workshop_data_mining

April 20, 2020

1 Data Mining

- Studiengang: Wirtschaftsingenieurwesen (6. Semester)
- Dozent: Tin Votan
- Datum: 21.04.2020

1.1 1. Python-Module importieren

[1]:
```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "end_to_end_project"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
```

```
    plt.savefig(path, format=fig_extension, dpi=resolution)

# Ignore useless warnings (see SciPy issue #5998)
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")
```

1.2 2. Datensätze herunterladen

Python-Skript zum Herunterladen der Datensätze, Erstellen einer Ordnerstruktur und Extrahieren der CSV-Datei.

Wann ist es sinnvoll ein Data-Scrapping-Tool in Python zu programmieren?

- Bei Änderungen der Datensätze hilft ein automatisiertes Skript die Daten unkompliziert und in der selben Ordnerstruktur herunterzuladen
- Datensätze werden auf mehreren Rechnern benötigt (Multiple-User)

[2]:
```
import os
import tarfile
from six.moves import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"
```

[3]:
```
"""
def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
"""
```

[3]:
```
'\ndef fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):\n
if not os.path.isdir(housing_path):\n        os.makedirs(housing_path)\n
tgz_path = os.path.join(housing_path, "housing.tgz")\n
urllib.request.urlretrieve(housing_url, tgz_path)\n    housing_tgz =
tarfile.open(tgz_path)\n    housing_tgz.extractall(path=housing_path)\n
housing_tgz.close()\n    if not os.path.isdir(housing_path):\n
os.makedirs(housing_path)\n'
```

Datensatz bereits heruntergeladen und Ordner erstellt

[4]:
```
#fetch_housing_data()
```

1.3 3. Auslesen der Daten

[5]:
```
import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

[6]:
```
housing = load_housing_data()
housing.head()
```

[6]:
```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
```

1.4 4. Einblick in die Datenstruktur

[7]:
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
```

```
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

1.4.1 4.1 Anzeigen der Kategorie `ocean_proximity`

[8]:
```python
housing["ocean_proximity"].value_counts()
```

[8]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

1.4.2 4.2 Zusammenfassung der numerischen Attribute

[9]:
```python
housing.describe()
```

[9]:
```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
mean    -119.569704     35.631861           28.639486   2635.763081
std        2.003532      2.135952           12.585558   2181.615252
min     -124.350000     32.540000            1.000000      2.000000
25%     -121.800000     33.930000           18.000000   1447.750000
50%     -118.490000     34.260000           29.000000   2127.000000
75%     -118.010000     37.710000           37.000000   3148.000000
max     -114.310000     41.950000           52.000000  39320.000000

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
mean       537.870553   1425.476744    499.539680       3.870671
std        421.385070   1132.462122    382.329753       1.899822
min          1.000000      3.000000      1.000000       0.499900
25%        296.000000    787.000000    280.000000       2.563400
50%        435.000000   1166.000000    409.000000       3.534800
75%        647.000000   1725.000000    605.000000       4.743250
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        20640.000000
mean        206855.816909
std         115395.615874
min          14999.000000
```

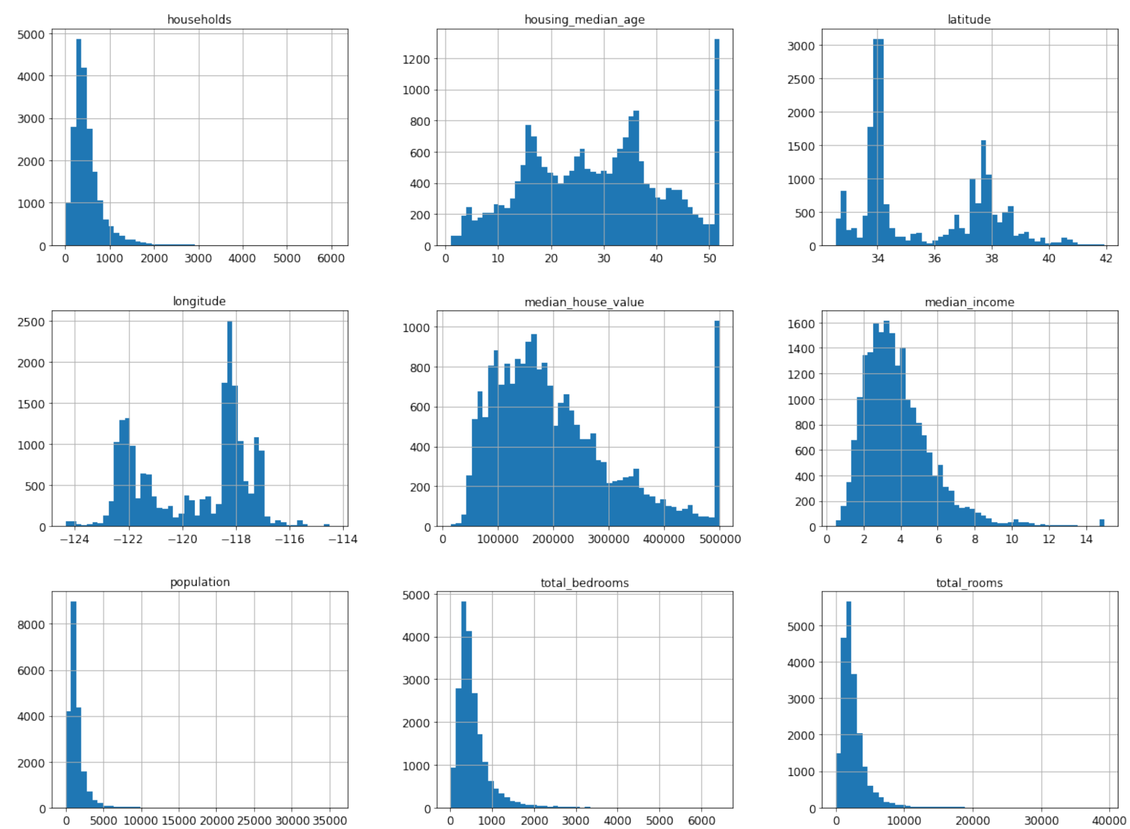
```
25%          119600.000000
50%          179700.000000
75%          264725.000000
max          500001.000000
```

1. Beispiel 25% der Distrikte in Kalifornien haben Häuser, die im Durchschnitt 18 Jahre oder jünger sind.

2. Beispiel 75% der Distrikte in Kalifornien haben 1725 Einwohner oder mehr.

1.4.3 4.3 Visualierung der numerischen Attribute über ein Histogram

[10]:
```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



- Vertikale = Anzahl der Instanzen
- Horizontale = Wertebereich

1. Beispiel Rund 450 Distrikte in Kalifornien haben Häuser, die im Durchschnitt 18 Jahre alt sind.

2. Beispiel Rund 210 Distrikte in Kalifornien haben Häuser, die im Durchschnitt 300.000 USD wert sind.

1.4.4 Vorverarbeitete Datensätze

- `housing_median_age`
- `housing_median_value`
- `median_income`

Algorithmus könnte fälschlicherweise aus den Rohdaten lernen, dass die Preise nie höher als die Limits sind.

1.4.5 Relevanz

Stellen die vorverarbeiteten Werte in `housing_median_value` eine hohe Relevanz für die Entscheidung dar?

Wenn ja, können zwei Dinge unternommen werden: 1. Die passenden Labels zu den gekappten Werten der Distrikte sammeln und aufbereiten. 2. Die Distrikte aus dem Data-Mining-Prozess entfernen, die davon betroffen sind.

1.5 5. Aufteilung in einen Trainingsdatensatz und einen Validierungsdatensatz

Data Snooping Bias Menschen neigen dazu Datensätze automatisch auszuwerten und interessante Muster in diesen zu erkennen. Dies birgt die Gefahr, dass im Vorfeld ein Machine-Learning-Model präferiert wird.

Um dem entgegen zu wirken wird die Voraussagekraft des Models getestet.

