Chapter 2 – End-to-end Machine Learning project

Welcome to Machine Learning Housing Corp.! Your task is to predict median house values in Californian districts, given a number of features from these districts.

This notebook contains all the sample code and solutions to the exercices in chapter 2.

Run in Google Colab (https://colab.research.google.com/github/ageron/handson-ml2/blob/master/02_end_to_end_machine_learning_project.ipynb)

Setup

First, let's import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures. We also check that Python 3.5 or later is installed (although Python 2.x may work, it is deprecated so we strongly recommend you use Python 3 instead), as well as Scikit-Learn ≥0.20.

In [1]:
```
# Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('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)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=30
0):
    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")
```

Get the data

In [2]:
```
import os
import tarfile
import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master
/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

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()
```

In [3]:
```
fetch_housing_data()
```

In [4]:
```
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)
```

In [5]:
```
housing = load_housing_data()
housing.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	

In [6]:
```
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
```

In [7]:
```
housing["ocean_proximity"].value_counts()
```

Out[7]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

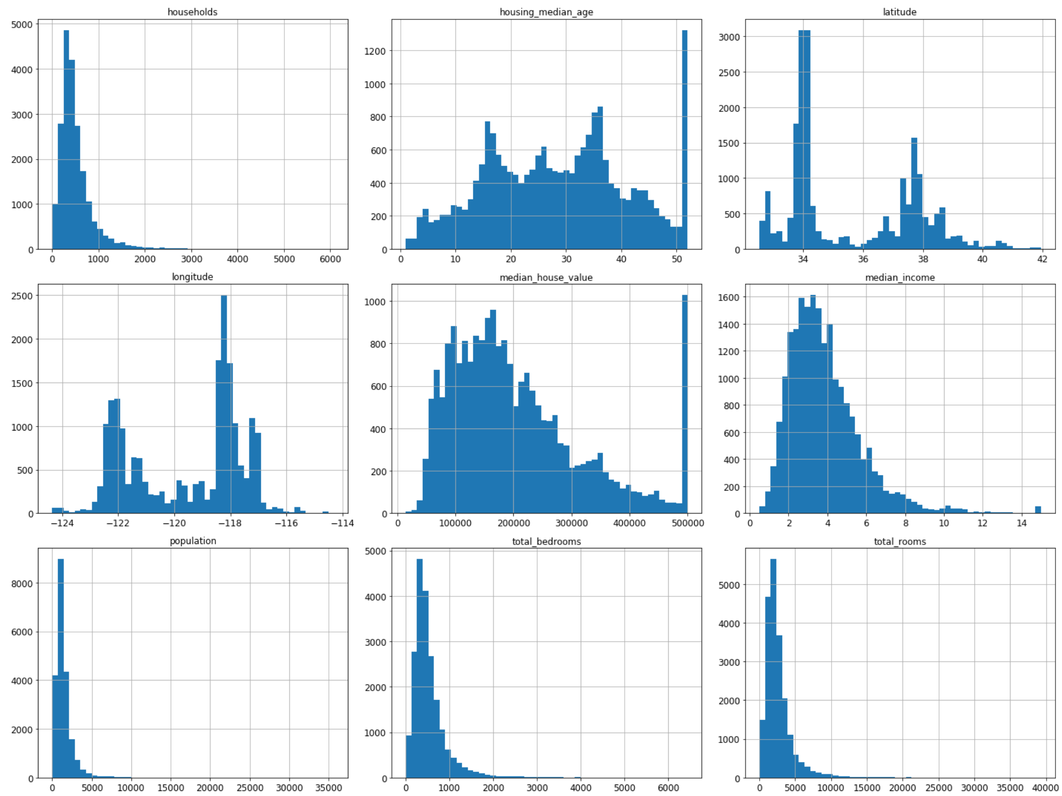
In [8]:
```
housing.describe()
```

Out[8]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	h
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20(
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6(

In [9]:
```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
save_fig("attribute_histogram_plots")
plt.show()
```

```
Saving figure attribute_histogram_plots
```



In [10]:
```
# to make this notebook's output identical at every run
np.random.seed(42)
```

In [11]:
```
from sklearn.model_selection import train_test_split

