

# Real Estate Data Science Project

Bearbeitet von: 2341463 und 1269263

## Inhaltsverzeichnis

1. Business Understanding

---

2. Data Exploration und Analyse

---

3. Data Preparation

---

4. Modeling - Regression mit Inferenz

---

5. Modeling und Evaluation

---

6. Deployment

---

7. Classifikation

---

## 1. Business Understanding (3 Punkte)

**Aufgabenstellung:** Formulieren Sie ein Ziel oder mehrere Ziele nach dem CRISP-DM Prozess, die für Immobilienspekulant\*innen sinnvoll sind. Bei Spekulationen werden typischerweise Immobilien erstanden, die wieder mit Gewinn abgestoßen werden. Beginnen Sie mit der Idee „Wir brauchen mehr Verständnis des Verkaufspreises (Z\_Verkaufspreis)!“. Geben Sie Ihre Ziele in Ihrem Jupyter-Notebook als Markup an (max. 1/2 Seite). Wichtig ist hier, eigene zu untersuchende Hypothesen aufzustellen, die dann in Aufgabenteil 2 untersucht werden. Nutzen Sie auch die vorhandenen Daten, um die Hypothesen zu ergänzen oder anzupassen, wenn notwendig.

### Ziele der Immobilienspekulant\*innen

Als Immobilienspekulant\*innen bezeichnet man Personen, die eine Vielzahl von Immobilien erwerben und darauf hoffen, dass der Preis der Immobilien in Zukunft steigen wird. Die Häuser oder Wohnungen können dann mit Gewinn verkauft werden. Für die Kaufentscheidung und die einfache Bewertung der Attraktivität einer Immobilie, stehen für diese Personengruppe folgende Ziele und Anforderungen im Vordergrund:

1. Mehr Verständnis für den Verkaufspreis (Z\_Verkaufspreis) schaffen
2. Auswirkungen der einzelnen Parameter auf den Verkaufspreis untersuchen
3. Identifikation von Attributen, die sich kaum oder gar nicht auf den Verkaufspreis auswirken

4. Klassifikation der Attraktivität der Angebote anhand des angebotenen Verkaufspreises in die drei Kategorien "gut", "neutral" und "schlecht"

## Ziele des Business Understanding (Geschäftsverständnis)

- Was sind die Ziele auf Geschäftsebene?
- Welche Anforderungen an das Ergebnis gibt es?
- Welche offenen Fragen sollen beantwortet werden?
- Wie könnten beispielhafte Antworten oder Ergebnisse aussehen?

Ein Hauptziel auf dem Bereich der Geschäftsebene ist es, den Entscheidungsprozess für die Bewertung und den Kauf von Immobilien zu unterstützen. Durch die Analyse der Daten sollen Vorhersagen zur Attraktivität der Angebote und die Auswirkungen der einzelnen Attribute (Parameter) auf den veranschlagten Preis bestimmt werden. Das Ergebnis sollte auch für nicht DataScience-kundige Anwender\*innen verständlich und aussagekräftig gestaltet sein. Die zu beantwortenden Fragen wurden als Ziele für die Untersuchung definiert (siehe vorherigen Abschnitt). Die Ergebnisse lassen sich in zwei Bereiche einteilen:

- Für Angebote mit vorgegebenen (bekannten) Verkaufspreis soll eine automatische Klassifikation in die drei Gruppen "gutes Angebot", "neutral" und "schlechtes Angebot" erfolgen, damit die Immobilienspekulierenden eine vorab Einschätzung und eine damit verbundene Zeit- und Aufwandsersparung erhalten.
- Für Angebote mit unbekanntem Verkaufspreis soll anhand der Immobilien-Attribute eine Einschätzung und Vorhersage des Preises erfolgen.

## 2. Data Exploration und Analyse (9 Punkte)

**Aufgabenstellung:** Laden und untersuchen Sie den Datensatz in `data_for_training.csv` nach den Regeln wie in der Vorlesung gelehrt. Nutzen Sie Mark-Up, um wichtige Erkenntnisse zu dokumentieren.

### Module und Datensätze importieren

```
In [ ]: # Import modules and packages
import os
import numpy as np
import pandas as pd
import sklearn as sk
import seaborn as sns
import matplotlib.pyplot as plt

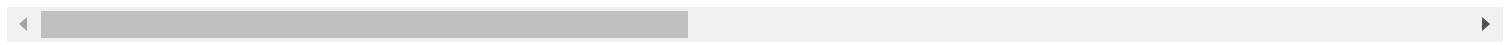
# Import training and test data
df_train = pd.read_csv("data_for_training.csv", delimiter=";").drop(columns="A_Index")
df_test = pd.read_csv("data_for_test.csv", delimiter=";").drop(columns="A_Index")

# Output training dataframe
df_train
```

Out[ ]:

	AnzahlZimmer	Ausbaustufe	Baeder	BaederKG	Baujahr	EG_qm	Garage_qm	Garagen	Gesamteindruck
<b>0</b>	3	1 Ebene	2	1	1992	125	49	2	3
<b>1</b>	2	1 Ebene	2	1	2010	170	79	3	3
<b>2</b>	2	1 Ebene	2	0	2015	119	40	2	3
<b>3</b>	2	2 Ebenen	3	1	2015	64	40	2	3
<b>4</b>	3	1 Ebene	2	0	2021	103	39	2	3
<b>...</b>	<b>...</b>	<b>...</b>	<b>...</b>	<b>...</b>	<b>...</b>	<b>...</b>	<b>...</b>	<b>...</b>	<b>...</b>
<b>2337</b>	3	1 Ebene	2	1	1989	109	40	2	4
<b>2338</b>	3	1 Ebene	2	1	1969	153	41	2	3
<b>2339</b>	3	2 Ebenen	3	0	1997	83	40	2	3
<b>2340</b>	3	2 Ebenen	2	0	1984	46	21	1	3
<b>2341</b>	3	2 Ebenen	3	1	2008	117	70	3	3

2342 rows × 20 columns



## Datentypen der einzelnen Attribute (Spalten) untersuchen

In [ ]:

```
# Output data types and non-null count of the individual columns
df_train.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2342 entries, 0 to 2341
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   AnzahlZimmer    2342 non-null    int64  
 1   Ausbaustufe     2342 non-null    object  
 2   Baeder          2342 non-null    int64  
 3   BaederKG        2342 non-null    int64  
 4   Baujahr         2342 non-null    int64  
 5   EG_qm           2342 non-null    int64  
 6   Garage_qm       2342 non-null    int64  
 7   Garagen         2342 non-null    int64  
 8   Gesamteindruck 2342 non-null    int64  
 9   Keller_Typ_qm  2342 non-null    int64  
 10  Keller_qm       2342 non-null    int64  
 11  Kellerhoehe     2342 non-null    object  
 12  Kellertyp       2342 non-null    object  
 13  Lage            2342 non-null    object  
 14  OG_qm           2342 non-null    int64  
 15  Umgebaut        2342 non-null    int64  
 16  Verkaufsjahr    2342 non-null    int64  
 17  Verkaufsmonat   2342 non-null    int64  
 18  Wohnflaeche_qm 2342 non-null    int64  
 19  Z_Verkaufspreis 2342 non-null    int64  
dtypes: int64(16), object(4)
memory usage: 366.1+ KB

```

```

In [ ]: # Show all individual entry names of the found object columns
for col in df_train.dtypes[df_train.dtypes == "object"].index:
    print(f"{col.rstrip()}: {df_train.loc[:, col].unique()}\n")

Ausbaustufe: ['1 Ebene' '2 Ebenen' '3 Ebenen']

Kellerhoehe: ['Gut' 'Durchschnitt' '0' 'Sehr gut' 'Schlecht' 'Sehr Schlecht']

Kellertyp: ['Guter Wohnraum' 'Rohbau' 'Mittlerer Wohnraum' 'Niedrige Qualität' '0'
 'Freizeitraum' 'Kein Wohnraum']

Lage: ['Bezirk 19' 'Bezirk 16' 'Bezirk 18' 'Bezirk 8' 'Bezirk 17' 'Bezirk 6'
 'Bezirk 23' 'Bezirk 9' 'Bezirk 15' 'Bezirk 20' 'Bezirk 14' 'Bezirk 1'
 'Bezirk 24' 'Bezirk 21' 'Bezirk 22' 'Bezirk 7' 'Bezirk 4' 'Bezirk 25'
 'Bezirk 5' 'Bezirk 26' 'Bezirk 27' 'Bezirk 12' 'Bezirk 2' 'Bezirk 13'
 'Bezirk 10' 'Bezirk 3' 'Bezirk 11' '0']

```

- Der Datensatz enthält 19 Features + eine Spalte für den Verkaufspreis der Immobilie. Im weiteren Verlauf der Aufgabe geht es darum herauszufinden, welche Features am relevantesten für die Vorhersage des Verkaufspreises sind.
- Der Datensatz enthält der ersten Prüfung nach keine Null-Werte. In der weiteren Data Exploration und in der Data Preparation wird auf anderweitig (z. B. kontextabhängige) ungültige Werte geprüft.
- Bis auf die Attribute "Ausbaustufe", "Kellerhoehe", "Kellertyp" und "Lage" sind alle Spalten bereits als numerische Werte (integer) gegeben. Die verbleibenden vier Attribute müssen in der Data Preparation (Aufgabenteil 3) encoded werden.

## Kriterien des Datensatzes erforschen

Über die describe() Funktion von pandas kann eine Übersicht mit Infos zu zentralen Tendenzen, Streuung und Verteilung des Datensatzes ausgegeben werden.

In [ ]: df\_train.describe()

	AnzahlZimmer	Baeder	BaederKG	Baujahr	EG_qm	Garage_qm	Garagen	Gesamtei
<b>count</b>	2342.000000	2342.000000	2342.000000	2342.000000	2342.000000	2342.000000	2342.000000	2342
<b>mean</b>	2.855252	1.906063	0.491460	1980.755337	95.856106	38.290777	1.707088	3
<b>std</b>	0.823598	0.806672	0.533111	29.626630	31.630087	17.389905	0.743206	0
<b>min</b>	0.000000	0.000000	0.000000	1884.000000	28.000000	0.000000	0.000000	1
<b>25%</b>	2.000000	1.000000	0.000000	1964.000000	73.000000	26.000000	1.000000	3
<b>50%</b>	3.000000	2.000000	0.000000	1982.000000	89.000000	39.000000	2.000000	3
<b>75%</b>	3.000000	2.000000	1.000000	2009.000000	114.000000	48.000000	2.000000	4
<b>max</b>	8.000000	6.000000	3.000000	2022.000000	324.000000	126.000000	5.000000	5

Die Untersuchung der Rohdaten liefert bereits erste Einblicke und mögliche Ansätze für die weitere Datenexploration. Für die spätere Vorhersage des Preises ist bereits interessant, dass die Spanne von 13.100 € bis 755.000 € sehr groß ist.

## Violinplot zur Preisverteilung

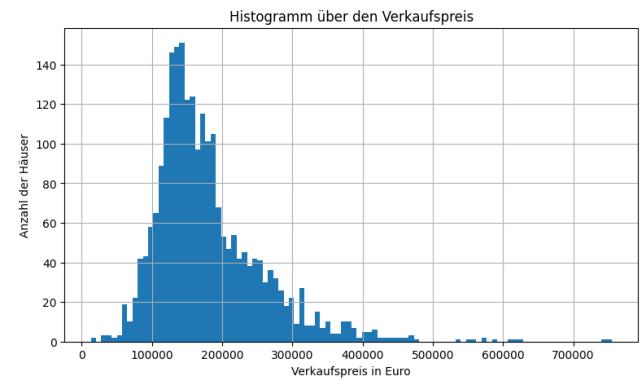
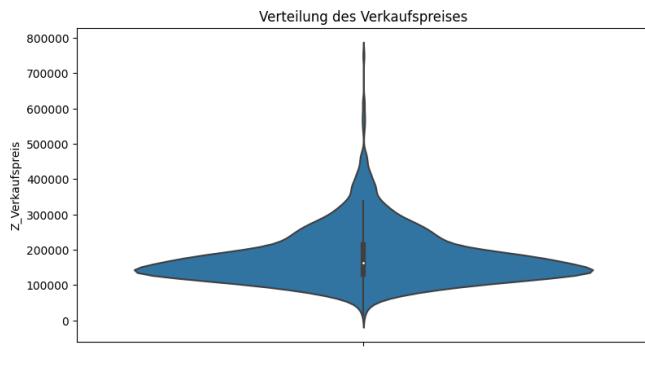
Die Verteilung und Häufigkeit des Verkaufspreises werden nach dieser ersten Eingrenzung nun genauer analysiert.

```
In [ ]: plt.figure(figsize=(20,5))

# Violin plot
plt.subplot(1,2,1)
plt.title("Verteilung des Verkaufspreises")
sns.violinplot(data=df_train, y=df_train["Z_Verkaufspreis"])

# Histogram plot
plt.subplot(1,2,2)
plt.title("Histogramm über den Verkaufspreis")
plt.xlabel("Verkaufspreis in Euro")
plt.ylabel("Anzahl der Häuser")
df_train["Z_Verkaufspreis"].hist(bins=100)

plt.show()
```

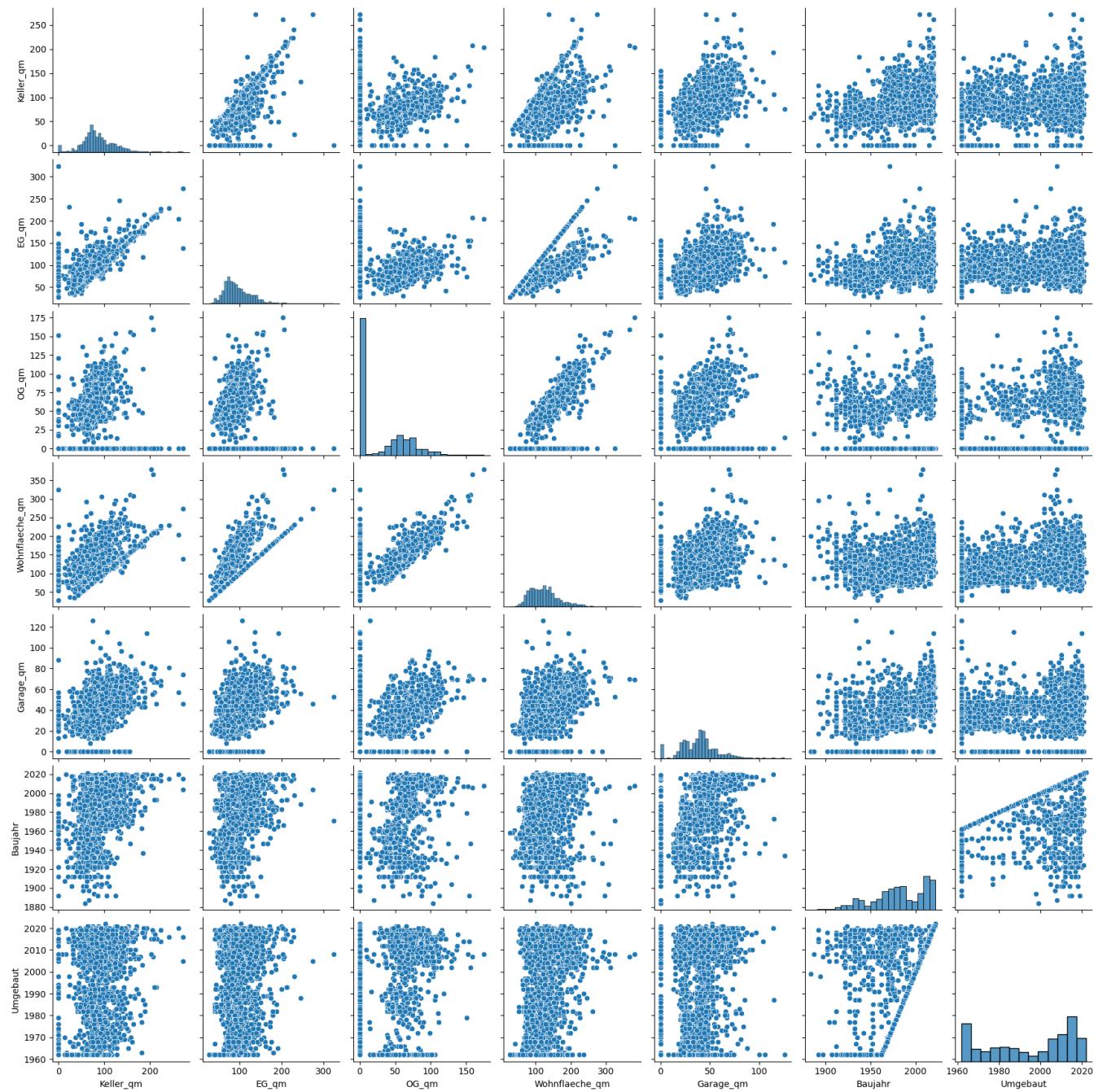


Die meisten Immobilien des Datensatzes befinden sich im Bereich des Durchschnittsverkaufspreises (180.443.51 €) und werden in einer Preisspanne von 100.000 € bis 130.000 € verkauft. Die günstigste Immobilien kostet 13.100 € und die teuerste 755.000 €. Einige wenige Häuser sind preislich deutlich über dem Durchschnitt angesiedelt.

## Scatterplotmatrix zur Erforschung der Attributs-Beziehungen

Anzeige einer Scatterplotmatrix, um die Zusammenhänge von ausgewählten Attributen zueinander zu untersuchen.

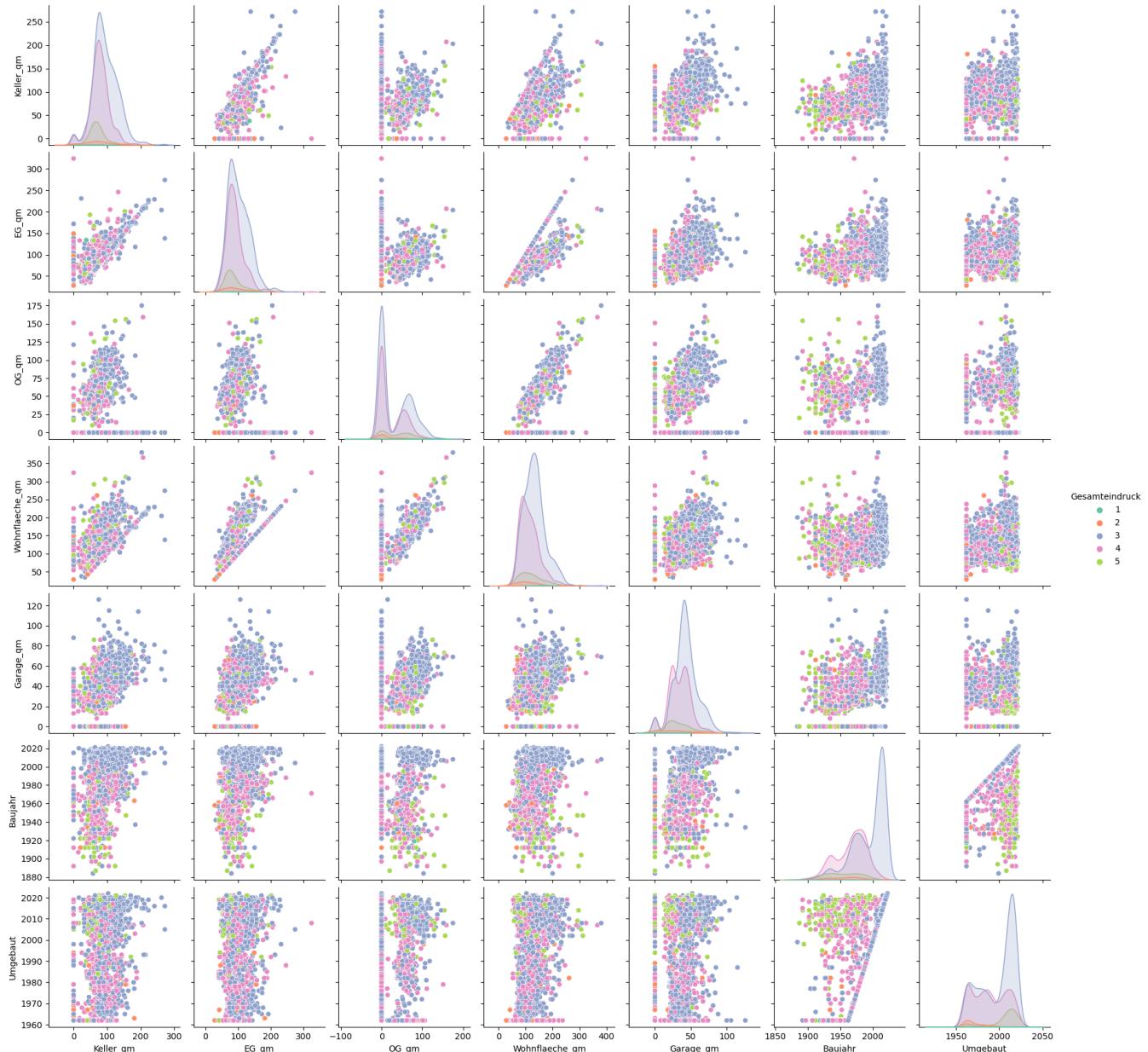
```
In [ ]: sns.pairplot(df_train.loc[:, ["Keller_qm", "EG_qm", "OG_qm", "Wohnflaeche_qm", "Garage_qm", "Plaetze_qm"]], diag_kind="kde")
```



- Die Histogramme auf der Diagonale zeigen:
  - Die Flächen ( Keller\_qm, EG\_qm, OG\_qm, Wohnflaeche\_qm ) sind überwiegend normalverteilt. Zudem ist eine Verschiebung der Verteilung zu erkennen.
  - Der allergrößte Teil der Immobilien besitzt kein Obergeschoss.
  - Die meisten Immobilien mindestens eine Garage.
  - Die Zahl der gebauten Häuser steigt seit Beginn des Datensatzes im Jahr 1900 stetig an. In den Jahren um 1950 und 1990 gab es deutliche Einbrüche im Diagramm. In den letzten 10 Jahren wurde viele Häuser gebaut.
  - Um 1962 und um das Jahr 2015 wurden sehr viele Häuser umgebaut. 1962 stellt den frühesten möglichen Wert für das Attribut Umgebaut im Datensatz dar.
- Es lässt sich ein starker Zusammenhang zwischen Keller\_qm und EG\_qm beobachten. Die Beziehung zwischen EG\_qm und OG\_qm ist hingegen schwach bis kaum ausgeprägt.

- Die Winkelhalbierende zwischen `EG_qm` und `Wohnflaeche_qm` repräsentiert diejenigen Häuser, die nur aus einem Erdgeschoss (und evtl. einem Keller) bestehen.
- Bis auf zwei Häuser besitzen alle Immobilien mit einer `Wohnflaeche_qm` über 200 m<sup>2</sup> einen Keller.
- Alle Häuser besitzen ein älteres `Baujahr` als `Umbaujahr`, was logisch sinnvoll ist.
- Über die Jahre (`Baujahr`) wurden `Keller_qm`, `EG_qm` und `Garage_qm` tendenziell zunehmend größer gebaut.

```
In [ ]: color_palette = sns.color_palette("Set2", 5)
sns.pairplot(df_train.loc[:, ["Keller_qm", "EG_qm", "OG_qm", "Wohnflaeche_qm", "Garage_qm", "Baujahr", "Umbaujahr"]], palette=color_palette)
plt.show()
```



Nimmt man das Attribut `Gesamteindruck` zur bestehenden Scatterplotmatrix hinzu, so fällt auf, dass der Gesamteindruck sehr gemischt auftritt. Die größte Korrelation besteht mit dem Baujahr. Die Immobilien, die in den letzten zwei Jahrzehnten erbaut wurden, werden nahezu alle mit einem Gesamteindruck von 5 (Sehr gut) bewertet.

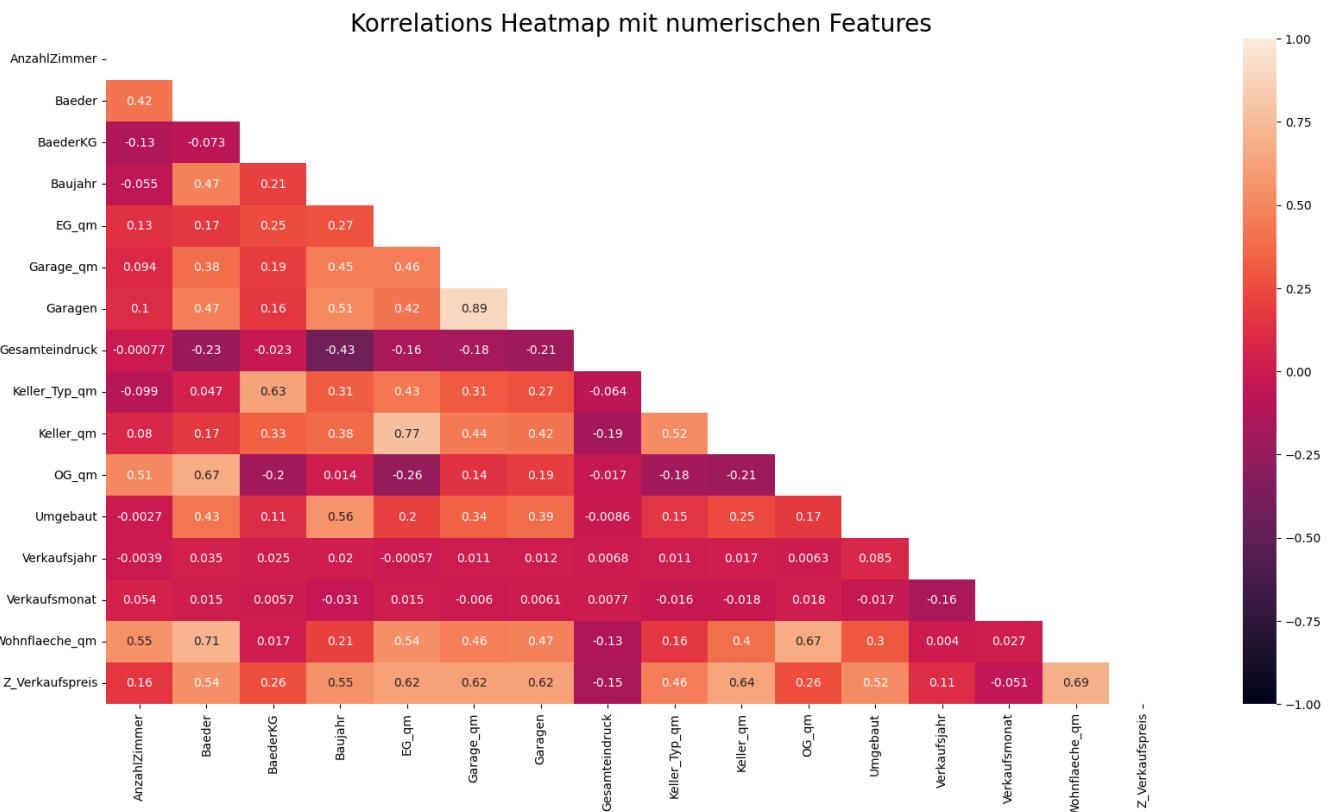
Dieses Phänomen lässt sich dadurch erklären, dass der 'Gesamteindruck' ein sehr subjektives Attribut ist,

das von jedem Menschen anders bewertet werden kann. Ob und inwiefern sich der `Gesamteindruck` für die Vorhersage des `Verkaufspreises` eignet, wird im weiteren Verlauf der Aufgabe untersucht.

## Korrelationsmatrix über alle Features des Datensatzes

Im ersten Schritt werden nur die numerischen Features des Datensatzes berücksichtigt. Nach der abgeschlossenen Data Preparation (Aufgabe 3) wird dann die Korrelationsmatrix als Heatmap erneut über alle (encodierten) Attribute ausgegeben und verglichen.

```
In [ ]: # Draw a correlation matrix heatmap and mask out the upper triangle
plt.figure(figsize=(20, 10))
mask = np.triu(np.ones_like(df_train.iloc[:, 0:20].corr(numeric_only=True), dtype=bool))
heatmap = sns.heatmap(df_train.corr(numeric_only=True), mask=mask, vmin=-1, vmax=1, annot=True)
plt.title('Korrelations Heatmap mit numerischen Features', fontdict={'fontsize': 20})
plt.show()
```



Analyse von auffälligen Korrelationen:

- Am stärksten korrelieren die Anzahl an Garagenstellplätzen ( `Garagen` ) sowie die Garagengröße ( `Garage_qm` ). Ein kausaler, baulich bedingter Zusammenhang ist erkennbar.
- `Keller_qm` und `EQ_qm` sowie `Wohnfläche_qm` und `Baeder` korrelieren ebenfalls sehr stark. Ein kausaler Zusammenhang ist ebenfalls erkennbar.
- Die stärkste negative Korrelation besteht zwischen `Gesamteindruck` und `Baujahr`. Daran lässt sich erkennen, dass eine Verringerung des Baujahrs einen schlechteren (kleineren) Gesamteindruck zur Folge hat und umgekehrt.
- Auffällig ist auch, dass der Verkaufspreis mit gerade einmal 0,16 nur sehr gering mit der `AnzahlZimmer` korreliert. Instinktiv würde man hier einen Zusammenhang vermuten, da die Zimmeranzahl für viele Immobilieninteressenten in der realen Welt ein ausschlaggebendes Kriterium zu sein scheint.
- Es besteht kein Zusammenhang zwischen `Verkaufsmonat` und `Verkaufsjahr`.
- `Z_Verkaufspreis` und `Verkaufsjahr` korrelieren nicht, obwohl man annehmen würde, dass durch die Inflation die Verkaufspreise der Immobilien über die Jahre steigen müssten.

Die Korrelationsmatrix zeigt, dass der Verkaufspreis ( `Z_Verkaufspreis` ) von folgenden Attributen abhängig ist:

- `Baeder` : 0,54
- `Baujahr` : 0,55
- `EG_qm` : 0,62
- `Garage_qm` : 0,62
- `Garagen` : 0,62
- `Keller_qm` : 0,64
- `Umgebaut` : 0,52
- `Wohnflaeche_qm` : 0,69

- Am stärksten korreliert die Größe der Wohnfläche mit dem Verkaufspreis. Je größer also die Immobilie, desto höher der Preis.
- Viele Bäder sind ein Indiz für ein großes Haus und somit einen hohen Verkaufspreis.
- Neue Häuser (aktuelles Baujahr) sind, bei vergleichbaren restlichen Attributen, immer teurer als ältere Exemplare.
- Die Quadratmeteranzahl im Erdgeschoss ist ein sehr wichtiger Indikator für den Verkaufspreis und zeigt deshalb eine hohe Korrelation auf. Eng damit verwandt sind `Wohnflaeche_qm` und `Keller_qm`, die bauliche Gemeinsamkeiten haben.
- Die Anzahl der Garagenstellplätze korreliert überraschend hoch mit dem Verkaufspreis. Dieser Zusammenhang wird im nächsten Abschnitt noch genauer untersucht.

## Untersuchen des Zusammenhangs einzelner Attribute mit dem Verkaufspreis

Um die Untersuchung zu erleichtern, werden zuerst Funktionen erstellt, die Visualisierungen zwischen einem ausgewählten Attribut und dem Verkaufspreis ermöglichen.