Dazu wird der Datensatz in einen $Trainingsdatensatz$ (80%) und einen $Validierungsdatensatz$ (20%) aufgeteilt.

1.5.1 5.1 Aufteilung mit `NumPy`

[11]:
```python
# to make this notebook's output identical at every run
np.random.seed(42)
```

[12]:
```python
# For illustration only. Sklearn has train_test_split()
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
```

```
    return data.iloc[train_indices], data.iloc[test_indices]
```

[13]:
```
train_set, test_set = split_train_test(housing, 0.2)
print(len(train_set), "train +", len(test_set), "test")
```

```
16512 train + 4128 test
```

[14]:
```
import hashlib

def test_set_check(identifier, test_ratio, hash=hashlib.md5):
    return bytearray(hash(np.int64(identifier)).digest())[-1] < 256 * test_ratio

def split_train_test_by_id(data, test_ratio, id_column):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio))
    return data.loc[~in_test_set], data.loc[in_test_set]
```

[15]:
```
housing_with_id = housing.reset_index()   # adds an `index` column
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")
```

[16]:
```
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "id")
```

[17]:
```
test_set.head()
```

[17]:
```
    index  longitude  latitude  housing_median_age  total_rooms  \
8       8    -122.26     37.84                42.0       2555.0
10     10    -122.26     37.85                52.0       2202.0
11     11    -122.26     37.85                52.0       3503.0
12     12    -122.26     37.85                52.0       2491.0
13     13    -122.26     37.84                52.0        696.0

    total_bedrooms  population  households  median_income  median_house_value  \
8            665.0      1206.0       595.0         2.0804            226700.0
10           434.0       910.0       402.0         3.2031            281500.0
11           752.0      1504.0       734.0         3.2705            241800.0
12           474.0      1098.0       468.0         3.0750            213500.0
13           191.0       345.0       174.0         2.6736            191300.0

   ocean_proximity         id
8         NEAR BAY -122222.16
10        NEAR BAY -122222.15
11        NEAR BAY -122222.15
12        NEAR BAY -122222.15
13        NEAR BAY -122222.16
```

1.5.2 5.2 Aufteilung mit Scikit-Learn

[18]:
```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

[19]:
```
test_set.head()
```

[19]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
20046    -119.01     36.06                25.0       1505.0             NaN
3024     -119.46     35.14                30.0       2943.0             NaN
15663    -122.44     37.80                52.0       3830.0             NaN
20484    -118.72     34.28                17.0       3051.0             NaN
9814     -121.93     36.62                34.0       2351.0             NaN

       population  households  median_income  median_house_value  \
20046      1392.0       359.0         1.6812             47700.0
3024       1565.0       584.0         2.5313             45800.0
15663      1310.0       963.0         3.4801            500001.0
20484      1705.0       495.0         5.7376            218600.0
9814       1063.0       428.0         3.7250            278000.0

      ocean_proximity
20046          INLAND
3024           INLAND
15663        NEAR BAY
20484       <1H OCEAN
9814       NEAR OCEAN
```

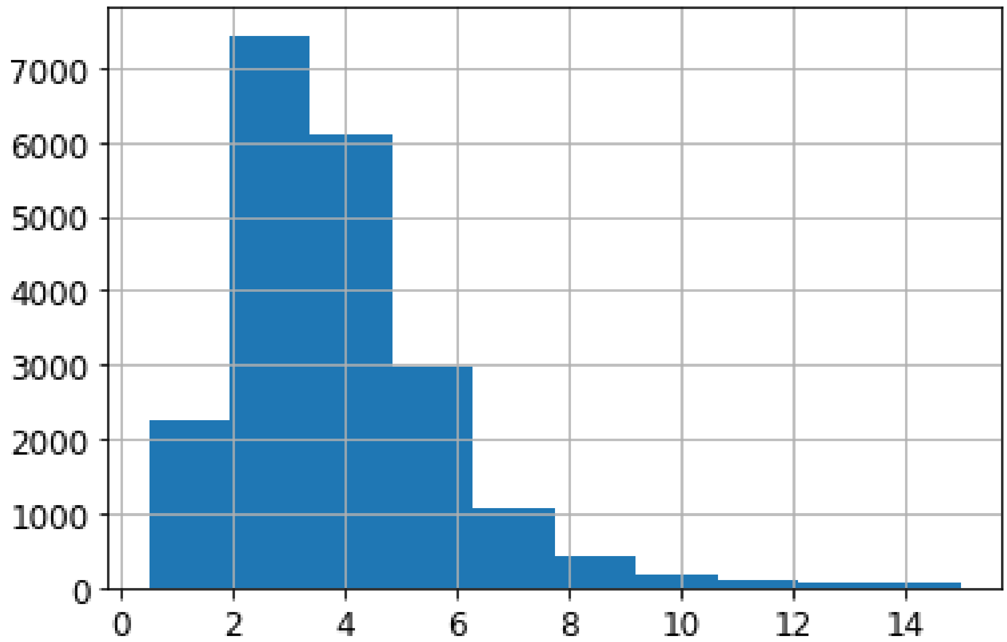
1.5.3 5.3 Stratified Sampling (Geschichtete Stichprobe)

Representative Darstellung / Wiedergabe der homogenen Untergruppen (= *Strata* oder *Startum*) und der richtigen Anzahl an Instanzen von jedem Startum, z.B. Anteil Männer/Frauen an der Gesamtpopulation.

5.3.1 median_income

[20]:
```
housing["median_income"].hist()
```

[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x128fe79a0>
```



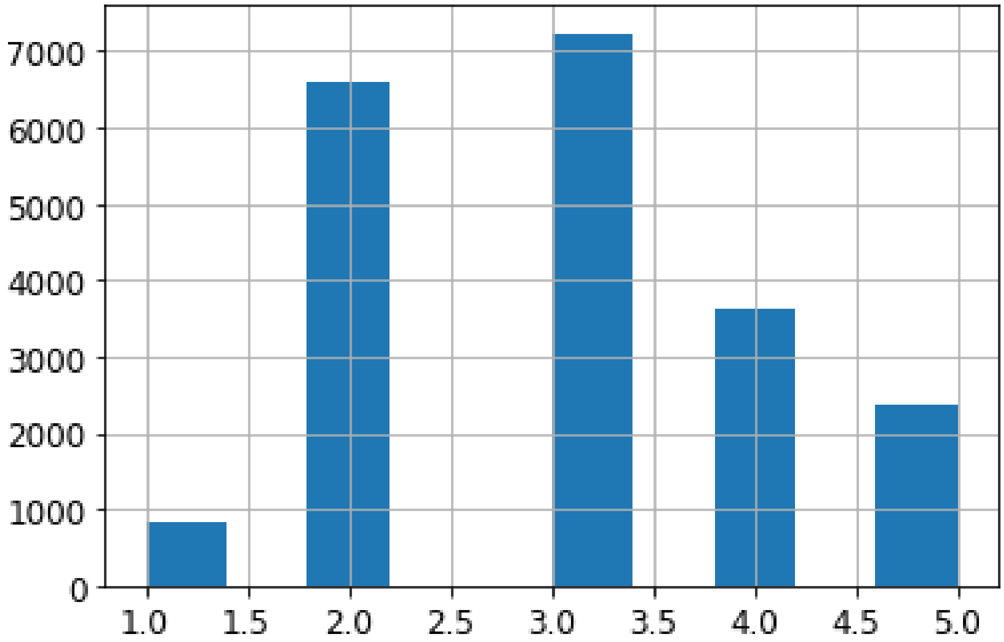
[21]:
```
# Divide by 1.5 to limit the number of income categories
housing["income_cat"] = np.ceil(housing["median_income"] / 1.5)
# Label those above 5 as 5
housing["income_cat"].where(housing["income_cat"] < 5, 5.0, inplace=True)
```

[22]:
```
housing["income_cat"].value_counts()
```

[22]:
```
3.0    7236
2.0    6581
4.0    3639
5.0    2362
1.0     822
Name: income_cat, dtype: int64
```

[23]:
```
housing["income_cat"].hist()
```

[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1290934f0>
```