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

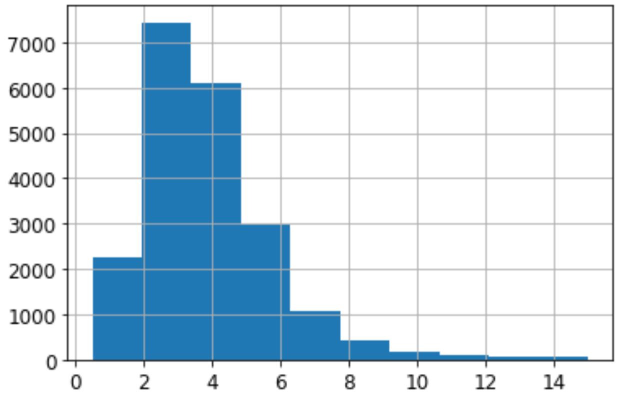
In [12]:
```
test_set.head()
```

Out[12]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	me
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	

In [13]:
```
housing["median_income"].hist()
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1dd40d8bd08>



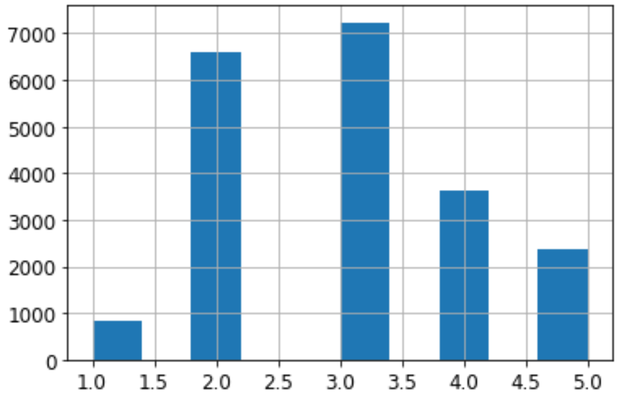
In [14]:
```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
```

In [15]:
```
housing["income_cat"].value_counts()
```

Out[15]:
```
3    7236
2    6581
4    3639
5    2362
1     822
Name: income_cat, dtype: int64
```

In [16]:
```
housing["income_cat"].hist()
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x1dd40f56c48>



In [17]:
```
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]
```

In [18]:
```
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

Out[18]:
```
3    0.350533
2    0.318798
4    0.176357
5    0.114583
1    0.039729
Name: income_cat, dtype: float64
```

In [19]:
```
housing["income_cat"].value_counts() / len(housing)
```

Out[19]:
```
3    0.350581
2    0.318847
4    0.176308
5    0.114438
1    0.039826
Name: income_cat, dtype: float64
```

In [20]:
```
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_prop
s["Overall"] - 100
compare_props["Strat. %error"] = 100 * compare_props["Stratified"] / compare
_props["Overall"] - 100
```

In [21]:
```
compare_props
```

Out[21]:

	Overall	Stratified	Random	Rand. %error	Strat. %error
1	0.039826	0.039729	0.040213	0.973236	-0.243309
2	0.318847	0.318798	0.324370	1.732260	-0.015195
3	0.350581	0.350533	0.358527	2.266446	-0.013820
4	0.176308	0.176357	0.167393	-5.056334	0.027480
5	0.114438	0.114583	0.109496	-4.318374	0.127011

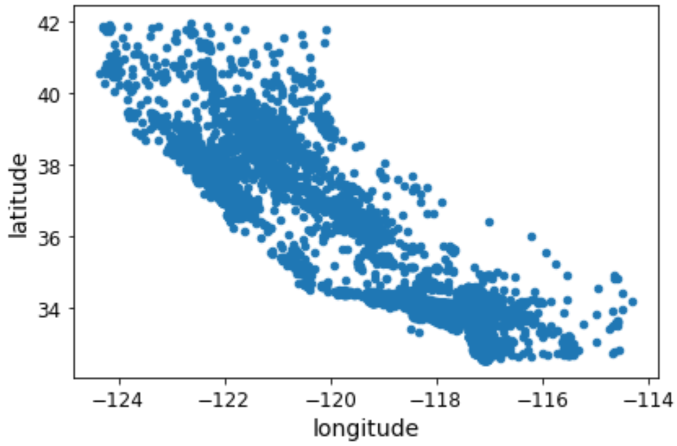
In [22]:
```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