- **display\_scatterplot:** Zeigt einen Scatterplot zum Vergleichen des Zusammenhangs der gegebenen Spalte mit dem Verkaufspreis.
- **display\_boxplot:** Zeigt einen Boxplot über Verkaufspreis und gegebener Spalte an.

```
In [ ]: # Function to display a scatterplot
def display_scatterplot(attribute, xlabel):
    # Create graph
    plt.figure(figsize=(14, 7))
    plt.scatter(df_train[attribute], df_train["Z_Verkaufspreis"])

    # Add and display captions
    plt.title(f"Verkaufspreis vs. {xlabel}")
    plt.xlabel(xlabel)
    plt.ylabel("Verkaufspreis in Euro")
    plt.show()

# Function to display a boxplot
def display_boxplot(attribute, xlabel, title, rotate_xlabel=False):
    # Create graph
    plt.figure(figsize=(14, 7))
    sns.boxplot(x=attribute, y="Z_Verkaufspreis", data=df_train, color="orange")
```

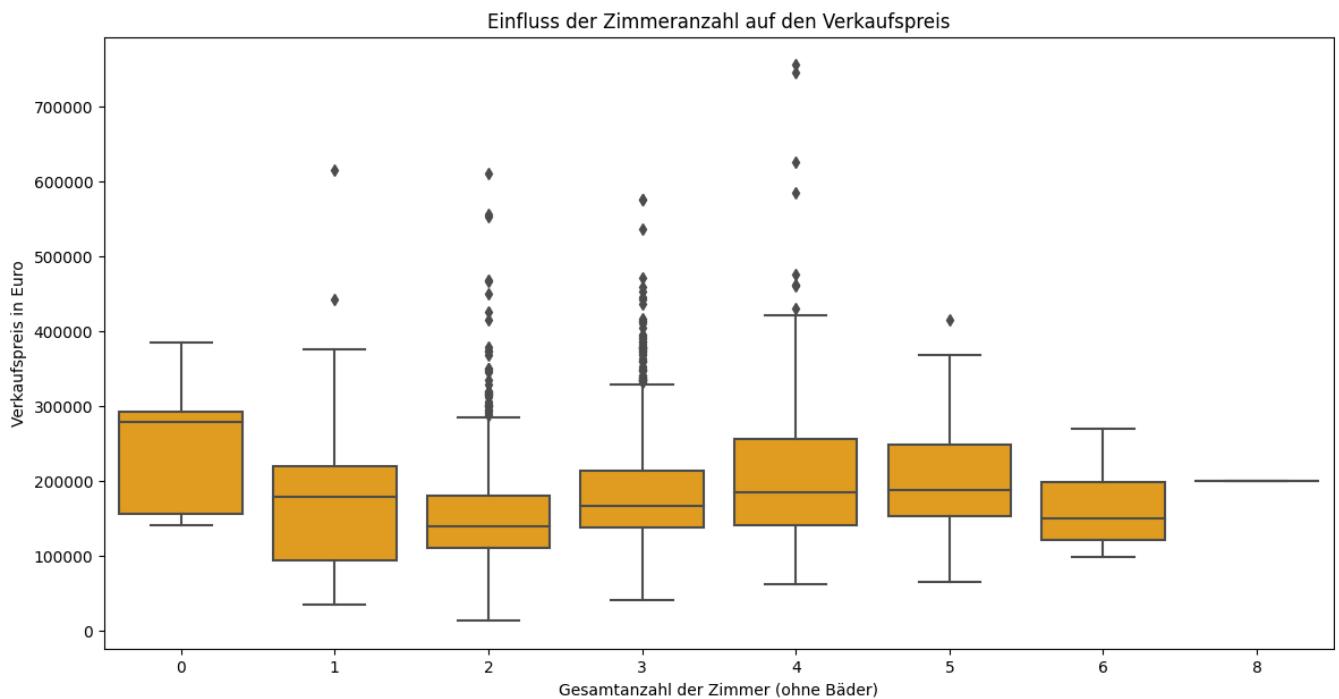
```

# Add and display captions
plt.title(f"Einfluss {title} auf den Verkaufspreis")
plt.xlabel(xlabel)
if rotate_xlabel == True: plt.xticks(rotation = 45)
plt.ylabel("Verkaufspreis in Euro")
plt.show()

```

## Einfluss der Zimmeranzahl auf den Verkaufspreis

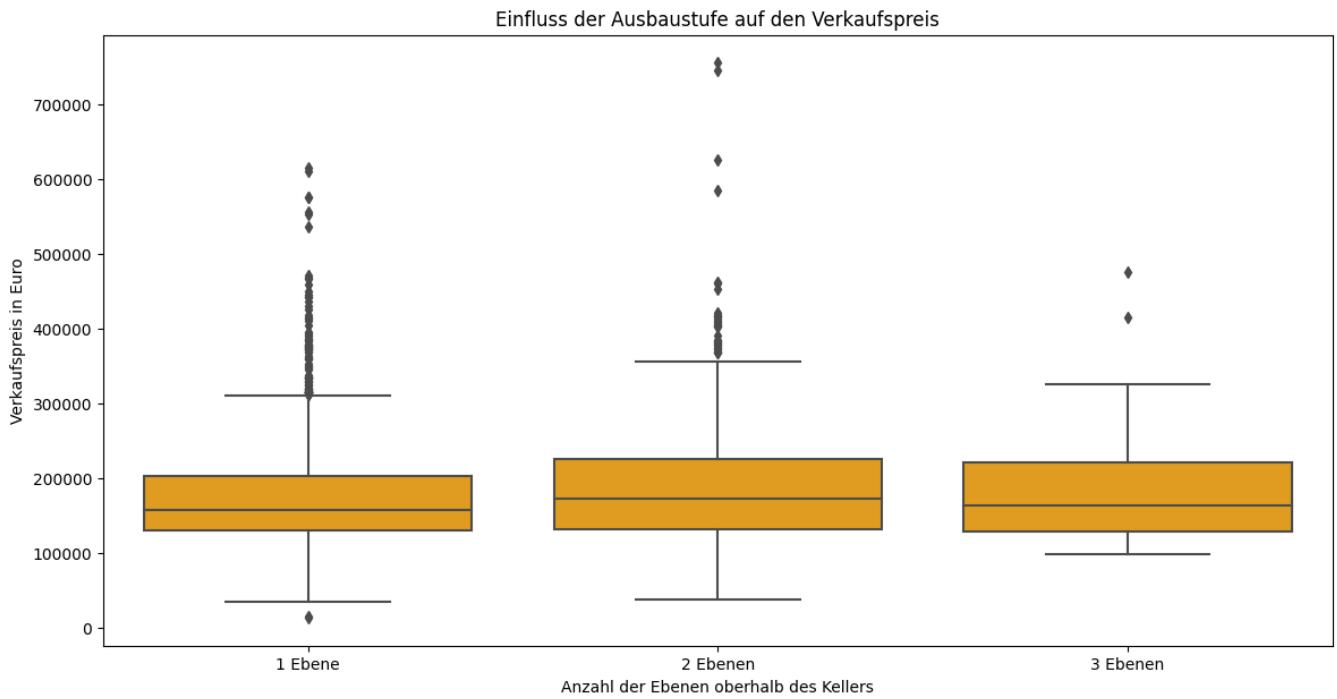
```
In [ ]: display_boxplot(attribute="AnzahlZimmer", xlabel="Gesamtanzahl der Zimmer (ohne Bäder)", titl
```



Die Korrelationsmatrix hat bereits gezeigt, dass es einen überraschend geringen Zusammenhang zwischen der Anzahl der Zimmer und dem Verkaufspreis gibt. Die erwartete, stetige Steigerung des Verkaufspreises in Abhängigkeit von der Zimmeranzahl bleibt hier aus. Häuser mit zwei Zimmern kosten im Durchschnitt ähnlich viel wie Häuser mit sechs Zimmern. Der einzelne Dateneintrag mit acht Zimmern ist nicht repräsentativ und kann später als Ausreißer entfernt werden. Des Weiteren sind die Immobilien mit 0 Zimmern (Kellerwohnungen) nicht repräsentativ und können in Aufgabe 3 ebenfalls entfernt werden.

## Einfluss der Ausbaustufe auf den Verkaufspreis

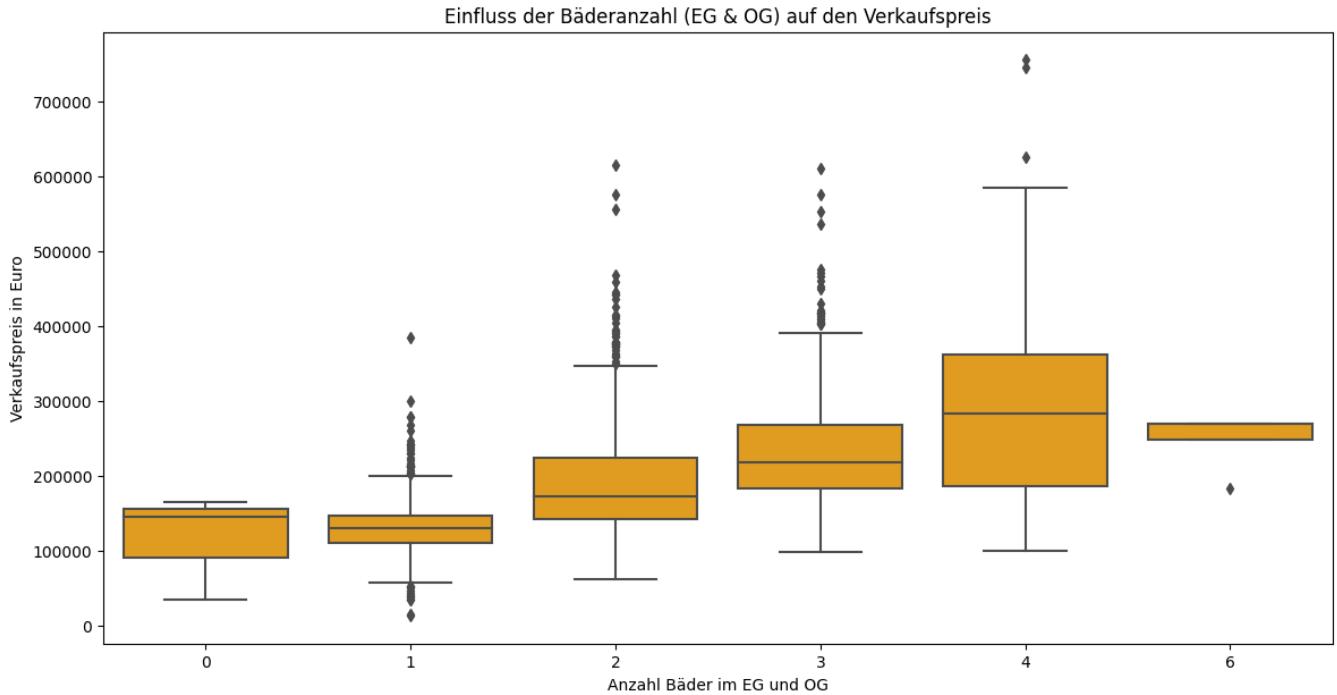
```
In [ ]: display_boxplot(attribute="Ausbaustufe", xlabel="Anzahl der Ebenen oberhalb des Kellers", tit
```



Es gibt keinen wesentlichen Zusammenhang zwischen der Ausbaustufe und dem Verkaufspreis.

### Einfluss der Bäderanzahl (EG und OG) auf den Verkaufspreis

```
In [ ]: display_boxplot(attribute="Baeder", xlabel="Anzahl Bäder im EG und OG", title="der Bäderanzahl")
```

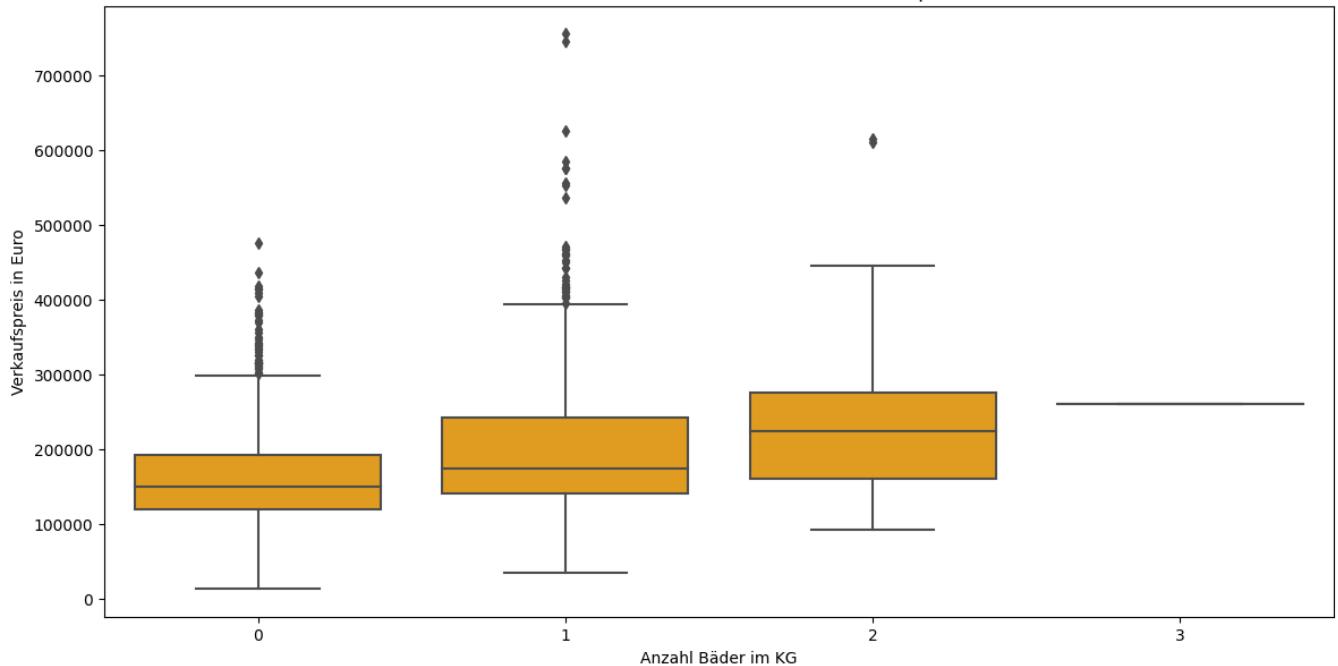


Bei den Immobilien mit ein bis vier Bädern ist ein starker Anstieg des Verkaufspreises mit der Anzahl an Bädern zu erkennen. Bei 6 Bädern handelt es sich mit 4 Einträgen um Ausreißer. Später werden ebenfalls alle Häuser mit 0 Bädern im EG entfernt (3 Einträge) da diese nicht repräsentativ sind.

### Einfluss der Bäderanzahl (KG) auf den Verkaufspreis

```
In [ ]: display_boxplot(attribute="BaederKG", xlabel="Anzahl Bäder im KG", title="der Bäderanzahl (KG)")
```

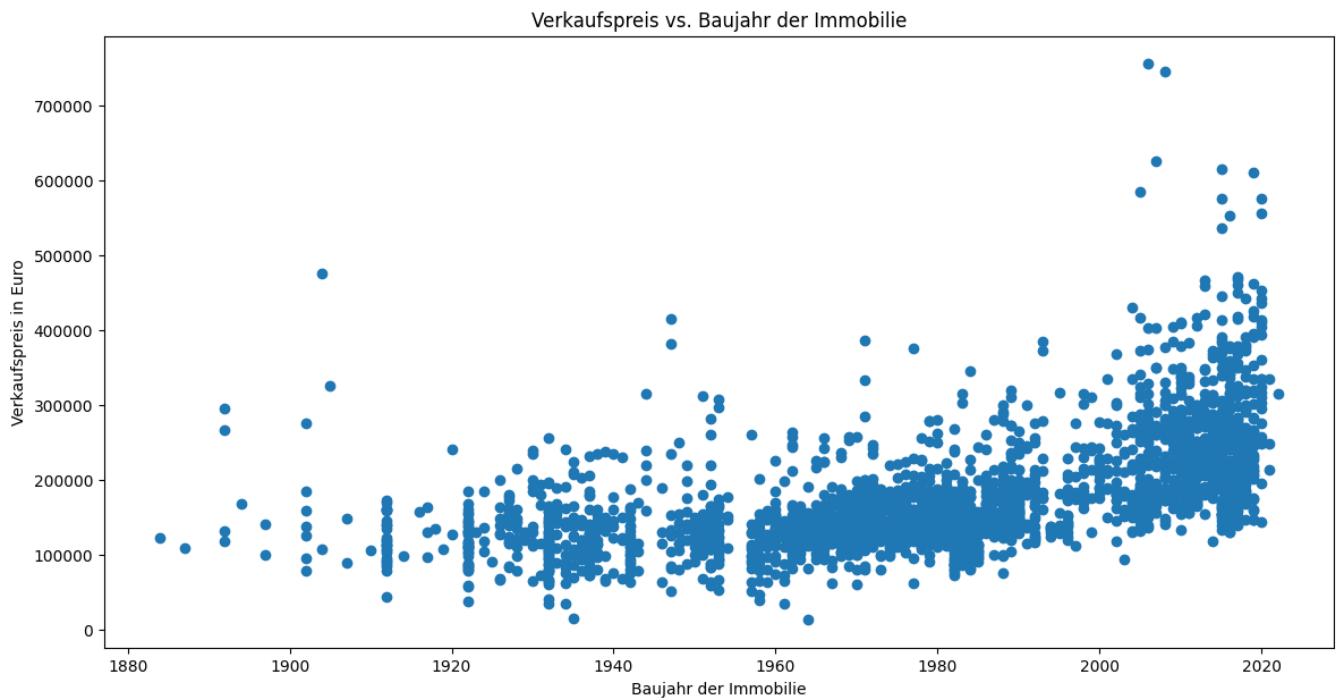
### Einfluss der Bäderanzahl (KG) auf den Verkaufspreis



Insgesamt ist ein starker Zusammenhang zwischen der Anzahl an Bädern im UG und dem Verkaufspreis zu erkennen, jedoch nicht so stark wie bei der Anzahl an Bädern in den oberen Geschossen.

### Einfluss des Baujahrs auf den Verkaufspreis

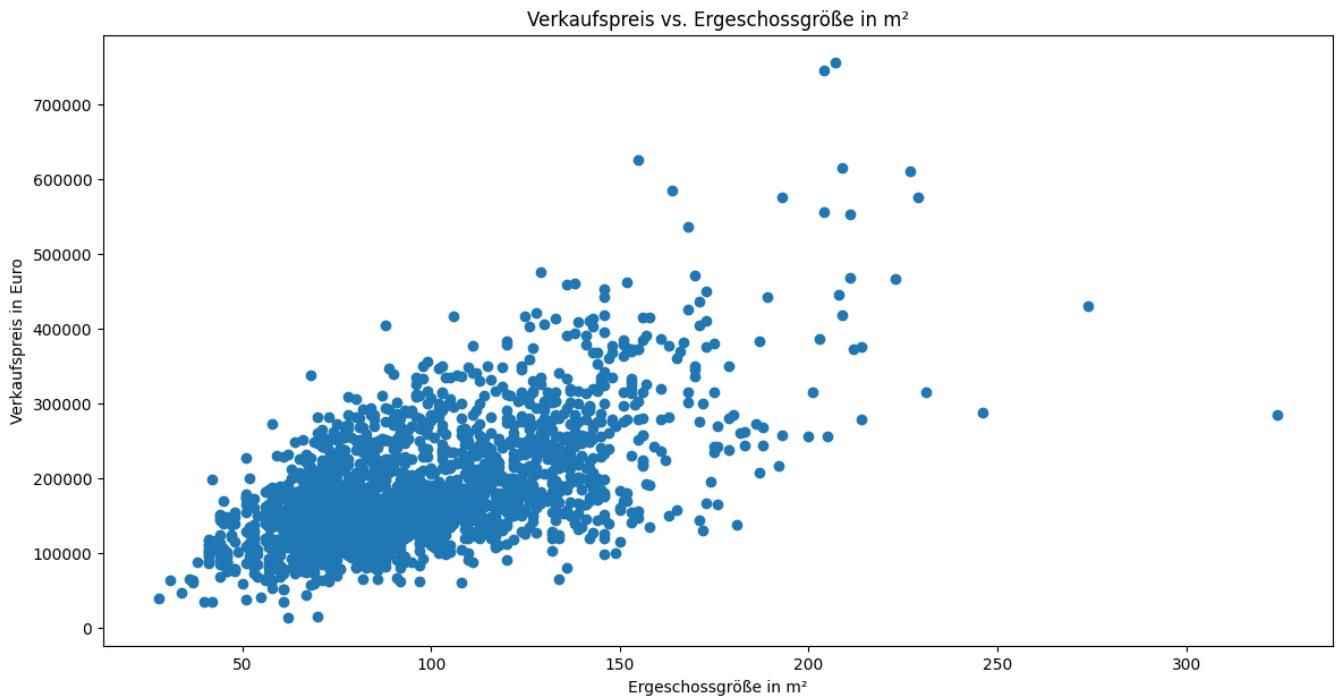
```
In [ ]: display_scatterplot(attribute="Baujahr", xlabel="Baujahr der Immobilie")
```



Hier lässt sich gut erkennen, dass neuere Häuser tendenziell einen höheren Verkaufspreis haben als ältere.

### Einfluss der Erdgeschossgröße auf den Verkaufspreis

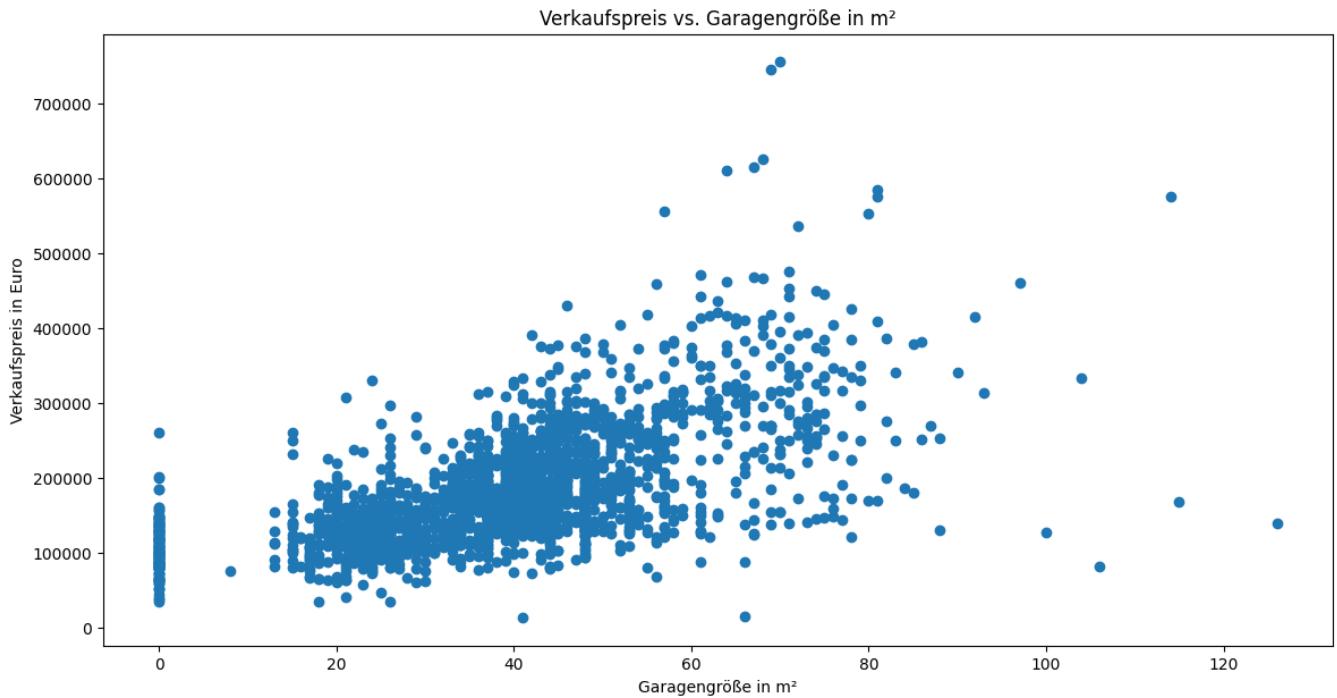
```
In [ ]: display_scatterplot(attribute="EG_qm", xlabel="Ergeschossgröße in m²")
```



Die starke Korrelation zwischen `EG_qm` und `Z_Verkaufspreis` aus der obigen Korrelationsmatrix wird durch diesen Scatterplot visualisiert. Die Ausreißer bei  $EG_{qm} > 250$  werden in Aufgabe 3 entfernt.

### Einfluss der Garagengröße auf den Verkaufspreis

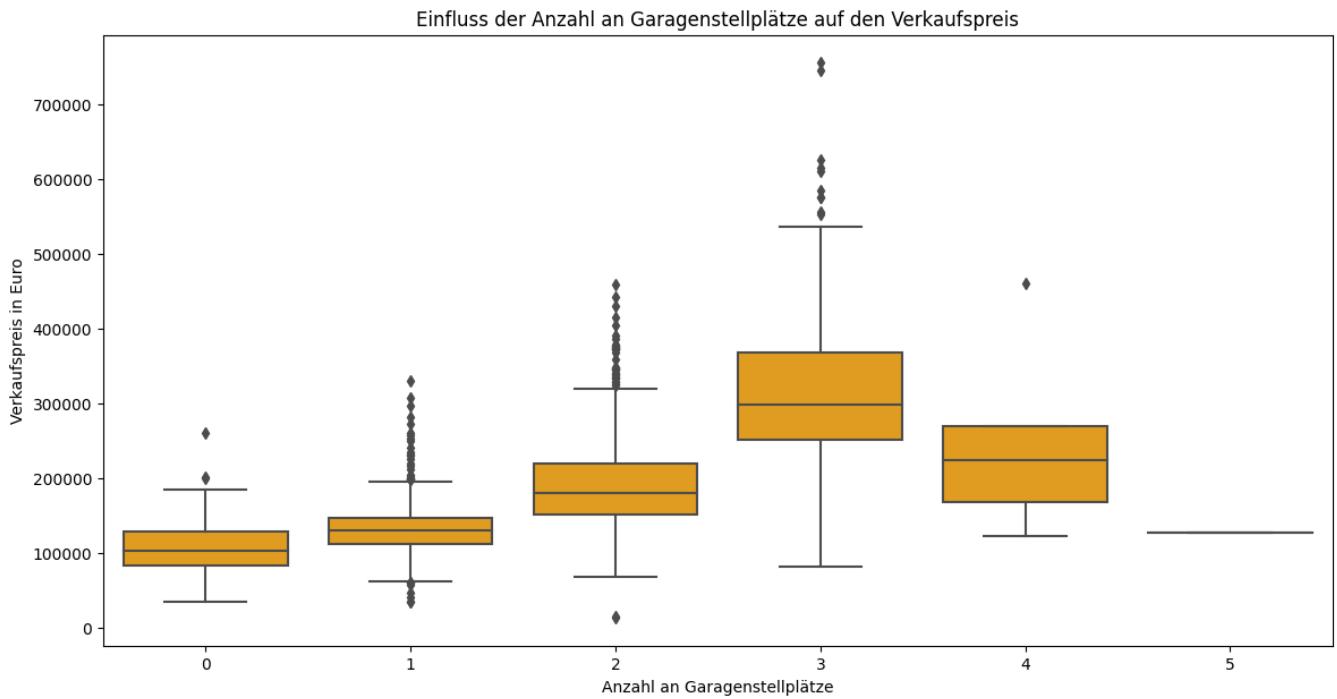
```
In [ ]: display_scatterplot(attribute="Garage_qm", xlabel="Garagengröße in m2)
```



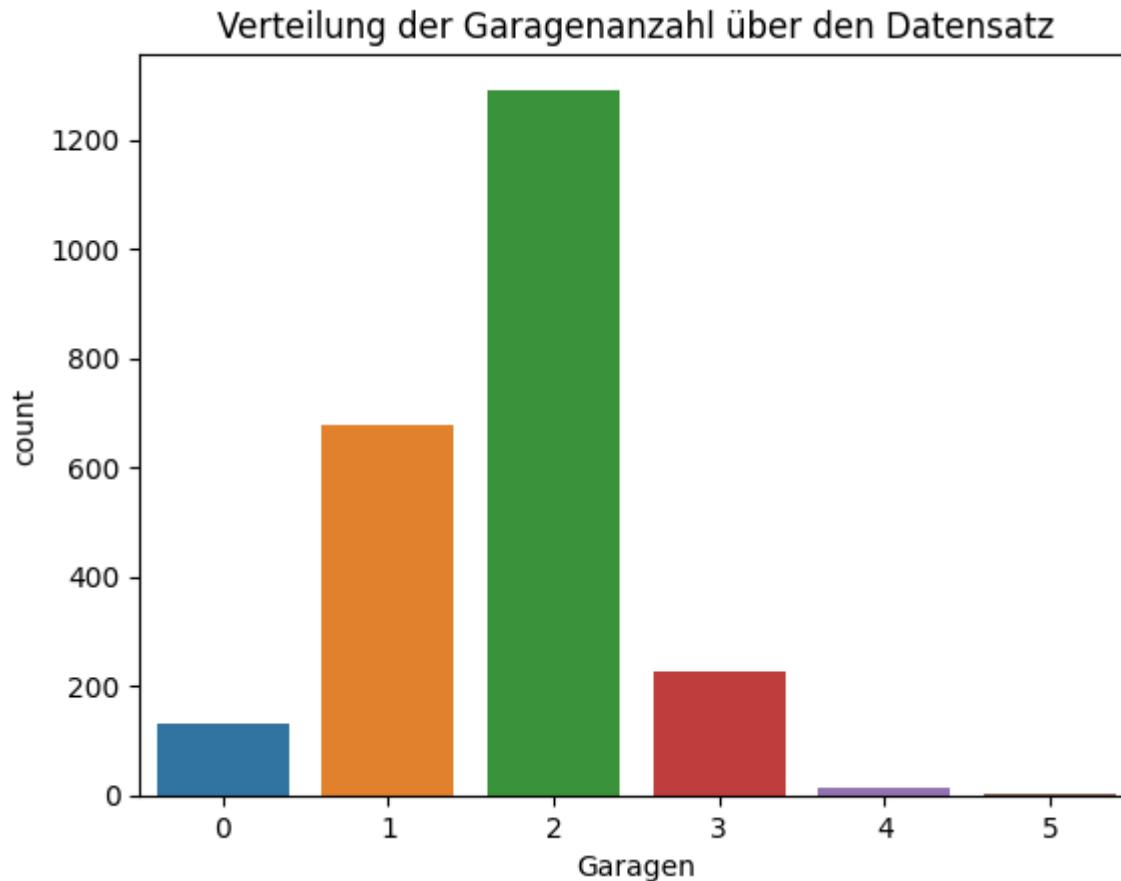
Auch hier lässt sich wie bei der Erdgeschossgröße schon ein Zusammenhang zwischen der Garagengröße und dem Verkaufspreis erkennen. Die Einträge mit einer Flächengröße von 0 bedeuten, dass manche Immobilien keine Garage besitzen.

### Einfluss der Anzahl an Garagenstellplätzen auf den Verkaufspreis

```
In [ ]: display_boxplot(attribute="Garagen", xlabel="Anzahl an Garagenstellplätzen", title="der Anzahl")
```



```
In [ ]: sns.countplot(data=df_train, x="Garagen")
plt.title("Verteilung der Garagenanzahl über den Datensatz")
plt.show()
```

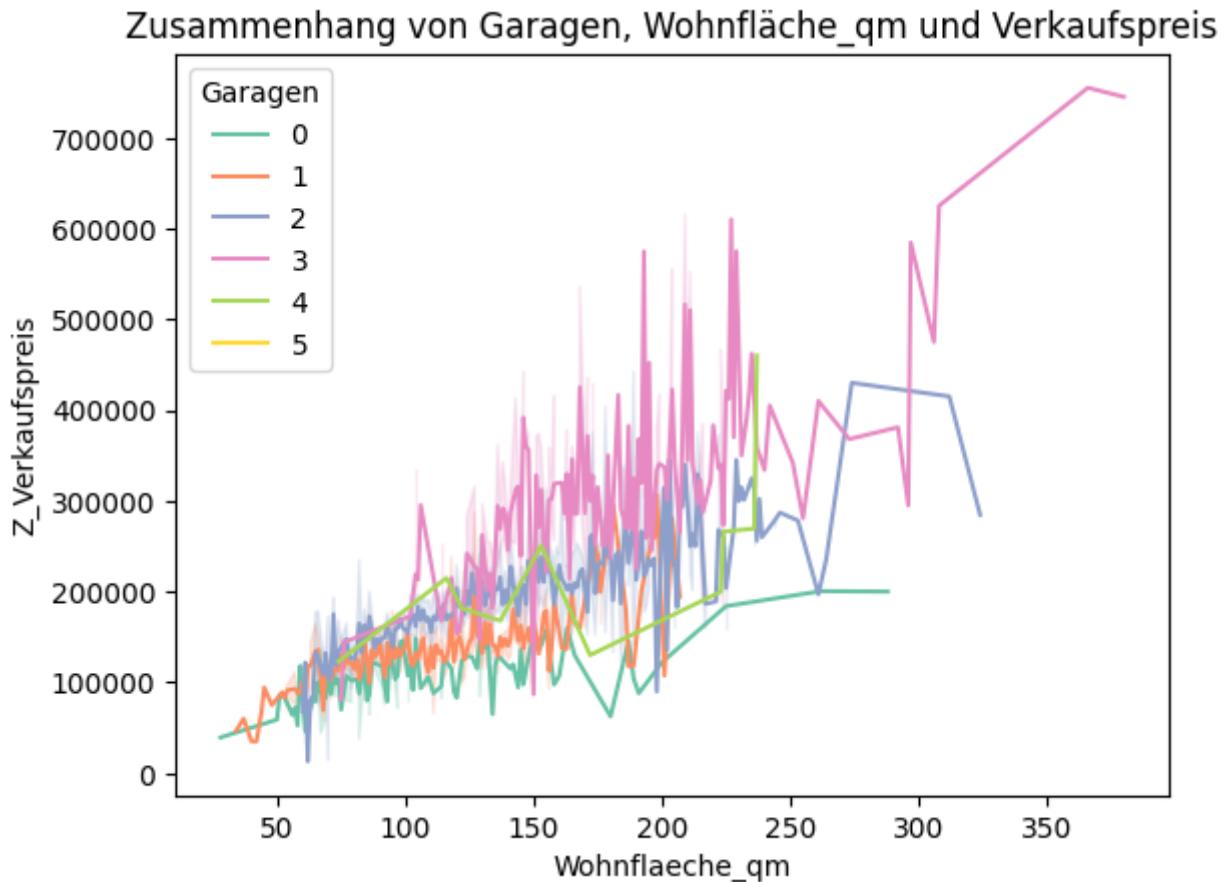


Die meisten Häuser besitzen zwei Garagenstellplätze und befinden sich im mittleren Preissegment. Von einer bis drei Garagen ist im Boxplot bezüglich Median und Maximum ein deutlicher Verlauf zu erkennen. Diese Beobachtung stimmt mit der Korrelation in der Heatmap überein.

