Get the Data

Download the Data

```
import pandas as pd

def load_housing_data():
    """
    In Windows, set csv_path = ".\\dataset\housing.csv"
    """
    csv_path = "./housing.csv"
    return pd.read_csv(csv_path)
```

Take a Quick Look at the Data Structure

```
housing = load_housing_data()
housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Next steps: Generate code with housing View recommended plots New interactive sheet

```
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
```

```
housing["ocean_proximity"].value_counts()
```

ocean_proximity	count
<1H OCEAN	9136
INLAND	6551
NEAR OCEAN	2658
NEAR BAY	2290
ISLAND	5

dtype: int64

```
housing.describe()
```

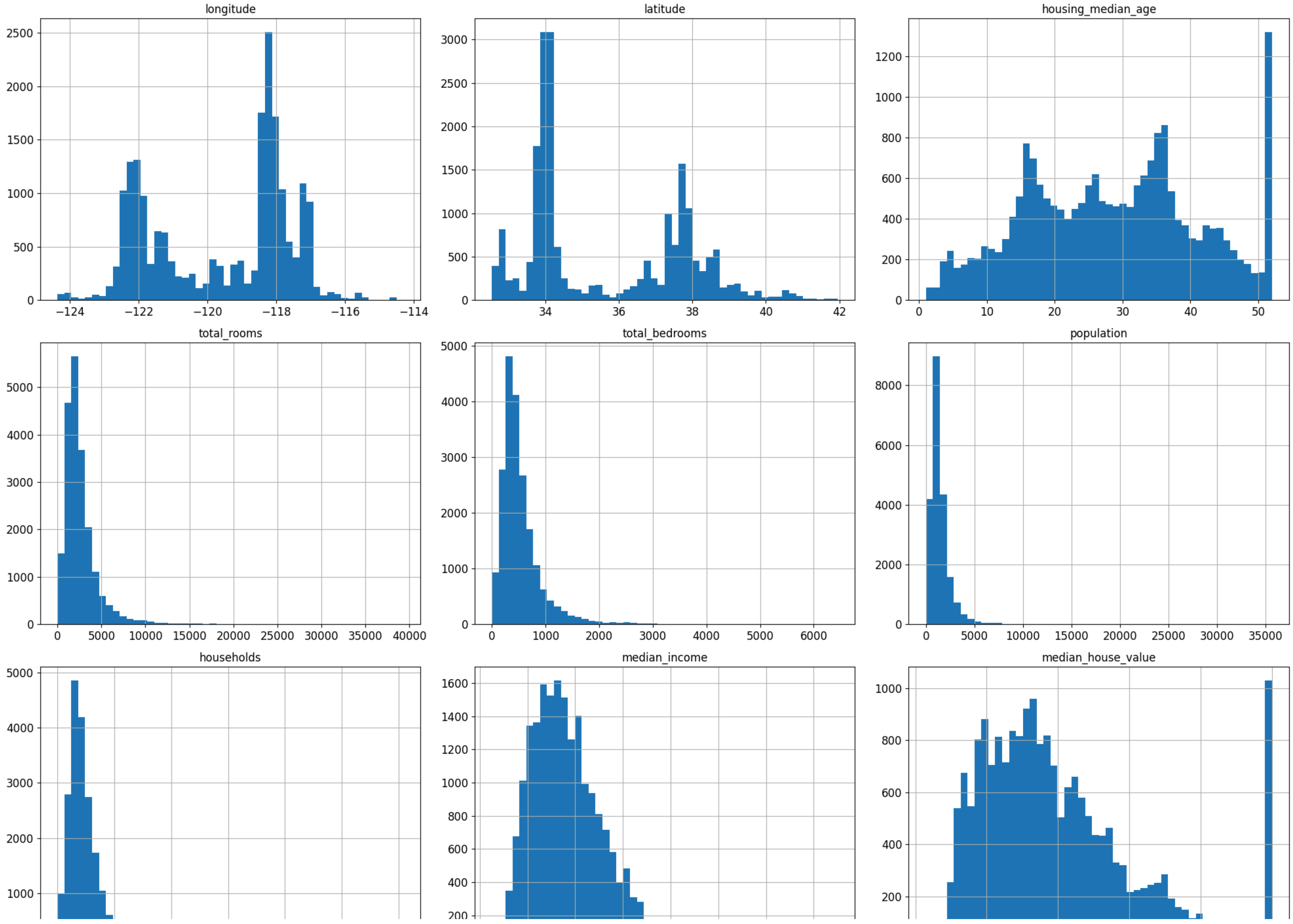
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

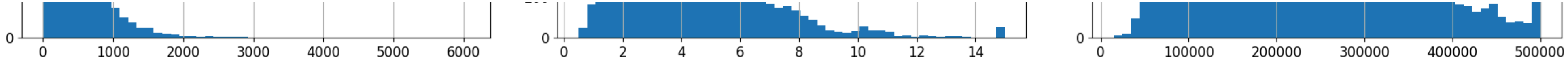
Set up the plots

```
# 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=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)

%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
```





Create a Test Set

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

import numpy as np

# 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]

train_set, test_set = split_train_test(housing, 0.2)
len(train_set)
```

```
16512
```

```
len(test_set)
```