Kategorie	Wertebereich	Einkommensspanne
1.0	0.0 bis 1.5	< 15.000 USD
2.0	1.5 bis 3.0	15.000 USD bis 30.000 USD
3.0	3.0 bis 4.5	30.000 USD bis 45.000 USD
4.0	4.5 bis 6.0	45.000 USD bis 60.000 USD
5.0	> 6.0	> 60.000 USD

5.3.2 StratifiedShuffleSplit

[24]:
```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)

for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

[25]:
```
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

[25]:
```
3.0    0.350533
2.0    0.318798
4.0    0.176357
5.0    0.114583
1.0    0.039729
```

```
Name: income_cat, dtype: float64
```

5.3.3 Vergleich zwischen zufällig generierten Stichproben und geschichteten Stichproben

[26]:
```
def income_cat_proportions(data):
    return data["income_cat"].value_counts() / len(data)

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)

compare_props = pd.DataFrame({
    "Overall": income_cat_proportions(housing),
    "Stratified": income_cat_proportions(strat_test_set),
    "Random": income_cat_proportions(test_set),
}).sort_index()

compare_props["Rand. %error"] = 100 * compare_props["Random"] /␣
 ↪compare_props["Overall"] - 100
compare_props["Strat. %error"] = 100 * compare_props["Stratified"] /␣
 ↪compare_props["Overall"] - 100
```

[27]:
```
compare_props
```

[27]:

	Overall	Stratified	Random	Rand. %error	Strat. %error
1.0	0.039826	0.039729	0.040213	0.973236	-0.243309
2.0	0.318847	0.318798	0.324370	1.732260	-0.015195
3.0	0.350581	0.350533	0.358527	2.266446	-0.013820
4.0	0.176308	0.176357	0.167393	-5.056334	0.027480
5.0	0.114438	0.114583	0.109496	-4.318374	0.127011

Zurücksetzen von `income_cat`

[28]:
```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

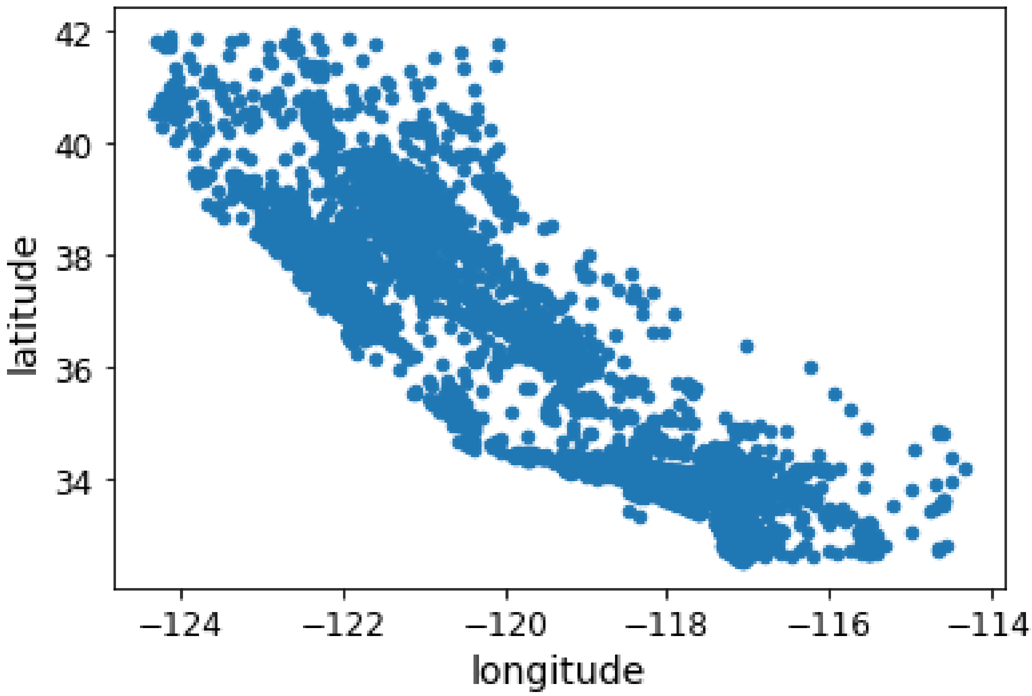
1.6 6. Erkunden und visualisieren des Datensatz

[29]:
```
housing = strat_train_set.copy()
```

1.6.1 6.1 Visualisierung der geografischen Daten

[30]:
```
housing.plot(kind="scatter", x="longitude", y="latitude")
```

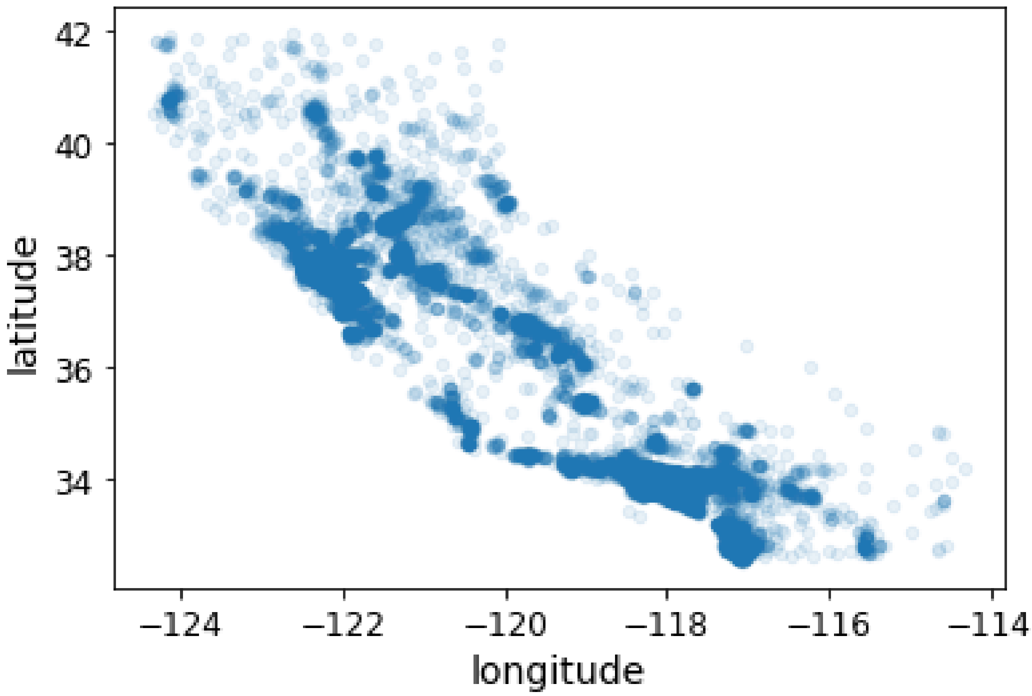
[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x119dda640>
```