Discover and visualize the data to gain insights

In [23]:
```
housing = strat_train_set.copy()
```

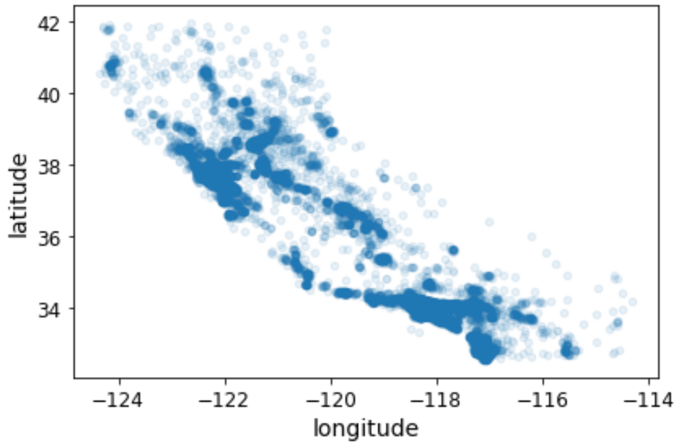
In [24]:
```
housing.plot(kind="scatter", x="longitude", y="latitude")
save_fig("bad_visualization_plot")
```

```
Saving figure bad_visualization_plot
```



In [25]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
save_fig("better_visualization_plot")
```

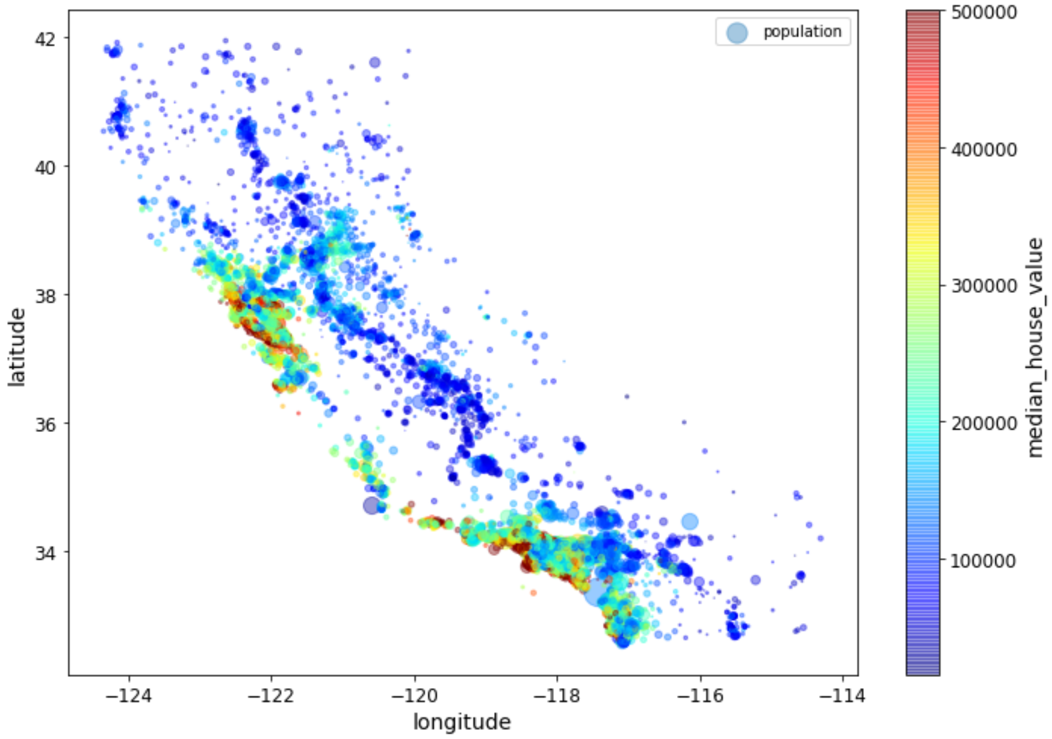
```
Saving figure better_visualization_plot
```



The argument `sharex=False` fixes a display bug (the x-axis values and legend were not displayed). This is a temporary fix (see: https://github.com/pandas-dev/pandas/issues/10611 (https://github.com/pandas-dev/pandas/issues/10611)). Thanks to Wilmer Arellano for pointing it out.

In [26]:
```
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()
save_fig("housing_prices_scatterplot")
```