4128

```
from zlib import crc32

def test_set_check(identifier, test_ratio):
    return crc32(np.int64(identifier)) & 0xffffffff < test_ratio * 2**32

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]
```

The implementation of `test_set_check()` above works fine in both Python 2 and Python 3. In earlier releases, the following implementation was proposed, which supported any hash function, but was much slower and did not support Python 2:

```
import hashlib

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

If you want an implementation that supports any hash function and is compatible with both Python 2 and Python 3, here is one:

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

```
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")
```

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

```
test_set.head()
```

	index	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity	id
8	8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	NEAR BAY	-122222.16
10	10	-122.26	37.85	52.0	2202.0	434.0	910.0	402.0	3.2031	281500.0	NEAR BAY	-122222.15
11	11	-122.26	37.85	52.0	3503.0	752.0	1504.0	734.0	3.2705	241800.0	NEAR BAY	-122222.15
12	12	-122.26	37.85	52.0	2491.0	474.0	1098.0	468.0	3.0750	213500.0	NEAR BAY	-122222.15
13	13	-122.26	37.84	52.0	696.0	191.0	345.0	174.0	2.6736	191300.0	NEAR BAY	-122222.16

Next steps: Generate code with `test_set` View recommended plots New interactive sheet

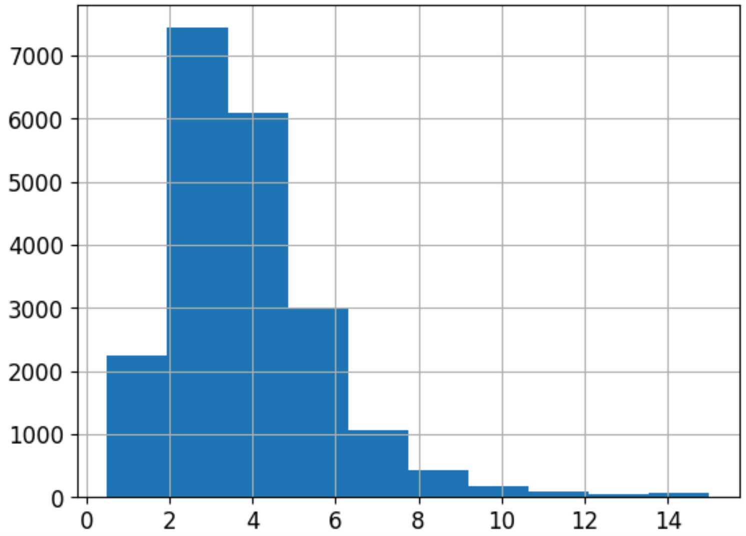
```
from sklearn.model_selection import train_test_split

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

test_set.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	1.6812	47700.0	INLAND
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	2.5313	45800.0	INLAND
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	3.4801	500001.0	NEAR BAY
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376	218600.0	<1H OCEAN
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	3.7250	278000.0	NEAR OCEAN

Next steps: Generate code with `test_set` View recommended plots New interactive sheet

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

<Axes: >



```
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])
```

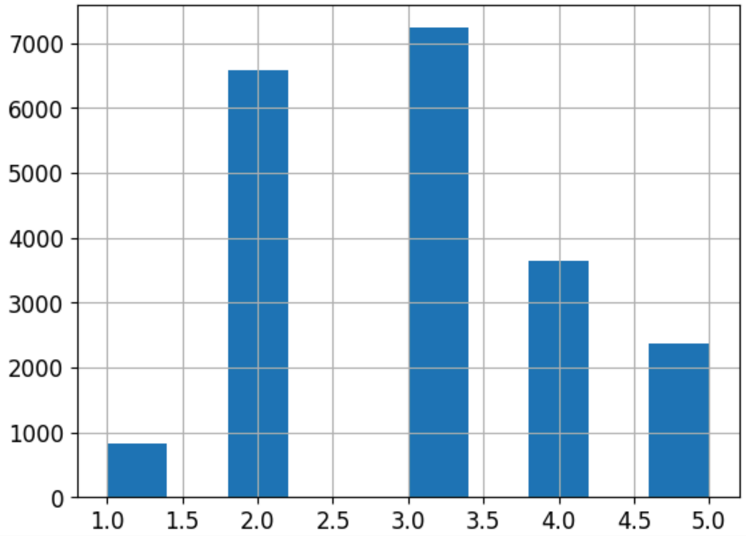
```
housing["income_cat"].value_counts()
```

	count
income_cat	
3	7236
2	6581
4	3639
5	2362
1	822

dtype: int64

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

```
<Axes: >
```




```python
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]

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

income_cat	count
3	0.350533
2	0.318798
4	0.176357
5	0.114341
1	0.039971

dtype: float64

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

income_cat	count
3	0.350581
2	0.318847
4	0.176308
5	0.114438
1	0.039826

dtype: float64