1.6.2 6.1.1 Visualisierung unter Berücksichtigungen der Dichte der Datenpunkte

[31]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

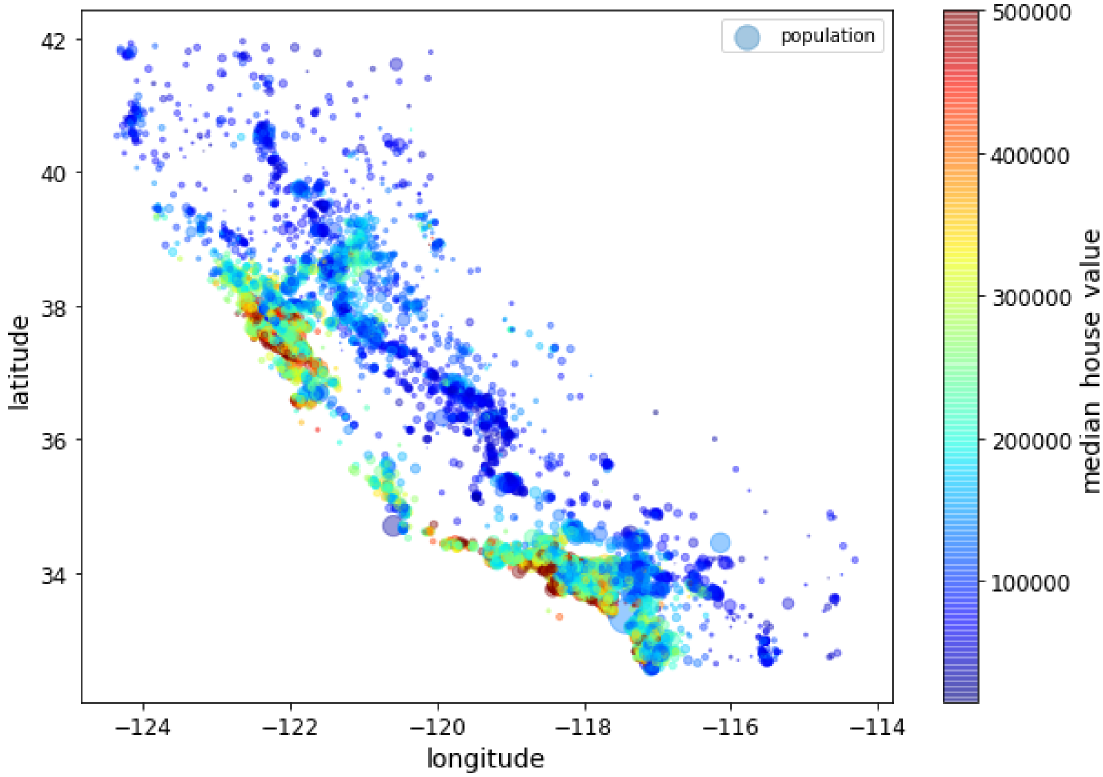
[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x10a2b4eb0>
```



1.6.3 6.1.2 Visualisierung unter Berücksichtigung von `population` und `median_house_value`

[32]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
    s=housing["population"]/100, label="population", figsize=(10,7),
    c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
    sharex=False)
plt.legend()
```

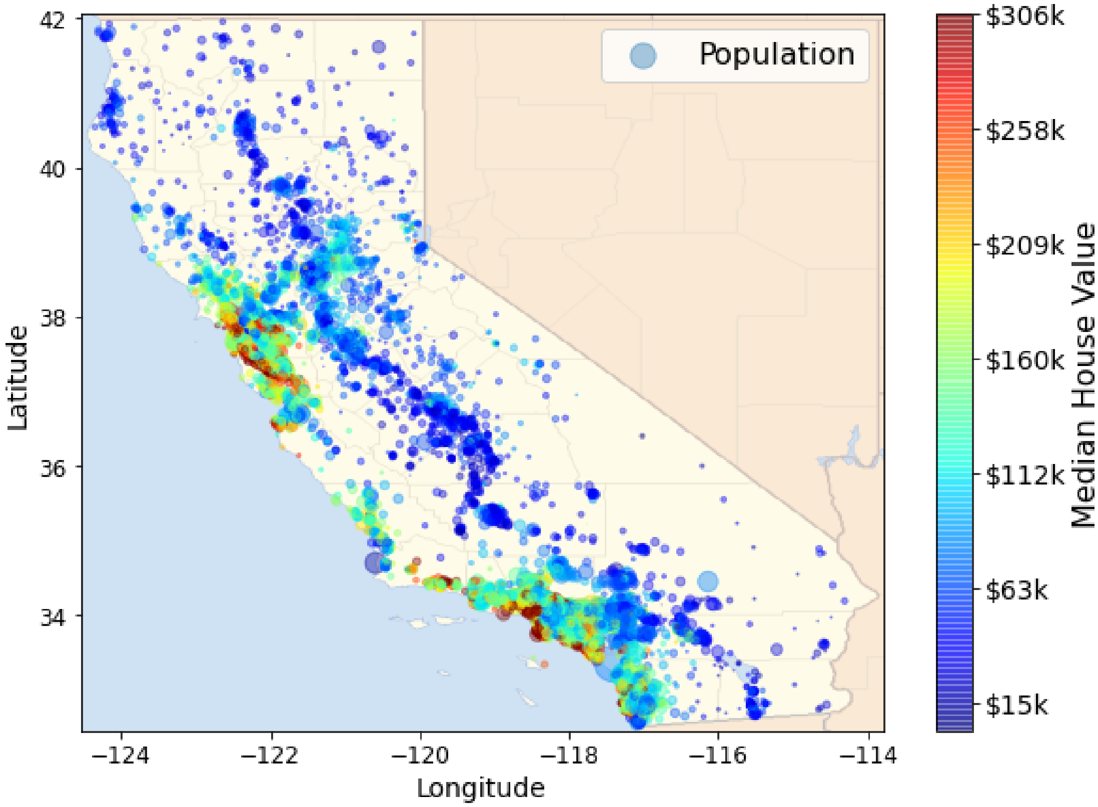
[32]:
```
<matplotlib.legend.Legend at 0x119dd3f70>
```



[33]:
```
import matplotlib.image as mpimg
california_img=mpimg.imread(PROJECT_ROOT_DIR + '/images/california.png')
ax = housing.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                       s=housing['population']/100, label="Population",
                       c="median_house_value", cmap=plt.get_cmap("jet"),
                       colorbar=False, alpha=0.4,
                      )
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05], alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

prices = housing["median_house_value"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cbar = plt.colorbar()
cbar.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values],␣
 ↪fontsize=14)
cbar.set_label('Median House Value', fontsize=16)

plt.legend(fontsize=16)
plt.show()
```



1.6.4 6.1.3 Korrelationen zu `median_house_value` mithilfe einer Korrelationsmatrix

Pearson-Korrelationskoeffizient:

[34]:
```
corr_matrix = housing.corr()
```

[35]:
```
corr_matrix["median_house_value"].sort_values(ascending=False)
```

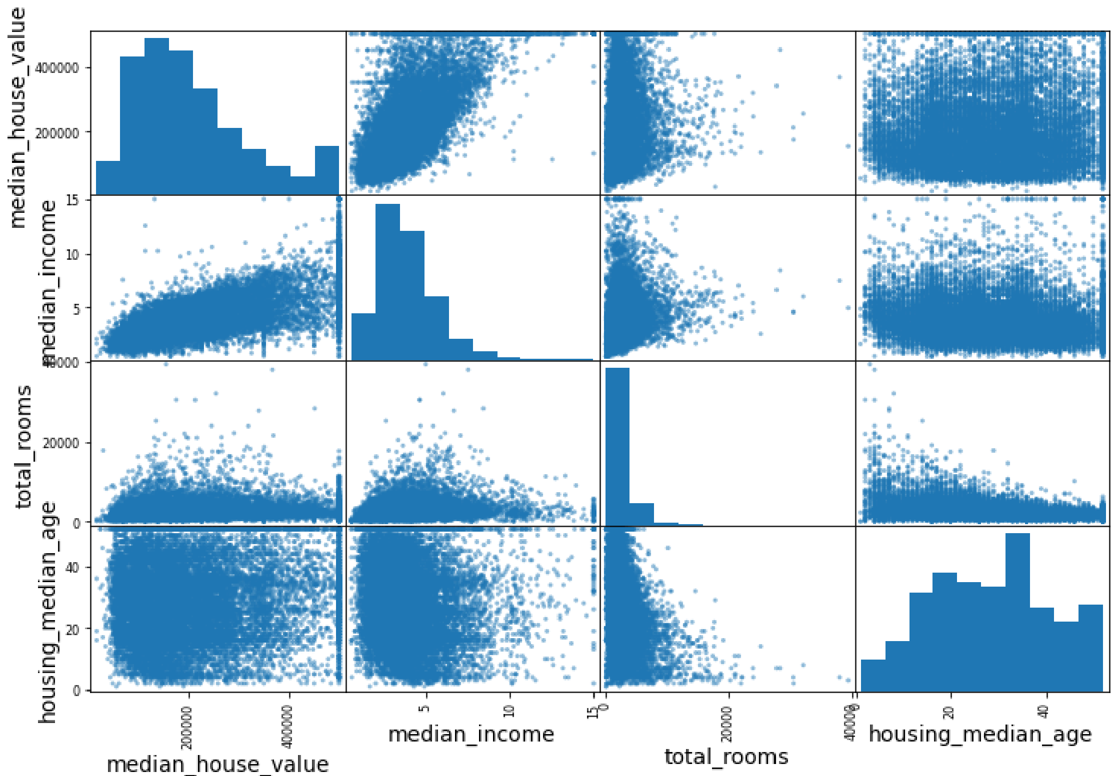
[35]:
```
median_house_value    1.000000
median_income         0.687160
total_rooms           0.135097
housing_median_age    0.114110
households            0.064506
total_bedrooms        0.047689
population           -0.026920
longitude            -0.047432
latitude             -0.142724
Name: median_house_value, dtype: float64
```

