Erstellt von 4757202 und 2100594.

Die zusätzliche csv-Datei kann am Anfang von Aufgabe 2 eingelesen werden.

Bewertungskriterien

1. Fachliche Bewertung (50%): Korrektheit, Lösungsqualität und Eleganz sowie Klarheit und Umfang der Betrachtung, Umsetzung von Data Science wie in der Vorlesung gelehrt in einem Code-Prototyp, korrekte Verwendung von wichtigen Funktionen / Bibliotheken, Güte der Endlösung, Nutzung der erworbenen Kenntnisse aus der Vorlesung, Vollständigkeit der Lösung in Bezug auf die Aufgabenstellung
2. Dokumentation (50%): Dokumentation des Vorgehens der Datenauswertung im Sinne von Data Science, Codekommentare wie in der Informatik üblich wo notwendig, Qualität der Diagramme, Videopräsentation

Aufgabe 1: Business Understanding (3P)

Formulieren Sie ein Ziel oder mehrere Ziele nach dem CRISP-DM Prozess, die für Investor*innen sinnvoll sind. Diese Personen investieren Geld mit dem Ziel, dieses zu vermehren, z. B. durch Kauf/Verkauf, Renovierung, Umbau. Beginnen Sie mit der Idee „Wir brauchen mehr Verständnis und eine Vorhersage des Verkaufspreises (Preis)!“, welche auf jeden Fall zu bearbeiten ist.

Ursprüngliche Abgabe (16.11.2020)

Ziele auf Geschäftsebene

Grundlegendstes Ziel: Gewinn durch Immobilienkäufe erzielen

Ähnliche Immobilien vergleichen und preisliche Trends am Immoblilienmarkt erkennen -> Bestmögliche (Preis/Leistung) Immobilie finden

Risikoeinschätzung vor dem Kauf der Immobilien ermöglichen -> Im Falle von Verlusten diese möglichst gering halten

Immobilien günstig kaufen, renovieren und weiterverkaufen -> Wertsteigerung erzielen

Immobilien kaufen, welche in Zukunft preislich steigen -> Akkurate Prognosen treffen

Zu beantwortende Fragen

Aus welchen Immobilien lässt sich der meiste Gewinn erzielen? Beispiel: "Immobilien in Ballungszentren sind besonders rentabel."

Welche Immobilien liegen wieso preislich unter dem Durchschnitt? Beispiel: "Das Haus ist riesig und befindet sich in einem top Zustand, die schlechte Lage drückt den Preis jedoch wieder nach unten."

Welche Kriterien beeinflussen den Verkaufspreis? Wie stark beeinflussen sie diesen? Welche anderen Metriken beeinflussen den Preis am stärksten? Beispiel: "Die Wohnfläche beeinflusst den Verkaufspreis maßgeblich, jedoch nicht so stark wie der Zustand der Immobilie."

Bei welchen Immobilien haben sich Renovierungen/Umbau in der Vergangenheit gelohnt und ist ein Trend zu erkennen? Beispiel: "Besonders im Bezirk XY konnte durch Renovierung in der Vergangenheit der Preis gesteigert werden."

Wie hoch ist das Risiko beim Kauf einer Immobilie? Steigen die Preise stetig und langsam oder schwankt der Preis von Jahr zu Jahr? Beispiel: "Die Preise sind in den letzten Jahren stetig gestiegen. Eine Investition in diese Immobilie ist verhältnismäßig sicher"

Hallo Team 10,

das sieht gut aus, Sie haben sich viel vorgenommen. Denken Sie daran die grundlegende Aufgabe (sage den Hauspreis gegeben aller anderen Daten vorher) nicht zu vergessen;) Es folgen einige Beobachtungen zu Ihrem Test.

*Von Ihren Zielen sind alle interessant, einige Fragen sind noch unscharf (was ist **"Leistung"** bei einer Immobilie, was heißt **"rentabel"**, etc.). Sie können im Verlauf des Projektes auch Ziele fallen lassen, da Sie sich viel vorgenommen haben. Die Frage mit preislich "unter Durchschnitt" ist nicht schlecht, auch hier ist es wichtig festzulegen **was genau "unter Durchschnitt" heißt**. Bei Renovierungen und Preisschwankung bin ich nicht sicher, ob die Daten das hergeben. Das mit den ähnlichen Immobilien klingt gut, hier brauchen Sie dann in der Evaluation eine Verwendung dieser Information (was hilft mir das, wenn das aktuell zum Verkauf stehende Haus ähnlich einem anderen Haus ist, was ich vor 3 Jahren gekauft habe?). Hier noch der allgemeine Hinweis: **keine Erkenntnis ist auch eine wertvolle Erkenntnis in Data Science Projekt, sofern diese gut dokumentiert ist**. Fokussieren Sie sich während des Projektes, um in einer "vernünftigen" Zeit fertig zu werden.*

Überarbeitete Ziele und Fragen (30.11.2020)

Ziele auf Geschäftsebene

- Grundlegendstes Ziel: Gewinn durch Immobilienkäufe erzielen.
- Überprüfung von Marktpreisen ermöglichen (neues Haus kommt auf den Markt -> Prüfen, ob Preis anhand der vorhandenen Daten realistisch ist). Somit können Häuser, welche zu vergleichsweise günstigen Preisen auf dem Markt angeboten werden, herausgefiltert und genauer betrachtet werden.
- Herausfinden, welche Metriken den größten Einfluss auf den Immobilienpreis haben.
- Herausfinden, ob durch Renovierungen von Immobilien ein Gewinn erzielt werden kann.
- Risikoeinschätzung vor dem Kauf der Immobilien ermöglichen -> Im Falle von Verlusten diese möglichst gering halten.

Zu beantwortende Fragen

- Welche Immobilien liegen preislich im Vergleich zu den anderen Immobilien unter dem Durchschnitt für das, was sie bieten? Beispiel: "Für die vorhandenen Metriken (Wohnfläche, Räume, ...) ist das Haus im Vergleich zu den anderen Häusern günstig"
- Welche Kriterien beeinflussen den Verkaufspreis? Wie stark beeinflussen sie diesen? Welche anderen Metriken beeinflussen den Preis am stärksten? Beispiel: "Die Wohnfläche beeinflusst den Verkaufspreis maßgeblich, jedoch nicht so stark wie der Zustand der Immobilie."
- Kann durch Renovierung einer Immobilie ein Gewinn erzielt werden? Wie viel darf eine Renovierung dafür i.d.R. kosten? Beispiel: "Renovierte Häuser sind durchschnittlich 50.000 mehr Wert als nichtrenovierte Häuser, welche im gleichen Jahr gebaut wurden. Eine Renovierung sollte nicht mehr als 40.000 kosten, um einen lohnenswerten Gewinn zu erzielen."

- Wie sicher ist der vorhergesagte Preis? Wie stark kann er abweichen?

Aufgabe 2: Data Exploration und Analyse (10P)

Untersuchen Sie den Datensatz. Analysieren Sie die gegebenen Informationen und suchen Sie Anomalien. Behalten Sie dabei die Zielstellung aus Teil 1 im Auge. Bewerten Sie Schlussfolgerungen daraufhin, wie viel Information Sie haben, um diese zu stützen. Beschreiben Sie die Ergebnisse in ganzen Sätzen auf Deutsch. Nutzen Sie Diagramme wo sinnvoll – achten Sie darauf, diese gut zu formatieren, zu beschriften und korrekt einzusetzen. Schreiben Sie die allerwichtigsten Erkenntnisse für die in Teil 1 definierten Ziele als Summary auf (3 bis 5 Erkenntnisse).

In [193]:

```
#Wichtige Imports
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [194]:

```
# Datensatz einlesen

# ------------Hier Ihren Datensatz einfügen-------------------------
# ------------------vvvvvvvvvvvvvvvvv-----------------------------------
df = pd.read_csv("DatenAusgegeben1.0.csv", sep = ";", encoding = "cp852")
df.head()
```

Out[194]:

	Grundstück in qm	Grundstücksform	Steigung	Bezirk	Zone	Lage	Typ	Zustand	Gebaut	Re
0	898	IR1	Nein	Somerset	RL	Norm	2Fam	4	2107	
1	1326	Reg	Nein	North East	RL	Norm	1Fam	5	2133	
2	725	Reg	Nein	Somerset	RL	Norm	1Fam	7	2096	
3	725	Reg	Nein	Somerset West	RL	Norm	1Fam	5	2135	
4	697	Reg	Nein	Miller	RL	Norm	1Fam	5	2129	

5 rows × 28 columns

Die Datei wurde erfolgreich eingelesen. Beim Einlesen muss beachtet werden, dass die Werte nicht durch ein Komma getrennt sind, sondern durch ein Semikolon. Es muss auch beachtet werden, dass im Datensatz Umlaute vorkommen und deshalb das Encoding geändert werden muss.

In [195]:

```
#Auf null-values überprüfen
df.isnull().sum()
```

Out[195]:

```
Grundstück in qm          0
Grundstücksform           0
Steigung                  0
Bezirk                    0
Zone                      0
Lage                      0
Typ                       0
Zustand                   0
Gebaut                    0
Renoviert                 0
Zustand Fassade           0
Kellerflche in qm         0
Heizung                   0
Heizungsqualitt           0
Klimaanlage               0
Erster Stock in qm        0
Zweiter Stock in qm       0
Wohnflche in qm           0
Schlafzimmer              0
Küchen                    0
Küchenqualitt             0
Rume                      0
Garage Typ               96
Garagenkapazitt           0
Pool                   1991
Verkaufsmonat             0
Verkaufsjahr              0
Preis                     0
dtype: int64
```

Die Spalten 'Pool' und 'Garage Typ' enthalten Null-Werte. Diese müssen bei der späteren Auswertung der Daten beachtet werden, sodass es nicht zu einer Verzerrung der Ergebnisse kommt.

In [196]:

```
# Überblick verschaffen
df.describe()
```

Out[196]:

	Grundstück in qm	Zustand	Gebaut	Renoviert	Zustand Fassade	Kellerfläche in qm	Er Stoc
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.00
mean	950.054000	5.697500	2099.031000	2113.344500	3.102500	96.140500	106.48
std	737.437654	1.129439	29.120114	20.168198	0.386094	38.249893	34.40
min	121.000000	1.000000	2005.000000	2080.000000	1.000000	0.000000	31.00
25%	701.750000	5.000000	2083.000000	2095.000000	3.000000	74.000000	82.00
50%	887.000000	5.000000	2101.000000	2121.000000	3.000000	91.000000	99.00
75%	1078.000000	6.000000	2126.000000	2132.000000	3.000000	117.000000	126.00
max	19997.000000	9.000000	2140.000000	2140.000000	5.000000	298.000000	355.00

Zunächst untersuchen wir den Datensatz ganz allgemein mit df.describe(). Beim Untersuchen ist uns die Anzahl der Pools zuerst ins Auge gesprungen. Häuser ohne Pool werden im Datensatz mit "NA" anstatt mit "0" markiert, was zur Folge hat, dass nur die 9 Immobilien mit Pool in der Auflistung vorkommen. Dies verfälscht beispielsweise den Mittelwert und andere Werte deutlich. Dies sollte im Datensatz in Aufgabe 3 auf jeden Fall bereinigt werden.

In [197]:

```
# Datentypen und weitere Infos
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 28 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Grundstück in qm     2000 non-null   int64  
 1   Grundstücksform      2000 non-null   object 
 2   Steigung             2000 non-null   object 
 3   Bezirk               2000 non-null   object 
 4   Zone                 2000 non-null   object 
 5   Lage                 2000 non-null   object 
 6   Typ                  2000 non-null   object 
 7   Zustand              2000 non-null   int64  
 8   Gebaut               2000 non-null   int64  
 9   Renoviert            2000 non-null   int64  
 10  Zustand Fassade      2000 non-null   int64  
 11  Kellerflche in qm    2000 non-null   int64  
 12  Heizung              2000 non-null   object 
 13  Heizungsqualitt      2000 non-null   object 
 14  Klimaanlage          2000 non-null   object 
 15  Erster Stock in qm   2000 non-null   int64  
 16  Zweiter Stock in qm  2000 non-null   int64  
 17  Wohnflche in qm      2000 non-null   int64  
 18  Schlafzimmer         2000 non-null   int64  
 19  Küchen               2000 non-null   int64  
 20  Küchenqualitt        2000 non-null   int64  
 21  Rume                 2000 non-null   int64  
 22  Garage Typ           1904 non-null   object 
 23  Garagenkapazitt      2000 non-null   int64  
 24  Pool                 9 non-null      float64
 25  Verkaufsmonat        2000 non-null   int64  
 26  Verkaufsjahr         2000 non-null   int64  
 27  Preis                2000 non-null   int64  
dtypes: float64(1), int64(17), object(10)
memory usage: 359.4+ KB
```

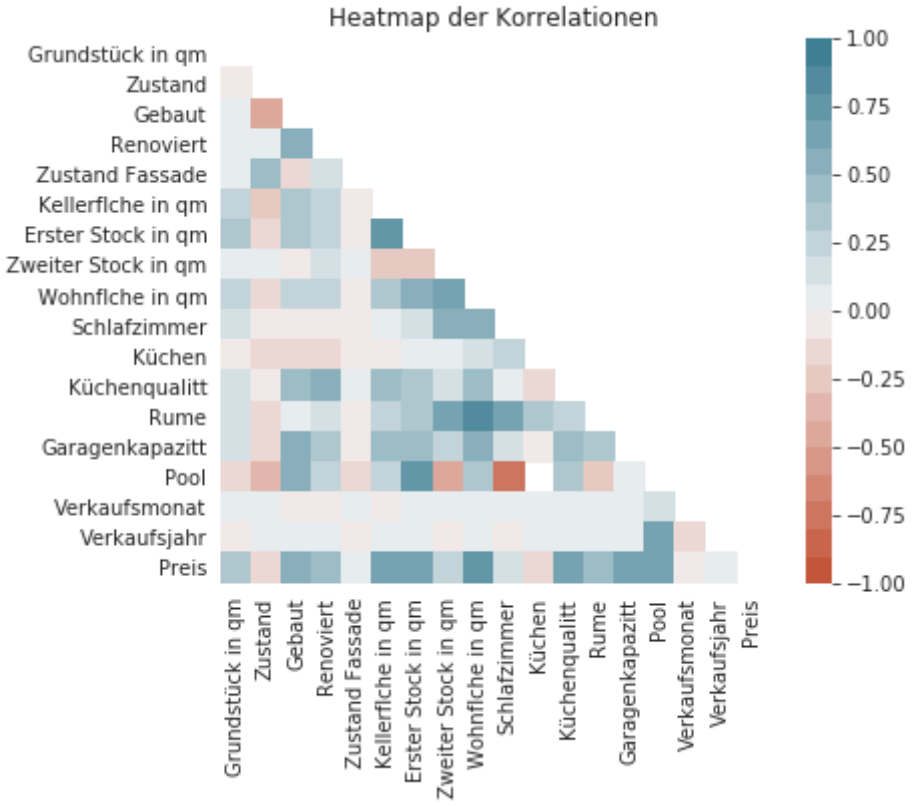
Es existieren 18 numerische und 10 kategorische Werte. Bei der Spalte 'Pool' handelt es sich um einen Float (Int wäre hier sinnvoller), was jedoch für die Untersuchung nicht weiter schlimm ist.

Korrelation in einer Heatmap zur Übersicht

Um einen Überblick für die Korrelationen zu bekommen, plotten wir diese in einer Heatmap

In [198]:

```
# Korrelationen berechnen
corr = df.corr()

# Maske für Heatmap erstellen
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("white"):
    f, ax = plt.subplots(figsize=(7, 5))
    ax = sns.heatmap(corr, cmap=sns.diverging_palette(20, 220, n=20), mask=mask, vmax=1
    plt.title("Heatmap der Korrelationen")
```



Korrelationen prüfen

In [199]:

```
# Preiskorrelationen anzeigen
df.corr()["Preis"].sort_values(ascending=False)
```

Out[199]:

```
Preis                  1.000000
Wohnflche in qm        0.740241
Erster Stock in qm     0.652158
Kellerflche in qm      0.651032
Küchenqualitt          0.642411
Garagenkapazitt        0.640781
Pool                   0.629721
Gebaut                 0.519263
Rume                   0.495667
Renoviert              0.479822
Grundstück in qm       0.314255
Zweiter Stock in qm    0.288142
Schlafzimmer           0.175599
Verkaufsjahr           0.023610
Zustand Fassade        0.015150
Verkaufsmonat         -0.002150
Zustand               -0.107618
Küchen                -0.114129
Name: Preis, dtype: float64
```

Es existieren folgende höhere Korrelationen zum Preis: Baujahr(0.52), Renovierungsjahr(0.48), Kellerfläche(0.65), Fläche erster Stock(0.65), Wohnfläche(0.74), Küchenqualität(0.64), Raumanzahl(0.5), Garagenkapazität(0.64), Pool(0.63). Achtung: die hohe Korrelation zwischen Preis und Pool ist vermutlich durch die oben genannten Null-Werte entstanden. Diesen Wert muss man mit Vorsicht genießen.

Die negative Korrelation zum Zustand(-0.10) und zur Anzahl der Küchen(-0.11) erscheint uns unlogisch. Diese betrachten wir gleich kurz genauer.

Der Zustand der Fassade sowie der Verkaufsmonat bzw. das Verkaufsjahr besitzen quasi keine Korrelation zum Preis.

Negative Korrelationen betrachten

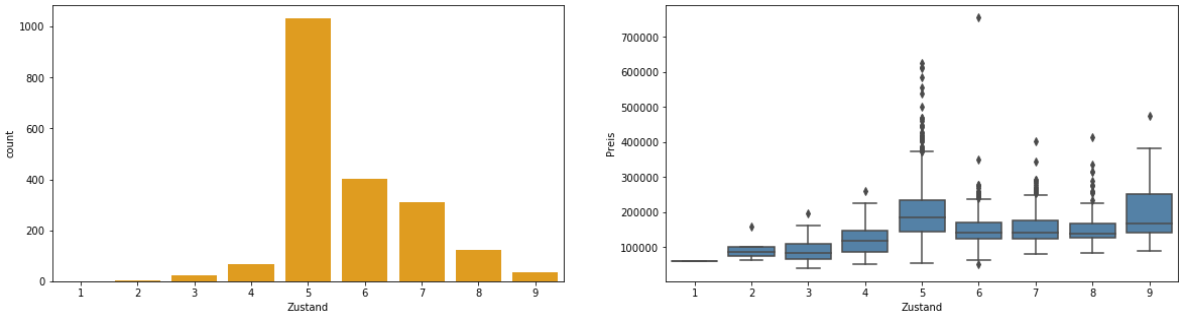
...um sich diese besser erklären zu können

In [200]:

```
# Subplots erstellen
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(20,5))

# Plots
column = "Zustand"
sns.countplot(x = column, data = df, ax=ax1, color="orange")
sns.boxplot(x = column, y = "Preis", data = df, ax=ax2, color="steelblue")
```

Out[200]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x4ab1590>
```



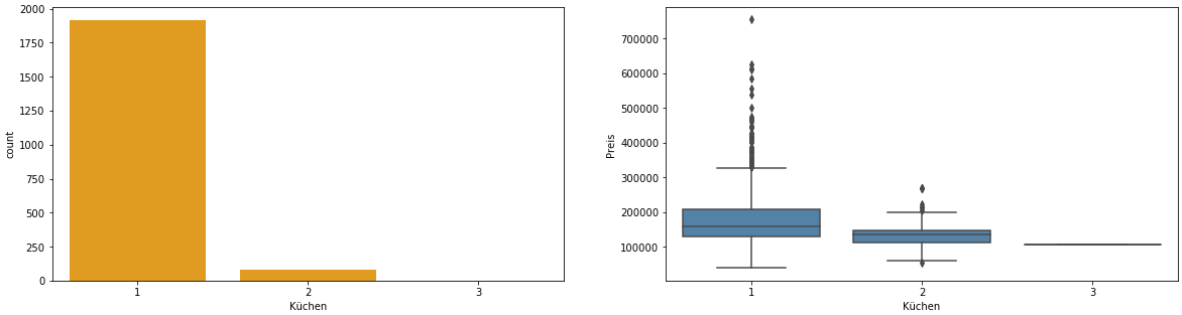
Mehr als die Hälfte aller Immobilien haben den Zustand 5. Diese fallen jedoch fast alle unter die teuersten Immobilien, wodurch die Korrelation beeinflusst wird.

In [201]:

```
# Subplots erstellen
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(20,5))

column = "Küchen"

# Plots
sns.countplot(x = column, data = df, ax=ax1, color="orange")
sns.boxplot(x = column, y = "Preis", data = df, ax=ax2, color="steelblue")
```

Out[201]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x4b7da30>
```

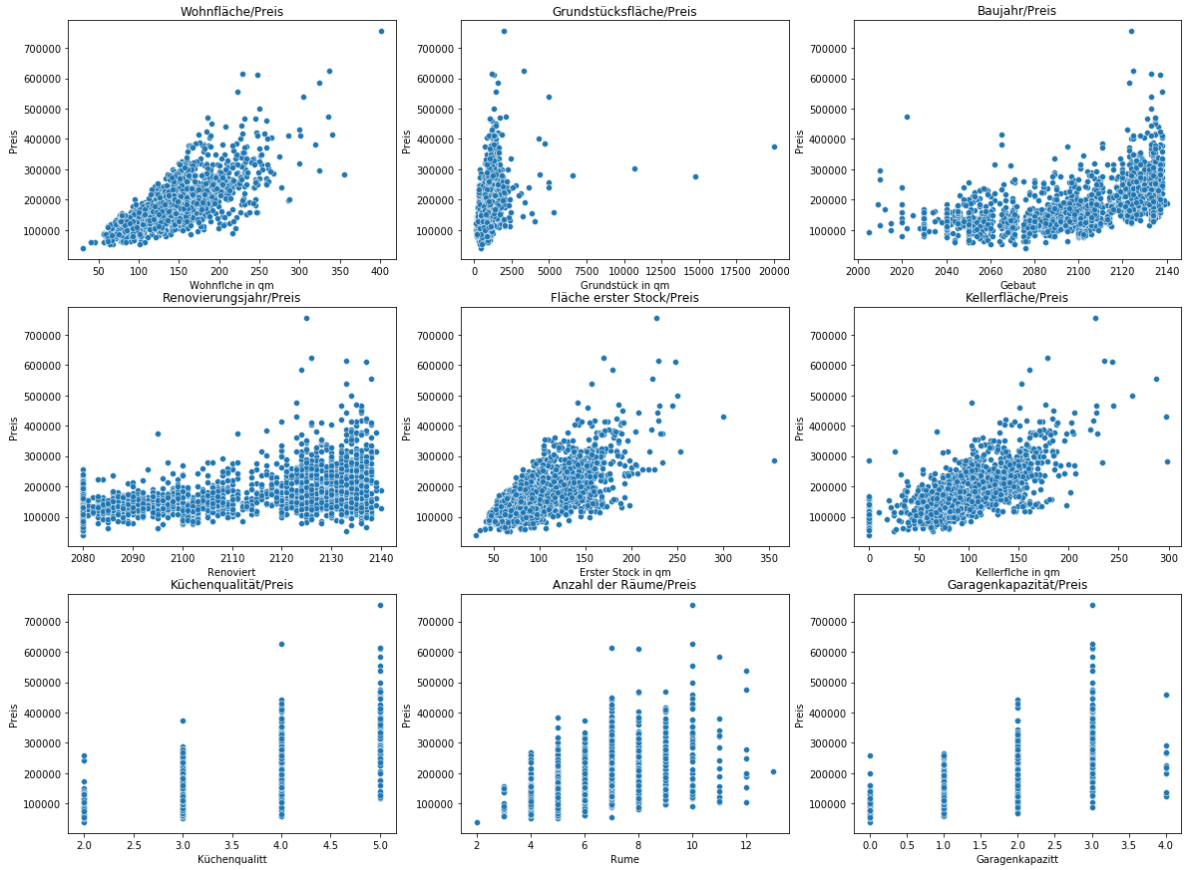


Ähnlich ist es bei der Anzahl der Küchen. Über 90% der Immobilien besitzen nur eine Küche. Von den Immobilien mit mehr als 2 Küchen erreicht keine einen Preis über 300.000. So erklärt sich die niedrige (negative) Korrelation.

Positive Korrelationen betrachten

In [202]:

```
# Subplots erstellen
fig, ((ax1, ax2, ax3), (ax4, ax5, ax6), (ax7, ax8, ax9)) = plt.subplots(3,3, figsize=(2