```
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

compare_props
```

income_cat	Overall	Stratified	Random	Rand. %error	Strat. %error
1	0.039826	0.039971	0.040213	0.973236	0.364964
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	0.114341	0.109496	-4.318374	-0.084674

Next steps: Generate code with compare_props | View recommended plots | New interactive sheet

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

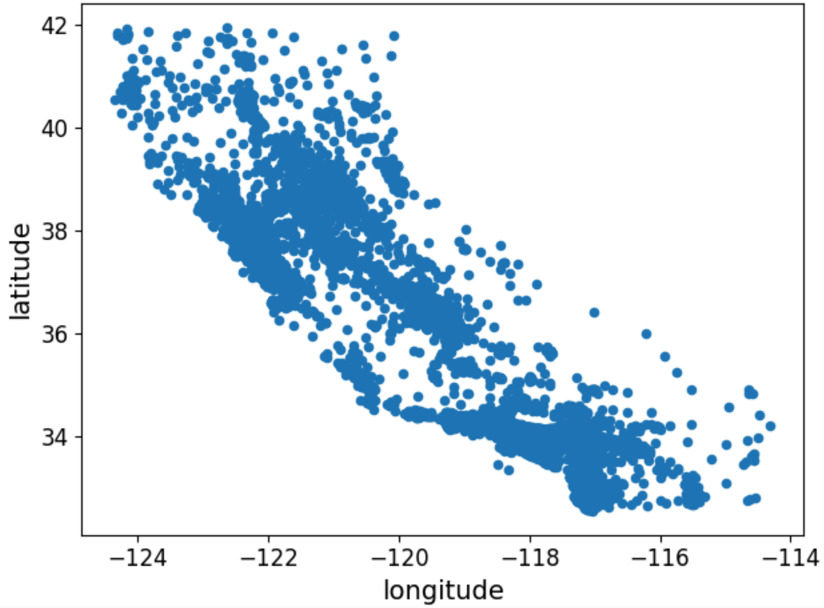
Discover and Visualize the Data to Gain Insights

```
housing = strat_train_set.copy()
```

Visualizing Geographical Data

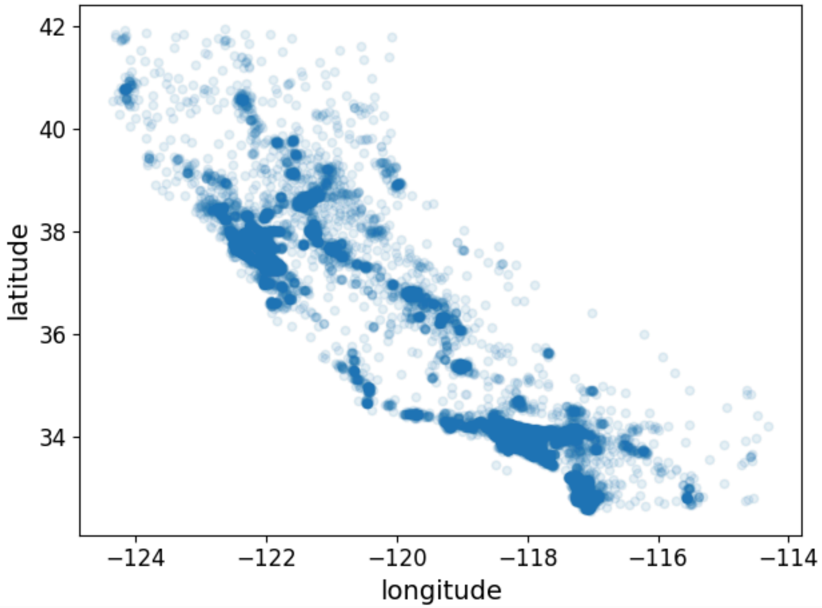
```
housing.plot(kind="scatter", x="longitude", y="latitude")
save_fig("bad_visualization_plot")
```

