DAV
DEUTSCHE
AKTUARVEREINIGUNG e.V.

Praktische Prüfung im Vertiefungswissen

Actuarial Data Science Immersion

Zeitraum: 14.04.2024 – 14.05.2024

Hinweise:

- Die Gesamtpunktzahl beträgt 180 Punkte. Die Prüfung ist bestanden, wenn mindestens 90 Punkte erreicht werden.
- Bitte prüfen Sie die Ihnen vorliegenden Prüfungsunterlagen auf Vollständigkeit. Die Prüfung besteht aus den vier Teilen A, B, C und D und beinhaltet insgesamt 18 Seiten. Zusätzliche Materialien (Datensätze, Datenbeschreibung) sollten Sie mit der Prüfung erhalten haben. Auf diese wird in den jeweiligen Aufgabenteilen eingegangen.
- Die Aufgaben sind in der vorgegebenen Reihenfolge in einem Notebook (Jupyter oder Rmd) zu bearbeiten. Die Lösung muss direkt nach der Aufgabenstellung in einer strukturierten und übersichtlichen Weise dargestellt werden. Nichtbeachtung dieser Vorgabe führt zu Punkteabzug. Aufgabenteile, bei denen der als Ergebnis des Codes zu erwartende Output fehlt, werden mit 0 Punkten bewertet.
- Die erforderliche Programmierung kann in R oder Python erfolgen. Es wird eine Umsetzung mit Python in einer Cloud-Umgebung, bei der zwischen CPU- und GPU-Ausführung gewechselt werden kann, empfohlen.
- Es sind zwei Varianten eines Notebooks zu erstellen und auszuführen:
  - Als erstes sollen die Aufgaben der Teile A, B, C und Aufgabe D-1 in einem Notebook bearbeitet und in einer CPU-Umgebung ausgeführt werden. Sowohl dieses Notebook als auch eine HTML-Version ist einzureichen.
  - Wenn diese Arbeiten abgeschlossen sind, soll die gesamte Prüfungungsaufgabe gemäß Aufgabenbeschreibung (siehe Teil D) in einem zweiten Notebook umgesetzt und in einer GPU-Umgebung ausgeführt werden. Dieses Notebook sowie die ensprechende HTML-Version soll mit dem Namenszusatz "GPU" versehen und ebenfalls eingereicht werden. Wenn kein GPU-Notebook

eingereicht wird, wird der gesamte Teil D mit 0 Punkten bewertet. Wird ein GPU-Notebook eingereicht, ist das entsprechende HTML-Dokument maßgeblich für die Punktevergabe in allen Aufgabenteilen.

- In den Aufgabenteilen A und B werden (u.a.) mehrere Hauspreisprognosen erstellt, wobei künstliche neuronale Netze generell mit TensorFlow/Keras umzusetzen sind. Die Modellgüte soll anhand der Wurzel der mittleren Fehlerquadratsumme (RMSE) ermittelt und ausgegeben werden. Sofern nicht anders angegeben soll hierzu die Validierungsstichprobe herangezogen werden. Für in der Aufgabenstellung näher bezeichnete Modelle und Umgebungen (CPU/GPU) sollen Laufzeiten der Modellanpassung an die Trainingsdaten ("fitten") sowie die Prognosegüten gespeichert und verglichen werden.
- Mit der Einreichung von Ergebnis-HTML-Dokumenten und -Notebooks erklärt die zu prüfende Person, dass der eingereichte Ergebnisbericht inklusive Code- und Beschreibungsanteilen eigenständig erzeugt worden ist. Verstöße gegen diese Grundlagen können vom Prüfungsausschuss sanktioniert werden und zum Ausschluss von der Prüfung führen.

1. GPU-Notebook: Inhaltsverzeichnis

Teil A: Hauspreise analysieren und einfach

Aufgabe A-0: Bibliotheken, Funktionen, Datenstrukturen [Lernziele 2.2 & 3.3/3.4; 2 Punkte]

Alle für die weiteren Aufgaben benötigten Bibliotheken und Funktionen sind zentral an dieser Stelle einzubinden bzw. zu definieren. Ebenso sollen alle globalen Variablen in diesem Teil angegeben werden. Des Weiteren sollen Datenstrukturen für den Vergleich der Laufzeiten der Teile A, B und C sowie Datenstrukturen für den Vergleich der Laufzeiten und Modellgüten RMSE bezüglich der Validierungsstichprobe für die Modelle der Aufgaben A-7 und B-1, B-2 b) bis B-4 erstellt werden.

Lösungsvorschlag

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import matplotlib.patches as mpatches
import seaborn as sns
from matplotlib import pyplot
import plotly.express as px
import pylab as p
import math

from pathlib import Path
import urllib.request

import time
import warnings
warnings.filterwarnings('ignore')

import sklearn
from sklearn.metrics import mean_squared_error, r2_score, get_scorer
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import StratifiedKFold, cross_validate
from sklearn.cluster import KMeans

from yellowbrick.model_selection import LearningCurve

import locale
locale.setlocale(locale.LC_ALL, "")

from sklearn.linear_model import Lasso, LassoCV, Ridge, RidgeCV
import statsmodels.api as sm
import statsmodels.formula.api as smf
from catboost import Pool, CatBoostRegressor, cv, CatBoostClassifier
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:
```
# Prevent TensorFlow From Fully Allocating GPU Memory
import tensorflow as tf
# Ref: https://www.tensorflow.org/guide/gpu#limiting_gpu_memory_growth
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
    # Currently, memory growth needs to be the same across GPU
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        logical_gpus = tf.config.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        # Memory growth must be set before GPUs have been initialized
        print(e)
else:
    print('Das ist das CPU-Notebook')
```

```
1 Physical GPUs, 1 Logical GPUs
```

In [3]:
```
!pip install tensorflow-docs
```

```
Collecting tensorflow-docs
  Obtaining dependency information for tensorflow-docs from https://files.pythonh
osted.org/packages/e8/72/b7399b0f818559d252a109f43ea436697b31aceaacfda05cb4a7098f
df4b/tensorflow_docs-2024.2.5.73858-py3-none-any.whl.metadata
  Downloading tensorflow_docs-2024.2.5.73858-py3-none-any.whl.metadata (955 byte
s)
Collecting astor (from tensorflow-docs)
  Obtaining dependency information for astor from https://files.pythonhosted.org/
packages/c3/88/97eef84f48fa04fbd6750e62dcceafba6c63c81b7ac1420856c8dcc0a3f9/astor
-0.8.1-py2.py3-none-any.whl.metadata
  Downloading astor-0.8.1-py2.py3-none-any.whl.metadata (4.2 kB)
Requirement already satisfied: absl-py in /opt/conda/lib/python3.10/site-packages
(from tensorflow-docs) (1.4.0)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.10/site-packages
(from tensorflow-docs) (3.1.2)
Requirement already satisfied: nbformat in /opt/conda/lib/python3.10/site-package
s (from tensorflow-docs) (5.9.2)
Requirement already satisfied: protobuf>=3.12 in /opt/conda/lib/python3.10/site-p
ackages (from tensorflow-docs) (3.20.3)
Requirement already satisfied: pyyaml in /opt/conda/lib/python3.10/site-packages
(from tensorflow-docs) (6.0.1)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.10/site-
packages (from jinja2->tensorflow-docs) (2.1.3)
Requirement already satisfied: fastjsonschema in /opt/conda/lib/python3.10/site-p
ackages (from nbformat->tensorflow-docs) (2.18.0)
Requirement already satisfied: jsonschema>=2.6 in /opt/conda/lib/python3.10/site-
packages (from nbformat->tensorflow-docs) (4.19.0)
Requirement already satisfied: jupyter-core in /opt/conda/lib/python3.10/site-pac
kages (from nbformat->tensorflow-docs) (5.3.1)
Requirement already satisfied: traitlets>=5.1 in /opt/conda/lib/python3.10/site-p
ackages (from nbformat->tensorflow-docs) (5.9.0)
Requirement already satisfied: attrs>=22.2.0 in /opt/conda/lib/python3.10/site-pa
ckages (from jsonschema>=2.6->nbformat->tensorflow-docs) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /opt/cond
a/lib/python3.10/site-packages (from jsonschema>=2.6->nbformat->tensorflow-docs)
(2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in /opt/conda/lib/python3.10/s
ite-packages (from jsonschema>=2.6->nbformat->tensorflow-docs) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in /opt/conda/lib/python3.10/site-p
ackages (from jsonschema>=2.6->nbformat->tensorflow-docs) (0.9.2)
Requirement already satisfied: platformdirs>=2.5 in /opt/conda/lib/python3.10/sit
e-packages (from jupyter-core->nbformat->tensorflow-docs) (4.1.0)
Downloading tensorflow_docs-2024.2.5.73858-py3-none-any.whl (182 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 182.5/182.5 kB 4.9 MB/s eta 0:00:00
Downloading astor-0.8.1-py2.py3-none-any.whl (27 kB)
Installing collected packages: astor, tensorflow-docs
Successfully installed astor-0.8.1 tensorflow-docs-2024.2.5.73858
```

In [4]:

```python
# Tensorflow
from tensorflow.keras import Sequential
from tensorflow.keras import layers
from numpy import sqrt
from tensorflow.keras.utils import plot_model
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Embedding, Flatten, concatenate, Dense, Dropo
from tensorflow.keras.optimizers import Adam
import tensorflow_docs as tfdocs
import tensorflow_docs.modeling
import tensorflow_docs.plots
from tensorflow.keras.models import load_model

# für Teil C
from shapely.geometry import Polygon, MultiPolygon, shape, Point
import geopandas as gpd
```

```
from matplotlib import colors
import contextily as ctx
from geopandas import GeoDataFrame
from sklearn.metrics import silhouette_score
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
```

In [5]:
```
# Datenstruktur für die Laufzeit (train.) und Güte (valid.) ausgewählter Modelle
results_df = pd.DataFrame(columns=['Model','time_train','RMSE_val','RMSE_test'])
results_df.set_index('Model',inplace=True)

# Datenstruktur für die Zeitmessung der Teile A, B und C anlegen und Zeitmessung
runtime_abc_df = pd.DataFrame(columns=['Teil','Laufzeit'])
runtime_abc_df.set_index('Teil',inplace=True)

# Zeit Notebook, Teil A
startzeit_teil_a = time.time()
```

In [6]:
```
# Verwendete globale Variablen
seed = 42
tf.keras.utils.set_random_seed(seed)
```

In [7]:
```
# Funktion zum Plot der RMSE-Werte
def plot_rmse(df):
    df.sort_values('RMSE_val',inplace=True)
    fig = plt.figure()
    df.RMSE_val.plot(kind='barh', color='green', title='Hauspreis Prognosefehler
    plt.show()
```

In [8]:
```
# Funktion zum Plot der RMSE-Val und RMSE-Test-Werte
def plot_rmse_2(df):
    df.sort_values('RMSE_test',inplace=True)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    width = 0.3
    df.RMSE_val.plot(kind='barh', color='green', ax=ax, width=width, position=1,
    df.RMSE_test.plot(kind='barh', color='blue', ax=ax, width=width, position=0,
    ax.set_ylabel('RMSE_val (sample B)', color='green')
    ax.set_xlabel('RMSE_test (sample C)', color='blue')
    plt.show()
```

In [9]:
```
# Hilfsfunktion zur Umwandlung von latitude-longitude-Paaren in GeoPandas-Points
def create_points(latitudes, longitudes):
    return gpd.points_from_xy(longitudes, latitudes)
```

Aufgabe A-1: Datensatz einlesen und prüfen [Lernziele 3.3/3.4; 5 Punkte]

Der Ihnen zur Verfügung gestellte Datensatz "california_housing_county_sample-de.csv" enthält demografische Informationen zum US-Bundesstaat Kalifornien. Insgesamt enthält der Datensatz die folgenden 12 Merkmale:

| Nummer | Feature | Beschreibung | | --- | --- | --- | | 1 | latitude | Breitengrad | | 2 | longitude | Längengrade | | 3 | housing_median_age | Median Alter der Häuser in einem Bezirk | | 4 | total_rooms | Anzahl der Räume in einem Bezirk | | 5 | total_bedrooms | Davon: Anzahl der Schlafzimmer in einem Bezirk | | 6 | population | Bevölkerungsanzahl in einem Bezirk | | 7 | households | Anzahl der

Haushalte in einem Bezirk | | 8 | median_income | Median Einkommen in einem Bezirk | | 9 | median_house_value | Median Wert der Häuser in einem Bezirk| | 10 | ocean_proximity | Beschreibt die Nähe zum Ozean | | 11 | county_name | Name der Region | | 12 | sample | Für jede Zeile des Datensatzes entweder mit A, B oder C belegt | Dabei ist zu beachten, dass es sich bei diesen Angaben 3-9 um aggregierte Informationen auf Bezirksebene (Wahlbezirke 1992) handelt. Die nachfolgenden Aufgaben sind der Reihe nach durchzuführen: a) Der zur Verfügung gestellte Datensatz ist einzulesen. Im Anschluss sind die Anzahl an Zeilen und Spalten sowie sieben zufällige Zeilen auszugeben. b) Die Datentypen des eingelesenen Datensatzes sind auszugeben, zu prüfen und zu kommentieren. Nominale Merkmale sollen den Datentyp für Kategorien (Python: category, R: factor) erhalten. Gegebenenfalls erforderliche Anpassungen sind durchzuführen und die finalen Datentypen sind auszugeben. c) Das Merkmal "sample", das fest vorgegebene Stichproben für das spätere Trainieren (A), Validieren (B) und Testen (C) der Modelle definiert, ist auszuzählen und die Stichprobengrößen sind zu kommentieren.

Lösungsvorschlag

Zu a):

In [10]:
```
# Datensatz "California House Prices" einlesen
df_raw = pd.read_csv('../input/california/california_housing_county_sample-de.csv
df = df_raw.copy(deep=True)

# Größe ermitteln
print("Anzahl Zeilen und Spalten: ", df.shape)

# 7 zufällige Zeilen ausgeben
df.sample(7, random_state=seed)
```

```
Anzahl Zeilen und Spalten:  (20640, 12)
```

Out[10]:

	latitude	longitude	housing_median_age	total_rooms	total_bedrooms	population	hous
20046	39.51	-121.56	46	1885	385.0	871	
3024	33.77	-117.23	5	2108	496.0	1666	
15663	37.74	-122.50	45	1771	349.0	1098	
20484	40.78	-124.16	43	2241	446.0	932	
9814	34.20	-118.87	26	1924	245.0	775	
13311	37.25	-121.92	34	2231	360.0	1035	
7113	34.05	-118.43	52	1693	290.0	727	

Zu b):

In [11]:
```
# Datentypen ausgeben
df.dtypes
```

Out[11]:
```
latitude              float64
longitude             float64
housing_median_age      int64
total_rooms             int64
total_bedrooms        float64
population              int64
households              int64
median_income         float64
median_house_value      int64
ocean_proximity        object
county_name            object
sample                 object
dtype: object
```

Jeweils fünf metrische Merkmale sind als Ganzzahl bzw. als Gleitkommazahl kodiert. Es ist kein offensichtlich kategorielles Merkmal numerisch kodiert. Die nominalen Merkmale haben in Python den Datentyp Object erhalten und werden im Folgenden auf 'category' gesetzt:

In [12]:
```
# Datentypen der nominalen Merkmale als festlegen (Python: category, R: factor)
cat = ['ocean_proximity', 'county_name', 'sample']
df[cat]=df[cat].astype('category')
```

Abschließende Kontrolle der Datentypen:

In [13]:
```
df.dtypes
```

Out[13]:
```
latitude              float64
longitude             float64
housing_median_age      int64
total_rooms             int64
total_bedrooms        float64
population              int64
households              int64
median_income         float64
median_house_value      int64
ocean_proximity      category
county_name          category
sample               category
dtype: object
```

In [14]:
```
num_columns = list(df.select_dtypes(include=[np.number]).columns )
cat_columns = list(df.select_dtypes(exclude=[np.number]).columns )

print("Numerische Merkmale: ", num_columns)
print("Kategorielle Merkmale: ", cat_columns)
```

```
Numerische Merkmale:  ['latitude', 'longitude', 'housing_median_age', 'total_room
s', 'total_bedrooms', 'population', 'households', 'median_income', 'median_house_
value']
Kategorielle Merkmale:  ['ocean_proximity', 'county_name', 'sample']
```

Zu c):

In [15]:
```
# Verteilung der Stichproben
df.value_counts("sample", normalize=True)
```

Out[15]:
```
sample
A    0.698692
B    0.150678
C    0.150630
Name: proportion, dtype: float64
```

Bei der späteren Modellierung sollen offenbar rund 70% der Daten für das Modelltraining, 15% für die Modellvalidierung und weitere 15% für das unabhängige Testen der Modelle verwendet werden.

Aufgabe A-2: Numerische Merkmale analysieren, aufbereiten und visualisieren [Lernziele 3.3/3.4 & 5.2; 19 Punkte]

a) Für die numerische Merkmale sind mindestens die Kennzahlen count, mean, std, min/max sowie die das 25, 50 und 75% Quantil auszugeben. Zudem soll ermittelt werden, bei welchen Merkmalen fehlende Werte (und falls ja, in welcher absoluten und prozentualen Größenordung) vorliegen. b) Anhand des Datensatzes ist zu begründen und geeignet grafisch zu visualisieren, warum eine Ersetzung bei _total_bedrooms_ durch den Median nicht empfehlenswert ist (mindestens zwei konkrete Zeilen des Datensatzes sind als Beispiele auszugeben) und was für den Bezug auf _total_rooms_ spricht. Im Anschluss sind für das Feature _total_bedrooms_ die fehlenden Werte durch den mittleren Quotienten zwischen _total_bedrooms_ und _total_rooms_ (berechnet auf Basis der Datensätze ohne fehlende Werte) zu ersetzen. Die Ersetzung ist geeignet zu verifizieren. c) Zu den bereits vorhandenen Merkmalen sollen noch die folgenden Merkmale zum Datensatz hinzugefügt werden:

→ bedrooms_per_house = total_bedrooms / households

→ rooms_per_house = total_rooms / households

Die Berechnung ist an zwei ausgewählten Datensätzen zu demonstrieren. d) Die numerischen Merkmale sind zusammen mit einem Kerndichteschätzer zu visualisieren und die Ausgaben zu kommentieren. e) Die Korrelationen der numerischen Merkmale sind zu berechnen und zu visualisieren. Im Anschluss ist auf Basis dieser Ergebnisse zu begründen, warum die Merkmale _households_, _total_rooms_, _total_bedrooms_, und _bedrooms_per_house_ aus dem Datensatz entfernt werden können. Das ist im Anschluss durchzuführen. Im Anschluss sind die Korrelationen der verbleibenden Merkmale erneut zu plotten.

Lösungsvorschlag

Zu a):

In [16]:
```
df[num_columns].describe()
```

Out[16]:

	latitude	longitude	housing_median_age	total_rooms	total_bedrooms	popula
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.00
mean	35.631861	-119.569704	28.639486	2635.763081	537.870553	1425.47
std	2.135952	2.003532	12.585558	2181.615252	421.385070	1132.46
min	32.540000	-124.350000	1.000000	2.000000	1.000000	3.00
25%	33.930000	-121.800000	18.000000	1447.750000	296.000000	787.00
50%	34.260000	-118.490000	29.000000	2127.000000	435.000000	1166.00
75%	37.710000	-118.010000	37.000000	3148.000000	647.000000	1725.00
max	41.950000	-114.310000	52.000000	39320.000000	6445.000000	35682.00

In [17]:
```
# Angabe absolut
df[num_columns].isna().sum()
```

Out[17]:
```
latitude              0
longitude             0
housing_median_age    0
total_rooms           0
total_bedrooms      207
population            0
households            0
median_income         0
median_house_value    0
dtype: int64
```

In [18]:
```
# Angabe prozentual
round(df.isna().sum()*100/len(df),2)
```

Out[18]:
```
latitude              0.0
longitude             0.0
housing_median_age    0.0
total_rooms           0.0
total_bedrooms        1.0
population            0.0
households            0.0
median_income         0.0
median_house_value    0.0
ocean_proximity       0.0
county_name           0.0
sample                0.0
dtype: float64
```

Zu b):

A) Bei einer Ersetzung der fehlenden Werte von _total_bedrooms_ durch den Median von 435 (s.o.) gäbe es folgende unpausiblen Fälle von Bezirken mit mehr Schlafzimmern als Zimmern insgesamt:

In [19]:
```
df[df['total_bedrooms'].isna()].query('total_rooms < 435')
```

Out[19]:

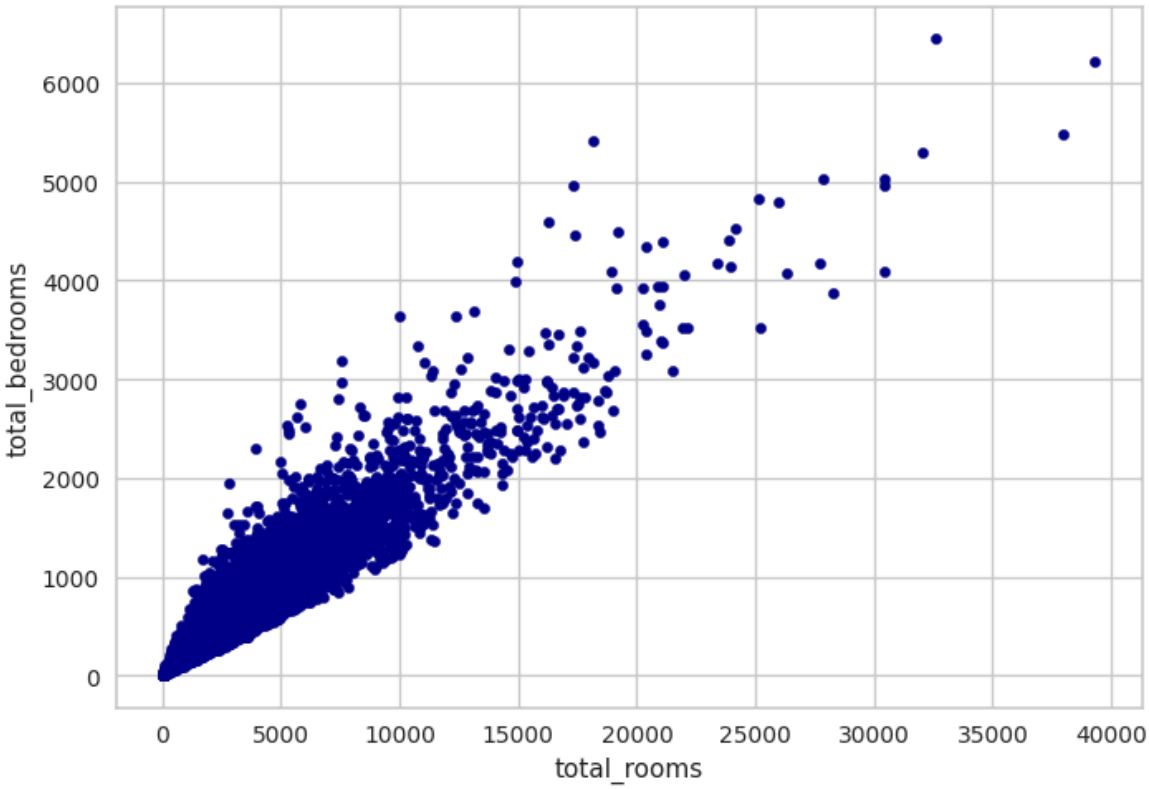
	latitude	longitude	housing_median_age	total_rooms	total_bedrooms	population	hous
8242	34.09	-117.28	44	376	NaN	273	
11648	35.42	-119.11	52	154	NaN	37	
12310	36.61	-121.85	38	238	NaN	191	
16259	37.78	-120.85	25	421	NaN	303	
18004	38.17	-122.23	45	350	NaN	225	

Die Anzahl der Datensätze, die durch diese Art der Ersetzung nicht plausibel wären, ist zwar überschaubar, zeigt aber die Spitze des Problems.

B) Bei einer Ersetzung mit dem Mittelwert würde man folgenden starken Zusammenhang unterschlagen:

In [20]:
```
print("Korrelationskoeffizient (Pearson): ", df['total_bedrooms'].corr(df['total_
df.plot.scatter(x='total_rooms', y='total_bedrooms', c='DarkBlue')
plt.show()
```

```
Korrelationskoeffizient (Pearson):  0.9303795046865075
```



In [21]:
```
# Gewichtung berücksichtigen
df["bedrooms_ratio"] = df["total_bedrooms"] / df["total_rooms"]
median_bedrooms_ratio = df["bedrooms_ratio"].median()
df["bedrooms_ratio"].fillna(median_bedrooms_ratio, inplace=True)
df["total_bedrooms"].fillna(median_bedrooms_ratio*df["total_rooms"], inplace=True
```

In [22]:
```
# An zwei Datensätzen das Ergebnis der Ersetzung der fehlenden Werte von "total_b
df.iloc[[8242, 11648]]
```

Out[22]:

	latitude	longitude	housing_median_age	total_rooms	total_bedrooms	population	hous
8242	34.09	-117.28	44	376	76.389075	273	
11648	35.42	-119.11	52	154	31.287015	37	

Zu c):

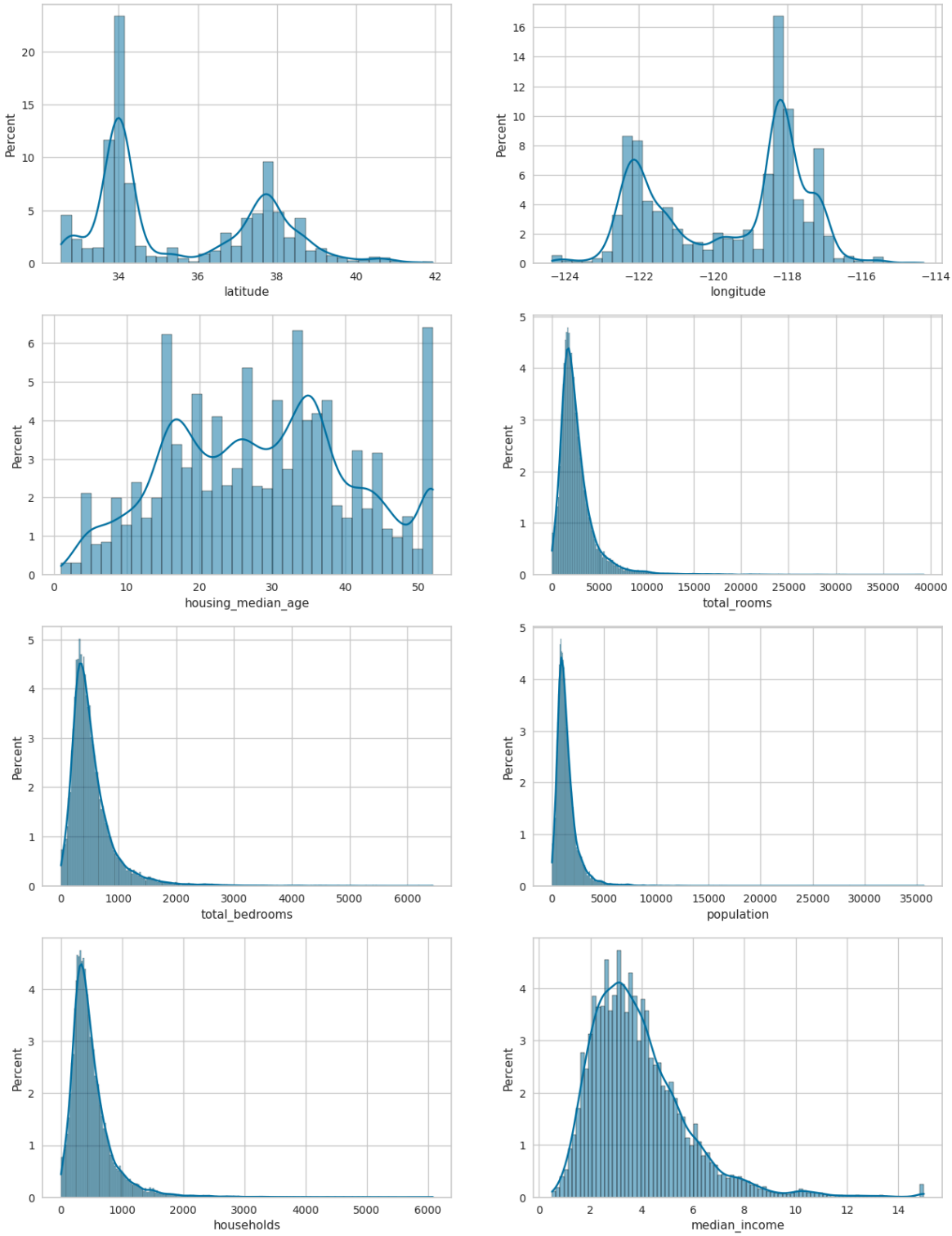
In [23]:
```
df["bedrooms_per_house"] = df["total_bedrooms"] / df["households"]
df["rooms_per_house"] = df["total_rooms"] / df["households"]

df.sample(2)
```

Out[23]:

	latitude	longitude	housing_median_age	total_rooms	total_bedrooms	population	hous
20046	39.51	-121.56	46	1885	385.0	871	
3024	33.77	-117.23	5	2108	496.0	1666	

Zu d):

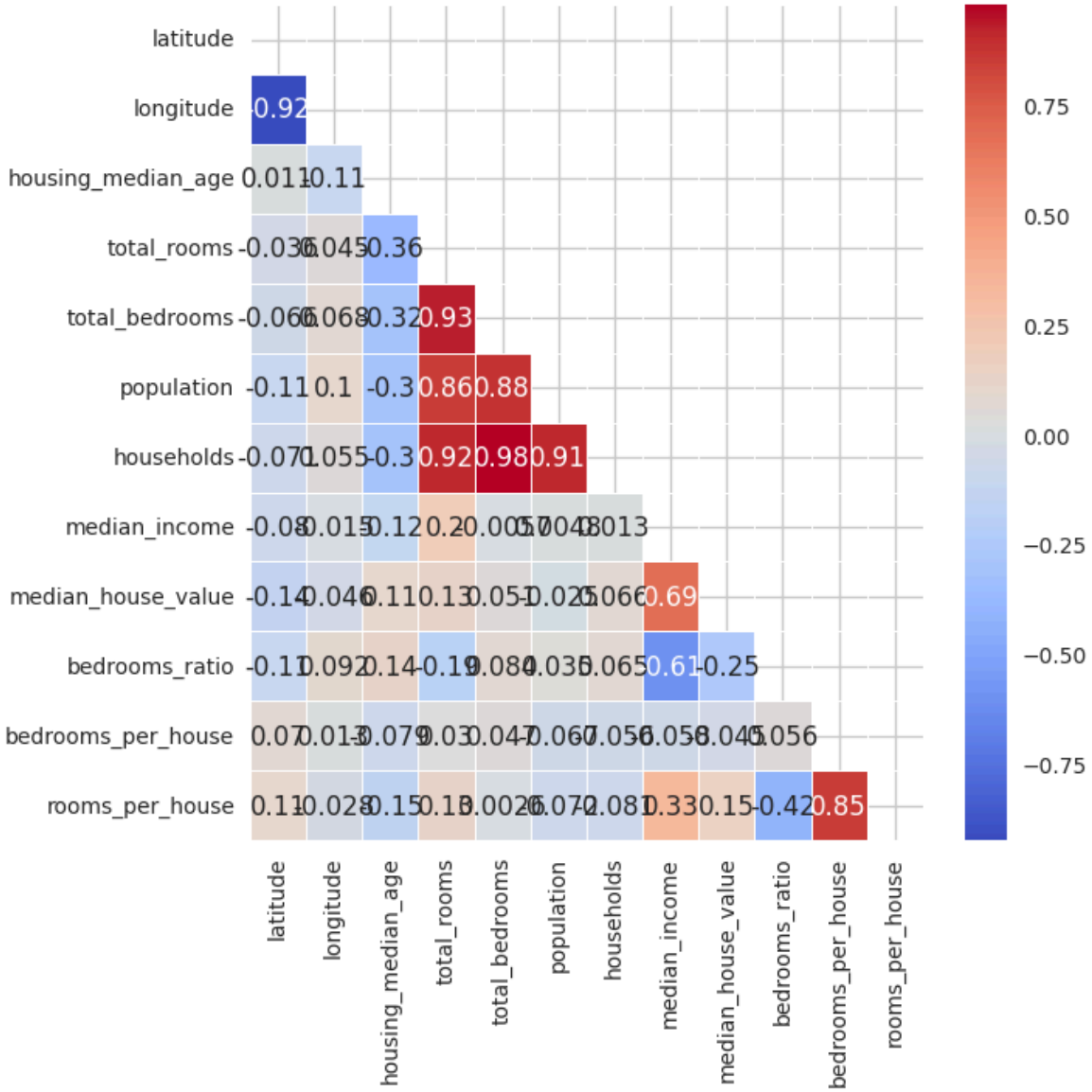
In [24]:
```
# Werteverteilung der numerischen Merkmale visualisieren
f, axes = plt.subplots(4, 2, figsize=(15, 20), sharex=False)
for ax, feature in zip(axes.flat, df.select_dtypes(include='number').columns):
    sns.histplot(data=df, x=feature , stat="percent", kde=True, ax=ax)
```



Die ersten beiden Graphiken zu latitude und longitude zeigt zwei Peaks bei ca. 34 und 38 (*latitude*) sowie -122 und -118 (*longitude*). Diese Peaks weisen auf bevölkerungsreiche Gebiete hin, die später genauer analysiert werden. Das durchschnittliche Hausalter (_house_median_age_) liegt zwischen einem und 52 Jahren, und es zeigen sich mehrere Peaks. Die Verteilungen der gesamten bzw. Schlafräume (_total_rooms_ bzw. _total_bedrooms_) zeigt eine stark rechtsschiefe Verteilung. Zudem zeigt die Graphik zu _median_house_value_ eine vorliegende Kappung bei einem Preis von ca. 500.000 $ (siehe auch A-2, Teil 1). Ähnliches scheint für _housing_median_age_ zu gelten (bei einem Alter von ca. 50 Jahren).

Zu e):

In [25]:
```
# Korrelationen berechnen und visualisieren
plt.figure(figsize=(7, 7))
corr = df.corr(numeric_only=True)
# Wir nutzen ein mask-array, um das rechte obere Dreieck auszublenden
mask = np.zeros_like(corr, dtype=bool)
mask[np.triu_indices_from(mask)] = True
ax = sns.heatmap(corr, linewidths=.5, annot=True, mask=mask, cmap='coolwarm')
```



Aus der Korrelationsmatrix ergeben sich (erwartbare) hohe Korrelationen zwischen:

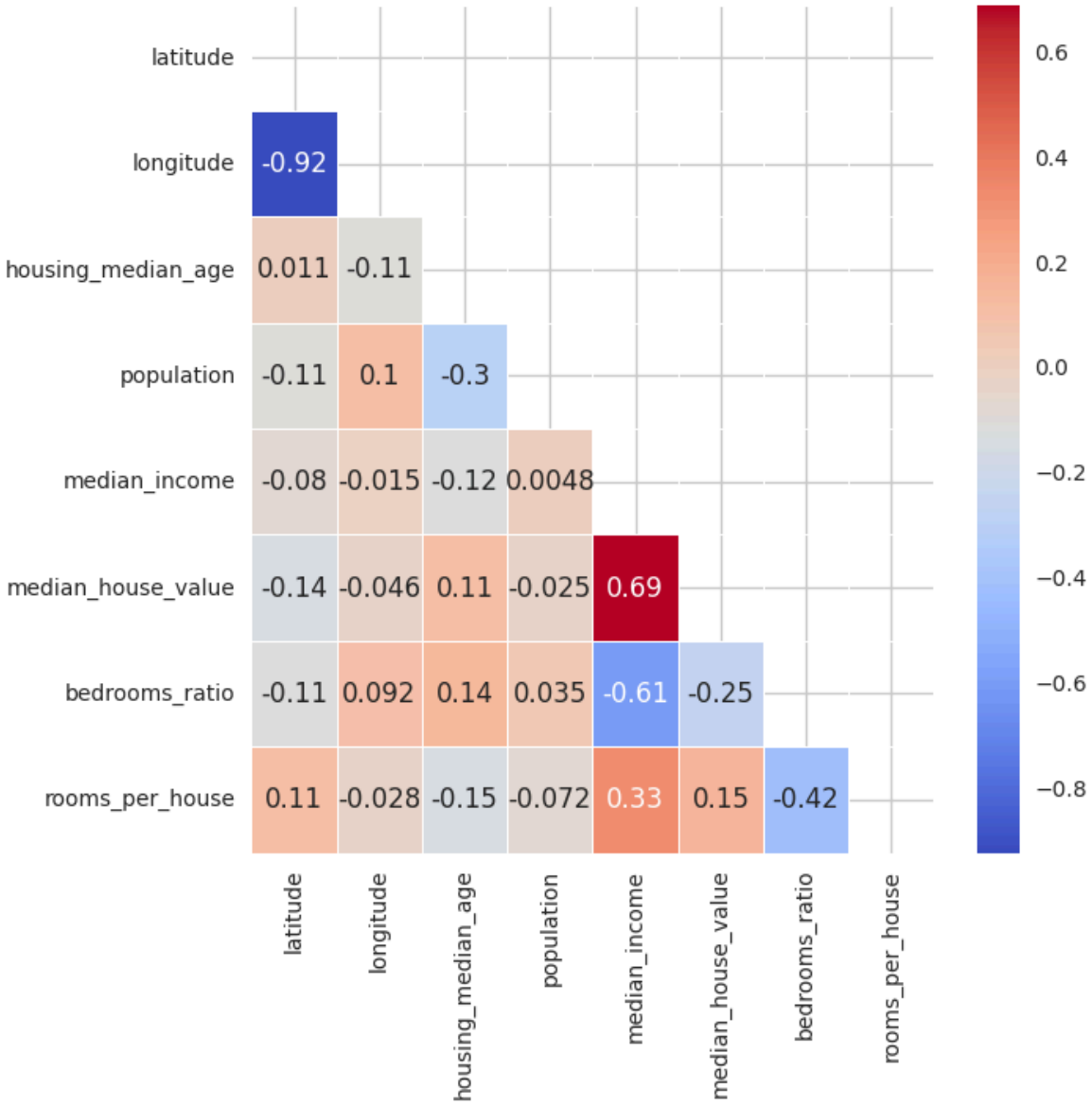
- *households* und *population*,
- _total_rooms_ und *population*,
- _total_bedrooms_ und *population* sowie
- _bedrooms_per_house_ und _rooms_per_house_

Daher werden *households*, _total_rooms_, _total_bedrooms_ sowie _bedrooms_per_house_ aus dem Datensatz entfernt.

In [26]:
```
# Folgende mit der Wohnbevölkerung bzw. Raumanzahl hochkorrelierten Merkmale entf
df.drop(["households","total_rooms", "total_bedrooms","bedrooms_per_house"], axis
num_columns = list(df.select_dtypes(include=[np.number]).columns )
```

In [27]:
```
# Korrelationen berechnen und visualisieren
plt.figure(figsize=(7, 7))
```

```
corr = df.corr(numeric_only=True)
# Wir nutzen ein mask-array, um das rechte obere Dreieck auszublenden
mask = np.zeros_like(corr, dtype=bool)
mask[np.triu_indices_from(mask)] = True
ax = sns.heatmap(corr, linewidths=.5, annot=True, mask=mask, cmap='coolwarm')
```



Aufgabe A-3: Hauspreise sowie regionale Strukturen untersuchen und visualisieren [Lernziele 3.3/3.4 & 5.2; 10 Punkte]

a) Visualisieren Sie den Datensatz mit Hilfe der Karte von Kalifornien (siehe: https://github.com/ageron/handson-ml3/blob/main/02_end_to_end_machine_learning_project.ipynb) und den folgenden Merkmalen:

→ _median_house_value_

→ _housing_median_age_

→ _median_income_ Die Ergebnisse sind zu kommentieren und zu interpretieren. b) Betrachtet werden soll die Abhängigkeit des Hauspreises von

den verfügbaren numerischen Merkmalen. Dazu ist eine geeignete Visualisierungsmöglichkeit zu wählen, auszuführen und zu interpretieren.