1.6.5 6.1.4 Korrelationen zu `median_house_value` mithilfe einer Scatter-Matrix

[36]:
```python
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
```

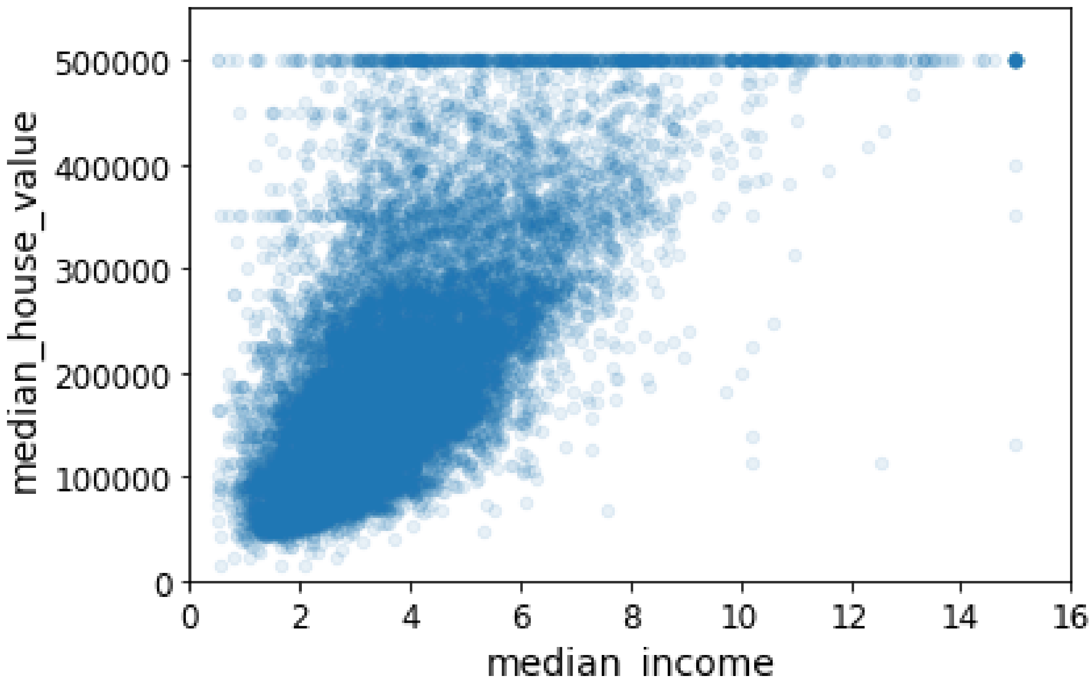
[36]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x119dbda00>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x119898b20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x119c45f40>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1198b48e0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1198e0d60>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x119796520>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x119796880>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1197c0370>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x119a7e370>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x119d22c10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x10a1f84f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11971b820>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x10a285340>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1290e0d60>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1199251f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1198ee640>]],
      dtype=object)
```



Scatter-Matrix zwischen `median_income` und `median_house_value`

[37]:
```
housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.
↪1)
plt.axis([0, 16, 0, 550000])
```

[37]: (0.0, 16.0, 0.0, 550000.0)



- starke (positive) Korrelation
- Preisgrenze bei 500.000 USD
- weitere horizontale Linien bei 450.000 USD, 350.000 USD und 280.000 USD

1.6.6 6.2 Experimentieren mit verschiedenen Attributkombinationen

- `total_rooms` ist schwach aussagefähig ohne `total_households` zu kennen
- interessant wäre die Anzahl der Zimmer pro Haushalt
- ebenfalls interessant ist die Anzahl der Bevölkerung pro Haushalt

[38]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

[39]:
```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

[39]:
```
median_house_value          1.000000
median_income               0.687160
rooms_per_household         0.146285
total_rooms                 0.135097
housing_median_age          0.114110
households                  0.064506
total_bedrooms              0.047689
```

```
population_per_household   -0.021985
population                 -0.026920
longitude                  -0.047432
latitude                   -0.142724
bedrooms_per_room          -0.259984
Name: median_house_value, dtype: float64
```

1.7 7. Datenvorbereitung für die Machine-Learning-Algorithmen

[40]:
```
housing = strat_train_set.drop("median_house_value", axis=1) # drop labels for
 ↪training set
housing_labels = strat_train_set["median_house_value"].copy()
```

[41]:
```
strat_train_set.head()
```

[41]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
17606    -121.89     37.29                38.0       1568.0           351.0
18632    -121.93     37.05                14.0        679.0           108.0
14650    -117.20     32.77                31.0       1952.0           471.0
3230     -119.61     36.31                25.0       1847.0           371.0
3555     -118.59     34.23                17.0       6592.0          1525.0

       population  households  median_income  median_house_value  \
17606       710.0       339.0         2.7042            286600.0
18632       306.0       113.0         6.4214            340600.0
14650       936.0       462.0         2.8621            196900.0
3230       1460.0       353.0         1.8839             46300.0
3555       4459.0      1463.0         3.0347            254500.0

      ocean_proximity
17606       <1H OCEAN
18632       <1H OCEAN
14650      NEAR OCEAN
3230           INLAND
3555        <1H OCEAN
```

[42]:
```
housing.head()
```

[42]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
17606    -121.89     37.29                38.0       1568.0           351.0
18632    -121.93     37.05                14.0        679.0           108.0
14650    -117.20     32.77                31.0       1952.0           471.0
3230     -119.61     36.31                25.0       1847.0           371.0
3555     -118.59     34.23                17.0       6592.0          1525.0

       population  households  median_income ocean_proximity
```

```
17606        710.0       339.0           2.7042       <1H OCEAN
18632        306.0       113.0           6.4214       <1H OCEAN
14650        936.0       462.0           2.8621      NEAR OCEAN
3230        1460.0       353.0           1.8839          INLAND
3555        4459.0      1463.0           3.0347       <1H OCEAN
```

[43]:
```python
housing_labels.head()
```

[43]:
```
17606    286600.0
18632    340600.0
14650    196900.0
3230      46300.0
3555     254500.0
Name: median_house_value, dtype: float64
```

1.7.1 7.1 Data Cleaning

[44]:
```python
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

[44]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
4629     -118.30     34.07                18.0       3759.0             NaN
6068     -117.86     34.01                16.0       4632.0             NaN
17923    -121.97     37.35                30.0       1955.0             NaN
13656    -117.30     34.05                 6.0       2155.0             NaN
19252    -122.79     38.48                 7.0       6837.0             NaN

       population  households  median_income ocean_proximity
4629       3296.0      1462.0         2.2708       <1H OCEAN
6068       3038.0       727.0         5.1762       <1H OCEAN
17923       999.0       386.0         4.6328       <1H OCEAN
13656      1039.0       391.0         1.6675          INLAND
19252      3468.0      1405.0         3.1662       <1H OCEAN
```