```
Saving figure bad_visualization_plot
```




```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
save_fig("better_visualization_plot")
```

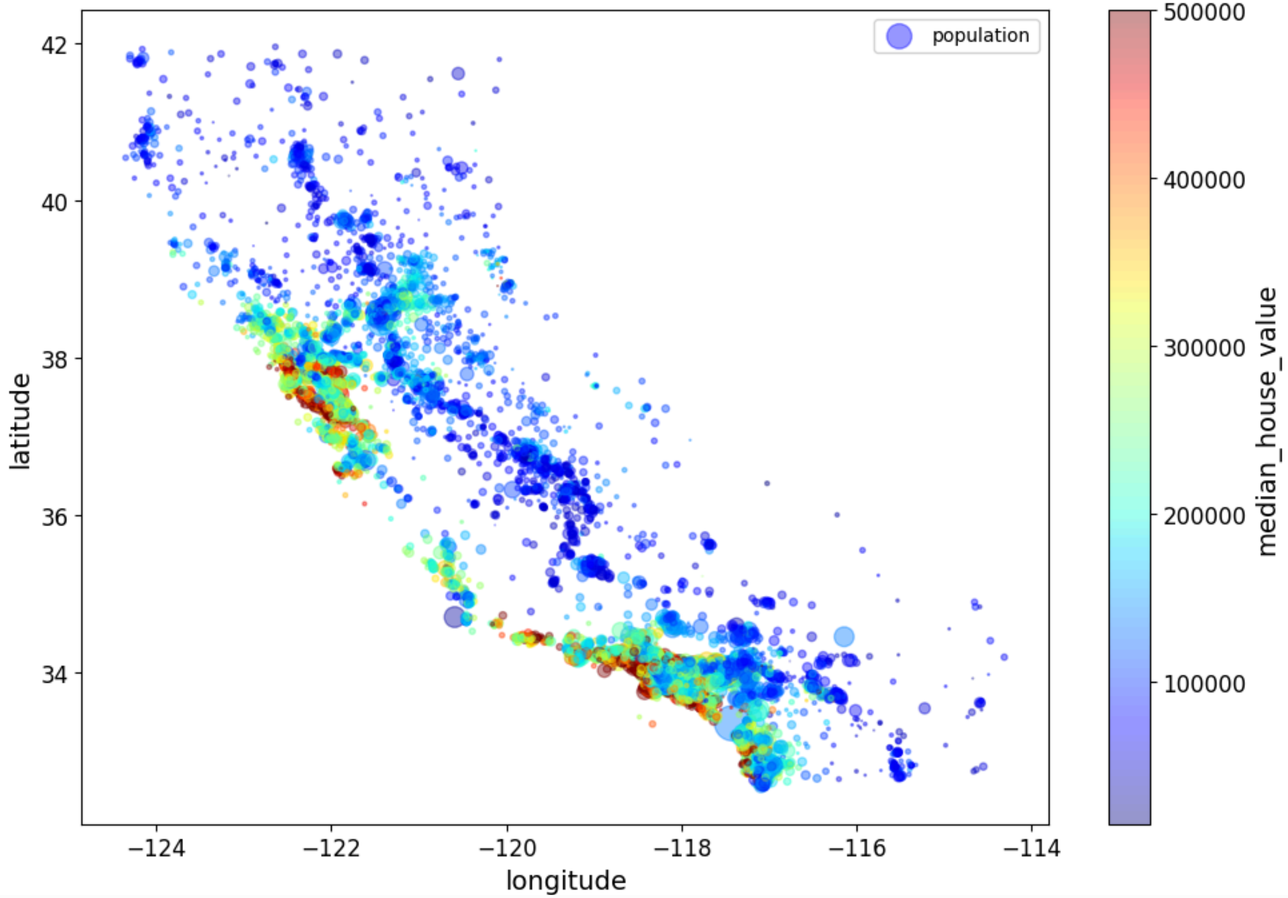
Saving figure better_visualization_plot



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). Thanks to Wilmer Arellano for pointing it out.

```
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
```



Looking for Correlations

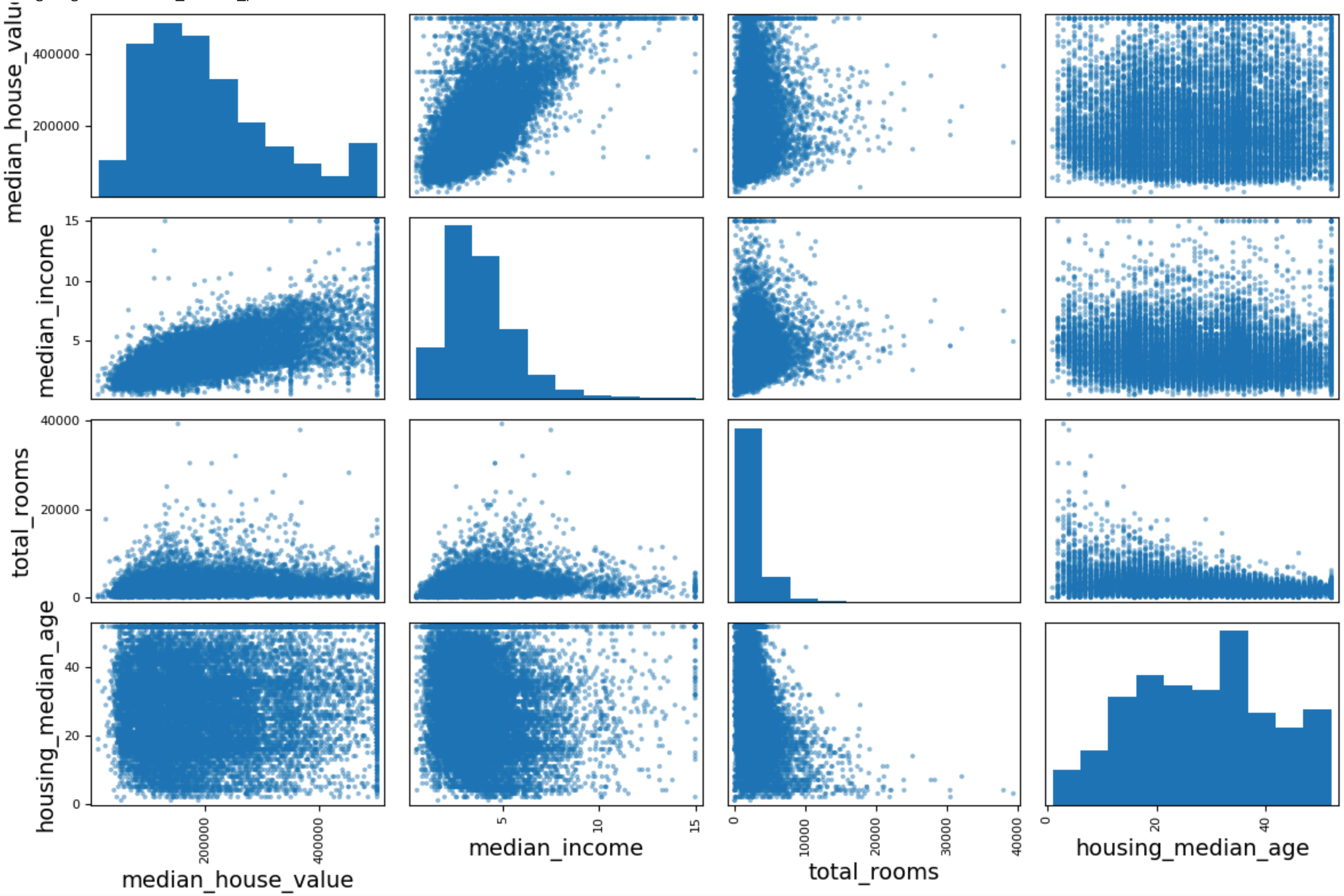
```
# Drop the non-numerical column before calculating correlation
housing_numeric = housing.drop("ocean_proximity", axis=1)