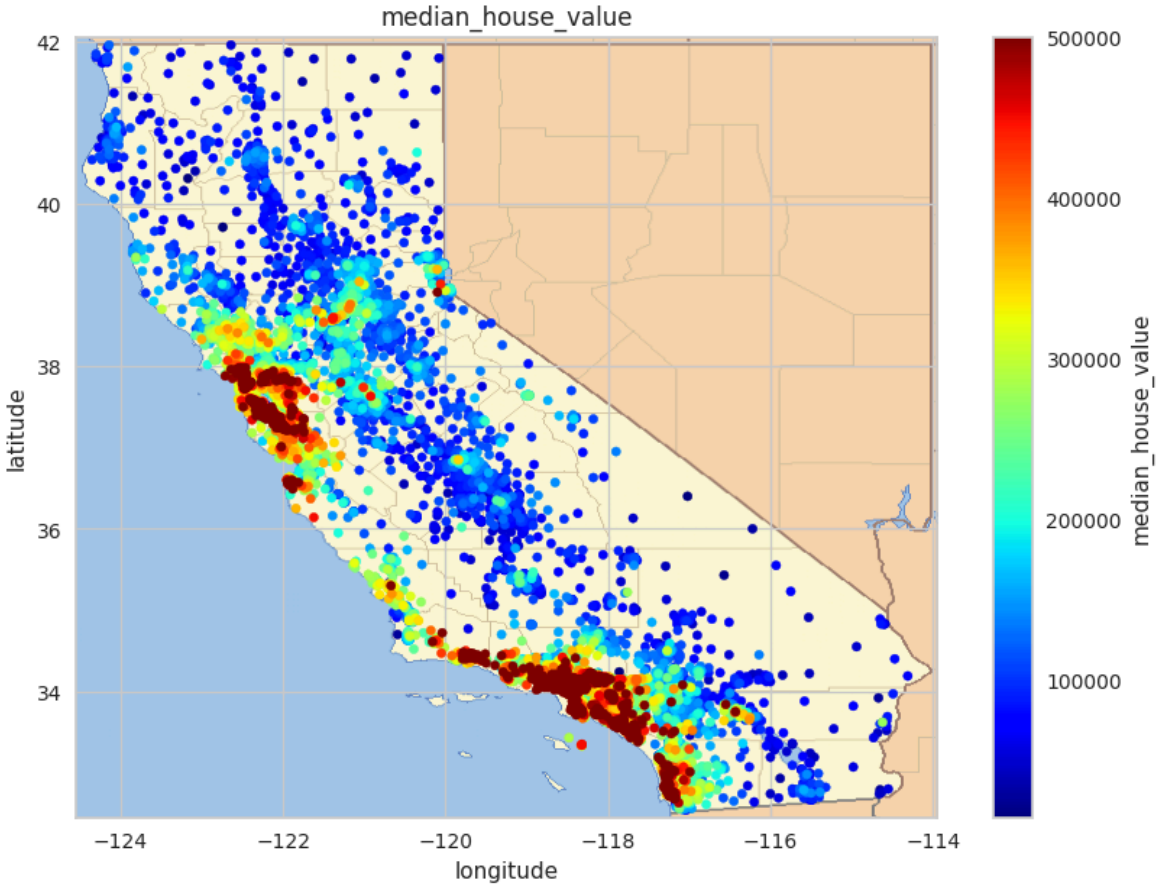
Lösungsvorschlag

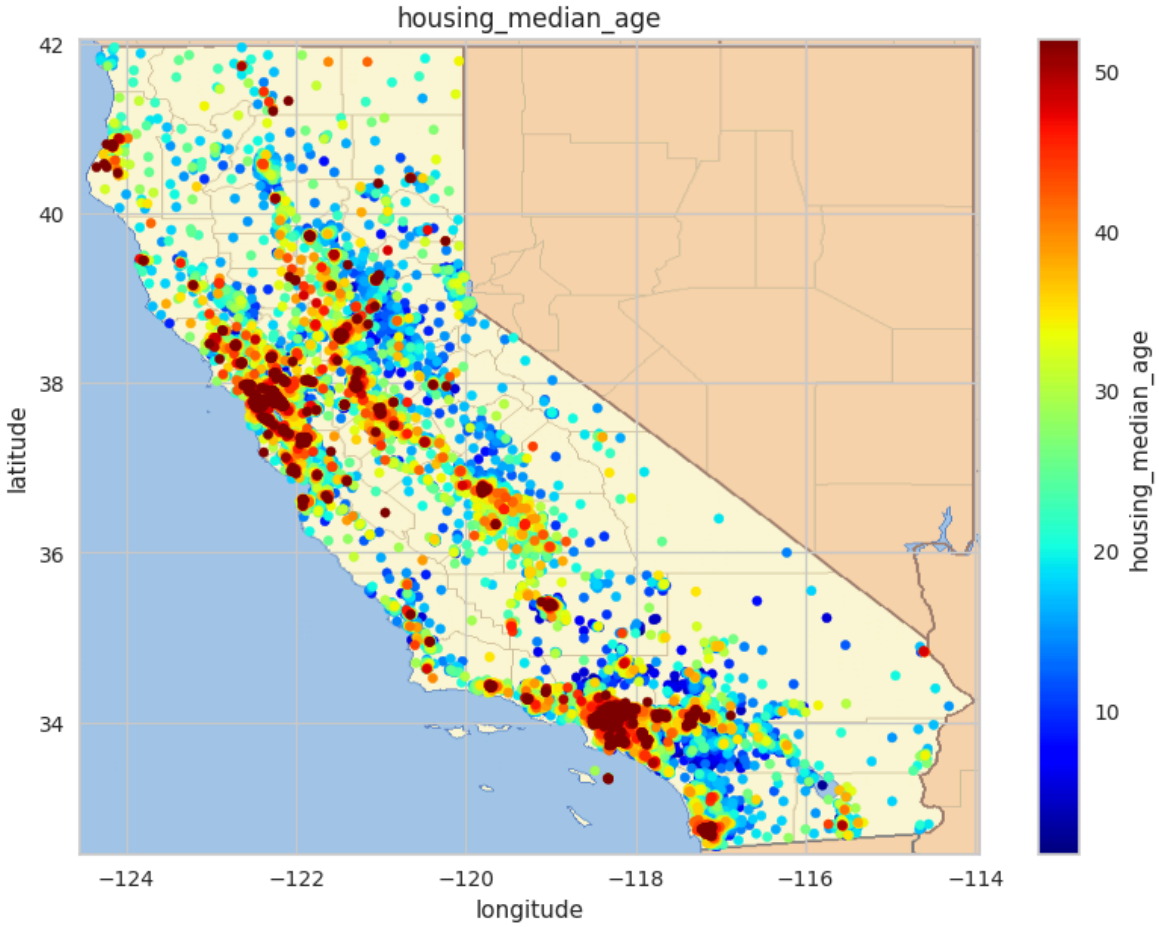
Zu a):

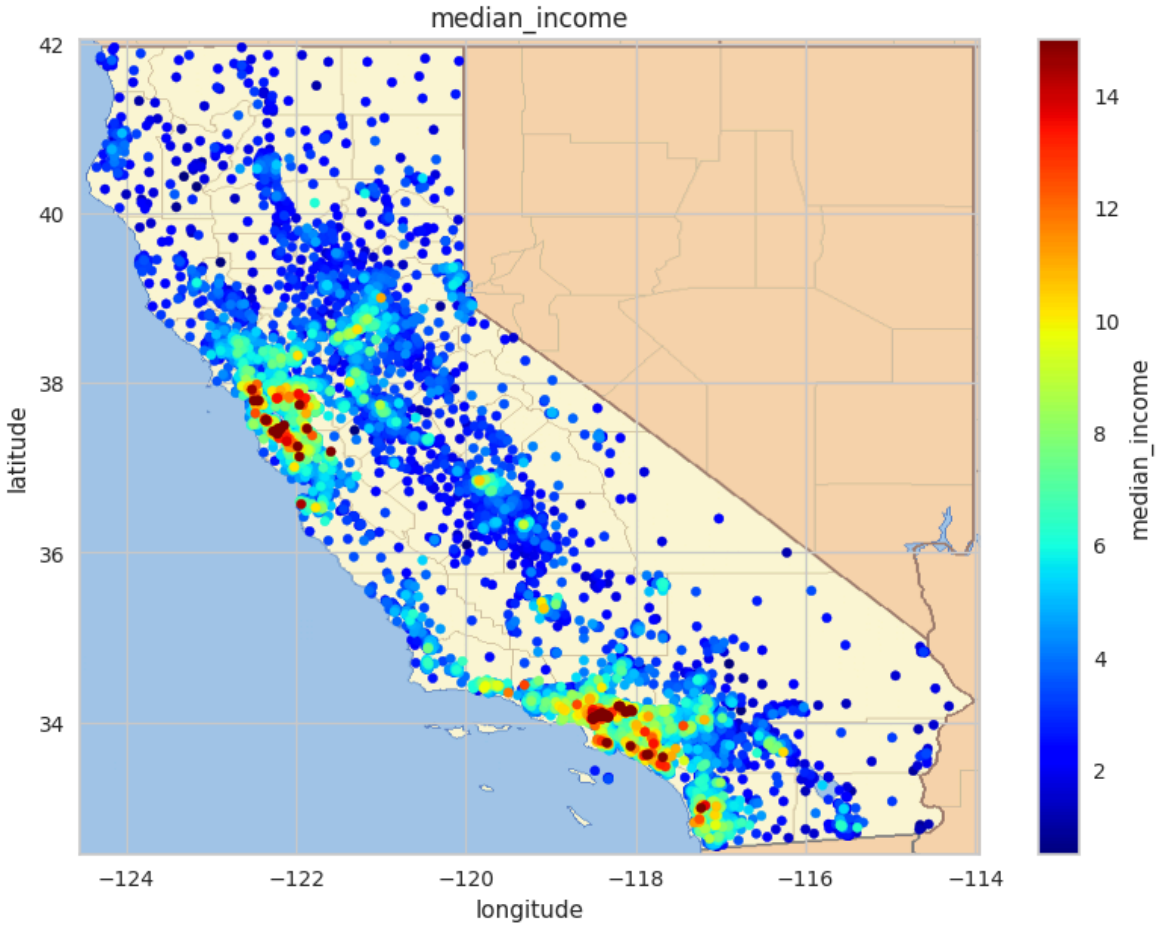
In [28]:
```
# Quelle: https://github.com/ageron/handson-ml3/blob/main/02_end_to_end_machine_l

# Landkarte California laden
IMAGES_PATH = Path() / "images" / "end_to_end_project"
IMAGES_PATH.mkdir(parents=True, exist_ok=True)

# Download the California image
filename = "california.png"
if not (IMAGES_PATH / filename).is_file():
    homl3_root = "https://github.com/ageron/handson-ml3/raw/main/"
    url = homl3_root + "images/end_to_end_project/" + filename
    print("Downloading", filename)
    urllib.request.urlretrieve(url, IMAGES_PATH / filename)
else:
  print(filename + " is already available")

# Plot-Funktion für Landkarte California definieren
def plot_ca(input_df,feature_name = 'population'):
    input_df.plot(kind="scatter", x="longitude", y="latitude", c=feature_name, ti
              cmap="jet", colorbar=True, legend=True, sharex=False, figsize=(10, 7
    california_img = plt.imread(IMAGES_PATH / filename)
    axis = -124.55, -113.95, 32.45, 42.05
    plt.axis(axis)
    plt.imshow(california_img, extent=axis)
    plt.show()
```

```
Downloading california.png
```

In [29]:
```
for feat in ['median_house_value', 'housing_median_age', 'median_income']:
  df.sort_values(by=feat,inplace=True)
  plot_ca(df, feat)
```


median_house_value
latitude
longitude
median_house_value
42
40
38
36
34
−124
−122
−120
−118
−116
−114
500000
400000
300000
200000
100000


housing_median_age
latitude
longitude
housing_median_age
42
40
38
36
34
−124
−122
−120
−118
−116
−114
50
40
30
20
10



Die Karten zeigen zunächst mehrere Anhäufungen von Punkten. Dabei stechen zwei Bereiche hervor: Der erste Bereich mit dem ungefähren Mittelpunkt bei ca. (-118;34) sowie (-122;37). Ein Abgleich mit einer Karte von Kalifornien zeigt den ersten Bereich als Los Angeles (bzw. Umgebung mit Palm Springs etwas weiter östlich) sowie den zweiten Bereich als San Francisco (bzw. Umgebung mit San Jose und Oakland). Desweiteren lassen sich die Hauptstadt Californiens, Sacramento sowie Bakersfield im mittleren Bereich der Karte erkennen. Zudem lassen sich vereinzelte Punkte zur Grenze zu Nevada erkennen.

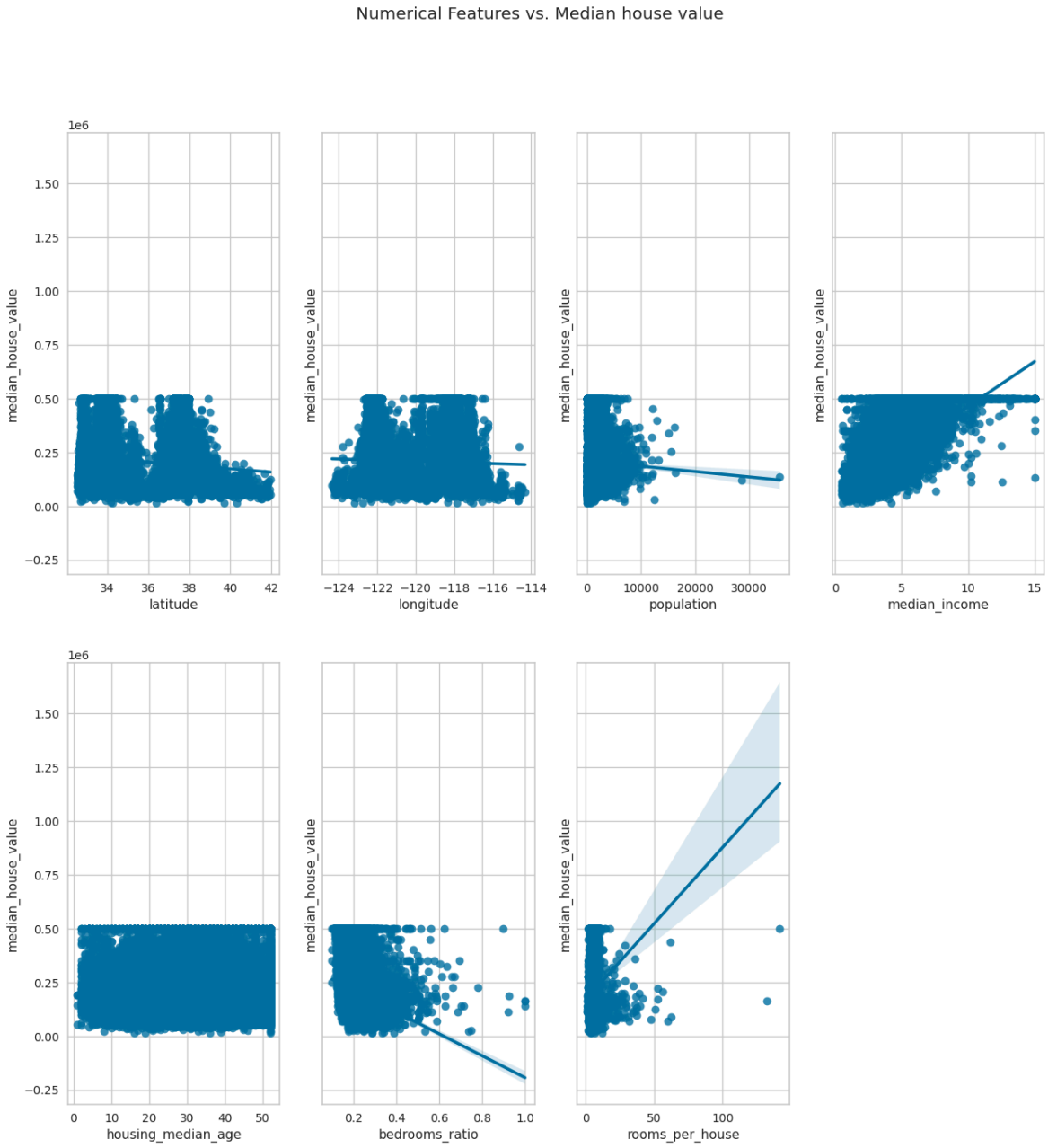
Die Karte zu _median_house_value_ zeigt dementsprechend, dass die hohe Preise entweder in Küstenregionen dieser beiden ersten oben genannten Bereiche liegen, oder in städtischen Gebieten. Niedrigere Werte finden sich tendentiell im Landesinneren oder in ländlichen Gebieten. Gleiches gilt für das Einkommen. Bei der Karte zur Population sind insbesondere die Bezirke nahe an der Grenze zu Nevada als tendenziell bevölkerungsarm zu erkennen. In den Bereichen die küstennah liegen sind nur vereinzelt Bezirke mit hoher Population zu identifizieren.

Die Karte zu _housing_median*age* zeigt, dass insbesondere in den Ballungsgebieten um Los Angeles und San Francisco Bezirke mit hohen Immobilienalter zu finden sind. Das ist nicht überraschend, da vor allem in den Bezirken der Großstädte mit älteren Häusern zu finden ist.

Aus der Karte zu _median_income_ zeigt sich, wie auf der ersten Karte, eine Herausstellung der beiden Gebiete Los Angeles und San Francisco.

Zu b):

In [30]:
```
fig, axes = plt.subplots(2, 4, figsize=(15, 15), sharey=True)
fig.suptitle('Numerical Features vs. Median house value')

sns.regplot(x=df["latitude"], y=df["median_house_value"], ax=axes[0,0])
sns.regplot(x=df["longitude"], y=df["median_house_value"], ax=axes[0,1])
sns.regplot(x=df["population"], y=df["median_house_value"], ax=axes[0,2])
sns.regplot(x=df["median_income"], y=df["median_house_value"], ax=axes[0,3])
sns.regplot(x=df["housing_median_age"], y=df["median_house_value"], ax=axes[1,0])
sns.regplot(x=df["bedrooms_ratio"], y=df["median_house_value"], ax=axes[1,1])
sns.regplot(x=df["rooms_per_house"], y=df["median_house_value"], ax=axes[1,2])

fig.delaxes(axes.flatten()[7])
```



Die Graphiken für latitude und longitude zeigen keinen deutlichen (linearen) Trend, man erkennt allerdings deutlich die bereits im vorhergehenden Teil herausgearbeiteten markanten Umgebungen Los Angeles und San Francisco. Wie bereits in der Korrelationsmatrix gesehen zeigt sich ein starker Zusammenhang mit _median_income_. Ebenfalls zeigt sich ein positiver Trend mit _rooms_per_house_, allerdings deutete die Korrelationsmatrix auf einen eher schwachen Zusammenhang hin.

Aufgabe A-4: Erste Hauspreisprognosemodelle auf Basis der numerischen Features [Lernziele 3.3/3.4, 4.1 & 6; 14 Punkte]

Ziel dieses Abschnitts ist die Erstellung erster Modelle basierend rein auf den numerischen Merkmalen. a) Es ist anzugeben, wie die Fehler- bzw. Gütemaße _Root Mean Squared Error (RMSE)_ bzw. _R^2_ eines Prognoseverfahrens bei einem Regressionsproblem berechnet werden. Im Anschluss soll der mathematischen Zusammenhang zwischen den beiden Größen aufgezeigt werden. Dabei soll einerseits darauf eingegangen werden, welche Rolle ein naives Nullmodell einnimmt, das stets den Mittelwert des Zielmerkmals ausgibt, und andererseits begründet werden, wieso der _RMSE_ bei steigendem _R^2_ niedriger wird und umgekehrt. b) Der Datensatz ist in einen Trainings- (x_train, y_train) und Validierungsteil (x_val, y_val) zu splitten. Dabei ist die Spalte _sample_ zu berücksichtigen: Zeilen mit dem Eintrag A sollten für das Training und Zeilen mit dem Eintrag B für die Validierung verwendet werden. Die entsprechend benötigten Datenstrukturen sind anzulegen. c) Für die Ergebnisse dieser und der nachfolgenden Modelle in Teil A ist eine geeignete Datenstruktur anzulegen. Diese soll für jedes Modell den Wert für den Root Mean Squared Error (RMSE) enthalten. d) Zu verwenden ist ein einfaches lineares Modell ohne besondere Anpassungen. Ein Ergebnisbericht ist auszugeben und die Ergebniswerte zu speichern und zu kommentieren. e) Analog zu Teilaufgabe d) ist CatBoost mit Standardparametern zu verwenden, wobei die Ausgabe der einzelnen Iterationen sowohl hier als auch in nachfolgenden Aufgaben zu unterdrücken ist. Ein Ergebnisbericht samt allen verwendeten Parametern ist auszugeben und die Ergebniswerte zu speichern. Zusätzlich ist die Feature-Importance auszugeben und zusammen mit den Ergebniswerten zu kommentieren.

Lösungsvorschlag

Zu a):

Im Folgenden seien $(y_i)_{i\in\{1,\ldots,n\}}$ die tatsächlichen Werte des Zielmerkmals, $(\hat{y}_i)_{i\in\{1,\ldots,n\}}$, die von einem (Machine-Learning-)Verfahren für das Zielmerkmal prognostizieren Werte und $\bar{y} := \frac{1}{n}\sum_{i=1}^{n} y_i$ das arithmetische Mittel über alle y_i.

Das Fehlermaß *Root Mean Squared Error* ist gegeben durch

$$RMSE := \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}.$$

Das Gütemaß R^2 ist gegeben durch

$$R^2 := 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}.$$

Hieraus erhalten wir

$$R^2 = 1 - \frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2} = 1 - \frac{RMSE^2}{\frac{1}{n}\cdot\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

wobei der Nenner auf der rechten Seite unabhängig ist von den Prognosewerten und damit vom verwendeten (Machine-Learning-)Verfahren.

Ein naives Nullmodell, das stets den Mittelwert des Zielmerkmals ausgibt, taucht bereits in der Definition von R^2 auf: Hier wird der mittlere quadratische Fehler des betrachteten (Machine-Learning-)Verfahrens ins Verhältnis gesetzt zum mittleren quadratischen Fehler des besagten Nullmodells. Dadurch stellt das Nullmodell eine Benchmark dar: Der R^2-Wert des Nullmodells ist 0, und jedes vernünftige (Machine-Learning-)Verfahren sollte zumindest diese untere Schranke übertreffen.

Da der *RMSE* stets nichtnegativ ist, folgt aus der letzten der obigen Gleichungen, dass der *RMSE* bei steigendem R^2 niedriger wird und bei fallendem R^2 größer wird.

Zu b):

In [31]:
```
# Einfache Stichprobeneinteilung
num_features = [elem for elem in num_columns if elem != 'median_house_value']
print(num_features)
x_train = df.query('sample =="A"')[num_features]
y_train = df.query('sample =="A"').median_house_value
x_val = df.query('sample =="B"')[num_features]
y_val = df.query('sample =="B"').median_house_value
```

```
['latitude', 'longitude', 'housing_median_age', 'population', 'median_income',
'bedrooms_ratio', 'rooms_per_house']
```

Zu c):

In [32]:
```
# Datenstruktur für die Modellgüte (rmse) in A anlegen
results_a_df = pd.DataFrame(columns=['Model','RMSE_val'])
results_a_df.set_index('Model',inplace=True)
```

Zu d):

In [33]:
```
# Einfaches lineares Regressionsmodell (OLS) fitten: Num. Features
# Berücksichtigung intercept
x_train = sm.add_constant(x_train)
x_val = sm.add_constant(x_val)
model_ols_num = sm.OLS(y_train, x_train)

results_ols_num = model_ols_num.fit()
print(results_ols_num.summary2())
```

```
                            Results: Ordinary least squares
==============================================================================
=======
Model:                      OLS                    Adj. R-squared:          0.61
8
Dependent Variable:         median_house_value     AIC:                     3632
70.1924
Date:                       2024-07-27 20:36       BIC:                     3633
30.8039
No. Observations:           14421                  Log-Likelihood:          -1.8
163e+05
Df Model:                   7                      F-statistic:             333
4.
Df Residuals:               14413                  Prob (F-statistic):      0.00
R-squared:                  0.618                  Scale:                   5.09
71e+09
------------------------------------------------------------------------------
-------
                       Coef.      Std.Err.     t     P>|t|      [0.025        0.
975]
------------------------------------------------------------------------------
-------
const              -3725455.2360 77619.4535 -47.9964 0.0000 -3877599.3459 -35733
11.1261
latitude             -41339.3543   853.0430 -48.4610 0.0000   -43011.4284   -396
67.2802
longitude            -42866.5765   891.1347 -48.1034 0.0000   -44613.3151   -411
19.8379
housing_median_age      915.8618    53.0070  17.2781 0.0000      811.9611     10
19.7624
population               -1.1476     0.5837  -1.9659 0.0493       -2.2917
-0.0034
median_income         42959.3531   433.2564  99.1546 0.0000    42110.1149    438
08.5914
bedrooms_ratio       344528.4647 14145.7329  24.3556 0.0000   316801.0091   3722
55.9202
rooms_per_house        2769.9881   281.0930   9.8543 0.0000     2219.0096     33
20.9666
------------------------------------------------------------------------------
-------
Omnibus:                    2868.247            Durbin-Watson:
1.905
Prob(Omnibus):              0.000               Jarque-Bera (JB):            7
582.208
Skew:                       1.077               Prob(JB):
0.000
Kurtosis:                   5.825               Condition No.:               2
32305
==============================================================================
=======
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.32e+05. This might indicate
that
there are strong multicollinearity or other numerical                problems.
```

In [34]:
```
# Modelgüte RMSE an Validierungsstichprobe ermitteln
pred_val_ols_num = results_ols_num.predict(x_val)
rmse = mean_squared_error(y_val, pred_val_ols_num , squared=False)
```

In [35]:
```
# Ergebnisse zum Modellvergleich innerhalb von A hinzufuegen und anzeigen
results_a_df.loc['Linear_num'] = [rmse]
results_a_df
```

Out[35]:

Model	RMSE_val
Linear_num	73084.95258

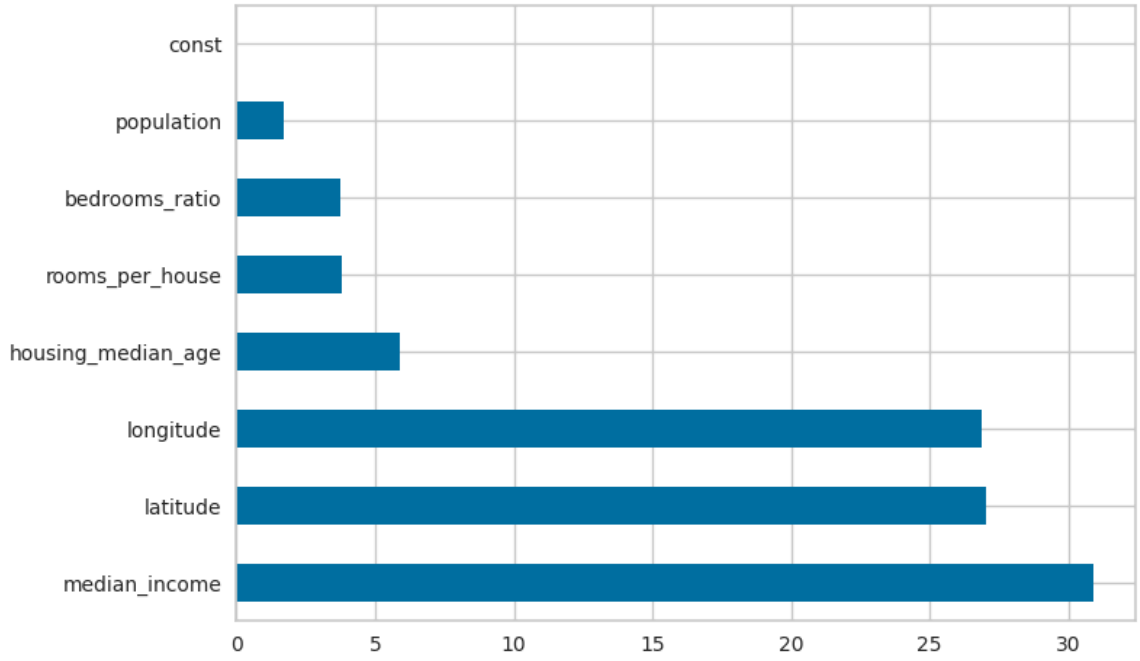
Die P-Werte zeigen an, dass die (ggf. mit Ausnahme von *population*, hier wären weitere Analysen notwendig) im Modell enthaltenen Merkmale signifikant sind. Insbesondere der Wert für R^2 von ca. 62 % gibt einen ersten Hinweis darauf, dass ein großer Anteil der Variabilität im Datensatz durch das einfache lineare Modell nicht erklärt werden kann. Mit dem Wert für RMSE ergibt sich damit eine erste Baseline, mit der die weiteren Modelle verglichen werden können.

Zu e):

In [36]:
```
# Einfaches CatBoost-Standardmodell aufsetzen, fitten und Standardparameter anze
model_cb_num = CatBoostRegressor(eval_metric='RMSE', random_seed=42, verbose=0)
model_cb_num.fit(x_train,y_train)
print(model_cb_num.get_all_params())
```

```
{'nan_mode': 'Min', 'eval_metric': 'RMSE', 'iterations': 1000, 'sampling_frequen
cy': 'PerTree', 'leaf_estimation_method': 'Newton', 'random_score_type': 'Normal
WithModelSizeDecrease', 'grow_policy': 'SymmetricTree', 'penalties_coefficient':
1, 'boosting_type': 'Plain', 'model_shrink_mode': 'Constant', 'feature_border_ty
pe': 'GreedyLogSum', 'bayesian_matrix_reg': 0.10000000149011612, 'eval_fractio
n': 0, 'force_unit_auto_pair_weights': False, 'l2_leaf_reg': 3, 'random_strengt
h': 1, 'rsm': 1, 'boost_from_average': True, 'model_size_reg': 0.5, 'pool_metain
fo_options': {'tags': {}}, 'subsample': 0.800000011920929, 'use_best_model': Fal
se, 'random_seed': 42, 'depth': 6, 'posterior_sampling': False, 'border_count':
254, 'classes_count': 0, 'auto_class_weights': 'None', 'sparse_features_conflict
_fraction': 0, 'leaf_estimation_backtracking': 'AnyImprovement', 'best_model_min
_trees': 1, 'model_shrink_rate': 0, 'min_data_in_leaf': 1, 'loss_function': 'RMS
E', 'learning_rate': 0.06241700053215027, 'score_function': 'Cosine', 'task_typ
e': 'CPU', 'leaf_estimation_iterations': 1, 'bootstrap_type': 'MVS', 'max_leave
s': 64}
```

In [37]:
```
# CatBoost: Feature-Importance plotten
(pd.Series(model_cb_num.feature_importances_, index=x_train.columns).nlargest(10
plt.show()
```



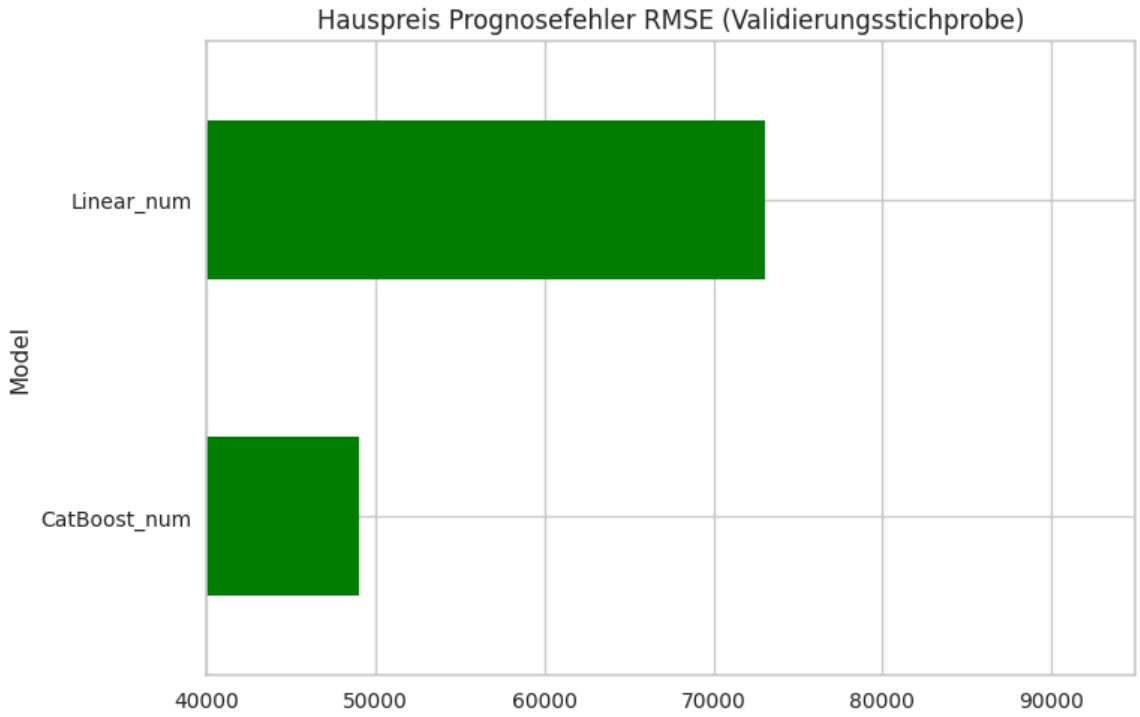
In [38]:
```
# Modelgüte RMSE an Validierungsstichprobe ermitteln
pred_val_cb_num = model_cb_num.predict(x_val)
rmse = mean_squared_error(y_val, pred_val_cb_num , squared=False)
```

In [39]:
```
# Ergebnisse zum Modellvergleich innerhalb von A hinzufuegen und anzeigen
results_a_df.loc['CatBoost_num'] = [rmse]
results_a_df
```

Out[39]:

Model	RMSE_val
Linear_num	73084.952580
CatBoost_num	48971.008265

In [40]:
```
plot_rmse(results_a_df)
```



Ein Catboost Modell ohne weitere Anpassungen zeigt bereits deutlich bessere Ergebnisse als das einfache lineare Modell. Der Plot der Feature-Importance zeigt, dass _median_income_ den größen Einfluss hat, gefolgt von *latitude* und *longitude*. Angesichts von A-2 e) sowie A-3 b) ist das nicht verwunderlich. Intuitiv kann das Abschneiden von *longitude* und *latitude* durch die Graphiken aus Aufgabe A-3 a) erklärt werden: Die Regionen mit hohem Preisniveau liegen an verschiedenen Küstenabschnitten mit Ausschlägen zum Landesinneren. Weitere Mermale folgen dann mit deutlich geringeren Werten.

Aufgabe A-5: Kategorielle Merkmale analysieren und aufbereiten [Lernziele 3.3/3.4 & 5.2; 7 Punkte]

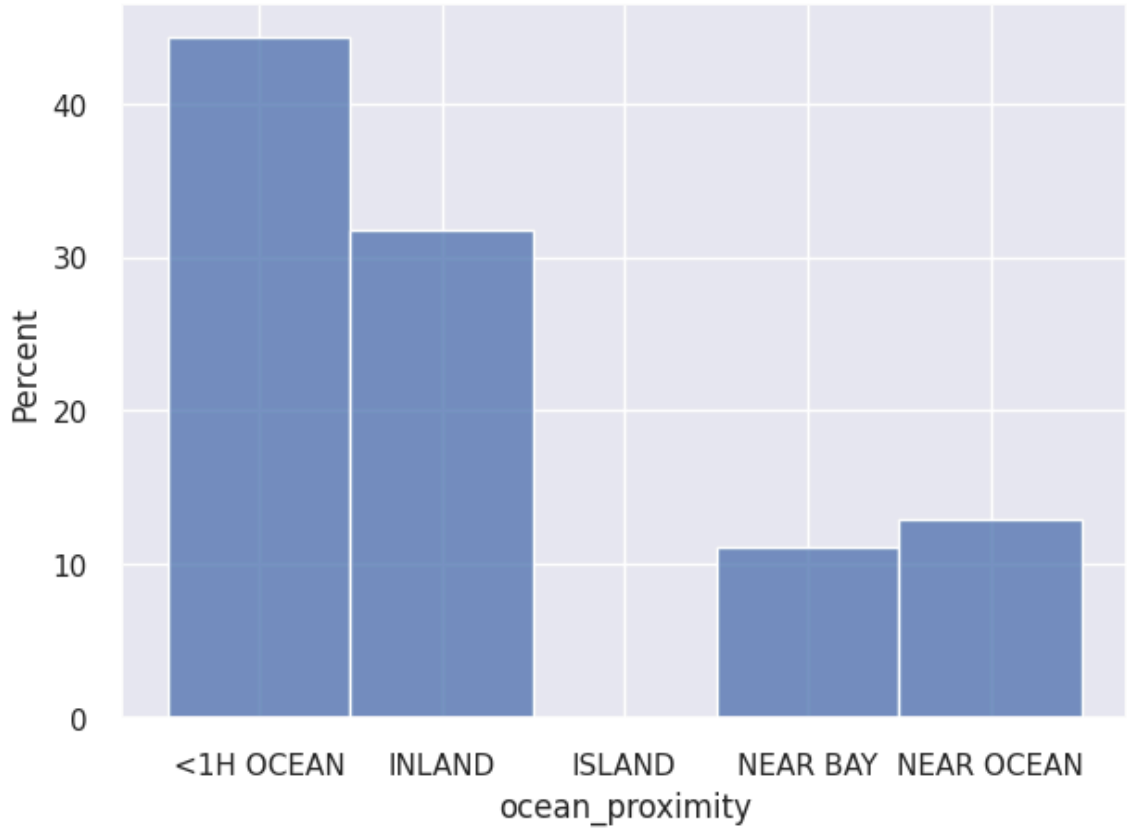
Hinweis: Der Datensatz ist für diesen Abschnitt (hinsichtlich der Features) auf die kategoriellen Merkmale zu beschränken. **a) Zu erstellen ist eine graphische prozentuale Häufigkeitsverteilung für die nominalen Merkmale _ocean _proximity_ und _country_name_. Die Ergebnisse sind zu erläutern. b) Der Einfluss der nominalen Merkmale auf die Hauspreise ist mit geeigneten Mitteln zu visualisieren. Auffälligkeiten sind zu kommentieren.**

Lösungsvorschlag

Zu a):

In [41]:
```
sns.set(rc={"figure.figsize":(7, 5)})
sns.histplot(data=df, x="ocean_proximity", stat='percent')
```

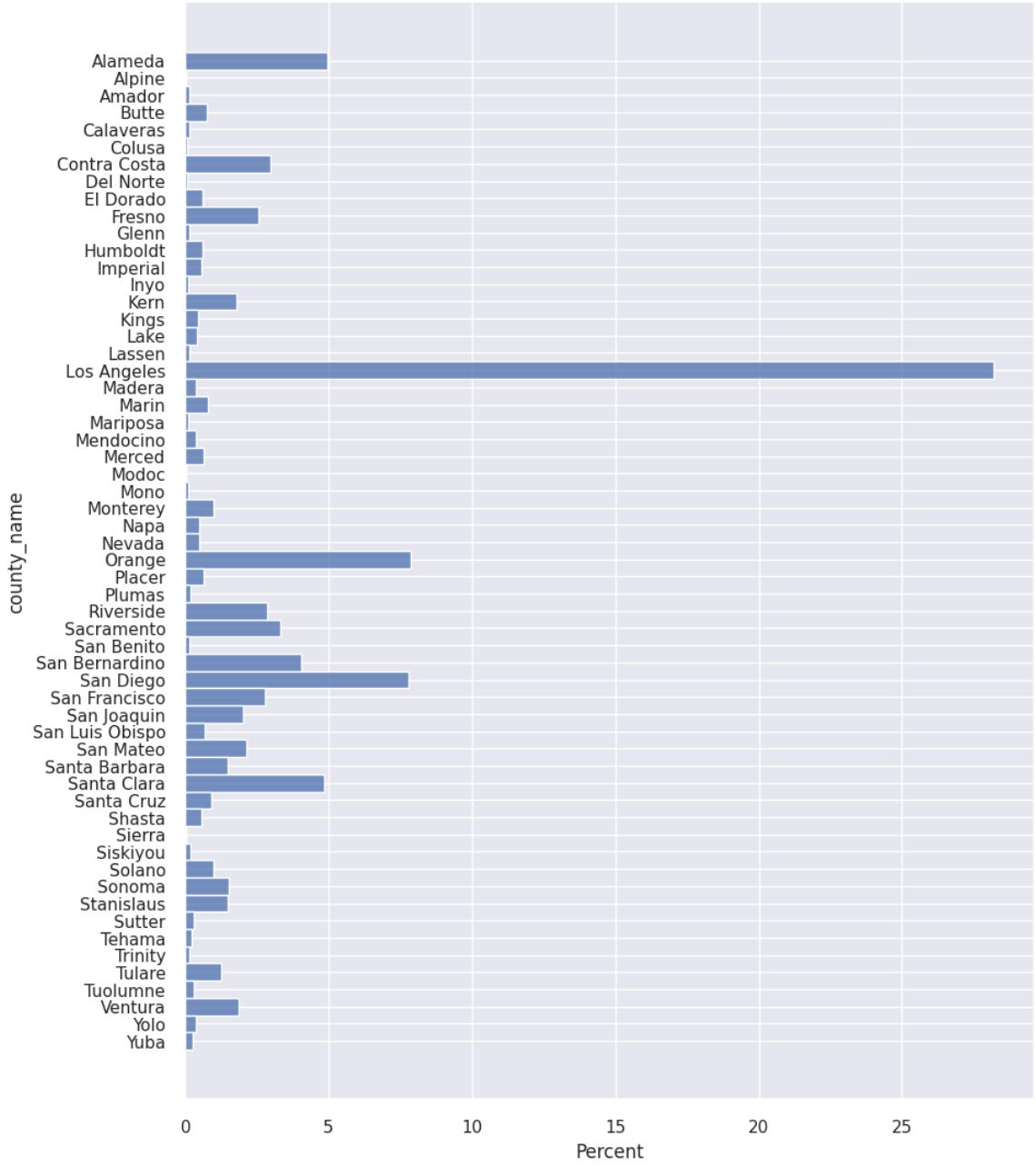
Out[41]:
```
<Axes: xlabel='ocean_proximity', ylabel='Percent'>
```



Die Auswertung zu ocean_proximity zeigt wenig überraschend, dass die meisten Bezirke überwiegend eine kurze Distanz zum Ozean haben oder im Inland zu finden sind. Der Anteil von Island ist verschwindend gering.