Option 1

[45]:
```python
sample_incomplete_rows.dropna(subset=["total_bedrooms"])
```

[45]:
```
Empty DataFrame
Columns: [longitude, latitude, housing_median_age, total_rooms, total_bedrooms,
population, households, median_income, ocean_proximity]
Index: []
```

Option 2

[46]:
```
sample_incomplete_rows.drop("total_bedrooms", axis=1)
```

[46]:
```
       longitude  latitude  housing_median_age  total_rooms  population  \
4629     -118.30     34.07                18.0       3759.0      3296.0
6068     -117.86     34.01                16.0       4632.0      3038.0
17923    -121.97     37.35                30.0       1955.0       999.0
13656    -117.30     34.05                 6.0       2155.0      1039.0
19252    -122.79     38.48                 7.0       6837.0      3468.0

       households  median_income ocean_proximity
4629       1462.0         2.2708       <1H OCEAN
6068        727.0         5.1762       <1H OCEAN
17923       386.0         4.6328       <1H OCEAN
13656       391.0         1.6675          INLAND
19252      1405.0         3.1662       <1H OCEAN
```

Option 3

[47]:
```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # option 3
sample_incomplete_rows
```

[47]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
4629     -118.30     34.07                18.0       3759.0           433.0
6068     -117.86     34.01                16.0       4632.0           433.0
17923    -121.97     37.35                30.0       1955.0           433.0
13656    -117.30     34.05                 6.0       2155.0           433.0
19252    -122.79     38.48                 7.0       6837.0           433.0

       population  households  median_income ocean_proximity
4629       3296.0      1462.0         2.2708       <1H OCEAN
6068       3038.0       727.0         5.1762       <1H OCEAN
17923       999.0       386.0         4.6328       <1H OCEAN
13656      1039.0       391.0         1.6675          INLAND
19252      3468.0      1405.0         3.1662       <1H OCEAN
```

[48]:
```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
```

[49]:
```
housing_num = housing.drop('ocean_proximity', axis=1)
```

[50]:
```
imputer.fit(housing_num)
```

[50]:
```
SimpleImputer(add_indicator=False, copy=True, fill_value=None,
              missing_values=nan, strategy='median', verbose=0)
```

[51]:
```
imputer.statistics_
```

[51]:
```
array([-118.51   ,   34.26   ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

[52]:
```
housing_num.median().values
```

[52]:
```
array([-118.51   ,   34.26   ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

Leere Felder durch errechnete Medianwerte ersetzen:

[53]:
```
X = imputer.transform(housing_num)
```

Umwandlung in Pandas DataFrame:

[54]:
```
housing_tr = pd.DataFrame(X, columns=housing_num.columns, index=housing_num.
 ↪index)
```

1.7.2 7.2 Umgang mit Text- und Kategorieattributen

[55]:
```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(10)
```

[55]:

	ocean_proximity
17606	<1H OCEAN
18632	<1H OCEAN
14650	NEAR OCEAN
3230	INLAND
3555	<1H OCEAN
19480	INLAND
8879	<1H OCEAN
13685	INLAND
4937	<1H OCEAN
4861	<1H OCEAN

[56]:
```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded[:10]
```

[56]:
```
array([[0.],
       [0.],
       [4.],
```

```
        [1.],
        [0.],
        [1.],
        [0.],
        [1.],
        [0.],
        [0.]])
```

[57]:
```
ordinal_encoder.categories_
```

[57]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

One-Hot Encoding Beispiel: - Bewertung = [“schlecht”, “durchschnittlich”, “gut”, “exzellent”] - `ocean_proximity` = [‘<1H OCEAN’, ‘INLAND’, ‘ISLAND’, ‘NEAR BAY’, ‘NEAR OCEAN’]

[58]:
```
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder(sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

[58]:
```
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       …,
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

[59]:
```
cat_encoder.categories_
```

[59]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

1.7.3 7.3 Transformer

[60]:
```
from sklearn.base import BaseEstimator, TransformerMixin

# column index
rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
```

```python
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
        population_per_household = X[:, population_ix] / X[:, household_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

[61]:
```python
housing_extra_attribs = pd.DataFrame(
    housing_extra_attribs,
    columns=list(housing.columns)+["rooms_per_household",␣
 ↪"population_per_household"])
housing_extra_attribs.head()
```

[61]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-121.89	37.29	38	1568	351	710
1	-121.93	37.05	14	679	108	306
2	-117.2	32.77	31	1952	471	936
3	-119.61	36.31	25	1847	371	1460
4	-118.59	34.23	17	6592	1525	4459

	households	median_income	ocean_proximity	rooms_per_household
0	339	2.7042	<1H OCEAN	4.62537
1	113	6.4214	<1H OCEAN	6.00885
2	462	2.8621	NEAR OCEAN	4.22511
3	353	1.8839	INLAND	5.23229
4	1463	3.0347	<1H OCEAN	4.50581

	population_per_household
0	2.0944
1	2.70796
2	2.02597
3	4.13598
4	3.04785

1.7.4 7.4 Merkmalsskalierung und Transformation Pipeline

[62]:
```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

[63]:
```python
housing_num_tr
```

[63]:
```
array([[-1.15604281,  0.77194962,  0.74333089, …, -0.31205452,
        -0.08649871,  0.15531753],
       [-1.17602483,  0.6596948 , -1.1653172 , …,  0.21768338,
        -0.03353391, -0.83628902],
       [ 1.18684903, -1.34218285,  0.18664186, …, -0.46531516,
        -0.09240499,  0.4222004 ],
       …,
       [ 1.58648943, -0.72478134, -1.56295222, …,  0.3469342 ,
        -0.03055414, -0.52177644],
       [ 0.78221312, -0.85106801,  0.18664186, …,  0.02499488,
         0.06150916, -0.30340741],
       [-1.43579109,  0.99645926,  1.85670895, …, -0.22852947,
        -0.09586294,  0.10180567]])
```

[64]:
```python
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

[65]:
```python
housing_prepared
```

[65]:
```
array([[-1.15604281,  0.77194962,  0.74333089, …,  0.        ,
         0.        ,  0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , …,  0.        ,
         0.        ,  0.        ],
```

```
       [ 1.18684903, -1.34218285,  0.18664186, …,  0.        ,
         0.        ,  1.        ],
       …,
       [ 1.58648943, -0.72478134, -1.56295222, …,  0.        ,
         0.        ,  0.        ],
       [ 0.78221312, -0.85106801,  0.18664186, …,  0.        ,
         0.        ,  0.        ],
       [-1.43579109,  0.99645926,  1.85670895, …,  0.        ,
         1.        ,  0.        ]])
```

[66]:
```
housing_prepared.shape
```

[66]:
```
(16512, 16)
```

1.8 8. Model auswählen und trainieren

1.8.1 Zusammenfassung:

1. Extrahierung des Datensatz
2. Erkundung der Attribute und Parametertypen
3. Aufteilung in ein Trainingsdatensatz und ein Validierungsdatensatz
4. Aufbereitung für die Machine-Learning-Algorithmen

1.8.2 8.1 Machine-Learning-Algorithmus #1: Linear-Regression

[67]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

[67]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[68]:
```
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)

print("Predictions:", lin_reg.predict(some_data_prepared))
```

```
Predictions: [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
```