```
Saving figure housing_prices_scatterplot
```



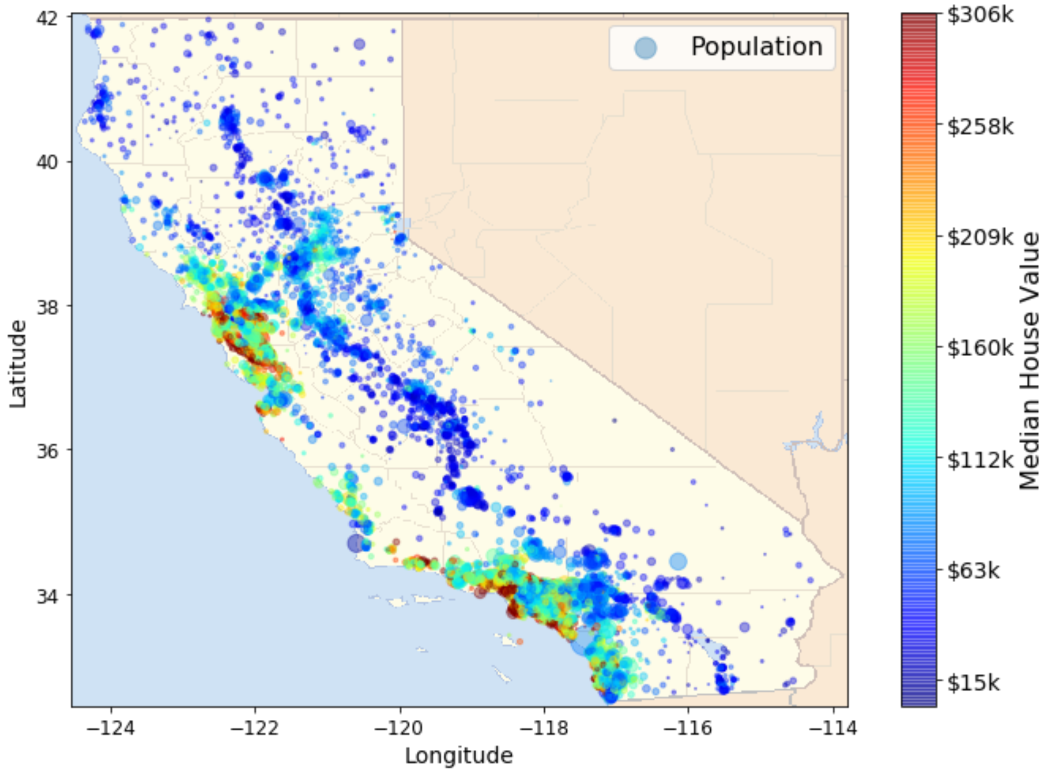
In [27]:
```
# Download the California image
images_path = os.path.join(PROJECT_ROOT_DIR, "images", "end_to_end_project")
os.makedirs(images_path, exist_ok=True)
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master
/"
filename = "california.png"
print("Downloading", filename)
url = DOWNLOAD_ROOT + "images/end_to_end_project/" + filename
urllib.request.urlretrieve(url, os.path.join(images_path, filename))
```

```
Downloading california.png
```

Out[27]:
```
('.\\images\\end_to_end_project\\california.png',
 <http.client.HTTPMessage at 0x1dd41e1d408>)
```

In [28]:
```
import matplotlib.image as mpimg
california_img=mpimg.imread(os.path.join(images_path, filename))
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], fonts
ize=14)
cbar.set_label('Median House Value', fontsize=16)

plt.legend(fontsize=16)
save_fig("california_housing_prices_plot")
plt.show()
```