In [42]:
```
# Werteverteilung von county_name visualisieren
sns.set(rc={"figure.figsize":(10, 13)})
sns.histplot(data=df, y="county_name", stat='percent')
```

Out[42]:
```
<Axes: xlabel='Percent', ylabel='county_name'>
```

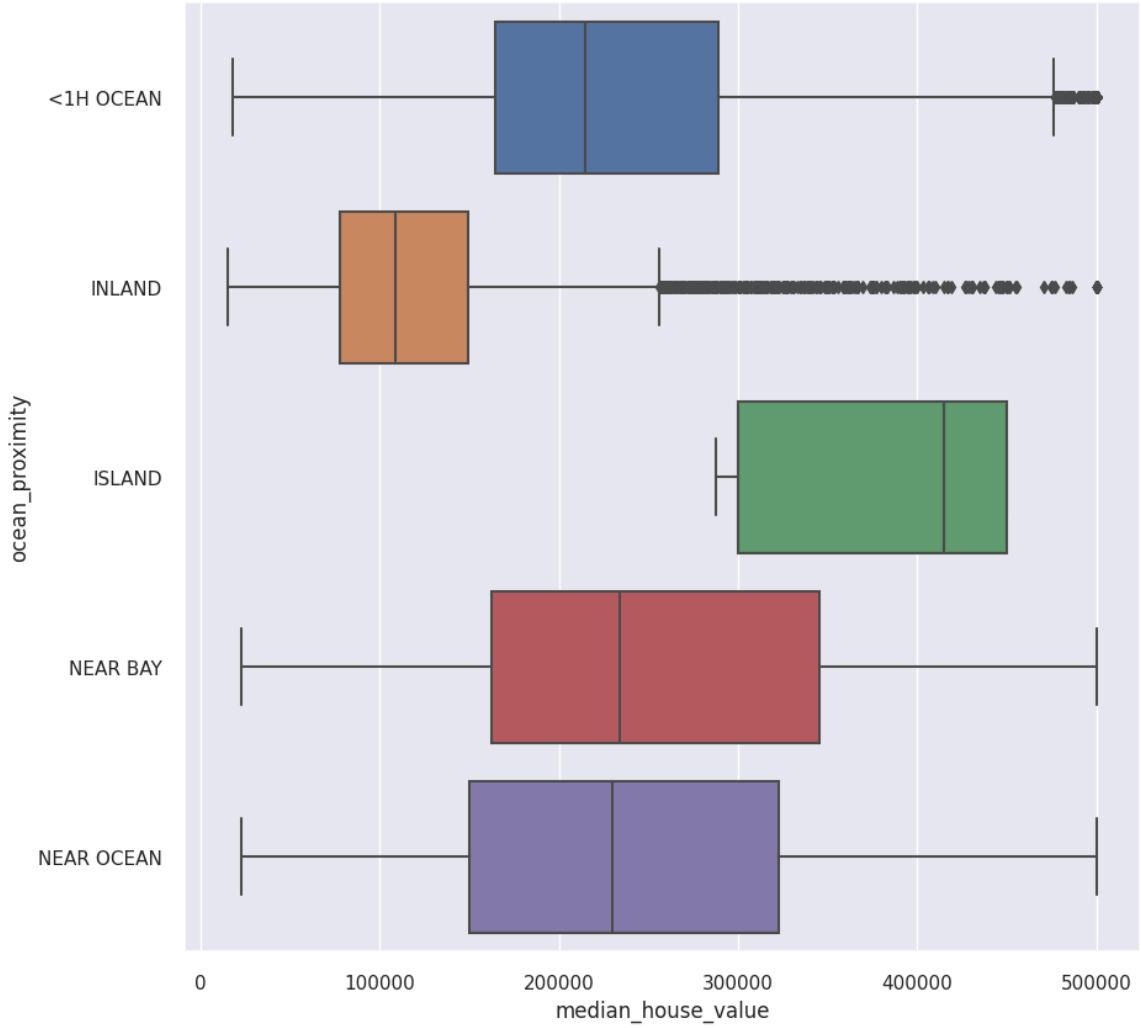


Bei Country Name ist Los Angeles mit einem Anteil von über 25 % führend. Dahinter folgt Orange (Orange Country), was ebenfalls im Süden Kaliforniens das (hinter Los Angeles Country) einwohnerreichste Country Kaliforniens ist, sowie San Diego. Das wird auch durch die Karte zu den Populationen aus Abschnitt A-3 visuell bestätigt.

Zu b):

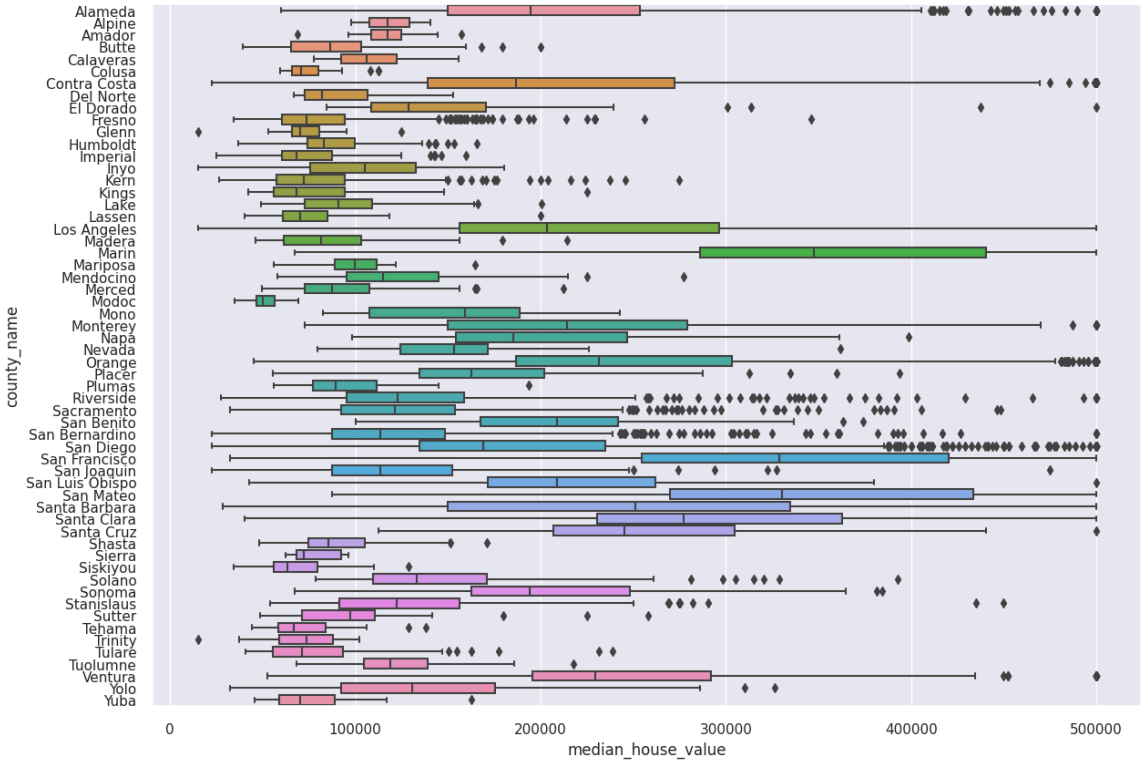
In [43]:
```
# Boxplots der Hauspreise in Abhängigkeit von:
sns.set(rc={"figure.figsize":(10, 10)})
sns.boxplot(y = 'ocean_proximity', x = 'median_house_value', data = df, orient='
```

Out[43]:
```
<Axes: xlabel='median_house_value', ylabel='ocean_proximity'>
```



Das Ergebnis spiegelt die Visualisierungen der Karten aus Aufgabenabschnitt A-3 wider: Bezüglich des Hauspreises liegen die Immobilien mit der Klasse Island vorne, direkt gefolgt mit den Gebieten in Küstennähe: Near Bay / Near Ocean bzw. < 1H ocean. Dahinter folgt Inland, wobei es viele Ausnahmen rechts des 75% Quantils zu geben scheint.

In [44]:
```
plt.figure(figsize=(14, 10))
sns.boxplot(y = 'county_name', x = 'median_house_value', data = df, orient='h')
plt.show()
```



Auch diese Grafik bestätigt das Bild aus Aufgabe A-3: Bezirke mit einem geringeren Hauspreis (wie etwa Modoc, Lassen, Plumas etc.) sind tendenziell an der Grenze zu Nevada (d.h. im Inland) zu finden, wobei Regionen in Küstennähe (z.B. Marin, San Francisco etc.) eher mit einem höheren Hauspreis versehen sind.

Aufgabe A-6: Hauspreisprognosemodelle auf Basis der kategoriellen Features erstellen [Lernziel 3.3/3.4, 4.1 & 6; 6 Punkte]

a) Ziel dieser Aufgabe ist die Erstellung weiterer Modelle basierend rein auf den kategoriellen Merkmalen. Die Datenstrukturen für Training und Validierung sind analog zu Aufgabe A-4 b) anzulegen und zu verwenden. b) Zu verwenden ist ein einfaches lineares Modell ohne besondere Anpassungen. Ein Ergebnisbericht ist auszugeben und die Ergebniswerte zu speichern und zu kommentieren. Was ergibt sich im Vergleich mit dem Modellen aus A-4? c) Analog zu Aufgabe A-4 ist catboost ohne weitere Anpassungen zu verwenden. Ein Ergebnisbericht ist auszugeben und die Ergebniswerte zu speichern. Zusätzlich ist die Feature-Importance auszugeben und zusammen mit den Ergebniswerten zu kommentieren. Was ergibt sich im Vergleich zu dem Modell aus A-4?

Lösungsvorschlag

Zu a):

In [45]:
```
# Einfache Stichprobeneinteilung
cat_features =['ocean_proximity', 'county_name']
train = df.query('sample =="A"')[cat_features +  ["median_house_value"]]
val = df.query('sample =="B"')[cat_features + ["median_house_value"]]
y_val = df.query('sample =="B"').median_house_value
```

Zu b):

In [46]:
```
# Einfaches lineares Regressionsmodell (OLS) fitten: Cat. Features
model_ols_cat = smf.ols(formula="median_house_value ~ C(ocean_proximity) + C(cou
results_ols_cat = model_ols_cat.fit()
print(results_ols_cat.summary2())
```

```
                                  Results: Ordinary least squares
==============================================================================
==================
Model:                          OLS                        Adj. R-squared:
0.373
Dependent Variable:             median_house_value         AIC:
370463.6912
Date:                           2024-07-27 20:36           BIC:
370933.4305
No. Observations:               14421                      Log-Likelihood:
-1.8517e+05
Df Model:                       61                         F-statistic:
141.8
Df Residuals:                   14359                      Prob (F-statistic):
0.00
R-squared:                      0.376                      Scale:
8.3624e+09
------------------------------------------------------------------------------
------------------
                                     Coef.      Std.Err.     t     P>|t|     [0.
025       0.975]
------------------------------------------------------------------------------
------------------
Intercept                          200212.3850  5590.7761  35.8112 0.0000  18925
3.7414  211171.0285
C(ocean_proximity)[T.INLAND]       -18512.0178  4155.9706  -4.4543 0.0000  -2665
8.2572  -10365.7785
C(ocean_proximity)[T.ISLAND]       135557.7329 64679.4234   2.0958 0.0361    877
7.7059  262337.7598
C(ocean_proximity)[T.NEAR BAY]      12330.7637  4878.5046   2.5276 0.0115    276
8.2642   21893.2631
C(ocean_proximity)[T.NEAR OCEAN]    24767.7124  3190.4938   7.7630 0.0000   1851
3.9324   31021.4925
C(county_name)[T.Alpine]           -63000.3672 53123.3009  -1.1859 0.2357 -16712
8.9010   41128.1667
C(county_name)[T.Amador]           -63409.8910 20804.3201  -3.0479 0.0023 -10418
9.0465  -22630.7354
C(county_name)[T.Butte]            -88484.6264 10584.8353  -8.3596 0.0000 -10923
2.2712  -67736.9816
C(county_name)[T.Calaveras]        -77896.0193 19954.7664  -3.9036 0.0001 -11700
9.9398  -38782.0989
C(county_name)[T.Colusa]          -103523.4441 26035.9865  -3.9762 0.0001 -15455
7.3417  -52489.5464
C(county_name)[T.Contra Costa]       9171.6898  5753.0596   1.5942 0.1109   -210
5.0504   20448.4300
C(county_name)[T.Del Norte]       -130960.9751 25140.1537  -5.2092 0.0000 -18023
8.9246  -81683.0256
C(county_name)[T.El Dorado]        -35677.3786 11433.6658  -3.1204 0.0018  -5808
8.8409  -13265.9164
C(county_name)[T.Fresno]           -98424.5051  7657.5982 -12.8532 0.0000 -11343
4.3869  -83414.6233
C(county_name)[T.Glenn]           -111044.1172 23606.3686  -4.7040 0.0000 -15731
5.6499  -64772.5845
C(county_name)[T.Humboldt]        -127193.8915 11384.2625 -11.1728 0.0000 -14950
8.5170 -104879.2660
C(county_name)[T.Imperial]        -109206.8889 11202.8928  -9.7481 0.0000 -13116
6.0063  -87247.7715
C(county_name)[T.Inyo]             -81806.3083 22945.9455  -3.5652 0.0004 -12678
3.3264  -36829.2902
C(county_name)[T.Kern]             -98805.5332  8091.1379 -12.2116 0.0000 -11466
5.2089  -82945.8575
C(county_name)[T.Kings]           -105844.1172 12855.8077  -8.2332 0.0000 -13104
3.1613  -80645.0730
C(county_name)[T.Lake]             -93512.4328 12970.2747  -7.2097 0.0000 -11893
```

```
5.8472  -68089.0185
C(county_name)[T.Lassen]          -110444.8116 22342.5204  -4.9433 0.0000 -15423
9.0385  -66650.5847
C(county_name)[T.Los Angeles]       39229.8822  5654.2540   6.9381 0.0000   2814
6.8137   50312.9507
C(county_name)[T.Madera]          -100113.5747 13870.4909  -7.2177 0.0000 -12730
1.5290  -72925.6204
C(county_name)[T.Marin]            140846.8515  9218.9542  15.2780 0.0000  12277
6.5101  158917.1928
C(county_name)[T.Mariposa]         -83478.9386 25138.1221  -3.3208 0.0009 -13275
2.9060  -34204.9712
C(county_name)[T.Mendocino]        -80739.7941 14155.6792  -5.7037 0.0000 -10848
6.7544  -52992.8338
C(county_name)[T.Merced]           -86543.2243 11247.3847  -7.6945 0.0000 -10858
9.5517  -64496.8969
C(county_name)[T.Modoc]           -133900.3672 46100.0655  -2.9046 0.0037 -22426
2.4520  -43538.2823
C(county_name)[T.Mono]             -23731.1364 26035.9865  -0.9115 0.3621  -7476
5.0341   27302.7613
C(county_name)[T.Monterey]          25937.6603  9403.8332   2.7582 0.0058    750
4.9322   44370.3884
C(county_name)[T.Napa]               1765.2225 11385.5602   0.1550 0.8768  -2055
1.9466   24082.3915
C(county_name)[T.Nevada]           -30195.1040 13465.4416  -2.2424 0.0249  -5658
9.1095   -3801.0985
C(county_name)[T.Orange]            59728.8317  6197.5292   9.6375 0.0000   4758
0.8737   71876.7897
C(county_name)[T.Placer]            -4711.0055 11116.2426  -0.4238 0.6717  -2650
0.2773   17078.2664
C(county_name)[T.Plumas]           -86865.5845 19954.7664  -4.3531 0.0000 -12597
9.5050  -47751.6641
C(county_name)[T.Riverside]        -43663.1228  7308.2679  -5.9745 0.0000  -5798
8.2722  -29337.9735
C(county_name)[T.Sacramento]       -49883.3413  7254.7445  -6.8760 0.0000  -6410
3.5779  -35663.1047
C(county_name)[T.San Benito]        26838.6714 19172.0265   1.3999 0.1616  -1074
0.9778   64418.3207
C(county_name)[T.San Bernardino]   -56945.0696  6985.0456  -8.1524 0.0000  -7063
6.6616  -43253.4776
C(county_name)[T.San Diego]        -18950.7869  6256.9264  -3.0288 0.0025  -3121
5.1712   -6686.4027
C(county_name)[T.San Francisco]    122068.6091  5838.5409  20.9074 0.0000  11062
4.3146  133512.9037
C(county_name)[T.San Joaquin]      -59044.0338  7904.6571  -7.4695 0.0000  -7453
8.1832  -43549.8845
C(county_name)[T.San Luis Obispo]    1558.4467 11258.2247   0.1384 0.8899  -2050
9.1285   23626.0218
C(county_name)[T.San Mateo]        125785.3266  7506.3864  16.7571 0.0000  11107
1.8393  140498.8140
C(county_name)[T.Santa Barbara]     49216.3500  8256.7219   5.9608 0.0000   3303
2.1082   65400.5917
C(county_name)[T.Santa Clara]      100343.8933  6136.8844  16.3510 0.0000   8831
4.8070  112372.9797
C(county_name)[T.Santa Cruz]        44584.0184  9864.5718   4.5196 0.0000   2524
8.1831   63919.8537
C(county_name)[T.Shasta]           -90852.0913 11433.6658  -7.9460 0.0000 -11326
3.5535  -68440.6291
C(county_name)[T.Sierra]          -113450.3672 46100.0655  -2.4610 0.0139 -20381
2.4520  -23088.2823
C(county_name)[T.Siskiyou]        -112204.5338 19571.4939  -5.7331 0.0000 -15056
7.1907  -73841.8769
C(county_name)[T.Solano]           -45720.9199  9180.2151  -4.9804 0.0000  -6371
5.3277  -27726.5121
C(county_name)[T.Sonoma]             4203.2200  8190.9344   0.5132 0.6079  -1185
```

```
2.0699   20258.5098
C(county_name)[T.Stanislaus]       -52894.3764  8552.5072  -6.1847 0.0000  -6965
8.3956  -36130.3571
C(county_name)[T.Sutter]           -86067.8672 15609.9187  -5.5137 0.0000 -11666
5.3247  -55470.4096
C(county_name)[T.Tehama]          -114348.6430 17971.3231  -6.3628 0.0000 -14957
4.7584  -79122.5276
C(county_name)[T.Trinity]         -110155.3672 21277.4594  -5.1771 0.0000 -15186
1.9369  -68448.7974
C(county_name)[T.Tulare]          -104148.2166  8920.0973 -11.6757 0.0000 -12163
2.7599  -86663.6733
C(county_name)[T.Tuolumne]         -59959.4581 14988.7972  -4.0003 0.0001  -8933
9.4373  -30579.4789
C(county_name)[T.Ventura]           48525.5924  7817.1303   6.2076 0.0000   3320
3.0070   63848.1778
C(county_name)[T.Yolo]             -41800.3672 13764.7606  -3.0368 0.0024  -6878
1.0764  -14819.6579
C(county_name)[T.Yuba]            -106344.2696 15445.7282  -6.8850 0.0000 -13661
9.8926  -76068.6466
------------------------------------------------------------------------------
------------------
Omnibus:                        2151.899                Durbin-Watson:
1.167
Prob(Omnibus):                  0.000                   Jarque-Bera (JB):
3405.894
Skew:                           1.039                   Prob(JB):
0.000
Kurtosis:                       4.163                   Condition No.:
95
==============================================================================
==================
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

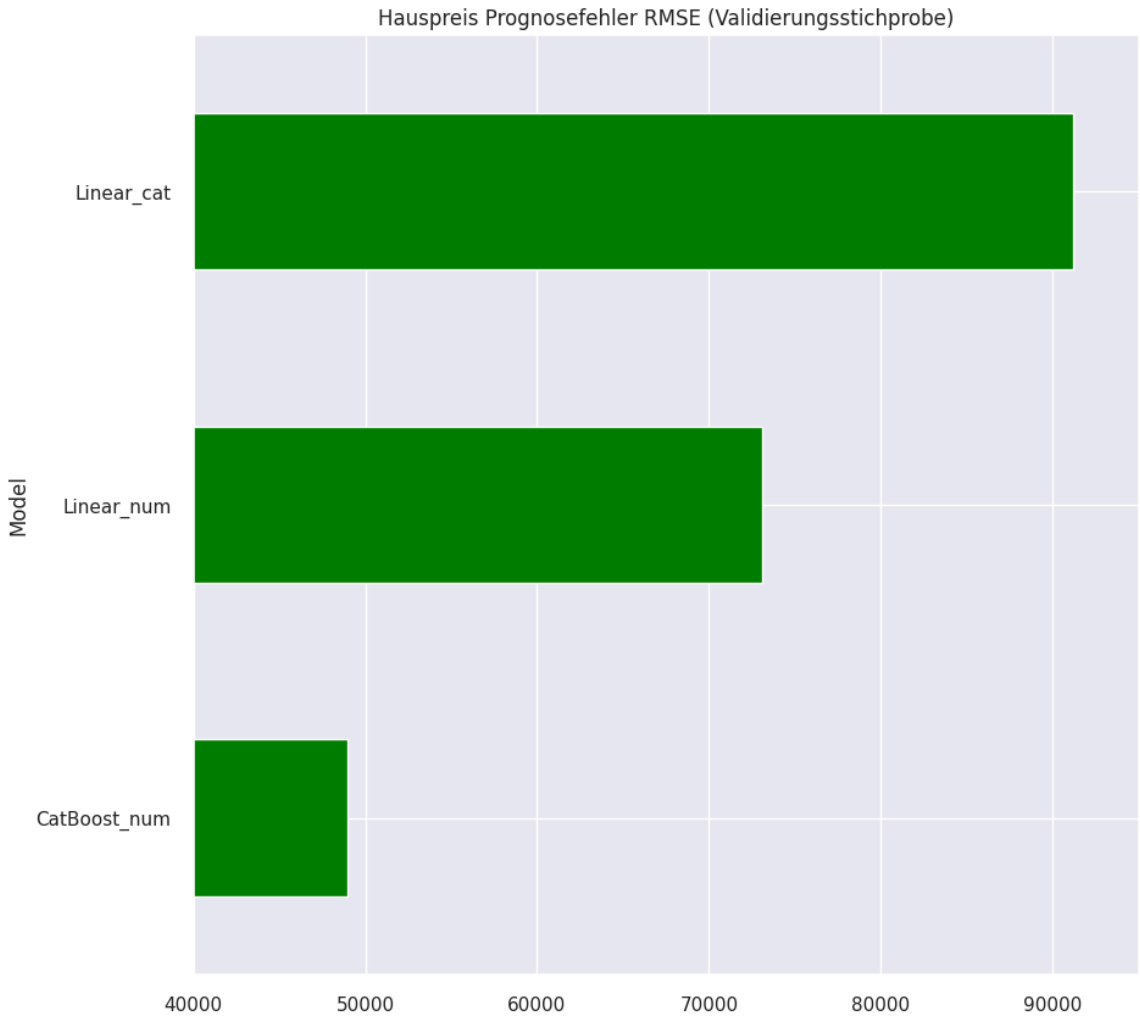
In [47]:
```
# Modelgüte RMSE an Validierungsstichprobe ermitteln
pred_val_ols_cat = results_ols_cat.predict(val)
rmse = mean_squared_error(y_val, pred_val_ols_cat , squared=False)
```

In [48]:
```
# Ergebnisse zum Modellvergleich innerhalb von A hinzufuegen und anzeigen
results_a_df.loc['Linear_cat'] = [rmse]
results_a_df
```

Out[48]:

Model	RMSE_val
CatBoost_num	48971.008265
Linear_num	73084.952580
Linear_cat	91193.295391

In [49]:
```
plot_rmse(results_a_df)
```



In der Ausgabe des linearen Modells ist zunächst zu erkennen, dass die Ausprägungen von _ocean_proximity_ signifikant sind. Dagegen sind nicht alle Ausprägungen von _country_name_ signifikant. Im Vergleich zum ersten linearen Modell stehen nur zwei Merkmale zur Verfügung. Beide beinhalten geographische Informationen: Nähe zum Ozean sowie den Namen des County. Allerdings scheinen diese Informationen nicht ausreichend für eine gute Prognose zu sein: Der Wert für RMSE liegen deutlich über dem entsprechenden Werten des Modells mit rein numerischen Merkmalen.

Zu c):

In [50]:
```
# Einfache Stichprobeneinteilung, ergänzt
x_train = df.query('sample =="A"')[cat_features]
y_train = df.query('sample =="A"').median_house_value
x_val = df.query('sample =="B"')[cat_features]
y_val = df.query('sample =="B"').median_house_value
```

In [51]:
```
x_train
```

Out[51]:

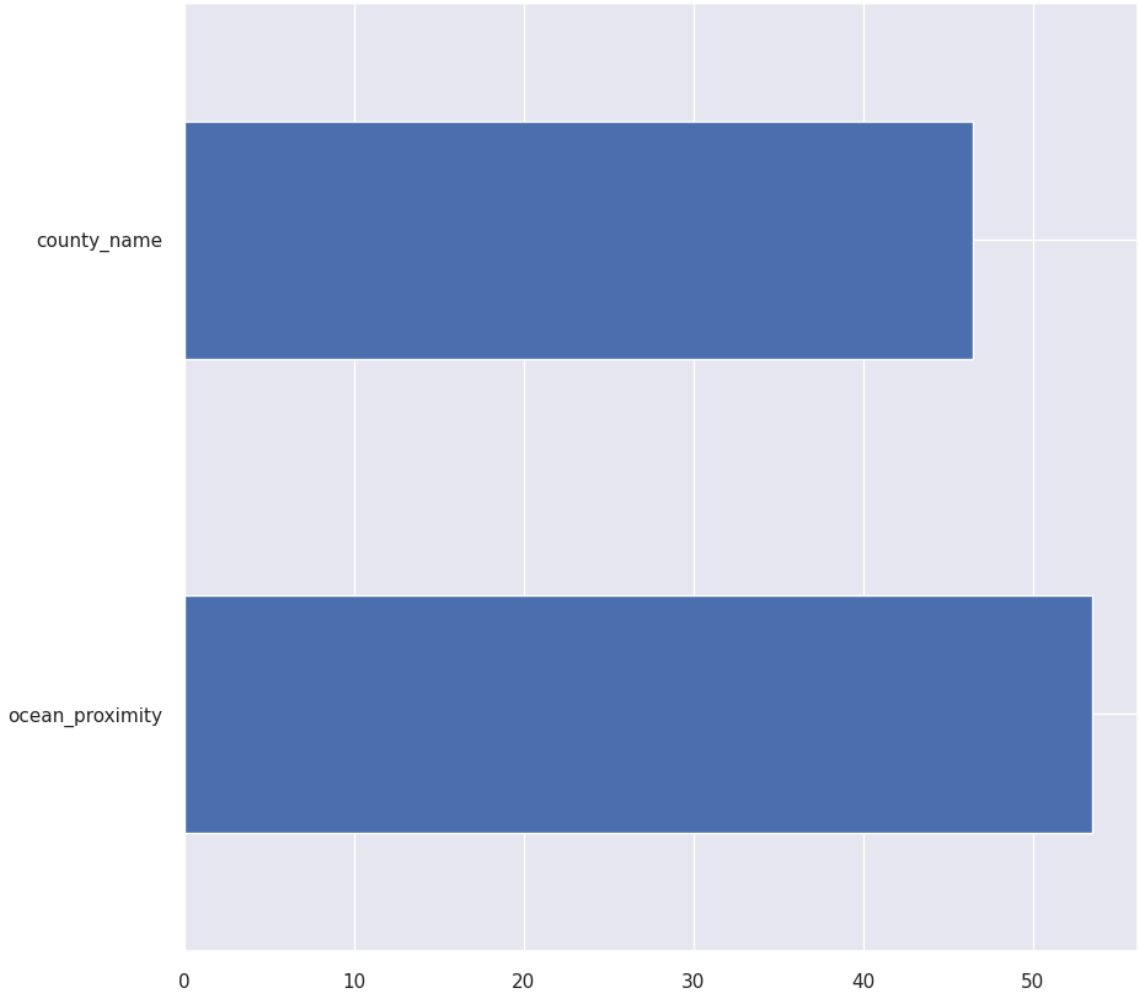
	ocean_proximity	county_name
17848	INLAND	Tuolumne
7517	INLAND	Los Angeles
19997	INLAND	Lake
6558	<1H OCEAN	Los Angeles
5053	<1H OCEAN	Los Angeles
...	...	...
7808	<1H OCEAN	Los Angeles
9786	<1H OCEAN	Los Angeles
7805	<1H OCEAN	Los Angeles
6876	<1H OCEAN	Los Angeles
9784	<1H OCEAN	Los Angeles

14421 rows × 2 columns

In [52]:
```
# Einfaches CatBoost-Standardmodell aufsetzen, fitten und Standardparameter anze
tic = time.time()
model_cb_cat = CatBoostRegressor(eval_metric='RMSE', random_seed=42)
model_cb_cat.fit(x_train,y_train, cat_features=cat_features, verbose=0)
print(model_cb_cat.get_all_params())
print("time (sec):" + "%6.0f" % (time.time() - tic))
```

```
{'nan_mode': 'Min', 'eval_metric': 'RMSE', 'combinations_ctr': ['Borders:CtrBord
erCount=15:CtrBorderType=Uniform:TargetBorderCount=1:TargetBorderType=MinEntrop
y:Prior=0/1:Prior=0.5/1:Prior=1/1', 'Counter:CtrBorderCount=15:CtrBorderType=Uni
form:Prior=0/1'], 'iterations': 1000, 'sampling_frequency': 'PerTree', 'fold_per
mutation_block': 0, 'leaf_estimation_method': 'Newton', 'random_score_type': 'No
rmalWithModelSizeDecrease', 'counter_calc_method': 'SkipTest', 'grow_policy': 'S
ymmetricTree', 'penalties_coefficient': 1, 'boosting_type': 'Plain', 'model_shri
nk_mode': 'Constant', 'feature_border_type': 'GreedyLogSum', 'ctr_leaf_count_lim
it': 18446744073709551615, 'bayesian_matrix_reg': 0.10000000149011612, 'one_hot_
max_size': 2, 'eval_fraction': 0, 'force_unit_auto_pair_weights': False, 'l2_lea
f_reg': 3, 'random_strength': 1, 'rsm': 1, 'boost_from_average': True, 'max_ctr_
complexity': 4, 'model_size_reg': 0.5, 'simple_ctr': ['Borders:CtrBorderCount=1
5:CtrBorderType=Uniform:TargetBorderCount=1:TargetBorderType=MinEntropy:Prior=0/
1:Prior=0.5/1:Prior=1/1', 'Counter:CtrBorderCount=15:CtrBorderType=Uniform:Prior
=0/1'], 'pool_metainfo_options': {'tags': {}}, 'subsample': 0.800000011920929,
'use_best_model': False, 'random_seed': 42, 'depth': 6, 'ctr_target_border_coun
t': 1, 'posterior_sampling': False, 'has_time': False, 'store_all_simple_ctr': F
alse, 'border_count': 254, 'classes_count': 0, 'auto_class_weights': 'None', 'sp
arse_features_conflict_fraction': 0, 'leaf_estimation_backtracking': 'AnyImprove
ment', 'best_model_min_trees': 1, 'model_shrink_rate': 0, 'min_data_in_leaf': 1,
'loss_function': 'RMSE', 'learning_rate': 0.06241700053215027, 'score_function':
'Cosine', 'task_type': 'CPU', 'leaf_estimation_iterations': 1, 'bootstrap_type':
'MVS', 'max_leaves': 64, 'permutation_count': 4}
time (sec):     4
```

In [53]:
```
# CatBoost: Feature-Importance plotten
(pd.Series(model_cb_cat.feature_importances_, index=x_train.columns).nlargest(10
plt.show()
```



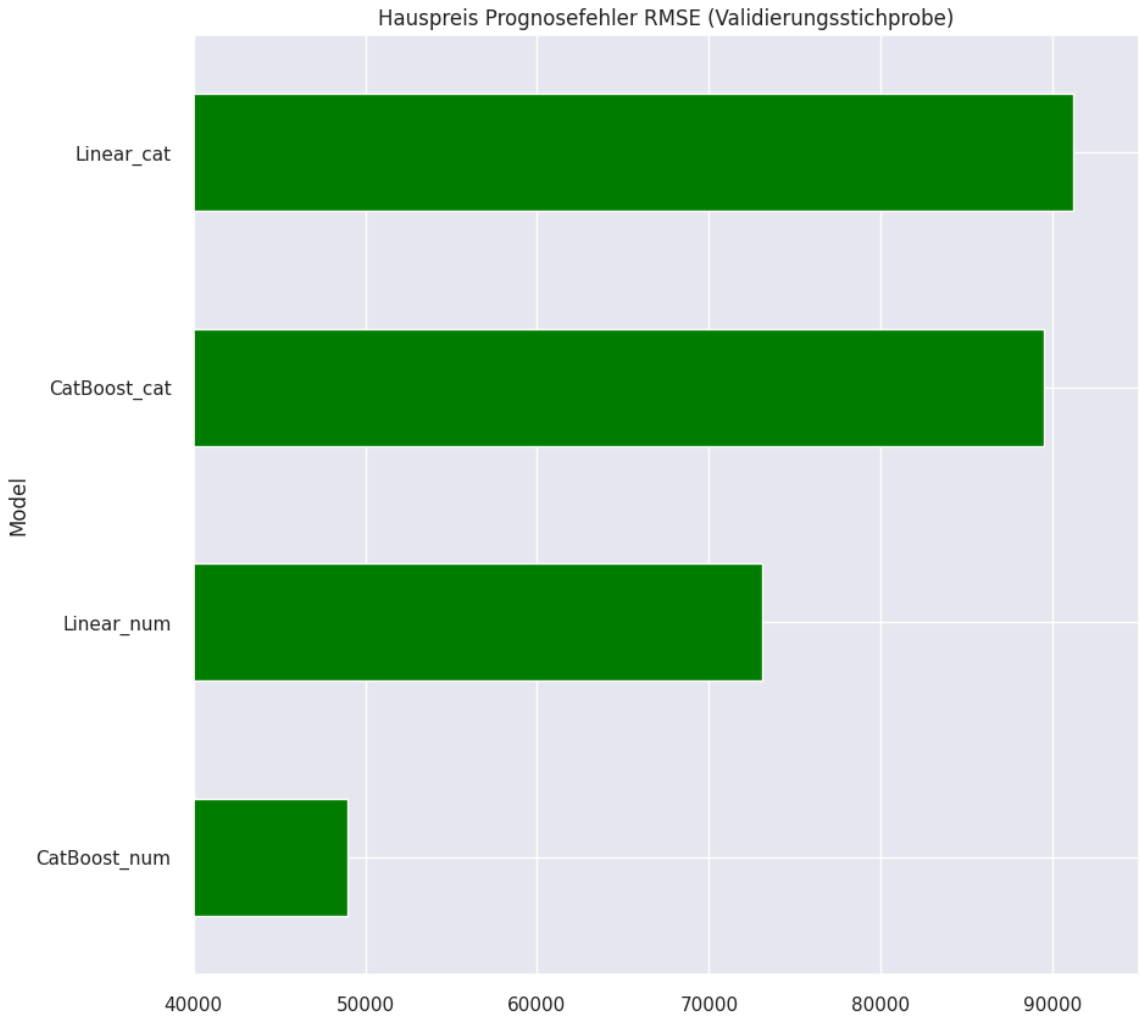
In [54]:
```
# Modelgüte RMSE an Validierungsstichprobe ermitteln
pred_val_cb_cat = model_cb_cat.predict(x_val)
rmse = mean_squared_error(y_val, pred_val_cb_cat, squared=False)
```

In [55]:
```
# Ergebnisse zum Modellvergleich innerhalb von A hinzufuegen und anzeigen
results_a_df.loc['CatBoost_cat'] = [rmse]
results_a_df
```

Out[55]:

Model	RMSE_val
CatBoost_num	48971.008265
Linear_num	73084.952580
Linear_cat	91193.295391
CatBoost_cat	89505.410728

In [56]:
```
plot_rmse(results_a_df)
```



Feature-Importance: Das Merkmal "ocean proximity" ist hier etwas wichtiger als "county".

Modellgüte: Catboost mit kategoriellen Merkmalen ist insgesamt nur wenig besser als das obige lineare Modell. Im Vergleich zu dem Catboost Modell mit numerischen Merkmalen enthalten die beiden verwendeten kategoriellen Merkmale nicht genügend Informationen, um ansatzweise in einen ähnlich guten Bereich zu gelangen.

Aufgabe A-7: Benchmark-Modelle mit numerischen und kategoriellen Features erstellen [Lernziele 3.3/3.4, 4.1 & 6; 7 Punkte]

Nach den bisher betrachteten Modellen, die entweder rein auf den numerischen bzw. kategoriellen Merkmalen bestanden, sollen in diesem Abschnitt kombinierte Modelle betrachtet werden. a) Analog zu den Aufgaben A-4 und A-6 ist ein lineares Modell mit Berücksichtigung sowohl der numerischen Merkmals als auch der kategoriellen Merkmale _ocean_proximity_ und _country_name_ zu erstellen. Die Ergebnisse der Modellierung sind auszugeben und zu kommentieren sowie die Ergebniswerte samt Laufzeit zu speichern. Wie ist dieses Modell im Vergleich zu den beiden anderen (linearen) zu bewerten? b) Analog zu den Aufgaben A-4 und A-6 ist CatBoost (ohne weitere Anpassungen) mit Berücksichtigung sowohl der numerischen Merkmals als auch der kategoriellen Merkmale _ocean_proximity_ und _country_name_ anzuwenden. Ein Ergebnisbericht ist auszugeben und die Ergebniswerte samt Laufzeit zu

speichern. Zusätzlich ist eine graphische Feature-Importance auszugeben und zusammen mit den Ergebniswerten zu kommentieren. Wie ist dieses Modell im Vergleich zu den beiden anderen (Catboost) zu bewerten? c) Wie sind die Laufzeiten und Prognosefehler der in diesem Aufgabe erstellten Modelle im Vergleich und wie sind die Werte und Unterschiede zu bewerten? Basierend auf den Ergebnissen dieses Abschnitts soll ein abschließendes Fazit gegeben werden.

Lösungsvorschlag

Zu a):

In [57]:
```
# Feature-Liste ergänzen #fl: housing_median_age hinzugefügt
num_features =['latitude','longitude','housing_median_age','population','median_
cat_features =['ocean_proximity','county_name']
num_cat_features = num_features + cat_features
```

In [58]:
```
# Einfache Stichprobeneinteilung, ergänzt
train = df.query('sample == "A"')[num_cat_features + ["median_house_value"]]
val = df.query('sample =="B"')[num_cat_features + ["median_house_value"]]
y_val = df.query('sample =="B"').median_house_value
```

In [59]:
```
# Nutzung des R-Interfaces für die (methodisch einfache) Berücksichtung der Kate
# Hinweis: Durch Nutzung von formula wird automatisch ein Intercept berücksichti
tic = time.time()

model_ols_numcat = smf.ols(formula="median_house_value ~ latitude + longitude +
results_ols_numcat = model_ols_numcat.fit()

runtime_ols_numcat = time.time() - tic

print(results_ols_numcat.summary2())
```

```
                                  Results: Ordinary least squares
==============================================================================
=======================
Model:                          OLS                          Adj. R-squared:
0.673
Dependent Variable:             median_house_value           AIC:
361087.2473
Date:                           2024-07-27 20:36             BIC:
361610.0217
No. Observations:               14421                        Log-Likelihood:
-1.8047e+05
Df Model:                       68                           F-statistic:
437.5
Df Residuals:                   14352                        Prob (F-statistic):
0.00
R-squared:                      0.675                        Scale:
4.3626e+09
------------------------------------------------------------------------------
-----------------------
                                     Coef.       Std.Err.     t     P>|t|
[0.025        0.975]
------------------------------------------------------------------------------
-----------------------
Intercept                         -4173887.6150 336727.4493 -12.3954 0.0000 -483
3916.9512 -3513858.2788
C(ocean_proximity)[T.INLAND]         -1349.2482   3202.7441  -0.4213 0.6736    -
7627.0408     4928.5444
C(ocean_proximity)[T.ISLAND]        144568.1703  46807.6023   3.0886 0.0020    5
2819.2180   236317.1225
C(ocean_proximity)[T.NEAR BAY]       21205.4289   3553.8388   5.9669 0.0000    1
4239.4453    28171.4125
C(ocean_proximity)[T.NEAR OCEAN]     21219.4050   2409.1362   8.8079 0.0000    1
6497.1866    25941.6234
C(county_name)[T.Alpine]             69001.5152  38971.7976   1.7705 0.0767    -
7388.2468   145391.2773
C(county_name)[T.Amador]             38125.6794  15584.3369   2.4464 0.0144
7578.3643    68672.9946
C(county_name)[T.Butte]              30620.3715  10337.6551   2.9620 0.0031    1
0357.2308    50883.5121
C(county_name)[T.Calaveras]          22487.0761  14995.4254   1.4996 0.1337    -
6905.8964    51880.0485
C(county_name)[T.Colusa]            -24490.0828  19440.1719  -1.2598 0.2078   -6
2595.3330    13615.1675
C(county_name)[T.Contra Costa]       -6130.6812   4251.5308  -1.4420 0.1493   -1
4464.2313     2202.8689
C(county_name)[T.Del Norte]         -41727.6524  24086.4344  -1.7324 0.0832   -8
8940.1780     5484.8731
C(county_name)[T.El Dorado]          64214.8445   9987.4123   6.4296 0.0000    4
4638.2252    83791.4638
C(county_name)[T.Fresno]              7303.7902   8428.3148   0.8666 0.3862    -
9216.7965    23824.3769
C(county_name)[T.Glenn]             -12834.9417  18297.3093  -0.7015 0.4830   -4
8700.0336    23030.1503
C(county_name)[T.Humboldt]          -73508.5894  14615.1876  -5.0296 0.0000  -10
2156.2466   -44860.9322
C(county_name)[T.Imperial]           91034.4928  24039.3467   3.7869 0.0002    4
3914.2652   138154.7205
C(county_name)[T.Inyo]              110414.7253  19440.4536   5.6796 0.0000    7
2308.9227   148520.5279
C(county_name)[T.Kern]                8691.4505  12581.7016   0.6908 0.4897   -1
5970.3112    33353.2123
C(county_name)[T.Kings]              -4231.9950  11952.6358  -0.3541 0.7233   -2
7660.7066    19196.7166
C(county_name)[T.Lake]              -35407.1477  10439.5318  -3.3916 0.0007   -5
```