[69]:
```
print("Labels:", list(some_labels))
```

```
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

[70]:
```
some_data_prepared
```

[70]:
```
array([[-1.15604281,  0.77194962,  0.74333089, -0.49323393, -0.44543821,
        -0.63621141, -0.42069842, -0.61493744, -0.31205452, -0.08649871,
         0.15531753,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , -0.90896655, -1.0369278 ,
        -0.99833135, -1.02222705,  1.33645936,  0.21768338, -0.03353391,
        -0.83628902,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, -0.31365989, -0.15334458,
        -0.43363936, -0.0933178 , -0.5320456 , -0.46531516, -0.09240499,
         0.4222004 ,  0.        ,  0.        ,  0.        ,  0.        ,
         1.        ],
       [-0.01706767,  0.31357576, -0.29052016, -0.36276217, -0.39675594,
         0.03604096, -0.38343559, -1.04556555, -0.07966124,  0.08973561,
        -0.19645314,  0.        ,  1.        ,  0.        ,  0.        ,
         0.        ],
       [ 0.49247384, -0.65929936, -0.92673619,  1.85619316,  2.41221109,
         2.72415407,  2.57097492, -0.44143679, -0.35783383, -0.00419445,
         0.2699277 ,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ]])
```

Evaluation anhand des Root-mean-square error (RMSE)

[71]:
```
from sklearn.metrics import mean_squared_error

housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

[71]: 68628.19819848922

[72]:
```
from sklearn.metrics import mean_absolute_error

lin_mae = mean_absolute_error(housing_labels, housing_predictions)
lin_mae
```

[72]: 49439.89599001897

Hinweis auf Underfitting (Das Model ist zu einfach bzw. der Datensatz ist für das Model zu komplex, um die tieferen Datenstrukturen zu verarbeiten.)

1.8.3 8.2 Machine-Learning-Algorithmus #2: Decision-Tree-Regressor

[73]:
```python
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

[73]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=42, splitter='best')
```

Evaluation anhand des Root-mean-square error (RMSE)

[74]:
```python
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

[74]:
```
0.0
```

Hinweis auf Overfitting (Das Model ist zu komplex bzw. der Datensatz ist für das Model zu einfach, um die tieferen Datenstrukturen zu verarbeiten.)

Evaluation anhand der Cross-Validation

[75]:
```python
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
```

Auswertung des Decision-Tree-Model

[76]:
```python
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

display_scores(tree_rmse_scores)
```

```
Scores: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
 71115.88230639 75585.14172901 70262.86139133 70273.6325285
 75366.87952553 71231.65726027]
```

```
Mean: 71407.68766037929
Standard deviation: 2439.4345041191004
```

Auswerung der Linear-Regression-Model

[77]:
```python
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
Scores: [66755.35819855 66966.14573098 70347.95244419 74769.18698807
 68031.13388938 71229.17716103 64959.86064183 68270.70198961
 71552.91566558 67665.10082067]
Mean: 69054.75335298848
Standard deviation: 2744.2187083829585
```

Model	RMSE	Mittelwert der Abweichungen	Standardabweichung
Linear Regression	68,628.20	69,054.75	2,744.22
Decision Tree	0.0	71,407.69	2,439.43

1.8.4 8.3 Machine-Learning-Algorithmus #3: Random-Forest-Regressor

[78]:
```python
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=10, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

[78]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=10, n_jobs=None, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

Auswerung der Random-Forest-Model

[79]:
```python
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

[79]:
```
21933.31414779769
```

[80]:
```
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [51646.44545909 48940.60114882 53050.86323649 54408.98730149
 50922.14870785 56482.50703987 51864.52025526 49760.85037653
 55434.21627933 53326.10093303]
Mean: 52583.72407377466
Standard deviation: 2298.353351147122
```

[81]:
```
scores = cross_val_score(lin_reg, housing_prepared, housing_labels,␣
 ↪scoring="neg_mean_squared_error", cv=10)
pd.Series(np.sqrt(-scores)).describe()
```

[81]:
```
count       10.000000
mean     69054.753353
std       2892.660505
min      64959.860642
25%      67140.884503
50%      68150.917939
75%      71008.870982
max      74769.186988
dtype: float64
```

Model	RMSE	Mittelwert der Abweichungen	Standardabweichung
Linear Regression	68,628.20	69,054.75	2,744.22
Decision Tree	0.0	71,407.69	2,439.43
Random Forest	21,933.31	52,583.72	2,298.35

1.9 9. Optimierung des Model

[82]:
```
from sklearn.model_selection import GridSearchCV

param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # then try 6 (2×3) combinations with bootstrap set as False
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# train across 5 folds, that's a total of (12+6)*5=90 rounds of training
```

```
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',␣
 ↪return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
```

[82]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators=100, n_jobs=None,
                                             oob_score=False, random_state=42,
                                             verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='neg_mean_squared_error', verbose=0)
```

[83]:
```
grid_search.best_params_
```

[83]:
```
{'max_features': 8, 'n_estimators': 30}
```

[84]:
```
grid_search.best_estimator_
```

[84]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features=8, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=30, n_jobs=None, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

Auswertung des Model

[85]:
```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
63669.11631261028 {'max_features': 2, 'n_estimators': 3}
55627.099719926795 {'max_features': 2, 'n_estimators': 10}
53384.57275149205 {'max_features': 2, 'n_estimators': 30}
60965.950449450494 {'max_features': 4, 'n_estimators': 3}
52741.04704299915 {'max_features': 4, 'n_estimators': 10}
50377.40461678399 {'max_features': 4, 'n_estimators': 30}
58663.93866579625 {'max_features': 6, 'n_estimators': 3}
52006.19873526564 {'max_features': 6, 'n_estimators': 10}
50146.51167415009 {'max_features': 6, 'n_estimators': 30}
57869.25276169646 {'max_features': 8, 'n_estimators': 3}
51711.127883959234 {'max_features': 8, 'n_estimators': 10}
49682.273345071546 {'max_features': 8, 'n_estimators': 30}
62895.06951262424 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54658.176157539405 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59470.40652318466 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52724.9822587892 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57490.5691951261 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51009.495668875716 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

Die `GridSearchCV`-Funktion hat den optimalen Hyperparameterwert für `max_features` und `n_estimators` ermittelt, nämlich `(8, 30)` bei einem Mean-Score von USD 49,682.27. Die Standard-Hyperparameterwerte hätten im Vergleich einen Mean-Score von USD 52,583.72 ausgegeben.

[86]:
```
pd.DataFrame(grid_search.cv_results_)
```

[86]:
```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0        0.074249      0.007625         0.004813        0.000928
1        0.234764      0.010902         0.010970        0.001109
2        0.648359      0.009885         0.028627        0.001457
3        0.102777      0.001191         0.003671        0.000156
4        0.353790      0.018348         0.011246        0.002116
5        1.009834      0.018935         0.028522        0.001992
6        0.135911      0.001791         0.003646        0.000423
7        0.457741      0.006491         0.009962        0.001131
8        1.390682      0.006062         0.027354        0.001639
9        0.182551      0.002001         0.003884        0.000402
10       0.602020      0.007112         0.009922        0.000877
11       1.812581      0.016168         0.027530        0.001522
12       0.106300      0.005683         0.004874        0.000561
13       0.336409      0.005414         0.011389        0.000792
14       0.132564      0.006620         0.004203        0.000219
15       0.434827      0.004807         0.012159        0.000893
16       0.161587      0.004781         0.004494        0.000579
17       0.546626      0.009533         0.011178        0.000662

   param_max_features param_n_estimators param_bootstrap  \
0                   2                  3             NaN
1                   2                 10             NaN
```

```
2                   2                 30            NaN
3                   4                  3            NaN
4                   4                 10            NaN
5                   4                 30            NaN
6                   6                  3            NaN
7                   6                 10            NaN
8                   6                 30            NaN
9                   8                  3            NaN
10                  8                 10            NaN
11                  8                 30            NaN
12                  2                  3          False
13                  2                 10          False
14                  3                  3          False
15                  3                 10          False
16                  4                  3          False
17                  4                 10          False