```
Saving figure california_housing_prices_plot
```



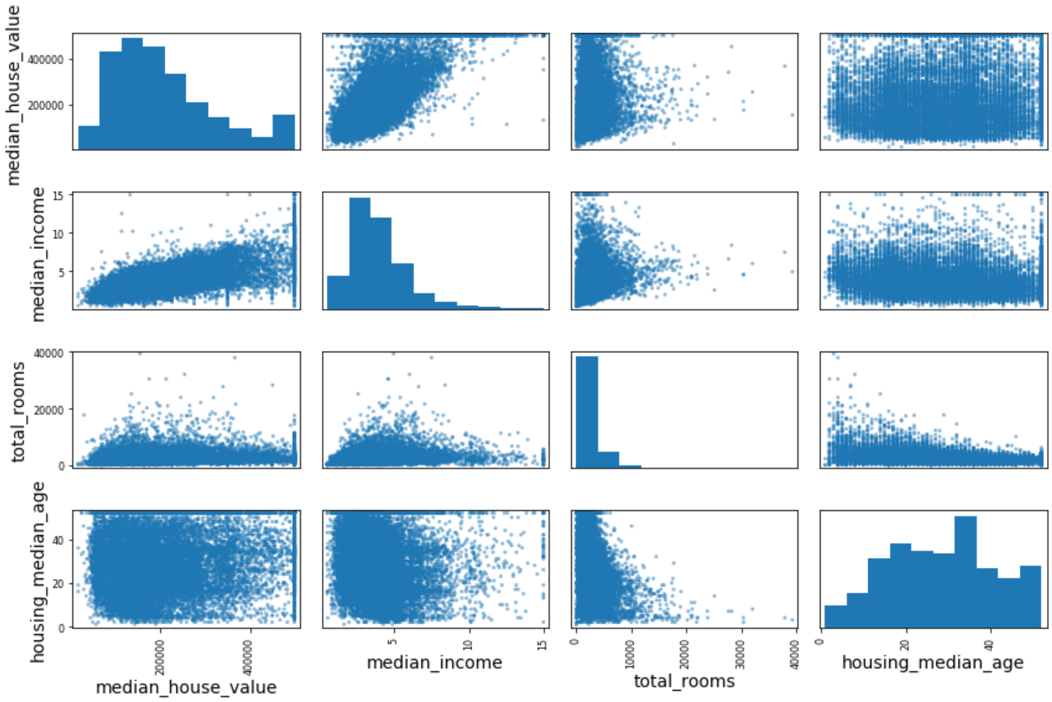
In [29]:
```
corr_matrix = housing.corr()
```

In [30]:
```
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[30]:
```
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
```

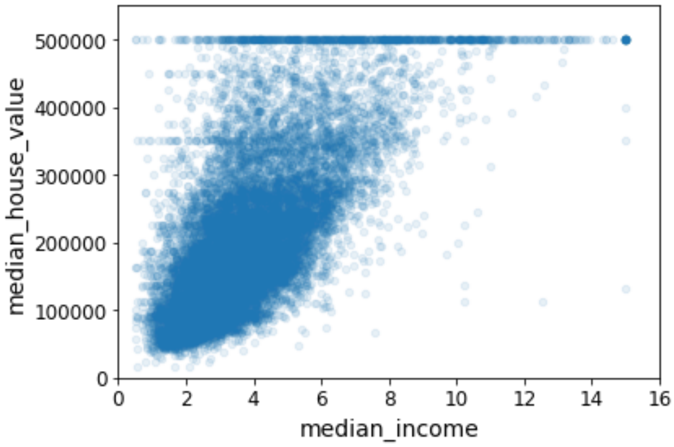
In [31]:
```
# from pandas.tools.plotting import scatter_matrix # For older versions of P
andas
from pandas.plotting import scatter_matrix

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

```
Saving figure scatter_matrix_plot
```



In [32]:
```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
plt.axis([0, 16, 0, 550000])
save_fig("income_vs_house_value_scatterplot")
```

```
Saving figure income_vs_house_value_scatterplot
```

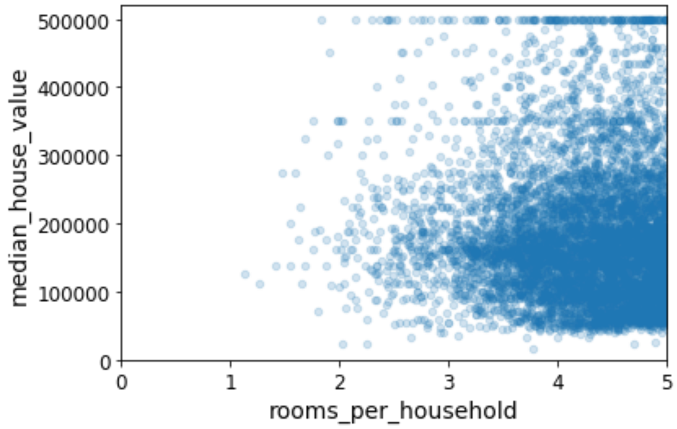


In [33]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["household
s"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_room
s"]
housing["population_per_household"]=housing["population"]/housing["household
s"]
```