```
5869.9797   -14944.3157
C(county_name)[T.Lassen]            60107.4010  19569.0098   3.0716 0.0021    2
1749.6117    98465.1903
C(county_name)[T.Los Angeles]      116546.7564  16143.8441   7.2193 0.0000    8
4902.7348   148190.7781
C(county_name)[T.Madera]            12845.9707  11358.2012   1.1310 0.2581    -
9417.5721    35109.5135
C(county_name)[T.Marin]             75519.6246   6803.0294  11.1009 0.0000    6
2184.8074    88854.4419
C(county_name)[T.Mariposa]          39200.7464  18883.7486   2.0759 0.0379
2186.1576    76215.3351
C(county_name)[T.Mendocino]        -37817.5739  12039.5961  -3.1411 0.0017   -6
1416.7388   -14218.4090
C(county_name)[T.Merced]             3565.1629   8899.8843   0.4006 0.6887   -1
3879.7610    21010.0868
C(county_name)[T.Modoc]             88763.1387  36821.3172   2.4106 0.0159    1
6588.5963   160937.6811
C(county_name)[T.Mono]              92106.1441  20369.8846   4.5217 0.0000    5
2178.5366   132033.7516
C(county_name)[T.Monterey]          32813.0767   7938.8932   4.1332 0.0000    1
7251.8195    48374.3338
C(county_name)[T.Napa]              13327.4131   8523.3272   1.5636 0.1179    -
3379.4103    30034.2365
C(county_name)[T.Nevada]            73359.5316  11609.1420   6.3191 0.0000    5
0604.1123    96114.9510
C(county_name)[T.Orange]           109780.1114  17541.7361   6.2582 0.0000    7
5396.0407   144164.1821
C(county_name)[T.Placer]            64861.1689   9581.8267   6.7692 0.0000    4
6079.5498    83642.7881
C(county_name)[T.Plumas]            49501.4588  17031.5222   2.9065 0.0037    1
6117.4733    82885.4443
C(county_name)[T.Riverside]         92697.0805  18785.9838   4.9344 0.0000    5
5874.1234   129520.0375
C(county_name)[T.Sacramento]        17266.3744   6175.1494   2.7961 0.0052
5162.2831    29370.4656
C(county_name)[T.San Benito]        50178.2838  14409.1255   3.4824 0.0005    2
1934.5348    78422.0328
C(county_name)[T.San Bernardino]    72317.3206  17414.4288   4.1527 0.0000    3
8182.7886   106451.8526
C(county_name)[T.San Diego]         88894.8638  20819.8075   4.2697 0.0000    4
8085.3494   129704.3782
C(county_name)[T.San Francisco]    102644.7187   4320.5167  23.7575 0.0000    9
4175.9474   111113.4900
C(county_name)[T.San Joaquin]        1527.0560   5950.4092   0.2566 0.7975   -1
0136.5154    13190.6273
C(county_name)[T.San Luis Obispo]   44838.1571  12198.4460   3.6757 0.0002    2
0927.6257    68748.6884
C(county_name)[T.San Mateo]         72045.8598   5458.0109  13.2000 0.0000    6
1347.4528    82744.2669
C(county_name)[T.Santa Barbara]     67927.0403  13252.9378   5.1254 0.0000    4
1949.5687    93904.5118
C(county_name)[T.Santa Clara]       56715.5296   4703.7863  12.0574 0.0000    4
7495.5003    65935.5589
C(county_name)[T.Santa Cruz]        37492.0943   7501.7932   4.9978 0.0000    2
2787.6097    52196.5789
C(county_name)[T.Shasta]            18505.1757  13204.1774   1.4015 0.1611    -
7376.7191    44387.0705
C(county_name)[T.Sierra]            50151.2083  34370.4650   1.4591 0.1445   -1
7219.3469   117521.7634
C(county_name)[T.Siskiyou]          23303.0900  19730.6996   1.1811 0.2376   -1
5371.6322    61977.8123
C(county_name)[T.Solano]           -32024.2464   6816.9189  -4.6978 0.0000   -4
5386.2888   -18662.2039
C(county_name)[T.Sonoma]            15580.5319   6590.6476   2.3640 0.0181
```

```
2662.0105    28499.0533
C(county_name)[T.Stanislaus]         24160.8346   6650.8351   3.6328 0.0003    1
1124.3379    37197.3314
C(county_name)[T.Sutter]              1900.4874  12278.3958   0.1548 0.8770   -2
2166.7558    25967.7306
C(county_name)[T.Tehama]            -10089.3858  15445.2510  -0.6532 0.5136   -4
0364.0747    20185.3031
C(county_name)[T.Trinity]           -31695.9104  18450.3921  -1.7179 0.0858   -6
7861.0645     4469.2436
C(county_name)[T.Tulare]             24802.4275  10646.3129   2.3297 0.0198
3934.2777    45670.5773
C(county_name)[T.Tuolumne]           55548.8427  11681.4167   4.7553 0.0000    3
2651.7556    78445.9297
C(county_name)[T.Ventura]            79345.0467  15035.4811   5.2772 0.0000    4
9873.5598   108816.5337
C(county_name)[T.Yolo]               27233.0280  10386.9763   2.6218 0.0088
6873.2115    47592.8446
C(county_name)[T.Yuba]               11225.3095  12354.3599   0.9086 0.3636   -1
2990.8332    35441.4522
latitude                            -24356.8020   3874.0364  -6.2872 0.0000   -3
1950.4141   -16763.1898
longitude                           -41627.3288   2791.8620 -14.9102 0.0000   -4
7099.7392   -36154.9184
population                             -3.2424      0.5542  -5.8504 0.0000
-4.3287       -2.1560
housing_median_age                    225.5188     56.0818   4.0212 0.0001
115.5913      335.4464
median_income                       37338.5477    433.8321  86.0668 0.0000    3
6488.1807    38188.9148
rooms_per_house                      2310.8994    273.7619   8.4413 0.0000
1774.2907     2847.5080
bedrooms_ratio                     181247.1781  13821.4255  13.1135 0.0000   15
4155.3972   208338.9590
------------------------------------------------------------------------------
-----------------------
Omnibus:                        3136.944                 Durbin-Watson:
1.896
Prob(Omnibus):                  0.000                    Jarque-Bera (JB):
9736.852
Skew:                           1.115                    Prob(JB):
0.000
Kurtosis:                       6.351                    Condition No.:
1100485
==============================================================================
=======================
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.1e+06. This might indicate                t
hat there are strong
multicollinearity or other numerical                problems.
```

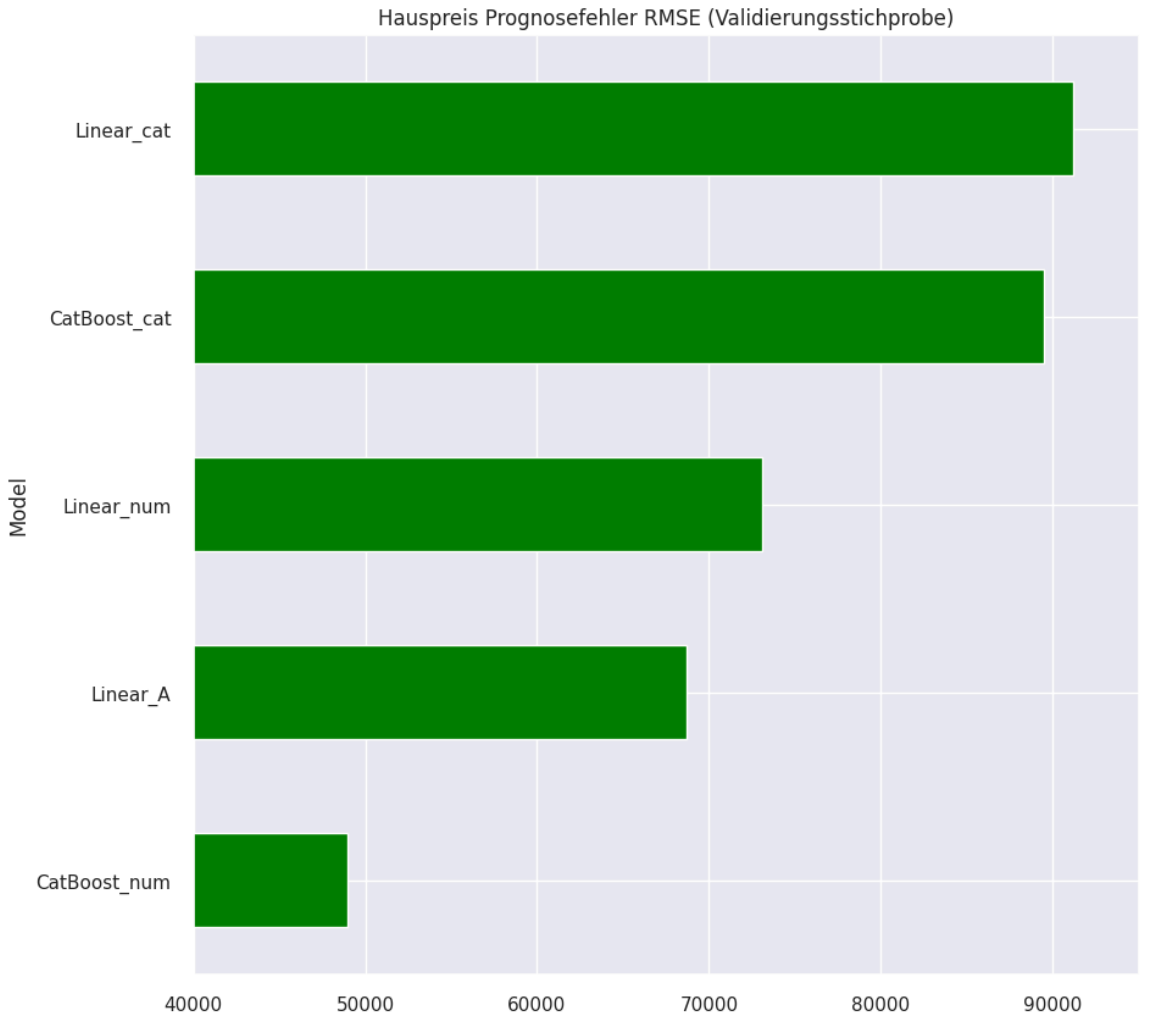
In [60]:
```
# Modelgüte RMSE an Validierungsstichprobe ermitteln
pred_val_ols_numcat = results_ols_numcat.predict(val)
rmse_val_ols_numcat = mean_squared_error(y_val, pred_val_ols_numcat , squared=Fa
```

In [61]:
```
# Ergebnisse zum Modellvergleich innerhalb von A hinzufuegen und anzeigen
results_a_df.loc['Linear_A'] = [rmse_val_ols_numcat]
results_a_df
```

Out[61]:

Model	RMSE_val
CatBoost_num	48971.008265
Linear_num	73084.952580
CatBoost_cat	89505.410728
Linear_cat	91193.295391
Linear_A	68709.117110

In [62]:
```
plot_rmse(results_a_df)
```



Ausgehend vom linearen Modell mit rein numerischen Merkmalen konnte der Wert für RMSE um ca. 6 % (von 73.085 auf 68.709) gesenkt werden. Hier zeigt sich, dass bei dem linearen Modell die zusätzlichen Informationen der kategoriellen Variablen zu einer (allerdings kleinen) Verbesserung der Prognose (auf Basis von RMSE) führen. Ein Grund hierfür könnte sein, dass die in *longitude* und *latitude* codierten Informationen aufgrund des nichtlinearen Verlaufs der Küstenlinie von dem linearen Modell nur schwer zu erfassen sind, und daher zusätzliche Informationen über die Lage (die in den kategoriellen Merkmalen codiert sind) zu der Verbesserung führen.

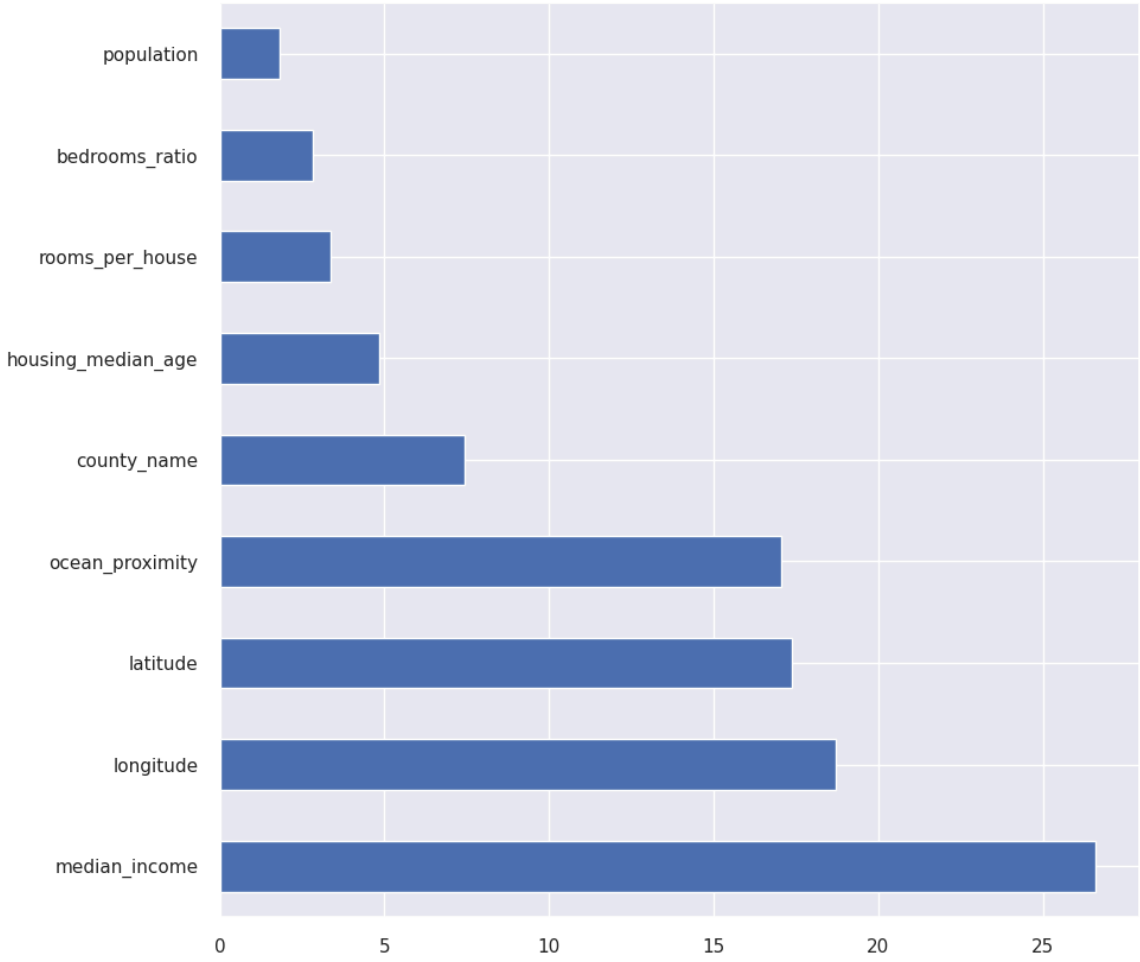
Zu b):

In [63]:
```
# Einfache Stichprobeneinteilung, ergänzt
x_train = df.query('sample =="A"')[num_cat_features]
y_train = df.query('sample =="A"').median_house_value
x_val = df.query('sample =="B"')[num_cat_features]
y_val = df.query('sample =="B"').median_house_value
```

In [64]:
```
# Einfaches CatBoost-Modell mit nominalen Merkmalen aufsetzen und fitten
tic = time.time()

model_cb_numcat = CatBoostRegressor(eval_metric='RMSE', random_seed=42)
model_cb_numcat.fit(x_train,y_train, cat_features=cat_features, verbose=0)

runtime_cb_numcat = time.time() - tic
print(model_cb_num.get_all_params())
print("time (sec):" + "%6.0f" % (runtime_cb_numcat))
```

```
{'nan_mode': 'Min', 'eval_metric': 'RMSE', 'iterations': 1000, 'sampling_frequen
cy': 'PerTree', 'leaf_estimation_method': 'Newton', 'random_score_type': 'Normal
WithModelSizeDecrease', 'grow_policy': 'SymmetricTree', 'penalties_coefficient':
1, 'boosting_type': 'Plain', 'model_shrink_mode': 'Constant', 'feature_border_ty
pe': 'GreedyLogSum', 'bayesian_matrix_reg': 0.10000000149011612, 'eval_fractio
n': 0, 'force_unit_auto_pair_weights': False, 'l2_leaf_reg': 3, 'random_strengt
h': 1, 'rsm': 1, 'boost_from_average': True, 'model_size_reg': 0.5, 'pool_metain
fo_options': {'tags': {}}, 'subsample': 0.800000011920929, 'use_best_model': Fal
se, 'random_seed': 42, 'depth': 6, 'posterior_sampling': False, 'border_count':
254, 'classes_count': 0, 'auto_class_weights': 'None', 'sparse_features_conflict
_fraction': 0, 'leaf_estimation_backtracking': 'AnyImprovement', 'best_model_min
_trees': 1, 'model_shrink_rate': 0, 'min_data_in_leaf': 1, 'loss_function': 'RMS
E', 'learning_rate': 0.06241700053215027, 'score_function': 'Cosine', 'task_typ
e': 'CPU', 'leaf_estimation_iterations': 1, 'bootstrap_type': 'MVS', 'max_leave
s': 64}
time (sec):     7
```

In [65]:
```
# CatBoost: Feature-Importance plotten
(pd.Series(model_cb_numcat.feature_importances_, index=x_train.columns).nlargest
plt.show()
```



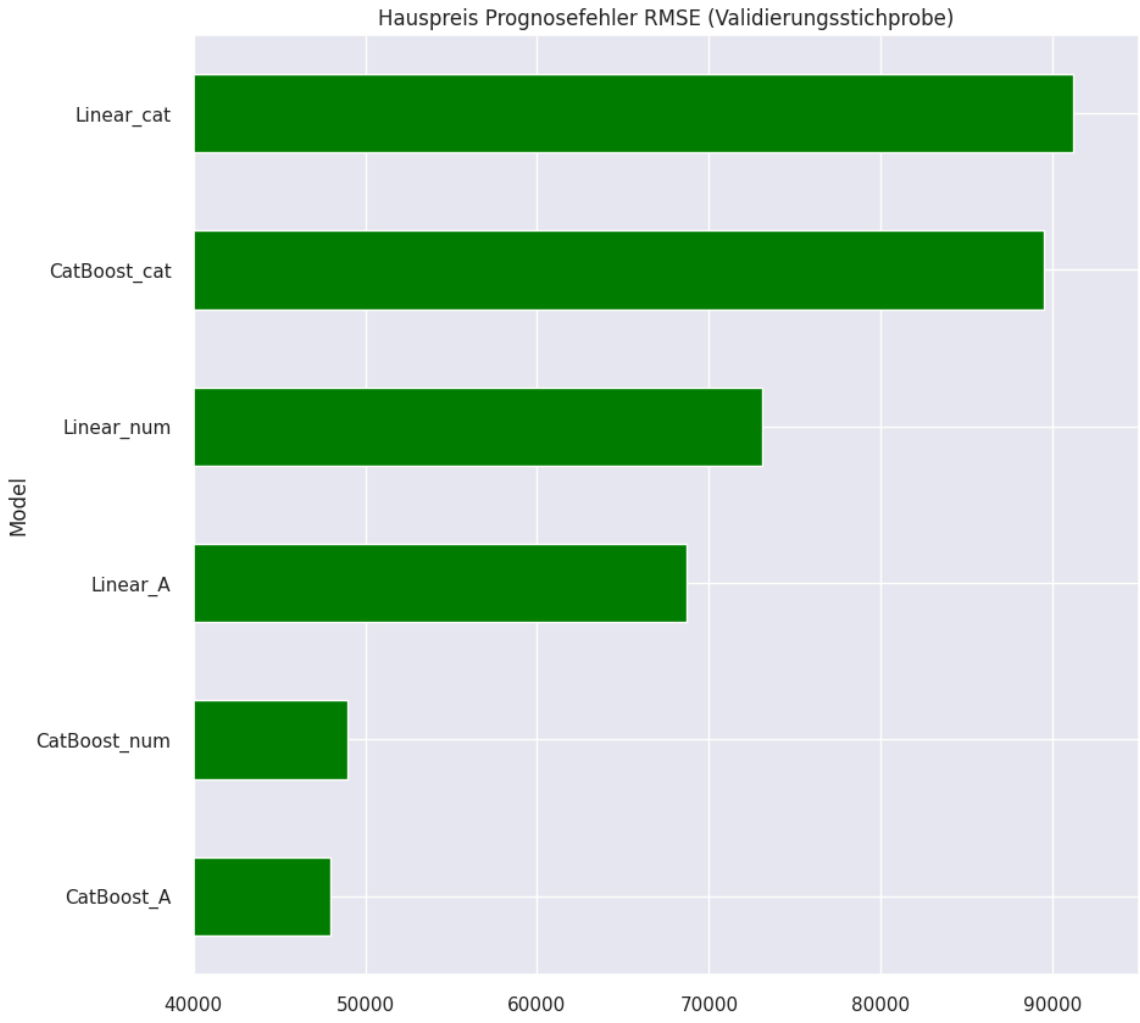
In [66]:
```
# Modelgüte RMSE an Validierungsstichprobe ermitteln
pred_val_cb_numcat = model_cb_numcat.predict(x_val)
rmse_val_cb_numcat = mean_squared_error(y_val, pred_val_cb_numcat, squared=False
```

In [67]:
```
# Ergebnisse zum Modellvergleich innerhalb von A hinzufuegen und anzeigen
results_a_df.loc['CatBoost_A'] = [rmse_val_cb_numcat]
results_a_df
```

Out[67]:

Model	RMSE_val
CatBoost_num	48971.008265
Linear_A	68709.117110
Linear_num	73084.952580
CatBoost_cat	89505.410728
Linear_cat	91193.295391
CatBoost_A	48006.829301

In [68]:
```
plot_rmse(results_a_df)
```



Für das Catboost-Modell sind, im Vergleich zum entsprechenden Modell mit numerischen Merkmalen, Veränderungen erkennbar. Im Modell mit nur numerischen Variablen waren ebenfalls die Merkmale _median_income_, *longitude* und *latitude* führend. Hierzu kommt in diesem Modell _ocean_proximity_, die eine vergleichbar hohe Importance aufweist. Obwohl sich in der Feature-Importance die kategoriellen Merkmale direkt hinter die starken Features _median_income_, *longitude* und *latitude* platziert haben, ist bei RMSE keine weitere (deutliche) Verringerung zu erkennen.

Zu c):

In [69]:
```
# Ergebnisse zum Modellvergleich hinzufügen und anzeigen
results_df.loc['Linear_A'] = [runtime_ols_numcat,rmse_val_ols_numcat,'']
results_df.loc['CatBoost_A'] = [runtime_cb_numcat,rmse_val_cb_numcat,'']
results_df
```

Out[69]:

Model	time_train	RMSE_val	RMSE_test
Linear_A	0.167730	68709.117110	
CatBoost_A	6.781105	48006.829301	

In diesem Abschnitt wurden (einfache) lineare Modelle sowie entscheidungsbaumbasierte Ensemble-Verfahren (Catboost) verwendet. Bei den lineren Modellen hat sich gezeigt, dass nichtlineare Zusammenhänge (longitude/latitude) wie zu

erwarten nicht durch das Modell erkannt werden. Daher bleiben diese (gemessan an RMSE) weit hinter den Baumverfahren zurück. Das beste Modell (_Catboost_numcat_) stellt mit einem RMSE-Wert von ca. 48.000 bereits eine sehr gute Benchmark auf, die vermutlich nicht einfach zu schlagen sein sind. Dabei ist zunächst keine weitere Optimierung des Modells (HP-Tuning etc.) erfolgt!

Laufzeitmessung:

In [70]:
```
# Zeit Notebook: Ende Teil A, Start Teil B
startzeit_teil_b = time.time()
laufzeit_teil_a = startzeit_teil_b - startzeit_teil_a

# Laufzeit anzeigen
runtime_abc_df.loc['A'] = [laufzeit_teil_a]
runtime_abc_df
```

Out[70]:

Teil	Laufzeit
A	64.397617

Teil B: Prognosemodelle optimieren und Overfitting

Aufgabe B-1: Encoding, Skalierung und Vergleich [Lernziele 3.3/3.4, 4.1 & 6; 11 Punkte]

a) Die numerischen Daten sind geeignet zu skalieren. Das ausgewählte Skalierungsverfahren ist kurz zu beschreiben und die Auswahl kurz zu begründen. Es sind sieben zufällige Zeilen der Daten je vor und nach Skalierung anzuzeigen (die gleichen Datensätze) und das Ergebnis ist zu bewerten. b) Die kategoriellen Daten sind geeignet in numerische Werte zu überführen. Das ausgewählte Encoding ist kurz zu beschreiben und die Auswahl zu begründen. Für die sieben Zeilen aus Teilaufgabe a) sind die bearbeiteten Datensätze anzuzeigen und das Ergebnis ist zu bewerten. Es ist darauf einzugehen, wie viele Merkmale ggf. neu hinzugekommen sind und ob die Daten zu einem überbestimmten linearen Modell führen würden. c) Im Anschluss ist der nun skalierte und encodierte Datensatz in Trainingsdaten (sample = A) und Validierungsdaten (sample = B) aufzusplitten und die Zielvariable von den beschreibenden Variablen zu trennen. d) An die so aufbereiteten Daten ist erneut ein CatBoost-Modell anzupassen. Dabei ist eine Zeitmessung durchzuführen und die Feature Importance auszugeben. Die Güte des Modells (RMSE) ist zunächst für den Trainingsdatensatz, im Anschluss für den Validierungsdatensatz zu ermitteln und die Ergebnisse sind zu vergleichen und zu interpretieren. Abschließend ist ein Vergleich mit dem CatBoost-Modell aus Aufgabe A-7 vorzunehmen und auf Laufzeit, Modellgüte (Validierung) und ggf.

Auffälligkeiten im Vergleich der Feature Importance einzugegen, mögliche Ursachen zu ergründen und eine Bewertung vorzunehmen.

Lösungsvorschlag

Zu a):

Zum ausgewählten Skalierungsverfahren:

Da in Aufgabe B-2 ein One-Hot-Encoding der kategoriellen Merkmale auf den jeweiligen Wertebereich 0/1 durchgeführt wird, wird hier für die in "num_features" gelisteten numerischen Merkmale eine Min-Max-Skalierung ebenfalls auf den Wertebereich zwischen 0 und 1 durchgeführt.

In [71]:
```
# Zufallsauswahl an Datensätzen vor Skalierung
df.sample(7, random_state=seed)
```

Out[71]:

	latitude	longitude	housing_median_age	population	median_income	median_house_va
1117	32.85	-117.26	30	978	8.2374	500
4913	33.92	-118.22	23	1856	2.1366	100
18149	38.28	-121.29	11	793	4.8073	156
14244	37.42	-122.16	34	2571	11.0492	500
7597	34.07	-118.37	52	468	3.4286	474
15727	37.74	-122.16	52	355	4.1458	143
8288	34.10	-118.29	39	1165	2.9417	254

In [72]:
```
# Skalierung der numerischen Daten: Min-Max-Scaling
scaler = MinMaxScaler()
df_scaled = df.copy(deep=True)
df_scaled[num_features] = scaler.fit_transform(df[num_features])

# Kennzahlen anzeigen
df_scaled.describe()
```

Out[72]:

	latitude	longitude	housing_median_age	population	median_income	median
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	
mean	0.328572	0.476125	0.541951	0.039869	0.232464	2
std	0.226988	0.199555	0.246776	0.031740	0.131020	1
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.147715	0.253984	0.333333	0.021974	0.142308	1
50%	0.182784	0.583665	0.549020	0.032596	0.209301	1
75%	0.549416	0.631474	0.705882	0.048264	0.292641	2
max	1.000000	1.000000	1.000000	1.000000	1.000000	5

In [73]:
```
# Die gleichen Datensätze wie oben nach Skalierung anzeigen
df_scaled.sample(7, random_state=seed)
```

Out[73]:

	latitude	longitude	housing_median_age	population	median_income	median_house_va
1117	0.032944	0.706175	0.568627	0.027327	0.533613	500
4913	0.146652	0.610558	0.431373	0.051935	0.112874	100
18149	0.609989	0.304781	0.196078	0.022142	0.297058	156
14244	0.518597	0.218127	0.647059	0.071975	0.727528	500
7597	0.162593	0.595618	1.000000	0.013033	0.201977	474
15727	0.552604	0.218127	1.000000	0.009866	0.251438	143
8288	0.165781	0.603586	0.745098	0.032568	0.168398	254

Bewertung: Die Werte der numerischen Daten liegen nun erwartungsgemäß zwischen 0 und 1. Zu beachten: Die Zielgröße median_house_value wurde hier nicht skaliert!

Zu b):

Zum ausgewählten Encoding:

Da die kategoriellen Merkmale des Datensatzes keine implizite Ordnung aufweisen ist ein Label Encoding, welches eine Ordnung unterstellt, hier nicht ratsam. Es wird ein One-Hot-Encoding für die in 'cat_features' gelistetender beiden kategoriellen Merkmale 'ocean_proximity' und 'county_name' auf den Wertebereich 0/1 durchgeführt. Durch drop_first = True iwrd die dabei entstehende jeweils erste Dummy-Variable sowie das jeweilige Ursprungsmerkmal gelöscht und Überbestimmtheit verhindert (siehe Aufgabenstellung). Es kommen dadurch 58 + 5 - 2= 59 Merkmale hinzu.

In [74]:
```
# Anzahl Datensätze und Merkmale vor Encoding
df_scaled.shape
```

Out[74]:
```
(20640, 11)
```

In [75]:
```
# Endoding der kategoriellen Daten
df_scaled_encoded = pd.get_dummies(df_scaled,columns = cat_features, dtype='int'
df_scaled_encoded.shape
```

Out[75]:
```
(20640, 70)
```

In [76]:
```
df_scaled_encoded.sample(7, random_state=seed)
```

Out[76]:

	latitude	longitude	housing_median_age	population	median_income	median_house_va
1117	0.032944	0.706175	0.568627	0.027327	0.533613	500
4913	0.146652	0.610558	0.431373	0.051935	0.112874	100
18149	0.609989	0.304781	0.196078	0.022142	0.297058	156
14244	0.518597	0.218127	0.647059	0.071975	0.727528	500
7597	0.162593	0.595618	1.000000	0.013033	0.201977	474
15727	0.552604	0.218127	1.000000	0.009866	0.251438	143
8288	0.165781	0.603586	0.745098	0.032568	0.168398	254

7 rows × 70 columns

Einschätzung: Die kategoriellen Merkmale ocean_proximity sowie county_code sind wie erwartet in Nullen und Einsen codiert.

Zu c):

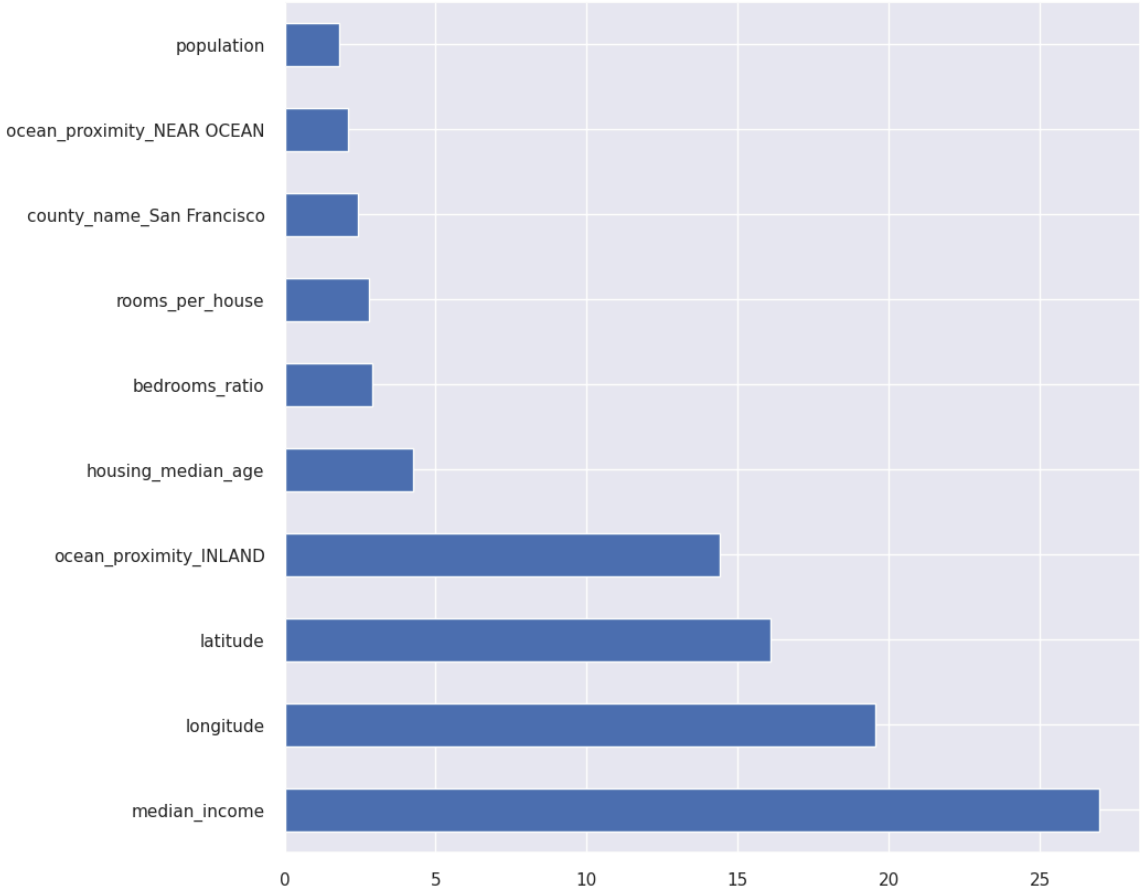
In [77]:
```
# Einfache Stichprobeneinteilung des skalierten encodeten Datensatzes
xse_train = df_scaled_encoded.query('sample =="A"').drop(['median_house_value',
yse_train = df_scaled_encoded.query('sample =="A"').median_house_value
xse_val = df_scaled_encoded.query('sample =="B"').drop(['median_house_value', 's
yse_val = df_scaled_encoded.query('sample =="B"').median_house_value
```

Zu d):

In [78]:
```
# CatBoost-Modell
tic = time.time()

model_cb_b = CatBoostRegressor(eval_metric='RMSE', random_seed=42)
model_cb_b.fit(xse_train, yse_train, verbose=0)

runtime_cb_b = time.time() - tic
print("time (sec):" + "%6.0f" % (runtime_cb_b))
```

