In [1]:

```
import pandas as pd
import numpy as np
import os
```

Silber Schicht

In [2]:

```
silber_df = pd.read_csv('bronze/auto_bronze.csv')
```

Umwandlung von "?" in NaN

In [3]:

```
silber_df.replace("?", np.nan, inplace = True)
```

In [4]:

```
# Fehlende Werte in jeder Spalte zählen
missing_data = silber_df.isnull().sum()
missing_data.sort_values(inplace=True, ascending=False)
display(missing_data)
```

```
normalisierter-verlustwert    41
preis                          4
anschlag                       4
bohrung                        4
spitzendrehzahl                2
türnummern                     2
pferdestärken                  2
motortyp                       0
autobahn-mpg                   0
stadt-mpg                      0
verdichtungsverhältnis         0
kraftstoffsystem               0
motorgröße                     0
anzahl-der-zylinder            0
risikoniveau                   0
höhe                           0
breite                         0
länge                          0
radstand                       0
motorstandort                  0
antriebsräder                  0
körperform                     0
absaugung                      0
kraftstofftyp                  0
marke                          0
leergewicht                    0
dtype: int64
```

Wie geht man mit fehlenden Daten um?

Daten löschen a. die gesamte Zeile löschen b. die gesamte Spalte löschen Daten ersetzen a. durch Mittelwert ersetzen b. Ersetzen durch Häufigkeit c. Ersetzen auf der Grundlage anderer Funktionen

Ersetzen durch Mittelwert:

"normalisierte-Verluste": 41 fehlende Daten, durch Mittelwert ersetzen "Schlaganfall": 4 fehlende Daten, durch Mittelwert ersetzen "Bohrung": 4 fehlende Daten, ersetzen Sie diese durch den Mittelwert "Pferdestärken": 2 fehlende Daten, durch Mittelwert ersetzen "Spitzen-Drehzahl": 2 fehlende Daten, ersetze sie durch den Mittelwert

Ersetzen durch Häufigkeit:

"Anzahl der Türen": 2 fehlende Daten, ersetze sie durch "vier". Grund: 84% der Limousinen sind viertürig. Da vier Türen am häufigsten vorkommen, ist es am wahrscheinlichsten, dass sie vorkommen.

Streiche die ganze Zeile:

"Preis": 4 fehlende Daten, einfach die ganze Zeile löschen Grund: Der Preis ist das, was wir vorhersagen wollen. Jeder Dateneintrag ohne Preisdaten kann nicht für die Vorhersage verwendet werden; daher ist jede Zeile ohne Preisdaten für uns nicht nützlich

In [5]:

```
# Ersetzen durch Mittelwert
avg_normalisierter_verlustwert = silber_df['normalisierter-verlustwert'].astype("float")
.mean(axis=0)
silber_df.fillna({'normalisierter-verlustwert': avg_normalisierter_verlustwert}, inplace
=True)

avg_bohrung = silber_df['bohrung'].astype('float').mean(axis=0)
silber_df.fillna({'bohrung': avg_bohrung}, inplace=True)

avg_anschlag = silber_df["anschlag"].astype("float").mean(axis = 0)
silber_df.fillna({'anschlag': avg_anschlag}, inplace=True)

avg_pferdestaerken = silber_df['pferdestärken'].astype('float').mean(axis=0)
silber_df.fillna({'pferdestärken': avg_pferdestaerken}, inplace=True)

avg_spitzendrehzahl = silber_df['spitzendrehzahl'].astype('float').mean(axis=0)
silber_df.fillna({'spitzendrehzahl': avg_spitzendrehzahl}, inplace=True)
```

In [6]:

```
# Ersetzen durch Häufigkeit:

max_tuernummern = silber_df['türnummern'].value_counts().idxmax()
print(max_tuernummern)
silber_df['türnummern'].replace(np.nan, max_tuernummern, inplace=True)
```

```
four
```

In [7]:

```
# Streiche die ganze Zeile
silber_df.dropna(subset=['preis'], axis=0, inplace=True)
silber_df.reset_index(drop=True, inplace=True)
```

Korrektur des Datenformats

In [8]:

```
display(silber_df.dtypes)
```

```
risikoniveau                   int64
normalisierter-verlustwert    object
marke                         object
kraftstofftyp                 object
absaugung                     object
türnummern                    object
körperform                    object
antriebsräder                 object
motorstandort                 object
radstand                     float64
länge                        float64
breite                       float64
höhe                         float64
leergewicht                    int64
motortyp                      object
anzahl-der-zylinder           object
```

```
motorgröße                 int64
kraftstoffsystem          object
bohrung                   object
anschlag                  object
verdichtungsverhältnis   float64
pferdestärken             object
spitzendrehzahl           object
stadt-mpg                  int64
autobahn-mpg               int64
preis                     object
dtype: object
```

Wie wir oben sehen können, haben einige Spalten nicht den richtigen Datentyp. Numerische Variablen sollten den Typ "float" oder "int" haben, und Variablen mit Zeichenketten wie Kategorien sollten den Typ "object" haben. Bei den Variablen "Bohrung" und "Hub" handelt es sich beispielsweise um numerische Werte, die die Motoren beschreiben, so dass man erwarten sollte, dass sie vom Typ "float" oder "int" sind; sie werden jedoch als Typ "object" angezeigt.