Des Weiteren fällt auf, dass ein Haus mit einer Garage nur eine geringe Wertsteigerung im Vergleich zu einem Haus mit keiner Garage erfährt. Jedoch ist ein Haus mit zwei Garagen im Median ca. 50.000 € mehr wert als ein Haus mit einer Garage. Ein Haus mit drei Garagen ist im Median sogar ca. 100.000 € mehr wert als ein Haus mit zwei Garagen. Der Verkaufspreis von Häusern mit vier Garagen ist aber

erstaunlicherweise deutlich geringer als der Preis bei drei Garagen. Dieser Verstoß gegen den Trend könnte daran liegen, dass es nur 13 Immobilien mit vier Garagen im Datensatz gibt. Da es nur einen Eintrag mit fünf Garagen gibt, kann dieser als Ausreißer entfernt werden.

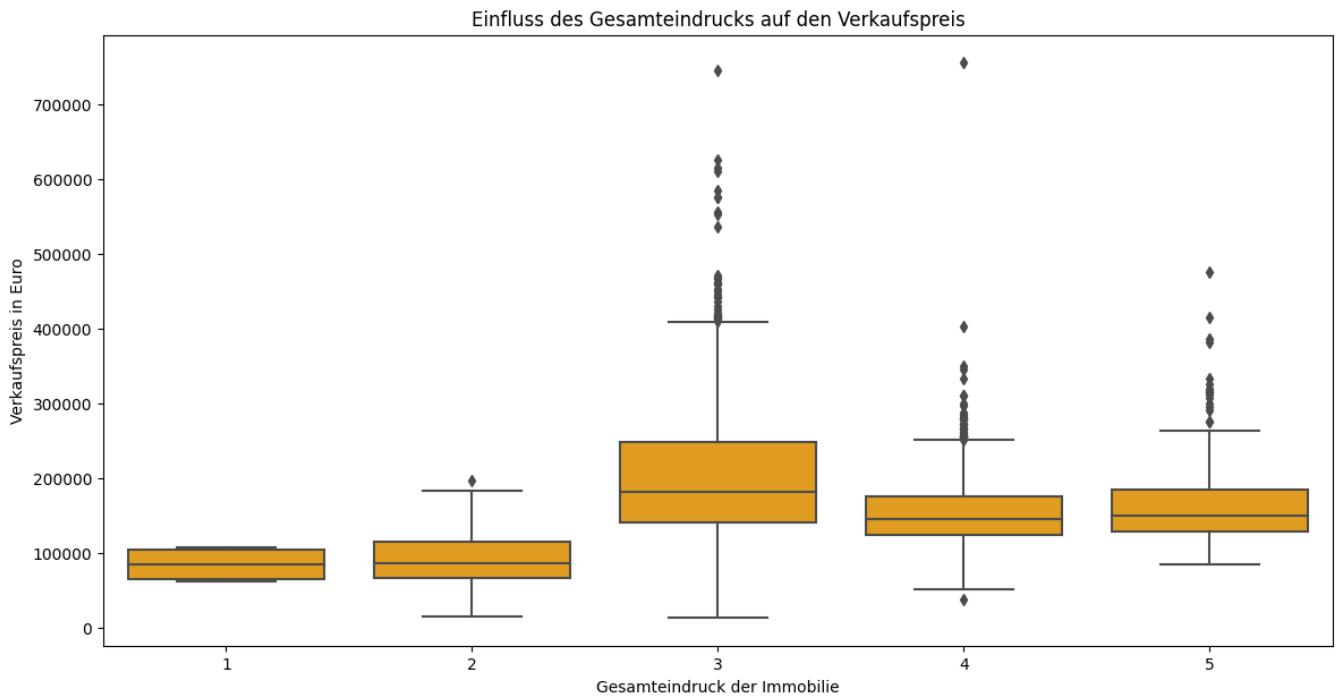
```
In [ ]: color_palette = sns.color_palette("Set2", 6)
sns.lineplot(data=df_train, x="Wohnflaeche_qm", y="Z_Verkaufspreis", hue="Garagen", palette=c
plt.title("Zusammenhang von Garagen, Wohnfläche_qm und Verkaufspreis")
plt.show()
```



Anhand des LinePlots soll untersucht werden, ob die Häuser mit vielen Garagen einfach nur größer und damit teurer sind ( Wohnfläche\_qm ) oder ob die Garagen an sich die Häuser wertvoller machen. Betrachtet man nur die Häuser mit einer Wohnfläche unter 250 m<sup>2</sup>, für die auch eine quantitative Aussage getroffen werden kann, so erkennt man, dass die Garagen nur einen relativ kleinen Einfluss haben. Der Verkaufspreis hängt nur transitiv von den Garagen über die Wohnfläche ab. In größeren Häusern wohnen meist auch mehr Personen, wodurch sich die größere Garagenanzahl erklärt.

## Einfluss des Gesamteindrucks auf den Verkaufspreis

```
In [ ]: display_boxplot(attribute="Gesamteindruck", xlabel="Gesamteindruck der Immobilie", title="des
```

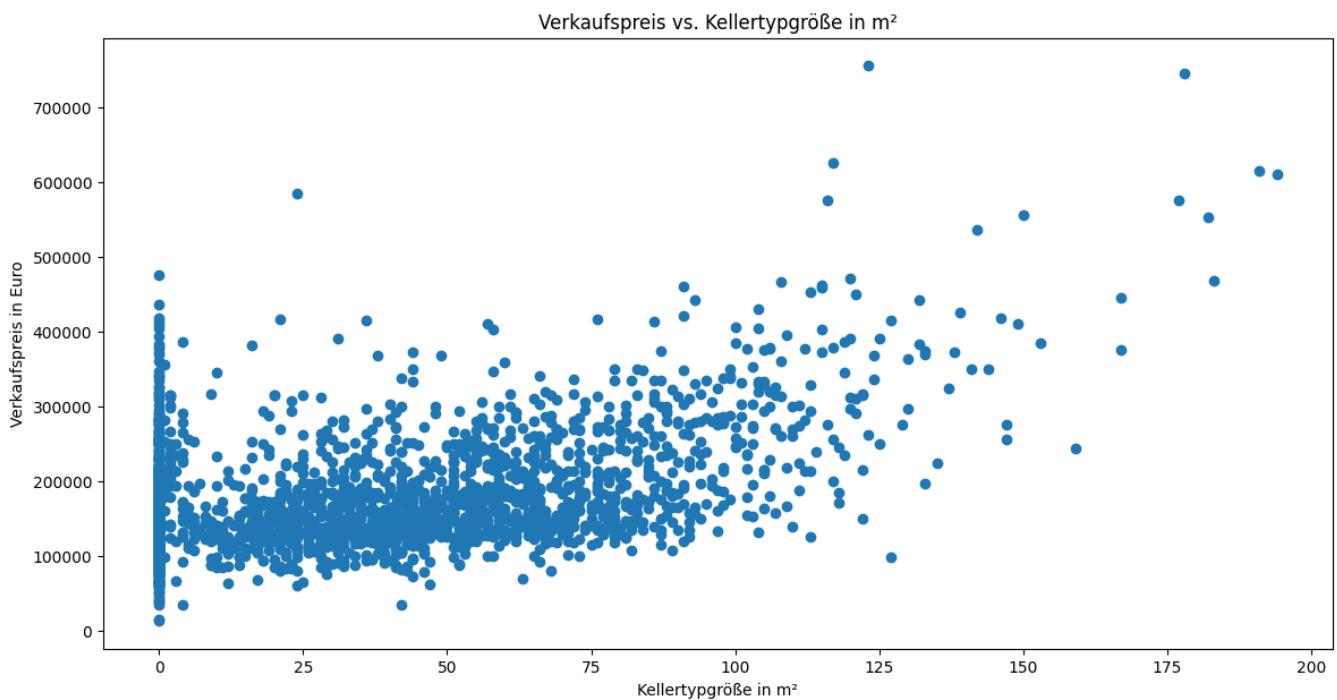


Aus der obigen Heatmap geht bereits hervor, dass es keinen Zusammenhang zwischen Gesamteindruck und Verkaufspreis gibt. Der Boxplot verdeutlicht, dass anhand des Medians kein aufsteigender Trend im Verkaufspreis mit den Bewertungen zu erkennen ist.

- Der Median von 1 und 2 sowie 4 und 5 ist jeweils auf der gleichen Höhe.
- 3 hat einen großen Interquartilsabstand (Spread). Minimum und Maximum liegen hier am weitesten auseinander.
- 3 - 4 besitzen sehr viele Ausreißer (nach oben), die sehr weit über dem Median liegen

## Einfluss der Kellertypgröße auf den Verkaufspreis

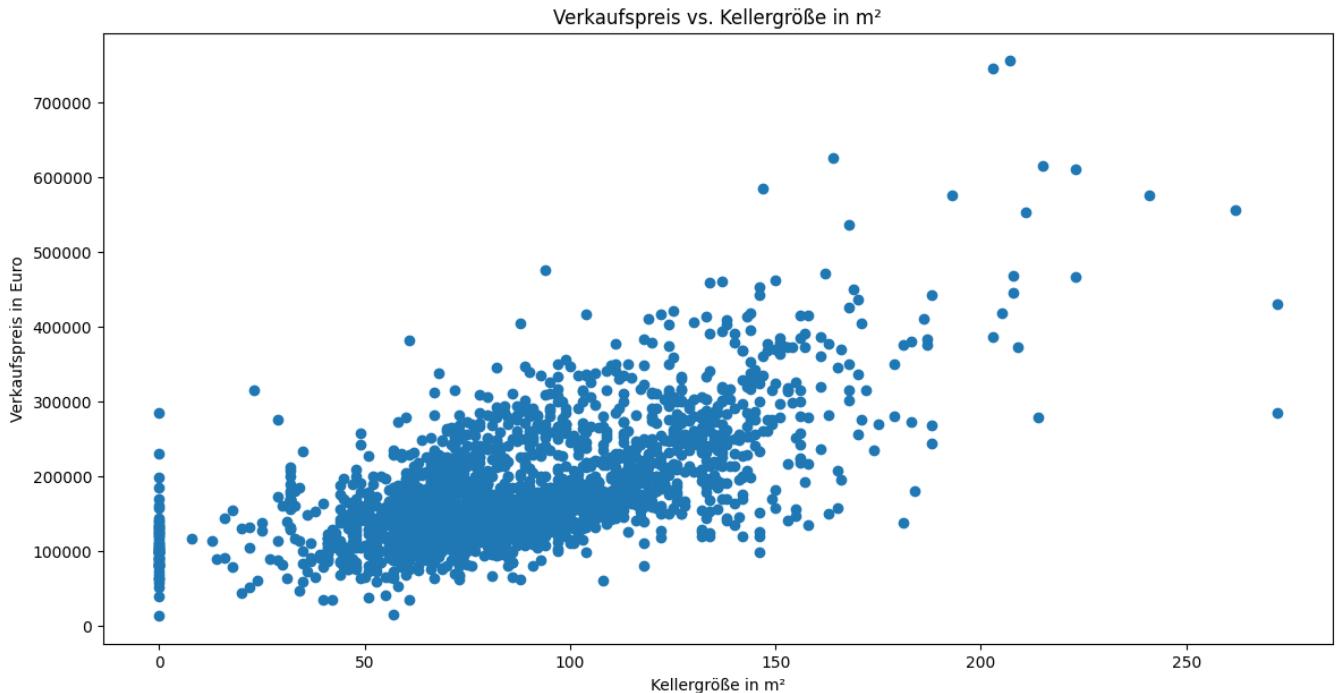
```
In [ ]: display_scatterplot(attribute="Keller_Typ_qm", xlabel="Kellertypgröße in m²")
```



Auch bei der Kellertypgröße ist wie schon bei der gesamten Kellergröße ( Keller\_qm ) ein Zusammenhang zum Verkaufspreis zu erkennen. Die Korrelation fällt allerdings schwächer aus. Einträge mit 0 m<sup>2</sup> lassen darauf schließen, dass die Immobilie keinen Keller besitzt.

## Einfluss der Kellergröße auf den Verkaufspreis

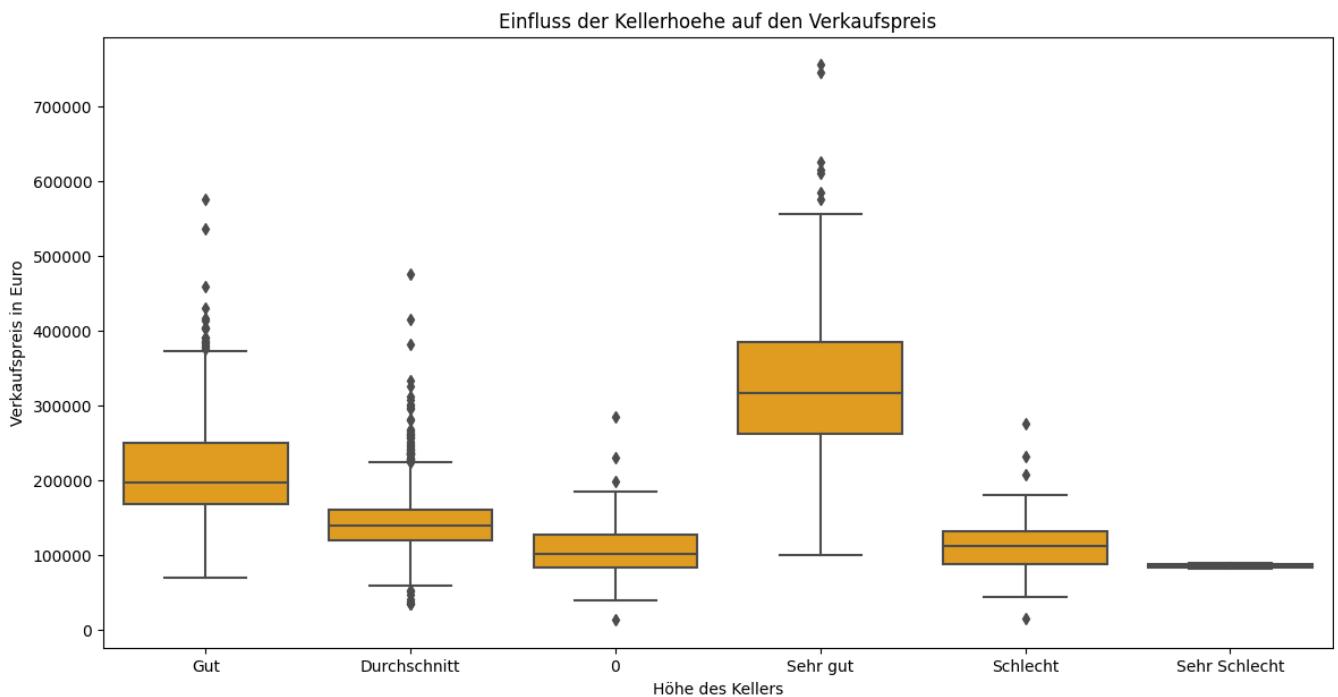
```
In [ ]: display_scatterplot(attribute="Keller_qm", xlabel="Kellergröße in m²")
```



Dieser Plot ist annähernd deckungsgleich mit dem Scatterplot über `EG_qm`. Dies ist keine Überraschung, da beide Attribute eine Korrelation von 0,77 haben. Im Gegensatz zur Erdgeschossfläche gibt es hier allerdings viele Einträge mit 0 m<sup>2</sup> Kellerfläche. Dies bedeutet, dass die Häuser keinen Keller besitzen.

## Einfluss der Kellerhöhe auf den Verkaufspreis

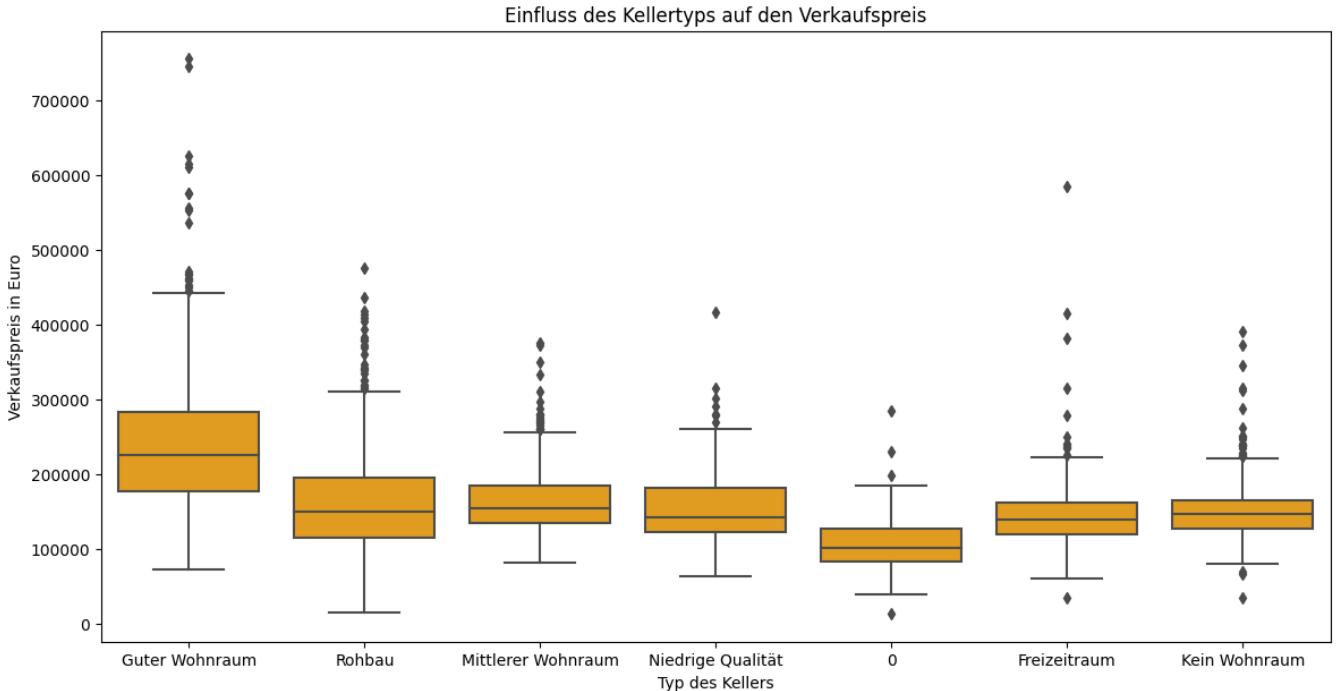
```
In [ ]: display_boxplot(attribute="Kellerhoehe", xlabel="Höhe des Kellers", title="der Kellerhoehe")
```



Anhand des Boxplots ist der Zusammenhang zwischen Kellerhöhe und Verkaufspreis ersichtlich. An den Medianen lässt sich ein aufsteigender Trend von *Sehr Schlecht* bis *Sehr gut* erkennen. Höhere Keller (Sehr gut: ca. 250 cm) sorgen im Mittel für höhere Verkaufspreise.

## Einfluss des Kellertyps auf den Verkaufspreis

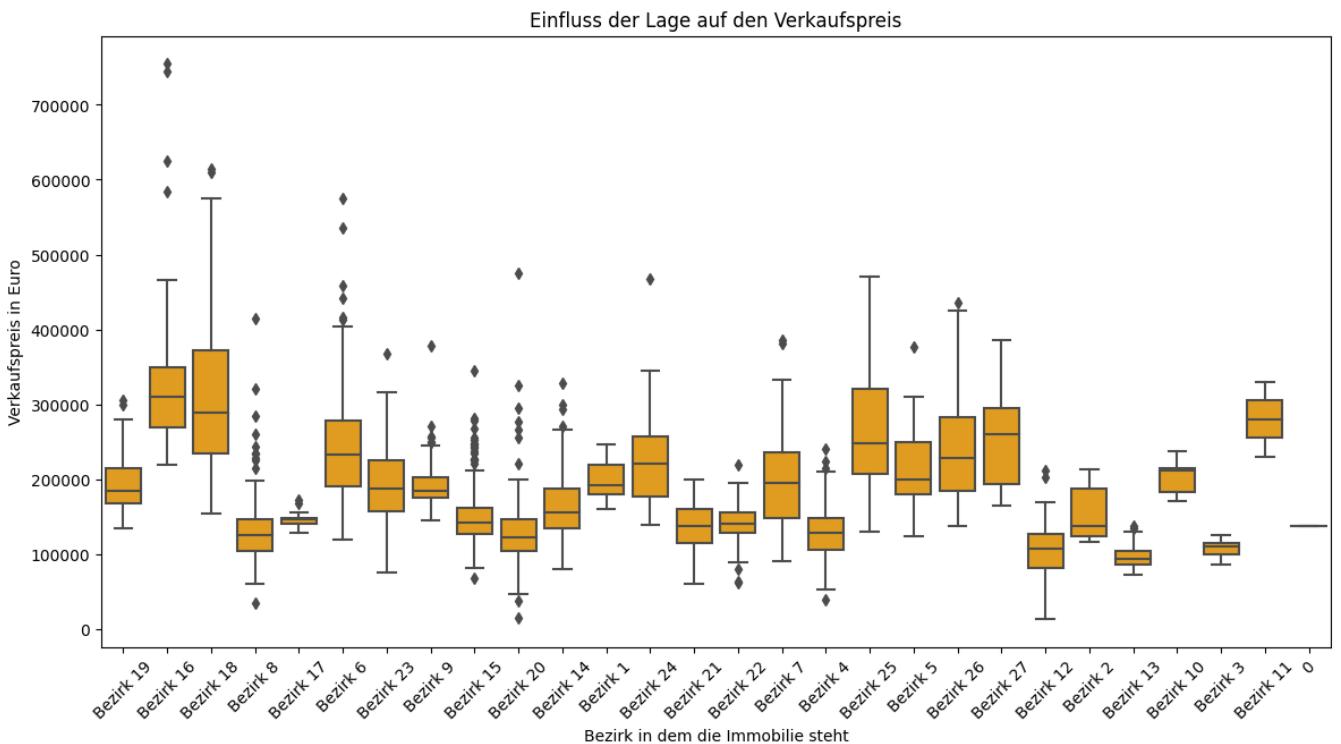
```
In [ ]: display_boxplot(attribute="Kellertyp", xlabel="Typ des Kellers", title="des Kellertyps")
```



Es ist kein wesentlicher Zusammenhang zwischen dem Kellertyp und dem Verkaufspreis ersichtlich. Auffällig ist nur, dass die Immobilien mit dem Kellertyp "Guter Wohnraum" im Mittel etwa um 70.000 € höhere Verkaufspreise besitzen als die übrigen Kellertypen. Bei allen anderen Kellertypen (außer "0") liegt der durchschnittliche Verkaufspreis bei ca. 150.000 €. Die Häuser mit dem Kellertyp 0 werden in der Datenbereinigung entfernt, da sie nicht repräsentativ sind.

## Einfluss der Lage auf den Verkaufspreis

```
In [ ]: display_boxplot(attribute="Lage", xlabel="Bezirk in dem die Immobilie steht", title="der Lage")
```



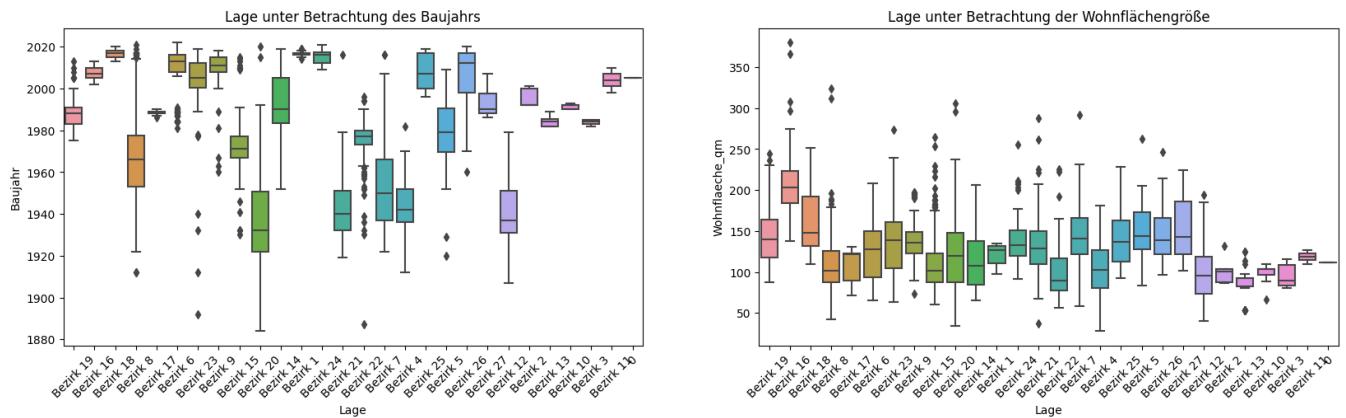
Auch die Lage wirkt sich auf den Preis aus. Dabei fällt vor allem auf, dass es Bezirke gibt, in denen die Immobilien teuer sind (z.B. Bezirk 16, Bezirk 11 und Bezirk 18). Dem gegenüber sind z.B. Bezirk 13 und Bezirk 3 eher billigere Wohngegenden. Es muss jedoch überprüft werden, ob die unterschiedliche Bewertung rein an der Lage liegt, oder eher daran, dass die Häuser in manchen Bezirken einfach neuer (`Baujahr`) oder größer (`Wohnflaeche_qm`) und dadurch teurer sind.

```
In [ ]: plt.figure(figsize=(20,5))

# Lage - Baujahr
plt.subplot(1,2,1)
ax = sns.boxplot(data=df_train, x="Lage", y="Baujahr")
plt.xticks(rotation=45)
plt.title("Lage unter Betrachtung des Baujahrs")

# Lage - Wohnflaeche_qm
plt.subplot(1,2,2)
ax = sns.boxplot(data=df_train, x="Lage", y="Wohnflaeche_qm")
plt.xticks(rotation=45)
plt.title("Lage unter Betrachtung der Wohnflächengröße")

plt.show()
```



### Lage unter Betrachtung des Baujahrs:

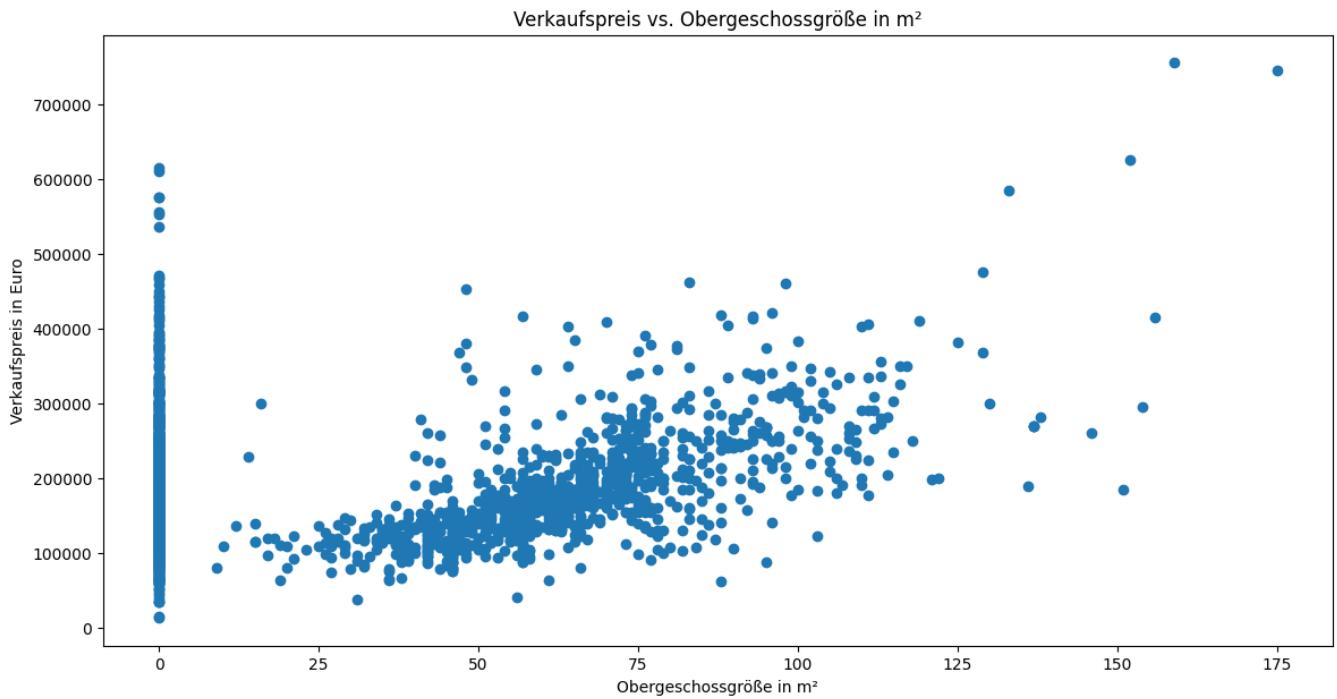
Die teuersten Bezirke (Bezirk 16, Bezirk 11 und Bezirk 18) sind tatsächlich Gegenden mit vielen neuen Häusern. Jedoch gibt es auch Gegenden wie Bezirk 1 und Bezirk 24, die neuere oder genauso neue Häuser haben, aber bei weitem nicht so teuer sind. Es ist also keine Korrelation zwischen Lage und Baujahr gegeben.

### Lage unter Betrachtung der Wohnflächengröße:

Die teuren Gegenden haben nicht zwingend auch die größte Wohnfläche. Bei Bezirk 16 trifft es zu, aber Bezirk 11 hat beispielsweise eine der kleinsten Wohnflächen trotz des hohen Preises. Es ist also keine Korrelation zwischen Lage und Wohnflächengröße gegeben.