```
time (sec):     3
```

In [79]:
```
# CatBoost: Feature-Importance plotten
(pd.Series(model_cb_b.feature_importances_, index=xse_train.columns).nlargest(10
plt.show()
```



In [80]:
```
# Modelgüte an Trainingsstichprobe ermitteln
pred_train_cb_b = model_cb_b.predict(xse_train)
rmse = mean_squared_error(yse_train, pred_train_cb_b , squared=False)
rmse
```

Out[80]:
```
36883.23782490357
```

In [81]:
```
# Modelgüte an Validierungsstichprobe ermitteln
pred_val_cb_b = model_cb_b.predict(xse_val)
rmse_val = mean_squared_error(yse_val, pred_val_cb_b, squared=False)
rmse_val
```

Out[81]:
```
48783.75757593899
```

In [82]:
```
# Ergebnisse zum Modellvergleich hinzufuegen
results_df.loc['CatBoost_B_OHE'] = [runtime_cb_b,rmse_val,'']
results_df
```

Out[82]:

Model	time_train	RMSE_val	RMSE_test
Linear_A	0.167730	68709.117110	
CatBoost_A	6.781105	48006.829301	
CatBoost_B_OHE	3.225582	48783.757576	

Bewertung:

Die Güte des hier gefitteten Modells ist auf den Trainingsdaten deutlich besser als auf den Validierungsdaten. Dies deutet auf ein erhebliches Overfitting hin. Im direkten

Vergleich mit "CatBoost_A" (mit Ursprungsdaten) wird das Modell auf den skalierten, enkodierten Daten deutlich schneller berechnet, ist allerdings von etwas geringerer Güte.

Beim Vergleich der Feature Importance ist nun die Bedeutung der wichtigsten einzelnen Kategorien von "ocean_proximity und "county_name" zu erkennen, allerdings kann die gesamte Bedeutung der beiden Merkmale so nicht mehr erkannt werden.

Da die Skalierung als Ursache bei entscheidungsbaumbasierten Verfahren ausscheidet, ist offenbar CatBoost in der Lage, die kategoriellen Merkmale intern zielgerichteter zu verarbeiten (z.B. Zusammenfassungen) als via OHE, allerdings auf Kosten eines höheren Rechenaufwandes.

Insgesamt ist erkennbar, dass CatBoost_A bereits ohne weitere Datenaufbereitung und ohne Hyperparametertuning eine nicht einfach zu schlagende Benchmark für die Modelle in B darstellt.

Aufgabe B-2: Under-/Overfitting am Beispiel von Neuronalen Netzen [Lernziele 3.2, 3.3/3.4, 4.1, 4.3 & 6; 18 Punkte]

a) Es ist ein Teildatensatz aus dem oben erstellten encodierten und skalierten Datensatz zu erstellen, der nur die ersten 100 Zeilen des Datensatzes enthält. Damit sind 3 einfache neuronale Netze unterschiedlicher Größe mit Keras/Tensorflow sequenziell zu definieren, zu kompilieren, anzuzeigen (via "summary") und jeweils über eine hohe Anzahl von Epochen (> 2000) zu trainieren:

→ klein: 1.000 bis 10.000 Neuronen

→ mittel: 50.000 bis 100.000 Neuronen

→ groß: mindestens eine Million Neuronen

Die history der einzelnen Modelle ist in einem (einzigen) Diagramm zu plotten. Dabei soll der Verlauf des RMSE auf den Trainingsdaten sowie der Verlauf des RMSE auf der internen Modellvalidierung (Split 0.2) dargestellt werden. Die Ergebnisse sind zu beschreiben und zu bewerten. b) Es soll ein Neuronales Netz für den kompletten Trainingsdatensatz (sample = A) erstellt werden. Hierbei sind bei der Architektur des Netzes ggf. Erkenntnisse aus Teilaufgabe 1 miteinzubeziehen und die vorgenommenen Änderungen an der Architektur des Netzes zu beschreiben. Die Größe und Architektur des Neuronalen Netzes muss ausreichend groß sein, um die Komplexität des Datensatzes abbilden zu können. Overfitting soll durch geeignete Regularisierung verhindert werden. Die Daten sind zu fitten und die Lernkurve aus (interner) Trainings- und Modellvalidierung ist zu plotten. Im Anschluss ist das Modell mit den "externen" Validierungsdaten (sample = B) zu validieren. Falls die Prognosegüte dieses hochparametrigen Modells schlechter als die des einfachen linearen Modells aus Aufgabe A-7 ist, sind die Ursachen zu suchen und zu beheben. Dieses Ergebnis (RMSE) ist samt Laufzeit anzuzeigen, dem Gesamtvergleich hinzuzufügen und zu bewerten.

Lösungsvorschlag

Zu a):

In [83]:
```
tic = time.time()
```

In [84]:
```
# verkleinerter Datensatz
df_try = df_scaled_encoded.query('sample =="A"').head(100)
df_try.to_csv('NNtry.csv')
xse_try = df_try.drop(['median_house_value', 'sample'],axis=1)
yse_try = df_try.median_house_value
```

In [85]:
```
N_TRAIN = xse_try.shape[0]
N_FEATURES = xse_try.shape[1]
BATCH_SIZE = 50
STEPS_PER_EPOCH = N_TRAIN//BATCH_SIZE
```

In [86]:
```
# Code angelehnt an https://www.tensorflow.org/tutorials/keras/overfit_and_under
def compile_and_fit(model, name, X, y, max_epochs= 5000):
  model.compile(optimizer=Adam(),#(lr_schedule),
                loss='mae',
                metrics=['mae', tf.keras.metrics.RootMeanSquaredError()])

  model.summary()

  history = model.fit(
      X,
      y,
      steps_per_epoch = STEPS_PER_EPOCH,
      epochs = max_epochs,
      validation_split = 0.2,
      verbose=False)
  return history
```

In [87]:
```
size_histories = {}
```

In [88]:
```
activation = 'relu'
```

In [89]:
```
small_model = tf.keras.Sequential([
    layers.Dense(64, activation=activation, input_shape=(N_FEATURES,)),
    layers.Dense(32, activation=activation),
    layers.Dense(1)
])
```

In [90]:
```
size_histories['Small'] = compile_and_fit(small_model, 'sizes/Small', xse_try, y
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                4416

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 6529 (25.50 KB)
Trainable params: 6529 (25.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [91]:
```
medium_model = tf.keras.Sequential([
    layers.Dense(256, activation=activation, input_shape=(N_FEATURES,)),
    layers.Dense(128, activation=activation, input_shape=(N_FEATURES,)),
    layers.Dense(64, activation=activation),
    layers.Dense(1)
])
```

In [92]:
```
size_histories['Medium']  = compile_and_fit(medium_model, "sizes/Medium", xse_tr
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 256)               17664

 dense_4 (Dense)             (None, 128)               32896

 dense_5 (Dense)             (None, 64)                8256

 dense_6 (Dense)             (None, 1)                 65

=================================================================
Total params: 58881 (230.00 KB)
Trainable params: 58881 (230.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

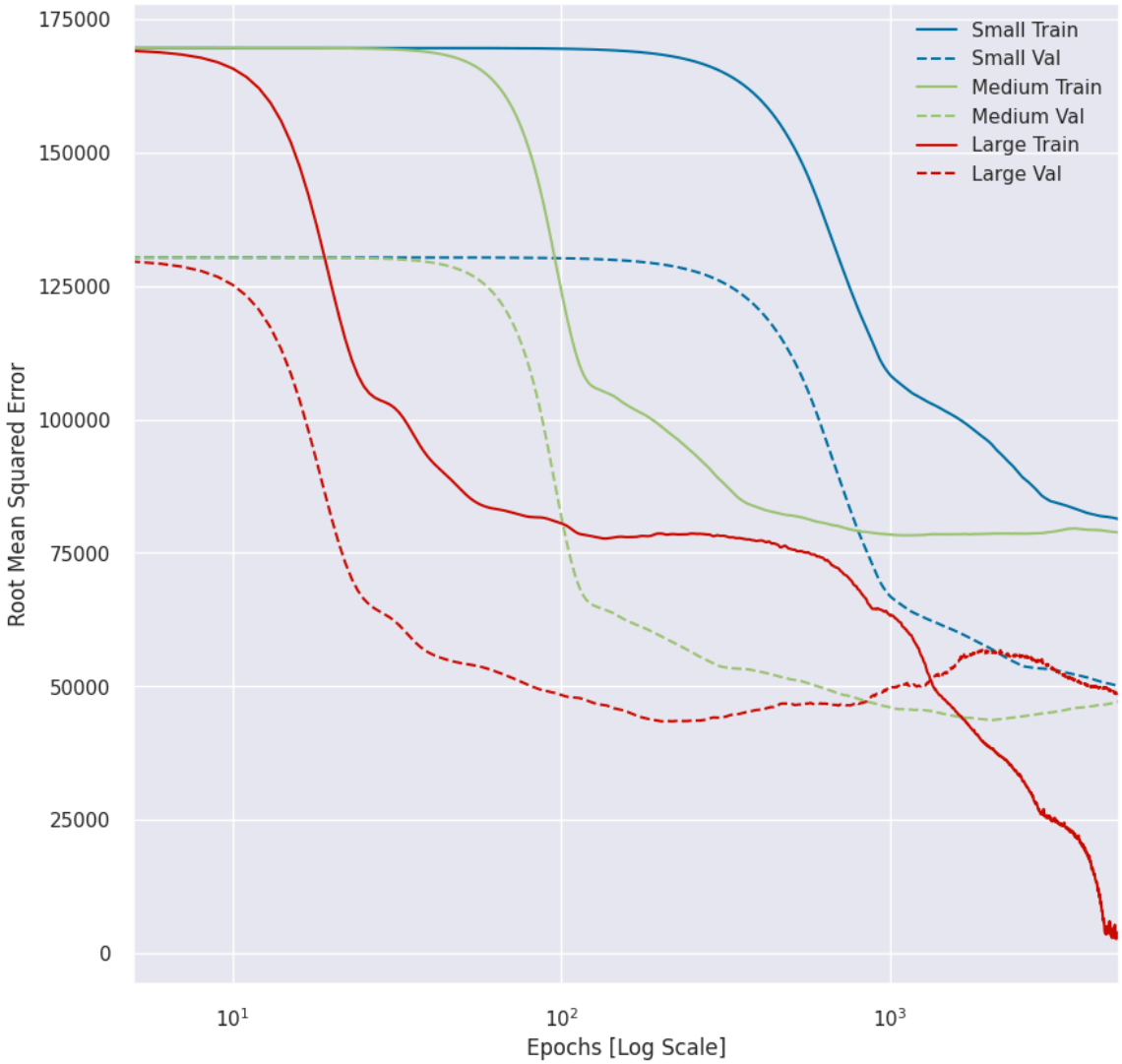
In [93]:
```
large_model = tf.keras.Sequential([
    layers.Dense(2048, activation=activation, input_shape=(N_FEATURES,)),
    layers.Dense(1024, activation=activation),
    layers.Dense(512, activation=activation),
    layers.Dense(256, activation=activation),
    layers.Dense(1)
])
```

In [94]:
```
size_histories['large'] = compile_and_fit(large_model, "sizes/large", xse_try, y
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 2048)              141312

 dense_8 (Dense)             (None, 1024)              2098176

 dense_9 (Dense)             (None, 512)               524800

 dense_10 (Dense)            (None, 256)               131328

 dense_11 (Dense)            (None, 1)                 257

=================================================================
Total params: 2895873 (11.05 MB)
Trainable params: 2895873 (11.05 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [95]:
```
plotter = tfdocs.plots.HistoryPlotter(metric = 'root_mean_squared_error', smooth
plotter.plot(size_histories)
a = plt.xscale('log')
plt.xlim([5, max(plt.xlim())])
plt.xlabel("Epochs [Log Scale]")
```

Out[95]:
```
Text(0.5, 0, 'Epochs [Log Scale]')
```



In [96]:
```
print("Dauer NNs (sec):" + "%6.0f" % (time.time() - tic))
```

```
Dauer NNs (sec):   508
```

Das kleine Netz ist zu klein, um die Zusammenhänge abbilden zu können und lernt nur ausgesprochen langsam. Das gilt sowohl für die Trainingsdaten als auch die interne Validierung. Das mittelgroße Netz zeigt ein Verhalten, wie es zu erwarten ist - der Fehler nimmt sowohl beim Training als auch der Validierung ab. Man kann beobachten, dass das große Netz am Ende des Trainings ins Overfitting läuft. Das bedeutet, dass das Netz die Werte nicht mehr ausreichend abstrahiert, sondern auswendig lernt, darum wird der Trainingsverlauf besser während die interne Validierung bei zunehmender Epochenanzahl immer schlechter wird.

Zu b):

In [97]:
```
# die Daten werden durchmischt
rand_df_scaled_encoded = df_scaled_encoded.sample(frac=1, random_state = seed)
```

In [98]:
```
# Einfache Stichprobeneinteilung des skalierten encodeten Datensatzes
xse_train = rand_df_scaled_encoded.query('sample =="A"').drop(['median_house_val
yse_train = rand_df_scaled_encoded.query('sample =="A"').median_house_value
```

In [99]:
```
N_TRAIN = xse_train.shape[0]
N_FEATURES = xse_train.shape[1]
BATCH_SIZE = 50
STEPS_PER_EPOCH = N_TRAIN//BATCH_SIZE
```

In [100…
```
layer_one = 512
layer_two = 256
layer_three = 128
```

In [101…
```
def create_model_incl_dropout():
    model = tf.keras.Sequential([
        layers.Dense(layer_one, activation = activation, kernel_regularizer=tf.k
        layers.Dropout(0.2),
        layers.Dense(layer_two, activation = activation, kernel_regularizer=tf.k
        layers.Dropout(0.2),
        layers.Dense(layer_three, activation = activation, kernel_regularizer=tf
        layers.Dropout(0.2),
        layers.Dense(1)
    ])
    return model
```

In [102…
```
NN_all = create_model_incl_dropout()
NN_all.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=[tf.kera
NN_all.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 512)               35328

 dropout (Dropout)           (None, 512)               0

 dense_13 (Dense)            (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_14 (Dense)            (None, 128)               32896

 dropout_2 (Dropout)         (None, 128)               0

 dense_15 (Dense)            (None, 1)                 129

=================================================================
Total params: 199681 (780.00 KB)
Trainable params: 199681 (780.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Die Architektur des Neuronalen Netzes wurde hier so gestaltet, dass Overfitting vermieden wird. Dazu wurde zwischen den einzelnen Layern jeweils ein "Dropout" eingefügt. Dies schaltet einen gewissen Prozentsatz der Neuronen (hier: 20%) aus und reduziert damit das Overfitting. Darüber hinaus wurde eine L2-Regularisierung des Netzes sowie early stopping eingefügt um ein mögliches Overfitting des Netzes zu verhindern.

In [103…
```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_root_mean_squared
model_checkpoint = tf.keras.callbacks.ModelCheckpoint('best_model.h5', monitor='

tic = time.time()
```

```
NN_histories = {}
NN_histories['NN_all'] = NN_all.fit(xse_train,
                     yse_train,
                     steps_per_epoch = STEPS_PER_EPOCH,
                     epochs = 1000,
                     validation_split = 0.2,
                     verbose=0,
                     callbacks=[early_stopping, model_checkpoint],
                     shuffle = True)

runtime_nn = time.time() - tic

print("Dauer NN (sec):" + "%6.0f" % (runtime_nn))
```

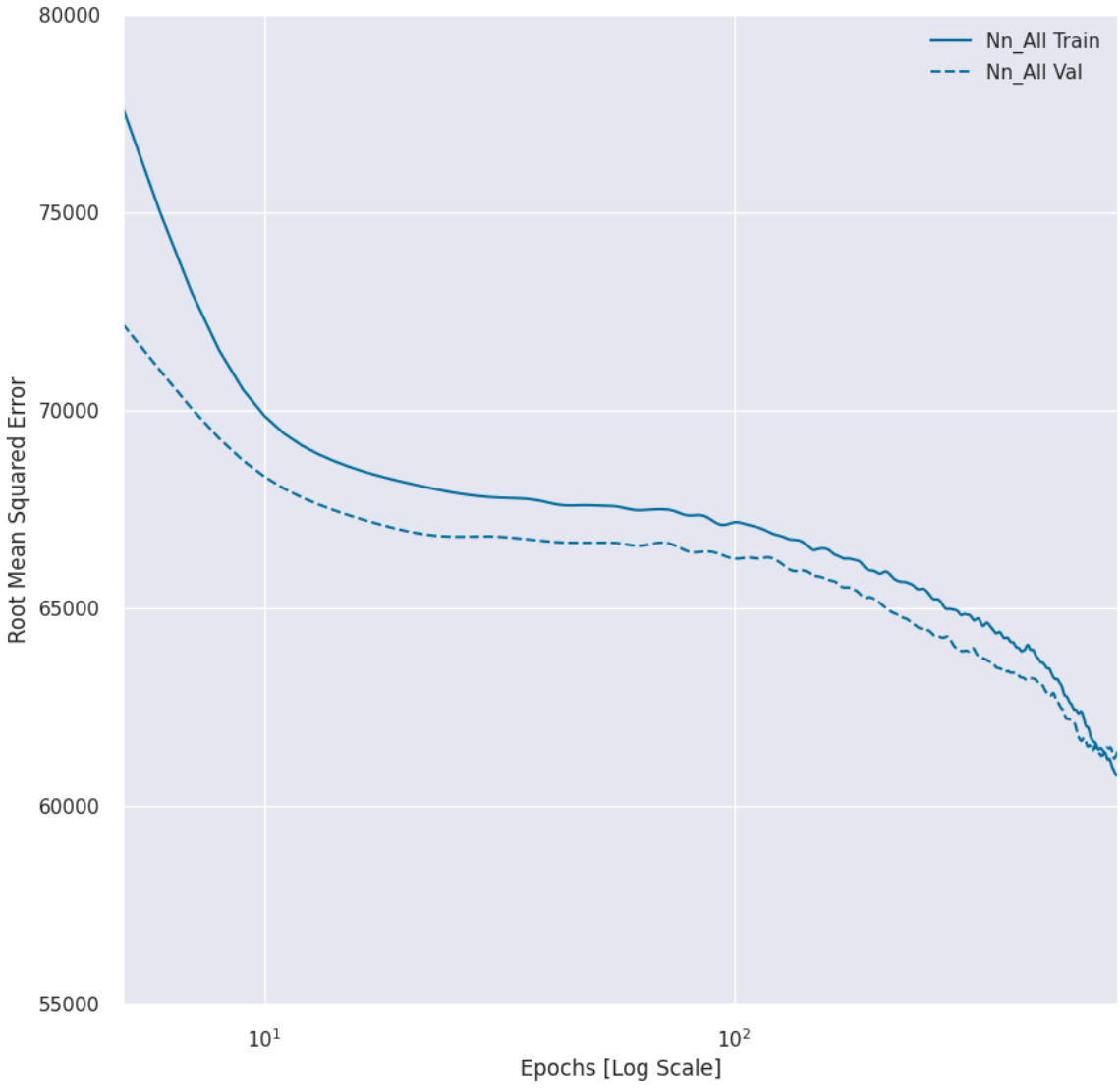
```
Epoch 659: early stopping
Dauer NN (sec):   615
```

In [104…

```
plotter = tfdocs.plots.HistoryPlotter(metric = 'root_mean_squared_error', smooth
plotter.plot(NN_histories)
a = plt.xscale('log')
plt.xlim([5, max(plt.xlim())])
plt.ylim(55000,80000)
plt.xlabel("Epochs [Log Scale]")
```

Out[104]:

```
Text(0.5, 0, 'Epochs [Log Scale]')
```



Im Trainingsverlauf des Netzes ist kein Overfitting erkennbar.

In [105…
```
# Modelgüte an Trainingsstichprobe ermitteln
pred_train_NN = NN_all.predict(xse_train)
rmse = mean_squared_error(yse_train, pred_train_NN , squared=False)
rmse
```

Out[105]:
```
451/451 [==============================] - 1s 1ms/step
58451.40250285381
```

In [106…
```
# Modelgüte an Validierungsstichprobe ermitteln
pred_val_NN = NN_all.predict(xse_val)
rmse_val_NN = mean_squared_error(yse_val, pred_val_NN , squared=False)
rmse_val_NN
```

Out[106]:
```
98/98 [==============================] - 0s 1ms/step
62381.0444182371
```

In [107…
```
# Ergebnisse zum Modellvergleich hinzufügen und anzeigen
results_df.loc['NeuralNet'] = [runtime_nn,rmse_val_NN,'']
results_df
```

Out[107]:

Model	time_train	RMSE_val	RMSE_test
Linear_A	0.167730	68709.117110	
CatBoost_A	6.781105	48006.829301	
CatBoost_B_OHE	3.225582	48783.757576	
NeuralNet	615.197549	62381.044418	

Bewertung:

Die Modellgüte des hier erstellten neuonalen Netzes ist zwar besser als die des einfachen linearen Modells aus Aufgabe A-7, aber deutlich schlechter als die des einfachen CatBoost-Modells aus Aufgabe A-7 und zudem ein vielfaches laufzeitintensiver.

Aufgabe B-3: Neuronales Netz mit Embeddings [Lernziele 3.2, 4.1, 4.3, 5.2 & 6; 17 Punkte]

a) Die kategoriellen Merkmale _ocean_proximity_ und _county_name_ sind über zweidimensionale Embeddings in ein Neuronales Netz einzufügen. Die verborgenen Schichten des Netzes sollen dabei aus dem Neuronalen Netz in Teilaufgabe B2-2 übernommen werden. Die nötigen Anpassungen der Modellumsetzung sind detailliert vorab zu beschreiben und dann durchzuführen. Die Trainingsdaten sind zu fitten und die Konvergenz ist zu bewerten. Falls Letzte nicht gegeben ist sind die notwendigen Veränderungen durchzuführen. b) Der RMSE der Validierungsdaten des Netzes mit Embeddings ist samt Fitting-Laufzeit zu ermitteln und dem Modellvergleich hinzuzufügen. Das Neuronale Netz mit Embeddings ist mit dem Neuronalen Netz aus Aufgabe B-2 b) zu vergleichen und zu bewerten. c) Die Embeddings der kategoriellen Merkmale sind aus dem Modell zu extrahieren und grafisch darzustellen. d) Das im Aufgabenteil a) aufgebaute Netz mit Embeddings soll als "clone" nochmals,

nun aber mit einem anderen Modellnamen und anderen (random) Startgewichten erzeugt, neu gefitted und die Konvergenz überprüft und ggf. hergestellt werden. Die entsprechenden Embedding-Gewichte sind zu extrahieren und in einer verbundenen Grafik den Embeddings aus Teilaufgabe c) gegenüberzustellen (z.B. 2x2 Grafiken über-/nebeneinander) und zu interpretieren. e) Der RMSE der Validierungsdaten des Netzes aus Teilaufgabe c) ist samt Fitting-Laufzeit zu ermitteln und dem Modellvergleich hinzuzufügen. Die entsprechenden Ergebnisse der beiden Neuronalen Netz mit Embeddings sind miteinander und mit dem Neuronalen Netz aus Aufgabe B-2 b) sowie dem einfachen linearen Modell aus Aufgabe A-7 zu vergleichen und zu bewerten.

Lösungsvorschlag

Zu a):

Um das geforderte neuronale Netz mit zwei Embeddings via Keras in TensoFlow umsetzen zu können, sind einige Änderungen gegenüber Aufgabe B-2 erforderlich:

1. Ein neuronalen Netz mit Embeddings kann nicht sequentiell umgesetzt werden, sondern erfordert die Zusammenführung mehrer Teile in einer "concatenate layer". Die einzelnen Teile werden vorab als Inputs definiert, dimensioniert und benannt. Neben den jeweils eindimensionalen Inputs für die beiden kategoriellen Merkmale werden als weiterer Strang die sieben numerischen Features gemeinsam als Input definiert.

2. Für die kategoriellen Merkmale wird jeweils ein "label encoding" durchgeführt, damit diese die notwendige Zahlendarstellung für das neuronale Netz erhalten.

3. für jedes so aufbereitete kategorielle Merkmal wird eine Embedding-Layer definiert, dimensioniert (2-D) und anschließend über eine "flatten layer" in eine eindimensionale Struktur gebracht.

4. Die so aufbereiteten Strukturen werden in der concatenate layer zusammengeführt. Anschließend werden die verborgenen Schichten samt "output layer" hinzugefügt.

5. Bei der finalen Modellerstellung (vor Kompilierung) müssen die drei Input-Stränge (siehe 1.) angegeben werden.

In [108…
```
df_embeddings = df_scaled.copy(deep=True)
```

In [109…
```
# Label Enkodierung für `ocean_proximity`
all_names_ocean_proximity = set(df['ocean_proximity'])
mapper_ocean_proximity = {k: i for i, k in enumerate(all_names_ocean_proximity)}
reverse_mapper_ocean_proximity = {v: k for k, v in mapper_ocean_proximity.items(
df_embeddings['ocean_proximity_num'] = df_embeddings['ocean_proximity'].map(mapp

# Label Enkodierung für `county_name`
all_names_county_name = set(df['county_name'])
mapper_county_name = {k: i for i, k in enumerate(all_names_county_name)}
```

```
reverse_mapper_county_name = {v: k for k, v in mapper_county_name.items()}
df_embeddings['county_name_num'] = df_embeddings['county_name'].map(mapper_count
```

In [110…

```
# Einfache Stichprobeneinteilung des skalierten nicht enkodierten Datensatzes
xs_train = df_embeddings.query('sample =="A"').drop(['median_house_value', 'samp
ys_train = df_embeddings.query('sample =="A"').median_house_value
xs_val = df_embeddings.query('sample =="B"').drop(['median_house_value', 'sample
ys_val = df_embeddings.query('sample =="B"').median_house_value
```

In [111…

```
# Anzahl der einmaligen kat. Features
lenOcean = len(df.ocean_proximity.unique())
lenCounty = len(df.county_name.unique())
lenOcean, lenCounty
```

Out[111]:

```
(5, 58)
```

In [112…

```
d  = 2    # Dimension der embedding layers für kat. Features
```

In [113…

```
# Einfaches Modell definieren
input_numerical =       Input(shape=(7,), name='numerical')
input_ocean_proximity = Input(shape=(1,), name='ocean_proximity')
input_county_name =     Input(shape=(1,), name='county_name')

# Embeddings für die kat. Merkmale einfuegen
embedding_dim = d  # Groesse des Vektors fuer die Embeddings
ocean_proximity_embedding = layers.Embedding(input_dim=lenOcean,
                                             output_dim=embedding_dim,
                                             input_length=1,
                                             name='ocean_proximity_embedding')(i
county_name_embedding = layers.Embedding(input_dim=lenCounty,
                                         output_dim=embedding_dim,
                                         input_length=1,
                                         name='county_name_embedding')(input_cou

# Flatten
ocean_proximity_embedding_flat = layers.Flatten()(ocean_proximity_embedding)
county_name_embedding_flat = layers.Flatten()(county_name_embedding)

# Concatenate
concatenated_embeddings = layers.Concatenate()([input_numerical, ocean_proximity

# Hinzufuegen von Hidden Layern
hidden1 = Dense(layer_one, activation=activation)(concatenated_embeddings)
dropout1 = Dropout(0.2)(hidden1)
hidden2 = Dense(layer_two,  activation=activation)(dropout1)
dropout2 = Dropout(0.2)(hidden2)
hidden3 = Dense(layer_three,  activation=activation)(dropout2)
dropout3 = Dropout(0.2)(hidden3)

# Output Layer
output = layers.Dense(1)(dropout3)
```

In [114…

```
# Modell erstellen und kompilieren
modelNNemb = Model(inputs=[input_numerical, input_ocean_proximity, input_county_
modelNNemb.compile(optimizer='adam', loss='mae', metrics=['mae', tf.keras.metric

# Modellzusammenfassung
modelNNemb.summary()

tic = time.time()

# Fit
```

```
modelNNemb.fit([xs_train[num_features], np.asarray(xs_train.ocean_proximity_num)
                np.asarray(ys_train), epochs=1000, batch_size=100, verbose = 0)

runtime_nnemb = time.time() - tic

print("Dauer NNemb (sec):" + "%6.0f" % (runtime_nnemb))
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 ocean_proximity (InputLaye  [(None, 1)]                  0         []
 r)

 county_name (InputLayer)    [(None, 1)]                  0         []

 ocean_proximity_embedding   (None, 1, 2)                 10        ['ocean_prox
imity[0][0]']
 (Embedding)

 county_name_embedding (Emb  (None, 1, 2)                 116       ['county_nam
e[0][0]']
 edding)

 numerical (InputLayer)      [(None, 7)]                  0         []

 flatten (Flatten)           (None, 2)                    0         ['ocean_prox
imity_embedding[0]
                                                                   [0]']

 flatten_1 (Flatten)         (None, 2)                    0         ['county_nam
e_embedding[0][0]'
                                                                   ]

 concatenate (Concatenate)   (None, 11)                   0         ['numerical
[0][0]',
                                                                    'flatten[0]
[0]',
                                                                    'flatten_1
[0][0]']

 dense_16 (Dense)            (None, 512)                  6144      ['concatenat
e[0][0]']

 dropout_3 (Dropout)         (None, 512)                  0         ['dense_16
[0][0]']

 dense_17 (Dense)            (None, 256)                  131328    ['dropout_3
[0][0]']

 dropout_4 (Dropout)         (None, 256)                  0         ['dense_17
[0][0]']

 dense_18 (Dense)            (None, 128)                  32896     ['dropout_4
[0][0]']

 dropout_5 (Dropout)         (None, 128)                  0         ['dense_18
[0][0]']

 dense_19 (Dense)            (None, 1)                    129       ['dropout_5
[0][0]']

==================================================================================================
Total params: 170623 (666.50 KB)
Trainable params: 170623 (666.50 KB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

```
Dauer NNemb (sec):   504
```

Bewertung: Die Modellkonvergenz ist gegeben.

Zu b):

In [115…
```
# Modelgüte an Trainingsstichprobe ermitteln
pred_train_NNemb = modelNNemb.predict([xs_train[num_features],
                 np.asarray(xs_train.ocean_proximity_num),
                 np.asarray(xs_train.county_name_num)
                                ]
                )
rmse = mean_squared_error(np.asarray(ys_train), pred_train_NNemb , squared=False
rmse
```

Out[115]:
```
451/451 [==============================] - 1s 2ms/step
58807.74101657234
```

In [116…
```
# Modelgüte an Validierungsstichprobe ermitteln
pred_val_NNemb = modelNNemb.predict([xs_val[num_features],
                 np.asarray(xs_val.ocean_proximity_num),
                 np.asarray(xs_val.county_name_num)
                                ]
                )
rmse = mean_squared_error(np.asarray(ys_val), pred_val_NNemb , squared=False)
rmse
```

Out[116]:
```
98/98 [==============================] - 0s 2ms/step
62817.44671776452
```

In [117…
```
# Ergebnisse zum Modellvergleich hinzufuegen
results_df.loc['NNemb'] = [runtime_nnemb,rmse,'']
results_df
```

Out[117]:

Model	time_train	RMSE_val	RMSE_test
Linear_A	0.167730	68709.117110	
CatBoost_A	6.781105	48006.829301	
CatBoost_B_OHE	3.225582	48783.757576	
NeuralNet	615.197549	62381.044418	
NNemb	503.869339	62817.446718	

Bewertung:

Die Modellgüte des NNs mit Embeddings hat sich hier - im Gegensatz zur cpu-Version - gegenüber dem NN ohne Embeddings etwas verschlechtert (RMSE größer). Mit einer kleineren batch_size von 32 wäre eine klare Verbesserung der Modellgüte gegenüber dem NN ohne Embeddings möglich, die Laufzeit allerdings viel länger (ca. 2000 cpu-sec statt ca. 800 cpu-sec, und das Modell weiterhin weit schlechter als CatBoot_A/B).

Zu c):

In [118…

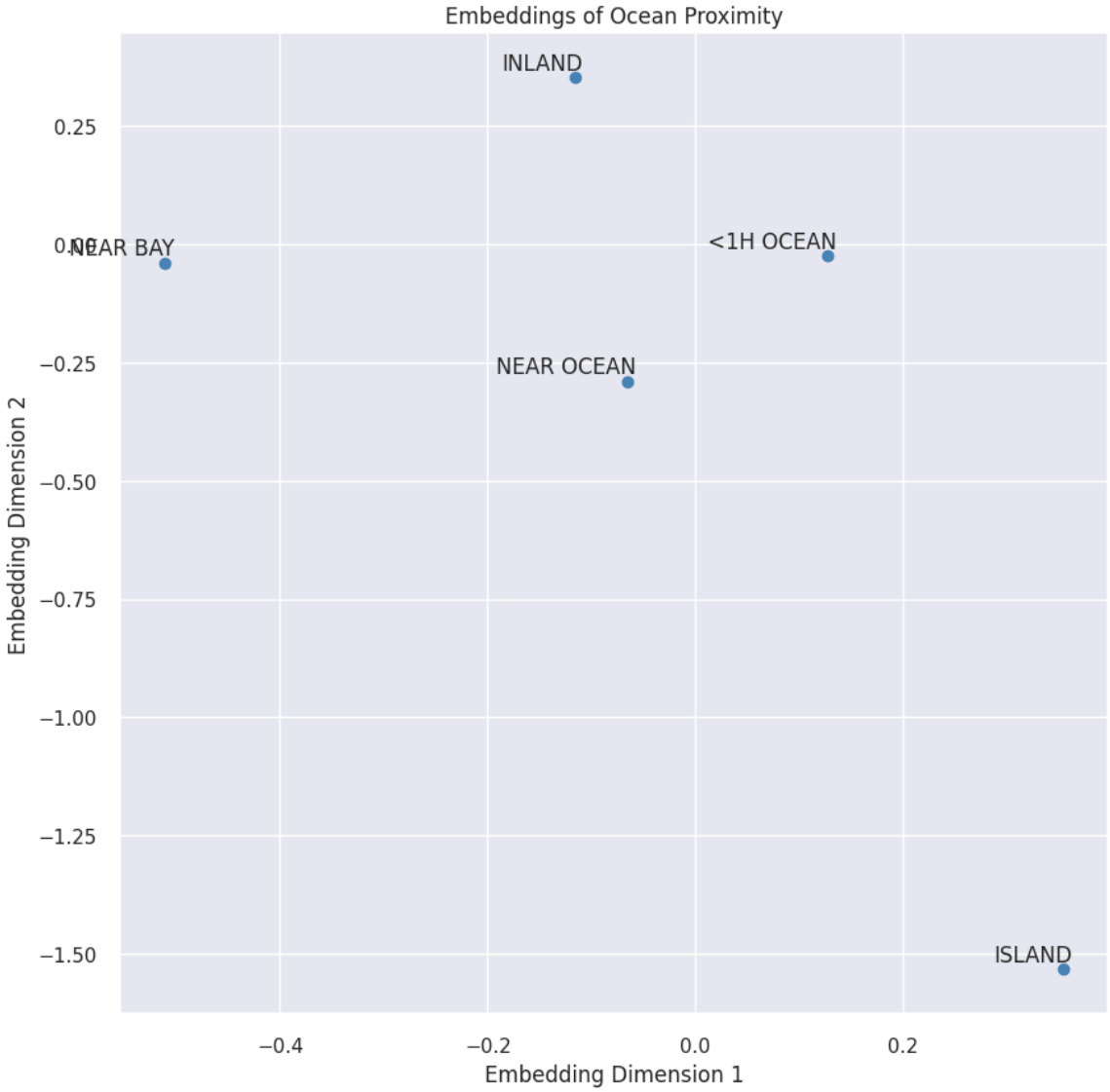
```
# Plotting
def plot_embeddings(embeddings, labels, title):
    plt.figure(figsize=(10, 10))
    plt.scatter(embeddings[:, 0], embeddings[:, 1], c='steelblue', cmap='Spectra
    for label, x, y in zip(labels, embeddings[:, 0], embeddings[:, 1]):
        plt.annotate(label, xy=(x, y), xytext=(5, 2),
                     textcoords='offset points', ha='right', va='bottom')
    plt.title(title)
    plt.xlabel('Embedding Dimension 1')
    plt.ylabel('Embedding Dimension 2')
    plt.grid(True)
    plt.show()
```

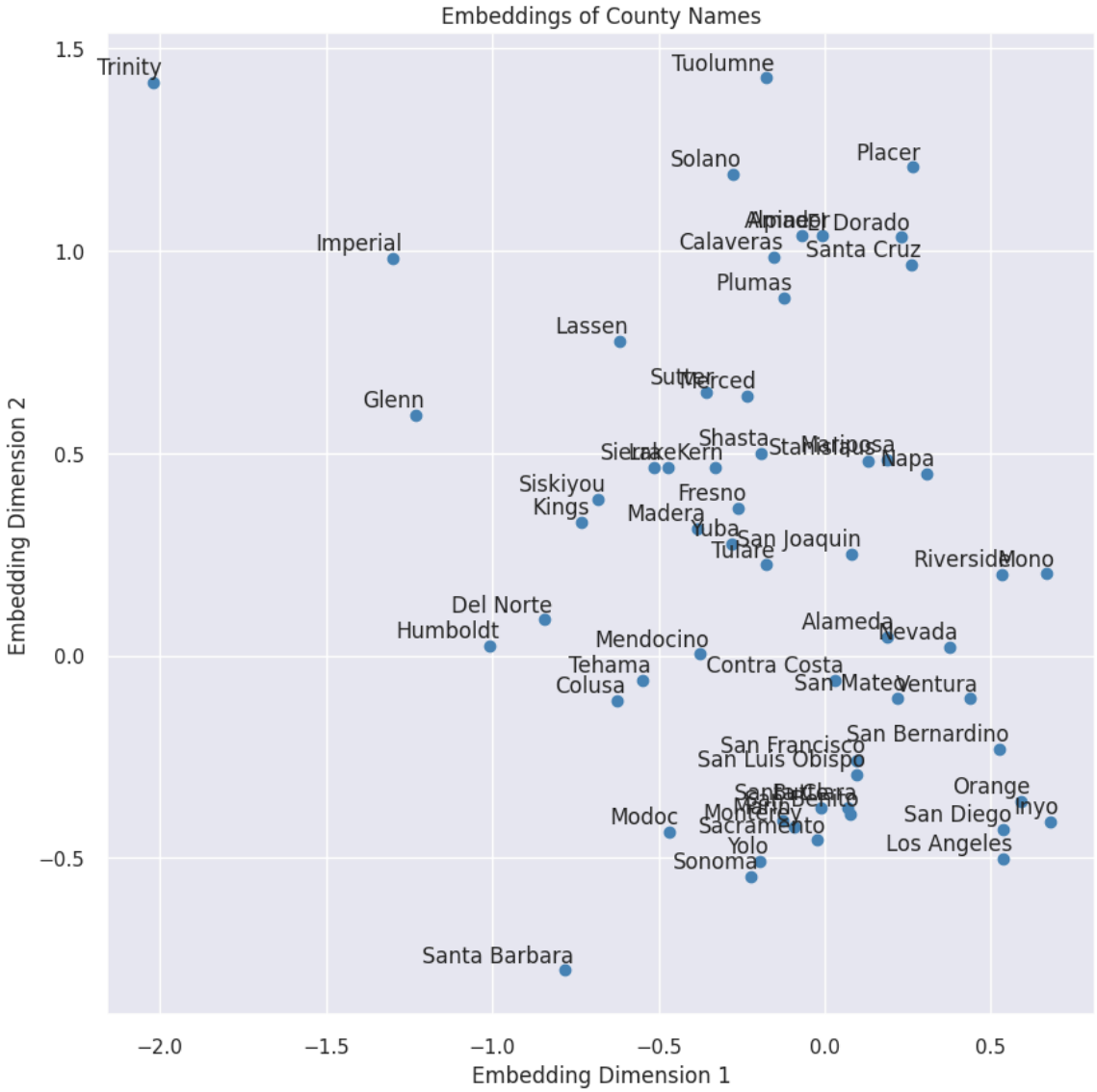
In [119…

```
# Liste mit Labels fuer die Embeddings
ocean_proximity_labels = [reverse_mapper_ocean_proximity.get(i) for i in range(l
county_name_labels = [reverse_mapper_county_name.get(i) for i in range(lenCounty
```

In [120…

```
# Nach dem Training werden die Embeddings extrahiert
ocean_proximity_layer = modelNNemb.get_layer('ocean_proximity_embedding')
county_name_layer = modelNNemb.get_layer('county_name_embedding')

ocean_proximity_embeddings_result = ocean_proximity_layer.get_weights()[0]
county_name_embeddings_result = county_name_layer.get_weights()[0]

# Plotten der Ocean Proximity Embeddings
plot_embeddings(ocean_proximity_embeddings_result, ocean_proximity_labels, 'Embe

# Plotten der County Name Embeddings
plot_embeddings(county_name_embeddings_result, county_name_labels, 'Embeddings o
```


Embeddings of Ocean Proximity
INLAND
<1H OCEAN
NEAR BAY
NEAR OCEAN
ISLAND
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
Embedding Dimension 2
−0.4
−0.2
0.0
0.2
Embedding Dimension 1



Zu d):

Wiederholung des Modells von 1):

