$')
axes[0, 1].plot(x, y2, color='darkorange', marker='.', linestyle='none')
axes[0, 1].plot(xfit, yfit, color='navy', lw=1)
axes[0, 1].set_title('$x_2, y_2$')
axes[1, 0].plot(x, y3, color='darkorange', marker='.', linestyle='none')
axes[1, 0].plot(xfit, yfit, color='navy', lw=1)
axes[1, 0].set_title('$x_3, y_3$')
axes[1, 1].plot(x4, y4, color='darkorange', marker='.', linestyle='none')
axes[1, 1].plot(xfit, yfit, color='navy', lw=1)
axes[1, 1].set_title('$x_4, y_4$')
plt.show()
```



Die orangen Punkte visualisieren die Daten selbst, die blauen Linien zeigen die Regressionsgerade für $y = 3 + 0{,}5 \cdot x$, die optimal ist für die Daten. Nur auf Basis der statistischen Informationen über die Daten würde man erwarten, dass die Daten ungefähr so wie beim Paar x_1, y_1 aussehen: Die Daten haben in etwa eine lineare Beziehung mit einer leichten Streuung ohne ein klar erkennbares Muster um die Regressionsgerade. Eine lineare Beziehung zwischen x und y bedeutet, dass wenn x sich verändert, sich y um ein konstantes Vielfaches von x verändert. Das bedeutet umgekehrt auch, dass man die Beziehung von x und y sich etwa als Gerade vorstellen kann. Beim Paar x_2, y_2 ist dies nicht der Fall, stattdessen sehen die Daten eher aus wie eine auf den Kopf gestellte Parabel. Das Paar x_3, y_3 ist eigentlich perfekt linear, bis auf einen einzelnen Datenpunkt, der nach oben ausreißt und dafür sorgt, dass die blaue Regressionsgerade nicht zur eigentlichen Geraden, auf der die Daten liegen, passt. Das Paar x_4, y_4 passt nicht zu dem, was die Statistiken aussagen, die auch wieder von einem Ausreißer stark beeinflusst werden.

Die Aussage von Anscombes Quartett ist somit eindeutig: Auch wenn Statistiken gut geeignet sein können, Daten zusammenzufassen, ist es ebenso möglich, dass die Statistiken irreführend sind. Die kritische Leserin oder der aufmerksame Leser wird vielleicht bemerkt haben, dass die durch Ausreißer verursachten Probleme beim arithmetischen Mittel und der Standardabweichung zu erwarten sind. Die Probleme von Statistiken sind jedoch grundlegender und es gibt weitere Beispiele, die auch mehr statistische Marker berücksichtigen [Matejka & Fitzmaurice 2017].

4.3.2 Einzelne Merkmale

Eine grundlegende Betrachtung der Daten besteht in der Visualisierung einzelner Merkmale der Daten. Hierdurch kann man die Verteilung dieser Merkmale verstehen, ähnlich zur Beschreibung durch Statistiken. Hierzu schauen wir uns Histogramme, Densityplots, Rugs und Boxplots an.

Wir betrachten dieses Feature anhand von Daten über Hauspreise aus Boston, die 1978 veröffentlicht wurden, die wir im Folgenden einfach als Bostondaten bezeichnen werden.

```
# sklearn is a large machine learning library that we use
from sklearn import datasets
from textwrap import TextWrapper

# we use this wrapper to avoid printing lines that are too long because this
# should be readable as a book
# usually you would still just use print
wrapper = TextWrapper(width=65, replace_whitespace=False, break_long_words=False)
def wrap_print(string):
    for line in string.split('\n'):
        print('\n'.join(wrapper.wrap(line)))

boston = datasets.fetch_openml(data_id=531)
wrap_print(boston.DESCR)
```

```
**Author**:
**Source**: Unknown - Date unknown
**Please cite**:

The Boston house-price data of Harrison, D. and Rubinfeld, D.L.
'Hedonic
prices and the demand for clean air', J. Environ. Economics &
Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression
diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the
table on
pages 244-261 of the latter.
Variables in order:
CRIM     per capita crime rate by town
ZN       proportion of residential land zoned for lots over
25,000 sq.ft.
INDUS    proportion of non-retail business acres per town
CHAS     Charles River dummy variable (= 1 if tract bounds river;
0 otherwise)
NOX      nitric oxides concentration (parts per 10 million)
RM       average number of rooms per dwelling
AGE      proportion of owner-occupied units built prior to 1940
DIS      weighted distances to five Boston employment centres
RAD      index of accessibility to radial highways
TAX      full-value property-tax rate per $10,000
PTRATIO  pupil-teacher ratio by town
B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks
by town
LSTAT    % lower status of the population
MEDV     Median value of owner-occupied homes in $1000's

Information about the dataset
CLASSTYPE: numeric
CLASSINDEX: last

Downloaded from openml.org.
```

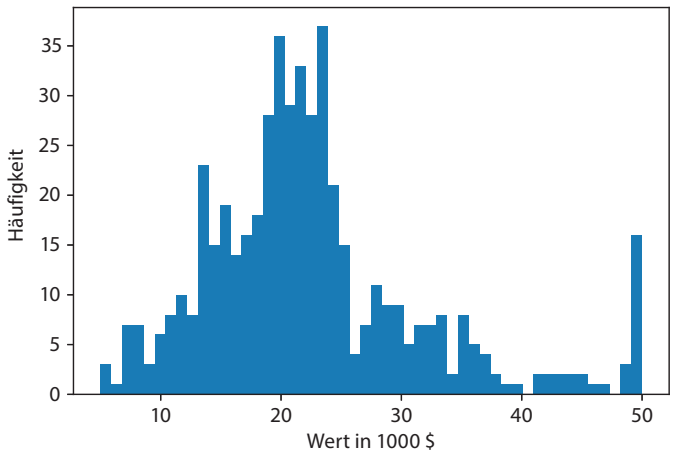
Die obige Beschreibung ist ein Beispiel für Metadaten, in diesem Fall die Dokumentation der Daten.

Bemerkung:

Die Bostondaten werden immer häufiger kritisiert und wurden zum Beispiel auch als Beispieldatensatz aus scikit-learn entfernt. Der Grund hierfür ist, dass die Daten aus ethischer Sicht hochproblematisch sind: Die Daten sind teilweise rassistisch, insbesondere das Merkmal `B`. In diesem Buch verwenden wir diese Daten dennoch weiter und nutzen den Datensatz, um nicht nur zu zeigen, wie man Daten analysieren kann, sondern auch als Beispiel, dass man ethische Aspekte von Daten und Modellen niemals vergessen sollte.

Wir werfen jetzt einen genaueren Blick auf das Merkmal `MEDV`, den Median des Werts der Eigenheime in 1000 Dollar. *Histogramme* sind eine einfache und oft effektive Art, etwas über die Verteilung von Daten zu lernen.