In [34]:
```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[34]:
```
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
```

In [35]:
```
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_valu
e",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



In [36]:
```
housing.describe()
```

Out[36]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	h
count	16512.000000	16512.000000	16512.000000	16512.000000	16354.000000	16512.000000	16!
mean	-119.575834	35.639577	28.653101	2622.728319	534.973890	1419.790819	‹
std	2.001860	2.138058	12.574726	2138.458419	412.699041	1115.686241	:
min	-124.350000	32.540000	1.000000	6.000000	2.000000	3.000000	
25%	-121.800000	33.940000	18.000000	1443.000000	295.000000	784.000000	:
50%	-118.510000	34.260000	29.000000	2119.500000	433.000000	1164.000000	‹
75%	-118.010000	37.720000	37.000000	3141.000000	644.000000	1719.250000	(
max	-114.310000	41.950000	52.000000	39320.000000	6210.000000	35682.000000	5:

Prepare the data for Machine Learning algorithms

In [37]:
```
housing = strat_train_set.drop("median_house_value", axis=1) # drop labels f
or training set
housing_labels = strat_train_set["median_house_value"].copy()
```

In [38]:
```
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

Out[38]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	me
4629	-118.30	34.07	18.0	3759.0	NaN	3296.0	1462.0	
6068	-117.86	34.01	16.0	4632.0	NaN	3038.0	727.0	
17923	-121.97	37.35	30.0	1955.0	NaN	999.0	386.0	
13656	-117.30	34.05	6.0	2155.0	NaN	1039.0	391.0	
19252	-122.79	38.48	7.0	6837.0	NaN	3468.0	1405.0	

In [39]:
```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])     # option 1
```

Out[39]:

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_

In [40]:
```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2
```

Out[40]:

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	oc
4629	-118.30	34.07	18.0	3759.0	3296.0	1462.0	2.2708	
6068	-117.86	34.01	16.0	4632.0	3038.0	727.0	5.1762	
17923	-121.97	37.35	30.0	1955.0	999.0	386.0	4.6328	
13656	-117.30	34.05	6.0	2155.0	1039.0	391.0	1.6675	
19252	-122.79	38.48	7.0	6837.0	3468.0	1405.0	3.1662	

In [41]:
```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # opti
on 3
```

In [42]:
```
sample_incomplete_rows
```

Out[42]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	me
4629	-118.30	34.07	18.0	3759.0	433.0	3296.0	1462.0	
6068	-117.86	34.01	16.0	4632.0	433.0	3038.0	727.0	
17923	-121.97	37.35	30.0	1955.0	433.0	999.0	386.0	
13656	-117.30	34.05	6.0	2155.0	433.0	1039.0	391.0	
19252	-122.79	38.48	7.0	6837.0	433.0	3468.0	1405.0	

In [43]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
```

Remove the text attribute because median can only be calculated on numerical attributes:

In [44]:
```
housing_num = housing.drop("ocean_proximity", axis=1)
# alternatively: housing_num = housing.select_dtypes(include=[np.number])
```

In [45]:
```
imputer.fit(housing_num)
```

Out[45]:
```
SimpleImputer(strategy='median')
```

In [46]:
```
imputer.statistics_
```

Out[46]:
```
array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

Check that this is the same as manually computing the median of each attribute:

In [47]:
```
housing_num.median().values
```

Out[47]:
```
array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

Transform the training set:

In [48]:
```
X = imputer.transform(housing_num)
```

In [49]:
```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing.index)
```

In [50]:
```
housing_tr.loc[sample_incomplete_rows.index.values]
```

Out[50]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	me
4629	-118.30	34.07	18.0	3759.0	433.0	3296.0	1462.0	
6068	-117.86	34.01	16.0	4632.0	433.0	3038.0	727.0	
17923	-121.97	37.35	30.0	1955.0	433.0	999.0	386.0	
13656	-117.30	34.05	6.0	2155.0	433.0	1039.0	391.0	
19252	-122.79	38.48	7.0	6837.0	433.0	3468.0	1405.0	

In [51]:
```
imputer.strategy
```

Out[51]: 'median'

In [52]:
```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing_num.index)
```

In [53]:
```
housing_tr.head()
```

Out[53]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	me
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	

Now let's preprocess the categorical input feature, `ocean_proximity` :