In [121…

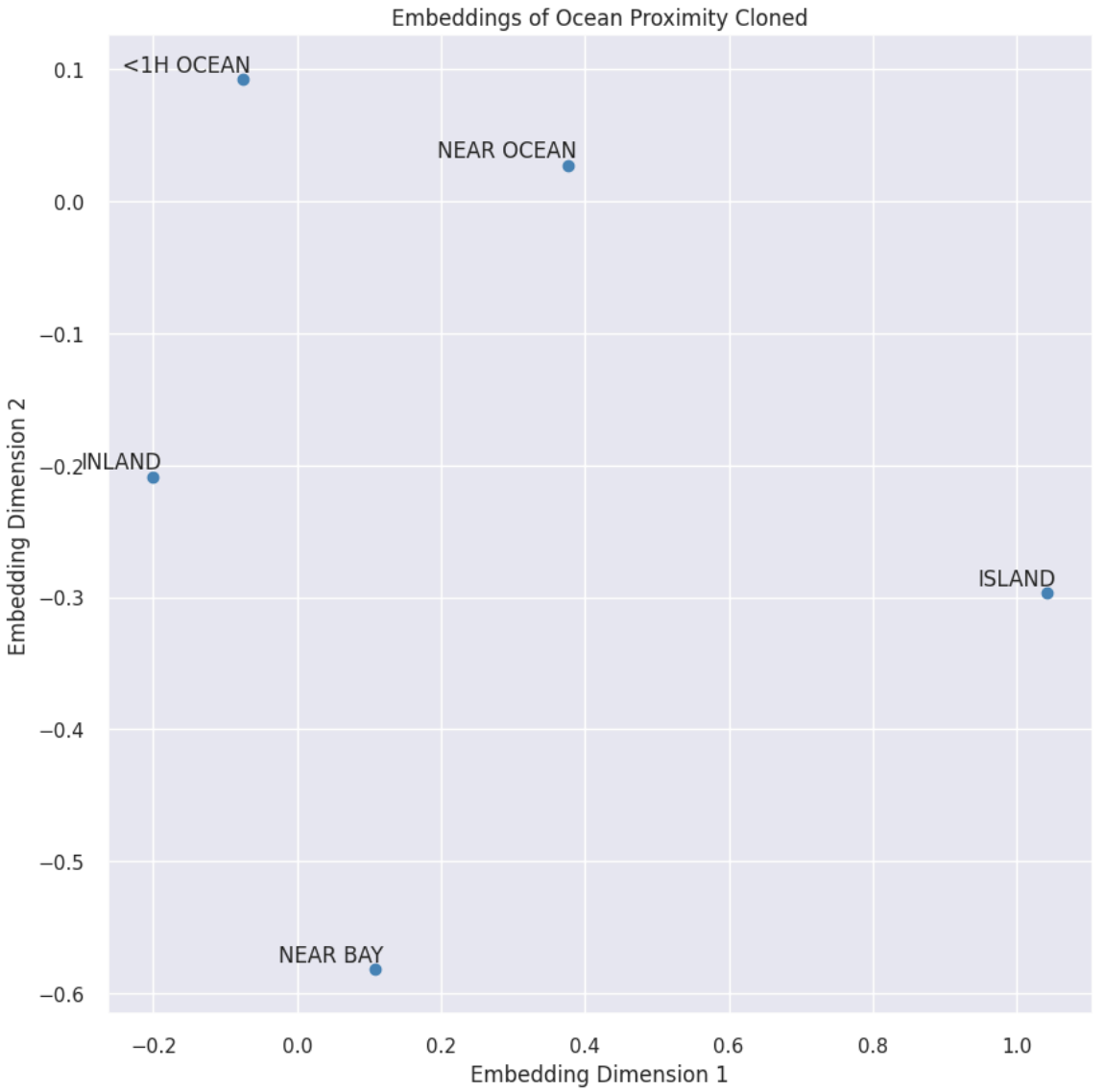
```
modelNNemb_cloned = tf.keras.models.clone_model(modelNNemb)
modelNNemb_cloned.compile(optimizer='adam', loss='mae', metrics=['mae', tf.keras

# Fit
tic = time.time()

modelNNemb_cloned.fit([xs_train[num_features], np.asarray(xs_train.ocean_proximi
               np.asarray(ys_train), epochs=1000, batch_size=100, verbose = 0)

runtime_nnemb_cloned = time.time() - tic

print("Dauer NNemb cloned (sec):" + "%6.0f" % (runtime_nnemb_cloned))
```

```
Dauer NNemb cloned (sec):   501
```

In [122…

```
# Nach dem Training werden die Embeddings extrahiert
ocean_proximity_layer_cloned = modelNNemb_cloned.get_layer('ocean_proximity_embe
county_name_layer_cloned = modelNNemb_cloned.get_layer('county_name_embedding')

ocean_proximity_embeddings_result_cloned = ocean_proximity_layer_cloned.get_weig
county_name_embeddings_result_cloned = county_name_layer_cloned.get_weights()[0]

# Plotten der Ocean Proximity Embeddings
```

```
plot_embeddings(ocean_proximity_embeddings_result_cloned, ocean_proximity_labels

# Plotten der County Name Embeddings
plot_embeddings(county_name_embeddings_result_cloned, county_name_labels, 'Embed
```



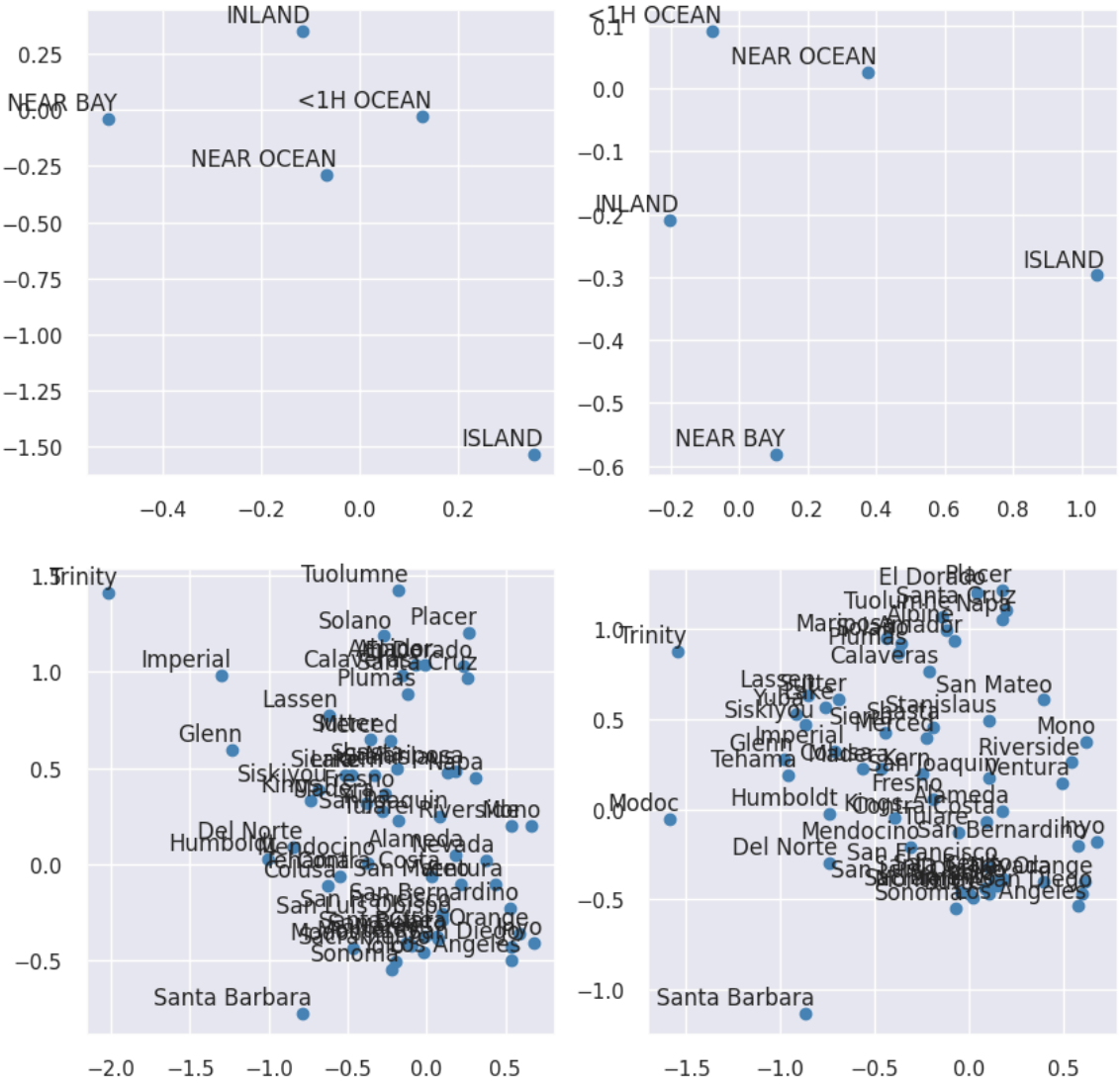
In [123…

```
# Plotting
def plot_multiple_embeddings(embeddings, labels, title, ax):
    ax.scatter(embeddings[:, 0], embeddings[:, 1], c='steelblue', cmap='Spectral
    for label, x, y in zip(labels, embeddings[:, 0], embeddings[:, 1]):
        ax.annotate(label, xy=(x, y), xytext=(5, 2),
                    textcoords='offset points', ha='right', va='bottom')
    ax.grid(True)
```

Gegenüberstellung der Embeddings beider Modelle (rechts "clone"):

In [124…

```
fig, axs = plt.subplots(nrows = 2, ncols = 2, figsize= (10, 10))
plot_multiple_embeddings(ocean_proximity_embeddings_result, ocean_proximity_labe
plot_multiple_embeddings(ocean_proximity_embeddings_result_cloned, ocean_proximi
plot_multiple_embeddings(county_name_embeddings_result, county_name_labels, 'Emb
plot_multiple_embeddings(county_name_embeddings_result_cloned, county_name_label
```



Interpretation:

Das Embedding für "ocean_proximity" ist die Besonderheit der Insellage zu erkennen. "NEAR BAY" befindet sich jeweils am anderen Ende der Diagonale. Es ist zu erkennen, dass selbst stabile Embeddings keine räumliche Ausrichtung haben und bei Wiederholung rotiert oder gespiegelt dargestellt werden können.

Das Embedding für "county" ist aufgrund der bereits recht hohen Anzahl an Kategorien (58) unübersichtlicher, zeigt aber auch stabile Achsen z.B. zwischen Trinity und Santa Barbara.

Zu e):

In [125…

```
# Modelgüte an Validierungsstichprobe ermitteln
pred_val_NNemb_cloned = modelNNemb_cloned.predict([xs_val[num_features],
                 np.asarray(xs_val.ocean_proximity_num),
                 np.asarray(xs_val.county_name_num)])

rmse = mean_squared_error(np.asarray(ys_val), pred_val_NNemb_cloned , squared=Fa
rmse
```

```
98/98 [==============================] - 0s 2ms/step
```

Out[125]:

```
62703.77896964207
```

In [126…
```
# Ergebnisse zum Modellvergleich hinzufuegen
results_df.loc['NNemb_clone'] = [runtime_nnemb_cloned,rmse,'']
results_df
```

Out[126]:

Model	time_train	RMSE_val	RMSE_test
Linear_A	0.167730	68709.117110	
CatBoost_A	6.781105	48006.829301	
CatBoost_B_OHE	3.225582	48783.757576	
NeuralNet	615.197549	62381.044418	
NNemb	503.869339	62817.446718	
NNemb_clone	501.270618	62703.778970	

Bewertung:

Die Modellgüte der NNs mit Embeddings hat sich hier - im Gegensatz zur cpu-Version - gegenüber dem NN ohne Embeddings bei etwas geringerer gpu-Trainingszeit etwas verschlechtert (RMSE größer). Insgesamt führt das spezifizierte NN mit Embeddings (in beiden Fällen) nun zu einer deutlich besseren Modellgüte als das einfache lineare Modell, allerdings zu Lasten eines stark erhöhten Aufwands sowohl im der Modelbildung als auch hinsichtlich des erforderlichen Rechenaufwandes.

Anmerkung: Bei diesen und den folgenden Modellen wurde aus Gründen des Resourcenverbrauchs und der Vergleichbarkeit bereits beim cpu-Notebook darauf geachtet, dass jedes Modell mit einer Trainingszeit von weniger als 1.000 Sekunden auskommt.

Aufgabe B-4: Hyperparameter-Tuning von lightGBM und XGBoost [Lernziele 3.2, 3.3/3.4, 4.1 & 6; 10 Punkte]

In den folgenden Modellen ist jeweils die Laufzeit der Durchführung des Codes zu messen. a) Es ist ein lightGBM-Modell inkl. Hypertuning mit mehrdimensionalem Grid Search und vierfacher Kreuzvalidierung zu erstellen. Dabei sind die Parameter "learning rate" und "num_leaves" zu optimieren, für den Parameter "n_estimators" ist der entsprechende Standardparameter des CatBoost Modells (!) zu verwenden. Die mit dem Grid Search ermittelten besten fünf Parametersätze sind anzuzeigen und zu diskutieren. Im Anschluss ist ein Modell mit den besten Parametern auf allen Folds zu fitten, der Fehler auf den Validierungsdaten zu berechnen und die Ergebnisse dem Modellvergleich hinzuzufügen. b) Analog ist ein XGBoost-Modell inkl. Hypertuning mit mehrdimensionalem Random Search und vierfacher Kreuzvalidierung zu erstellen. Dabei sind die Parameter "depth", "learning rate", "subsample" und "colsample_bytree" zu optimieren. Dabei ist die gleiche Anzahl Iterationen wie bei CatBoost und lightGBM zu verwenden. Die mit dem Randomized Search ermittelten besten fünf Parametersätze sind anzuzeigen und zu diskutieren. Im

Anschluss ist ein Modell mit den besten Parametern auf allen Folds zu fitten, der Fehler auf den Validierungsdaten zu berechnen und die Ergebnisse dem Modellvergleich hinzuzufügen. c) Abschließend ist der Aufwand und der Nutzen der hier optimierten Modelle gegenüber dem CatBoost-Benchmark-Modell aus Aufgabe A-7 zu bewerten.

Lösungsvorschlag

Zu a):

In [127…

```
device ='gpu' # cpu-Ausführung. Alternativen: 'cpu', gpu' or 'cuda', siehe https

tic = time.time()
# LGBMRegressor: Hyperparameteroptimierung mit GridSearch und 1000 Iterationen (

# Parameter mit Rücksicht auf die Laufzeit angepasst
param_grid = {
    'learning_rate': [0.01, 0.02, 0.03, 0.05],
    'num_leaves': [25, 50, 75, 100],
    'n_estimators': [1000],
    'device': [device]
}

# Variablenliste für die Anzeige des CV-Ergebnisses anlegen
sel_params_LGB = ['param_learning_rate','param_num_leaves','mean_test_score','ra

LGB_grid_search = GridSearchCV(
    LGBMRegressor(),
    param_grid,
    scoring='neg_root_mean_squared_error',
    cv=4)

LGB_grid_search.fit(xse_train,yse_train)

# Beste Parameter (sowie die Nächstbesten) ausgeben
pd.DataFrame(LGB_grid_search.cv_results_)[sel_params_LGB].sort_values("rank_test
```

```
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
```

Out[127]:

	param_learning_rate	param_num_leaves	mean_test_score	rank_test_score
9	0.03	50	-46729.950451	1
5	0.02	50	-46757.130159	2
12	0.05	25	-46797.489404	3
13	0.05	50	-46892.164797	4
6	0.02	75	-46957.075615	5

Diskussion:

Die besten zwei Parametersätze liegen beim Test-Score dicht beeinander und verwenden 50 Blätter sowie eine Lernrate von 2% bzw. 3% und stammen nicht von Rand des Suchgitters (Gitter wäre dann zu erweitern).

Anmerkung: Die Anzahl der hier ausprobierten verschiedenen Parametersätze (5*4 = 20) wird auf das folgende XGBoost-Tuning übertragen.

In [128…

```
# LGBMRegressor: Erstelle neues Modell auf allen Folds mit den besten Parametern
LGBM = LGBMRegressor(**LGB_grid_search.best_params_)
LGBM.fit(xse_train, yse_train)

runtime_LGBM = time.time() - tic

print("time (sec):" + "%6.0f" % (runtime_LGBM))
```

```
time (sec):   386
```

In [129…

```
rmse_LGBM = mean_squared_error(yse_val, LGBM.predict(xse_val), squared=False)

# Ergebnisse zum Modellvergleich hinzufuegen
results_df.loc['lightGBM_tuned'] = [runtime_LGBM,rmse_LGBM,'']
results_df
```

Out[129]:

Model	time_train	RMSE_val	RMSE_test
Linear_A	0.167730	68709.117110	
CatBoost_A	6.781105	48006.829301	
CatBoost_B_OHE	3.225582	48783.757576	
NeuralNet	615.197549	62381.044418	
NNemb	503.869339	62817.446718	
NNemb_clone	501.270618	62703.778970	
lightGBM_tuned	386.166260	46861.066618	

Zu b):

XGBoost

In [130…

```
tree_method = 'gpu_hist' # cpu-Ausführung. Alternativen: 'hist' (cpu), 'gpu_hist

# XGBRegressor: Hyperparameteroptimierung mit RandomizedSearchCV
tic = time.time()

# Parametersuchbereich für XGBoost
param_grid ={'learning_rate': [0.01,0.015,0.02,0.03,0.05,0.08], 'max_depth':[4,5
             'subsample':[0.5, 0.70, 0.85, 1.0], 'colsample_bytree':[0.5, 0.70,
             'n_estimators': [1000],'tree_method': [tree_method] }

# Variablenliste für die Anzeige des CV-Ergebnisses anlegen
sel_params_XGB = ['param_learning_rate','param_max_depth','param_subsample',
                  'param_colsample_bytree','mean_test_score','rank_test_score']

# HP-Suche
XGB_random_search = RandomizedSearchCV(XGBRegressor(),param_grid, scoring='neg_r
XGB_random_search.fit(xse_train,yse_train)

# Beste Parameter (sowie die Nächstbesten) ausgeben
pd.DataFrame(XGB_random_search.cv_results_)[sel_params_XGB].sort_values("rank_te
```

Out[130]:

	param_learning_rate	param_max_depth	param_subsample	param_colsample_bytree	mean_t
0	0.03	8	0.85	0.7	-460
1	0.05	10	0.85	0.5	-463
6	0.05	8	1.0	0.7	-463
4	0.05	8	0.85	0.85	-464
15	0.01	10	1.0	0.7	-465

Diskussion:

In den Top5 ist über einen relativ breiten Bereich der Lernrate (3% bis 8%) auffällig oft die Baumtiefe 8 vertreten. Bei den Top2 werden 85% der Datensätze (subsample) und 50% bzw. 70% der Features (colsample_btree) verwendet.

Die Parameter des besten Modells stammen nicht vom Rand des Suchgitters. Bei der zweitbesten Lösung stammt die Baumtiefe vom oberen Rand des Suchgitters. Deutlich höhere Baumtiefen als 10 führen hier zu deutlich längeren Laufzeiten und schlechteren Modellen.

In [131…
```
# XGBRegressor: Erstelle neues Modell auf allen Folds mit den besten Parametern
XGB = XGBRegressor(**XGB_random_search.best_params_)
XGB.fit(xse_train, yse_train)

runtime_XGB = time.time() - tic

print("time (sec):" + "%6.0f" % (runtime_XGB))
```

```
time (sec):   330
```

In [132…
```
rmse_XGB = mean_squared_error(yse_val, XGB.predict(xse_val), squared=False)
rmse_XGB
```

Out[132]:
```
46684.17340865079
```

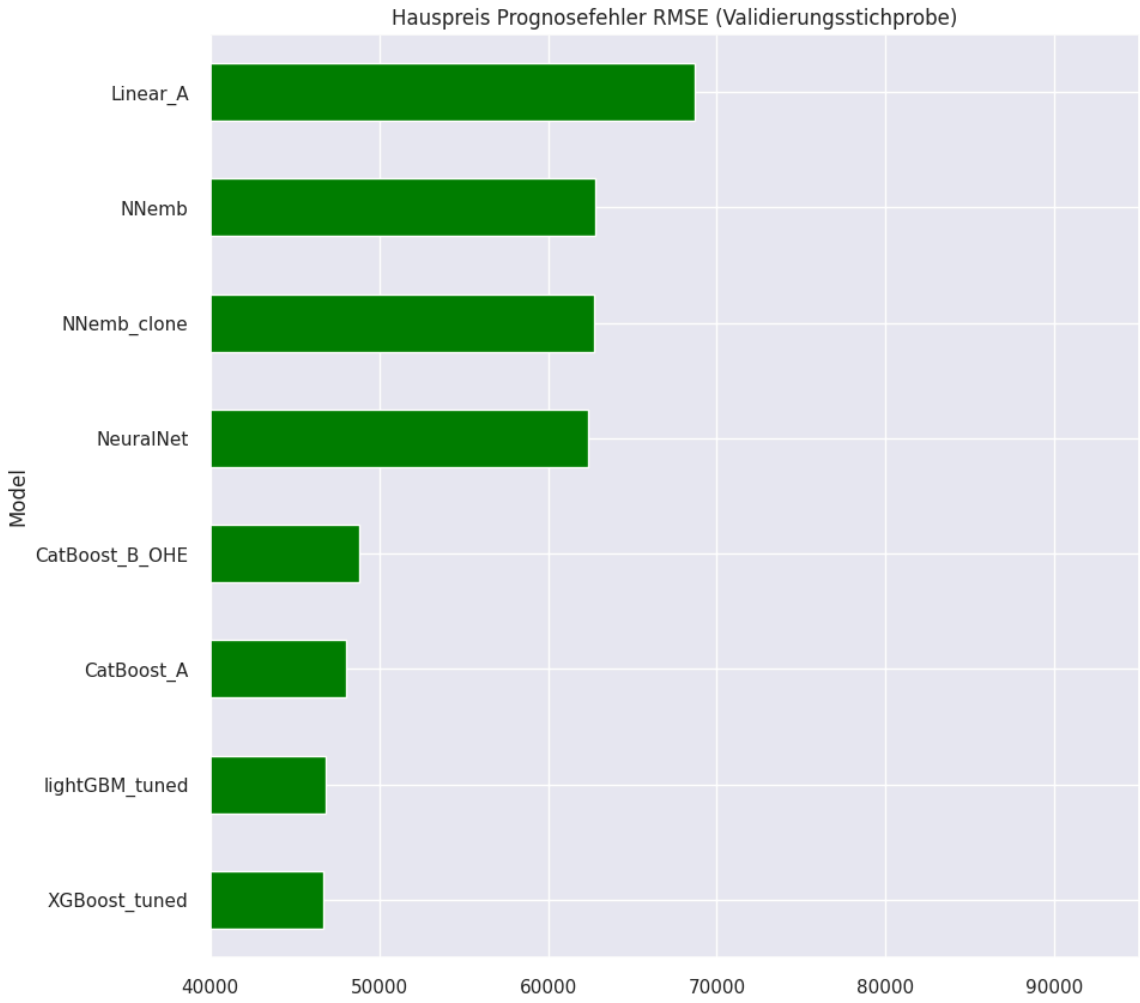
In [133…
```
# Ergebnisse zum Modellvergleich hinzufuegen
results_df.loc['XGBoost_tuned'] = [runtime_XGB, rmse_XGB,'']
results_df
```

Out[133]:

Model	time_train	RMSE_val	RMSE_test
Linear_A	0.167730	68709.117110	
CatBoost_A	6.781105	48006.829301	
CatBoost_B_OHE	3.225582	48783.757576	
NeuralNet	615.197549	62381.044418	
NNemb	503.869339	62817.446718	
NNemb_clone	501.270618	62703.778970	
lightGBM_tuned	386.166260	46861.066618	
XGBoost_tuned	329.775272	46684.173409	

Zu c):

In [134…
```
plot_rmse(results_df)
```



In [135…
```
results_df
```

Out[135]:

Model	time_train	RMSE_val	RMSE_test
XGBoost_tuned	329.775272	46684.173409	
lightGBM_tuned	386.166260	46861.066618	
CatBoost_A	6.781105	48006.829301	
CatBoost_B_OHE	3.225582	48783.757576	
NeuralNet	615.197549	62381.044418	
NNemb_clone	501.270618	62703.778970	
NNemb	503.869339	62817.446718	
Linear_A	0.167730	68709.117110	

Bewertung:

Gegenüber dem einfachen CatBoost-Benchmark-Modell aus Aufgabe A-7 kann mittels Hyperparametertuning mit lightGBM und XGBoost eine leicht bessere Prognosegüte (kleinerer RMSE) erreicht werden, allerdings ist damit ein deutlich höherer Umsetzungsaufwand (Code) und ein rund 100- bis 200-fach höherer Rechenaufwand verbunden, also ein vergleichsweise hoher Aufwand für einen bescheidenen Nutzen.

Aufgabe B-5: Finale Modellbewertung an Testdaten [Lernziele 4.1 & 6; 4 Punkte]

a) Das Benchmark-CatBoost-Modell aus Aufgabe A-7 sowie alle im Teil B des Notebooks enthaltenen Modelle sind abschließend mit den mit den Testdaten (sample = C) zu bewerten und die Ergebnisse in den Testdaten abzulegen. b) Die Modellgüte der Modelle ist grafisch aufzubereiten. Welche Modelle haben die beste Prognosegüte? Gibt es im Hinblick auf Prognosegüte, Optimierungsaufwand und Laufzeit ein besonders geeignetes und empfehlenswertes Modell?

Lösungsvorschlag

Zu a):

In [136…
```
# Einfache Stichprobeneinteilung, ergänzt
x_test = df.query('sample =="C"')[num_cat_features]
y_test = df.query('sample =="C"').median_house_value
```

In [137…
```
# Linear_A
rmse_OLS_numcat = mean_squared_error(y_test, results_ols_numcat.predict(x_test),
rmse_OLS_numcat
```

Out[137]:
```
66261.75409887604
```

In [138…
```
# CatBoost_A (Benchmark)
rmse_CB_numcat = mean_squared_error(y_test, model_cb_numcat.predict(x_test), squ
rmse_CB_numcat
```

Out[138]:
```
45613.780729220794
```

In [139…
```
# Einfache Stichprobeneinteilung des skalierten encodeten Datensatzes
xse_test = df_scaled_encoded.query('sample =="C"').drop(['median_house_value', '
yse_test = df_scaled_encoded.query('sample =="C"').median_house_value
```

In [140…
```
# CatBoost_B_OHE
rmse_cb_b = mean_squared_error(yse_test, model_cb_b.predict(xse_test), squared=F
rmse_cb_b
```

Out[140]:
```
45829.66982393344
```

In [141…
```
# NN
pred_test_NN = NN_all.predict(xse_test)
rmse_NN = mean_squared_error(yse_test, pred_test_NN , squared=False)
rmse_NN
```

Out[141]:
```
98/98 [==============================] - 0s 1ms/step
62000.841030214986
```

In [142…
```
# Einfache Stichprobeneinteilung des skalierten Datensatzes
xs_test = df_embeddings.query('sample =="C"').drop(['median_house_value', 'sampl
ys_test = df_embeddings.query('sample =="C"').median_house_value
```

In [143…
```
# NNemb
pred_test_NNemb = modelNNemb.predict([xs_test[num_features],
                 np.asarray(xs_test.ocean_proximity_num),
                 np.asarray(xs_test.county_name_num)
                                      ]
                )
rmse_NNemb = mean_squared_error(ys_test, pred_test_NNemb , squared=False)
rmse_NNemb
```

```
98/98 [==============================] - 0s 2ms/step
```
Out[143]:
```
59788.90025771615
```

In [144…
```
# NNemb_clone
pred_test_NNemb_cloned = modelNNemb_cloned.predict([xs_test[num_features],
                 np.asarray(xs_test.ocean_proximity_num),
                 np.asarray(xs_test.county_name_num)
                                      ]
                )
rmse_NNemb_cloned = mean_squared_error(ys_test, pred_test_NNemb_cloned , squared
rmse_NNemb_cloned
```

```
98/98 [==============================] - 0s 2ms/step
```
Out[144]:
```
60063.07913640933
```

In [145…
```
# LGBM
rmse_LGBM = mean_squared_error(yse_test, LGBM.predict(xse_test), squared=False)
rmse_LGBM
```

Out[145]:
```
44813.72599618698
```

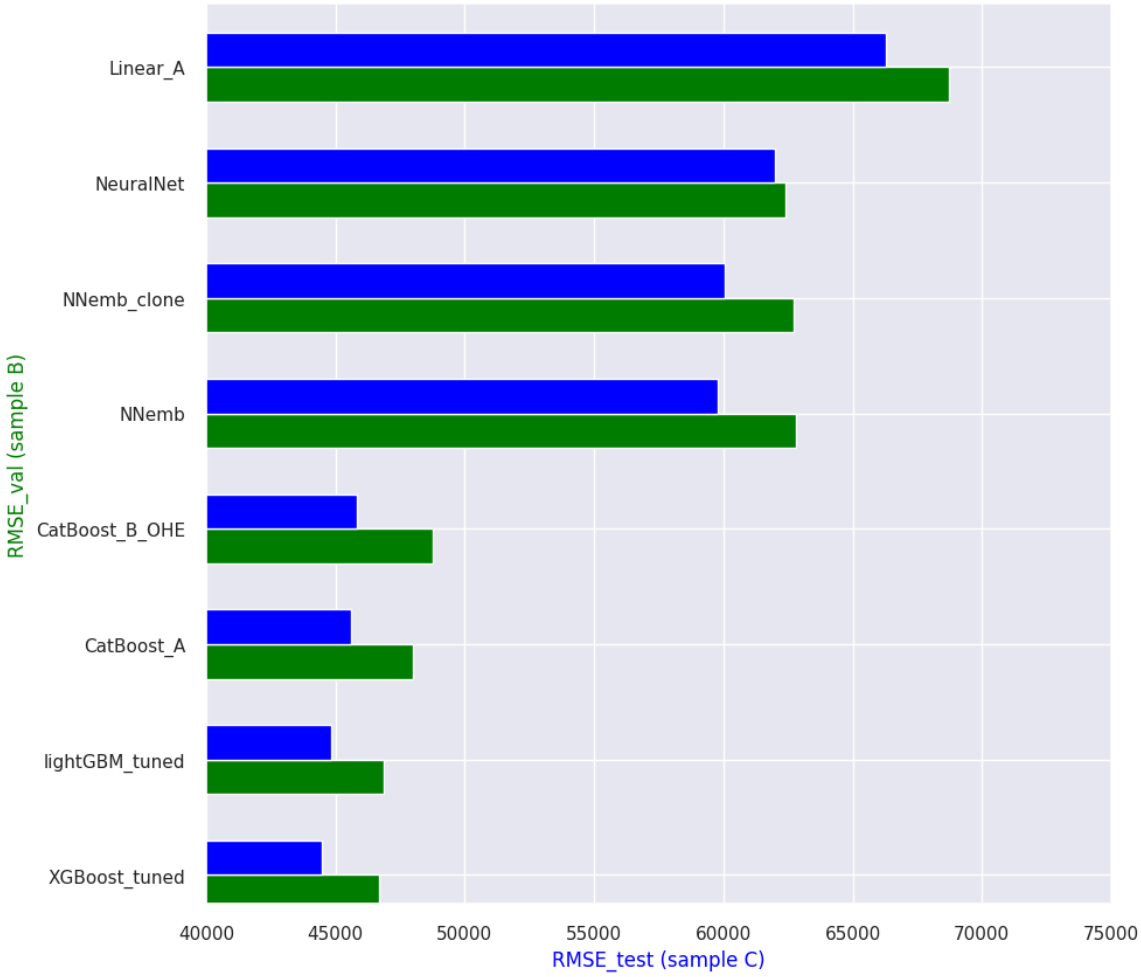
In [146…
```
# CatBoost
rmse_XGB = mean_squared_error(yse_test, XGB.predict(xse_test), squared=False)
rmse_XGB
```

Out[146]:
```
44485.63410566068
```

In [147…
```
# Werte von RMSE_test für die einzelnen Modelle in die Datenstruktur "results_df
rmse_CB_numcat
results_df.at['Linear_A','RMSE_test'] = rmse_OLS_numcat
results_df.at['CatBoost_A','RMSE_test'] = rmse_CB_numcat
results_df.at['CatBoost_B_OHE','RMSE_test'] = rmse_cb_b
results_df.at['NeuralNet','RMSE_test'] = rmse_NN
results_df.at['NNemb','RMSE_test'] = rmse_NNemb
results_df.at['NNemb_clone','RMSE_test'] = rmse_NNemb_cloned
results_df.at['lightGBM_tuned','RMSE_test'] = rmse_LGBM
results_df.at['XGBoost_tuned','RMSE_test'] = rmse_XGB
```

Zu b):

In [148…
```
# Plot der Güte der betrachteten Modelle bzgl. Validierungs- und Testdaten
plot_rmse_2(results_df)
```



In [149…

```
#Ergebnisse tabellarisch einschließlich Laufzeit des Modelltrainings
results_df
```

Out[149]:

Model	time_train	RMSE_val	RMSE_test
XGBoost_tuned	329.775272	46684.173409	44485.634106
lightGBM_tuned	386.166260	46861.066618	44813.725996
CatBoost_A	6.781105	48006.829301	45613.780729
CatBoost_B_OHE	3.225582	48783.757576	45829.669824
NNemb	503.869339	62817.446718	59788.900258
NNemb_clone	501.270618	62703.778970	60063.079136
NeuralNet	615.197549	62381.044418	62000.84103
Linear_A	0.167730	68709.117110	66261.754099

Bewertung und Empfehlung:

Insgesamt wird mit den Gradient-Tree-Boosting-Tools CatBoost, lightGBM und XGBoost auch mit den Testdaten (sample=C) eine weit bessere Modellgüte erzielt als mit den anderen verwendeten Verfahren. XGBoost und lightGBM liegen auch bei den Testdaten fast gleichauf, mit geringem Abstand folgt CatBoost.

Alle Modelle liefern auf den Testdaten (sample=C) ein besseres Ergebnis als auf den Validierungsdaten (sample=B).

Als besonders emfehlenswert erscheint hier CatBoost, da es nur minimale Datenaufbereitung benötigt und bereits ohne weiteres Tuning in sehr kurzer Zeit zuverlässig ein nur mit erheblichem Aufwand zu verbesserndes Prognoseergebnis erzielt.

Laufzeitmessung:

In [150…
```
# Zeit Notebook: Ende Teil B, Start Teil C
startzeit_teil_c = time.time()
laufzeit_teil_b = startzeit_teil_c - startzeit_teil_b

# Laufzeit anzeigen
runtime_abc_df.loc['B'] = [laufzeit_teil_b]
runtime_abc_df
```

Out[150]:

	Laufzeit
Teil	
A	64.397617
B	2887.858738

Teil C: Clustering unter Verwendung von

Aufgabe C-1: Erweiterung des Hauptdatensatzes um Feuergefährdungs-zonen [Lernziele 3.3/3.4, 5.2 & 6; 12 Punkte]

Der Datensatz *california_fire_fones.zip* beinhaltet geografische Informationen zu Feuergefährdungszonen in Kalifornien. Konkret enthält jede Zeile dieses Datensatzes ein Polygon (gespeichert im Feature *geometry*), dem eine Feuergefährdungszone (gespeichert im Feature *FHSZ* mit Interpretation 1 ≙ moderate Gefährdung, 2 ≙ hohe Gefährdung und 3 ≙ sehr hohe Gefährdung) sowie Fläche (gespeichert im Feature *Shape_Area*) und Länge (gespeichert im Feature *Shape_Leng*) zugeordnet sind. [Datenquelle: https://gis.data.ca.gov/maps/CALFIRE-Forestry::california-fire-hazard-severity-zones-fhsz/about (geringfügig modifiziert)] a) Der Datensatz *california_fire_zones.zip* ist einzulesen (vgl. Anhang 1), im Anschluss sind 10 zufällige Zeilen davon auszugeben. Die absolute Häufigkeitsverteilung der Werte des Merkmals *FHSZ* ist zu erstellen. Weiter ist eine Funktion zu schreiben, welche für ein gegebenes Polygon die Anzahl an Eckpunkten zurückgibt. Unter Verwendung dieser Funktion ist ein Histogramm zur Anzahl an Eckpunkten der Polygone des eingelesenen Datensatzes zu erstellen, bei welchem die x-Achse logarithmiert dargestellt wird, und zu kommentieren. Schließlich ist ein Plot der Feuergefährdungszonen zu erzeugen und mit einer Karte von Kalifornien zu hinterlegen. b) Der Hauptdatensatz, der nach der Bearbeitung von Teil A vorliegt, ist um die in Teilaufgabe a) eingelesenen Feuergefährdungszonen zu erweitern. Zu diesem Zweck soll festgestellt werden, ob und ggf. in welcher Feuergefährdungszone die Longitude-Latitude-Paare der einzelnen Zeilen

liegen. Das Resultat ist in der neuen Spalte *fire_zone* abzuspeichern. Liegt ein Longitude-Latitude-Paar in keiner Feuergefährdungszone, so ist als Wert 0 einzutragen. Liegt ein Longitude-Latitude-Paar in mehr als einer Feuergefährdungszone, so ist eine aktuariell angemessene Lösungsstrategie zu diesem Problem durchzuführen. Schließlich soll das Ergebnis überprüft werden, indem ein geeigneter Scatter-Plot der Daten erzeugt wird.

Hinweis: Gegebenenfalls sind die Koordinatensysteme der Polygone und der Longitude-Latitude-Paare zu synchronisieren. Mögliche Code-Schnipsel für Python und R sind in Anhang 1 angegeben.

Lösungsvorschlag

Zu a):

Zunächst lesen wird den Geodatensatz mittels der Funktion `read_file` des Moduls `geopandas` ein. Anschließend werden 10 zufällige Zeilen des Datensatzes ausgegeben.

In [151…
```
# lese den Geodatensatz ein
df_fire_zones = gpd.read_file('../input/calfire/california_fire_zones')

# gebe 10 zufällige Zeilen des Datensatzes aus
df_fire_zones.sample(10, random_state=seed)
```

Out[151]:

	FHSZ	FHSZ_Descr	Shape_Leng	Shape_Area	geometry
1263	2	High	16980.000000	1.319400e+06	POLYGON ((-13476878.151 4420400.737, -13476877...
1163	2	High	9000.000000	1.037700e+06	POLYGON ((-13503182.516 4336375.203, -13503182...
2259	3	Very High	6397.577596	2.558013e+06	POLYGON ((-12961209.302 3963785.272, -12963119...
810	1	Moderate	41165.625171	1.248081e+07	POLYGON ((-13798691.909 5044837.213, -13798802...
3292	3	Very High	69121.422548	2.889942e+07	POLYGON ((-13363438.505 4812199.416, -13363454...
1476	2	High	10080.000000	2.434500e+06	POLYGON ((-13275315.594 4619154.040, -13275315...
144	1	Moderate	9262.700284	1.561922e+06	POLYGON ((-13420795.699 4169314.854, -13420978...
3454	3	Very High	9033.972339	1.903265e+06	POLYGON ((-13410540.317 4899044.672, -13410539...
1176	2	High	21167.305421	3.066696e+06	POLYGON ((-13507728.526 4348021.921, -13507728...
759	1	Moderate	8400.000000	1.049400e+06	POLYGON ((-13795419.851 4962911.433, -13795418...

Wir geben die absolute Häufigkeitsverteilung der Werte des Merkmals `FHSZ` aus.