```
fig, ax = plt.subplots()
ax.hist(boston.target, bins=50)
ax.set_xlabel('Wert in 1000 $')
ax.set_ylabel('Häufigkeit')
plt.show()
```



Das Histogramm zeigt, wie häufig Werte vorkommen. Hierfür wird der Wertebereich in sogenannte *Bins* unterteilt. Der Begriff *Bin* ist vom englischen Wort für Gruppieren abgeleitet. Das Histogramm gibt an, wie viele Datenpunkte in jedem Bin liegen. Im obigen Beispiel haben wir 50 Bins. Das Histogramm verrät uns viel über die Daten.

- Die Daten zwischen 0 und 30 scheinen normalverteilt zu sein. Dies erkennt man daran, dass die Daten ungefähr eine *Glockenform* (engl. *bell-shape*) bilden, was für die Normalverteilung typisch ist. Der Mittelwert dieser Normalverteilung

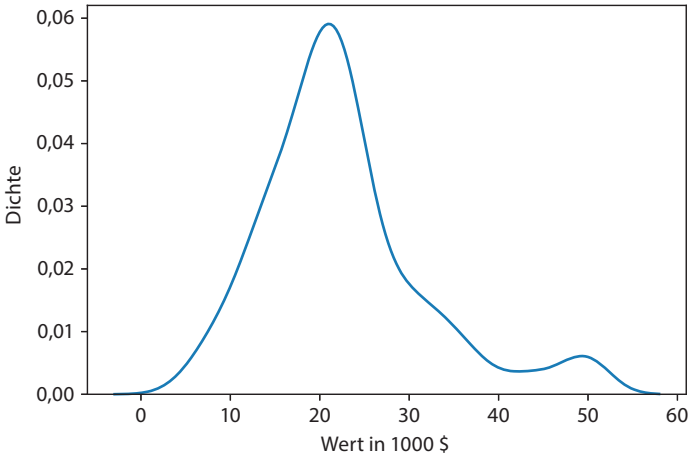
liegt ca. bei 22. Dies erkennt man am *Peak* in diesem Wertebereich. Die Standardabweichung kann man nicht genau ablesen, aber mithilfe der oben diskutierten 68-95-99-Regel grob abschätzen. Der Wert sollte sich zwischen 7 und 11 befinden.

- Neben dieser Normalverteilung gibt es noch viele Werte am rechten Rand der Grafik, also bei genau 50. Dies deutet darauf hin, dass es in Wirklichkeit vermutlich auch noch Werte oberhalb von 50 gibt, diese aber zusammengefasst wurden. Die eigentliche Bedeutung des Werts 50 sind somit nicht Häuser mit einem Wert von 50.000 Dollar, sondern Häuser, die mindestens 50.000 Dollar wert sind.
- Es gibt einen *Tail* auf der rechten Seite des Grafik, also Häuser, die teuer sind. Diese Häuser kommen zwar vor, lassen sich aber nicht mehr durch die Normalverteilung erklären. Solche Daten nennt man auch *Rechtsschief* (engl. *right skew*).

Wir können uns `MEDV` auch mithilfe von einem Densityplot anschauen.

```
import seaborn as sns  # seaborn is a visualization library build on top of
matplotlib