# Plotting
sns.scatterplot(data=df, x = "Wohnflche in qm", y = "Preis", ax=ax1)
ax1.set_title("Wohnfläche/Preis")
sns.scatterplot(data=df, x = "Grundstück in qm", y = "Preis", ax=ax2)
ax2.set_title("Grundstücksfläche/Preis")
sns.scatterplot(data=df, x = "Gebaut", y = "Preis", ax=ax3)
ax3.set_title("Baujahr/Preis")
sns.scatterplot(data=df, x = "Renoviert", y = "Preis", ax=ax4)
ax4.set_title("Renovierungsjahr/Preis")
sns.scatterplot(data=df, x = "Erster Stock in qm", y = "Preis", ax=ax5)
ax5.set_title("Fläche erster Stock/Preis")
sns.scatterplot(data=df, x = "Kellerflche in qm", y = "Preis", ax=ax6)
ax6.set_title("Kellerfläche/Preis")
sns.scatterplot(data=df, x = "Küchenqualitt", y = "Preis", ax=ax7)
ax7.set_title("Küchenqualität/Preis")
sns.scatterplot(data=df, x = "Rume", y = "Preis", ax=ax8)
ax8.set_title("Anzahl der Räume/Preis")
sns.scatterplot(x="Garagenkapazitt", y="Preis", data=df, ax=ax9)
ax9.set_title("Garagenkapazität/Preis")
plt.show()
```



Insbesondere bei der Wohnfläche, der Fläche des ersten Stocks und bei der Kellerfläche ist die höhere Korrelation zum Preis am Graph zu erkennen (cone-shaped, ansatzweise linearer Verlauf).

Neuere Häuser sind durchschnittlich teurer, jedoch ist hier die Korrelation weniger stark als oben. Beim Renovierungsjahr muss aufgepasst werden, da im Datensatz oft Baujahr = Renovierungsjahr vermerkt ist und die Immobilie somit noch nicht renoviert wurde.

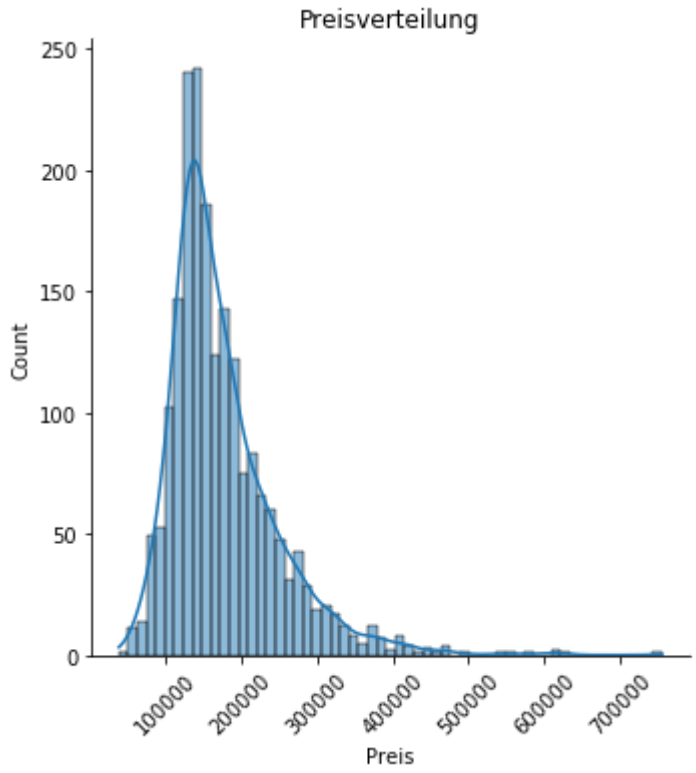
Einzelne Ausreißer lassen sich in fast jedem der Graphen erkennen, bei der Grundstücksgröße sind diese besonders erkennbar. Die Ausreißer werden später in Aufgabe 3 entfernt.

Untersuchung der Preisverteilung

In [203]:

```
# Preisverteilung plotten
sns.displot(x="Preis", data=df, kde=True)
plt.xticks(rotation=45)
plt.title("Preisverteilung")
plt.show()
```

```
c:\users\buehring\appdata\local\programs\python\python37-32\lib\site-package
s\matplotlib\cbook\__init__.py:1402: FutureWarning: Support for multi-dimens
ional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a
future version.  Convert to a numpy array before indexing instead.
  ndim = x[:, None].ndim
c:\users\buehring\appdata\local\programs\python\python37-32\lib\site-package
s\matplotlib\axes\_base.py:276: FutureWarning: Support for multi-dimensional
indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future
version.  Convert to a numpy array before indexing instead.
  x = x[:, np.newaxis]
c:\users\buehring\appdata\local\programs\python\python37-32\lib\site-package
s\matplotlib\axes\_base.py:278: FutureWarning: Support for multi-dimensional
indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future
version.  Convert to a numpy array before indexing instead.
  y = y[:, np.newaxis]
```

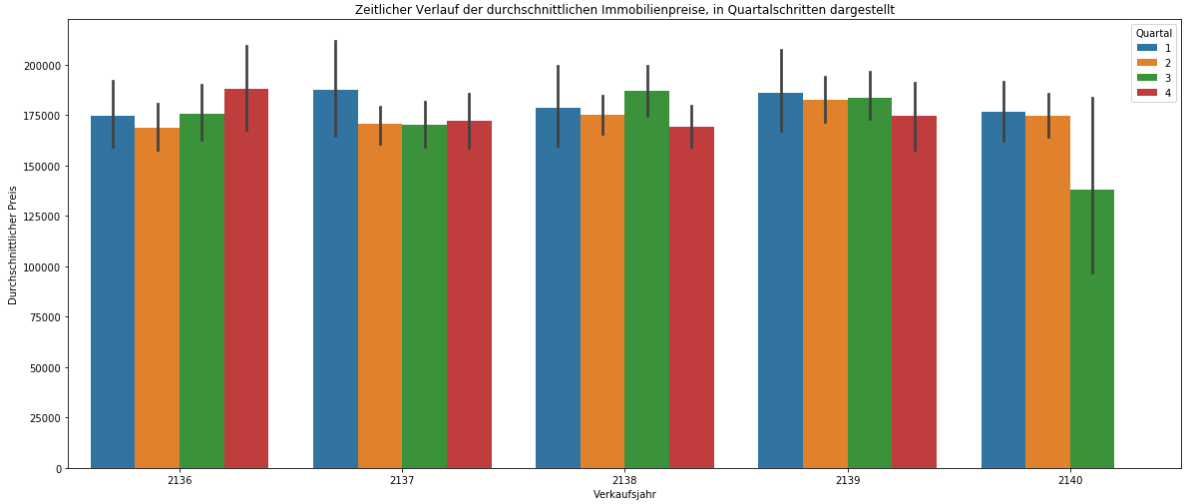


Die Verteilung der Preise ist stark nach rechts verschoben. Es ist ratsam, eine Normalverteilung zu erreichen, damit die Performance der Regression in Aufgabe 3 nicht durch die Ausreißer beeinflusst wird. Die Entfernung der Ausreißer folgt in Aufgabe 3.

Betrachtung der zeitlichen Daten

In [204]:

```
# Monate in Quartale unterteilen
quartal = df.copy()
quartal["Verkaufsmonat"] = quartal["Verkaufsmonat"].replace([1,2,3], 1)
quartal["Verkaufsmonat"] = quartal["Verkaufsmonat"].replace([4,5,6], 2)
quartal["Verkaufsmonat"] = quartal["Verkaufsmonat"].replace([7,8,9], 3)
quartal["Verkaufsmonat"] = quartal["Verkaufsmonat"].replace([10,11,12], 4)
quartal = quartal["Verkaufsmonat"].rename("Quartal")

a4_dims = (20, 8.27)
fig, ax = plt.subplots(figsize=a4_dims)
sns.barplot(data=df, x = "Verkaufsjahr", y = "Preis", hue=quartal, ax=ax)
plt.title("Zeitlicher Verlauf der durchschnittlichen Immobilienpreise, in Quartalschrit
plt.ylabel("Durchschnittlicher Preis")
plt.show()
```

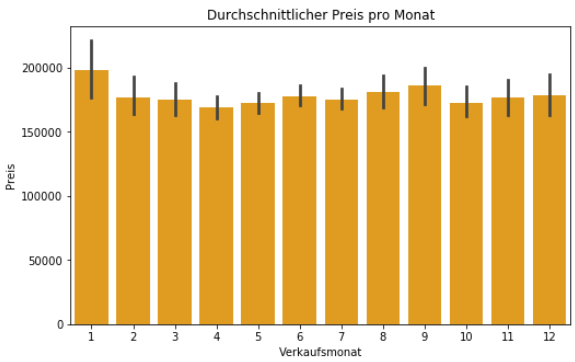


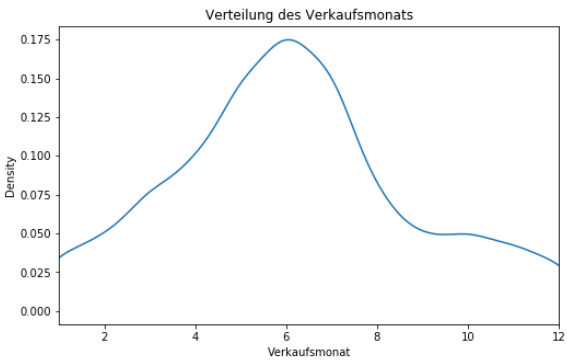
Die durchschnittlichen Preise der Immobilien sind über die dokumentierte Zeit relativ konstant geblieben, es lassen sich keine Trends erkennen. Der niedrige Wert im dritten Quartal 2140 lässt sich eher auf die geringe Datenmenge als auf einen Immobiliencrash zurückführen. Um den zeitlichen Verlauf akkurat darzustellen, werden deutlich mehr Daten benötigt.

In [205]:

```
# Subplots erstellen
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(18,5))

# Plots
sns.barplot(x="Verkaufsmonat", y="Preis", data=df, ax=ax1, color="orange")
ax1.set_title("Durchschnittlicher Preis pro Monat")

sns.kdeplot(x="Verkaufsmonat", data=df, ax=ax2)
plt.xlim(1,12)
ax2.set_title("Verteilung des Verkaufsmonats")
plt.show()
```

```
c:\users\buehring\appdata\local\programs\python\python37-32\lib\site-package
s\matplotlib\cbook\__init__.py:1402: FutureWarning: Support for multi-dimens
ional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a
future version.  Convert to a numpy array before indexing instead.
  ndim = x[:, None].ndim
c:\users\buehring\appdata\local\programs\python\python37-32\lib\site-package
s\matplotlib\axes\_base.py:276: FutureWarning: Support for multi-dimensional
indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future
version.  Convert to a numpy array before indexing instead.
  x = x[:, np.newaxis]
c:\users\buehring\appdata\local\programs\python\python37-32\lib\site-package
s\matplotlib\axes\_base.py:278: FutureWarning: Support for multi-dimensional
indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future
version.  Convert to a numpy array before indexing instead.
  y = y[:, np.newaxis]
```





Diese beiden Diagramme zeigen uns, dass die Preise zwar von Monat zu Monat nicht wesentlich variieren, dass es aber Monate gibt, in denen viel mehr Häuser verkauft werden als in anderen. Wenn wir vorhersagen wollten, wann ein Haus verkauft wird, wäre diese Information sehr nützlich. Da wir stattdessen eher am tatsächlichen Preis interessiert sind, ist das weniger der Fall.

Kategorische Werte betrachten

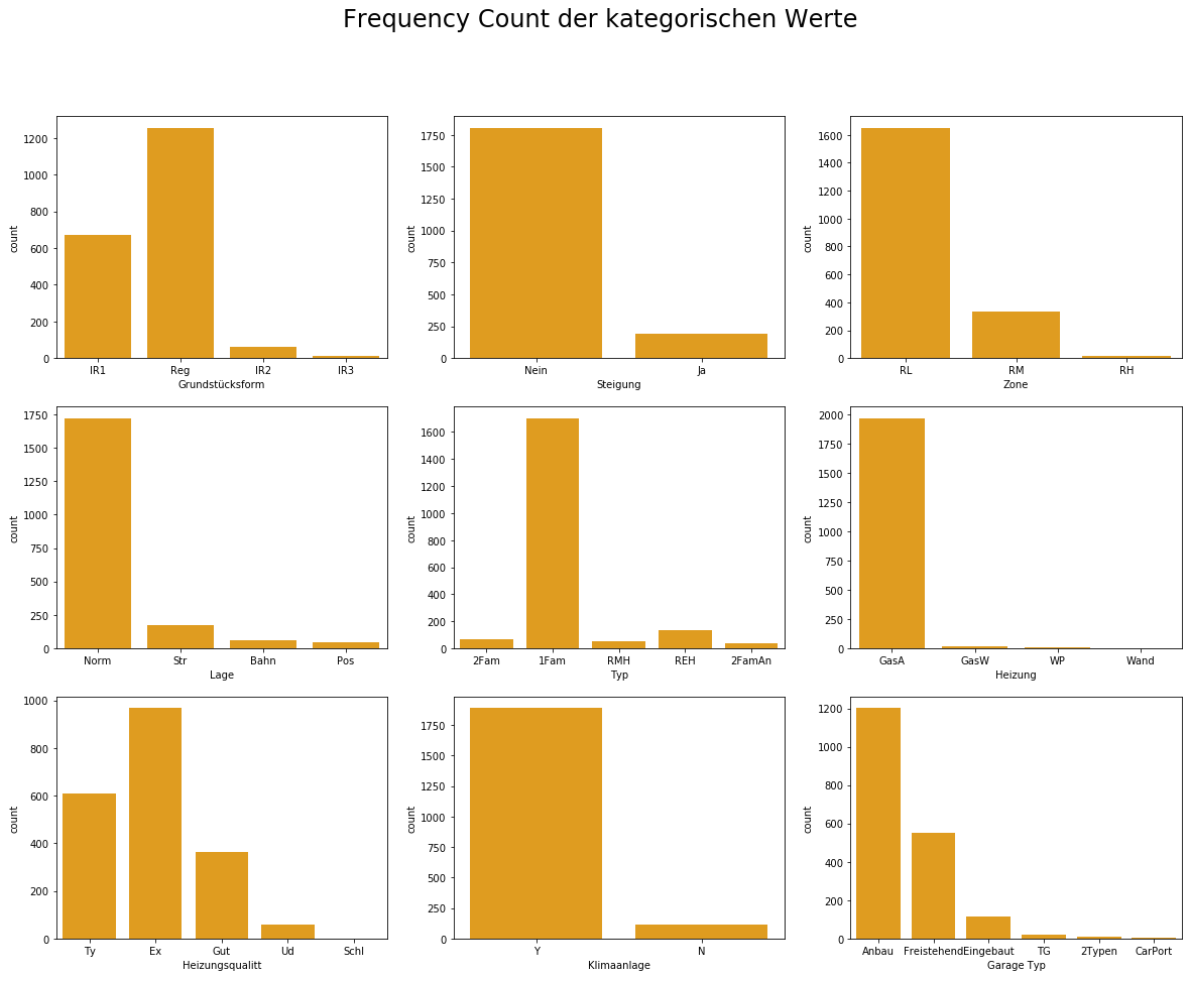
Da die kategorischen Werte in den Korrelationen nicht vorkommen, betrachten wir diese hier nochmal genauer, um uns einen Überblick zu verschaffen.

In [206]:

```
# Kategorische Werte
categorical_var = ["Grundstücksform", "Steigung", "Zone", "Lage", "Typ", "Heizung", "He
```

In [207]:

```
# Subplots erstellen
fig, axes = plt.subplots(3,3, figsize=(20,15))
plt.suptitle('Frequency Count der kategorischen Werte', fontsize=24)

# Plotting
for i, ax in zip(categorical_var, axes.flat):
    sns.countplot(x = i, data = df, ax=ax, color="orange")

plt.show()
```

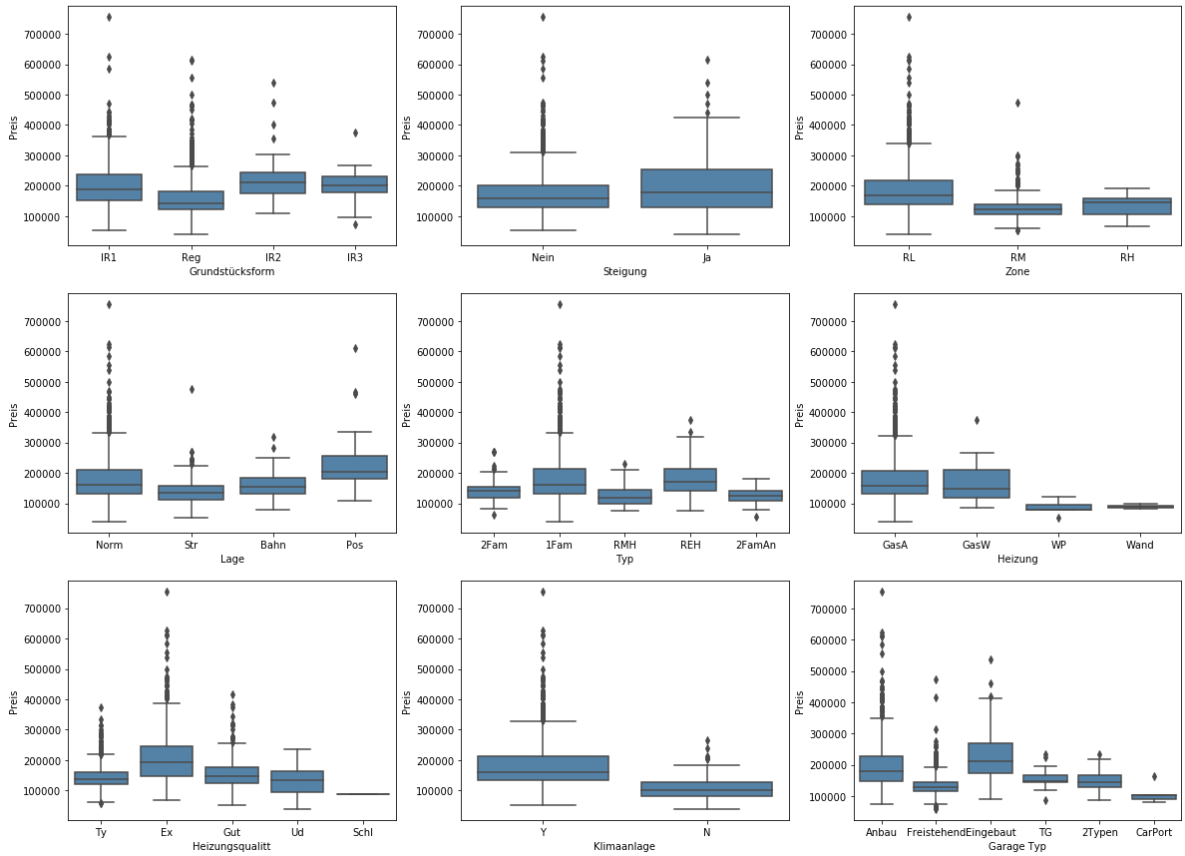


In der Häufigkeit der einzelnen Kategorien gibt es starke Unterschiede. Es lässt sich erkennen, dass meistens eine Kategorie der Standard und die anderen Sonderfälle sind (zum Beispiel haben über 90% der Immobilien eine Klimaanlage oder eine Heizung mit dem Typ GasA).

In [208]:

```
# Subplots erstellen
fig, axes = plt.subplots(3,3, figsize=(20,15))
plt.suptitle('Übersicht der kategorischen Werte in Relation zum Preis', fontsize=24)

# Plotting
for i, ax in zip(categorical_var, axes.flat):
    sns.boxplot(x = i, y = "Preis", data = df, ax=ax, color="steelblue")

plt.show()
```

Übersicht der kategorischen Werte in Relation zum Preis



An den kategorischen Plots lässt sich erkennen, dass viele Ausreißer unter den Daten sind. Wir sind uns nicht sicher, wie hier die Korrelationen aussehen bzw. wie sich bei den kategorischen Werten die Ausreißer sinnvoll entfernen lassen.

Wichtige Erkenntnisse

- Die Verteilung der Preise (und auch einiger anderer Daten) ist verschoben. Es sollten zumindest die Preise auf eine Normalverteilung angepasst werden, um eine bessere Vorhersage zu ermöglichen. Es wäre

vermutlich auch ratsam, alle Ausreißer in den anderen Spalten zu entfernen. Das sprengt wahrscheinlich jedoch den Rahmen des Projekts. Wir können es jedoch versuchen, die Ausreißer aus den Spalten zu entfernen, welche ohnehin bereits eine hohe Korrelation aufweisen. Wir vermuten, dass die Preisvorhersage in Aufgabe 3 dadurch verbessert werden kann.
- Es wäre super interessant, die Korrelationen nach Bereinigung der Daten erneut zu prüfen, da diese sich durch das Entfernen von Ausreißern verändern.
- Die Korrelationen der kategorischen Werte lassen sich in dieser Form nur schwer überprüfen.
- Die Wohnfläche, Küchenqualität und Garagenkapazität beeinflussen den Preis am meisten.
- Bei den Plots zur Häufigkeit der kategorischen Werte lässt sich beobachten, dass meistens eine Kategorie der Standard ist. Zum Beispiel sind über 80% aller Immobilien Einfamilienhäuser, besitzen eine Klimaanlage oder sind an keine Steigung gebaut.

Aufgabe 3: Data Preparation und Modeling (6P)

Bereinigen Sie die Daten, falls notwendig. Führen Sie Feature Engineering durch, wenn notwendig. Führen Sie mit geeigneten Verfahren eine Vorhersage des Preises (Preis) durch. Eine davon soll eine erklärbare, verständliche und interpretierbare lineare Regression sein. Erklären Sie diese im Detail in Bezug auf die Ziele aus Aufgabe 1. Wählen Sie mehrere geeignete Regressionsverfahren. Vergleichen Sie die angewendete Verfahren grafisch. Begründen Sie Ihre Wahl in Bezug auf die Ziele.

Data preparation

In [209]:

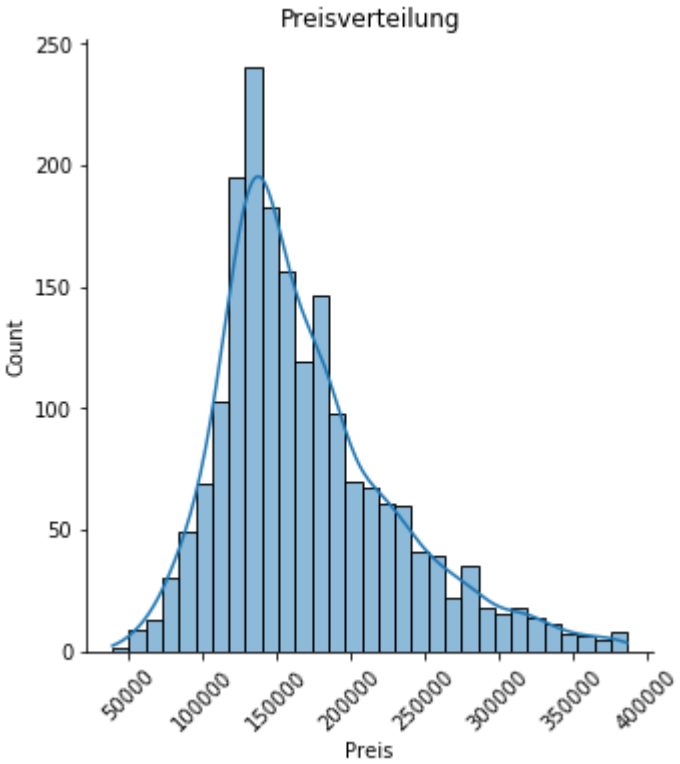
```
# Daten bereinigen mit Z-score

import scipy.stats as stats

# Absoluten Z-Score für die Spalten finden, aus welchen Ausreißer entfernt werden solle
# (Nur die Ausreißer der numerischen Daten mit hoher Korrelation werden entfernt,
# bei den kategorischen Werten haben wir uns hier etwas schwer getan)
z_score = np.abs(stats.zscore(df[["Preis", "Wohnflche in qm", "Grundstück in qm", "Erst

# Nur die Zeilen behalten mit einem absoluten Z-score < 3
df = df[(z_score<3).all(axis=1)]

# Herausfinden, wie viele Zeilen noch übrig sind
print("Shape: ", df.shape)
```

```
Shape:  (1908, 28)
```

In [210]:

```
# Preisverteilung plotten
sns.displot(data=df, x="Preis", kde=True)
plt.title("Preisverteilung")
plt.xticks(rotation=45)
plt.show()
```

```
c:\users\buehring\appdata\local\programs\python\python37-32\lib\site-package
s\matplotlib\cbook\__init__.py:1402: FutureWarning: Support for multi-dimens
ional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a
future version.  Convert to a numpy array before indexing instead.
  ndim = x[:, None].ndim
c:\users\buehring\appdata\local\programs\python\python37-32\lib\site-package
s\matplotlib\axes\_base.py:276: FutureWarning: Support for multi-dimensional
indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future
version.  Convert to a numpy array before indexing instead.
  x = x[:, np.newaxis]
c:\users\buehring\appdata\local\programs\python\python37-32\lib\site-package
s\matplotlib\axes\_base.py:278: FutureWarning: Support for multi-dimensional
indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future
version.  Convert to a numpy array before indexing instead.
  y = y[:, np.newaxis]
```



Die Preisverteilung ist immer noch nach rechts verschoben... jedoch schon deutlich weniger als zuvor. Damit geben wir uns zufrieden :)

In [211]:

```
# Preiskorrelationen anzeigen
df.corr()["Preis"].sort_values(ascending=False)
```

Out[211]:

```
Preis                 1.000000
Wohnflche in qm       0.724047
Garagenkapazitt       0.661030
Küchenqualitt         0.627173
Kellerflche in qm     0.610268
Erster Stock in qm    0.599740
Gebaut                0.573680
Renoviert             0.507507
Rume                  0.487546
Grundstück in qm      0.406444
Pool                  0.346800
Zweiter Stock in qm   0.286879
Schlafzimmer          0.184410
Verkaufsjahr          0.033805
Verkaufsmonat         0.013414
Zustand Fassade       0.006019
Zustand              -0.118397
Küchen               -0.142431
Name: Preis, dtype: float64
```

In [212]:

```
# Pool-Daten mit 'NA' mit '0' befüllen
df['Pool'] = df['Pool'].fillna(0)

# Garagen Typ-Daten mit 'NA' mit 'Keine Garage' befüllen
df['Garage Typ'] = df['Garage Typ'].fillna('Keine Garage')

df.head()
```

Out[212]:

	Grundstück in qm	Grundstücksform	Steigung	Bezirk	Zone	Lage	Typ	Zustand	Gebaut	Re
0	898	IR1	Nein	Somerset	RL	Norm	2Fam	4	2107	
1	1326	Reg	Nein	North East	RL	Norm	1Fam	5	2133	
2	725	Reg	Nein	Somerset	RL	Norm	1Fam	7	2096	
3	725	Reg	Nein	Somerset West	RL	Norm	1Fam	5	2135	
4	697	Reg	Nein	Miller	RL	Norm	1Fam	5	2129	

5 rows × 28 columns

In [213]:

```
# Spalten des Dataframes ausgeben
df.columns
```

Out[213]:

```
Index(['Grundstück in qm', 'Grundstücksform', 'Steigung', 'Bezirk', 'Zone',
       'Lage', 'Typ', 'Zustand', 'Gebaut', 'Renoviert', 'Zustand Fassade',
       'Kellerflche in qm', 'Heizung', 'Heizungsqualitt', 'Klimaanlage',
       'Erster Stock in qm', 'Zweiter Stock in qm', 'Wohnflche in qm',
       'Schlafzimmer', 'Küchen', 'Küchenqualitt', 'Rume', 'Garage Typ',
       'Garagenkapazitt', 'Pool', 'Verkaufsmonat', 'Verkaufsjahr', 'Preis'],
      dtype='object')
```

In [214]:

```
# Dataframe in Daten und Output (X und y) unterteilen
X = df[['Grundstück in qm', 'Grundstücksform', 'Steigung',
        'Bezirk', 'Zone', 'Lage', 'Typ', 'Zustand', 'Gebaut', 'Renoviert',
        'Zustand Fassade', 'Kellerflche in qm', 'Heizung', 'Heizungsqualitt',
        'Klimaanlage', 'Erster Stock in qm', 'Zweiter Stock in qm',
        'Wohnflche in qm', 'Schlafzimmer', 'Küchen', 'Küchenqualitt', 'Rume',
        'Garage Typ', 'Garagenkapazitt', 'Pool', 'Verkaufsmonat', 'Verkaufsjahr']]
y = df['Preis']

# Kategorische Werte in numerische Werte umwandeln
X = pd.get_dummies(data=X)
X.head()
```

Out[214]:

	Grundstück in qm	Zustand	Gebaut	Renoviert	Zustand Fassade	Kellerflche in qm	Erster Stock in qm	Zweiter Stock in qm	Wohnflche in qm
0	898	4	2107	2107	3	183	183	0	183
1	1326	5	2133	2133	3	122	113	109	222
2	725	7	2096	2138	3	80	83	0	83
3	725	5	2135	2135	3	83	83	74	157
4	697	5	2129	2129	3	92	96	72	168

5 rows × 78 columns

Regressionsmodelle

In [215]:

```
# Aufteilung der Daten in Training und Test
from sklearn.model_selection import train_test_split

# Testdaten sind 20%, Trainingsdaten sind 80% der Gesamtdaten
# Randomstate wurde gesetzt, um später genau auf die Ergebniswerte eingehen zu können
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state
```

In [216]:

```
#Prüfen der Dimensionen
print("Größe der Trainingsdaten:\t", X_train.shape,"\t==>\t",y_train.shape)
print("Größe der Testdaten:\t\t", X_test.shape,"\t==>\t",y_test.shape)
print("Größe aller Daten:\t\t", X.shape,"\t==>\t",y.shape)
```

```
Größe der Trainingsdaten:       (1526, 78)      ==>     (1526,)
Größe der Testdaten:            (382, 78)       ==>     (382,)
Größe aller Daten:              (1908, 78)      ==>     (1908,)
```

In [217]:

```
# Bewertungsumgebung für Regressionen anpassbar
def count_errors_in_range(y_pred, umgebung):
    summe = 0
    count = 0
    for i in y_test:
        under_range = i - umgebung
        upper_range = i + umgebung
        if y_pred_lin[count] > under_range:
            if y_pred_lin[count] < upper_range:
                summe = summe + 1
        count = count + 1
    fehler = X_test.shape[0] - summe
    print("Anzahl der falsch zugeordneten Preise: %d von %d Datensätzen (%d Prozent) be
    return
```

In [218]:

```
# Bibliotheken importieren
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import max_error

# Funktion zum Berechnen der gewünschten Metriken (MAE, MSE, ...)
def calculate_metrics(y_test, y_pred):

    # MSE berechnen
    mse = mean_squared_error(y_test, y_pred)
    print("MSE: %d" % mse)

    # RMSE berechnen
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    print("RMSE: %d" % rmse)

    # R2 berechnen
    r2 = r2_score(y_test, y_pred)
    print("R2:", round(r2, 5))

    # MAE berechnen
    mae = mean_absolute_error(y_test, y_pred)
    print("MAE: %d" % mae)

    # MAPE berechnen
    mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
    print("MAPE: ", round(mape,3), "%")

    # MAX berechnen
    maxerror = max_error(y_test, y_pred)
    print("MAX Error: %d" % maxerror)

    return mse, rmse, r2, mae, mape, maxerror
```

In [219]:

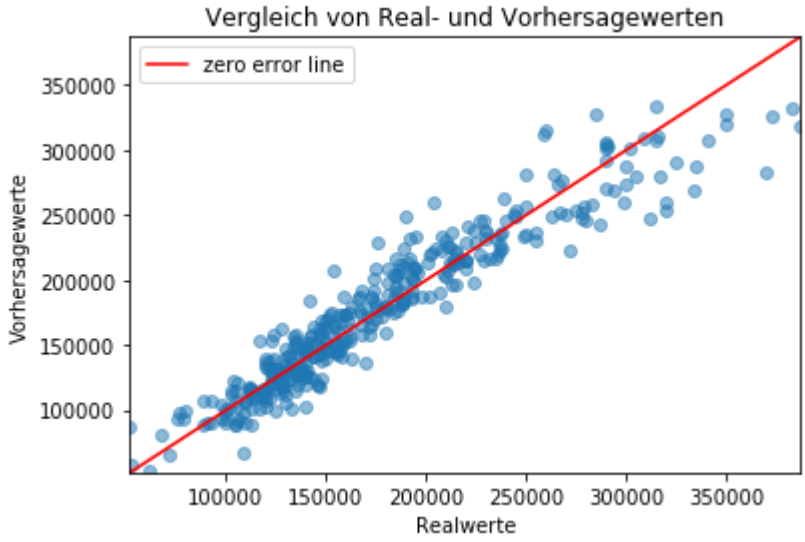
```
# Funktion um das Plotten der Regressionen zu vereinfachen
def create_regression_plots(y_test, y_pred):
    # Scatterplot: Real- vs. Vorhersagewerte
    plt.scatter(y_test,y_pred,alpha=0.5)
    plt.plot(
        [min(min(y_test),min(y_pred)),max(max(y_test),max(y_pred))],
        [min(min(y_test),min(y_pred)),max(max(y_test),max(y_pred))],
            c="red",
            label="zero error line")
    plt.xlim(min(min(y_test),min(y_pred)),max(max(y_test),max(y_pred)))
    plt.ylim(min(min(y_test),min(y_pred)),max(max(y_test),max(y_pred)))
    plt.title('Vergleich von Real- und Vorhersagewerten')
    plt.xlabel("Realwerte")
    plt.ylabel("Vorhersagewerte")
    plt.legend()
    plt.show()

    # Abweichungsverteilung
    sns.displot((y_pred - y_test), bins=50)
    plt.title('Abweichungsverteilung der Vorhersagewerte')
    plt.xlabel("Abweichung")
    plt.ylabel("Anzahl")
    plt.show()

    return
```

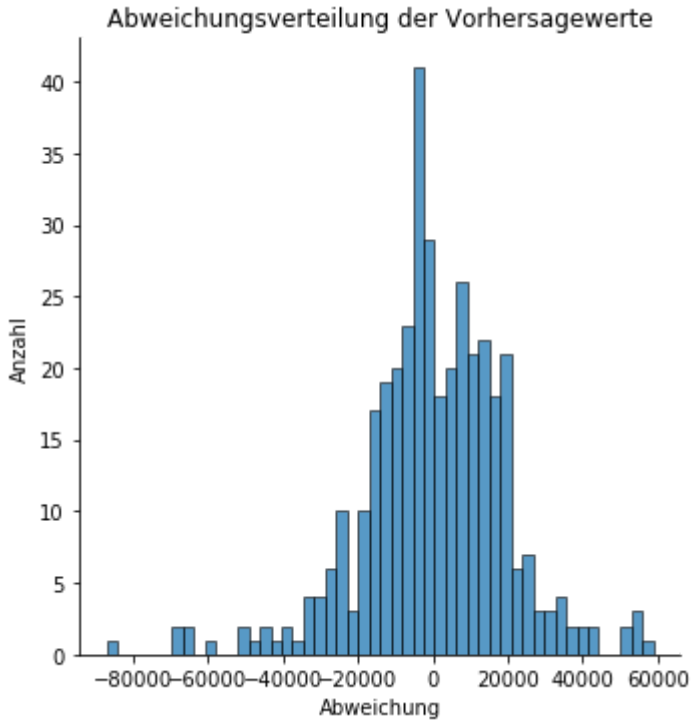
Lineare Regression

Die lineare Regression ist ein Modell zur Vorhersage eines bestimmten Attributs eines Datensatzes (in unserem Fall der Immobilienpreis). Die lineare Regression wird mit Hilfe unserer Trainingsdaten (X_train, y_train) trainiert um dann an Hand der Attribute der Daten (X_test) das gewünschte Attribut vorherzusagen (y_pred). Das Modell wird anschließend mit Testdaten (y_test) kontolliert und kann an Hand werten wie dem R2-Score, Mean absolut error (MAE), Mean absolut percentage error (MAPE) und Max error bewertet werden.

In [220]:

```
# Laden des Modells
from sklearn.linear_model import LinearRegression

# Instanz erzeugen
lin_reg = LinearRegression()

# Training
model = lin_reg.fit(X_train, y_train)

# Validierung mit Testdaten
y_pred_lin = lin_reg.predict(X_test)

# Plotting
create_regression_plots(y_test, y_pred_lin)

# Berechnen und ausgeben der Metriken
mse_lin, rmse_lin, r2_lin, mae_lin, mape_lin, maxerror_lin = calculate_metrics(y_test,
count_errors_in_range(y_pred_lin, 25000)
```





```
MSE: 384275303
RMSE: 19602
R2: 0.90362
MAE: 14450
MAPE:  8.442 %
MAX Error: 87015
Anzahl der falsch zugeordneten Preise: 54 von 382 Datensätzen (14 Prozent) b
ei einer maximalen Abweichung von 25000
```

Der MAE beträgt 14450 und beschreibt die durchschnittlichen absolute Fehler zwischen Vorhersagepreis und Realpreis.

Der MAPE, die durchschnittliche prozentuale Abweichung des Vorhersagepreis zum Realpreis, beträgt 8,442%.

Der R2-Score beträgt 0.90362 und beschreibt den Wert, der die durchschnittliche Variation des Modells für den Verkaufspreis anhand des Testdatensatzes erklärt.

Der MSE beträgt 384275303 und beschreibt den durchschnittlichen Fehler im Quadrat.

Der RMSE ist die Wurzel des MSE und beträgt 19602.

Der größte Fehler des Modells (MAX Error) beträgt 87015.

Ein gutes Modell kennzeichnet sich dadurch, dass Realwerte und Vorhersagewerte nicht weit auseinander liegen. In dem ersten Plot, der Real- und Vorhersagewerte vergleicht ist deshalb die Zero-Error-Line eingezeichnet, da Immobilien die auf dieser Linie liegen perfekt bewertet wurden. Außerdem ist zu erkennen, dass sich die Vorhersage an dieser Linie orientiert, was für die Qualiät der Vorhersage spricht

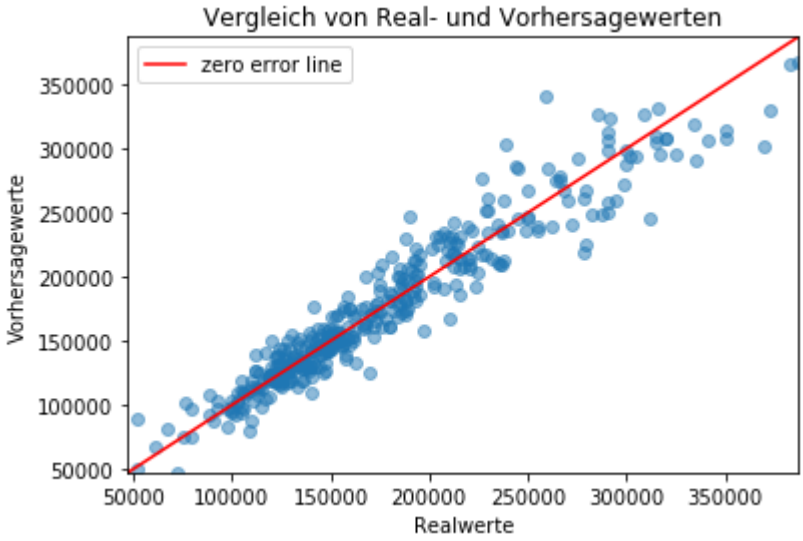
Die Abweichungsverteilung der Vorhersagewerte hat Ähnlichkeit mit einer Normalverteilung, was auch ein gutes Zeichen ist, da die Vorhersagen dann oft korrekt sind und größere Abweichungen nur selten auftreten.

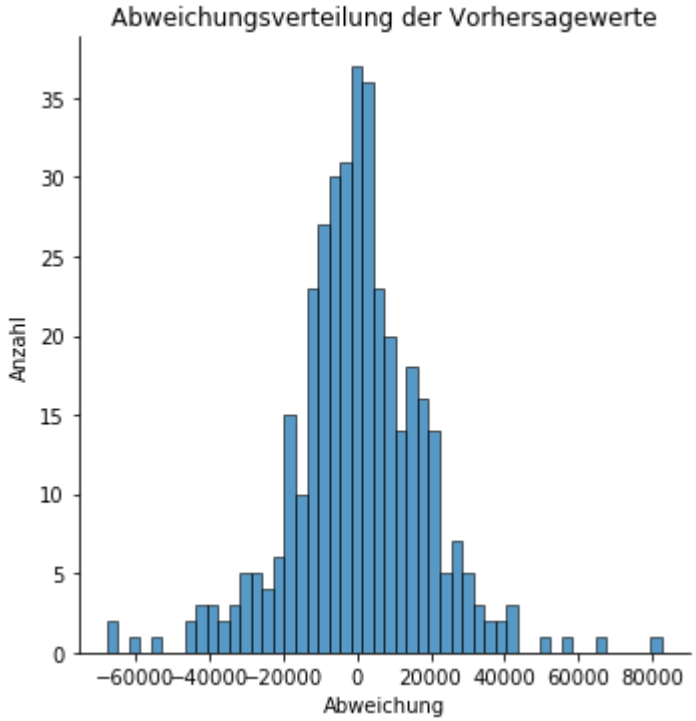
Im folgenden haben wir uns vier weitere Regressionsmodelle angesehen, um uns einen breiten Überblick über die Performance der verschiedenen Modelle zu verschaffen.

Gradient Boosting Regressionsverfahren

In [221]:

```
# Laden des Modells
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance

# Instanz erzeugen
reg = ensemble.GradientBoostingRegressor(learning_rate=0.1, n_estimators= 400)

# Training
reg.fit(X_train, y_train)

# Validierung mit Testdaten
y_pred_gbr = reg.predict(X_test)

# Plotting
create_regression_plots(y_test, y_pred_gbr)

# Berechnen und ausgeben der Metriken
mse_gbr, rmse_gbr, r2_gbr, mae_gbr, mape_gbr, maxerror_gbr = calculate_metrics(y_test,
count_errors_in_range(y_pred_gbr, 25000)
```



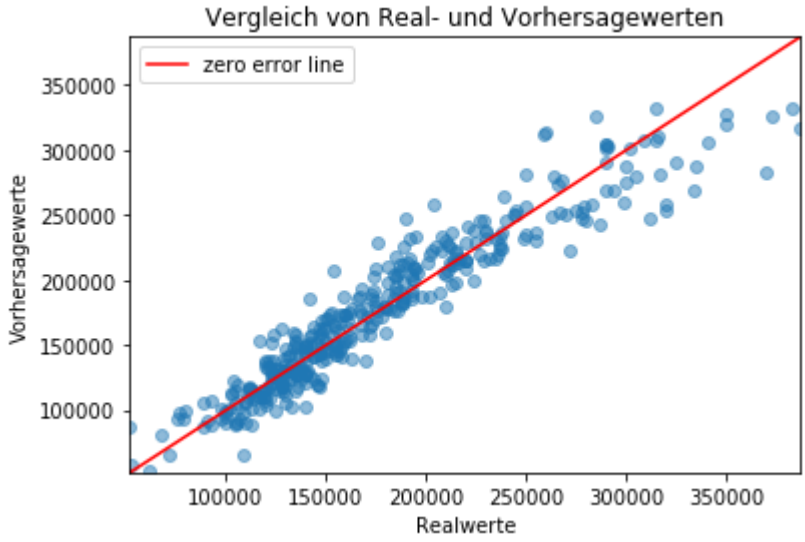


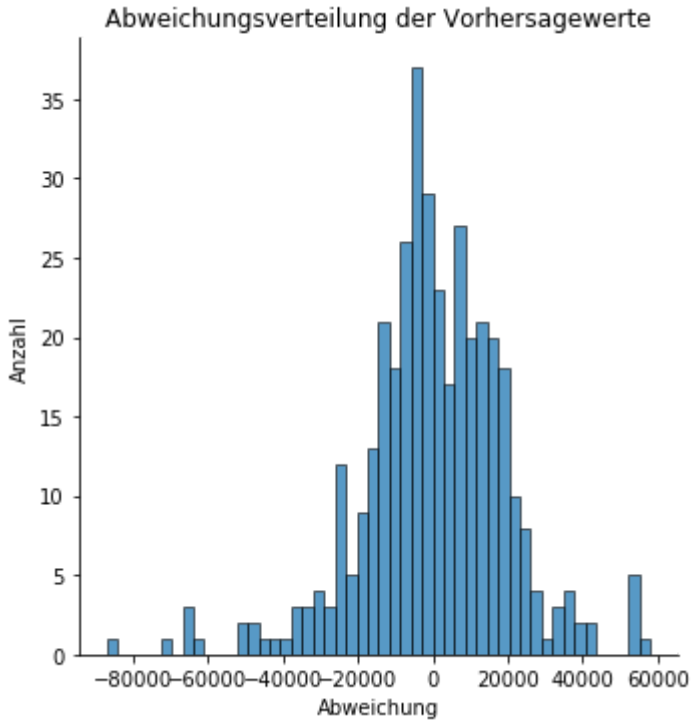

```
MSE: 327474483
RMSE: 18096
R2: 0.91786
MAE: 13120
MAPE:  7.49 %
MAX Error: 82459
Anzahl der falsch zugeordneten Preise: 54 von 382 Datensätzen (14 Prozent) b
ei einer maximalen Abweichung von 25000
```

Ridge Prediction

In [222]:

```
# Laden des Modells
from sklearn.linear_model import Ridge

# Instanz erzeugen
model = Ridge()

# Training
model.fit(X_train, y_train)

# Validierung mit Testdaten
y_pred_rid = model.predict(X_test)

# Plotting
create_regression_plots(y_test, y_pred_rid)

# Berechnen und ausgeben der Metriken
mse_rid, rmse_rid, r2_rid, mae_rid, mape_rid, maxerror_rid = calculate_metrics(y_test,
count_errors_in_range(y_pred_rid, 25000)
```



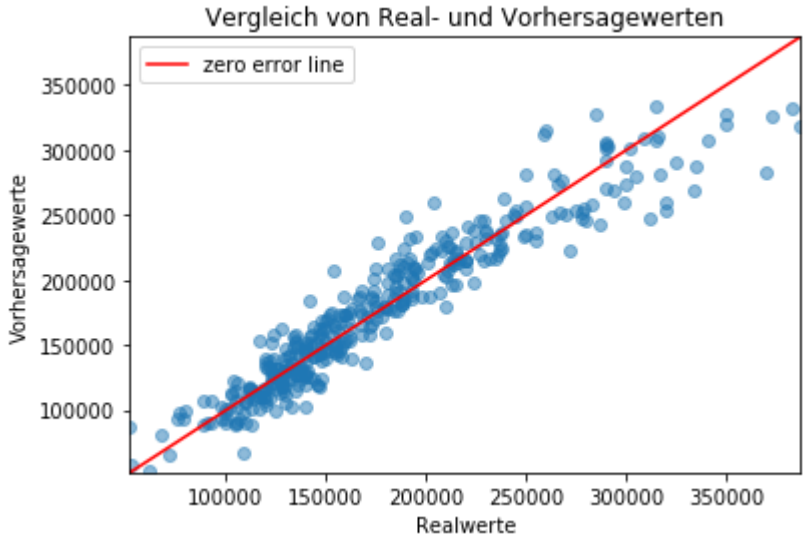


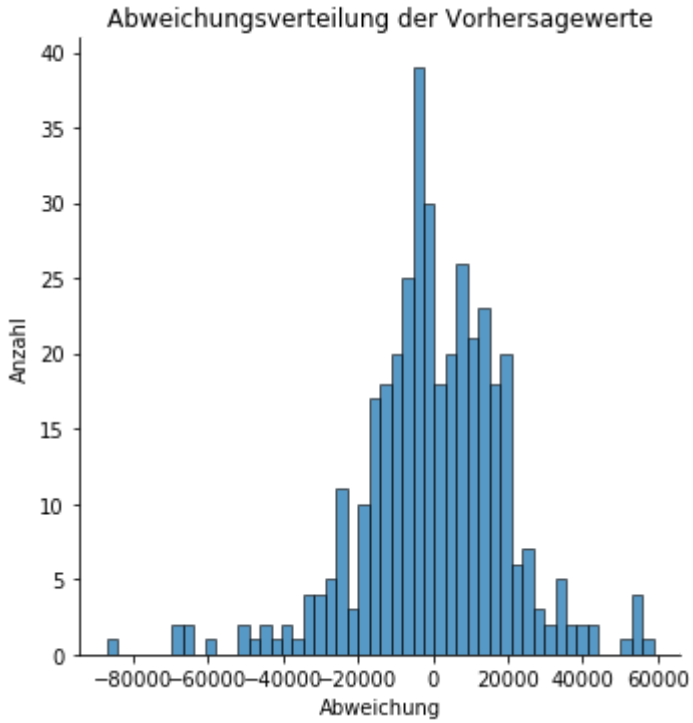
```
MSE: 384112228
RMSE: 19598
R2: 0.90366
MAE: 14434
MAPE:  8.428 %
MAX Error: 87005
Anzahl der falsch zugeordneten Preise: 54 von 382 Datensätzen (14 Prozent) b
ei einer maximalen Abweichung von 25000
```

Lasso Prediction

In [223]:

```
# Laden des Modells
from sklearn.linear_model import Lasso

# Instanz erzeugen
model = Lasso(max_iter=5000)

# Training
model.fit(X_train, y_train)

# Validierung mit Testdaten
y_pred_las = model.predict(X_test)

# Plotting
create_regression_plots(y_test, y_pred_las)

# Berechnen und ausgeben der Metriken
mse_las, rmse_las, r2_las, mae_las, mape_las, maxerror_las = calculate_metrics(y_test,
count_errors_in_range(y_pred_las, 25000)
```



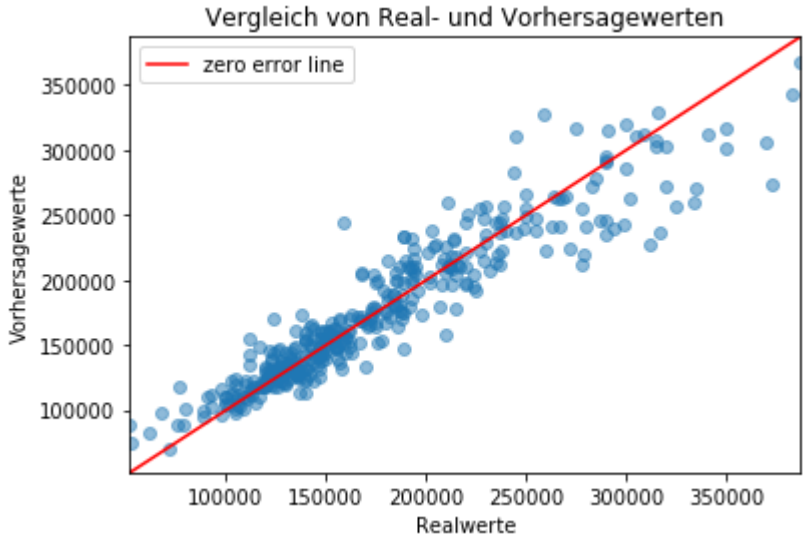


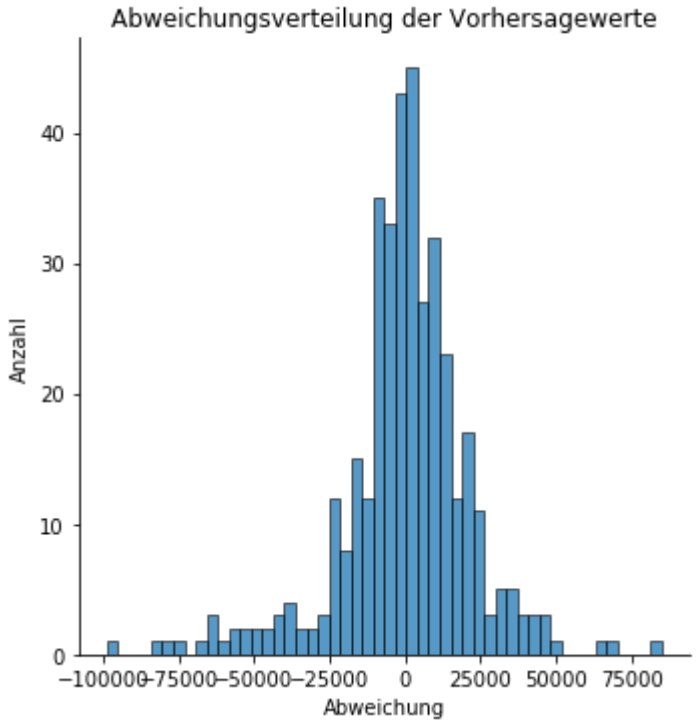
```
MSE: 384150692
RMSE: 19599
R2: 0.90365
MAE: 14444
MAPE:  8.435 %
MAX Error: 87053
Anzahl der falsch zugeordneten Preise: 54 von 382 Datensätzen (14 Prozent) b
ei einer maximalen Abweichung von 25000
```

Random Forrest

In [224]:

```
# Laden des Modells
from sklearn.ensemble import RandomForestRegressor

# Instanz erzeugen
model = RandomForestRegressor()

# Training
model.fit(X_train, y_train)

# Validierung mit Testdaten
y_pred_rf = model.predict(X_test)

# Plotting
create_regression_plots(y_test, y_pred_rf)

# Berechnen und ausgeben der Metriken
mse_rf, rmse_rf, r2_rf, mae_rf, mape_rf, maxerror_rf = calculate_metrics(y_test, y_pred
count_errors_in_range(y_pred_rf, 25000)
```





```
MSE: 470716446
RMSE: 21696
R2: 0.88194
MAE: 14805
MAPE:  8.479 %
MAX Error: 98755
Anzahl der falsch zugeordneten Preise: 54 von 382 Datensätzen (14 Prozent) b
ei einer maximalen Abweichung von 25000
```

Feature: Top houses to watch

Liegt ein Verkaufspreis deutlich unter der Vorhersage unserer Regression, so kann es sich dabei um ein gutes Geschäft handeln, da die Immobilie preislich anhand unserer Daten unter ihrem Wert verkauft wird. Diese Immobilien sind es jedenfalls wert, dass man sie als Investor genauer betrachtet. So kann man entscheiden, ob es sich bei der Immobilie wirklich um ein Schnäppchen handelt, oder ob der vermeindliche günstigere Preis nicht doch seine Gründe hat.

Um dies zu ermöglichen, werden im folgenden Feature die Immobilien herausgefiltert, welche die größte Differenz zwischen Vorhersage und Realpreis aufbringen. Dieses Feature soll den Investoren ermöglichen, den Preis einer Immobilie im Vergleich zu seinen Features und dessen Risiko beim Kauf einzuschätzen.

In [225]:

```
# Funktion, um ein sortiertes Dataframe mit den 20 größten Differenzen zwischen Real- u
def create_top_20(y_pred, y_test, name):
    diff = (y_pred - y_test).round()
    diff = pd.DataFrame(diff)
    diff = diff["Preis"].rename(name)
    diff_sorted = diff.sort_values()
    index = diff_sorted[-20:].index
    dftop = df.loc[index]
    dftop = dftop.join(diff)
    dftop[name] = dftop[name].astype(int)
    dftop.rename(columns={'Preis': 'Realpreis'}, inplace=True)
    dftop['Vorhersagepreis'] = dftop["Realpreis"] + dftop[name]
    colnames = dftop.columns.tolist()
    colnames = colnames[-3:] + colnames[:-3]
    dftop = dftop[colnames]
    dftop = dftop.sort_values(by=[name], ascending=False)
    return dftop, diff
```

Top 20 Lineare Regression

In [226]:

```
dftop_lin, diff_lin = create_top_20(y_pred_lin, y_test, "Differenz (Linear Regression)"
dftop_lin
```

Out[226]:

	Realpreis	Differenz (Linear Regression)	Vorhersagepreis	Grundstück in qm	Grundstücksform	Steigung	
1917	190000	58839	248839	946	IR1	Nein	E
1780	204000	55043	259043	914	IR1	Nein	T
720	260000	54811	314811	1331	IR1	Ja	Diamon
1395	259000	53269	312269	1312	IR1	Nein	Nc
1642	154000	52814	206814	227	Reg	Nein	Nc
996	175900	52491	228391	338	Reg	Nein	Upp
14	141500	43320	184820	892	Reg	Nein	New
650	285000	42191	327191	992	Reg	Nein	E
1785	192000	39439	231439	1143	IR3	Nein	Nc
227	194700	38839	233539	704	IR1	Nein	Nc
385	117000	36739	153739	1022	Reg	Nein	Gra
957	188500	36159	224659	887	Reg	Nein	No
1776	128000	34900	162900	892	Reg	Nein	S
456	52000	34826	86826	384	IR1	Nein	Di
1529	175000	33848	208848	1064	IR1	Nein	T
36	124000	33403	157403	831	Reg	Nein	New
1468	184500	32400	216900	476	IR1	Nein	Nc
1051	250000	31998	281998	1062	IR1	Nein	Nc
838	122500	30282	152782	178	Reg	Nein	Novicl
544	159000	28783	187783	892	Reg	Nein	Somers

20 rows × 30 columns

Top 20 Gradient Boosting

In [227]:

```
dftop_gbr, diff_gbr = create_top_20(y_pred_gbr, y_test, "Differenz (Gradient Boosting)"
dftop_gbr.head()
```

Out[227]:

	Realpreis	Differenz (Gradient Boosting)	Vorhersagepreis	Grundstück in qm	Grundstücksform	Steigung	Bezirk
1395	259000	82460	341460	1312	IR1	Nein	North Eas
784	239000	64512	303512	1108	IR1	Nein	ChinaTow
1917	190000	57763	247763	946	IR1	Nein	East En
28	226500	49896	276396	1178	IR1	Nein	North Wes
1384	243500	42277	285777	1003	Reg	Nein	North Wes

5 rows × 30 columns

Top 20 Ridge Prediction

In [228]:

```
dftop_rid, diff_rid = create_top_20(y_pred_rid, y_test, "Differenz (Ridge Prediction)")
dftop_rid.head()
```

Out[228]:

	Realpreis	Differenz (Ridge Prediction)	Vorhersagepreis	Grundstück in qm	Grundstücksform	Steigung	
1917	190000	58031	248031	946	IR1	Nein	Ea
1780	204000	54786	258786	914	IR1	Nein	Tr
1642	154000	53701	207701	227	Reg	Nein	Nor
720	260000	53045	313045	1331	IR1	Ja	Diamond
996	175900	52975	228875	338	Reg	Nein	Uppe

5 rows × 30 columns

Top 20 Lasso Prediction

In [229]:

```
dftop_las, diff_las = create_top_20(y_pred_las, y_test, "Differenz (Lasso Prediction)")
dftop_las.head()
```

Out[229]:

	Realpreis	Differenz (Lasso Prediction)	Vorhersagepreis	Grundstück in qm	Grundstücksform	Steigung	
1917	190000	58793	248793	946	IR1	Nein	Ea
1780	204000	55007	259007	914	IR1	Nein	Tr
720	260000	54387	314387	1331	IR1	Ja	Diamond
1395	259000	53192	312192	1312	IR1	Nein	Nor
1642	154000	52975	206975	227	Reg	Nein	Nor

5 rows × 30 columns

Top 20 Random Forest

In [230]:

```
dftop_rf, diff_rf = create_top_20(y_pred_rf, y_test, "Differenz (Random Forrest)")
dftop_rf.head()
```

Out[230]:

	Realpreis	Differenz (Random Forrest)	Vorhersagepreis	Grundstück in qm	Grundstücksform	Steigung	Bezirk
1875	159000	85016	244016	715	Reg	Nein	Burnley
1395	259000	67818	326818	1312	IR1	Nein	North East
1210	245000	65277	310277	1477	IR2	Nein	Miller
1053	210900	48952	259852	2000	IR2	Nein	ChinaTown
769	124000	46121	170121	1728	Reg	Nein	Grand Park

5 rows × 30 columns

Top Immobilien die in mindestens 4 von 5 der Top 20 der Modelle vorkommen

Um den Investoren die attraktivsten Immobilien mit dem geringsten Risiko aufzuzeigen, wurden nun die Top 20 Listen der einzelnen Vorhersagemodelle zusammengefügt, indem nur Immobilien die mindestens von 4 der 5 Modellen als Top 20 bewertet wurden ausgewählt werden.

Die zusammengefügte Top Liste sollte nun die Schwächen eines einzelenen Vorhersagemodells ausgleichen, da die Chance das bis zu 4 Vorhersagemodelle eine Immoblie falsch bewerten sehr gering ist. Natürlich kann es trotzdem Fehler geben, es sollte jedoch bei langfristigen Investitionen anhand dieser Top Liste zu Gewinnen kommen.

In [231]:

```
# Überprüft anhand des Indexes welche Immobilien in mehreren Top 20 Listen der einzelne
# und schreibt diese in einen neuen Datenframe
list = []
for index, row in dftop_lin.iterrows():
    list.append(index)
for index, row in dftop_gbr.iterrows():
    list.append(index)
for index, row in dftop_rid.iterrows():
    list.append(index)
for index, row in dftop_las.iterrows():
    list.append(index)
for index, row in dftop_rf.iterrows():
    list.append(index)

vorkommen = []
werteliste = []
zahlen = []
for wert in list:
    if not wert in zahlen:
        zahlen.append(wert)
        if list.count(wert) >= 4:
            werteliste.append(wert)
            vorkommen.append(list.count(wert))

dftop = df.loc[werteliste]
dftop['Vorkommen in Top 20'] = vorkommen
dftop = dftop.join(diff_lin)
dftop = dftop.join(diff_gbr)
dftop = dftop.join(diff_rid)
dftop = dftop.join(diff_las)
dftop = dftop.join(diff_rf)
dftop.rename(columns={'Preis': 'Realpreis'}, inplace=True)
dftop["Differenz Durchschnitt"] = ((dftop["Differenz (Linear Regression)"]+dftop["Diffe
                                    +dftop["Differenz (Ridge Prediction)"]+dftop["Diffe
                                    +dftop["Differenz (Random Forrest)"])/5).astype(int
dftop["Differenz (Linear Regression)"] = dftop["Differenz (Linear Regression)"].astype(
dftop["Differenz (Gradient Boosting)"] = dftop["Differenz (Gradient Boosting)"].astype(
dftop["Differenz (Ridge Prediction)"] = dftop["Differenz (Ridge Prediction)"].astype(in
dftop["Differenz (Lasso Prediction)"] = dftop["Differenz (Lasso Prediction)"].astype(in
dftop["Differenz (Random Forrest)"] = dftop["Differenz (Random Forrest)"].astype(int)

dftop = dftop[['Realpreis','Vorkommen in Top 20','Differenz Durchschnitt',
        'Differenz (Linear Regression)','Differenz (Gradient Boosting)',
        'Differenz (Ridge Prediction)','Differenz (Lasso Prediction)',
        'Differenz (Random Forrest)','Grundstück in qm', 'Grundstücksform', 'Steigung',
        'Bezirk', 'Zone', 'Lage', 'Typ', 'Zustand', 'Gebaut', 'Renoviert',
        'Zustand Fassade', 'Kellerflche in qm', 'Heizung', 'Heizungsqualitt',
        'Klimaanlage', 'Erster Stock in qm', 'Zweiter Stock in qm',
        'Wohnflche in qm', 'Schlafzimmer', 'Küchen', 'Küchenqualitt', 'Rume',
        'Garage Typ', 'Garagenkapazitt', 'Pool', 'Verkaufsmonat', 'Verkaufsjahr']]
dftop = dftop.sort_values(by=["Differenz Durchschnitt"], ascending=False)
dftop
```

Out[231]:

	Realpreis	Vorkommen in Top 20	Differenz Durchschnitt	Differenz (Linear Regression)	Differenz (Gradient Boosting)	Differenz (Ridge Prediction)	Differenz (Lasso Prediction)
1395	259000	5	61822	53269	82460	52375	53192
1917	190000	4	51580	58839	57763	58031	58793
996	175900	4	42913	52491	33326	52975	52406
957	188500	5	38863	36159	40599	36620	36218
14	141500	4	37529	43320	34973	43511	43335
456	52000	5	35758	34826	37874	34979	34853
650	285000	4	31904	42191	41077	41500	42162

7 rows × 35 columns

Es ist bei dieser Auflistung zu beachten, dass die Vorhersage auf den gegeben Daten beruht. Die hier aufgelisteten Immobilien wurden vom ML-Modell teurer eingeschätzt, als sie es eigentlich sind. Das bedeutet, dass die gegebenen Input Values wahrscheinlich einen höheren Wert als die des Durchschnitts der gegebenen Daten besitzen.

Diese Auflistung sollte nicht als absolut angesehen werden, jedoch sollte sie den Investoren eine Richtung geben, welche Immobilien es Wert sein könnten, angesehen zu werden. Die finale Kaufentscheidung der Investoren sollte natürlich immer noch aus einem persönlichen Eindruck heraus entstehen (es existieren schließlich noch weitere Daten zum Haus welche im Modell nicht gegeben waren. Jedoch können solche Daten, die wir nicht besitzen, nicht im Modell mit einbezogen werden.). Je mehr Daten das Modell bekommt, desto genauer werden dementsprechend die Vorhersagen. Bei einem "perfekten" Modell würde diese Liste keinen Sinn mehr ergeben, da der Preis der Immobilien nicht mehr interpretiert werden müsste.

Lohnt sich Renovierung?

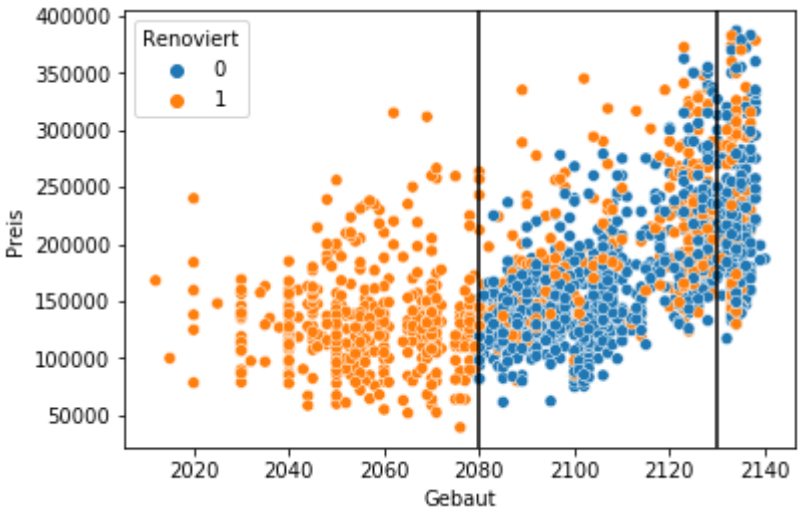
Eines unserer Ziele beschäftigt sich mit der Frage, ob Renovierung einer Immobilie Gewinn erzielt werden kann. Zunächst trennen wir die renovierten von den nicht-renovierten Immobilien.

In [232]:

```
# Nicht-renovierte Immobilien mit 0 kennzeichnen (Immobilie zählt als nicht renoviert,
df["Renoviert"] = df[["Renoviert"]].where(df["Renoviert"] == df["Gebaut"], 1)
df["Renoviert"] = df[["Renoviert"]].where(df["Renoviert"] != df["Gebaut"], 0)
```

In [233]:

```
# Plotting
sns.scatterplot(data=df, x = "Gebaut", y = "Preis", hue= "Renoviert")
plt.axvline(x=2080, color="black")
plt.axvline(x=2130, color="black")
plt.show()
```



Um zu ermitteln ob sich der Kauf von älteren Immobilien lohnt, um diese dann zu Renovieren und zu einem teureren Preis zu verkaufen, wurde ein Zeitintervall von 2080 bis 2130 gewählt. Die Untergrenze wurde gewählt, da Immobilien mit einem Baujahr vor 2080 bereits alle renoviert wurden und es hier somit nichts zu vergleichen gibt. Die Obergrenze wurde gewählt, weil Häuser, die später gebaut wurden, i.d.R. noch nicht renovierungsbedürftig sind und somit aus der Zielgruppe fallen.

In [234]:

```
# Zeitintervall erstellen und Durchschnitt berechnen
df4= df.where(df["Gebaut"] < 2130)
Preisdurchschnitt= df4["Preis"].where(df4["Gebaut"] > 2080).mean()
df2= df.where(df["Renoviert"] == 1)
df2= df2.where(df2["Gebaut"] < 2130)
Preisdurchschnitt_Renoviert= df2["Preis"].where(df2["Gebaut"] > 2080).mean()
df3 = df.where(df["Renoviert"] == 0)
df3= df3.where(df3["Gebaut"] < 2130)
Preisdurchschnitt_Nicht_Renoviert= df3["Preis"].where(df3["Gebaut"] > 2080).mean()

print("Zeitintervall 2080-2130")
print("Preisdurchschnitt aller Immobilien im Zeitraum: %d" % Preisdurchschnitt)
print("Preisdurchschnitt aller renovierten Immobilien im Zeitraum: %d" % Preisdurchschr
print("Preisdurchschnitt aller nicht renovierten Immobilien im Zeitraum: %d" % Preisdur
```

```
Zeitintervall 2080-2130
Preisdurchschnitt aller Immobilien im Zeitraum: 166639
Preisdurchschnitt aller renovierten Immobilien im Zeitraum: 185366
Preisdurchschnitt aller nicht renovierten Immobilien im Zeitraum: 158662
```

Anhand der errechneten durschnittlichen Preise lässt sich ausmachen, dass renovierte Immobilien durchschnittlich 25000 Dollar teurer sind als nichtrenovierte Immobilien.

Während diese Betrachtung relativ oberflächlich ist, gibt sie dennoch einen Hinweis darauf, dass sich Renovierungsarbeiten bei Immobilien lohnen können, so lange diese nicht allzu teuer ausfallen. Auch hier sollte diese Aussage natürlich nicht als pauschal korrekt gesehen werden und man sollte sich als Investor vor seiner Investition noch einmal gründlich Gedanken machen.

Aufgabe 4: Evaluation (3P)

Evaluieren Sie ihr finales Modell. Quantifizieren Sie bei der Evaluation die Konfidenz ab. Achten Sie auf die Sinnhaftigkeit der Bewertung im Bezug zu den Geschäftszielen aus Aufgabe 1. Stellen Sie 3 bis 5 wichtige Erkenntnisse in einem Summary heraus. Beschreiben Sie diese in ganzen Sätzen auf Deutsch.

Vergleich der Modelle

Um die Modelle miteinander Vergleichen zu können, werden die Werte der Metriken MSE, RMSE, R2, MAE, MAPE und MAX miteinander verglichen.

In [235]:

```
# Dataframe mit den gewünschten Metriken erstellen und diese einfügen
a = [["Linear Regression", round(mse_lin), round(rmse_lin), round(r2_lin,5), round(mae_
["Gradient Boosting", round(mse_gbr), round(rmse_gbr), round(r2_gbr,5), round(mae_gbr),
["Ridge Prediction", round(mse_rid), round(rmse_rid), round(r2_rid,5), round(mae_rid),
["Lasso Prediction", round(mse_las), round(rmse_las), round(r2_las,5), round(mae_las),
["Random Forrest", round(mse_rf), round(rmse_rf), round(r2_rf,5), round(mae_rf), round(
comparison = pd.DataFrame(a, columns=["Model Type", "MSE", "RMSE", "R2", "MAE", "MAPE",
comparison
```

Out[235]:

	Model Type	MSE	RMSE	R2	MAE	MAPE	MAX
0	Linear Regression	384275304.0	19603.0	0.90362	14451.0	8.442	87016.0
1	Gradient Boosting	327474484.0	18096.0	0.91786	13121.0	7.490	82460.0
2	Ridge Prediction	384112229.0	19599.0	0.90366	14435.0	8.428	87006.0
3	Lasso Prediction	384150692.0	19600.0	0.90365	14445.0	8.435	87053.0
4	Random Forrest	470716447.0	21696.0	0.88194	14806.0	8.479	98756.0

In den meisten Durchläufen hat sich das Gradient Boosting Verfahren als das Verfahren mit den besten Ergebnissen herauskristallisiert. Die Werte sind meistens nur marginal besser als die der anderen Verfahren, jedoch schneidet das Gradient Boosting bei fast jeder Aufteilung der Daten in Trainings- und Testdaten mit den besten Werten ab.

Die Lineare Regression sowie die Ridge und die Lasso Prediction haben in jeder Metrik sehr ähnlich abgeschnitten und teilen sich in unseren Augen den zweiten Platz.

In [236]:

```
r2_durchschnitt = (r2_rf+r2_lin+r2_las+r2_gbr+r2_rid)/5
print("Durchschnittlicher R2-Score:", round(r2_durchschnitt, 5))
```

```
Durchschnittlicher R2-Score: 0.90214
```

Beurteilung der Ziele

Unsere Modelle bieten den Investoren ein Tool an um Preise anhand der gegebenen Metriken vorherzusagen. Hierbei erklären unsere Modelle durchschnittlich ~90% der Variation des Verkaufspreises anhand des Testdatensatzes (r2 score). Diese Vorhersagen können dann zur Auswahl von möglichen Immobilien genutzt werden, in die eine Investition attraktiv, also mit einer hohen Chance auf Gewinn verbunden wäre. Die Modelle lernen anhand der gegebenen Daten, den Preis einer Immobilie vorherzusagen. Um die Vorhersagewerte nicht zu verfälschen, haben wir zusätzlich Ausreißerwerte aus den Daten mit Hilfe des Z-Scores entfernt. Immobilien, die von den Modellen als günstig beurteilt werden, stellen jedoch nicht in jedem Fall attraktive Immobilien für eine Investition dar. Metriken, die nicht in den Daten vorhanden sind, können den Verkaufspreis auch beeinflussen, werden jedoch von den Modellen nicht berücksichtigt. Die Immobilien sollten deshalb trotzdem noch näher betrachtet werden und mit Hilfe der gewonnen Erkenntnisse in der Data Exploration und Analyse vor der Investition geprüft werden. Diese Erkentnisse reichen von sehr wichtigen Fakten wie, dass die Wohnfläche den Preis stark beeinflusst bis zu Erkenntnissen, die den Preis nicht weiter stark beeinflussen, wie

zum Beispiel, dass die meisten Häuser im Sommer verkauft werden. Die Erkenntnisse, die keinen großen Einfluss auf den Preis haben, spielen zwar eine weniger große Rolle für die Preisvorhersage, sind aber dennoch aufschlussreich für Investoren.

Die Modelle haben natürlich auch eine geringe Abweichung vom Realpreis, da kein Modell den Preis perfekt vorhersagen kann, mit Hilfe der durchschnittliche Abweichung kann diese Abweichung aber eingegrenzt werden und ein Risiko vorallem über eine Vielzahl an Investitionen vermindert werden. Bei unseren Modellen orientiert sich die Prozentualepreisabweichung durchschnittlich bei ca 8%, mit dieser Information können Investoren dann diese als Puffer mit einberechnen, so auf dauer Profit erzielen und das Risiko durch aufkommende Abweichungen eindämmen. Die Immobilien, die wir als Investition vorschlagen, stützen sich auf der Tatsache, dass diese von der Mehrheit der Modelle als attraktiv eingestuft werden. Dadurch verringert sich auch das Risiko, da sich die Modelle gegenseitig ausgleichen und Fehler bei der Preisvorhersage gefiltert werden.

Als zusätzliches Ziel hatten wir es uns vorgenommen, die Rentabilität von Renovationsarbeiten zu beurteilen. Es ergab sich, dass Renovierungsarbeiten sich anhand der vorgegebenen Daten lohnen können. Bei der Renovierung sollten die Renovierungskosten berücksichtigt werden, um eine mögliche Gewinnspanne zu berechnen.

Zusammenfassend können wir den Preis einer Immobilie relativ genau anhand der gegebenen Metriken bestimmen und zusäzlich Aussagen über das Risiko bei einer Investition treffen.

Summary

- Das Gradient Boosting-Verfahren hat sich in den meisten Fällen als das (wenn auch nur in geringem Maße) performanteste Verfahren für diese Regressionsaufgabe herausgestellt. Die anderen verwendeten Regressionsverfahren haben allerdings ebenfalls gut abgeschnitten und lagen nicht weit hinter dem Gradient Boosting.
- Je mehr Daten ein Modell bekommen hat, desto besser hat es abgeschnitten. Selbst niedrig korrelierende Spalten haben das Modell genauer gemacht und wurden deshalb nicht entfernt. Das Modell wurde allerdings präziser, nachdem Ausreißer mit Hilfe des Z-Scores entfernt wurden.
- Die Ziele vor dem Beginn der Untersuchung zu definieren half uns dabei, die Aufgaben zielorientiert zu bearbeiten und sinnvolle Features zu entwickeln.

Aufgabe 5: Deployment (3+3P)

Erstellen Sie eine Anleitung / Handreichung / Vorgehen für InvestorInnen basierend auf den Erkenntnissen von Aufgabe 1 bis 4. Sie können dies per Hand, mit Software, als Formel, als Folie, oder anders umsetzen. Wählen Sie eine geeignete Präsentationsform für Ihre Zielgruppe. Stellen Sie Ihre gesamten in Punkt 1 – 4 erarbeiteten Ergebnisse in 5 bis 7 Folien Ihren AuftraggeberInnen als Videopräsentation vor.

Siehe Videos

Technische Anforderung (2P)

Stellen Sie die Ablauffähigkeit sicher. Ändern Sie nichts an der csv-Datei, die vorgegeben ist. Ermöglichen Sie es, eine zusätzliche csv-Datei in dem gegebenen Format einzulesen, mit der dann das finale Modell durchlaufen und für die Hauspreisvorhersage folgende fünf Werte berechnet werden: R2, MSE, RMSE, MAPE,

durchläuft und für die Hausbesitzverhältnisse folgende Kennwerte berechnet werden: AVG, MED, ANZAHL, MIN & MAX. Sie können dies testen, indem Sie von der vorgegebenen csv-Datei z. B. die ersten 3 oder die letzten 3 Zeilen einlesen und prüfen, ob das Programm sich erwartungskonform verhält. Dokumentieren Sie die Code-Stelle und Verwendung.

Siehe oben