In [152…
```
df_fire_zones.value_counts('FHSZ')
```

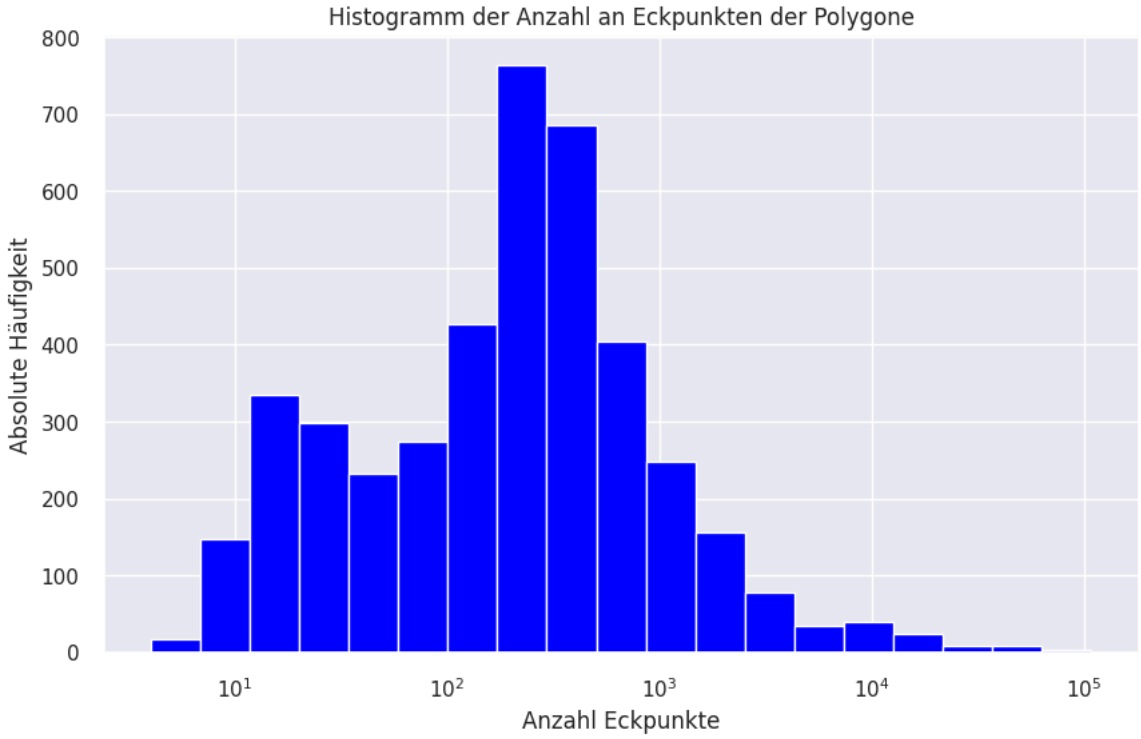
Out[152]:
```
FHSZ
3    1983
2    1315
1     881
Name: count, dtype: int64
```

Nun definieren wir eine Funktion, welche für ein gegebenes Polygon die Anzahl an Eckpunkten zurückgibt. Anschließend wenden wir diese Funktion auf die Spalte `geometry` an, um die Eckpunkte des Polygons einer jeden Zeile zu zählen.

In [153…
```
# erstelle eine Funktion, welche die Anzahl an Eckpunkten eines Polygons zurückg
def count_polygon_nodes(polygon):
    # ziehe von der Länge der Koordinaten eins ab, da Start- und Endpunkt gleich
    num_nodes = len(polygon.exterior.coords) - 1
    return num_nodes

# ergänze den Datensatz `df_fire_zones` um die Anzahl an Eckpunkten der Polygone
df_fire_zones['num_nodes'] = df_fire_zones['geometry'].apply(count_polygon_nodes
```

Anschließend verwenden wir die eben erstellte Funktion, um ein Histogramm zur Anzahl an Eckpunkten der Polygone des Datensatzes `df_fire_zones` zu erstellen, bei welchem die *x*-Achse logarithmiert dargestellt wird.

In [154…
```
# bereite den Plot vor und setze die Plotgröße
fig, ax = plt.subplots(figsize=(10, 6))

# erzeuge ein Histogramm zur Anzahl an Eckpunkten der Polygone mit
# logarithmiertem Binning für eine logarithmierte x-Achse
bin_edges = np.logspace(np.log10(df_fire_zones['num_nodes'].min()),
                        np.log10(df_fire_zones['num_nodes'].max()), 20)
df_fire_zones['num_nodes'].hist(bins=bin_edges, ax=ax, color='blue')

# verwende eine logarithmierte x-Achse
ax.set_xscale('log')

# füge Titel und Achsenbeschriftungen hinzu
ax.set_title('Histogramm der Anzahl an Eckpunkten der Polygone')
ax.set_xlabel('Anzahl Eckpunkte')
ax.set_ylabel('Absolute Häufigkeit')

# Show the plot
plt.show()
```

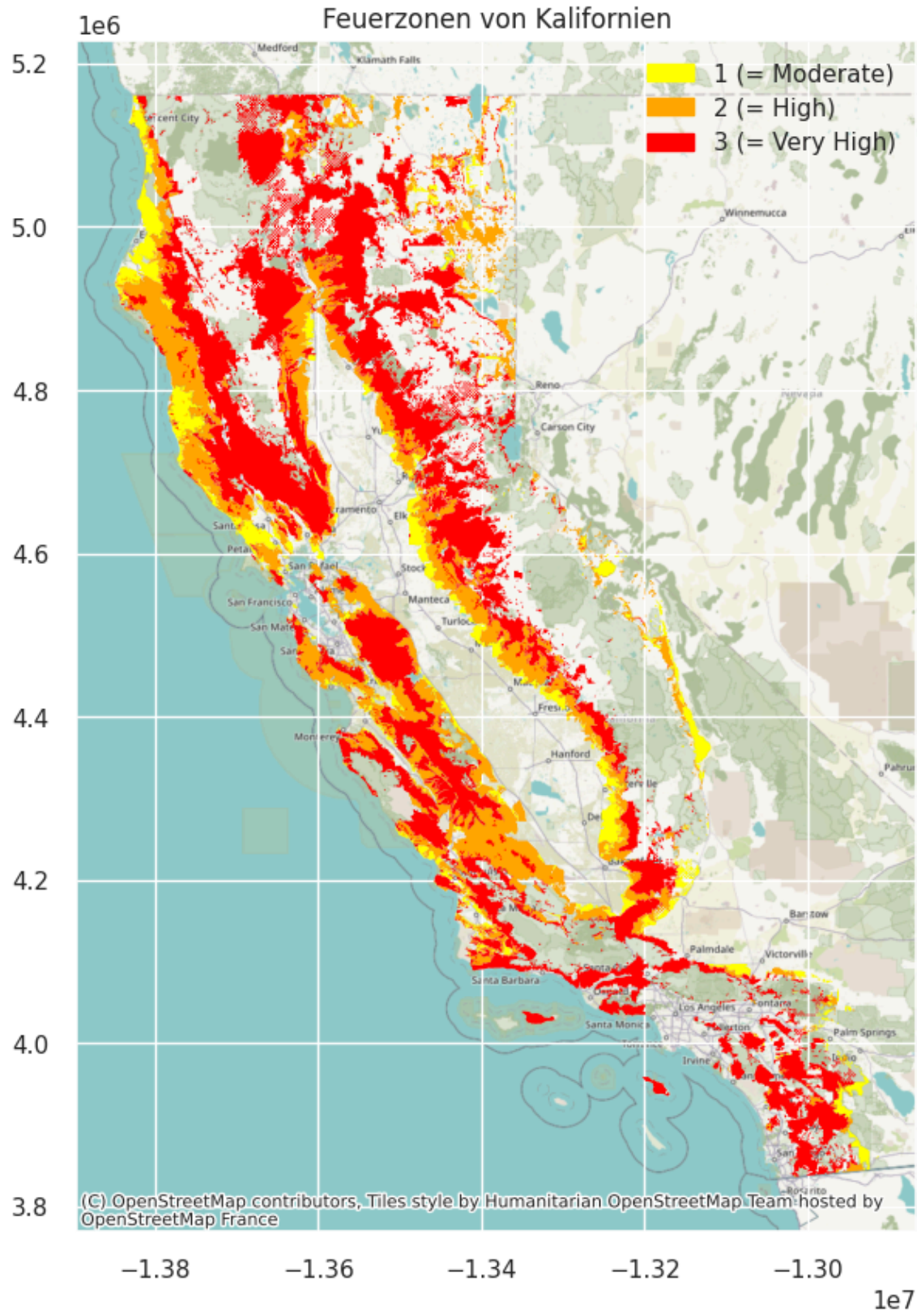


Das obige Histogramm zeigt eine starke Streuung der Anzahl an Eckpunkten der Polygone: Es gibt Polygone mit weniger als 10 Eckpunkten, aber auch Polygone mit extrem vielen Eckpunkten bis hin zur Größenordnung Hunderttausend.

Der nachfolgende Code zeichnet die eben eingelesen Polygone gefärbt nach ihrer jeweiligen Feuergefährdungszone über eine Karte von Kalifornien ein.

In [155…

```
# plotte die gefärbten Polygone
ax = df_fire_zones.plot(column="FHSZ", figsize=(10,10), legend=False,
                        cmap=colors.ListedColormap(['yellow', 'orange', 'red']),
                        edgecolor="face", linewidth=0.001, categorical=True)

# füge Titel und Legende hinzu
plt.title('Feuerzonen von Kalifornien')
patch_yellow = mpatches.Patch(color='yellow', label='1 (= Moderate)')
patch_orange = mpatches.Patch(color='orange', label='2 (= High)')
patch_red    = mpatches.Patch(color='red',    label='3 (= Very High)')
plt.legend(handles=[patch_yellow, patch_orange, patch_red])

# füge eine Karte von Kalifornien als Hintergrund hinzu
ctx.add_basemap(ax)
```



Zu b):

Wir erzeugen uns zunächst eine Kopie des Hauptdatensatzes, der nach der Bearbeitung von Teil A vorliegt. Anschließend wandeln wir die durch `latitude` und `longitude` gegebenen zweidimensionalen Koordinaten in `Point` -Werte im `geopandas` -Setting um und speichern diese im GeoDataFrame `gpd_points`, von welchem wir uns die ersten fünf Zeilen ausgeben lassen.

In [156…

```
# erzeuge eine Kopie des Hauptdatensatzes
df_C = df.copy(deep=True)
```

```
# erzeuge einen temporären GeoDataFrame bestehend aus den umgewandelten Points
gdf_points = gpd.GeoDataFrame(geometry=create_points(df_C['latitude'], df_C['lon
gdf_points.set_crs(epsg=4326, inplace=True)
gdf_points = gdf_points.to_crs(epsg=3857)

# setze einen Index zum späteren Join mit df_C
gdf_points['index'] = df_C.index

# gebe die ersten fünf Zeilen von `gdf_points` aus
gdf_points.head(5)
```

Out[156]:

	geometry	index
0	POINT (-13379489.598 4589319.190)	17848
1	POINT (-13144605.473 4048961.189)	9168
2	POINT (-13107870.041 4036861.408)	7517
3	POINT (-13680052.224 4782012.781)	19997
4	POINT (-13166869.371 4031487.854)	6558

Nun führen wir einen "Spatial Join" durch, welcher für jeden Punkt von `gdf_points` die – wenn existent – zugehörige Feuergefährdungszone hinzufügt. Auch hier lassen wir die ersten fünf Zeilen ausgeben.

In [157…

```
# führe einen "Spatial Join" durch, um allen Points eine Feuergefährungszone zuz
gdf_fire_zones = df_fire_zones[['geometry', 'FHSZ']]
gdf_joined = gpd.sjoin(gdf_points, gdf_fire_zones, how='left', op='within')

# gebe die ersten fünf Zeilen von `gdf_joined` aus
gdf_joined.head(5)
```

Out[157]:

	geometry	index	index_right	FHSZ
0	POINT (-13379489.598 4589319.190)	17848	2858.0	3.0
1	POINT (-13144605.473 4048961.189)	9168	NaN	NaN
2	POINT (-13107870.041 4036861.408)	7517	NaN	NaN
3	POINT (-13680052.224 4782012.781)	19997	NaN	NaN
4	POINT (-13166869.371 4031487.854)	6558	NaN	NaN

Schließlich joinen wir die erhaltenen Feuergefährdungszonen an den ursprünglichen Datensatz `df_C`. Fehlende Werte beim Merkmal `fire_zone` werden mit dem Wert 0 aufgefüllt. Schließlich werden noch einige Zeilen des Datensatzes zur Überprüfung ausgegeben.

In [158…

```
# merge die Feuergefährungszonen zurück an df_C
df_C = df_C.merge(gdf_joined[['index', 'FHSZ']], left_index=True, right_on='inde

# weise dem neuen Merkmal `fire_zone` die Feuergefährungszone zu und ersetze feh
df_C['fire_zone'] = df_C['FHSZ'].fillna(0).astype(int)

# lösche die nicht mehr benötigten Spalten `FHSZ` und `index`
df_C.drop(columns=['FHSZ', 'index'], inplace=True)
```

```
# gebe 10 zufällige Zeilen des Datensatzes aus
df_C.sample(10, random_state=seed)
```

Out[158]:

	latitude	longitude	housing_median_age	population	median_income	median_house_va
14742	33.91	-118.32	34	1920	4.5304	181
15080	37.73	-122.42	35	1055	4.6250	279
3178	33.93	-118.18	31	1820	2.1641	122
8362	37.78	-122.45	45	1320	3.1576	333
6939	37.75	-122.42	52	1460	2.9052	400
3345	33.49	-117.67	15	983	2.1935	183
6530	33.95	-117.42	32	2494	2.8173	110
20424	36.85	-119.84	8	1424	10.5144	345
8583	33.77	-117.99	29	922	3.1902	202
10243	33.77	-117.92	28	3282	3.5220	190

Wir betrachten nun die Anzahl an Zeilen des Datensatzes vor und nach der Ergänzung von `fire_zone`:

In [159…

```
# Ausgabe der Anzahl der Zeilen und Spalten des Datensatzes (vor Ergänzung von `
print('Zeilen und Spalten des Datensatzes (vor Ergänzung von `fire_zone`):')
print(df.shape)

# Ausgabe der Anzahl der Zeilen und Spalten des Datensatzes (nach Ergänzung von
print('Zeilen und Spalten des Datensatzes (nach Ergänzung von `fire_zone`):')
print(df_C.shape)
```

```
Zeilen und Spalten des Datensatzes (vor Ergänzung von `fire_zone`):
(20640, 11)
Zeilen und Spalten des Datensatzes (nach Ergänzung von `fire_zone`):
(20690, 12)
```

Auf Basis der unterschiedlichen Anzahl an Zeilen der Datensätze lässt sich erkennen, dass vereinzelt Latitude-Longitude-Paare in mehreren Feuergefährdungszonen zugeordnet sind. Da diese Situationen nur sehr selten (weniger als 0,5% aller Zeilen) auftreten, entfernen wir pragmatisch doppelte Zeilen des eben erhaltenen Datensatzes, und behalten jeweils nur diejenige Zeile mit dem höchsten Wert der Feuergefährdungszone bei. Dies entspricht einem vorsichtigen Ansatz, welcher aus aktuerieller Sicht angemessen ist.

In [160…

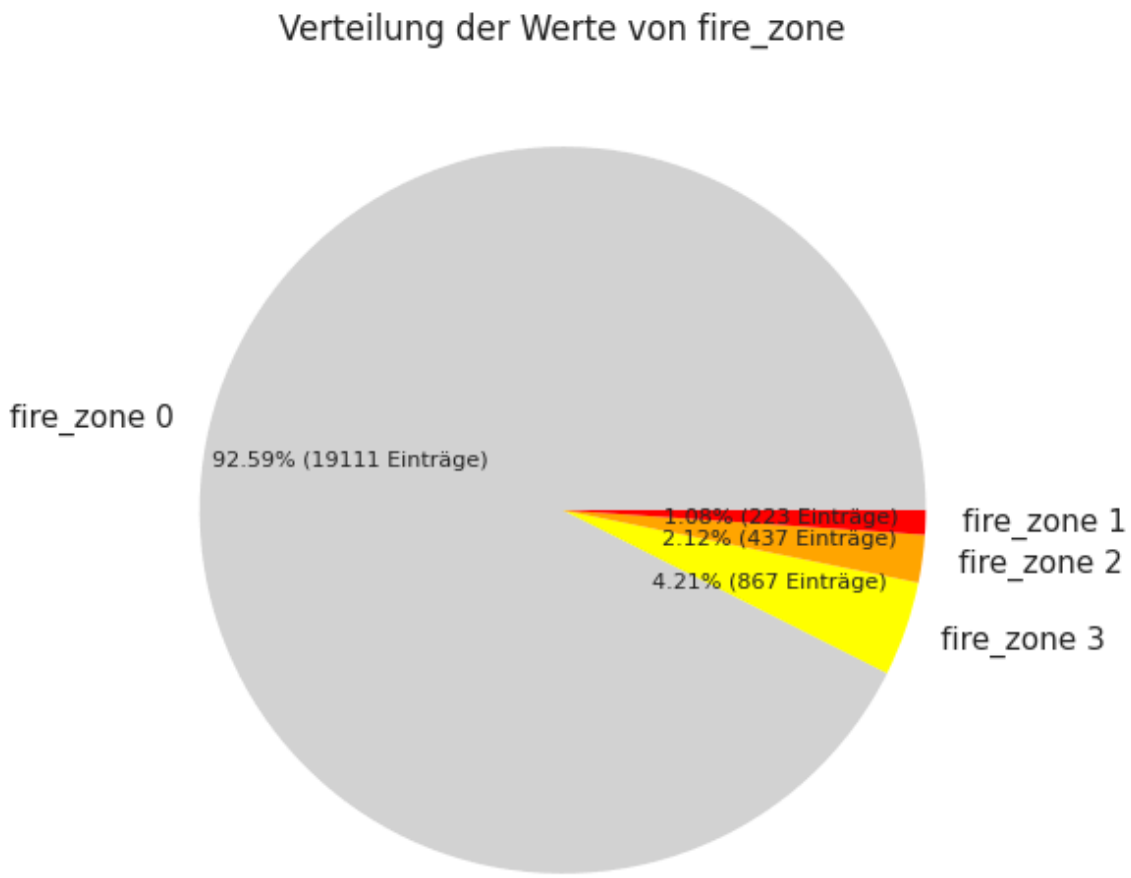
```
# definiere Spalten, die auf Duplikate überprüft werden sollen (alle außer `fire
subset_cols = [col for col in df_C.columns if col != 'fire_zone']

# sortiere DataFrame nach `fire_zone` in absteigender Reihenfolge, sodass höhere
df_C = df_C.sort_values(by='fire_zone', ascending=False)

# behalte bei Duplikaten genau die Zeile mit dem höchsten Wert von `fire_zone``
df_C = df_C.drop_duplicates(subset=subset_cols, keep='first').sort_index()

# Ausgabe der Anzahl von Zeilen und Spalten nach Entfernung von Duplikaten
print('Zeilen und Spalten des Datensatz (nach Entfernung von Duplikaten):')
print(df_C.shape)
```

```
Zeilen und Spalten des Datensatz (nach Entfernung von Duplikaten):
(20640, 12)
```

Nicht direkt gefragt, aber interessant zu wissen, ist, wie oft die einzelnen Feuergefährdungszonen zugeordnet wurden. Zu diesem Zweck erstellen wir ein Tortendiagramm zur Verteilung der Werte von `fire_zone`.

In [161…

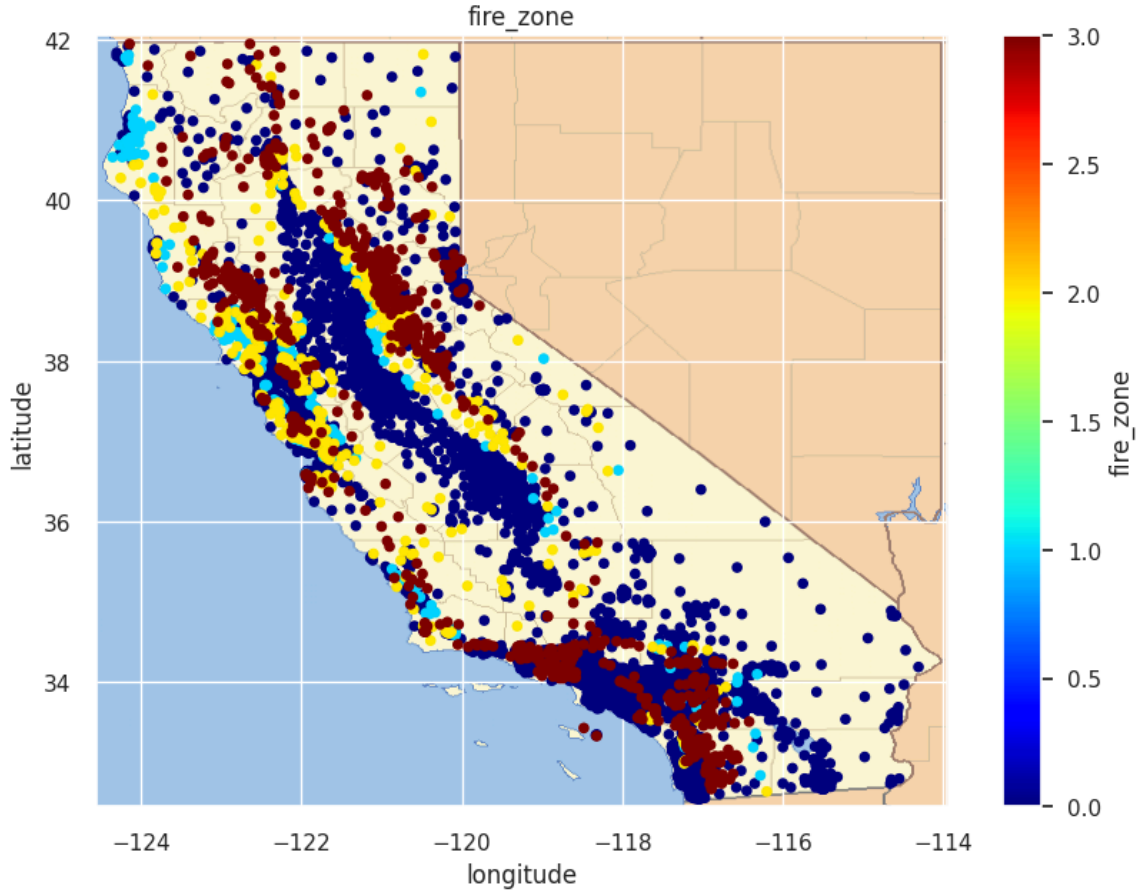
```
# zähle Vorkommen jedes eindeutigen Wertes in der Spalte `fire_zone`
target_counts = df_C['fire_zone'].value_counts()

# berechne Gesamtanzahl der Einträge
total_entries = len(df_C['fire_zone'])

# definiere eine Formatierungsfunktion für die Tortendiagramm-Beschriftungen def
def custom_autopct(pct):
    absolute = int(pct / 100. * total_entries)
    return "{:.2f}% ({:d} Einträge)".format(pct, absolute)

# setze die Plotgröße
plt.figure(figsize=(6, 6))

# erstelle ein Tortendiagramm zur Visualisierung der Verteilung der Werte von `f
pie_chart, _, autotexts = plt.pie(target_counts, autopct=custom_autopct, startan
                                  labels=target_counts.index.map({0: 'fire_zone
                                                                  1: 'fire_zone
                                                                  2: 'fire_zone
                                                                  3: 'fire_zone
                                  colors=['lightgrey', 'yellow', 'orange', 'red'
                                  wedgeprops={'edgecolor': 'none'})

# passe Schriftgröße der Beschriftungen im Tortendiagramm an
for autotext in autotexts:
    autotext.set_fontsize(8)

# erzeuge den Plot samt Titel
plt.title('Verteilung der Werte von fire_zone')
plt.show()
```



Zum Schluss dieser Aufgabe erstellen wir einen Scatter-Plot des erzeugten Datensatzes und färben die einzelnen Punkte gemäß ihrer Feuergefährdungszone ein, dazu verwenden wir die in Teil A eingeführte Funktion `plot_ca`. Im Ergebnis finden wir die Verteilung der Feuergefährdungszonen vom Beginn der Aufgabe wieder.

In [162…

```
# erzeuge einen Scatter-Plot der Daten mit Einfärbung gemäß `fire_zone`,
# wobei die Sortierung höhere Werte dieses Merkmals später plottet
plot_ca(df_C.sort_values(by='fire_zone'), feature_name='fire_zone')
```



Aufgabe C-2: Einsatz von Clustering bei der Tarifierung von Feuerversicherung [Lernziele 4.2, 5.2 & 6; 13 Punkte]

a) Um den aus Aufgabe C-1 erhaltenen Datensatz zu clustern, ist der *k*-Means-Algorithmus anzuwenden. In das Clustering sollen lediglich die Merkmale *latitude*, *longitude*, *median_house_value* und *fire_zone* einfließen. Die Wahl der Cluster-Anzahl ist (etwa durch eine geeignete Heuristik) zu begründen. Ein Scatter-Plot zur Visualisierung der Cluster ist zu erzeugen. b) Um den aus Aufgabe C-1 erhaltenen Datensatz zu clustern, ist das Verfahren Agglomerative Hierarchical Clustering anzuwenden. In das Clustering sollen lediglich die Merkmale *latitude*, *longitude*, *median_house_value* und *fire_zone* einfließen. Die Wahl der Cluster-Anzahl ist (etwa durch eine geeignete Heuristik) zu begründen. Ein Scatter-Plot zur Visualisierung der Cluster ist zu erzeugen. c) Die Clusterings der beiden vorherigen Teilaufgaben sind zu vergleichen, indem auf drei Gemeinsamkeiten oder Unterschiede eingehen wird. d) Ein Ansatz, wie die von einem der Clustering-Verfahren erzeugten Cluster bei der Tarifierung von Feuerversicherung in Kalifornien eingesetzt werden können, ist in zwei bis drei Sätzen zu erläutern.

Lösungsvorschlag

Bevor wir im weiteren Verlauf der Aufgabe die Clustering-Verfahren *k*-Means-Algorithmus und Agglomerative Hierarchical Clustering anwenden, werden wir zunächst deren Input-Daten geeignet skalieren. Diese Skalierung ist notwendig, da unterschiedlich

skalierte Merkmale die Distanzmessung von Datenpunkten und Clustern andernfalls verfälschen würden. Zur Überprüfung geben wir die ersten 10 Zeilen des skalierten Datensatzes aus.

In [163…
```
# speichere die benötigten Feature-Namen in der Variablen `clustering_columns`
clustering_columns = ['latitude', 'longitude', 'median_house_value', 'fire_zone'

# erzeuge eine Kopie des Datensatzes `df_C`, bestehend aus nur den obigen Spalte
df_C_scaled = df_C[clustering_columns].copy(deep=True)

# skaliere den eben erstellten Datensatz mittels `MinMaxScaler`
min_max_scaler = MinMaxScaler()
df_C_scaled[clustering_columns] = min_max_scaler.fit_transform(df_C_scaled)

# gebe 10 zufällige Zeilen aus
pd.DataFrame(df_C_scaled).sample(10, random_state=seed+3)
```

Out[163]:

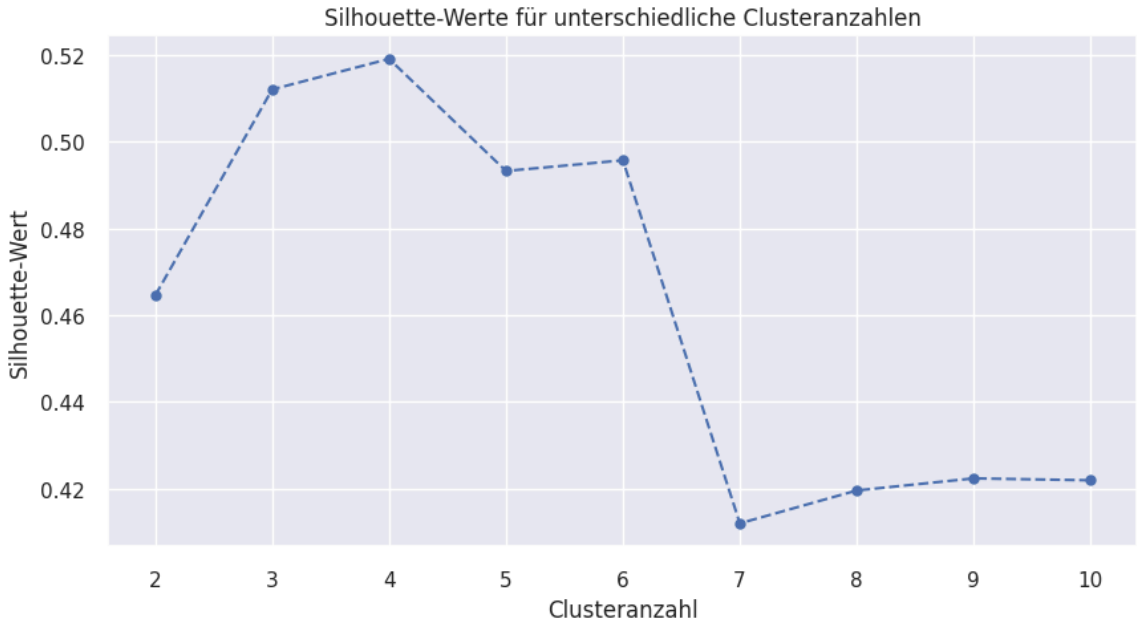
	latitude	longitude	median_house_value	fire_zone
3046	0.521785	0.218127	0.798968	0.000000
6803	0.177471	0.588645	0.316702	0.000000
10112	0.465462	0.256972	0.370722	0.000000
3126	0.854410	0.197211	0.127630	0.000000
5008	0.116897	0.697211	0.194228	0.000000
12772	0.575983	0.202191	0.345980	0.000000
17396	0.155154	0.660359	0.339588	0.000000
18498	0.561105	0.205179	0.123713	0.000000
10378	0.583422	0.534861	0.139177	0.333333
17812	0.454835	0.469124	0.327423	0.000000

Zu a):

Um den k-Means-Algorithmus zum Clustering eines Datensatzes anwenden zu können, muss zunächst die Anzahl an Clustern im Vorfeld festgelegt werden. Zu diesem Zweck verwenden wir die "Silhouette-Methode": Diese Methode misst, wie ähnlich ein Objekt zu seinem eigenen Cluster (Kohäsion) im Vergleich zu anderen Clustern (Separation) ist. Die in diesem Sinne optimale Clusteranzahl ist erreicht, wenn der mittlere Silhouettenkoeffizient für alle Objekte maximal ist, was darauf hindeutet, dass die Objekte gut zu ihren eigenen Clustern passen und gut von anderen Clustern getrennt sind.

In [164…
```
# initialisiere Liste zur Speicherung der Silhouette-Werte
silhouette_scores = []

# berechne und speichere die Silhouette-Werte für unterschiedliche Clusteranzahl
for n in range(2, 11):
    kmeans = KMeans(n_clusters=n, random_state=0)
    kmeans.fit(df_C_scaled)
    score = silhouette_score(df_C_scaled, kmeans.labels_)
    silhouette_scores.append(score)

# erzeuge einen Plot der Silhouette-Werte
```

```
plt.figure(figsize=(10, 5))
plt.plot(range(2, 11), silhouette_scores, marker='o', linestyle='--')
plt.title('Silhouette-Werte für unterschiedliche Clusteranzahlen')
plt.xlabel('Clusteranzahl')
plt.ylabel('Silhouette-Wert')
plt.show()
```



Auf Basis der obigen Silhouette-Werte setzen wir die Anzahl an Clustern für den k-Means-Algorithmus auf 4, da für diese Clusteranzahl der höchste Silhouette-Wert erzielt wird. Im nachfolgenden Code wird der k-Means-Algorithmus zu dieser Clusteranzahl angewendet, die resultierenden Cluster werden dabei in der Spalte `cluster` des Datensatzes `df_kMeans` gespeichert.

In [165…

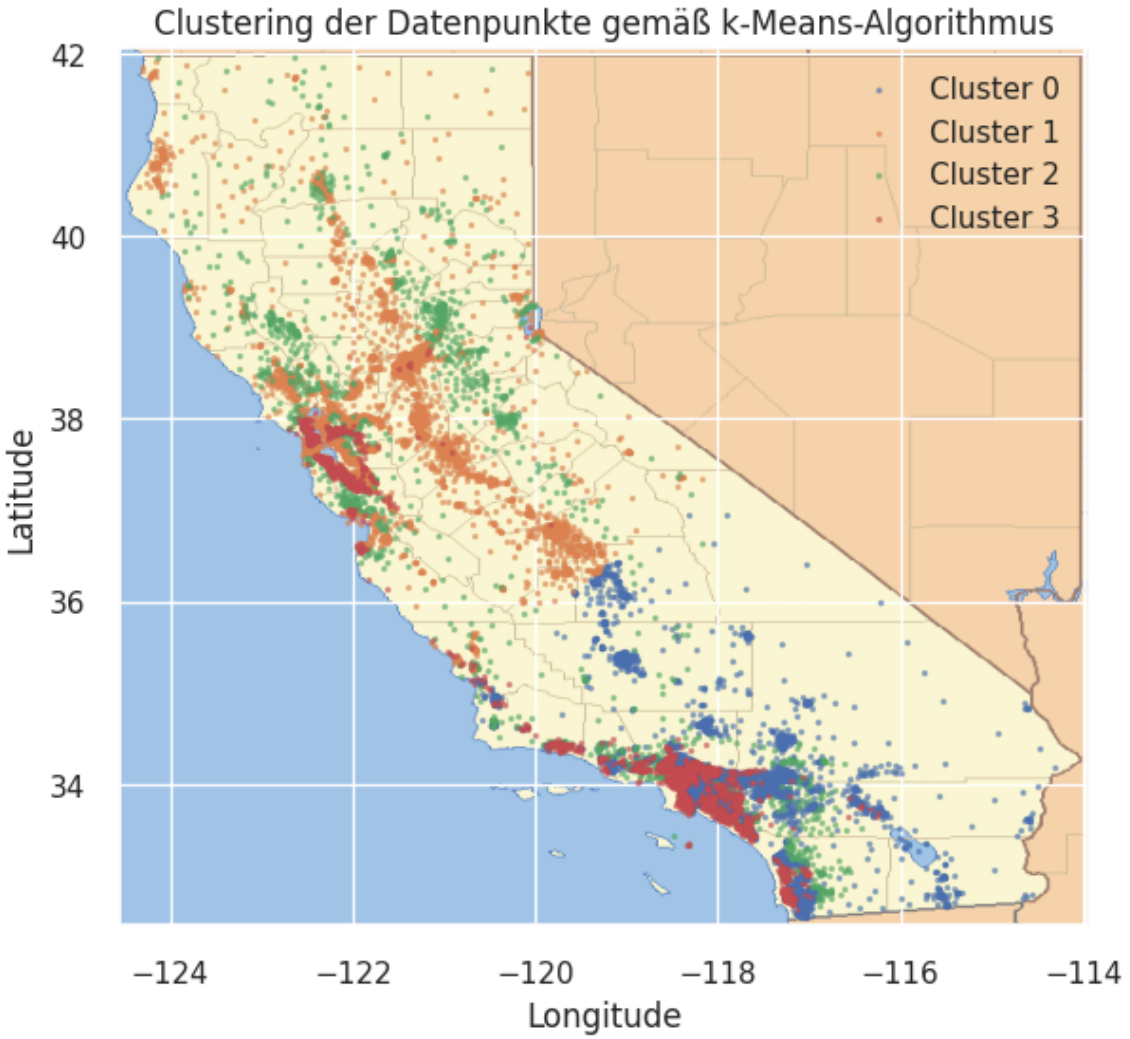
```
# wende den k-Means-Algorithmus mit Clusteranzahl 4 an
num_clusters_kMeans = 4
kmeans = KMeans(n_clusters=num_clusters_kMeans, random_state=0)
kmeans.fit(df_C_scaled)

# füge die zugeordneten Cluster dem Datensatz hinzu
labels = kmeans.labels_
df_kMeans = df_C.copy(deep=True)
df_kMeans['cluster'] = labels
```

Schließlich erzeugen wir einen Scatter-Plot der Latitude-Longitude-Paare und färben diese gemäß der durch den k-Means-Algorithmus erhaltenen Clusterung ein.

In [166…

```
# erzeuge einen Scatter-Plot der Latitude-Longitude-Paare
# mit Clustering gemäß k-Means-Algorithmus
plt.figure(figsize=(10, 6))
for cluster_number in range(num_clusters_kMeans):
    cluster_data = df_C[df_kMeans['cluster'] == cluster_number]
    plt.scatter(cluster_data['longitude'], cluster_data['latitude'], s=2,
                label=f'Cluster {cluster_number}', cmap='viridis', alpha=0.5)

# plotte eine Karte von Kalifornien im Hintergrund
california_img = plt.imread(IMAGES_PATH / filename)
axis = -124.55, -113.95, 32.45, 42.05
plt.imshow(california_img, extent=axis)

# ergänze Titel und Achsenbeschriftungen
```

```
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Clustering der Datenpunkte gemäß k-Means-Algorithmus')
plt.legend(framealpha=1, facecolor='white')
plt.show()
```



Zur Überprüfung des Clusterings ließen sich auch noch dreidimensionale Plots betrachten, etwa zu den Merkmalen `latitude`, `longitude` und `median_house_value` oder `latitude`, `longitude` und `fire_zone`. Nachfolgend wird ein solcher Plot für die ersten drei genannten Merkmale erzeugt.

In [167…

```
# erzeuge einen 3D-Scatterplot zu den Merkmalen `latitude`, `longitude` und
# `median_house_value`, eingefärbt nach dem zugeordneten Cluster
fig = px.scatter_3d(df_kMeans, x="longitude", y="latitude", z="median_house_valu
                    title=f"3D-Scatterplot der Cluster aus dem k-Means-Algorithm
fig.update_traces(marker_colorscale='viridis', selector=dict(type='scatter_3d'))
fig.show()
```

3D-Scatterplot der Cluster aus dem k-Means-Algorithmu

Im obigen 3D-Scatterplot ist neben einer Nord-Süd-Trennung auch eine Separierung nach hohen und niedrigen Werten von `median_house_value` erkennbar.

Zu b):

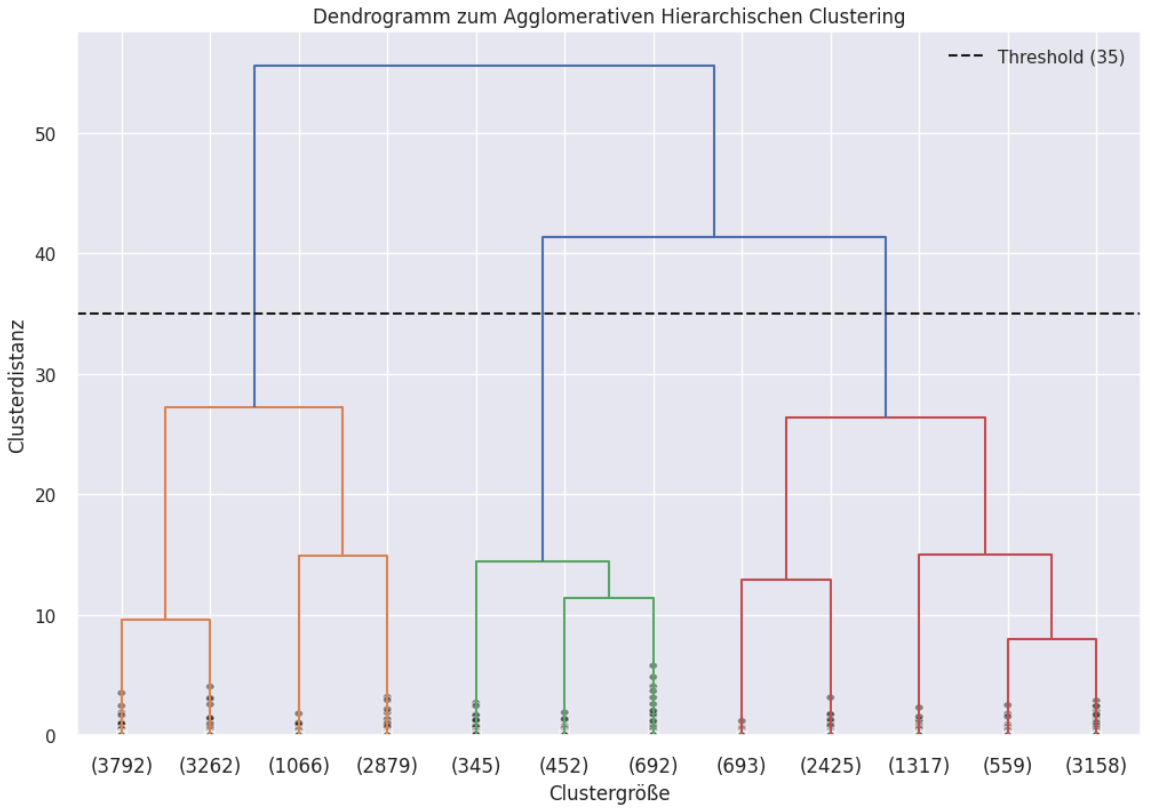
Im Vergleich zum k-Means-Algorithmus besitzt das Agglomerative Hierarchische Clustering den Vorteil, dass die Anzahl an Clustern nicht a priori festgelegt werden muss. Wir wenden daher das Agglomerative Hierarchische Clustering unter Verwendung der Ward-Linkage und der euklidischen Distanz als Metrik an.