# Calculate the correlation matrix on the numeric data
corr_matrix = housing_numeric.corr()

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

	median_house_value
median_house_value	1.000000
median_income	0.687151
total_rooms	0.135140
housing_median_age	0.114146
households	0.064590
total_bedrooms	0.047781
population	-0.026882
longitude	-0.047466
latitude	-0.142673

dtype: float64

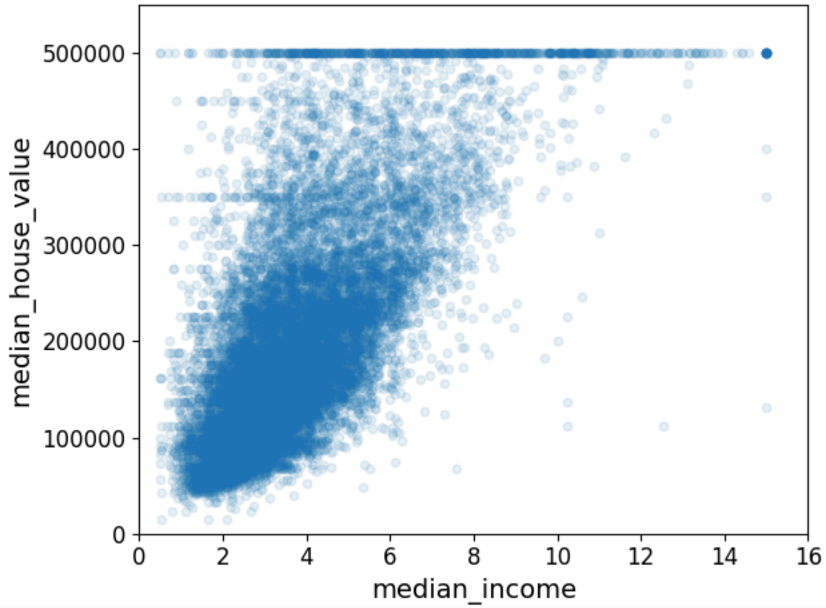
```
# from pandas.tools.plotting import scatter_matrix # For older versions of Pandas
from pandas.plotting import scatter_matrix

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
```




```
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
```



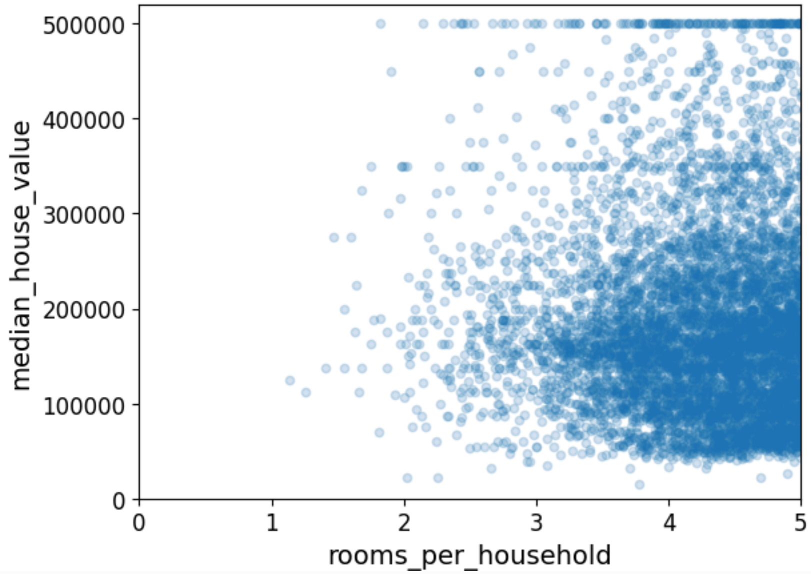
Experimenting with Attribute Combinations

```
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"]

# Drop the non-numerical column before calculating correlation
# This ensures that the correlation is calculated only on numeric columns,
# including the newly created ones.
housing_numeric = housing.drop("ocean_proximity", axis=1)

# Calculate the correlation matrix on the numeric data
corr_matrix = housing_numeric.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

	median_house_value
median_house_value	1.000000
median_income	0.687151
rooms_per_household	0.146255
total_rooms	0.135140
housing_median_age	0.114146
households	0.064590
total_bedrooms	0.047781
population_per_household	-0.021991
population	-0.026882
longitude	-0.047466
latitude	-0.142673
bedrooms_per_room	-0.259952

dtype: float64

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