fig, ax = plt.subplots()
sns.kdeplot(boston.target, ax=ax)
ax.set_xlabel('Wert in 1000 $')
ax.set_ylabel('Dichte')
plt.show()
```



Vereinfacht gesagt, zeigen Densityplots eine Schätzung der Dichtefunktion der Wahrscheinlichkeitsverteilung der Daten. Wir können uns das als eine Art kontinuierliches Histogramm vorstellen. Der Vorteil von Densityplots gegenüber Histogrammen ist, dass es oft einfacher ist, die Verteilung der Daten zu erkennen. Im obigen Beispiel treten die Glockenform und die Position des Peaks deutlicher hervor und man kann dadurch die Normalverteilung in der linken Hälfte des Plots besser er-

kennen. Densityplots haben jedoch auch einige Nachteile im Vergleich zu Histogrammen. Ein kleiner Nachteil ist, dass es schwierig ist, die Werte der y-Achse zu interpretieren. Während ein Histogramm eine klare Aussage über die Anzahl der Datenpunkte in einem Bin erlaubt, sieht man im Densityplot lediglich die *Dichte* als Wert zwischen 0,0 und 1,0. Die Dichte ist mehr oder weniger der Anteil der Daten, der an einem bestimmten Punkt auf der x-Achse zu erwarten ist.

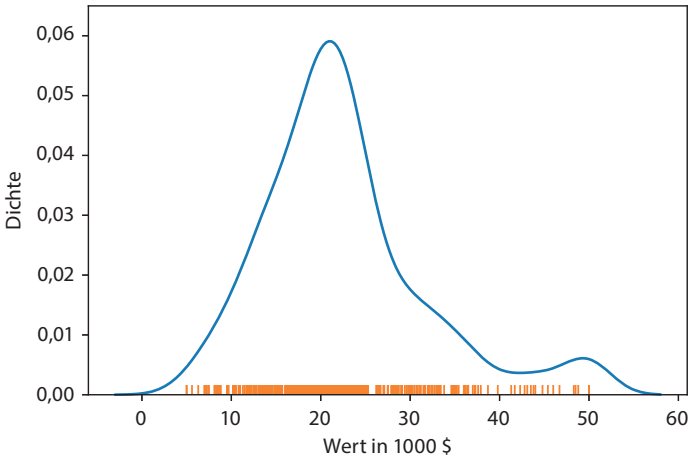
Der zweite Nachteil der Densityplots ist gravierender. Das Histogramm zeigt uns klar die vielen Datenpunkte mit dem Wert 50 und auch, dass es keine Datenpunkte mit einem höheren Wert gibt. Im Densityplot sieht man bei der 50 nur einen kleinen Peak, den man auch als weitere Normalverteilung mit dem Mittelwert 50 am rechten Rand der Daten interpretieren könnte. Das ist irreführend und versteckt die wahre Verteilung der Daten. Dieses Risiko lässt sich bei Densityplots auch nicht vermeiden, da es mit der Technik, wie die Dichte geschätzt wird, zusammenhängt. Solche Fehlinterpretationen der Daten sind dann wahrscheinlich, wenn die Daten nicht sehr dicht verteilt sind (großer Datenbereich mit verhältnismäßig wenig Datenpunkten), sowie an den Grenzen der beobachteten Daten. Hier geht die geschätzte Dichtefunktion automatisch über den beobachteten Datenbereich hinaus, was eventuell aber der Interpretation eines Merkmals widerspricht. Man erkennt im obigen Plot zum Beispiel auch, dass die Werte kleiner 0 nicht eine Dichte von 0 haben, was bedeuten würde, dass einige Eigenheime einen negativen Wert hätten.

Bemerkung:

Densityplots werden mithilfe von *Kernel Density Functions* (KDE) erstellt, in der Regel mittels der Dichtefunktion der Normalverteilung. Diese Dichtefunktion wird dann an jedem Datenpunkt mit *Skalierungs-* und *Bandbreitenparametern* geschätzt. Anschließend werden alle diese geschätzten Dichtefunktionen aufaddiert, um den Densityplot zu erstellen. Hierdurch kann man auch das Verhalten an den Grenzen erklären: Die Dichtefunktion, die an den äußersten Punkten der Daten geschätzt wird, geht aufgrund der Symmetrie der Normalverteilung automatisch über den Datenbereich hinaus. Die Skalierung der Schätzung führt dazu, dass ein seltsames Verhalten, wie viele gleiche Werte an einer Grenze in der aufsummierten Dichtefunktion, nicht stark auffällt.

Ein einfaches Mittel, dieses Problem zu vermeiden, besteht darin, einen *Rug* (dt. Teppich) anzuzeigen. Der *Rug* hat seinem Namen daher, weil er wie ein Teppich unter den Plot gelegt wird.

```
fig, ax = plt.subplots()
sns.kdeplot(boston.target, ax=ax)
sns.rugplot(boston.target, ax=ax)
ax.set_xlabel('Wert in 1000 $')
ax.set_ylabel('Dichte')
plt.show()
```



Der Rug zeigt, wo sich wirklich Datenpunkte befinden. In Kombination mit dem Densityplot können wir also sehen, dass es keine Datenpunkte kleiner als 5 oder größer 50 gibt. Wir erkennen auch, dass die Region zwischen 38 und 50 nur relativ dünn besiedelt ist, wobei die Punkte in diesem Bereich etwa gleichverteilt zu sein scheinen. Hierzu passt, dass der Densityplot in diesem Bereich fast parallel zur x-Achse verläuft. Dies zeigt, dass der Rug hilfreich ist, um weitere Erkenntnisse über die Daten zu gewinnen und Fehlinterpretationen zu vermeiden.

Man kann auch einfach alle oben betrachteten Ansätze kombinieren und Histogramm, Densityplot und Rug zusammen visualisieren.

```
fig, ax = plt.subplots()
# stat='density' scales the plot to match the y-axis of the density plot
# alpha=0.2 makes the bars transparent for better readability
sns.histplot(boston.target, stat='density', alpha=0.2, bins=50, ax=ax)
sns.kdeplot(boston.target, ax=ax)
sns.rugplot(boston.target, ax=ax)
ax.set_xlabel('Wert in 1000 $')
ax.set_ylabel('Dichte')
plt.show()
```