## Einfluss der Obergeschoßsgröße auf den Verkaufspreis

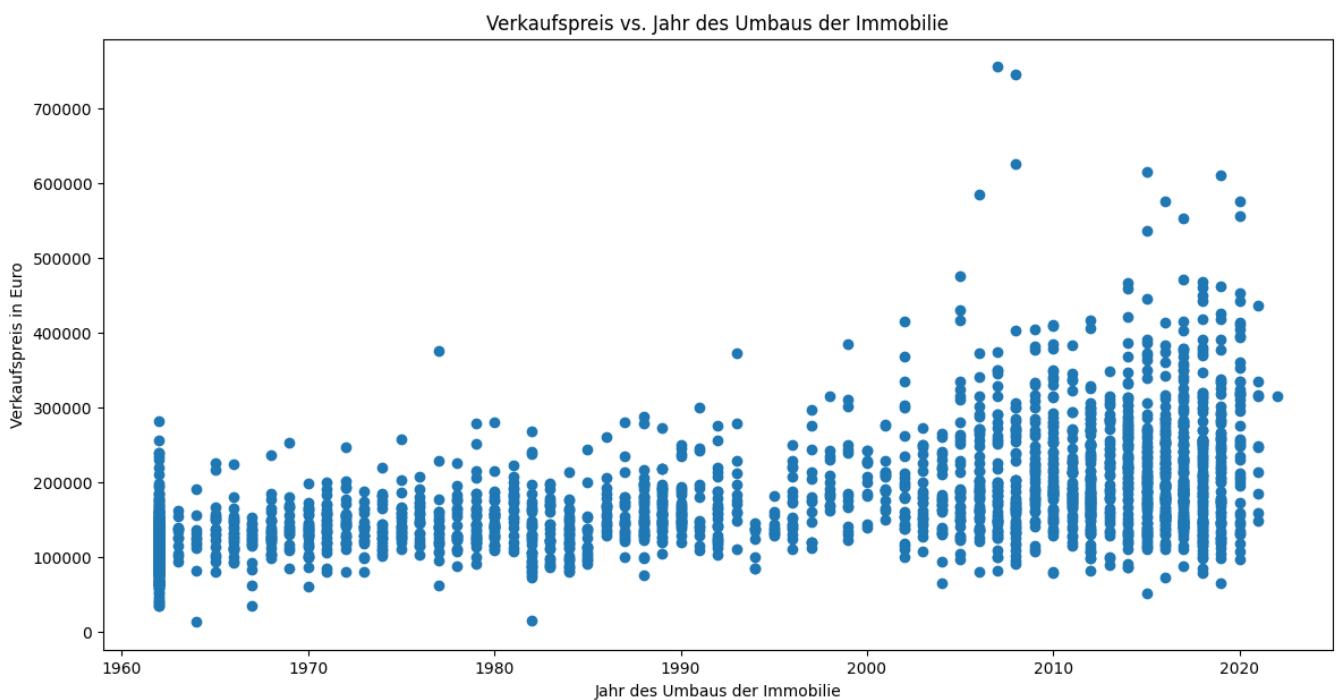
```
In [ ]: display_scatterplot(attribute="OG_qm", xlabel="Obergeschoßsgröße in m²")
```



Am Scatterplot fällt zunächst ins Auge, dass der aller größte Teil der Immobilien im Datensatz, ca. 56 %, kein(e) Obergeschoss(e) besitzt. Für die übrigen Einträge ist ein starker Zusammenhang erkennbar. Da aber der überwiegende Teil der Einträge für dieses Attribut 0 ist, beträgt die Korrelation in der Heatmap zwischen `OG_qm` und `Z_Verkaufspreis` nur 0,26. Da das Attribut `Wohnflaeche_qm` unter anderem bereits den Wert der Obergeschossgröße enthält, sollte für die späteren Vorhersagen das Attribut `Wohnflaeche_qm` verwendet werden. Eine gleichzeitige Betrachtung beider Features brächte Redundanz.

### Einfluss des Umbaujahres auf den Verkaufspreis

```
In [ ]: display_scatterplot(attribute="Umgebaut", xlabel="Jahr des Umbaus der Immobilie")
```

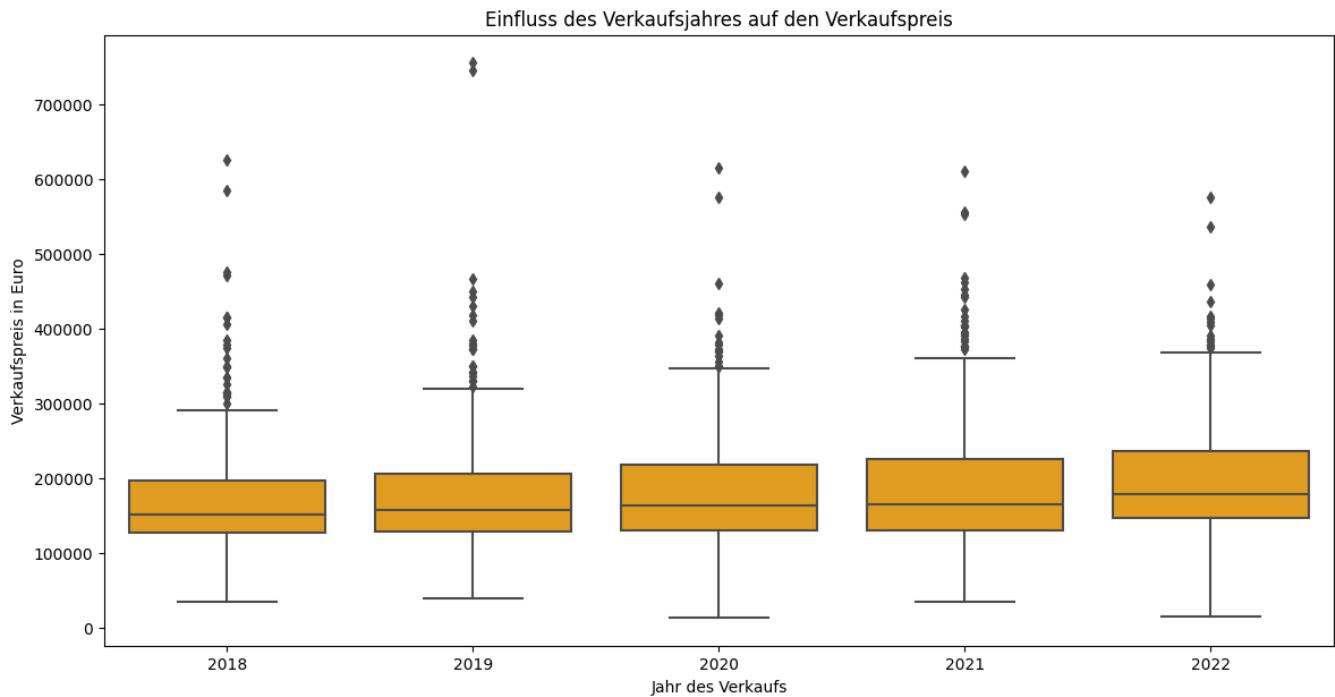


Dieser Scatterplot ähnelt dem obigen über das Baujahr stark und verdeutlicht den Zusammenhang zwischen Verkaufspreis und dem Jahr noch deutlicher. Immobilien, die im obigen Graphen herausstechen (z.B. 475.000 € bei Baujahr 1907, Umgebaut 2005) sind im `Umgebaut`-Plot kaum noch

vorhanden. Daraus lässt sich schließen, dass die Renovierung den Wert eines Hauses steigert. Außerdem bestätigt sich der Zusammenhang "Neuere Häuser sind teurer" auch für alte, aber umgebaute Häuser.

## Einfluss des Verkaufsjahres auf den Verkaufspreis

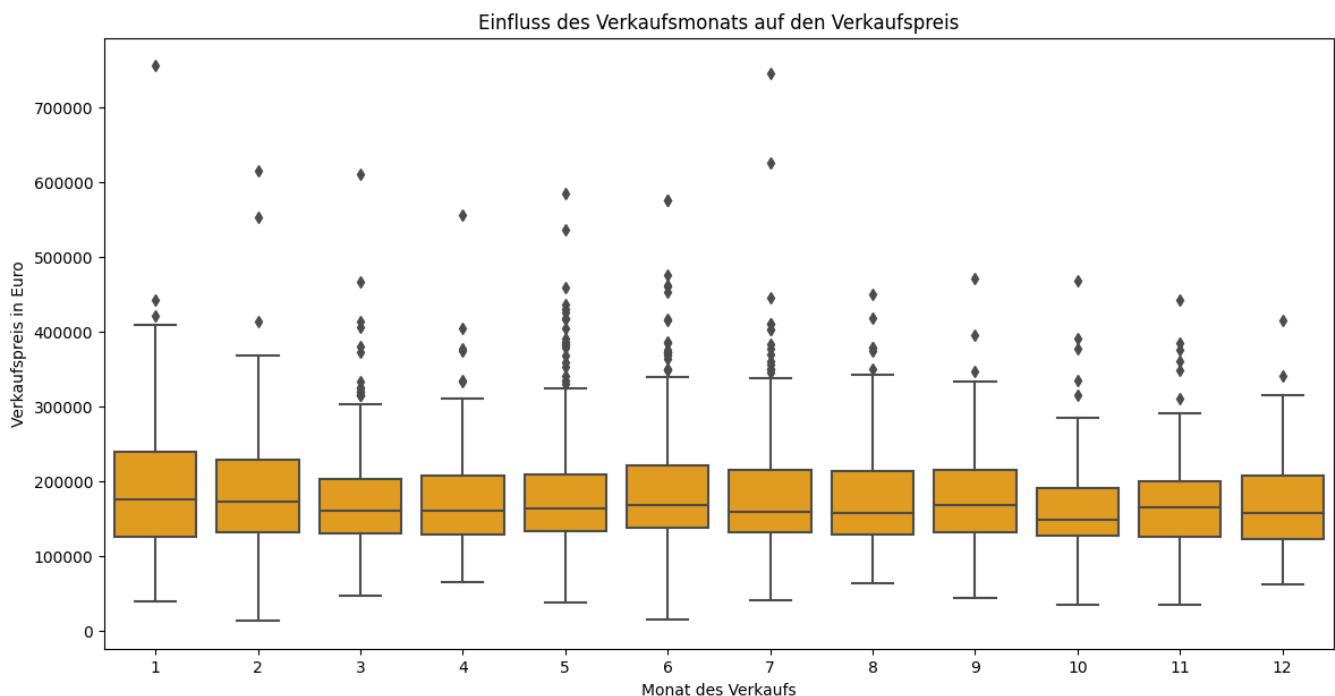
```
In [ ]: display_boxplot(attribute="Verkaufsjahr", xlabel="Jahr des Verkaufs", title="des Verkaufsjahr")
```



Hier ist ein leichter Aufwärtstrend zu erkennen. Häuser, die später verkauft wurden, wurden in der Regel etwas teurer verkauft. Dies hängt wahrscheinlich mit allgemein steigenden Immobilienpreisen und/oder der Inflation zusammen. Zur Klärung des Sachverhalts wäre eine Rückfrage an Expert\*innen hilfreich.

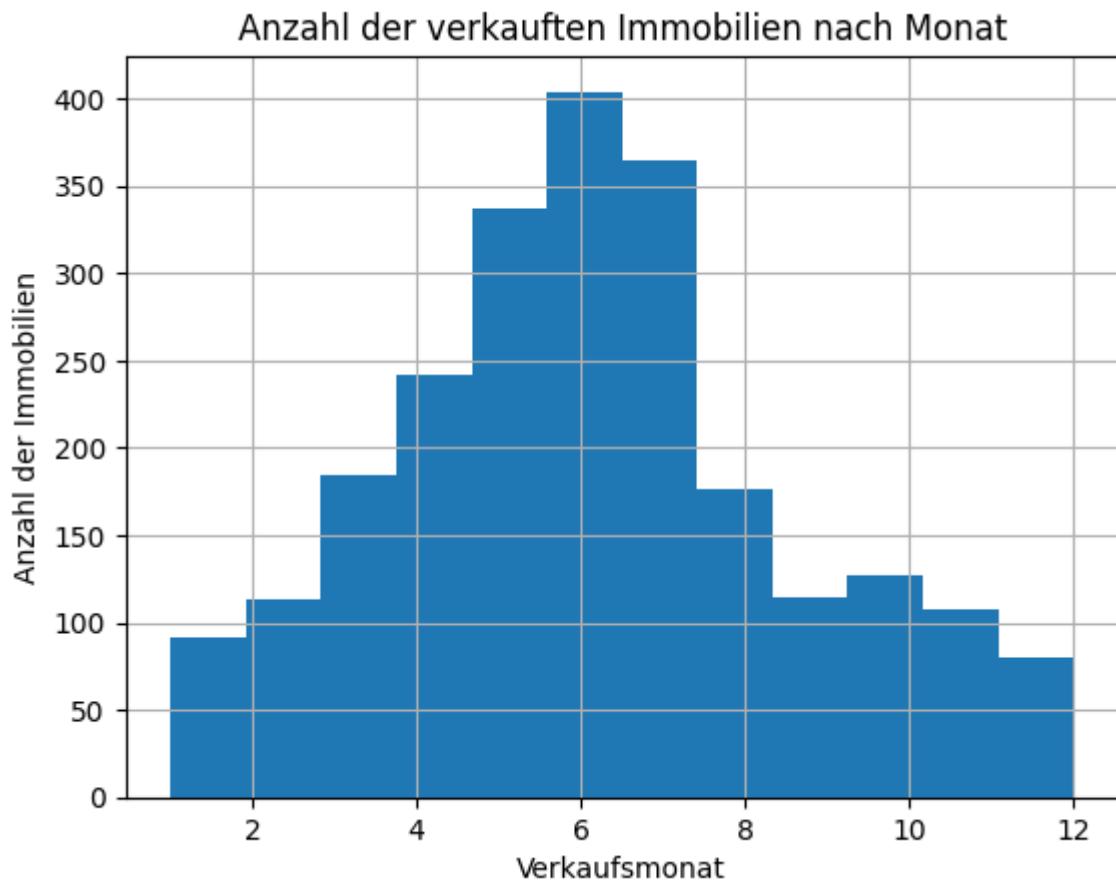
## Einfluss des Verkaufsmonats auf den Verkaufspreis

```
In [ ]: display_boxplot(attribute="Verkaufsmonat", xlabel="Monat des Verkaufs", title="des Verkaufsmo")
```



```
In [ ]: # Generate the histogram  
df_train["Verkaufsmonat"].hist(bins=12)
```

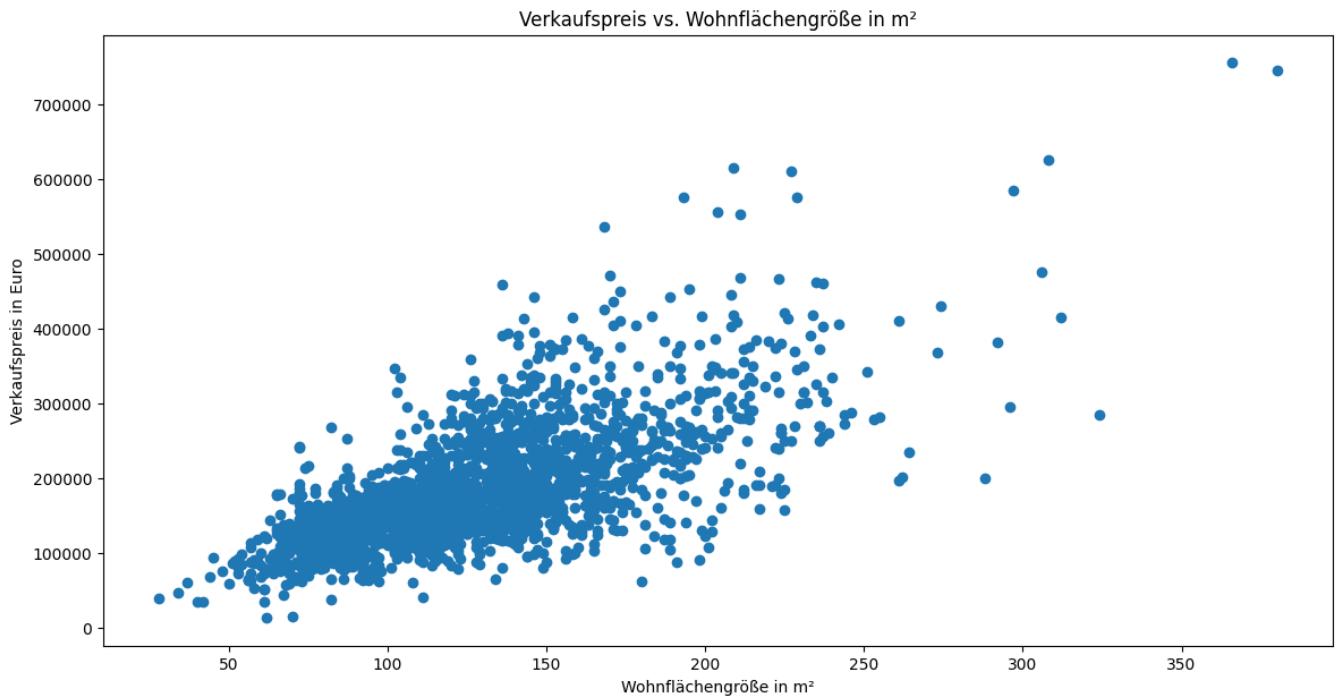
```
# Title and axis labels
plt.title("Anzahl der verkauften Immobilien nach Monat")
plt.xlabel("Verkaufsmonat")
plt.ylabel("Anzahl der Immobilien")
plt.show()
```



Es gibt keinen starken Zusammenhang zwischen Verkaufsmonat und Verkaufspreis. Anhand des Histogramms erkennt man, dass die meisten Verkäufe in der ersten Jahreshälfte, genauer gesagt von April bis Juli stattfinden. Jedoch werden im Median die Häuser am teuersten im Januar verkauft. Außerdem gibt es in den Sommermonaten Mai bis Juli die meisten Ausreißer. Man kann also davon ausgehen, dass in diesen Monaten eher sehr hochpreisige Häuser verkauft werden.

### Einfluss der Wohnflächengröße auf den Verkaufspreis

```
In [ ]: display_scatterplot(attribute="Wohnflaeche_qm", xlabel="Wohnflächengröße in m²")
```



Da sich die Wohnfläche unter anderem aus `EG_qm` und `Keller_qm` zusammensetzt, ist hier ein ähnlicher Zusammenhang zum Verkaufspreis zu erkennen. Die Größe der Wohnfläche korreliert stark mit dem Verkaufspreis. Je mehr Wohnfläche ein Haus bietet, desto teurer kann es verkauft werden.

## Zusammenfassung der Erkenntnisse

### Zentrale Erkenntnisse:

- Allgemein liegen 50% der Häuser innerhalb einer Preisspanne von 130.000 € bis 213.882 € mit einem durchschnittlichen Preis von 180.443 €.
- Durch die einzelnen Attributs-Plots konnten die Aussagen der Korrelationsmatrix nachvollzogen und bestätigt werden.
- Der Verkaufspreis einer Immobilie setzt sich primär (Korrelation > 0,5) aus folgenden numerischen Attributen zusammen:
  - `Wohnflaeche_qm` : 0,69
  - `Keller_qm` : 0,64
  - `EG_qm` : 0,62
  - `Garage_qm` : 0,62
  - `Garagen` : 0,62
  - `Baujahr` : 0,55
  - `Baeder` : 0,54
  - `Umgebaut` : 0,52
- Die Anzahl der Zimmer hat entgegen der Erwartung kaum einen Einfluss auf den Verkaufspreis.
- Durch den Bau von bis zu drei Garagenstellplätzen, kann der Verkaufspreis der Immobilie, verglichen mit dem Aufwand, relativ einfach gesteigert werden. Weitere Möglichkeiten zur Preisseigerung erweisen sich als aufwändiger (z.B. Vergrößerung der `Wohnfläche_qm` oder Bau weiterer `Baeder`) oder baulich nicht möglich (z.B. Vergrößerung der `Kellerhoehe`).
- Der Gesamteindruck ist ein recht subjektiver Wert und eignet sich nicht für die Vorhersage des Verkaufspreises.
- Generell gilt: Je neuer das Haus, desto teurer ist es. Aus den Daten geht hervor, dass dieser Effekt auch für Renovierungen gilt.

- Die meisten Häuser werden im ersten Halbjahr verkauft. Tendenziell werden die teuren Immobilien vor allem in den Sommermonaten (Mai bis Juli) verkauft.
- Durch die nähere Betrachtung der einzelnen Attribut-Werte in den Scatter- und Boxplots, konnten einige Ausreißer erkannt werden, die in der Data Preparation (Aufgabe 3) entfernt werden.

### **Expert\*innen-Rückfragen**

- Warum werden im Frühjahr bis Sommer mehr Häuser verkauft?
- Wie ist der leichte Aufwärtstrend des Verkaufspreises über die letzten 5 Verkaufsjahre zu erklären?
- Wie sehen die aktuellen Trends in der Immobilienbranche aus und wie stark wirken sie sich auf den Preis aus? Gibt es z.B. aktuell eine Nachfrage nach kleineren Wohnflächen um die Energiekosten zu reduzieren?
- Ab welcher Größenordnung sind Wohnflächen nicht mehr realistisch und sollten aus den Trainingsdaten entfernt werden?

### **Weitere Datenquellen:**

- Datenquelle mit Einträgen über das Durchschnittliche Einkommen bzw. die Soziale Schicht in den verschiedenen Bezirken
- Interessant wären auch Datenquellen zu Immobilienmärkten anderer Großstädte

## **3. Data Preparation (3 Punkte)**

**Aufgabenstellung:** Bereinigen Sie die Daten und führen Sie Feature Engineering durch. Hinweis: Kann bereits für Aufgabe 2 teilweise notwendig sein, dann kenntlich machen und zusammenfassend aufführen.

### **Fragestellungen der Data Preparation (Datenvorbereitung)**

- Können die Daten in der vorliegenden Form verwendet werden?
- Wie können diese vorverarbeitet werden, um sie zu verwenden?

### **Encoding kategorischer Daten**

Im Bereich der Datenvorbereitung ist die Kodierung kategorischer Daten eine zentrale Aufgabe. Die meisten Daten im realen Leben bestehen bekannterweise aus kategorischen String-Werten. Für die computergestützte Verarbeitung benötigen die Modelle jedoch Fließkommazahlen oder ganze Zahlen.

- **Kategoriale Daten:** gesammelte Informationen, die in Gruppen eingeteilt sind
  - Nominalskala: Diese Art von kategorialen Daten besteht aus der Namensvariablen ohne numerische Werte.
  - Ordinalskala: Diese Art von kategorialen Daten besteht aus einer Reihe von Ordnungen oder Skalen.
- **Label Encoding:** Diese Art der Kodierung wird verwendet, wenn die Variablen in den Daten ordinal sind. Bei der ordinalen Kodierung wird jede Bezeichnung in ganzzahlige Werte umgewandelt, und die kodierten Daten stellen die Reihenfolge der Bezeichnungen dar.
- **One-Hot Encoding:** Bei der One-Hot-Codierung erhält jede Kategorie einer kategorialen Variable eine neue Variable. Dabei wird jede Kategorie mit Binärzahlen abgebildet. Diese Art der Kodierung wird verwendet, wenn die Daten nominal sind. Neu erstellte binäre Merkmale können als Dummy-

Variablen betrachtet werden. Nach einer Hot-Codierung hängt die Anzahl der Dummy-Variablen von der Anzahl der in den Daten vorhandenen Kategorien ab.

```
In [ ]: # Show all columns which need to be encoded
columns_object = df_train.dtypes[df_train.dtypes == "object"].index
print(columns_object, "\n-----")

# Show all individual entry names of the found columns
for col in columns_object:
    print(f"{col.rstrip()}: {df_train.loc[:, col].unique()}\n")

# Create a copy of the Training DataFrame
df_train_cleaned = df_train.copy(deep=True)

Index(['Ausbaustufe', 'Kellerhoehe', 'Kellertyp', 'Lage'], dtype='object')
-----
Ausbaustufe: ['1 Ebene' '2 Ebenen' '3 Ebenen']

Kellerhoehe: ['Gut' 'Durchschnitt' '0' 'Sehr gut' 'Schlecht' 'Sehr Schlecht']

Kellertyp: ['Guter Wohnraum' 'Rohbau' 'Mittlerer Wohnraum' 'Niedrige Qualität' '0'
 'Freizeitraum' 'Kein Wohnraum']

Lage: ['Bezirk 19' 'Bezirk 16' 'Bezirk 18' 'Bezirk 8' 'Bezirk 17' 'Bezirk 6'
 'Bezirk 23' 'Bezirk 9' 'Bezirk 15' 'Bezirk 20' 'Bezirk 14' 'Bezirk 1'
 'Bezirk 24' 'Bezirk 21' 'Bezirk 22' 'Bezirk 7' 'Bezirk 4' 'Bezirk 25'
 'Bezirk 5' 'Bezirk 26' 'Bezirk 27' 'Bezirk 12' 'Bezirk 2' 'Bezirk 13'
 'Bezirk 10' 'Bezirk 3' 'Bezirk 11' '0']
```

## Ausbaustufe

Labelencoding durchführen, um die Ebenen-Angabe in ganzzahlige Werte umzuwandeln.

Ordinalskala von 1 bis 3

**3:** 3 Ebenen

**2:** 2 Ebenen

**1:** 1 Ebene

```
In [ ]: # Define mapping, apply Labelencoding and output for verification
mapping = {"1 Ebene": 1, "2 Ebenen": 2, "3 Ebenen": 3}
df_train_cleaned["Ausbaustufe"] = df_train_cleaned["Ausbaustufe"].replace(mapping)
df_train_cleaned[['Ausbaustufe']].head()
```

Out[ ]: Ausbaustufe

0	1
1	1
2	1
3	2
4	1

## Kellerhoehe

Labelencoding durchführen um die Bewertung der Kellerhöhe (Werte auf der Skala von "Sehr schlecht" bis "Sehr gut") in ganzzahlige Werte umzuwandeln. Die Einträge mit Nullwerten bedeuten, dass die Immobilie keinen Keller hat.

Ordinalskala von 0 bis 5:

- 5:** Sehr gut - ca. 250 cm
- 4:** Gut - ca. 225 cm
- 3:** Durchschnitt - ca. 200 cm
- 2:** Schlecht - ca. 175 cm
- 1:** Sehr schlecht - niedriger als 175 cm
- 0:** Keine Angabe - kein Keller

```
In [ ]: # Define mapping, apply Labelencoding and output for verification
mapping = {"0": 0, "Sehr Schlecht": 1, "Schlecht": 2, "Durchschnitt": 3, "Gut": 4, "Sehr gut": 5}
df_train_cleaned["Kellerhoehe"] = df_train_cleaned["Kellerhoehe"].replace(mapping)
df_train_cleaned[['Kellerhoehe']].head()
```

Out[ ]: **Kellerhoehe**

<b>0</b>	4
<b>1</b>	4
<b>2</b>	4
<b>3</b>	4
<b>4</b>	4

## Kellertyp

Für den Kellertyp wird ein Label Encoding durchgeführt. Das Mapping und die Rangfolge basieren dabei auf den Angaben in der Aufgabenstellung. Die Einträge mit Nullwerten bedeuten, dass die Immobilie keinen Keller hat.

Nominalskala von 0 bis 6:

- 6:** Guter Wohnraum
- 5:** Mittlerer Wohnraum
- 4:** Kein Wohnraum
- 3:** Freizeitraum
- 2:** Niedrige Qualität
- 1:** Rohbau
- 0:** Keine Angabe - kein Keller

```
In [ ]: # Define mapping, apply Labelencoding and output for verification
mapping = {"0": 0, "Rohbau": 1, "Niedrige Qualität": 2, "Freizeitraum": 3, "Kein Wohnraum": 4, "Guter Wohnraum": 5, "Mittlerer Wohnraum": 6}
df_train_cleaned["Kellertyp"] = df_train_cleaned["Kellertyp"].replace(mapping)
df_train_cleaned[['Kellertyp']].head()
```

Out[ ]: **Kellertyp**

<b>0</b>	6
<b>1</b>	6
<b>2</b>	1
<b>3</b>	6
<b>4</b>	6

## Lage

One-Hot Encoding durchführen, um die Lage in ganzzahlige Werte umzuwandeln. Die Immobilien, bei denen der Bezirk nicht angegeben ist (Nullwerte) werden entfernt, da sie sich sonst negativ auf die Vorhersage des Verkaufspreises auswirken.

```
In [ ]: # Delete entries with null values
# This deletion is completed by "selecting" rows where "Lage" numbers are non zero
df_train_cleaned = df_train_cleaned.loc[df_train_cleaned["Lage"] != '0']

# Define mapping and apply Labelencoding to get shorter column names
mapping = {
    "Bezirk 1": 1, "Bezirk 2": 2, "Bezirk 3": 3, "Bezirk 4": 4, "Bezirk 5": 5, "Bezirk 6": 6,
    "Bezirk 11": 11, "Bezirk 12": 12, "Bezirk 13": 13, "Bezirk 14": 14, "Bezirk 15": 15, "Bezirk 21": 21, "Bezirk 22": 22, "Bezirk 23": 23, "Bezirk 24": 24, "Bezirk 25": 25, "Bezirk 26": 26
}
df_train_cleaned["Lage"] = df_train_cleaned["Lage"].replace(mapping)

# Apply One-Hot Encoding and output for verification
encoder_lage = pd.get_dummies(df_train_cleaned["Lage"], prefix="Lage")
df_train_cleaned[encoder_lage.columns] = encoder_lage
#df_train_cleaned = df_train_cleaned.drop("Lage", axis=1) # optional: drop original "lage" column
df_train_cleaned.iloc[:, 20:47].head()
```

```
Out[ ]:   Lage_1  Lage_2  Lage_3  Lage_4  Lage_5  Lage_6  Lage_7  Lage_8  Lage_9  Lage_10 ... Lage_18  Lage_19
0      0      0      0      0      0      0      0      0      0      0      0      0      0      ...
1      0      0      0      0      0      0      0      0      0      0      0      0      0      ...
2      0      0      0      0      0      0      0      0      0      0      0      0      0      ...
3      0      0      0      0      0      0      0      0      0      0      0      0      0      ...
4      0      0      0      0      0      0      0      0      1      0      0      0      0      ...

5 rows × 27 columns
```

## Ausreißererkennung und Datenbereinigung

Bei Ausreißern handelt es sich um Werte, die nicht den Erwartungen entsprechen bzw. nicht zu den restlichen Werten der Verteilung passen. Viele Algorithmen reagieren schlecht auf Ausreißer. Es existiert keine klare Regel oder gar ein fester Schwellwert für die eindeutige Identifikation von Ausreißern. Welche Werte als Ausreißer gekennzeichnet und aus dem Datensatz entfernt werden, entscheidet der Data Scientist mithilfe von Fachwissen oder Experten.