                                               params  split0_test_score  \
0             {'max_features': 2, 'n_estimators': 3}      -3.837622e+09
1            {'max_features': 2, 'n_estimators': 10}      -3.047771e+09
2            {'max_features': 2, 'n_estimators': 30}      -2.689185e+09
3             {'max_features': 4, 'n_estimators': 3}      -3.730181e+09
4            {'max_features': 4, 'n_estimators': 10}      -2.666283e+09
5            {'max_features': 4, 'n_estimators': 30}      -2.387153e+09
6             {'max_features': 6, 'n_estimators': 3}      -3.119657e+09
7            {'max_features': 6, 'n_estimators': 10}      -2.549663e+09
8            {'max_features': 6, 'n_estimators': 30}      -2.370010e+09
9             {'max_features': 8, 'n_estimators': 3}      -3.353504e+09
10           {'max_features': 8, 'n_estimators': 10}      -2.571970e+09
11           {'max_features': 8, 'n_estimators': 30}      -2.357390e+09
12  {'bootstrap': False, 'max_features': 2, 'n_est…      -3.785816e+09
13  {'bootstrap': False, 'max_features': 2, 'n_est…      -2.810721e+09
14  {'bootstrap': False, 'max_features': 3, 'n_est…      -3.618324e+09
15  {'bootstrap': False, 'max_features': 3, 'n_est…      -2.757999e+09
16  {'bootstrap': False, 'max_features': 4, 'n_est…      -3.134040e+09
17  {'bootstrap': False, 'max_features': 4, 'n_est…      -2.525578e+09

    split1_test_score  …  mean_test_score  std_test_score  rank_test_score  \
0        -4.147108e+09  …    -4.053756e+09    1.519591e+08               18
1        -3.254861e+09  …    -3.094374e+09    1.327062e+08               11
2        -3.021086e+09  …    -2.849913e+09    1.626875e+08                9
3        -3.786886e+09  …    -3.716847e+09    1.631510e+08               16
4        -2.784511e+09  …    -2.781618e+09    1.268607e+08                8
5        -2.588448e+09  …    -2.537883e+09    1.214614e+08                3
6        -3.586319e+09  …    -3.441458e+09    1.893056e+08               14
7        -2.782039e+09  …    -2.704645e+09    1.471569e+08                6
8        -2.583638e+09  …    -2.514673e+09    1.285080e+08                2
```

9	-3.348552e+09	…	-3.348850e+09	1.241939e+08	13
10	-2.718994e+09	…	-2.674041e+09	1.392777e+08	5
11	-2.546640e+09	…	-2.468328e+09	1.091662e+08	1
12	-4.166012e+09	…	-3.955790e+09	1.900964e+08	17
13	-3.107789e+09	…	-2.987516e+09	1.539234e+08	10
14	-3.441527e+09	…	-3.536729e+09	7.795057e+07	15
15	-2.851737e+09	…	-2.779924e+09	6.286720e+07	7
16	-3.559375e+09	…	-3.305166e+09	1.879165e+08	12
17	-2.710011e+09	…	-2.601969e+09	1.088048e+08	4

	split0_train_score	split1_train_score	split2_train_score	\
0	-1.064113e+09	-1.105142e+09	-1.116550e+09	
1	-5.927175e+08	-5.870952e+08	-5.776964e+08	
2	-4.381089e+08	-4.391272e+08	-4.371702e+08	
3	-9.865163e+08	-1.012565e+09	-9.169425e+08	
4	-5.097115e+08	-5.162820e+08	-4.962893e+08	
5	-3.838835e+08	-3.880268e+08	-3.790867e+08	
6	-9.245343e+08	-8.886939e+08	-9.353135e+08	
7	-4.980344e+08	-5.045869e+08	-4.994664e+08	
8	-3.838538e+08	-3.804711e+08	-3.805218e+08	
9	-9.228123e+08	-8.553031e+08	-8.603321e+08	
10	-4.932416e+08	-4.815238e+08	-4.730979e+08	
11	-3.841658e+08	-3.744500e+08	-3.773239e+08	
12	-0.000000e+00	-0.000000e+00	-0.000000e+00	
13	-6.056477e-02	-0.000000e+00	-0.000000e+00	
14	-0.000000e+00	-0.000000e+00	-0.000000e+00	
15	-2.089484e+01	-0.000000e+00	-0.000000e+00	
16	-0.000000e+00	-0.000000e+00	-0.000000e+00	
17	-0.000000e+00	-1.514119e-02	-0.000000e+00	

	split3_train_score	split4_train_score	mean_train_score	std_train_score
0	-1.112342e+09	-1.129650e+09	-1.105559e+09	2.220402e+07
1	-5.716332e+08	-5.802501e+08	-5.818785e+08	7.345821e+06
2	-4.376955e+08	-4.452654e+08	-4.394734e+08	2.966320e+06
3	-1.037400e+09	-9.707739e+08	-9.848396e+08	4.084607e+07
4	-5.436192e+08	-5.160297e+08	-5.163863e+08	1.542862e+07
5	-4.040957e+08	-3.845520e+08	-3.879289e+08	8.571233e+06
6	-9.009801e+08	-8.624664e+08	-9.023976e+08	2.591445e+07
7	-4.990325e+08	-5.055542e+08	-5.013349e+08	3.100456e+06
8	-3.856095e+08	-3.901917e+08	-3.841296e+08	3.617057e+06
9	-8.881964e+08	-9.151287e+08	-8.883545e+08	2.750227e+07
10	-5.155367e+08	-4.985555e+08	-4.923911e+08	1.459294e+07
11	-3.882250e+08	-3.810005e+08	-3.810330e+08	4.871017e+06
12	-0.000000e+00	-0.000000e+00	0.000000e+00	0.000000e+00
13	-0.000000e+00	-2.967449e+00	-6.056027e-01	1.181156e+00
14	-0.000000e+00	-6.072840e+01	-1.214568e+01	2.429136e+01
15	-0.000000e+00	-5.465556e+00	-5.272080e+00	8.093117e+00

```
16        -0.000000e+00        -0.000000e+00        0.000000e+00        0.000000e+00
17        -0.000000e+00        -0.000000e+00       -3.028238e-03        6.056477e-03

[18 rows x 23 columns]
```

1.10 10. Analyse des besten Model und dessen Fehler

[87]:
```python
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

[87]:
```
array([7.33442355e-02, 6.29090705e-02, 4.11437985e-02, 1.46726854e-02,
       1.41064835e-02, 1.48742809e-02, 1.42575993e-02, 3.66158981e-01,
       5.64191792e-02, 1.08792957e-01, 5.33510773e-02, 1.03114883e-02,
       1.64780994e-01, 6.02803867e-05, 1.96041560e-03, 2.85647464e-03])
```

[88]:
```python
extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
cat_encoder = full_pipeline.named_transformers_["cat"]
cat_one_hot_attribs = list(cat_encoder.categories_[0])
attributes = num_attribs + extra_attribs + cat_one_hot_attribs
sorted(zip(feature_importances, attributes), reverse=True)
```

[88]:
```
[(0.36615898061813423, 'median_income'),
 (0.16478099356159054, 'INLAND'),
 (0.10879295677551575, 'pop_per_hhold'),
 (0.07334423551601243, 'longitude'),
 (0.06290907048262032, 'latitude'),
 (0.056419179181954014, 'rooms_per_hhold'),
 (0.053351077347675815, 'bedrooms_per_room'),
 (0.04114379847872964, 'housing_median_age'),
 (0.014874280890402769, 'population'),
 (0.014672685420543239, 'total_rooms'),
 (0.014257599323407808, 'households'),
 (0.014106483453584104, 'total_bedrooms'),
 (0.010311488326303788, '<1H OCEAN'),
 (0.0028564746373201584, 'NEAR OCEAN'),
 (0.0019604155994780706, 'NEAR BAY'),
 (6.0280386727366e-05, 'ISLAND')]
```

=> Je näher die Häuser eines Stadtteils am Ozean liegen, desto höher ist der Preis der Häuser.

1.11 11. Evaluation am Validierungsdatensatz

[89]:
```
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

[90]:
```
final_rmse
```

[90]: 47730.22690385927

[ ]:
```

```