In [54]:
```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(10)
```

Out[54]:

	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

In [55]:
```
from sklearn.preprocessing import OrdinalEncoder

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

Out[55]:
```
array([[0.],
       [0.],
       [4.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.]])
```

In [56]:
```
ordinal_encoder.categories_
```

Out[56]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

In [57]:
```
from sklearn.preprocessing import OneHotEncoder

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

Out[57]:
```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
        with 16512 stored elements in Compressed Sparse Row format>
```

By default, the `OneHotEncoder` class returns a sparse array, but we can convert it to a dense array if needed by calling the `toarray()` method:

In [58]:
```
housing_cat_1hot.toarray()
```

Out[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.]])
```

Alternatively, you can set `sparse=False` when creating the `OneHotEncoder` :

In [59]:
```
cat_encoder = OneHotEncoder(sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

Out[59]:
```
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.]])
```

In [60]:
```
cat_encoder.categories_
```

Out[60]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

Let's create a custom transformer to add extra attributes:

In [61]:
```
from sklearn.base import BaseEstimator, TransformerMixin

# column index
rooms_ix, bedrooms_ix, population_ix, households_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
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_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)
```

In [62]:
```
housing_extra_attribs = pd.DataFrame(
    housing_extra_attribs,
    columns=list(housing.columns)+["rooms_per_household", "population_per_ho
usehold"],
    index=housing.index)
housing_extra_attribs.head()
```

Out[62]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	me
17606	-121.89	37.29	38	1568	351	710	339	
18632	-121.93	37.05	14	679	108	306	113	
14650	-117.2	32.77	31	1952	471	936	462	
3230	-119.61	36.31	25	1847	371	1460	353	
3555	-118.59	34.23	17	6592	1525	4459	1463	

Now let's build a pipeline for preprocessing the numerical attributes:

In [63]:
```
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)
```

In [64]:
```
housing_num_tr
```

Out[64]:
```
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]])
```

In [65]:
```
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)
```

In [66]:
```
housing_prepared
```

Out[66]:
```
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.        ]])
```

In [67]:
```
housing_prepared.shape
```

Out[67]:
```
(16512, 16)
```

Select and train a model

In [68]:
```
from sklearn.linear_model import LinearRegression

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

Out[68]:
```
LinearRegression()
```

In [69]:
```
# let's try the full preprocessing pipeline on a few training instances
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]
```

Compare against the actual values:

In [70]:
```
print("Labels:", list(some_labels))
```

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

In [71]:
```
some_data_prepared
```

Out[71]:
```
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.        ]])
```

In [72]:
```
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
```

Out[72]:
```
68628.19819848923
```

In [73]:
```
from sklearn.metrics import mean_absolute_error

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

Out[73]:
```
49439.89599001897
```

Fine-tune your model

In [74]:
```
from sklearn.tree import DecisionTreeRegressor

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

Out[74]:
```
DecisionTreeRegressor(random_state=42)
```

In [75]:
```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

Out[75]:
```
RandomForestRegressor(random_state=42)
```

In [76]:
```
from sklearn.svm import SVR

svm_reg = SVR(kernel="linear")
svm_reg.fit(housing_prepared, housing_labels)
```

Out[76]:
```
SVR(kernel='linear')
```

In [77]:
```
from sklearn.model_selection import cross_val_score

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)
pd.Series(lin_rmse_scores).describe()
```

Out[77]:
```
count       10.000000
mean     69052.461363
std       2879.437224
min      64969.630564
25%      67136.363758
50%      68156.372635
75%      70982.369487
max      74739.570526
dtype: float64
```

In [78]:
```
tree_scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-tree_scores)
pd.Series(tree_rmse_scores).describe()
```

Out[78]:
```
count       10.000000
mean     71407.687660
std       2571.389745
min      66855.163639
25%      70265.554176
50%      70937.310637
75%      72132.351151
max      75585.141729
dtype: float64
```

Note: we specify `n_estimators=100` to be future-proof since the default value is going to change to 100 in Scikit-Learn 0.22 (for simplicity, this is not shown in the book).