- **univariater Ausreißer:** einzelner außergewöhnlich hoher oder niedriger Wert eines bestimmten erhobenen Merkmals
- **multivariater Ausreißer:** Datensatz, der mehrere für sich genommen normale Merkmalsausprägungen aufweist, die aber in ihrer Kombination äußerst ungewöhnlich sind (schwerer zu finden)

## AnzahlZimmer

Entferne alle Immobilien, die 0 Zimmer haben. Dabei kann es sich höchstens um Kellerwohnungen handeln, die nicht repräsentativ für den Datensatz sind.

```
In [ ]: # Delete entries with null values and display unique entries
df_train_cleaned = df_train_cleaned.loc[df_train_cleaned["AnzahlZimmer"] != 0]
df_train_cleaned.loc[:, "AnzahlZimmer"].unique()
```

```
Out[ ]: array([3, 2, 5, 4, 1, 6, 8], dtype=int64)
```

## Baeder

Entferne alle Immobilien, die 0 Bäder bzw. Toiletten im Erdgeschoss haben. Diese Einträge sind nicht repräsentativ für den Datensatz.

```
In [ ]: # Delete entries with null values and display unique entries
df_train_cleaned = df_train_cleaned.loc[(df_train_cleaned["Baeder"] != 0)]
df_train_cleaned.loc[:, "Baeder"].unique()
```

```
Out[ ]: array([2, 3, 1, 4, 6], dtype=int64)
```

## BaederKG

Es existiert genau ein Eintrag mit 3 Bädern im Kellergeschoss. Dieser wird entfernt.

```
In [ ]: # Delete entries with 3 bathrooms and display unique entries
df_train_cleaned = df_train_cleaned.loc[df_train_cleaned["BaederKG"] != 3]
df_train_cleaned.loc[:, "BaederKG"].unique()
```

```
Out[ ]: array([1, 0, 2], dtype=int64)
```

## Garagen

Es existiert genau ein Eintrag mit 5 Garagen. Dieser wird entfernt.

```
In [ ]: # Delete entries with 5 garages and display unique entries
df_train_cleaned = df_train_cleaned.loc[df_train_cleaned["Garagen"] != 5]
df_train_cleaned.loc[:, "Garagen"].unique()
```

```
Out[ ]: array([2, 3, 1, 0, 4], dtype=int64)
```

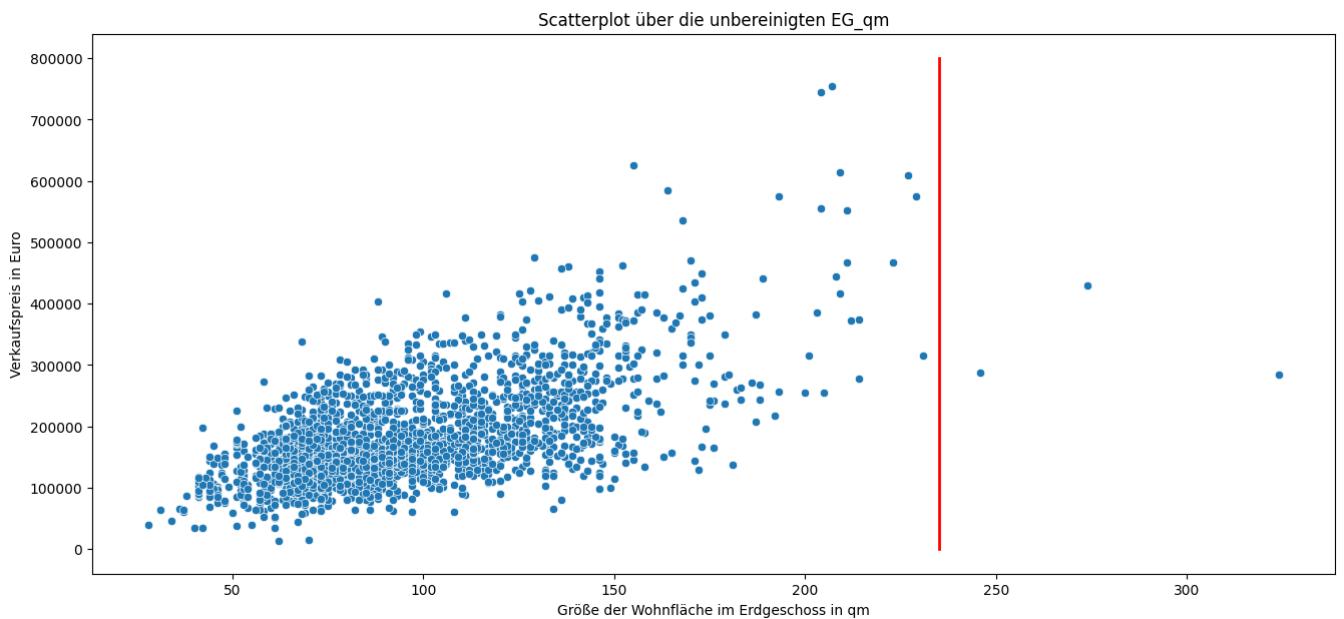
## EG\_qm

Anhand eines Scatterplots prüfen, ob es starke Ausreißer bei der Größe der Wohnfläche im Erdgeschoss gibt. Falls vorhanden werden diese Einträge aus dem Datensatz entfernt.

```
In [ ]: # Draw scatterplot with red line to illustrate the outliers
plt.figure(figsize=(16, 7))
sns.scatterplot(data=df_train, x="EG_qm", y="Z_Verkaufspreis")
plt.plot([235, 235], [800000, 0], linewidth=2, color="red")

plt.title("Scatterplot über die unbereinigten EG_qm")
plt.xlabel("Größe der Wohnfläche im Erdgeschoss in qm")
plt.ylabel("Verkaufspreis in Euro")
```

```
Out[ ]: Text(0, 0.5, 'Verkaufspreis in Euro')
```



```
In [ ]: # Delete all outliers from the data set and display unique entries
df_train_cleaned = df_train_cleaned.loc[df_train_cleaned["EG_qm"] <= 235]
df_train_cleaned.loc[:, "EG_qm"].unique()
```

```
Out[ ]: array([125, 170, 119, 64, 103, 89, 75, 70, 80, 97, 120, 91, 78,
   128, 117, 87, 62, 69, 115, 88, 68, 77, 94, 110, 106, 83,
   142, 90, 104, 82, 143, 63, 73, 189, 145, 109, 84, 122, 67,
   130, 99, 74, 81, 58, 155, 121, 66, 161, 44, 48, 57, 61,
   98, 131, 92, 85, 173, 93, 127, 154, 100, 108, 141, 60, 53,
   76, 126, 71, 102, 116, 111, 211, 175, 135, 164, 137, 124, 133,
   45, 72, 86, 193, 65, 118, 146, 96, 46, 140, 51, 79, 214,
   113, 151, 156, 138, 139, 129, 95, 107, 134, 101, 112, 132, 31,
   168, 201, 205, 114, 37, 152, 147, 149, 54, 59, 153, 212, 123,
   41, 179, 105, 158, 229, 165, 56, 47, 167, 223, 186, 34, 136,
   52, 36, 181, 144, 209, 163, 171, 150, 148, 207, 172, 203, 157,
   49, 187, 204, 50, 227, 200, 159, 192, 188, 43, 28, 42, 166,
   174, 183, 180, 176, 208, 38, 231, 55, 162], dtype=int64)
```

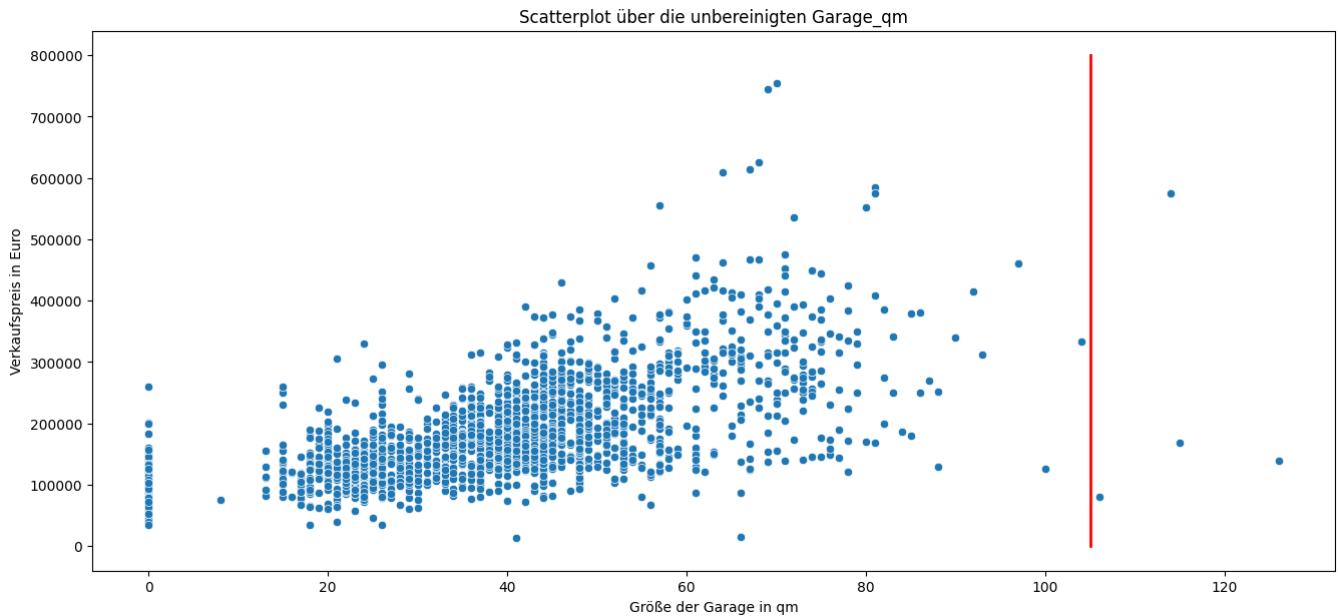
## Garage\_qm

Anhand eines Scatterplots prüfen, ob es starke Ausreißer bei der Größe der Garage gibt. Falls vorhanden werden diese Einträge aus dem Datensatz entfernt.

```
In [ ]: # Draw scatterplot with red line to illustrate the outliers
plt.figure(figsize=(16, 7))
sns.scatterplot(data=df_train, x="Garage_qm", y="Z_Verkaufspreis")
plt.plot([105, 105], [800000, 0], linewidth=2, color="red")

plt.title("Scatterplot über die unbereinigten Garage_qm")
plt.xlabel("Größe der Garage in qm")
plt.ylabel("Verkaufspreis in Euro")
```

```
Out[ ]: Text(0, 0.5, 'Verkaufspreis in Euro')
```



```
In [ ]: # Delete all outliers from the data set and display unique entries
df_train_cleaned = df_train_cleaned.loc[df_train_cleaned["Garage_qm"] <= 105]
df_train_cleaned.loc[:, "Garage_qm"].unique()
```

```
Out[ ]: array([ 49,  79,  40,  39,  27,  26,  53,  38,  24,  47,  41,  29,  33,
   42,  23,  32,  0,  34,  55,  44,  66,  19,  68,  54,  50,  72,
   28,  61,  22,  20,  52,  63,  37,  30,  43,  48,  25,  46,  18,
   31,  73,  21,  74,  57,  17,  15,  65,  51,  67,  69,  81,  71,
   35,  13,  70,  56,  78,  88,  45,  75,  36,  77,  58,  62,  64,
   90,  82,  59,  97,  86,  76,  80,   8,  60,  83,  16,  85,  92,
   93,  87, 104,  84], dtype=int64)
```

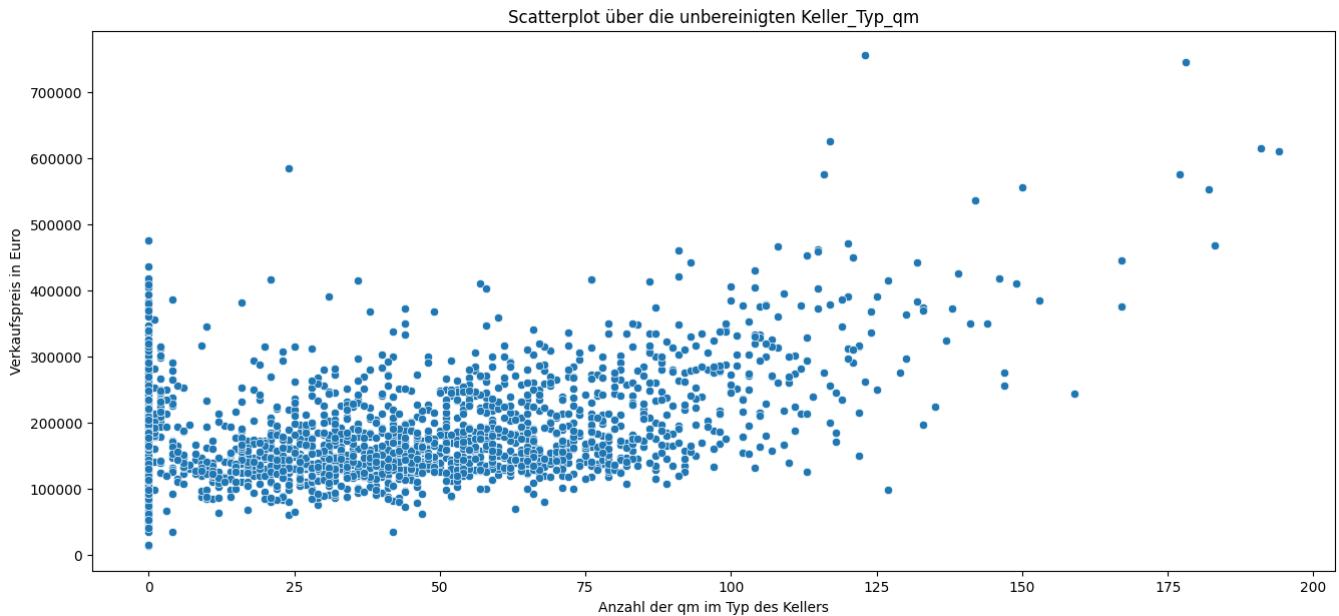
## Keller\_Typ\_qm

Anhand eines Scatterplots prüfen, ob es starke Ausreißer bei der Größe des Kellertyps (Keller\_Typ\_qm) gibt. Falls vorhanden werden diese Einträge aus dem Datensatz entfernt.

```
In [ ]: # Draw scatterplot with red line to illustrate the outliers
plt.figure(figsize=(16, 7))
sns.scatterplot(data=df_train, x="Keller_Typ_qm", y="Z_Verkaufspreis")
# plt.plot([160, 160], [800000, 0], linewidth=2, color="red")

plt.title("Scatterplot über die unbereinigten Keller_Typ_qm")
plt.xlabel("Anzahl der qm im Typ des Kellers")
plt.ylabel("Verkaufspreis in Euro")
```

```
Out[ ]: Text(0, 0.5, 'Verkaufspreis in Euro')
```



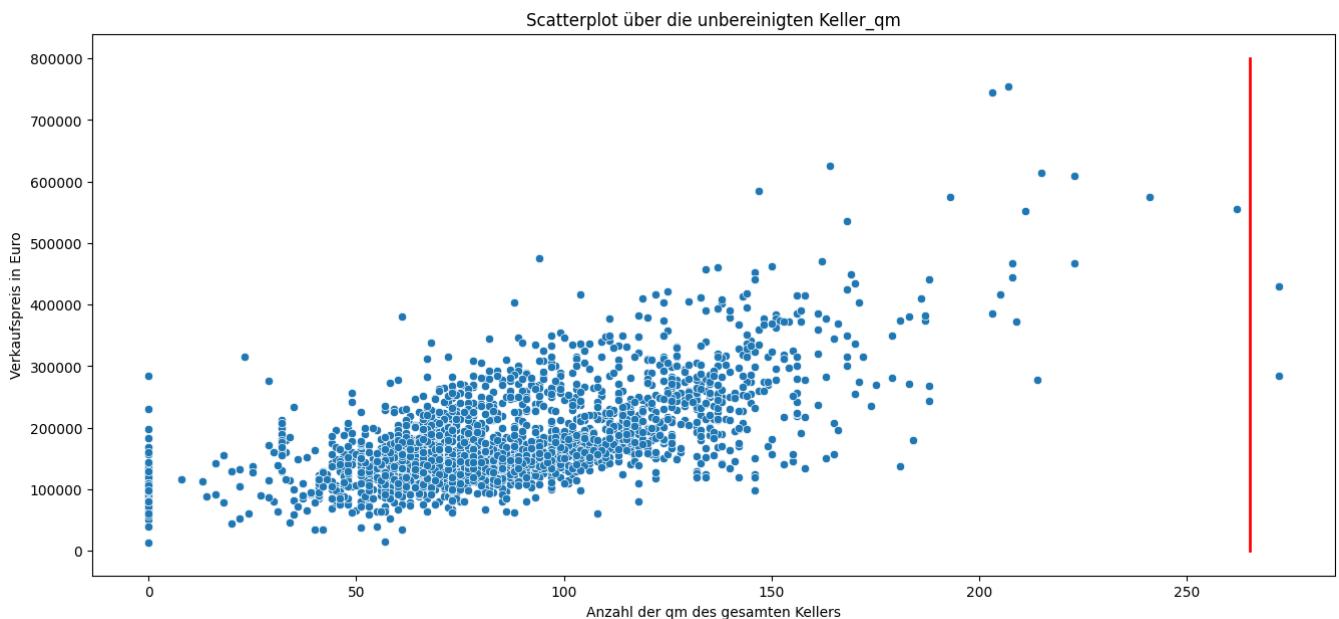
## Keller\_qm

Anhand eines Scatterplots prüfen, ob es starke Ausreißer bei der Größe des Kellers (Keller\_qm) gibt. Falls vorhanden werden diese Einträge aus dem Datensatz entfernt.

```
In [ ]: # Draw scatterplot with red Line to illustrate the outliers
plt.figure(figsize=(16, 7))
sns.scatterplot(data=df_train, x="Keller_qm", y="Z_Verkaufspreis")
plt.plot([265, 265], [800000, 0], linewidth=2, color="red")

plt.title("Scatterplot über die unbereinigten Keller_qm")
plt.xlabel("Anzahl der qm des gesamten Kellers")
plt.ylabel("Verkaufspreis in Euro")
```

Out[ ]: Text(0, 0.5, 'Verkaufspreis in Euro')



```
In [ ]: # Delete all outliers from the data set and display unique entries
df_train_cleaned = df_train_cleaned.loc[df_train_cleaned["Keller_qm"] <= 265]
df_train_cleaned.loc[:, "Keller_qm"].unique()
```

```
Out[ ]: array([116, 168, 119, 64, 103, 89, 75, 70, 78, 97, 56, 92, 76,
   0, 61, 41, 53, 115, 91, 80, 90, 68, 79, 94, 110, 82,
   69, 88, 129, 77, 133, 51, 73, 188, 145, 122, 57, 100, 125,
   99, 95, 58, 158, 67, 106, 32, 66, 60, 44, 81, 48, 54,
   83, 124, 72, 123, 74, 85, 120, 169, 93, 127, 154, 108, 84,
   35, 143, 102, 59, 98, 140, 111, 208, 118, 172, 135, 147, 137,
   62, 155, 18, 40, 139, 86, 126, 22, 144, 13, 96, 45, 146,
  113, 46, 132, 214, 71, 105, 151, 156, 138, 134, 29, 149, 33,
   87, 101, 130, 16, 142, 104, 109, 65, 49, 63, 20, 121, 112,
  107, 153, 114, 24, 36, 42, 47, 209, 179, 52, 136, 241, 30,
  165, 55, 223, 183, 128, 34, 39, 162, 50, 181, 25, 205, 157,
   43, 117, 37, 163, 171, 211, 150, 27, 148, 8, 207, 152, 38,
  203, 161, 186, 187, 141, 31, 262, 170, 215, 164, 174, 166, 184,
   14, 175, 131, 23], dtype=int64)
```

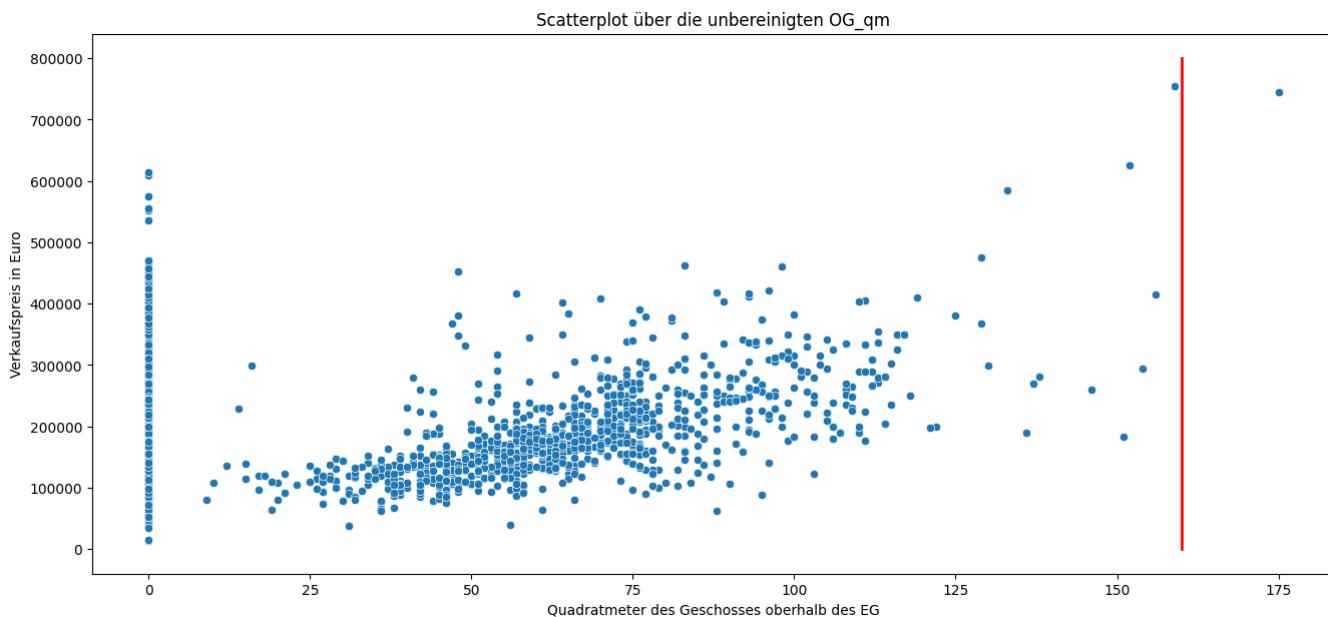
## OG\_qm

Anhand eines Scatterplots prüfen, ob es starke Ausreißer bei der Größe des Geschosses oberhalb des EG (OG\_qm) gibt. Falls vorhanden werden diese Einträge aus dem Datensatz entfernt.

```
In [ ]: # Draw scatterplot with red line to illustrate the outliers
plt.figure(figsize=(16, 7))
sns.scatterplot(data=df_train, x="OG_qm", y="Z_Verkaufspreis")
plt.plot([160, 160], [800000, 0], linewidth=2, color="red")

plt.title("Scatterplot über die unbereinigten OG_qm")
plt.xlabel("Quadratmeter des Geschosses oberhalb des EG")
plt.ylabel("Verkaufspreis in Euro")
```

```
Out[ ]: Text(0, 0.5, 'Verkaufspreis in Euro')
```



```
In [ ]: # Delete all outliers from the data set and display unique entries
df_train_cleaned = df_train_cleaned.loc[df_train_cleaned["OG_qm"] <= 160]
df_train_cleaned.loc[:, "OG_qm"].unique()
```

```
Out[ ]: array([  0,   73,   72,   65,   77,   52,   58,   75,   79,   83,   74,  119,   59,
       62,   31,   49,   71,   42,   96,   47,   48,   70,   63,   91,   39,   44,
       64,   54,  154,  100,  108,   61,   53,   87,   45,   29,   88,  133,   80,
       78,   35,   57,   51,   67,   69,   56,   76,   43,   68,   82,   25,   46,
       66,   50,  156,   26,   85,   86,  103,  110,   89,  105,   36,  113,   99,
       60,   55,   38,   84,   97,   81,   93,  106,   41,  111,   98,  136,   90,
      125,  137,  109,   17,   92,   28,   27,  138,  102,   32,   94,   37,   33,
      40,  159,  112,  116,   14,   95,  118,  104,  115,  130,   34,   19,  146,
      12,   10,   18,   15,   30,  129,  152,   23,  122,  117,   20,    9,  114,
     21,  151,  121,  107,  101], dtype=int64)
```

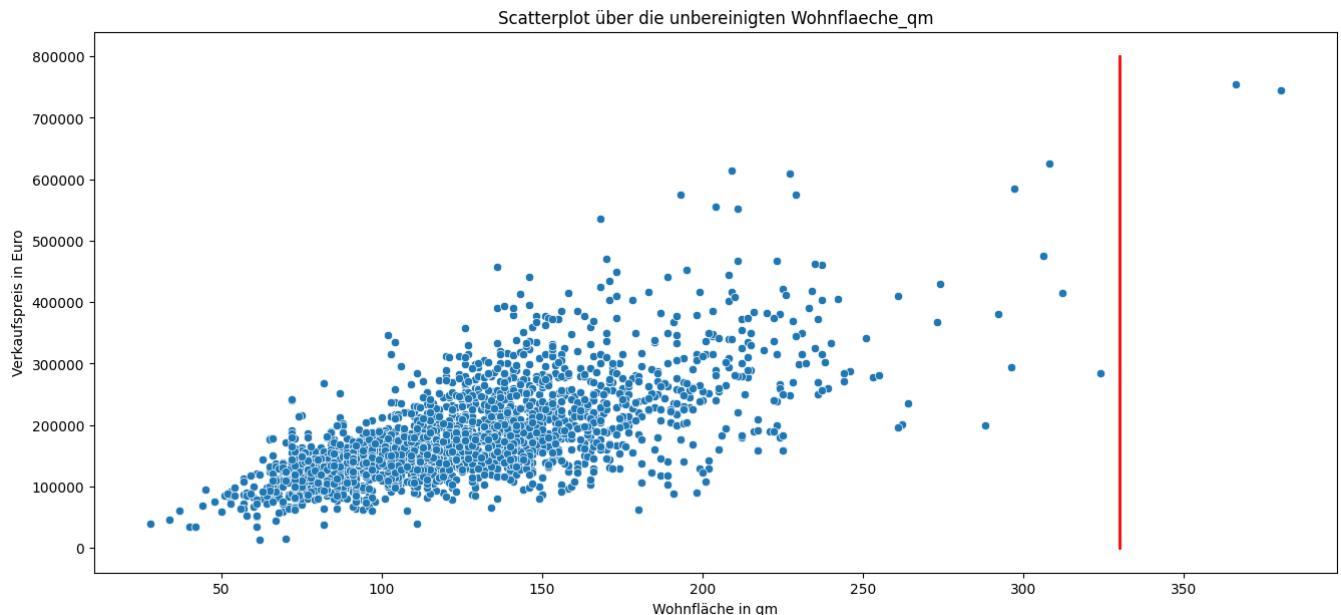
## Wohnflaeche\_qm

Anhand eines Scatterplots prüfen, ob es starke Ausreißer bei der Größe der gesamten Wohnfläche (Wohnflaeche\_qm) gibt. Falls vorhanden werden diese Einträge aus dem Datensatz entfernt.

```
In [ ]: # Draw scatterplot with red Line to illustrate the outliers
plt.figure(figsize=(16, 7))
sns.scatterplot(data=df_train, x="Wohnflaeche_qm", y="Z_Verkaufspreis")
plt.plot([330, 330], [800000, 0], linewidth=2, color="red")

plt.title("Scatterplot über die unbereinigten Wohnflaeche_qm")
plt.xlabel("Wohnfläche in qm")
plt.ylabel("Verkaufspreis in Euro")
```

```
Out[ ]: Text(0, 0.5, 'Verkaufspreis in Euro')
```



```
In [ ]: # Delete all outliers from the data set and display unique entries
df_train_cleaned = df_train_cleaned.loc[df_train_cleaned["Wohnflaeche_qm"] <= 330]
df_train_cleaned.loc[:, "Wohnflaeche_qm"].unique()
```

```
Out[ ]: array([125, 170, 119, 138, 103, 89, 75, 142, 145, 97, 198, 91, 78,
   128, 117, 140, 121, 69, 115, 80, 88, 141, 152, 166, 177, 110,
   157, 143, 261, 90, 181, 77, 63, 73, 189, 133, 149, 122, 225,
   227, 99, 106, 154, 123, 155, 136, 118, 131, 253, 82, 146, 109,
   94, 44, 81, 93, 111, 120, 296, 67, 98, 231, 92, 83, 139,
   191, 173, 151, 127, 186, 100, 108, 60, 76, 126, 169, 190, 211,
   102, 162, 220, 175, 135, 297, 74, 171, 203, 112, 104, 156, 137,
   159, 116, 45, 72, 150, 86, 165, 87, 96, 84, 194, 214, 113,
   124, 216, 312, 129, 172, 192, 158, 153, 212, 134, 101, 206, 130,
   217, 147, 107, 208, 144, 79, 132, 65, 204, 174, 193, 168, 160,
   201, 95, 61, 71, 180, 68, 205, 114, 37, 57, 223, 163, 235,
   53, 242, 179, 221, 262, 251, 105, 70, 229, 178, 187, 237, 292,
   85, 51, 219, 34, 236, 207, 183, 148, 213, 185, 234, 244, 255,
   164, 161, 209, 184, 64, 62, 48, 224, 202, 196, 226, 200, 56,
   66, 238, 230, 215, 167, 197, 50, 264, 239, 199, 58, 176, 188,
   195, 306, 228, 308, 240, 182, 233, 222, 52, 28, 42, 59, 288,
   273, 210, 54, 232], dtype=int64)
```

## Kellerhoehe

Es existieren nur zwei "Sehr schlecht" Werte. Diese werden entfernt.

```
In [ ]: # Delete entries with very poor basement height and display unique entries
df_train_cleaned = df_train_cleaned.loc[df_train_cleaned["Kellerhoehe"] != 1]
df_train_cleaned.loc[:, "Kellerhoehe"].unique()
```

```
Out[ ]: array([4, 3, 0, 5, 2], dtype=int64)
```

## Baujahr, Gesamteindruck, Umgebaut, Verkaufsjahr, Verkaufsmonat, Z\_Verkaufspreis

Die verbleibenden Spalten weisen keine Ausreißer, unzulässige Werte oder sonstige Auffälligkeiten auf.  
Daher sind keine Bereinigungen erforderlich.

## Multivariater Ausreißer: Umbau vor dem Baujahr des Hauses

Es sollten Datensätze entfernt werden, bei denen Immobilien früher umgebaut wurden als ihr Baujahr.  
Davon gibt es genau einen.

```
In [ ]: # Show records that match the rule and delete them
rule = (df_train_cleaned["Umgebaut"] < df_train_cleaned["Baujahr"])
display(df_train_cleaned[rule].loc[:, ["Umgebaut", "Baujahr"]])
df_train_cleaned.drop(df_train_cleaned[rule].index, inplace=True)
```

Umgebaut	Baujahr
655	2013

## Multivariater Ausreißer: Hohe Ausbaustufe ohne Obergeschoss-Quadratmeterangabe

Es sollten Datensätze entfernt werden, bei denen die Ausbaustufe größer als 1 ist (Obergeschosse vorhanden) aber der Wert bei OG\_qm gleich 0 ist. Davon gibt es 29 Stück.

```
In [ ]: # Show records that match the rule and delete them
rule = (df_train_cleaned["Ausbaustufe"] > 1) & (df_train_cleaned["OG_qm"] == 0)
display(df_train_cleaned[rule].loc[:, ["Ausbaustufe", "OG_qm"]])
df_train_cleaned.drop(df_train_cleaned[rule].index, inplace=True)
```

Ausbaustufe	OG_qm
<b>113</b>	2 0
<b>256</b>	2 0
<b>333</b>	2 0
<b>342</b>	2 0
<b>371</b>	2 0
<b>548</b>	2 0
<b>695</b>	2 0
<b>768</b>	2 0
<b>820</b>	2 0
<b>836</b>	2 0
<b>920</b>	2 0
<b>949</b>	2 0
<b>1030</b>	2 0
<b>1157</b>	2 0
<b>1243</b>	2 0
<b>1282</b>	2 0
<b>1348</b>	2 0
<b>1352</b>	2 0
<b>1492</b>	2 0
<b>1539</b>	2 0
<b>1559</b>	2 0
<b>1576</b>	2 0
<b>1620</b>	2 0
<b>1689</b>	2 0
<b>1767</b>	2 0
<b>1799</b>	2 0
<b>2043</b>	2 0
<b>2285</b>	2 0
<b>2334</b>	2 0

## Multivariater Ausreißer: Mehr Garagen als Zimmer

Es sollten Datensätze entfernt werden, bei denen die Anzahl der Garagen die Anzahl an Zimmern um den doppelten Wert übersteigt. Davon gibt es genau 2.

```
In [ ]: # Show records that match the rule and delete them
rule = (df_train_cleaned["AnzahlZimmer"] * 2 < df_train_cleaned["Garagen"])
display(df_train_cleaned[rule].loc[:, ["AnzahlZimmer", "Garagen"]])
df_train_cleaned.drop(df_train_cleaned[rule].index, inplace=True)
```

AnzahlZimmer	Garagen
1564	1
2266	3

## Multivariater Ausreißer: Mehr Bäder/Toiletten im Keller als in den Obergeschossen

Es sollten Datensätze entfernt werden, bei denen die Anzahl der Bäder im Keller höher ist als in den Obergeschossen. Davon gibt es genau 3.

```
In [ ]: # Show records that match the rule and delete them
rule = (df_train_cleaned["Baeder"] < df_train_cleaned["BaederKG"])
display(df_train_cleaned[rule].loc[:, ["Baeder", "BaederKG"]])
df_train_cleaned.drop(df_train_cleaned[rule].index, inplace=True)
```

Baeder	BaederKG
557	1
2222	2
2278	2

## Multivariater Ausreißer: Immobilien mit 6 Bädern im Obergeschoss

Es sollten Datensätze entfernt werden, bei denen die Anzahl der Bäder im Obergeschoss sechs ist. Davon gibt es genau 4.

Anmerkung: Es gibt keine Einträge mit 5 Bädern im Obergeschoss.

```
In [ ]: # Show records that match the rule and delete them
rule = (df_train_cleaned["Baeder"] == 6)
display(df_train_cleaned[rule].loc[:, ["Baeder"]])
df_train_cleaned.drop(df_train_cleaned[rule].index, inplace=True)
```

Baeder
244
599
1146
1354

## Multivariater Ausreißer: Immobilien mit 8 Zimmern

Es sollten Datensätze entfernt werden, bei denen die Anzahl der Zimmer acht ist. Davon gibt es genau 1.