```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	rooms_per_household	bedrooms_per_room	population_
count	16512.000000	16512.000000	16512.000000	16512.000000	16354.000000	16512.000000	16512.000000	16512.000000	16512.000000	16512.000000	16354.000000	
mean	-119.575635	35.639314	28.653404	2622.539789	534.914639	1419.687379	497.011810	3.875884	207005.322372	5.440406	0.212873	
std	2.001828	2.137963	12.574819	2138.417080	412.665649	1115.663036	375.696156	1.904931	115701.297250	2.611696	0.057378	
min	-124.350000	32.540000	1.000000	6.000000	2.000000	3.000000	2.000000	0.499900	14999.000000	1.130435	0.100000	
25%	-121.800000	33.940000	18.000000	1443.000000	295.000000	784.000000	279.000000	2.566950	119800.000000	4.442168	0.175304	
50%	-118.510000	34.260000	29.000000	2119.000000	433.000000	1164.000000	408.000000	3.541550	179500.000000	5.232342	0.203027	
75%	-118.010000	37.720000	37.000000	3141.000000	644.000000	1719.000000	602.000000	4.745325	263900.000000	6.056361	0.239816	
max	-114.310000	41.950000	52.000000	39320.000000	6210.000000	35682.000000	5358.000000	15.000100	500001.000000	141.909091	1.000000	

Prepare the Data for Machine Learning Algorithms

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

Data Cleaning

In the book 3 options are listed:

```
housing.dropna(subset=["total_bedrooms"])    # option 1
housing.drop("total_bedrooms", axis=1)       # option 2
median = housing["total_bedrooms"].median()  # option 3
housing["total_bedrooms"].fillna(median, inplace=True)
```

To demonstrate each of them, let's create a copy of the housing dataset, but keeping only the rows that contain at least one null. Then it will be easier to visualize exactly what each option does:

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

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
1606	-122.08	37.88	26.0	2947.0	NaN	825.0	626.0	2.9330	NEAR BAY
10915	-117.87	33.73	45.0	2264.0	NaN	1970.0	499.0	3.4193	<1H OCEAN
19150	-122.70	38.35	14.0	2313.0	NaN	954.0	397.0	3.7813	<1H OCEAN
4186	-118.23	34.13	48.0	1308.0	NaN	835.0	294.0	4.2891	<1H OCEAN
16885	-122.40	37.58	26.0	3281.0	NaN	1145.0	480.0	6.3580	NEAR OCEAN

Next steps: Generate code with sample_incomplete_rows | View recommended plots | New interactive sheet

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

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity

```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2
```

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	ocean_proximity
1606	-122.08	37.88	26.0	2947.0	825.0	626.0	2.9330	NEAR BAY
10915	-117.87	33.73	45.0	2264.0	1970.0	499.0	3.4193	<1H OCEAN
19150	-122.70	38.35	14.0	2313.0	954.0	397.0	3.7813	<1H OCEAN
4186	-118.23	34.13	48.0	1308.0	835.0	294.0	4.2891	<1H OCEAN
16885	-122.40	37.58	26.0	3281.0	1145.0	480.0	6.3580	NEAR OCEAN

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

```
/tmp/ipython-input-144-760120979.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the

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

```
sample_incomplete_rows
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
1606	-122.08	37.88	26.0	2947.0	433.0	825.0	626.0	2.9330	NEAR BAY
10915	-117.87	33.73	45.0	2264.0	433.0	1970.0	499.0	3.4193	<1H OCEAN
19150	-122.70	38.35	14.0	2313.0	433.0	954.0	397.0	3.7813	<1H OCEAN
4186	-118.23	34.13	48.0	1308.0	433.0	835.0	294.0	4.2891	<1H OCEAN
16885	-122.40	37.58	26.0	3281.0	433.0	1145.0	480.0	6.3580	NEAR OCEAN

Next steps: Generate code with sample_incomplete_rows | View recommended plots | New interactive sheet

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
```

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

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

```
imputer.fit(housing_num)
```

```
▾ SimpleImputer
SimpleImputer(strategy='median')
```

```
imputer.statistics_
```

```
array([-118.51    ,   34.26    ,   29.     , 2119.     ,  433.     ,
        1164.     ,  408.     ,    3.54155])
```

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

```
housing_num.median().values
```

```
array([-118.51    ,   34.26    ,   29.     , 2119.     ,  433.     ,
        1164.     ,  408.     ,    3.54155])
```

Transform the training set:

```
X = imputer.transform(housing_num)
```

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing.index)
```

```
housing_tr.loc[sample_incomplete_rows.index.values]
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
1606	-122.08	37.88	26.0	2947.0	433.0	825.0	626.0	2.9330
10915	-117.87	33.73	45.0	2264.0	433.0	1970.0	499.0	3.4193
19150	-122.70	38.35	14.0	2313.0	433.0	954.0	397.0	3.7813
4186	-118.23	34.13	48.0	1308.0	433.0	835.0	294.0	4.2891
16885	-122.40	37.58	26.0	3281.0	433.0	1145.0	480.0	6.3580