In [9]:

```
silber_df[['bohrung', 'anschlag']] = silber_df[['bohrung', 'anschlag']].astype("float")
silber_df['normalisierter-verlustwert'] = silber_df['normalisierter-verlustwert'].astype
("int64")
silber_df['preis'] = silber_df['preis'].astype("float")
silber_df['spitzendrehzahl'] = silber_df['spitzendrehzahl'].astype("float")
silber_df['pferdestärken'] = silber_df['pferdestärken'].astype("int64", copy=True)
```

Data Standardization

Umwandlung von mpg in L/100km:

In unserem Datensatz werden die Kraftstoffverbrauchsspalten "Stadt-mpg" und "Autobahn-mpg" in der Einheit mpg (miles per gallon) dargestellt. Angenommen, wir entwickeln eine Anwendung in einem Land, in dem der Kraftstoffverbrauch in L/100km angegeben wird.

Die Formel für die Einheitenumrechnung lautet

L/100km = 235 / mpg

In [10]:

```
silber_df['stadt-L/100km'] = 235/silber_df['stadt-mpg']
silber_df['autobahn-L/100km'] = 235/silber_df['autobahn-mpg']
```

Data Normalization

Unter Normalisierung versteht man den Prozess der Umwandlung von Werten mehrerer Variablen in einen ähnlichen Bereich. Typische Normalisierungen umfassen die Skalierung der Variablen, so dass der Durchschnitt der Variablen 0 ist, die Skalierung der Variablen, so dass die Varianz 1 ist, oder die Skalierung der Variablen, so dass die Variablenwerte zwischen 0 und 1 liegen.

In [11]:

```
silber_df['länge-norm'] = silber_df['länge'] / silber_df['länge'].max()
silber_df['breite-norm'] = silber_df['breite'] / silber_df['breite'].max()
silber_df['höhe-norm'] = silber_df['höhe'] / silber_df['höhe'].max()

display(silber_df[['länge-norm','breite-norm','höhe-norm']].head())
```

	länge-norm	breite-norm	höhe-norm
0	0.811148	0.890278	0.816054
1	0.811148	0.890278	0.816054
2	0.822681	0.909722	0.876254

	länge-norm	breite-norm	höhe-norm
3	0.848630	0.919444	0.908027
4	0.848630	0.922222	0.908027

Binning

Binning ist ein Verfahren zur Umwandlung kontinuierlicher numerischer Variablen in diskrete kategorische "Bins" für eine gruppierte Analyse.

In [12]:

```
bins = np.linspace(min(silber_df['pferdestärken']), max(silber_df['pferdestärken']), 4)
gruppen_namen = ['niedrig', 'mittel', 'hoch']
silber_df['pferdestärken-binned'] = pd.cut(silber_df['pferdestärken'], bins, labels=grup
pen_namen, include_lowest=True )
# silber_df['pferdestärken-binned'] = pd.qcut(silber_df['pferdestärken'], 3, labels=['nie
drig', 'mittel', 'hoch'])
display(silber_df[['pferdestärken','pferdestärken-binned']].head(20))
```

	pferdestärken	pferdestärken-binned
0	111	niedrig
1	111	niedrig
2	154	mittel
3	102	niedrig
4	115	niedrig
5	110	niedrig
6	110	niedrig
7	110	niedrig
8	140	mittel
9	101	niedrig
10	101	niedrig
11	121	mittel
12	121	mittel
13	121	mittel
14	182	mittel
15	182	mittel
16	182	mittel
17	48	niedrig
18	70	niedrig
19	70	niedrig

Indikatorvariable (oder Dummy-Variable)

In [13]:

```
dummy = pd.get_dummies(silber_df['kraftstofftyp'])
silber_df = pd.concat([silber_df, dummy], axis=1)
silber_df.rename(columns={'gas': 'benzin'}, inplace=True)
```

export als Parquet

In [14]:

```
parquet_file_path = os.path.join('.', 'silber')
```

```
parquet_file_name = 'auto_silber.parquet'

if not os.path.exists(parquet_file_path):
    os.mkdir(parquet_file_path)

silber_df.to_parquet(os.path.join(parquet_file_path, parquet_file_name))
display(silber_df)
print(silber_df.columns)
```

	risikoniveau	normalisierter-verlustwert	marke	kraftstofftyp	absaugung	türnummern	körperform	antriebsräder	motorstandort	ra
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	
3	2	164	audi	gas	std	four	sedan	fwd	front	
4	2	164	audi	gas	std	four	sedan	4wd	front	
...	...	...	...	...	...	...	...	...	...	
196	-1	95	volvo	gas	std	four	sedan	rwd	front	
197	-1	95	volvo	gas	turbo	four	sedan	rwd	front	
198	-1	95	volvo	gas	std	four	sedan	rwd	front	
199	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	
200	-1	95	volvo	gas	turbo	four	sedan	rwd	front	

201 rows × 34 columns

```
Index(['risikoniveau', 'normalisierter-verlustwert', 'marke', 'kraftstofftyp',
       'absaugung', 'türnummern', 'körperform', 'antriebsräder',
       'motorstandort', 'radstand', 'länge', 'breite', 'höhe', 'leergewicht',
       'motortyp', 'anzahl-der-zylinder', 'motorgröße', 'kraftstoffsystem',
       'bohrung', 'anschlag', 'verdichtungsverhältnis', 'pferdestärken',
       'spitzendrehzahl', 'stadt-mpg', 'autobahn-mpg', 'preis',
       'stadt-L/100km', 'autobahn-L/100km', 'länge-norm', 'breite-norm',
       'höhe-norm', 'pferdestärken-binned', 'diesel', 'benzin'],
      dtype='object')
```