Anmerkung: Es gibt keine Einträge mit 7 Zimmern.

```
In [ ]: # Show records that match the rule and delete them
rule = (df_train_cleaned["AnzahlZimmer"] == 8)
display(df_train_cleaned[rule].loc[:, ["AnzahlZimmer"]])
df_train_cleaned.drop(df_train_cleaned[rule].index, inplace=True)
```

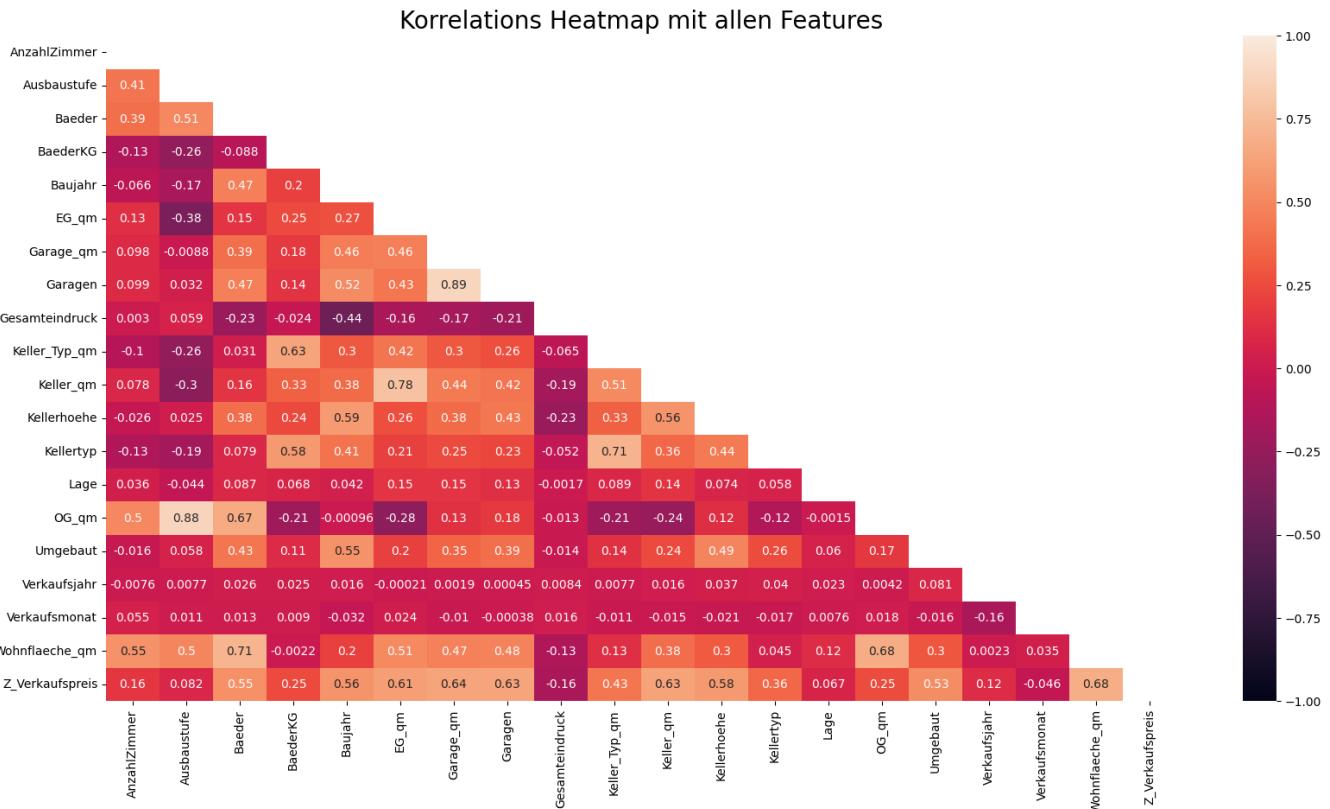
AnzahlZimmer
1808

## Korrelationsmatrix nach Abgeschlossener Data Preparation

Anhand der Korrelationsmatrix aus Aufgabe 2 soll abschließend untersucht werden, wie sich die Data Preparation Schritte auf die Qualität der Korrelation zwischen Attributen und Verkaufspreis ausgewirkt haben.

```
In [ ]: # Draw a correlation matrix heatmap and mask out the upper triangle
plt.figure(figsize=(20, 10))
mask = np.triu(np.ones_like(df_train_cleaned.iloc[:, 0:20].corr(), dtype=bool))
heatmap = sns.heatmap(df_train_cleaned.iloc[:, 0:20].corr(), mask=mask, vmin=-1, vmax=1, annot=True)
plt.title('Korrelations Heatmap mit allen Features', fontdict={'fontsize': 20})
```

Out[ ]: Text(0.5, 1.0, 'Korrelations Heatmap mit allen Features')



- Durch das Encoding der kategorischen Daten kamen neue Features mit einer hohen Korrelation zum Verkaufspreis hinzu (z.B. Kellerhoehe ).
- Durch die Ausreißer-Erkennung und Datenbereinigung konnten die Korrelationen einiger bestehender Features mit dem Verkaufspreis gesteigert werden.

## 4. Modeling – Regression mit Inferenz (3 Punkte)

**Aufgabenstellung:** Führen Sie mit einem geeigneten Verfahren der linearen Regression eine Vorhersage des Preises ( $Z_{\text{Verkaufspreis}}$ ) durch. Ggf. brauchen Sie dafür mehrere Versionen der „einfachen“ Regressionslösungen, um eine akzeptable Performance zu erreichen. Erklären Sie wichtige identifizierten Zusammenhänge menschenverständlich als Text (z. B. „Eine Haustür erhöht den Preis um 2,75 EUR.“).

```
In [ ]: from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.model_selection import cross_validate
from sklearn.metrics import r2_score
```

## Cross Validierung:

In dieser Funktion wird die Cross-Validation des Models mit 10-fach Faltung ausgeführt. Der Scoring="r2" Parameter gibt an, welcher Wert für die Bewertung verwendet werden soll. R<sup>2</sup> ist zwar Standard, aber mit expliziter Angabe ist es eindeutiger und besser verständlich. Die Funktion gibt danach den Score für jede einzelne Kreuzvalidierung aus und berechnet den Durchschnitt aller Durchläufe. Weitere Infos folgen im Abschnitt "Die Auswahl der reduzierten Features".

```
In [ ]: def cross_validate_model(model, x, y):
    show_decimals = 8
    validation = cross_validate(model, x, df_train_cleaned[y], cv=10, scoring="r2")
    print("Cross-Validation:")
    for score in validation['test_score']:
        print('\t' + str(np.round(score, decimals=show_decimals)))
    print('Mean of cross validation set: ' + str(np.round(np.mean(validation['test_score']), 2)))
    print('With a standard deviation of: ' + str(np.round(np.std(validation['test_score']), 2)), d
```

## Regression

Es wird zuerst das Model kreuzvalidiert, danach wird es gefittet. Das fitten ist notwendig, um händisch den R<sup>2</sup> Wert des Modells zu berechnen, wenn es auf den gesamten Daten trainiert wird. Außerdem ist das fitten notwendig, um später in get\_coefs() Wertsteigerung pro einer Einheit eines Features berechnen zu können.

```
In [ ]: def do_regression(model, x, y):
    cross_validate_model(model, x, y_fields)
    model.fit(x, y)
    y_predict = pd.DataFrame(model.predict(x).astype(int), columns=['Prediction'])
    r2 = r2_score(y_true=df_train_cleaned['Z_Verkaufspreis'], y_pred=y_predict)
    print("R^2 of whole set: " + str(r2))
    return model
```

## Berechnung der Abhängigkeiten eines einzelnen Features

Hier wird die Wertsteigerung, welche eine Einheit eines Features ausmacht, berechnet und ausgegeben. Es werden alle Features verwendet, um einen gesamten Überblick zu gewähren. Zur genaueren Auswertung folgt unten ein weiter Text.

```
In [ ]: def get_coefs(model, x, y):
    coefficients = pd.concat([pd.DataFrame(x.columns), pd.DataFrame(np.transpose(model.coef_))])
    print(coefficients)
```

## Die Auswahl der reduzierten Features

Um die Feature-Kombinationen untereinander vergleichen zu können, wurde die Funktion cross\_validate\_model verwendet. Diese crossvalidiert mit einer 10-fachen Faltung. Die jeweiligen R<sup>2</sup> Werte wurden dann im Durchschnitt genommen. Die einzelnen Werte jeder Faltung geben an, wie sehr die Daten sich unterscheiden. Man kann daraus also ablesen, wie sehr die Einteilung in Trainings- und Testdaten sich auf die Genauigkeit der Prognose auswirken. Der Durchschnitt daraus ist dann sinnvoll, um jede entstandene Einteilungskombination der Daten gleich zu gewichten. Gegenüber dem R<sup>2</sup>-Wert, der in do\_regression auf das gesamte Model berechnet wird hat dies als Vorteil, dass bei der Crossvalidierung mit Daten getestet wird, welche das Modell nicht zum Trainieren verwendete. Bei der

simpfen R<sup>2</sup> Berechnung in do\_regression() ist dies nicht so, da alle Daten für das Training verwendet wurden. Darum ist der Durchschnitt der Cross-Validierung als Parameter für die Feature Auswahl gewählt worden.

Das Vorgehen für die Feature-Reduktion war, mit allen Features anzufangen (Ohne Nominalskalen) und dann vereinzelte Features wegzulassen. Für diese Auswahl wurde sich grob an der Korrelationsheatmap mit allen Features orientiert (s.o.). Wirkte sich ein Feature sehr auf die Genauigkeit aus, wurde es behalten. Dokumentiert sind nachfolgend einige Zwischenstufen, um einen groben Verlauf darzustellen. Theoretisch wäre auch möglich, jede Kombination auszuprobieren und die Kombination mit der höchsten Genauigkeit zu wählen. Da dies 262143 Kombinationen darstellt, wurde auf diesen Rechenaufwand verzichtet (Alle Binomialkoeffizienten von 18 über 1 bis hin zu 18 über 18 aufsummiert).

Es wurde dabei von Anfang an der Verkaufspreis aus den Daten entfernt, da das Modell diesen vorher sagen soll und nicht zur Vorhersage verwenden soll.

- Alle Features:

["AnzahlZimmer", "Ausbaustufe", "Baeder", "BaederKG", "Baujahr", "EG\_qm", "Garage\_qm", "Garagen", "Gesamtein  
"Keller\_Typ\_qm", "Keller\_qm", "Kellerhoehe", "Kellertyp", "Lage", "OG\_qm", "Umgebaut", "Verkaufsjahr", "Verkaufsr

`0.80156342`

- Ohne Nominalskalen:

["AnzahlZimmer", "Ausbaustufe", "Baeder", "BaederKG", "Baujahr", "EG\_qm", "Garage\_qm", "Garagen", "Gesamtein  
"Keller\_Typ\_qm", "Keller\_qm", "Kellerhoehe", "Kellertyp",  
, "OG\_qm", "Umgebaut", "Verkaufsjahr", "Verkaufsmonat", "Wohnflaeche\_qm"]:

`0.79685377`

- ["AnzahlZimmer", "Baeder", "Baujahr", "EG\_qm", "Garage\_qm", "Garagen", "Keller\_Typ\_qm", "Keller\_qm", "Kell

0.78037179

- ["AnzahlZimmer", "Baeder", "Baujahr", "Garagen", "Keller\_Typ\_qm", "Keller\_qm", "OG\_qm", "Umgebaut", "Woh

0.77778304

- ["AnzahlZimmer", "Baujahr", "Garagen", "Keller\_Typ\_qm", "Keller\_qm", "Umgebaut", "Wohnflaeche\_qm"]:

0.77739578

Das Feature 'Lage' wurde gestrichen, da es sich hier um eine Nominalskala handelt, es kann also an der Skala nicht abgelesen werden, was besser ist. Da lineare Regression allerdings eine Rangordnung benötigt, (also mind. Ordinalskala) kann dieser Wert hierfür nicht verwendet werden. In "Alle Features" ist er für die Vollständigkeit allerdings trotzdem enthalten.

Es wurde sich schlussendlich für die letzten Features entschieden, da sie nicht weiter reduzierbar sind, ohne die Genauigkeit zu sehr einzuschränken. Als Bemerkung hierzu kann noch gesagt werden, dass dies vermutlich nicht die beste, aber eine sehr gute Selektion ist (da sie mit wenig Features auskommt).

```
In [ ]: x_fields_all = ["AnzahlZimmer", "Ausbaustufe", "Baeder", "BaederKG", "Baujahr", "EG_qm", "Garage_qm",  
                      "Kellerhoehe", "Kellertyp", "Lage", "OG_qm", "Umgebaut", "Verkaufsjaehr", "Verkaufsm  
x_fields_reduced = ["AnzahlZimmer", "Baujahr", "Garagen", "Keller_Typ_qm", "Keller_qm", "Umgebaut"  
y_fields = "Z_Verkaufspreis"  
x_all = df_train_cleaned[x_fields_all]  
x_reduced = df_train_cleaned[x_fields_reduced]  
y = df_train_cleaned[y_fields]  
  
_ = do_regression(LinearRegression(), x_reduced, y)  
  
Cross-Validation:  
0.75974436  
0.77755587  
0.77246763  
0.8265534  
0.79764716  
0.78171757  
0.77757536  
0.77925664  
0.74898027  
0.75681353  
Mean of cross validation set: 0.77783118  
With a standard deviation of: 0.0210091  
R^2 of whole set: 0.7822593404038786
```

## Auswertung Regression mit reduzierten Features:

Die Werte der einzelnen Aufteilung der Cross-Validierung ähneln sich recht stark, die Standard-Abweichung ist relativ niedrig. Dadurch kann der Durchschnitt als aussagekräftig angesehen werden.

```
In [ ]: model_for_coefs = do_regression(LinearRegression(), x_all, y)  
  
Cross-Validation:  
0.79960074  
0.81801552  
0.7793978  
0.83869302  
0.82481669  
0.79050262  
0.77595845  
0.81621057  
0.77765918  
0.78757471  
Mean of cross validation set: 0.80084293  
With a standard deviation of: 0.02107026  
R^2 of whole set: 0.8082182282586385
```

## Auswertung Regression mit allen Features:

Die Werte sind genau wie bei der Regression mit reduzierten Features ohne starke Ausreißer um den Durchschnitt verteilt und die Standard-Abweichung ist auch relativ gering. Die ungefähren 2,5% Einbuße in der Leistung werden hingenommen im Austausch dafür, die Features von 19 auf 7 reduziert zu haben.

## Unterschied von allen Features auf reduzierte Features:

```
In [ ]: get_coefs(model_for_coefs, x_all, y)
```

		0
0	AnzahlZimmer	-10554.322039
1	Ausbaustufe	-1526.172840
2	Baeder	-2148.240667
3	BaederKG	1992.051578
4	Baujahr	538.723456
5	EG_qm	659.492957
6	Garage_qm	598.613644
7	Garagen	1200.341278
8	Gesamteindruck	9842.043315
9	Keller_Typ_qm	235.991827
10	Keller_qm	366.605781
11	Kellerhoehe	6213.673213
12	Kellertyp	-476.818357
13	Lage	-816.349107
14	OG_qm	594.040206
15	Umgebaut	414.885788
16	Verkaufsjaehr	5198.348060
17	Verkaufsmonat	-842.774424
18	Wohnflaeche_qm	396.922499

## Auswertung Gewichtung einzelner Features:

In dieser Metrik kann man nun die Gewichtung einzelner Features im erhaltenen Modell erkennen. Man kann bspw. sagen eine Garage erhöht den durchschnittlichen Verkaufspreis um 1333€. Auffällige Werte hier sind:

- **Anzahl Zimmer:** Dieser relativ hohe Wert pro Zimmer wirkt kontraintuitiv, da eigentlich ein Haus mit mehr Zimmern teurer sein müsste. Aus dem Plot geht hervor, dass Häuser mit vielen Zimmern (5 und 6) geringere Maximalpreise erzielen. Dadurch diese negative Korrelation.
- **Ausbaustufe:** Hier ziehen die Häuser mit Ausbaustufe 3 die Vorhersage so stark runter, dass eine negative Korrelation entsteht.
- **Baeder:** Dieser negative Koeffizient ist unverständlich. Die Korrelation mit 0.55 ist gegeben, alle Betrachtungen der Anzahlsverteilung und Preisverteilung implizieren einen positiven Zusammenhang und einen positiven Koeffizienten. Eine Vermutung ist, dass ein unbekannter Zusammenhang mit den anderen Parameter für die Koeffizientenberechnung diesen Wert so negativ werden lassen.
- **Kellertyp:** Da es sich bei diesem Feature um keine Ordinalskala handelt, ist der Wert nicht aussagekräftig.
- **Lage:** Da es sich bei diesem Feature um keine Ordinalskala handelt, ist der Wert nicht aussagekräftig.
- **Verkaufsmonat:** Es ist zwar ein Zusammenhang (auch im Diagramm) zu erkennen, allerdings wird hier nicht weiter darauf eingegangen, da vermutet wird, dass es sich um keinen kausalen Zusammenhang zwischen Verkaufsmonat und Verkaufspreis handelt. (Siehe auch Korrelationsmatrix oben, sie Korrelieren mit -0.043)

```
In [ ]: sns.countplot(data=df_train_cleaned, x="Baeder")
plt.title("Verteilung der Garagenanzahl über den Datensatz")
plt.show()

plt.figure(figsize=(20, 10))
```

```

plt.subplot(2,2,1)
sns.violinplot(data=df_train_cleaned, x="AnzahlZimmer", y="Z_Verkaufspreis")
plt.title("Violinplot Anzahl Zimmer gegen Verkaufspreis")

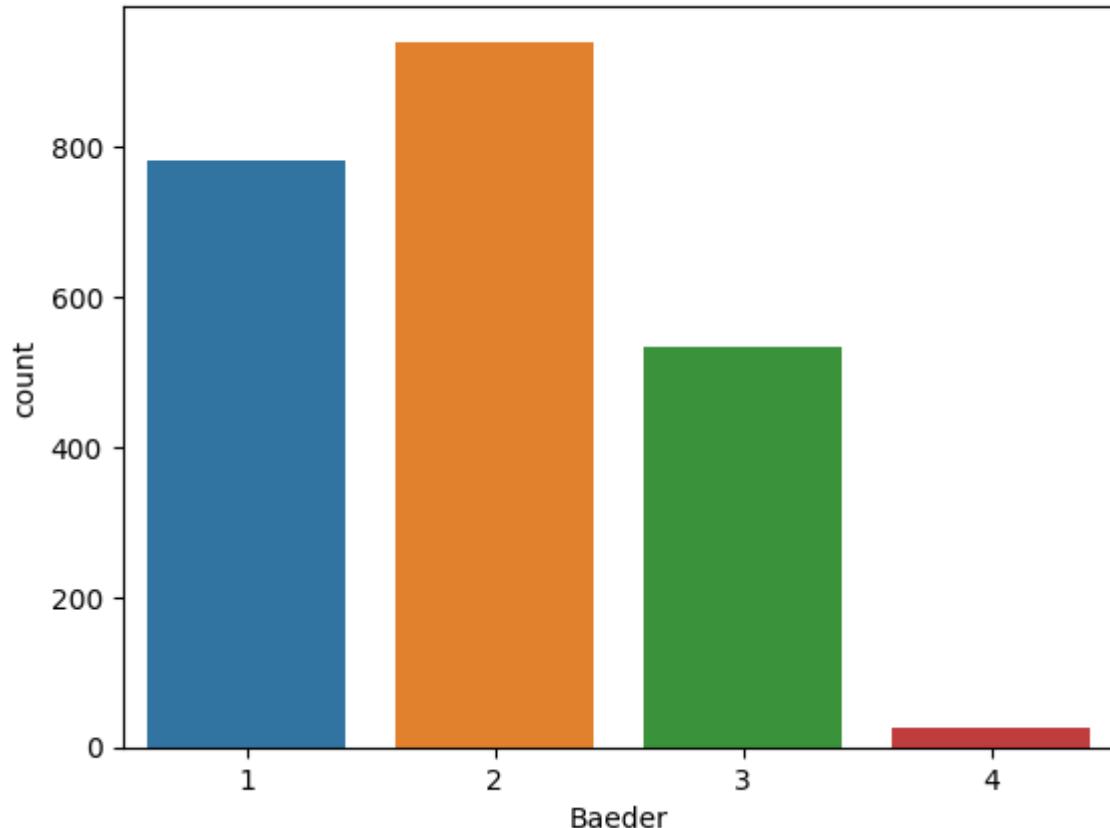
plt.subplot(2,2,2)
sns.violinplot(data=df_train_cleaned, x="Ausbaustufe", y="Z_Verkaufspreis")
plt.title("Violinplot Ausbaustufe gegen Verkaufspreis")

plt.subplot(2,2,3)
sns.violinplot(data=df_train_cleaned, x="Baeder", y="Z_Verkaufspreis")
plt.title("Violinplot Anzahl Baeder gegen Verkaufspreis")

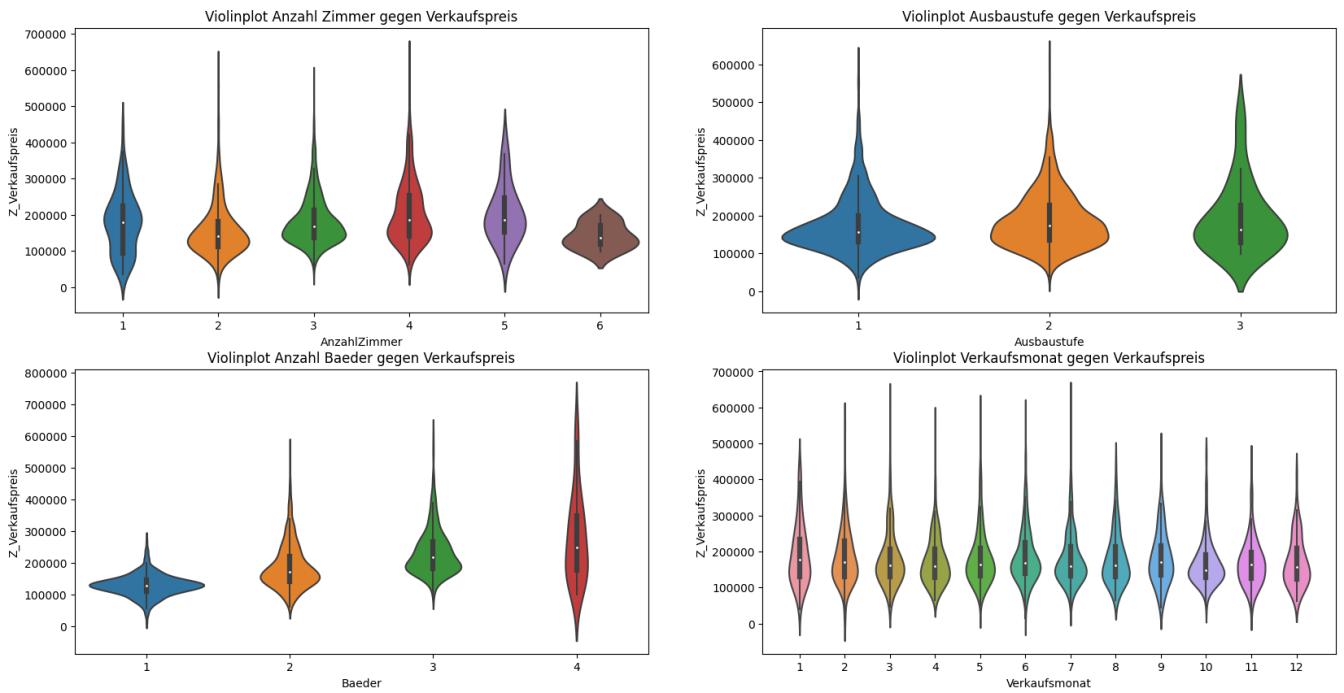
plt.subplot(2,2,4)
sns.violinplot(data=df_train_cleaned, x="Verkaufsmonat", y="Z_Verkaufspreis")
plt.title("Violinplot Verkaufsmonat gegen Verkaufspreis")
plt.plot()

```

Verteilung der Garagenanzahl über den Datensatz



Out[ ]: []



## 5. Modeling und Evaluation (6 Punkte)

**Aufgabenstellung:** Vergleichen und optimieren Sie ein oder mehrere weitere Verfahren zur Vorhersage des Verkaufspreises. Gehen Sie vor wie in der Vorlesung gelehrt mit Trainings- und Validierungsdaten (80-20). Optimieren Sie Ihre Vorhersage wenn sinnvoll.

Geben Sie für den Trainings- und Validierungsdatensatz die Zielwerte R2, MSE, RMSE, MAPE, MAX aus. Dokumentieren Sie dies auch.

Interpretieren Sie das Ergebnis und den Einfluss der Features (falls möglich). Untersuchen Sie Varianz und Verzerrung in der Vorhersage.

Schreiben Sie in die `data_for_test.csv` die auf Basis Ihres besten Modells vorhergesagte Werte in eine neue Spalte und geben Sie diese Datei mit ab. (Hinweis: Sortieren Sie nicht um).

### Splitten des Datensatzes in Trainings- und Validierungsdaten

Die gelabelten Daten (`data_for_training.csv`) werden im Verhältnis 80:20 in Trainings- und Validierungsdaten aufgeteilt.

- **Training:** Daten zum Lernen der Muster und Zusammenhänge für die Erzeugung der Modelle
- **Validierung:** Messung des Fehlers der Modellkandidaten und Auswahl des besten Modells
- **Test:** Überprüfung der Qualität des gewählten Modells am Ende

Die Parameterauswahl erfolgt anhand der Ergebnisse aus den Aufgaben 3 und 4. Es werden folgende Attribute für die Vorhersage des Verkaufspreises gewählt:

```
"AnzahlZimmer", "Baeder", "Baujahr", "EG_qm", "Garage_qm", "Keller_Typ_qm",
"Keller_qm", "Wohnflaeche_qm", "Umgebaut", "Verkaufsjahr", "Verkaufsmont",
"Z_Verkaufspreis"
```

```
In [ ]: from sklearn.model_selection import train_test_split

# Select parameters
selected_columns_old = ["Baeder", "Baujahr", "EG_qm", "Garage_qm", "Garagen", "Keller_qm", "Wohnflaeche_qm"]
selected_columns = ["AnzahlZimmer", "Baeder", "Baujahr", "EG_qm", "Garage_qm", "Keller_Typ_qm"]
df_train_selected_features = df_train_cleaned.loc[:, selected_columns]
```

```

# Split the data in input features (X) and target values (Y)
X = df_train_selected_features.drop(columns=["Z_Verkaufspreis"])
Y = df_train_selected_features["Z_Verkaufspreis"]

# Split data set into training and validation data
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

```

## Vergleich von Verfahren zur Vorhersage des Verkaufspreises

Nun werden verschiedene Verfahren zur Vorhersage des Verkaufspreises verglichen. Das Verfahren mit der besten Vorhersage wird dann weiter optimiert und auf die Testdaten (`data_for_test.csv`) angewendet. Diese Verfahren werden im Folgenden angewendet und verglichen:

- Random Forest
- Extra Trees
- Gradient Boosting
- Linear Support Vector Regressor
- Passive Aggressive Regression

Für alle Verfahren werden auch die zugehörigen Fehlermetriken berechnet. Mit Hilfe von Fehlermetriken ist es möglich, die Güte eines Modells zu quantifizieren. Durch sie ist es möglich, die Eignung von Modellen für bestimmte Aufgaben objektiv zu bewerten. Dazu werden die folgenden Fehlermetriken herangezogen:

- **Coefficient of determination (R2):** R2 ist der Prozentsatz der Streuung in der Antwortvariablen. Also ein statistisches Maß dafür, wie dicht die Daten an der angepassten Regressionslinie liegen.
  - Je höher das R-Quadrat, desto besser ist das Modell an die Daten angepasst.
- **Mean Squared Error (MSE):** Der MSE bewertet die Qualität des Prognosemodells oder Prädiktors. Er beinhaltet sowohl die Varianz (die Streuung der vorhergesagten Werte zueinander) als auch die Verzerrung (der Abstand des vorhergesagten Wertes von seinem wahren Wert).
  - Je näher an Null, desto besser
- **Root Mean Squared Error (RMSE):** RMSE ist eine Erweiterung von MSE und ist definiert als die Quadratwurzel des mittleren quadratischen Fehlers. Der RMSE-Wert ist im Vergleich zum MSE in der gleichen Einheit wie der prognostizierte Wert.
  - kleinere Werte sind besser
- **Mean Absolute Percentage Error (MAPE):** Der MAPE ist der prozentuale Mittelwert der absoluten Differenz zwischen prognostizierten Werten und wahren Werten, geteilt durch den wahren Wert.
  - kleinere Werte sind besser
- **Max Error (MAX):** MAX berechnet den maximalen Restfehler, eine Metrik, die den größten Fehler zwischen dem vorhergesagten Wert und dem wahren Wert erfasst.
  - kleinere Werte sind besser

Die Wichtigkeit eines Merkmals (feature importance) wird als die normierte Gesamtreduktion des Kriteriums durch dieses Merkmal berechnet. Sie ist auch als Gini-Bedeutung bekannt. Je höher, desto wichtiger ist das Merkmal.

```

In [ ]: from sklearn.metrics import r2_score, mean_absolute_percentage_error, mean_squared_error, max_
         # Function to calculate and output the error metrics
         def output_error_metrics(y_test, y_prediction):
             print(f"R2:\t{np.around(r2_score(y_test, y_prediction), decimals=4)}")
             print(f"MSE:\t{np.around(mean_squared_error(y_test, y_prediction), decimals=4)}")

```

```

print(f"RMSE:\t{np.around(np.sqrt(mean_squared_error(y_test, y_prediction)), decimals=4)}")
print(f"MAPE:\t{np.around(mean_absolute_percentage_error(y_test, y_prediction), decimals=4)}")
print(f"MAX:\t{np.around(max_error(y_test, y_prediction), decimals=4)}")

# Function to output scatterplots with regression lines
def output_regression_scatterplots(y_prediction):
    fig, axes = plt.subplots(1, len(selected_columns)-1, figsize=(35, 5), sharey=True)
    axes[2].set_title("Calculated regression lines over the selected features")
    axes[0].set_ylabel("Predicted sales price")

    # Draw the individual scatterplots based on the selected attribute columns
    selected_columns_without_price = selected_columns[0:(len(selected_columns)-1)]
    for col in selected_columns_without_price:
        sns.regplot(x=col, y=y_prediction, data=x_train, ax=axes[selected_columns_without_price])
    plt.show()

# Function for the execution of the various procedures
def perform_prediction_and_evaluate_results(regr, output_feature_importance = True):
    regr.fit(x_train, y_train)

    print("Evaluation on train data:")
    y_train_prediction = regr.predict(x_train)
    output_error_metrics(y_train, y_train_prediction)

    print("\nEvaluation on test data:")
    y_test_prediction = regr.predict(x_test)
    output_error_metrics(y_test, y_test_prediction)

    if output_feature_importance == True:
        print("\nFeature importances:")
        feature_names = regr.feature_names_in_
        feature_importances = regr.feature_importances_
        for i in range(regr.n_features_in_):
            if feature_names[i] == "Baeder" or feature_names[i] == "EG_qm":
                print(f"{feature_names[i]}:\t{round(feature_importances[i], 4)}")
            else:
                print(f"{feature_names[i]}:\t{round(feature_importances[i], 4)})"

    print("\nScatterplots with regression lines:")
    output_regression_scatterplots(y_train_prediction)

```

## Random Forest Regression

Nun wird eine Random Forest Regression mit den zuvor reduzierten Attributen durchgeführt. Die Bewertung erfolgt anhand der vorgegebenen Fehlermetriken.