```
imputer.strategy
```

```
'median'
```

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing_num.index)
```

```
housing_tr.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
12655	-121.46	38.52	29.0	3873.0	797.0	2237.0	706.0	2.1736
15502	-117.23	33.09	7.0	5320.0	855.0	2015.0	768.0	6.3373
2908	-119.04	35.37	44.0	1618.0	310.0	667.0	300.0	2.8750
14053	-117.13	32.75	24.0	1877.0	519.0	898.0	483.0	2.2264
20496	-118.70	34.28	27.0	3536.0	646.0	1837.0	580.0	4.4964

Next steps: Generate code with `housing_tr` View recommended plots New interactive sheet

Handling Text and Categorical Attributes

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

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

	ocean_proximity
12655	INLAND
15502	NEAR OCEAN
2908	INLAND
14053	NEAR OCEAN
20496	<1H OCEAN
1481	NEAR BAY
18125	<1H OCEAN
5830	<1H OCEAN
17989	<1H OCEAN
4861	<1H OCEAN

Next steps: Generate code with `housing_cat` View recommended plots New interactive sheet

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
```

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

```
array([[1.],
       [4.],
       [1.],
       [4.],
       [0.],
       [3.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

```
ordinal_encoder.categories_
```

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

```
from sklearn.preprocessing import OneHotEncoder

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

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
	with 16512 stored elements and shape (16512, 5)>
```

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:

```
housing_cat_1hot.toarray()
```

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

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

```
from sklearn.preprocessing import OneHotEncoder

# Remove the sparse=False argument as it's not supported in this scikit-learn version
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
# Convert the sparse output to a dense array
housing_cat_1hot = housing_cat_1hot.toarray()
housing_cat_1hot
```

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

```
cat_encoder.categories_
```

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

Custom Transformers

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

```
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)
```

Note that I hard coded the indices (3, 4, 5, 6) for concision and clarity in the book, but it would be much cleaner to get them dynamically, like this:

```
col_names = "total_rooms", "total_bedrooms", "population", "households"
rooms_ix, bedrooms_ix, population_ix, households_ix = [
    housing.columns.get_loc(c) for c in col_names] # get the column indices
```

Also, `housing_extra_attribs` is a NumPy array, we've lost the column names (unfortunately, that's a problem with Scikit-Learn). To recover a `DataFrame`, you could run this:

```
housing_extra_attribs = pd.DataFrame(
    housing_extra_attribs,
    columns=list(housing.columns)+["rooms_per_household", "population_per_household"],
    index=housing.index)
housing_extra_attribs.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	rooms_per_household	population_per_household
12655	-121.46	38.52	29.0	3873.0	797.0	2237.0	706.0	2.1736	INLAND	5.485836	3.168555
15502	-117.23	33.09	7.0	5320.0	855.0	2015.0	768.0	6.3373	NEAR OCEAN	6.927083	2.623698
2908	-119.04	35.37	44.0	1618.0	310.0	667.0	300.0	2.875	INLAND	5.393333	2.223333
14053	-117.13	32.75	24.0	1877.0	519.0	898.0	483.0	2.2264	NEAR OCEAN	3.886128	1.859213
20496	-118.7	34.28	27.0	3536.0	646.0	1837.0	580.0	4.4964	<1H OCEAN	6.096552	3.167241

Next steps: Generate code with housing_extra_attribs View recommended plots New interactive sheet

Transformation Pipelines

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

```
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)

housing_num_tr
```

```
array([[-0.94135046,  1.34743822,  0.02756357, ...,  0.01739526,
         0.00622264, -0.12112176],
       [ 1.17178212, -1.19243966, -1.72201763, ...,  0.56925554,
        -0.04081077, -0.81086696],
       [ 0.26758118, -0.1259716 ,  1.22045984, ..., -0.01802432,
        -0.07537122, -0.33827252],
       ...,
       [-1.5707942 ,  1.31001828,  1.53856552, ..., -0.5092404 ,
        -0.03743619,  0.32286937],
       [-1.56080303,  1.2492109 , -1.1653327 , ...,  0.32814891,
        -0.05915604, -0.45702273],
       [-1.28105026,  2.02567448, -0.13148926, ...,  0.01407228,
         0.00657083, -0.12169672]])
```

```
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)

housing_prepared
```

```
array([[-0.94135046,  1.34743822,  0.02756357, ...,  0.        ,
         0.        ,  0.        ],
       [ 1.17178212, -1.19243966, -1.72201763, ...,  0.        ,
         0.        ,  1.        ],
       [ 0.26758118, -0.1259716 ,  1.22045984, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [-1.5707942 ,  1.31001828,  1.53856552, ...,  0.        ,
         0.        ,  0.        ],
       [-1.56080303,  1.2492109 , -1.1653327 , ...,  0.        ,
         0.        ,  0.        ],
       [-1.28105026,  2.02567448, -0.13148926, ...,  0.        ,
         0.        ,  0.        ]])
```

```
housing_prepared.shape
```

```
(16512, 16)
```

For reference, here is the old solution based on a `DataFrameSelector` transformer (to just select a subset of the Pandas `DataFrame` columns), and a `FeatureUnion`:

```
from sklearn.base import BaseEstimator, TransformerMixin

# Create a class to select numerical or categorical columns
class OldDataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

Now let's join all these components into a big pipeline that will preprocess both the numerical and the categorical features:

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

old_num_pipeline = Pipeline([
        ('selector', OldDataFrameSelector(num_attribs)),
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

old_cat_pipeline = Pipeline([
        ('selector', OldDataFrameSelector(cat_attribs)),
        # Remove the sparse=False argument as it's not supported in this scikit-learn version
        ('cat_encoder', OneHotEncoder()),
    ])

from sklearn.pipeline import FeatureUnion

old_full_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", old_num_pipeline),
        ("cat_pipeline", old_cat_pipeline),
    ])

old_housing_prepared = old_full_pipeline.fit_transform(housing)
old_housing_prepared
```