In [79]:
```
forest_scores = cross_val_score(forest_reg, housing_prepared, housing_label
s,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
pd.Series(forest_rmse_scores).describe()
```

Out[79]:
```
count       10.000000
mean     50182.303100
std       2210.517524
min      47461.911582
25%      48803.201309
50%      49770.694467
75%      51751.217424
max      53490.106998
dtype: float64
```

In [80]:
```
svm_scores = cross_val_score(svm_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
svm_rmse_scores = np.sqrt(-svm_scores)
pd.Series(svm_rmse_scores).describe()
```

Out[80]:
```
count        10.000000
mean     111809.840096
std        2911.818591
min      105342.091420
25%      110655.068116
50%      112004.679161
75%      113667.942015
max      115675.832002
dtype: float64
```

In [81]:
```
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)
```

Out[81]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

The best hyperparameter combination found:

In [82]:
```
grid_search.best_params_
```

Out[82]:
```
{'max_features': 8, 'n_estimators': 30}
```

In [83]:
```
grid_search.best_estimator_
```

Out[83]:
```
RandomForestRegressor(max_features=8, n_estimators=30, random_state=42)
```

Let's look at the score of each hyperparameter combination tested during the grid search:

In [84]:
```
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': 1
0}
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': 1
0}
```

In [85]:
```
pd.DataFrame(grid_search.cv_results_)
```

Out[85]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_features	param_n_estimator
0	0.060681	0.001426	0.002983	0.000037	2	
1	0.194322	0.003589	0.008981	0.000028	2	1
2	0.583491	0.005095	0.025140	0.000773	2	3
3	0.093365	0.002416	0.003011	0.000023	4	
4	0.302422	0.002909	0.008978	0.000047	4	1
5	0.963829	0.043499	0.026337	0.000795	4	3
6	0.125465	0.002487	0.003011	0.000019	6	
7	0.422492	0.007103	0.009374	0.000476	6	1
8	1.309725	0.022151	0.025704	0.000436	6	3
9	0.175359	0.009939	0.003371	0.000525	8	
10	0.630333	0.043427	0.010558	0.000819	8	1
11	1.801205	0.017485	0.028310	0.000492	8	3
12	0.105512	0.001595	0.003991	0.000018	2	
13	0.378807	0.027508	0.013155	0.001940	2	1
14	0.169348	0.005897	0.004988	0.000631	3	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_features	param_n_estimatoı
15	0.537342	0.051332	0.012968	0.001538	3	1
16	0.191867	0.014059	0.004980	0.000628	4	

In [86]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {
        'n_estimators': randint(low=1, high=200),
        'max_features': randint(low=1, high=8),
    }

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distri
bs,
                                n_iter=10, cv=5, scoring='neg_mean_squared_e
rror', random_state=42)
rnd_search.fit(housing_prepared, housing_labels)
```

Out[86]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
                   param_distributions={'max_features': <scipy.stats._distn_i
nfrastructure.rv_frozen object at 0x000001DD43194548>,
                                        'n_estimators': <scipy.stats._distn_i
nfrastructure.rv_frozen object at 0x000001DD43194B48>},
                   random_state=42, scoring='neg_mean_squared_error')
```

In [87]:
```
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
49150.70756927707 {'max_features': 7, 'n_estimators': 180}
51389.889203389284 {'max_features': 5, 'n_estimators': 15}
50796.155224308866 {'max_features': 3, 'n_estimators': 72}
50835.13360315349 {'max_features': 5, 'n_estimators': 21}
49280.9449827171 {'max_features': 7, 'n_estimators': 122}
50774.90662363929 {'max_features': 3, 'n_estimators': 75}
50682.78888164288 {'max_features': 3, 'n_estimators': 88}
49608.99608105296 {'max_features': 5, 'n_estimators': 100}
50473.61930350219 {'max_features': 3, 'n_estimators': 150}
64429.84143294435 {'max_features': 5, 'n_estimators': 2}
```

In [88]:
```
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

Out[88]:
```
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])
```

In [89]:
```
extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
#cat_encoder = cat_pipeline.named_steps["cat_encoder"] # old solution
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)
```

Out[89]:
```
[(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')]
```

In [90]:
```
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)
```

In [91]:
```
final_rmse
```

Out[91]:
```
47730.22690385927
```

We can compute a 95% confidence interval for the test RMSE:

In [92]:
```
from scipy import stats

confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
                         scale=stats.sem(squared_errors)))
```

Out[92]:
```
array([45685.10470776, 49691.25001878])
```

Congratulations! You already know quite a lot about Machine Learning. :)

In [ ]:
```
```