In [ ]: `from sklearn.ensemble import RandomForestRegressor`

```

randomForest = RandomForestRegressor(n_estimators=100)
perform_prediction_and_evaluate_results(randomForest)

```

Evaluation on train data:

R2: 0.9756  
MSE: 132076035.1254  
RMSE: 11492.4338  
MAPE: 0.0479  
MAX: 86150.79

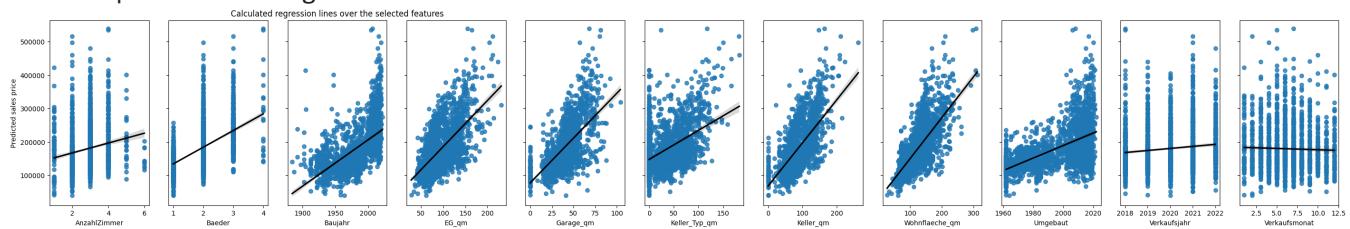
Evaluation on test data:

R2: 0.859  
MSE: 883161966.8154  
RMSE: 29718.0411  
MAPE: 0.1206  
MAX: 133171.13

Feature importances:

AnzahlZimmer: 0.0093  
Baeder: 0.0055  
Baujahr: 0.3964  
EG\_qm: 0.0752  
Garage\_qm: 0.144  
Keller\_Typ\_qm: 0.0469  
Keller\_qm: 0.0796  
Wohnflaeche\_qm: 0.1794  
Umgebaut: 0.0305  
Verkaufsjaehr: 0.0191  
Verkaufsmontat: 0.0141

Scatterplots with regression lines:



## Extra Trees Regression

Nun wird eine Extra Trees Regression mit den zuvor reduzierten Attributen durchgeführt. Die Bewertung erfolgt anhand der vorgegebenen Fehlermetriken.

```
In [ ]: from sklearn.ensemble import ExtraTreesRegressor

extraTrees = ExtraTreesRegressor(n_estimators=100)
perform_prediction_and_evaluate_results(extraTrees)
```

Evaluation on train data:

R2: 1.0  
MSE: 0.4388  
RMSE: 0.6624  
MAPE: 0.0  
MAX: 20.0

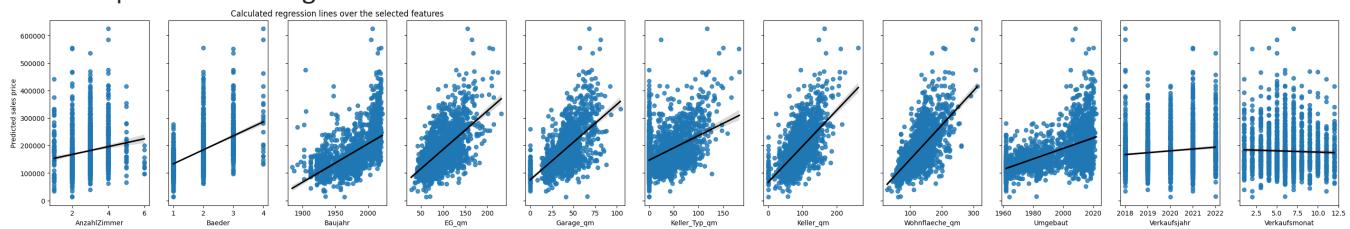
Evaluation on test data:

R2: 0.8656  
MSE: 841535227.1151  
RMSE: 29009.2266  
MAPE: 0.1183  
MAX: 146724.53

Feature importances:

AnzahlZimmer: 0.0208  
Baeder: 0.0822  
Baujahr: 0.1776  
EG\_qm: 0.0854  
Garage\_qm: 0.1216  
Keller\_Typ\_qm: 0.0492  
Keller\_qm: 0.1204  
Wohnflaeche\_qm: 0.1917  
Umgebaut: 0.1055  
Verkaufsjaahr: 0.0284  
Verkaufsmontat: 0.0173

Scatterplots with regression lines:



## Gradient Boosting Regression

Nun wird eine Gradient Boosting Regression mit den zuvor reduzierten Attributen durchgeführt. Die Bewertung erfolgt anhand der vorgegebenen Fehlermetriken.

```
In [ ]: from sklearn.ensemble import GradientBoostingRegressor  
  
gradientBoosting = GradientBoostingRegressor(n_estimators=100)  
perform_prediction_and_evaluate_results(gradientBoosting)
```

Evaluation on train data:

R2: 0.9065  
MSE: 505162073.8941  
RMSE: 22475.8109  
MAPE: 0.1018  
MAX: 115597.5334

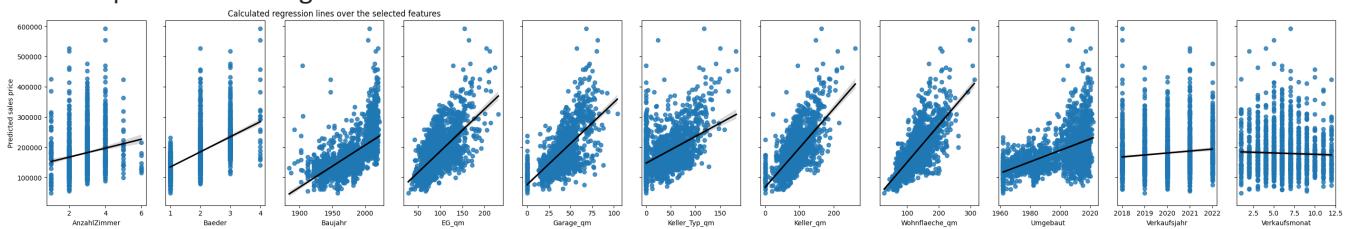
Evaluation on test data:

R2: 0.8484  
MSE: 949519626.1225  
RMSE: 30814.2763  
MAPE: 0.1222  
MAX: 184865.2366

Feature importances:

AnzahlZimmer: 0.0035  
Baeder: 0.0193  
Baujahr: 0.3389  
EG\_qm: 0.0667  
Garage\_qm: 0.1354  
Keller\_Typ\_qm: 0.0438  
Keller\_qm: 0.123  
Wohnflaeche\_qm: 0.2007  
Umgebaut: 0.0493  
Verkaufsjaehr: 0.0179  
Verkaufsmonat: 0.0014

Scatterplots with regression lines:



Die drei auf Suchbäumen basierenden Regressionsverfahren eignen sich mit einer Genauigkeit von ca. 84 % bis 87 % (R2-Wert) im Testdatensatz gut für die Vorhersage des Verkaufspreises. Durch die geplotteten Graphen wird ebenfalls deutlich, dass die Daten dicht an der angepassten Regressionslinie liegen. An den MSE- und RMSE-Werten fällt auf, dass Varianz und Verzerrung bei allen drei Verfahren recht ähnlich sind. Am geringsten sind sie bei der Extra Trees Regression. Der größte Fehler zwischen dem vorhergesagten Wert und dem wahren Wert (MAX) liegt bei Gradient Boosting vor.

Die Bedeutung und der Einfluss der einzelnen Attribute (Gini-Bedeutung) unterscheidet sich ebenfalls stark von Verfahren zu Verfahren. Bei Random Forest stehen Baujahr (0,40), Wohnflaeche\_qm (0,17), Garage\_qm (0,15) und Keller\_qm (0,09) im Vordergrund. Bei Extra Trees ist die Gewichtung gleichmäßiger über alle Attribute verteilt. Am bedeutendsten sind Wohnflaeche\_qm (0,18), Baujahr (0,17) und Garage\_qm (0,13). Bei der Gradient Boosting Regression wird die Gewichtung durch Baujahr (0,34), Wohnflaeche\_qm (0,20), Garagen\_qm (0,14) und Keller\_qm (0,12) angeführt. Auffällig ist hier auch dass die Zimmeranzahl mit 0,0034 und der Verkaufsmonat mit 0,002 so gut wie nicht in die Vorhersagenberechnung einbezogen werden.

## Epsilon-Support Vector Regression

Nun wird eine Epsilon-Support Vector Regression mit den zuvor reduzierten Attributen durchgeführt. Die Bewertung erfolgt anhand der vorgegebenen Fehlermetriken.

```
In [ ]: from sklearn.svm import SVR

svr = SVR(kernel="linear")
perform_prediction_and_evaluate_results(svr, output_feature_importance = False)
```

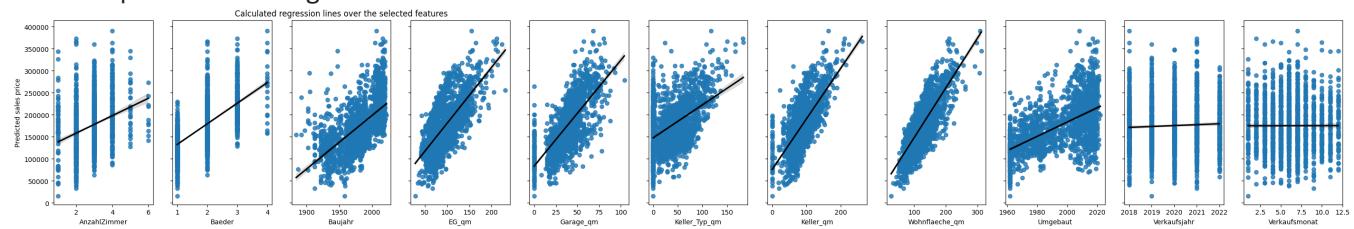
Evaluation on train data:

R2: 0.7553  
 MSE: 1322232875.6552  
 RMSE: 36362.5202  
 MAPE: 0.1453  
 MAX: 233983.6639

Evaluation on test data:

R2: 0.784  
 MSE: 1353246258.6465  
 RMSE: 36786.4956  
 MAPE: 0.1343  
 MAX: 224341.2783

Scatterplots with regression lines:



## Passive Aggressive Regression

Nun wird eine Passive Aggressive Regressor Regression mit den zuvor reduzierten Attributen durchgeführt. Die Bewertung erfolgt anhand der vorgegebenen Fehlermetriken.

```
In [ ]: from sklearn.linear_model import PassiveAggressiveRegressor

passiveAggressive = PassiveAggressiveRegressor()
perform_prediction_and_evaluate_results(passiveAggressive, output_feature_importance = False)
```

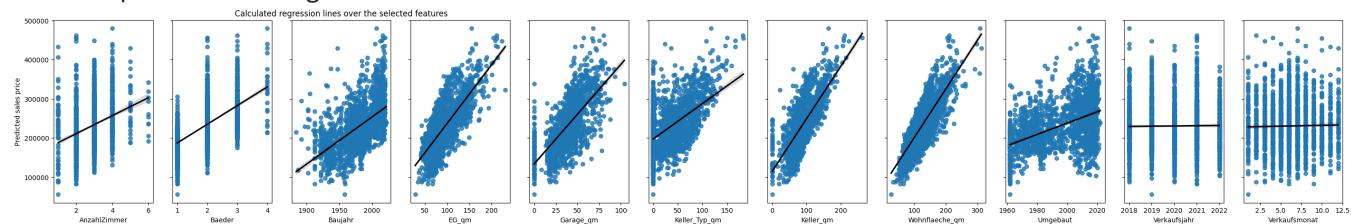
Evaluation on train data:

R2: 0.2657  
 MSE: 3967786476.9466  
 RMSE: 62990.3681  
 MAPE: 0.3811  
 MAX: 196800.1855

Evaluation on test data:

R2: 0.4055  
 MSE: 3723881596.2558  
 RMSE: 61023.6151  
 MAPE: 0.3682  
 MAX: 146623.226

Scatterplots with regression lines:



Die beiden Verfahren Epsilon-Support Vector Regression und Passive Aggressive Regression schneiden im direkten Vergleich zu den vorherigen drei Verfahren signifikant schlechter ab. Die R2-Werte betragen

0,75 bei Epsilon-Support Vector und 0,63 bei Passive Aggressive. Die Performance auf den Testdaten ist allerdings beides mal besser als auf den Trainingsdaten. An den RMSE-Werten sieht man, dass die durchschnittliche Distanz zwischen prognostiziertem und tatsächlichem Verkaufspreis (in Euro) deutlich zugenommen hat. Von durchschnittlichen 33.310 € bei den suchbaum-basierten Regressionsverfahren, steigt der Wert um 5.641 € bei Epsilon-Support Vector und um ganze 13.590 € bei Passive Aggressive.

## Hyperparameteroptimierung für die beste Vorhersage-Methode

In unserem Fall hat sich die **Extra Trees Regression** als geeignete Methode für die Vorhersage des Verkaufspreises herausgestellt. Um die Vorhersage weiter zu optimieren, wird nun eine Hyperparameteroptimierung durchgeführt. Dafür stellt sklearn die Methode `GridSearchCV` bereit.

Da die Hyperparameteroptimierung über die Funktion GridSearchCV eine rechen- und zeitintensive Operation ist, ist sie im Notebook standardmäßig deaktiviert. Der durchschnittliche Score beträgt **0,85**.

```
In [ ]: from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import GridSearchCV

# Activate/Deactivate GridSearch function
gridSearch_active = False
gridSearch_repeats = 3

# Calculate optimal hyperparameters
if gridSearch_active == True:
    for i in range(gridSearch_repeats):
        extraTrees = ExtraTreesRegressor()

        grid_search = GridSearchCV(extraTrees, {
            # Dictionary with parameter names as keys and lists of settings to try as values
            "max_depth": [21, 22, 23, 24, 25, 26, 27, 28],
            "min_samples_split": [2, 3, 4, 5, 6, 7, 8, 9, 10],
            "n_estimators": [150, 155, 160, 165, 170, 175, 180, 185, 190, 195, 200]
        }, n_jobs=-1, verbose=1)

        grid_search.fit(X, Y)

        # Output results
        print("The best parameters are:")
        for key, value in grid_search.best_params_.items():
            print(f" • {key}: {value}")
        print("The score is:", grid_search.best_score_)
```

Beispieldaten:

```
Fitting 5 folds for each of 792 candidates, totalling 3960 fits
The best parameters are:
• max_depth: 25
• min_samples_split: 5
• n_estimators: 170
The score is: 0.8498264789384029
Fitting 5 folds for each of 792 candidates, totalling 3960 fits
The best parameters are:
• max_depth: 27
• min_samples_split: 5
• n_estimators: 190
The score is: 0.8497874589933844
Fitting 5 folds for each of 792 candidates, totalling 3960 fits
```

```
The best parameters are:  
  • max_depth: 27  
  • min_samples_split: 5  
  • n_estimators: 150  
The score is: 0.8501926503521723
```

## Trick 17

Nachdem die optimalen Hyperparameter berechnet wurden, wird das ausgewählte Modell abschließend mit dem kompletten Datensatz (Trainings- + Validierungsdaten) final trainiert. Dadurch werden die vorhandenen Informationen bestmöglich ausgenutzt. Die Vorhersagen des neuen Modells sind mindestens so gut, wie die des Vorherigen, können aber auch besser sein.

```
In [ ]: from sklearn.ensemble import ExtraTreesRegressor  
  
# Create the gradient boosting regressor with the selected hyperparameters  
extraTrees_final = ExtraTreesRegressor(  
    random_state=0,  
    max_depth=25,  
    min_samples_split=5,  
    n_estimators=170  
)  
  
# Training the model  
extraTrees_final.fit(X, Y)
```

```
Out[ ]:   
▼ ExtraTreesRegressor  
ExtraTreesRegressor(max_depth=25, min_samples_split=5, n_estimators=170,  
                    random_state=0)
```

## Vorhersagen berechnen und in `data_for_test_filled.csv` schreiben

Mit dem finalen Modell werden nun Verkaufspreise für den Testdatensatz vorhergesagt. Anschließend werden die Vorhersagen in das `data_for_test_filled.csv` File geschrieben.

```
In [ ]: # Select chosen parameters on the test data frame  
#selected_columns_old = ["Baeder", "Baujahr", "EG_qm", "Garage_qm", "Garagen", "Keller_qm", "  
selected_columns = ["AnzahlZimmer", "Baeder", "Baujahr", "EG_qm", "Garage_qm", "Keller_Typ_qm  
df_test_selected_features = df_test.loc[:, selected_columns]  
  
# Define mapping and apply Labelencoding for the required columns in the test dataset  
#mapping = {"0": 0, "Sehr Schlecht": 1, "Schlecht": 2, "Durchschnitt": 3, "Gut": 4, "Sehr gut": 5  
#df_test_selected_features["Kellerhoehe"] = df_test_selected_features["Kellerhoehe"].replace(mapping)  
  
# Make predictions for the "data_for_test.csv" data with the trained model  
y_predicted_final = extraTrees_final.predict(df_test_selected_features)  
  
# Read data_for_test, add the predictions to the corresponding rows and save as a new CSV file  
try:  
    # Create new file and open test and test_filled  
    data_for_test_filled = open("data_for_test_filled.csv", "x").close()  
    data_for_test = open("data_for_test.csv", "r")  
    data_for_test_filled = open("data_for_test_filled.csv", "a")  
  
    # Loop through every row and append the prediction at the end of the line
```

```

# The first line is a special case
first_row = True
count_rows = df_test_selected_features.shape[0]
for row, i in zip(data_for_test, range(-1, count_rows)):
    if first_row == True:
        data_for_test_filled.write(row.removesuffix("\n") + ";" + "Z_Verkaufspreis" + "\n")
        first_row = False
    else:
        data_for_test_filled.write(row.removesuffix("\n") + ";" + str(int(np.round(y_pred

# Close both files
data_for_test.close()
data_for_test_filled.close()
print("File data_for_test_filled.csv created successfully.")
except FileExistsError:
    print("The file data_for_test_filled.csv does already exist. No changes were made.")

```

File data\_for\_test\_filled.csv created successfully.

## 6. Deployment (3 Punkte)

**Aufgabenstellung:** Erstellen Sie eine Anleitung oder Handreichung für die in Aufgabe 1 genannte Zielgruppe. Dies soll aus Zielgruppensicht wichtige Erkenntnisse der Aufgaben 2 bis 5 zusammenfassen und maximal 2 Seiten im pdf-Ausdruck umfassen, welche auf Basis der Texte aus Aufgabe 1 dann komplett eigenständig lesbar sein sollen.

### Anleitung für Immobilienspekulant\*innen

Durch die ausführliche Analyse der zur Verfügung stehenden Daten, konnten viele Erkenntnisse gewonnen werden. Es können nun Antworten auf die anfangs gestellten Fragen gegeben werden. Um den Spekulant\*innen eine grobe Bewertung der Attraktivität der Angebote zu geben, folgen hier nun Einsichten, welche aus den Daten gewonnen wurden.

Die unten stehende Liste gibt alle Attribute an, welche den **Verkaufspreis** erhöhen können. Intuitiv sind hier **Baujahr**, **Wohnfläche** und das **Umgebaut-Datum** enthalten. Dabei etwas weniger intuitiv sind die Attribute **Garagengröße** und **Stellplätze**, sowie die **Kellerhoehe**, welche auch einen hohen Einfluss haben. Nebenstehenden Werte geben an, um wie viel Euro eine Einheit des Attributes durchschnittlich den Verkaufspreis beeinflusst. Die **Bäderanzahl** und die **Quadratmeter des Kellers** wirken sich ebenfalls positiv auf den **Verkaufspreis** aus, allerdings deutlich geringer als die Attribute aus der ersten Liste. Es kann sich also an **Bäderanzahl** und Kellerfläche orientiert werden, allerdings mit geringerer Gewichtung.

Attribute, welche positive Auswirkungen auf den Verkaufspreis haben:

Attributbezeichnung	Preisseigerung / Einheit
Baujahr	538€ / Jahr
Erdgeschoßgröße	659€ / m <sup>2</sup>
Garagengröße	598€ / m <sup>2</sup>
Garagenstellplätze	1200€ / Auto-Platz
Kellerhoehe	6213€ / Kategorie
Umgebaut	414€ / Jahr
Wohnfläche	396€ / m <sup>2</sup>

Attribute, welche (geringe) Auswirkungen auf den Verkaufspreis haben:

- Bäderanzahl
- Keller\_qm

Gegenteilig zu den positiven Auswirkungen auf den Preis haben sich auch Attribute herausgestellt, welche keinen wirklichen Einfluss auf den Preis haben:

Attribute, welche sich kaum direkt auswirken:

- Anzahl Zimmer
- Ausbaustufe
- Gesamteindruck
- Kellertyp

Überraschend ist hier die **Anzahl der Zimmer** und der **Gesamteindruck**. Dieser ist ein subjektiver Wert eines Individuums und ist nur vom äußeren Aussehen der Immobilie abhängig. Dadurch schneiden hier etwas ältere Häuser pauschal immer schlechter ab, obwohl sie nicht billiger sind.

## Die Vorhersage

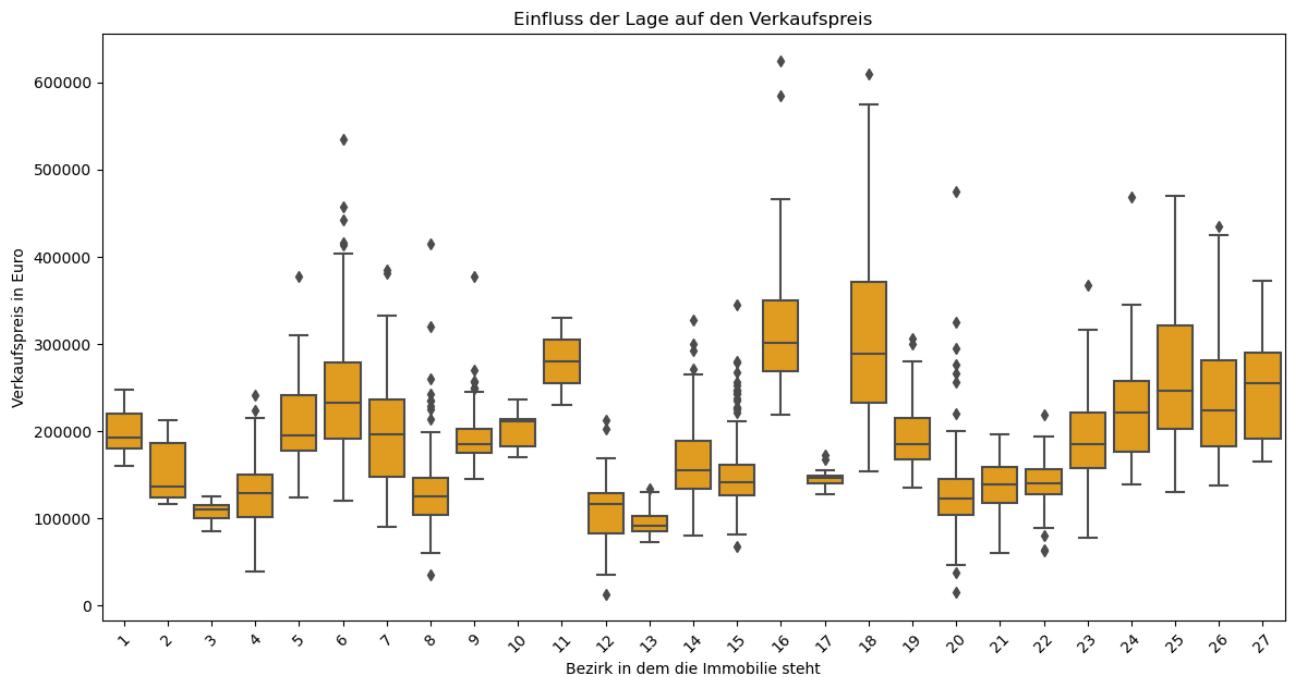
Zur Klassifikation der Attraktivität der Angebote wurde ein Extra Trees Regression Modell erstellt, welches den Preis einer Immobilie aufgrund der Attribute vorhersagt. Dies geschieht dank Optimierung und Trick 17 mit mindestens 85% Genauigkeit. Wenn man ein neues Angebot einschätzen möchte, nimmt man also den geforderten Preis ( `Z_Verkaufspreis` ) und subtrahiert den vorhergesagten Wert des Modells davon. Wenn ein negatives Ergebnis herauskommt, also der vorhergesagte Preis höher ist, dann handelt es sich sehr wahrscheinlich um ein gutes Angebot. Wenn ein positives Ergebnis herauskommt, dann ist der reale Wert der Immobilie wahrscheinlich geringer als der veranschlagte Preis. Je größer die Differenz des Preises und des vorhergesagten Wertes ist, desto besser (-) oder schlechter (+) wird das Angebot. Hier kann man nun Grenzwerte für gute, neutrale und schlechte Angebote festlegen. Als Beispiel sind 0% Preisdifferenz (gleiche Vorhersage wie geforderter Preis) und alles darunter gut, zwischen 0% und 10% ist ein Angebot neutral und über 10% Differenz ist es schlecht.

## Die wichtigsten Erkenntnisse

Zusammengefasst sind die Informationen über die Attribute sehr aufschlussreich. Die Unterscheidung zwischen den verschiedenen Stufen der Auswirkung kann beim Einschätzen der Angebote sehr helfen. Auffällig ist, wie sehr eine **Garage** und ihre **Fläche** den Preis beeinflusst. Auf der gegenteiligen Seite fällt ins Auge, wie wenig die **Anzahl der Zimmer** mit dem Preis korreliert. Aus der Datenanalyse ging allerdings auch hervor, dass die teuersten Häuser zwischen zwei und vier **Zimmer** haben. Der **Gesamteindruck** korreliert ebenfalls wenig mit dem Verkaufspreis. Die teuersten Immobilien haben fast ausnahmslos einen Gesamteindruck von drei erhalten. Es zeigt also, dass dieses Attribut nicht viel über den Wert einer Immobilie aussagt.

Beim Verkaufszeitpunkt fällt auf, dass in den letzten Jahren die Preise für Immobilien leicht gestiegen sind. Als Grund dafür werden allgemein steigende Immobilienpreise oder Inflation vermutet. Zwischen dem Verkaufsmonat und dem Verkaufspreis besteht auch ein leichter Zusammenhang, hier fällt auf, dass in den Monaten April bis Juli sehr viele Immobilien verkauft werden, allerdings auch ein wenig teurer als in den restlichen Monaten.

Die Lage einer Immobilie kann auch Aufschluss über ein Angebot geben. Hier lässt sich allerdings nur ein Gebiet mit generellen Tendenzen zuordnen, es ist kein Zusammenhang zwischen den Gebieten und ihren Nummern zu erkennen. Die teuersten Gebiete sind 11, 16 und 18. Die folgende Grafik gibt genauere Auskunft über die Preise der einzelnen Bezirke:



## 7. Classification (3 Punkte)

**Aufgabenstellung:** Versuchen Sie Immobilien dem richtigen Bezirk zuzuordnen, dabei können Sie den Preis als Eingabewert nehmen. Bewerten Sie die Qualität Ihrer Lösung und kommentieren Sie Ihre Erkenntnisse aus diesem kleinen Test. Erwarteter Umfang entspricht 3 Punkten von 30.

### Vorhersage des Wohnbezirkes einer Immobilie anhand ihres Preises

Da die Lage einer Immobilie zum Teil mit ihrem Verkaufspreis korreliert, ist das Ziel in diesem Abschnitt, eine Immobilie anhand ihres Preises korrekt dem Bezirk zuzuordnen. Dafür werden mehrere Klassifizierungsverfahren implementiert und miteinander verglichen:

- K Nearest Neighbors
- Random Forest
- Naive Bayes
- SVC

Der Code orientiert sich stark an Aufgabe 5, da eine gleiche Vorgehensweise gewählt wurde. Die Funktion `evaluate_model()` ist eine reduzierte Variante von `perform_prediction_and_evaluate_results()` aus Aufgabe 5.

```
In [ ]: from sklearn import metrics
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def evaluate_model(model, x_train, x_test, y_train, y_test):
    # Perform classification
    model.fit(x_train, y_train)
    y_train_prediction = model.predict(x_train)
```

```

y_test_prediction = model.predict(x_test)

# Output evaluation metrics
print("\tEvaluation on train data:")
print('\t Accuracy Score: ', metrics.accuracy_score(y_train, y_train_prediction))
print('\t R2: ', np.around(r2_score(y_train, y_train_prediction), decimals=4))
print("\tEvaluation on test data:")
print('\t Accuracy Score: ', metrics.accuracy_score(y_test, y_test_prediction))
print('\t R2: ', np.around(r2_score(y_test, y_test_prediction), decimals=4))

# Plot confusion matrix
matrix = confusion_matrix(y_train, y_train_prediction)
matrix.diagonal()/matrix.sum(axis=1)
matrix_display = ConfusionMatrixDisplay(matrix)
fig, ax = plt.subplots(figsize=(11,6))
plt.title("Confusionmatrix: Lage")
matrix_display.plot(ax=ax)

```

Hier werden die x und y Daten neu gesetzt. Es wird nur das Attribut `Z_Verkaufspreis` zum Lernen verwendet. Anschließend werden die Trainings- und Testdaten aufgeteilt, im Verhältnis 80:20.

```

In [ ]: from sklearn.model_selection import train_test_split

columns_default = ["Lage", "Z_Verkaufspreis"]
columns_extended = ["EG_qm", "Garagen", "Keller_qm", "Wohnflaeche_qm", "Lage", "AnzahlZimmer", "Ausbaustufe", "Baeder", "BaederKG", "Baujahr", "EG_qm", "Garage_qm", "Kellerhoehe", "Kellertyp", "Lage", "OG_qm", "Umgebaut", "Verkaufsjaehr", "V选中_columns = columns_default

df_train_selected_features = df_train_cleaned.loc[:, selected_columns]

# Split the data in input features (X) and target values (Y)
X = df_train_selected_features.drop(columns=["Lage"])
Y = df_train_selected_features["Lage"]

# Split data set into training and validation data
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

```

```

In [ ]: from sklearn.neighbors import KNeighborsClassifier

def k_nearest_neighbour(x_train, x_test, y_train, y_test):
    print('Result for Nearest Neighbor: ')
    evaluate_model(KNeighborsClassifier(5), x_train, x_test, y_train, y_test)
k_nearest_neighbour(x_train, x_test, y_train, y_test)

```

Result for Nearest Neighbor:

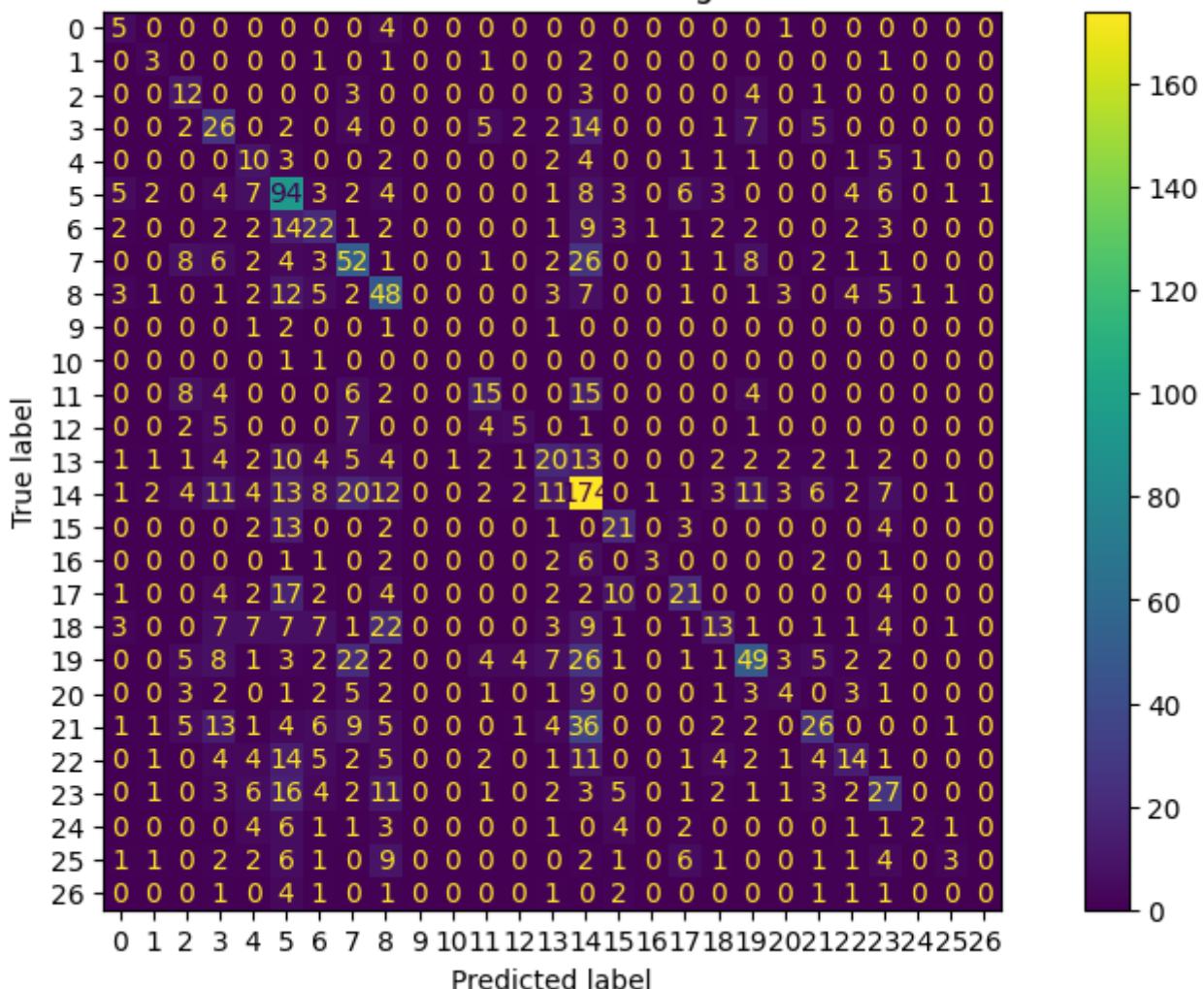
Evaluation on train data:

Accuracy Score: 0.36697750959956116  
R2: -0.6328

Evaluation on test data:

Accuracy Score: 0.14692982456140352  
R2: -1.1667

Confusionmatrix: Lage

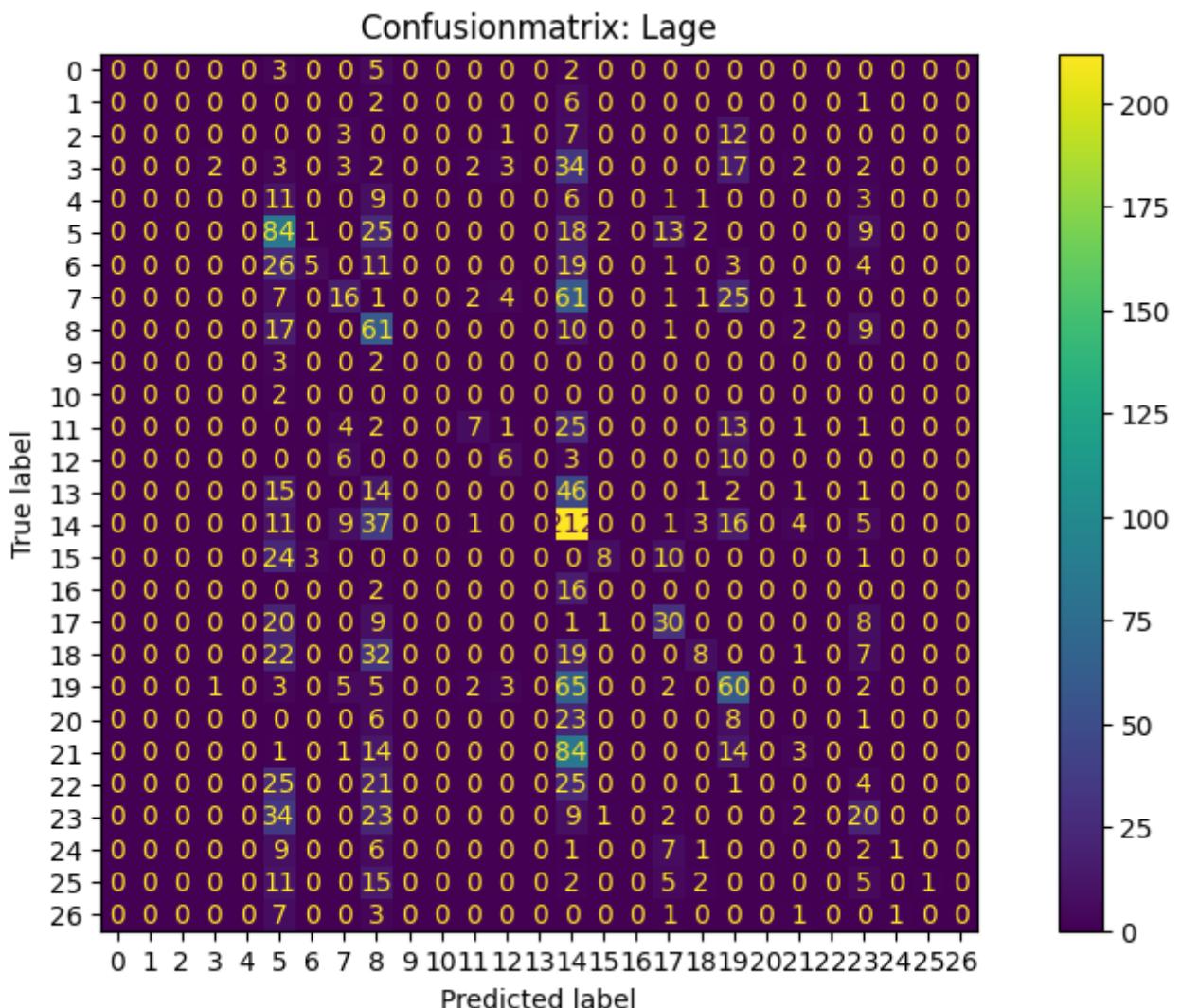


```
In [ ]: from sklearn.ensemble import RandomForestClassifier
```

```
def random_forest_classifier(x_train, x_test, y_train, y_test):
    print('Result for Extra Trees: ')
    evaluate_model(RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1), x_tr
random_forest_classifier(x_train, x_test, y_train, y_test)
```

Result for Extra Trees:

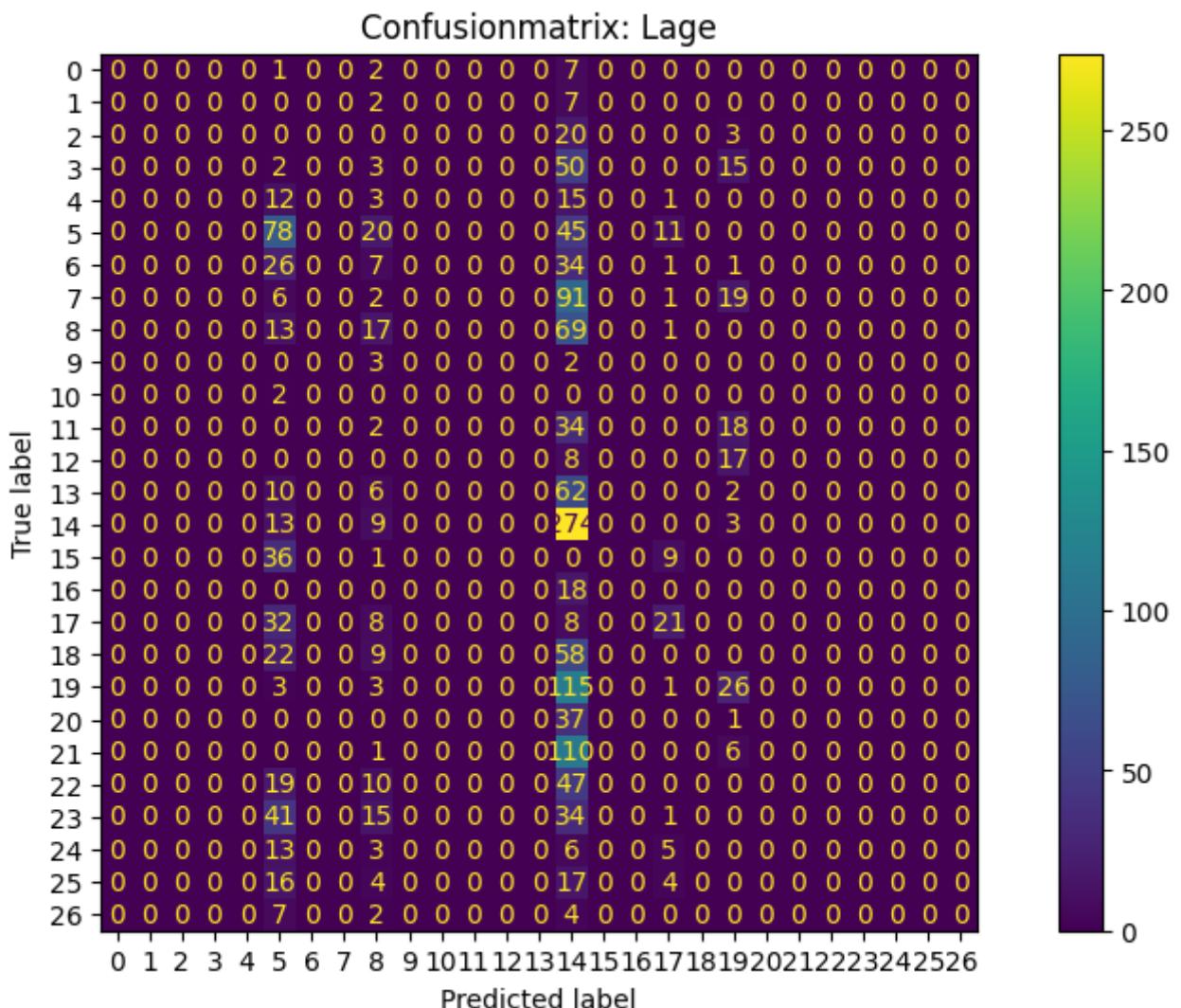
```
Evaluation on train data:
Accuracy Score:  0.2874382885353812
R2:              -0.5395
Evaluation on test data:
Accuracy Score:  0.23903508771929824
R2:              -0.7752
```



```
In [ ]: from sklearn.naive_bayes import GaussianNB

def naive_bayes(x_train, x_test, y_train, y_test):
    print('Result for Naive Bayes: ')
    evaluate_model(GaussianNB(), x_train, x_test, y_train, y_test)
naive_bayes(x_train, x_test, y_train, y_test)
```

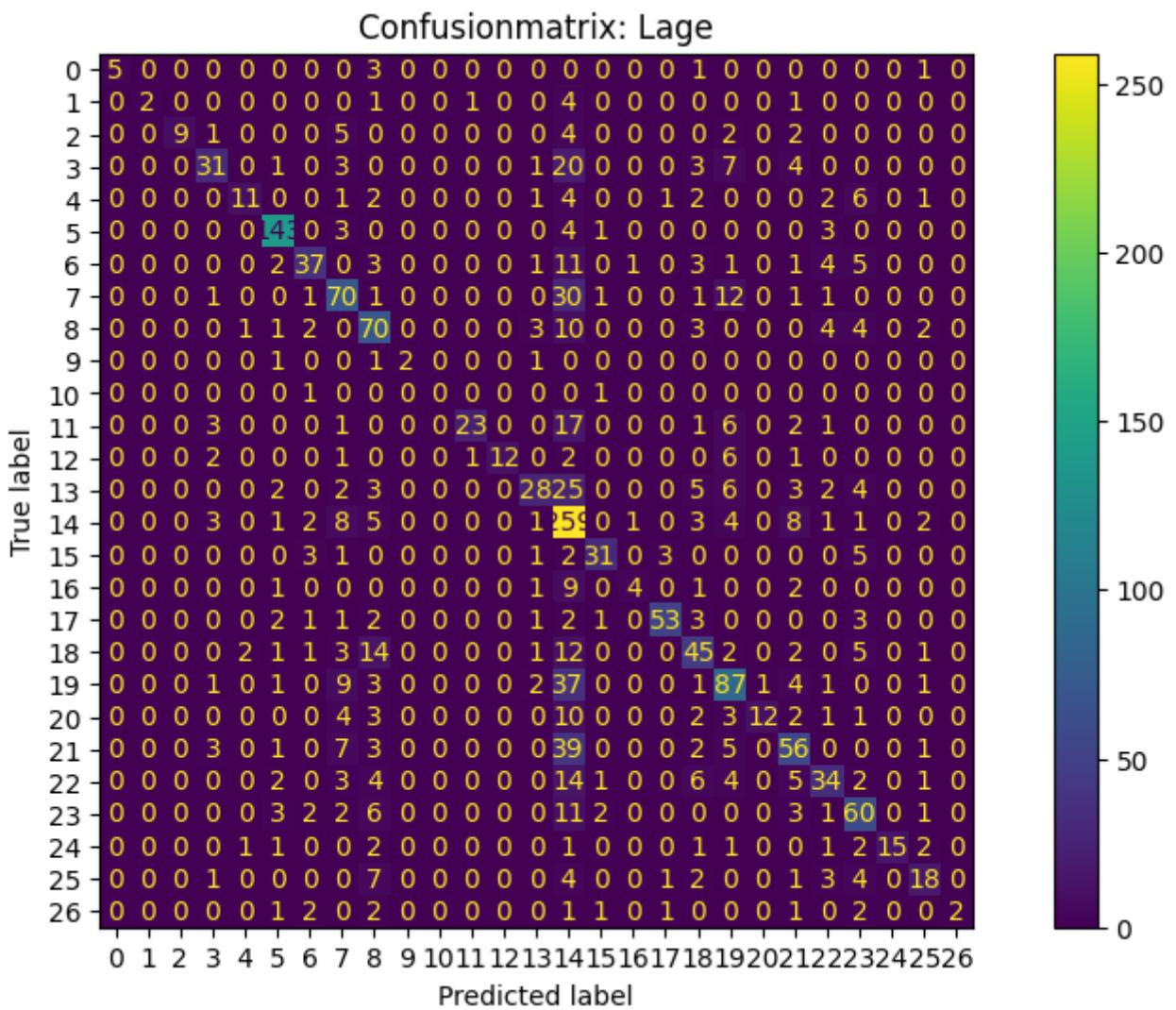
Result for Naive Bayes:  
Evaluation on train data:  
Accuracy Score: 0.22819528250137136  
R2: -0.4587  
Evaluation on test data:  
Accuracy Score: 0.2631578947368421  
R2: -0.4593



```
In [ ]: from sklearn.svm import SVC

def svc(x_train, x_test, y_train, y_test):
    print('Result for SVC: ')
    evaluate_model(SVC(C=10, gamma="auto", kernel="rbf", probability = True), x_train, x_test)
    svc(x_train, x_test, y_train, y_test)

Result for SVC:
Evaluation on train data:
Accuracy Score:  0.6138233680746022
R2:              0.2331
Evaluation on test data:
Accuracy Score:  0.16228070175438597
R2:              -0.5546
```



## Resultat Vorhersage

Die Ergebnisse aller fünf verschiedener Regressionsarten sind zu schlecht, um daraus eine vernünftige Vorhersage erstellen zu können. Sowohl die Accuracy (Trefferrate), als auch der R<sup>2</sup>-Wert sind optimal bei eins und bei null sehr schlecht bzw. am schlechtesten. Dies bedeutet, dass kein ausreichender Zusammenhang zwischen Lage und Preis entsteht (maximal 26 % accuracy). In der Grafik am Ende von Aufgabe 6 ist dies auch gut zu erkennen. Sehr viele Bezirke überschneiden oder decken sich in dem Bereich ihrer Preisspanne. Das macht das Klassifizieren einer Immobilie in einen Bezirk sehr schwierig. Das Resultat für eine Vorhersage nur auf Grundlage des Preises ist also, dass es nicht möglich ist, eine annehmbare Vorhersage zu treffen.

Um dennoch eine akzeptable Vorhersage zu erreichen, können weitere Attribute hinzugezogen werden. Damit basiert die Vorhersage nicht mehr nur auf dem Preis, sondern hat mehr Daten für die Prognose zur Verfügung. Die ausgewählten Attribute, um den Preis zu erweitern, sind "EG\_qm", "Garage\_qm", "Garagen", "Keller\_qm", "Wohnfläche\_qm". Die Attribute wurden aus der Korrelationsheatmap am Ende von Aufgabe 3 genommen. Es sind die Attribute, die am meisten mit der Lage korrelieren.

```
In [ ]: #select the extended columns instead of the default ones with only Z_Verkaufspreis and Lage
selected_columns = columns_extended

df_train_selected_features = df_train_cleaned.loc[:, selected_columns]

# Split the data in input features (X) and target values (Y)
X = df_train_selected_features.drop(columns=["Lage"])
Y = df_train_selected_features["Lage"]
```

```
# Split data set into training and validation data
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

k_nearest_neighbour(x_train, x_test, y_train, y_test)
random_forest_classifier(x_train, x_test, y_train, y_test)
naive_bayes(x_train, x_test, y_train, y_test)
svc(x_train, x_test, y_train, y_test)
```

Result for Nearest Neighbor:

```
Evaluation on train data:
Accuracy Score: 0.41634668129456937
R2: -0.5679
Evaluation on test data:
Accuracy Score: 0.1600877192982456
R2: -1.1336
```

Result for Extra Trees:

```
Evaluation on train data:
Accuracy Score: 0.3971475589687329
R2: -0.2415
Evaluation on test data:
Accuracy Score: 0.3530701754385965
R2: -0.4572
```

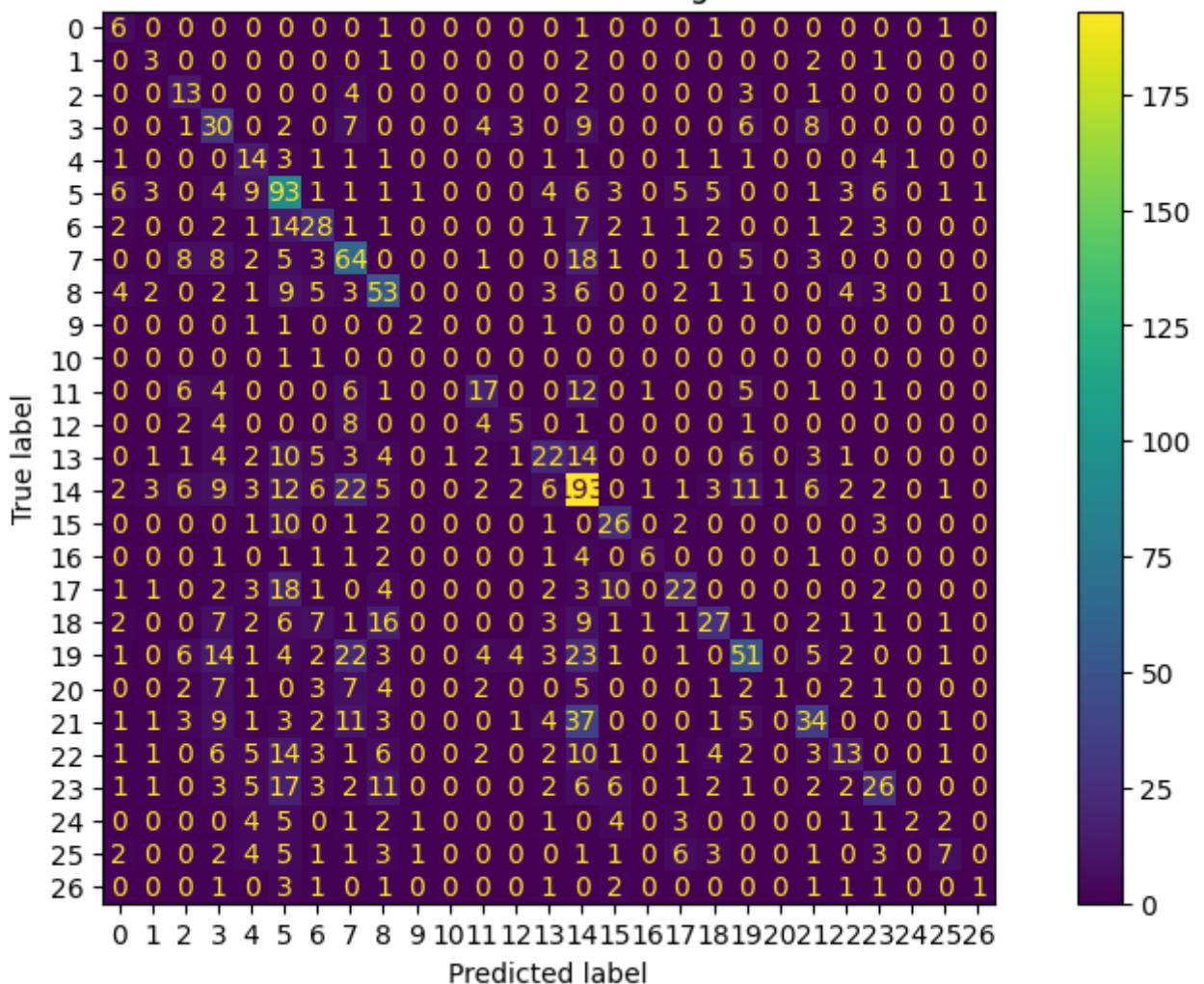
Result for Naive Bayes:

```
Evaluation on train data:
Accuracy Score: 0.27317608337904553
R2: -0.6958
Evaluation on test data:
Accuracy Score: 0.2675438596491228
R2: -0.5869
```

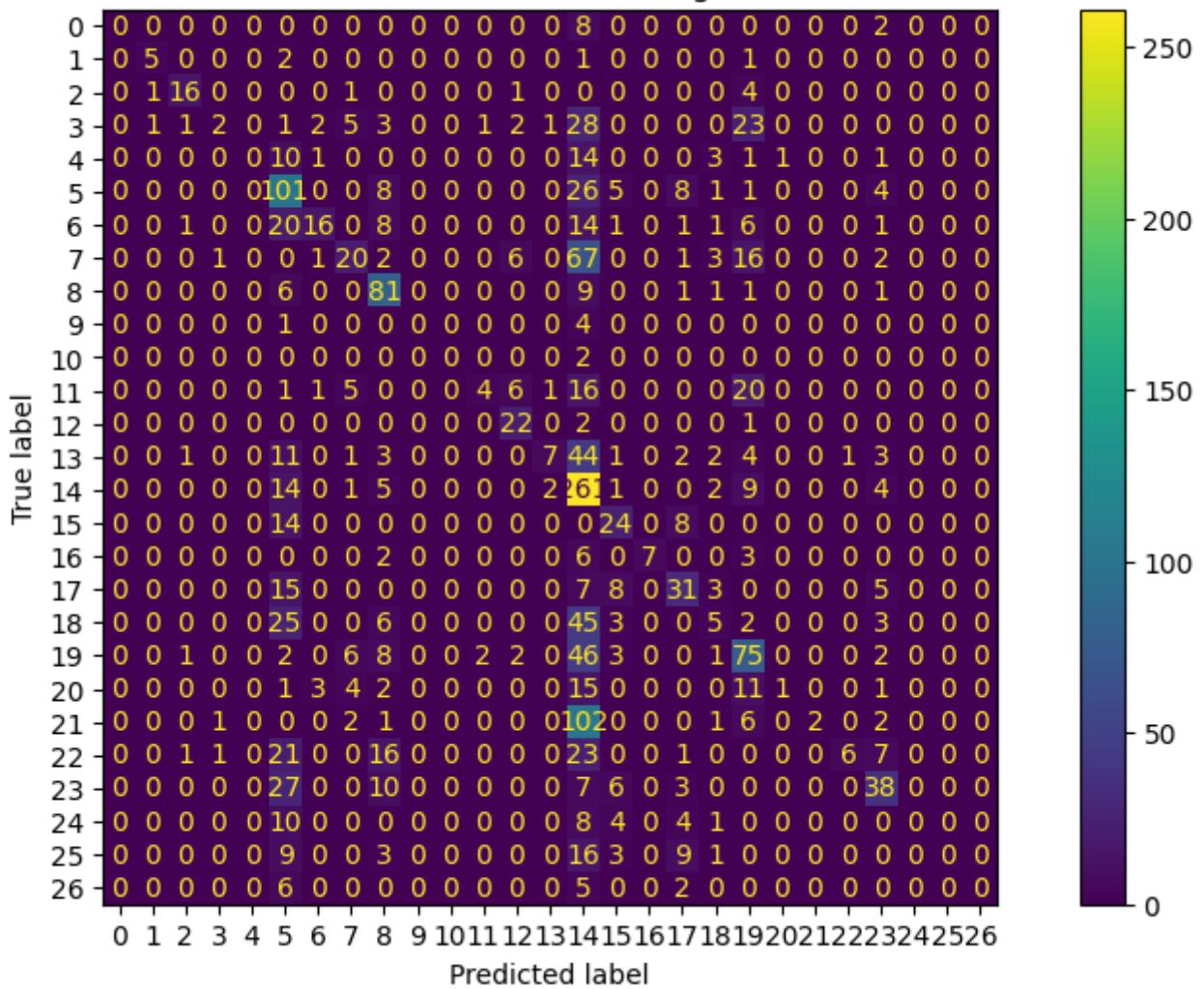
Result for SVC:

```
Evaluation on train data:
Accuracy Score: 1.0
R2: 1.0
Evaluation on test data:
Accuracy Score: 0.18421052631578946
R2: -0.0052
```

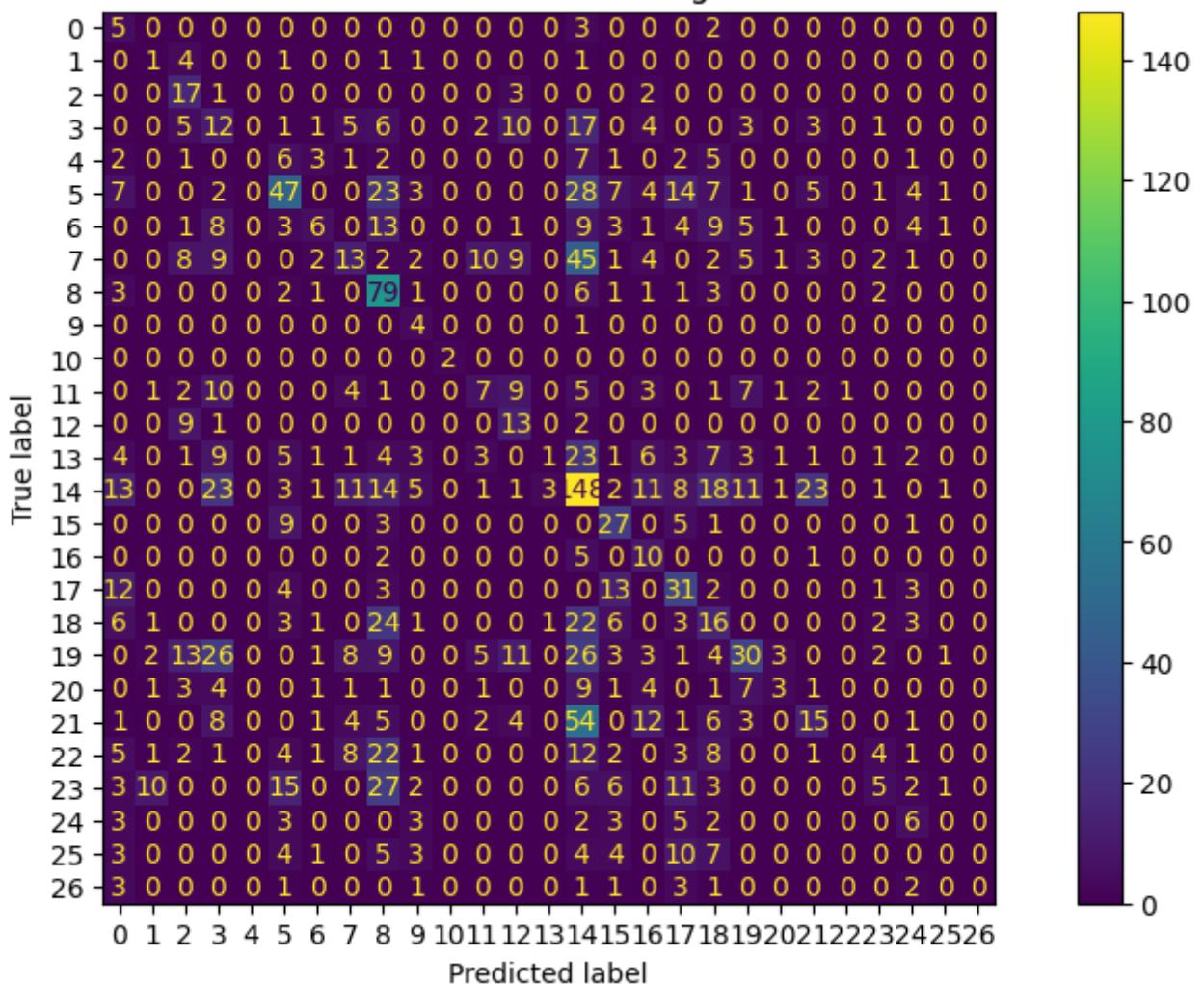
Confusionmatrix: Lage

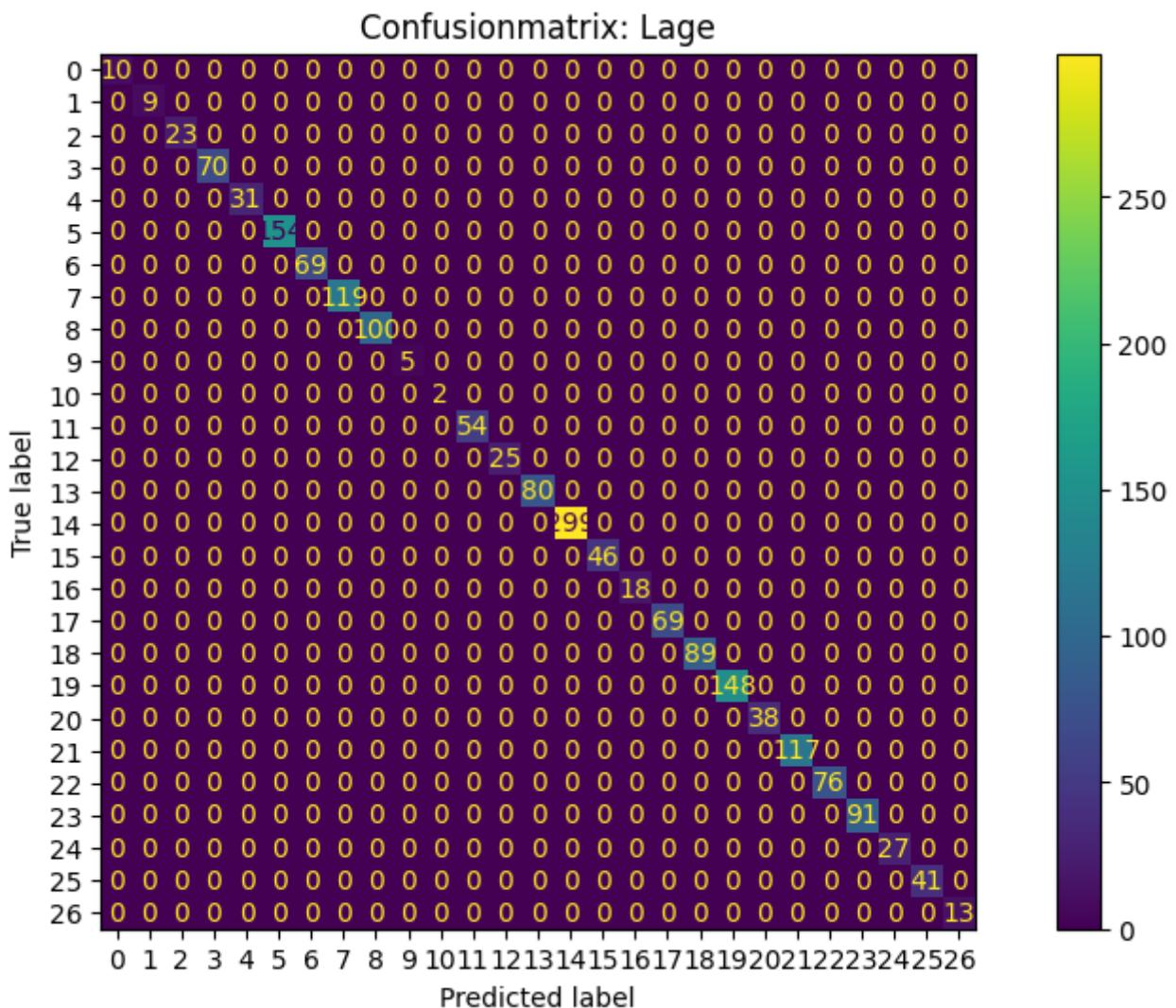


## Confusionmatrix: Lage



Confusionmatrix: Lage





## Resultat der verbesserten Vorhersage

Im Vergleich zu der ursprünglichen Vorhersage ist diese verbesserte Vorhersage mit mehr Attributen und ca. 32 % accuracy zwar tendenziell ein wenig besser geworden, allerdings immer noch nicht gut. Es ist also auch mit der verbesserten Vorhersage mit keinem der getesteten Modelle möglich, eine annehmbare Klassifizierung der Lage zu geben.