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
	with 198144 stored elements and shape (16512, 16)>
```

The result is the same as with the `ColumnTransformer`:

```
# Ensure the output of the pipelines are dense arrays for comparison
# housing_prepared is already a dense NumPy array, so no need to call .toarray()
housing_prepared_dense = housing_prepared
# old_housing_prepared is a sparse matrix, so we need to convert it to a dense array
old_housing_prepared_dense = old_housing_prepared.toarray()

# Now compare the dense arrays
np.allclose(housing_prepared_dense, old_housing_prepared_dense)
```

```
True
```

Select and Train a Model

Training and Evaluating on the Training Set

```
from sklearn.linear_model import LinearRegression

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

```
▾ LinearRegression
LinearRegression()
```

```
# 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: [ 85657.90192014 305492.60737488 152056.46122456 186095.70946094
 244550.67966089]
```

Compare against the actual values:

```
print("Labels:", list(some_labels))
```

```
Labels: [72100.0, 279600.0, 82700.0, 112500.0, 238300.0]
```

```
some_data_prepared
```

```
array([[-0.94135046,  1.34743822,  0.02756357,  0.58477745,  0.64037127,
         0.73260236,  0.55628602, -0.8936472 ,  0.01739526,  0.00622264,
        -0.12112176,  0.        ,  1.        ,  0.        ,  0.        ,
         0.        ],
       [ 1.17178212, -1.19243966, -1.72201763,  1.26146668,  0.78156132,
         0.53361152,  0.72131799,  1.292168  ,  0.56925554, -0.04081077,
        -0.81086696,  0.        ,  0.        ,  0.        ,  0.        ,
         1.        ],
       [ 0.26758118, -0.1259716 ,  1.22045984, -0.46977281, -0.54513828,
        -0.67467519, -0.52440722, -0.52543365, -0.01802432, -0.07537122,
        -0.33827252,  0.        ,  1.        ,  0.        ,  0.        ,
         0.        ],
       [ 1.22173797, -1.35147437, -0.37006852, -0.34865152, -0.03636724,
        -0.46761716, -0.03729672, -0.86592882, -0.59513997, -0.10680295,
         0.96120521,  0.        ,  0.        ,  0.        ,  0.        ,
         1.        ],
       [ 0.43743108, -0.63581817, -0.13148926,  0.42717947,  0.27279028,
         0.37406031,  0.22089846,  0.32575178,  0.2512412 ,  0.00610923,
        -0.47451338,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ]])
```

```
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
```

```
np.float64(68627.87390018745)
```

Note: since Scikit-Learn 0.22, you can get the RMSE directly by calling the `mean_squared_error()` function with `squared=False`.

```
"""
from sklearn.metrics import mean_absolute_error

lin_mae = mean_absolute_error(housing_labels, housing_predictions)
```

```
lin_mae
"""
```

```
'\nfrom sklearn.metrics import mean_absolute_error\n\nlin_mae = mean_absolute_error(housing_labels, housing_predictions)\nlin_mae\n'
```

```
from sklearn.tree import DecisionTreeRegressor

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

```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

```
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

```
np.float64(0.0)
```

Better Evaluation Using Cross-Validation

```
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)

def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

display_scores(tree_rmse_scores)
```

```
Scores: [72831.45749112 69973.18438322 69528.56551415 72517.78229792
 69145.50006909 79094.74123727 68960.045444   73344.50225684
 69826.02473916 71077.09753998]
Mean: 71629.89009727491
Standard deviation: 2914.035468468928
```

```
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: [71762.76364394 64114.99166359 67771.17124356 68635.19072082
 66846.14089488 72528.03725385 73997.08050233 68802.33629334
 66443.28836884 70139.79923956]
Mean: 69104.07998247063
Standard deviation: 2880.3282098180694
```

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).

```
from sklearn.ensemble import RandomForestRegressor

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

```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

```
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

```
np.float64(18650.698705770003)
```

```
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: [51559.63379638 48737.57100062 47210.51269766 51875.21247297
 47577.50470123 51863.27467888 52746.34645573 50065.1762751
 48664.66818196 54055.90894609]
Mean: 50435.58092066179
Standard deviation: 2203.3381412764606
```

```
scores = cross_val_score(lin_reg, housing_prepared, housing_labels, scoring="neg_mean_squared_error",
pd.Series(np.sqrt(-scores)).describe()
```

	0
count	10.000000
mean	69104.079982
std	3036.132517
min	64114.991664
25%	67077.398482
50%	68718.763507
75%	71357.022543
max	73997.080502

dtype: float64

Fine-Tune Your Model

Grid Search

```
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)
```

```
GridSearchCV
best_estimator_:
RandomForestRegressor
RandomForestRegressor
```

The best hyperparameter combination found:

```
grid_search.best_params_
```

```
{'max_features': 8, 'n_estimators': 30}
```

```
grid_search.best_estimator_
```

```
RandomForestRegressor
RandomForestRegressor(max_features=8, n_estimators=30, random_state=42)
```