In [168…
```
# wende Agglomeratives Hierarchisches Clustering mit
# Ward-Linkage und euklidischer Distanz als Metrik an
Z = linkage(df_C_scaled, method='ward', metric='euclidean')
```

Anschließend werfen wir einen Blick auf das zugehörige Dendrogramm, da sich mithilfe dieser grafischen Repräsentation der einzelnen Clusterbildungen über verschiedene Heuristiken eine möglichst gute Wahl der Clusteranzahl treffen lässt. Wir setzen einen horizontalen Cut auf Höhe 35 an, welcher drei vergleichsweise lange vertikale Linien schneidet und somit auf eine Wahl von 3 als Clusteranzahl hinweist.

In [169…
```
# erzeuge ein Dendrogramm zum Agglomerativen Hierarchischen Clustering
plt.figure(figsize=(12, 8))
dendrogram(Z, color_threshold=None, truncate_mode='lastp', p=12, show_contracted
```

```
# ergänze Titel und Achsenbeschriftungen
plt.title('Dendrogramm zum Agglomerativen Hierarchischen Clustering')
plt.xlabel('Clustergröße')
plt.ylabel('Clusterdistanz')

# ergänze Threshold-Linie zur Auswahl der Anzahl an Clustern
example_threshold = 35
plt.axhline(y=example_threshold, color='k', linestyle='--', label=f'Threshold ({
plt.legend()
plt.show()
```

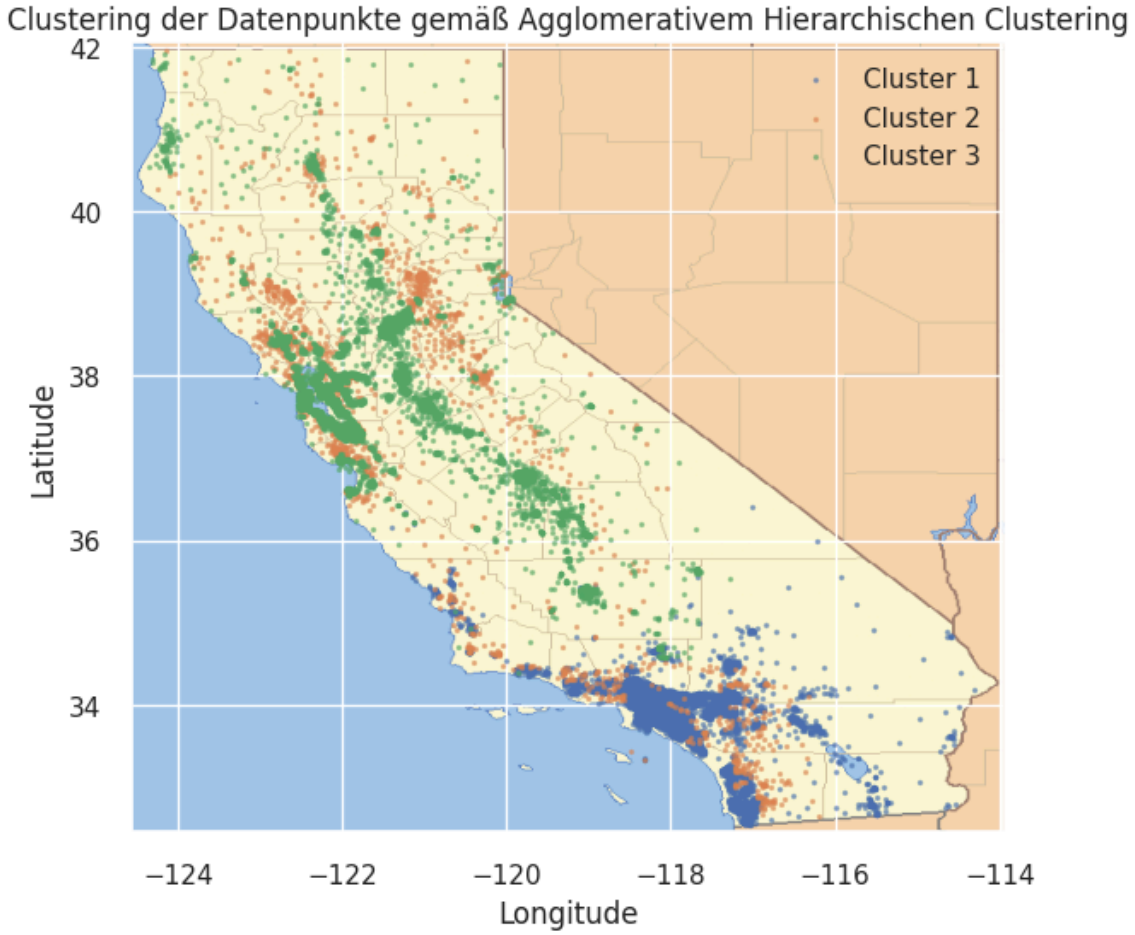


Abschließend weisen wir allen Punkten unseres Datensatzes `df_C` ihre aus dem Agglomerativen Hierarchischen Clustering stammenden Cluster zu und erzeugen hierzu einen Scatter-Plot.

In [170…

```
# schreibe den einzelnen Zeilen ihre Cluster gemäß dem Agglomerativen Hierarchis
num_clusters_hierarchical = 3
df_agglo = df_C.copy(deep=True)
df_agglo['cluster'] = fcluster(Z, num_clusters_hierarchical, criterion='maxclust

# erzeuge einen Scatter-Plot der Latitude-Longitude-Paare
# mit Clustering gemäß Agglomerativem Hierarchischen Clustering
plt.figure(figsize=(10, 6))
for cluster_number in range(1, num_clusters_hierarchical + 1):  # Cluster labels
    cluster_data = df_agglo[df_agglo['cluster'] == cluster_number]
    plt.scatter(cluster_data['longitude'], cluster_data['latitude'],
                label=f'Cluster {cluster_number}', s=2, alpha=0.5)

# plotte eine Karte von Kalifornien im Hintergrund
california_img = plt.imread(IMAGES_PATH / filename)
axis = -124.55, -113.95, 32.45, 42.05
plt.imshow(california_img, extent=axis)

# ergänze Titel und Achsenbeschriftungen
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Clustering der Datenpunkte gemäß Agglomerativem Hierarchischen Cluste
```

```
plt.legend(framealpha=1, facecolor='white', edgecolor='black')
plt.show()
```

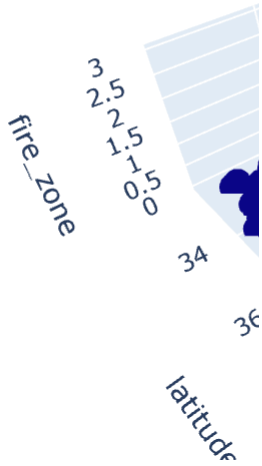


Zur Überprüfung des Clusterings ließen sich auch hier noch dreidimensionale Plots betrachten. Dieses Mal erzeugen wir einen 3D-Scatterplot zu den Merkmalen `latitude`, `longitude` und `fire_zone`.

In [171…

```
# erzeuge einen 3D-Scatterplot zu den Merkmalen `latitude`, `longitude` und
# `fire_zone`, eingefärbt nach dem zugeordneten Cluster
fig = px.scatter_3d(df_agglo, x="longitude", y="latitude", z="fire_zone", color=
                    title=f"3D-Scatterplot der Cluster aus dem Agglomerativen Hi
fig.update_traces(marker_colorscale='viridis', selector=dict(type='scatter_3d'))
fig.show()
```

3D-Scatterplot der Cluster aus dem Agglomerativen Hier



Auch in diesem 3D-Scatterplot erkennen wir eine Nord-Süd-Teilung, allerdings nur für diejenigen Datenpunkte mit Feuerzone 0. Die Datenpunkte zu den restlichen Feuerzonen erstrecken sich über ganz Kalifornien. Aufgrund dieser Beobachtungen erscheint die gewählte Clusteranzahl als zu niedrig.

Zu c):

Aus den Scatter-Plots der beiden Clustering-Verfahren lassen sich u. a. folgende Erkenntnisse gewinnen:

- Die Anzahlen an Clustern, die bei den beiden Clustering-Verfahren verwendet werden, unterscheiden sich im vorliegenden Fall (muss nicht so sein; hängt u. a. von der verwendeten Heuristik zur Bestimmung der Cluster-Anzahl ab).
- Die Nord-Süd-Trennung ist in beiden Clustering-Verfahren deutlich zu erkennen, was auf eine ausgeprägte geografische Segmentierung hindeutet.
- Differenzierungen nach `median_house_value` bzw. nach `fire_zone` sind bei den beiden Clustering-Verfahren unterschiedlich stark ausgeprägt. Während das erhaltene Clustering aus dem k-Means-Algorithmus eine feinere Aufteilung nach `median_house_value` erlaubt, zeigt das aus dem Agglomerativen Hierarchischen Clustering stammende Clustering einen stärkeren Fokus auf die Segmentierung nach Feuergefährdungszonen.

Zu d):

In einem Schadentarif, beispielsweise für die Feuerversicherung, wird die Prämie basierend auf der Tarifzelle bestimmt, der das zu versichernde Risiko zugeordnet ist. Eine Tarifzelle umfasst eine Kombination von Risikomerkmalen, für die eine einheitliche Prämie festgelegt wird. Das Clustering, wie es in dieser Aufgabe durchgeführt wurde, kann als Grundlage für die Definition dieser Tarifzellen dienen, indem es ähnliche Risikogruppen identifiziert und somit eine genauere und gerechtere Tarifierung ermöglicht.

Laufzeitmessung abschließen:

In [172…
```
# Zeit Notebook: Ende Teil B, Start Teil D
startzeit_teil_d = time.time()
laufzeit_teil_c = startzeit_teil_d - startzeit_teil_c
laufzeit_teil_abc = startzeit_teil_d - startzeit_teil_a

# Laufzeit anzeigen
runtime_abc_df.loc['C'] = [laufzeit_teil_c]
runtime_abc_df.loc['ABC'] = [laufzeit_teil_abc]
print(runtime_abc_df)
```

```
         Laufzeit
Teil
A       64.397617
B     2887.858738
C      149.950109
ABC   3102.206465
```

Teil D: GPU-Ausführung und Vergleiche [25 Punkte]

Aufgabe D-1: Ergebnisse des CPU-Notebooks persistieren [Lernziele 3.3/3.4; 2 Punkte]

Dieser Aufgabenteil ist nur im CPU-Notebook (siehe Hinweise) zu erledigen. Im GPU-Notebook sind die Programmzeilen (Code-Boxen) dieses Aufgabenteil zu entfernen oder auf andere Weise unwirksam zu stellen. a) Die in Aufgabe A-0 genannten und im Laufe der Aufgabenlösung befüllten Datenstrukturen für Laufzeiten und Performance sind in csv-Dateien permanent zu sichern. b) Die beiden in Aufgabe B-3 erstellten und gefitteten Neuonalen Neze mit Embeddings sind im HDF5-Datenformat zu sichern. Gegebenenfalls ist bereits bei der Wahl des Sicherungsortes dafür Sorge zu tragen, dass die gesicherten Dateien im GPU-Notebook wieder eingelesen werden können.

Lösungsvorschlag

Zu a):

In [173…
```
# Laufzeiten und Performance sichern (in CPU Notebook)
# runtime_abc_df.to_csv("runtime_abc.csv", index=True)
```

```
# results_df.to_csv("results_rmse_time.csv", index=True)
```

Zu b):

In [174…

```
# https://www.geeksforgeeks.org/save-and-load-models-in-tensorflow/

# Modelle mit Embeddings speichern (für D)
# modelNNemb.save('save_model_NNemb.h5')
# modelNNemb_cloned.save('save_model_NNemb_cloned.h5')
```

Aufgabe D-2: GPU-Notebook erstellen, ändern und ausführen [Lernziele 2.2 & 3.3/3.4; 8 Punkte]

a) Nach Abschluss der Arbeiten am CPU-Notebook (Teile A, B, C und D-1) ist eine Kopie des CPU-Notebooks in einer geeigneten GPU-Umgebung zu erstellen und ggf. erforderliche Anpassungen sind durchzuführen. b) In Aufgabe A-0 ist die GPU-memory-allocation von TensorFlow anpassen und in den Aufgaben B-2, B-3 und B-4 ist die GPU-Ausführung geeignet zu aktivieren bzw. sicherzustellen. Optional kann auch CatBoost auf GPU ausgeführt werden, dann aber einheitlich bei allen Anwendungen c) Die Modellnamen in den Aufgaben B-2 und B-3 sind zu ändern und das Notebook ist in der GPU-Umgebung gemäß a) unter Nutzung von CPU- und GPU-Ressourcen ausführen. Hinweis: Die Bearbeitung der oben genannten Aufgabenteile erfordert keine Kommentierung oder Programmierung an dieser Stelle, sondern Änderungen in den Teilen A und B im GPU-Notebook.

Lösungsvorschlag

Siehe oben

Aufgabe D-3: Vergleich der Ergebnisse und Erfahrungen [Lernziele 2.2, 3.3/3.4 & 6; 15 Punkte]

a) Die zur Verfügung stehenden Resourcen der CPU- und GPU-Umgebung sind zu nennen und zu vergleichen. Bei Nutzung einer öffentlich zugänglichen Cloud-Umgebung sind diese Angaben mit einer Quellenangabe zu versehen. b) Die in Aufgabe D-1 gespeicherten Ergebnisse (Performance, Laufzeiten) und Modelle des CPU-Notebooks sind einzulesen (ggf. müssen diese Daten zuvor für das GPU-Notebook verfügbar gemacht werden). c) Die vier CPU-Embeddings sind zu extrahieren und zusammen mit den vier GPU-Embeddings anzeigen (je ein "Kleeblatt" für County und Ocean_Proximity), analysieren und die Ursachen beobachteter Unterschiede sind zu beleuchten. d) Die Laufzeiten und Performance der CPU- und GPU-Ausführung sind anschaulich zu vergleichen und die Unterschiede sind zu interpretieren. e) Als Gesamtfazit sind Empfehlungen hinsichtlich der verwendeten Modelle, grafischen Darstellungen und Rechenumgebungen (CPU/GPU) abzuleiten.

Lösungsvorschlag

Zu a):

Verwendete CPU- und GPU-Ressourcen:

Sowohl das CPU- als auch das GPU-Notebook wurden in der kostenlosen Cloud-Umgebung für registierte Nutzer auf Kaggle.com ausgeführt. Aktuell stehen dort folgende Ressourcen zur Verfügung (weitere Details siehe z.B. https://www.kaggle.com/code/bconsolvo/hardware-available-on-kaggle):

- In der CPU-Umgebung stehen vier vCPU-Kerne und 32 GB RAM ohne Zeitlimit zur Verfügung.
- In der der GPU-Umgebung wurde eine NVIDIA TESLA P100 GPU mit 16 GB RAM verwendet, die 30 Stunden je Woche genutzt werden kann, siehe https://www.kaggle.com/page/GPU-tips-and-tricks . Bei Nutzung der GPU-Umgebung reduziert sich die CPU-Anzahl auf zwei vCPUs.

Während der interaktiven Ausführung des GPU-Notebooks hat sich gezeigt, dass die GPU bei diesem vergleichsweise kleinen Datensatz zu weniger als 20% ausgelastet war und daher nicht das volle Beschleunigungspotential zeigen konnte.

Zu b):

Ergebnisse und Modelle des CPU-Notebooks einlesen:

In [175…
```
# Laufzeiten und Performance des CPU-Notebooks einlesen
runtime_abc_cpu = pd.read_csv('../input/ads3-2024-results-cpu/runtime_abc.csv')
results_cpu = pd.read_csv('../input/ads3-2024-results-cpu/results_rmse_time.csv'
```

In [176…
```
# Modelle mit Embeddings des CPU_Notebooks einlesen
modelNNemb_cpu = tf.keras.models.load_model('../input/ads3-2024-results-cpu/save
modelNNemb_cloned_cpu = tf.keras.models.load_model('../input/ads3-2024-results-c
```

Zu c):

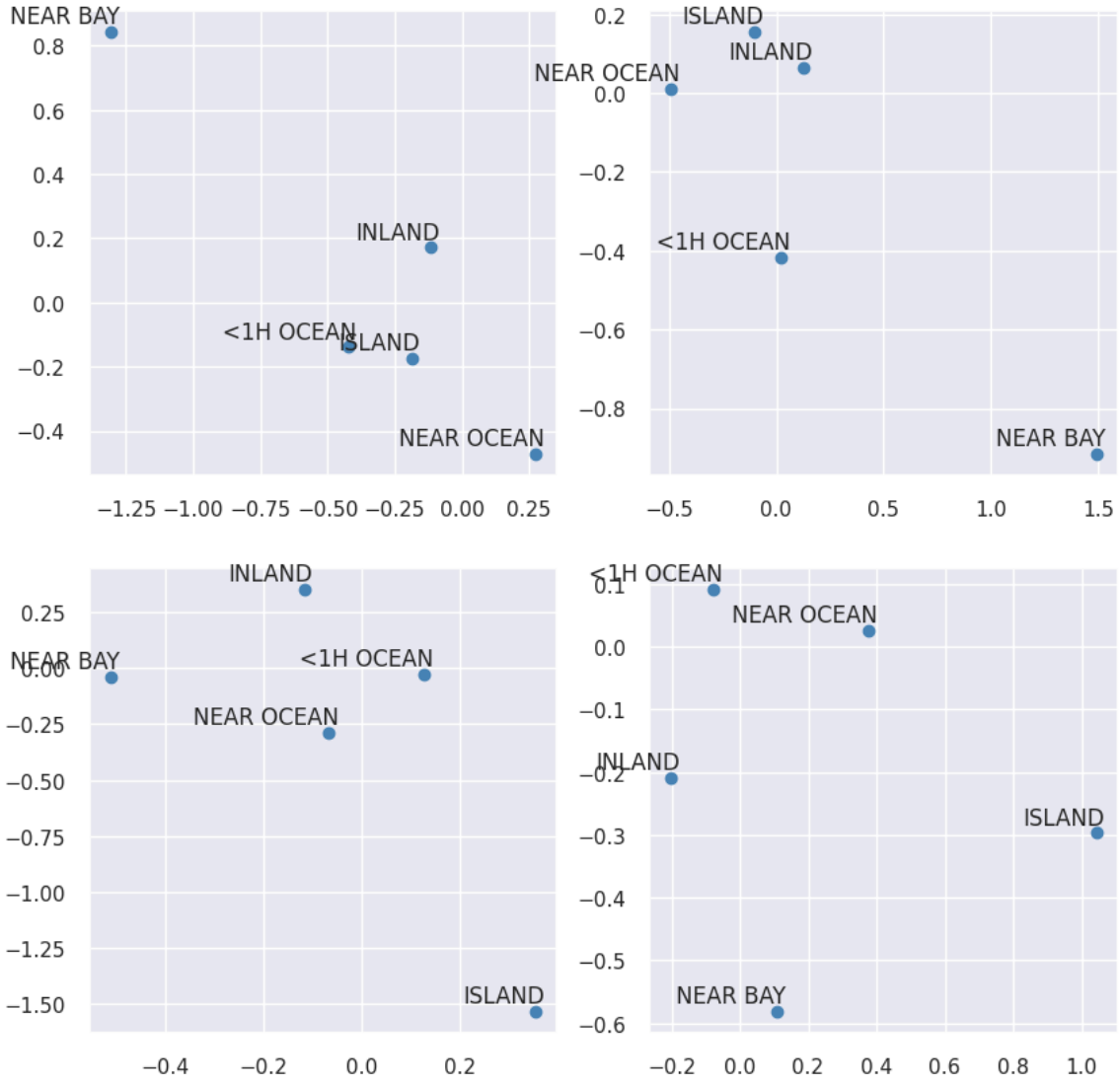
CPU-Embeddings extrahieren und mit den GPU-Embeddings vergleichen:

In [177…
```
# Embeddings der CPU-Modelle extrahieren: get layer
ocean_proximity_layer_cpu = modelNNemb_cpu.get_layer('ocean_proximity_embedding'
county_name_layer_cpu = modelNNemb_cpu.get_layer('county_name_embedding')
ocean_proximity_layer_cloned_cpu = modelNNemb_cloned_cpu.get_layer('ocean_proxim
county_name_layer_cloned_cpu = modelNNemb_cloned_cpu.get_layer('county_name_embe

# .. get weights
ocean_proximity_embeddings_result_cpu = ocean_proximity_layer_cpu.get_weights()[
county_name_embeddings_result_cpu = county_name_layer_cpu.get_weights()[0]
ocean_proximity_embeddings_result_cloned_cpu = ocean_proximity_layer_cloned_cpu.
county_name_embeddings_result_cloned_cpu = county_name_layer_cloned_cpu.get_weig
```

1. "Ocean Proximity" Embeddings: Oben CPU, Unten GPU, rechts Wiederholung ("Cloned")

In [178…

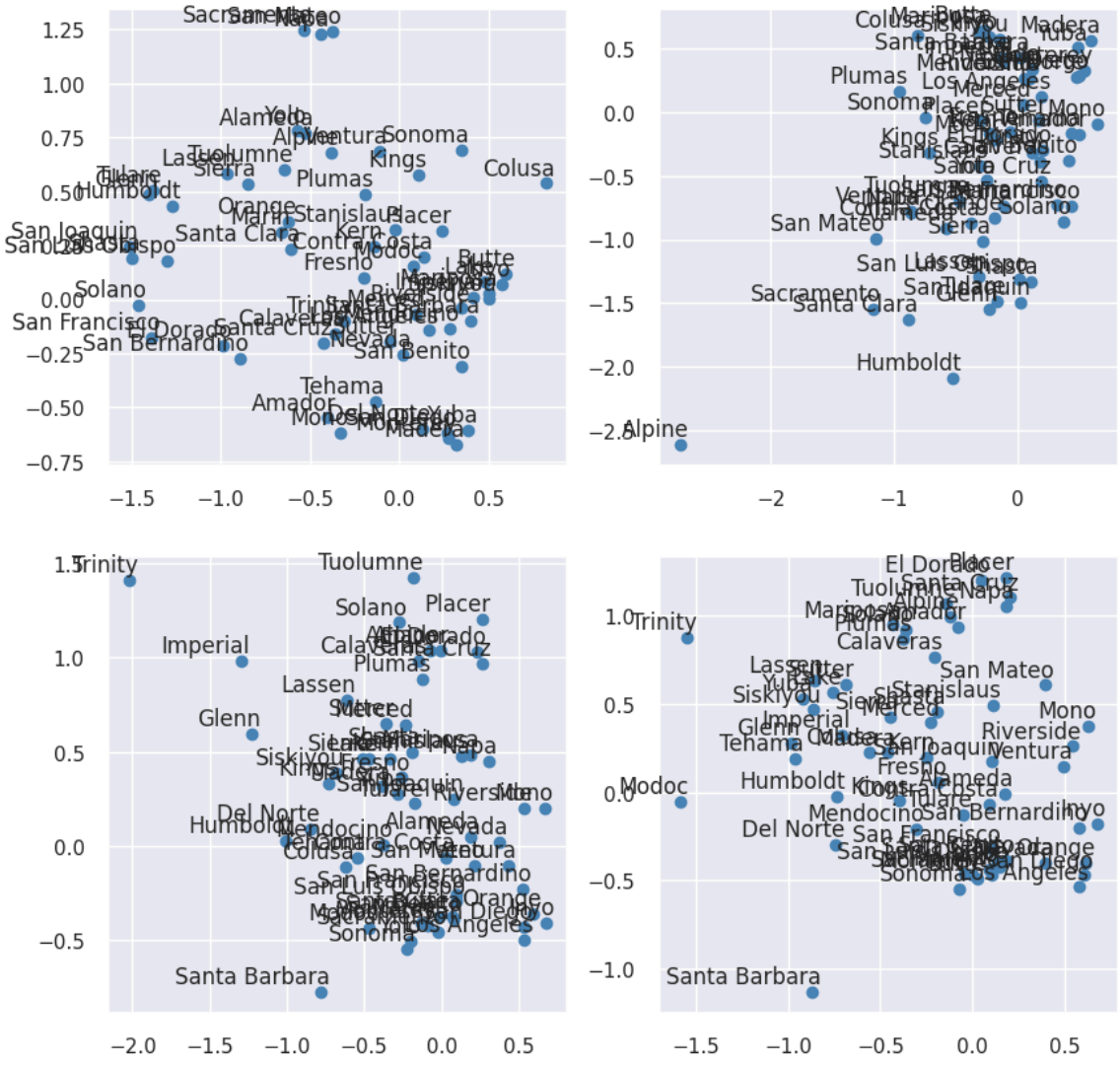
```
fig, axs = plt.subplots(nrows = 2, ncols = 2, figsize= (10, 10))
plot_multiple_embeddings(ocean_proximity_embeddings_result_cpu, ocean_proximity_
plot_multiple_embeddings(ocean_proximity_embeddings_result_cloned_cpu, ocean_pro
plot_multiple_embeddings(ocean_proximity_embeddings_result, ocean_proximity_labe
plot_multiple_embeddings(ocean_proximity_embeddings_result_cloned, ocean_proximi
```



1. "County" Embeddings: Oben CPU, Unten GPU, rechts Wiederholung ("Cloned")

In [179…

```
fig, axs = plt.subplots(nrows = 2, ncols = 2, figsize= (10, 10))
plot_multiple_embeddings(county_name_embeddings_result_cpu, county_name_labels,
plot_multiple_embeddings(county_name_embeddings_result_cloned_cpu, county_name_l
plot_multiple_embeddings(county_name_embeddings_result, county_name_labels, 'GPU
plot_multiple_embeddings(county_name_embeddings_result_cloned, county_name_label
```



Die Embeddings zeigen das bereits aus Aufgabe B-3 bekannte Muster: In der Visualisierung der entsprechenden Gewichte des Neuronalen Netzes für "ocean_proximity" ist meist gut die Besonderheit der Insellage und als Gegenpol die Bay Area (um San Francisco) als anderes Achssenende zu erkennen. Rotationen und Spiegelungen können hier ebenfalls beobachtet werden. Das Embedding für "county_name" ist hingegen aufgrund der bereits recht hohen Anzahl an Kategorien deutlich unübersichtlicher und die Orientierungs- bzw. Randpunkte Trinity und Santa Barbara nur in der gpu-Version deutlich erkennbar.

Ursächlich für die unterschiedlichen Embeddings ist hier die Initialisierung der Gewichte mit Zufallszahlen, die bei jedem neuen "Training" des Neuronalen Netzes zu anderen Gewichten und damit zu einem zumindest leicht anderen Ergebnis führt.

Zu d):

CPU- vs. GPU-Ausführung: Laufzeiten und Performance

1. Laufzeit der Teile des Notebooks

In [180…

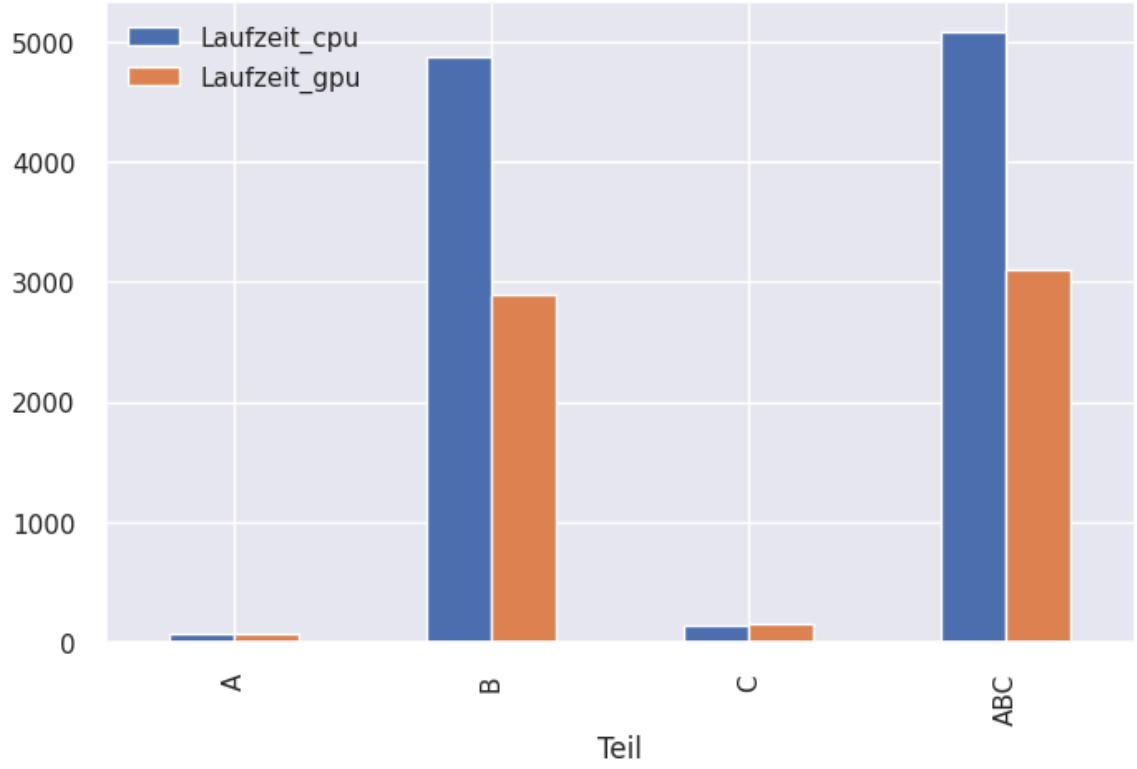
```
# Kennzahlen des CPU- und GPU-Notebooks zusammenführen
runtime_abc = runtime_abc_cpu.merge(runtime_abc_df, on = 'Teil', suffixes=['_cpu
runtime_abc
```

Out[180]:

	Teil	Laufzeit_cpu	Laufzeit_gpu
0	A	64.277292	64.397617
1	B	4873.301997	2887.858738
2	C	135.403725	149.950109
3	ABC	5072.983015	3102.206465

In [181…

```
# Laufzeit der Teile des Notebooks
runtime_abc.plot(x="Teil", y=["Laufzeit_cpu", "Laufzeit_gpu"], kind="bar", figsi
plt.legend(loc='upper left')
plt.show()
```



Kommentierung und Interpretation (1): Der mit weitem Abstand größte Teil der Laufzeit wird erwartungsgemäß in Teil B verbraucht. Der Unterschied zwischen der CPU- und GPU-Laufzeit ist zwar deutlich, aber weniger groß als vor dem Hintergrund der im Aufgabenteil a) genannten Ressourcen erwartet. Mögliche Ursachen werden bei der folgenden, modellbezogenen Betrachtung diskutiert.

2. Laufzeiten und Performance der Modelle

In [182…

```
# Kennzahlen des CPU- und GPU-Notebooks zusammenführen
results = results_cpu.merge(results_df, on = 'Model', suffixes=['_cpu','_gpu']).
results
```

Out[182]:

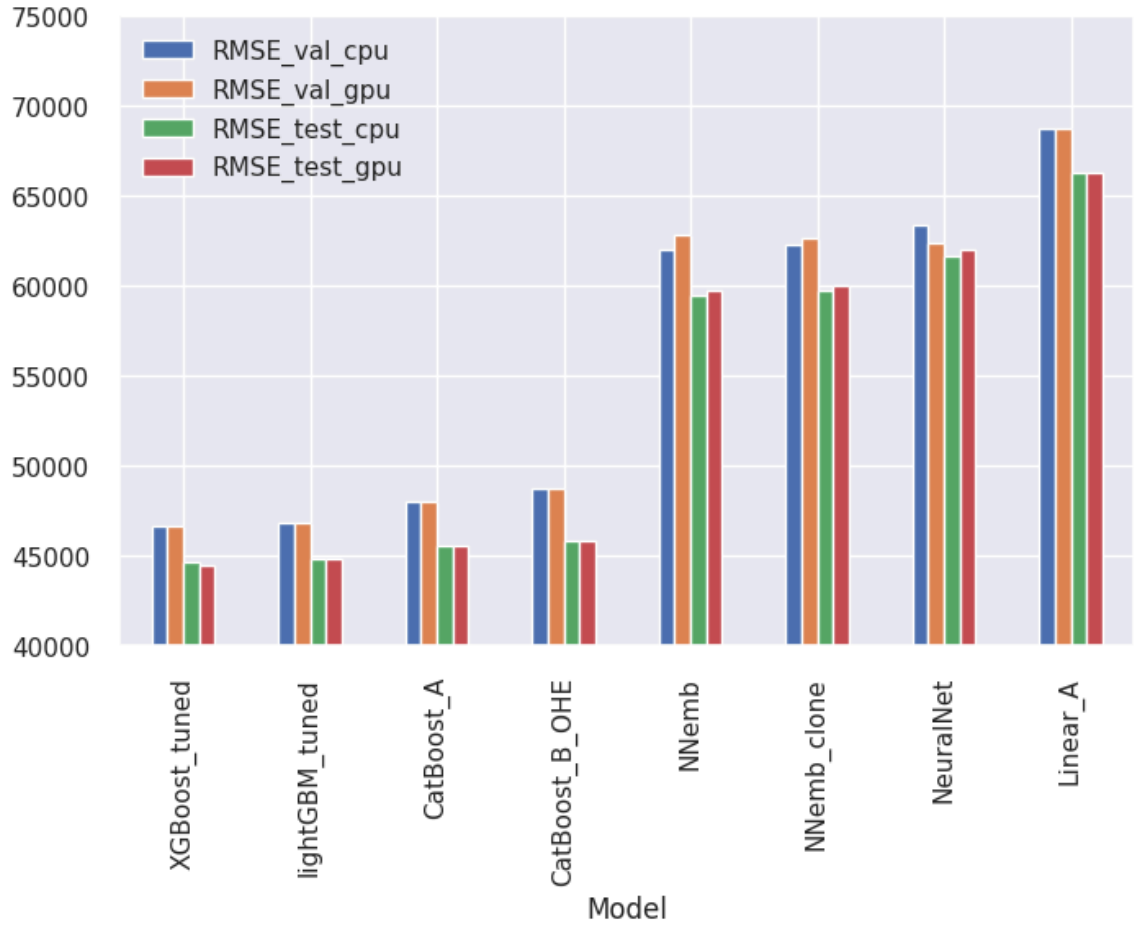
	Model	time_train_cpu	RMSE_val_cpu	RMSE_test_cpu	time_train_gpu	RMSE_val_gp
0	XGBoost_tuned	934.061739	46674.902885	44642.088009	329.775272	46684.17340
1	lightGBM_tuned	732.927880	46836.319829	44800.200369	386.166260	46861.06661
2	CatBoost_A	8.026004	48006.829301	45613.780729	6.781105	48006.82930
3	CatBoost_B_OHE	3.398816	48783.757576	45829.669824	3.225582	48783.75757
4	NNemb	782.138305	61993.399600	59494.732299	503.869339	62817.44671
5	NNemb_clone	784.429478	62313.580070	59796.656707	501.270618	62703.77897
6	NeuralNet	773.078974	63399.066487	61656.717525	615.197549	62381.04441
7	Linear_A	0.179327	68709.117110	66261.754099	0.167730	68709.11711

Kommentierung und Interpretation (2): Zunächst fällt auf, dass in diesem Vergleich die Berechnung der beiden CatBoost-Modelle sowie die des Linearen Modells eine nahezu verschwindend geringe Laufzeit benötigt hat und fast die gesamte Rechenzeit für die neuronalen Netze und die Hyperparameteroptimierung von LightGBM und XGBoost erforderlich war. Das HP-Tuning von XGBoost per Random-Search ist bei gleicher Anzahl an durchlaufenen Parametersätzen in der CPU-Umgebung langsamer als das HP-Tuning von LightGBM per Grid-Search erfolgt, konnte aber stärker von der GPU-Unterstützung profitieren und ist im GPU-Vergleich nun sogar das schnellste der fünf rechenzeitintensiven Modelle (GBMs und NNs). Die NNs mit Embedding konnten (vermutlich aufgrund der größeren batch_size und damit Auslastung) etwas stärker von der GPU-Unterstützung als das NN ohne Embedding profitieren, insgesamt aber deutlich weniger als die GBM-Modelle. Dies ist zunächst etwas überraschend, da Graphikkarten als unumgängliche Beschleuniger bei (großen) neuronalen Netzen gelten, ist aber vermutlich auf die bereits im Aufgabenteil a) genannte geringe Auslastung der GPU zurückzuführen.

Insgesamt hängen die Laufzeitergebnisse natürlich stark von den zur Verfügung stehenden Rechenkapazitäten ab. Bereits mit einem moderat leistungsfähigen PC ist mit einer schnelleren CPU-Ausführung als in der hier verwendeten Umgebung zu rechnen und sind entsprechend andere Beobachtungen hinsichtlich der Geschwindigkeitsvorteile CPU/GPU bei der vorliegenden Aufgabe möglich.

In [184…

```
# Performance der Modelle (CPU/GPU, val/test)
results.plot(x="Model", y=["RMSE_val_cpu", "RMSE_val_gpu", "RMSE_test_cpu", "RMS
plt.legend(loc='upper left')
plt.show()
```



Kommentierung und Interpretation (3): Hinsichtlich des Messfehlers RMSE (Modellgüte) bei der Prognose mittlerer Hauspreise in Kalifornischen Wahlbezirken (in USD im Jahr 1992) sind in der Grafik zwei Niveaus auszumachen. Auf der linken Seite die Gradient-Tree-Boosting-Tools, die zu einem sehr ähnlichen, niedrigen Messfehler (RMSE) um 45.000 USD führen und rechts die Regressionsmodelle (Linear, NN), deren Messfehler fast 20.000 USD höher ist. Die Prognosegüte der Modelle fällt bei den Testdaten jeweils etwas besser aus als bei den Validierungsdaten und unterscheidet sich zwischen CPU- und GPU-Ausführung nur wenig oder gar nicht. Insgesamt ist XGBoost das beste Modell und erzielt wie auch das zweitbeste Modell LightGBM bei den Testdaten einen Meßfehler von unter 45.000 USD. CatBoost_A folgt knapp dahinter und erzielt dieses hervorragende Ergebnis geradezu mühelos (für Coder & Computer).

Zu e):

Als Gesamtfazit können folgende Empfehlungen abgeleitet werden:

Die Nutzung der GPU-Umgebung zusätzlich zur CPU-Umgebung hat dazu beigetragen, dass die Performance der Modelle und die Stabilität der Embeddings weiter beleuchtet werden konnte. Der eigentlich erwünschte Effekt einer massiven Beschleunigung der Berechnungen ist (auch mangels vollständiger Auslastung der GPU) hier nicht in vollem Umfang eingetreten. Während das HP-Tuning bei XGBoost durch die GPU-Nutzung immerhin dreifach beschleunigt werden konnte, hatte die GPU-Unterstützung bei den Neuronalen Netzen mit Embeddings nur einen sehr geringen Beschleunigungseffekt. Das kann in einer anderen CPU- und GPU-Umgebung, mit anderen NN-Architekturen oder

größeren Datensätzen jedoch ganz anders ausfallen und kann daher nicht verallgemeinert werden.

Bezüglich der grafischen Darstellungen konnte im gesamten Notebook gezeigt werden, dass mit Python in einer Notebook-Umgebung sowohl typische Datenvisualisierungen als auch sehr spezielle geografische Darstellungen und sogar professionelle Karten auf Basis öffentlich verfügbarer Polygondaten einfach und gut umgesetzt werden können und hierfür keine Spezialsoftware nötig ist.

Wie bereits bei früheren Prüfungsaufgaben für "Actuarial Data Science Immersion" erweisen sich auch hier die Gradient-Boosting-Tools CatBoost, LightGBM und XGBoost als eine sehr gute Wahl für beste Prognosemodelle auf Basis tabularer Daten. Als besonders empfehlenswert erweist sich CatBoost, da es mit minimalem Datenaufbereitungsaufwand und ohne weiteres Tuning in sehr kurzer Zeit zuverlässig ein sehr gutes Prognosemodell erzeugt.