Income Category	Overall %	Stratified %	Random %	Strat. Error %	Rand. Error %
1	3.98	4.00	4.24	0.36	6.45
2	31.88	31.88	30.74	-0.02	-3.59
3	35.06	35.05	34.52	-0.01	-1.53
4	17.63	17.64	18.41	0.03	4.42
5	11.44	11.43	12.09	-0.08	5.63

Figure 2-10. Sampling bias comparison of stratified versus purely random sampling

You won't use the `income_cat` column again, so you might as well drop it, reverting the data back to its original state:

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

We spent quite a bit of time on test set generation for a good reason: this is an often neglected but critical part of a machine learning project. Moreover, many of these ideas will be useful later when we discuss cross-validation. Now it's time to move on to the next stage: exploring the data.

Explore and Visualize the Data to Gain Insights

So far you have only taken a quick glance at the data to get a general understanding of the kind of data you are manipulating. Now the goal is to go into a little more depth.

First, make sure you have put the test set aside and you are only exploring the training set. Also, if the training set is very large, you may want to sample an exploration set, to make manipulations easy and fast during the exploration phase. In this case, the training set is quite small, so you can just work directly on the full set. Since you're going to experiment with various transformations of the full

training set, you should make a copy of the original so you can revert to it afterwards:

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

Visualizing Geographical Data

Because the dataset includes geographical information (latitude and longitude), it is a good idea to create a scatterplot of all the districts to visualize the data (Figure 2-11):

```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True)
plt.show()
```

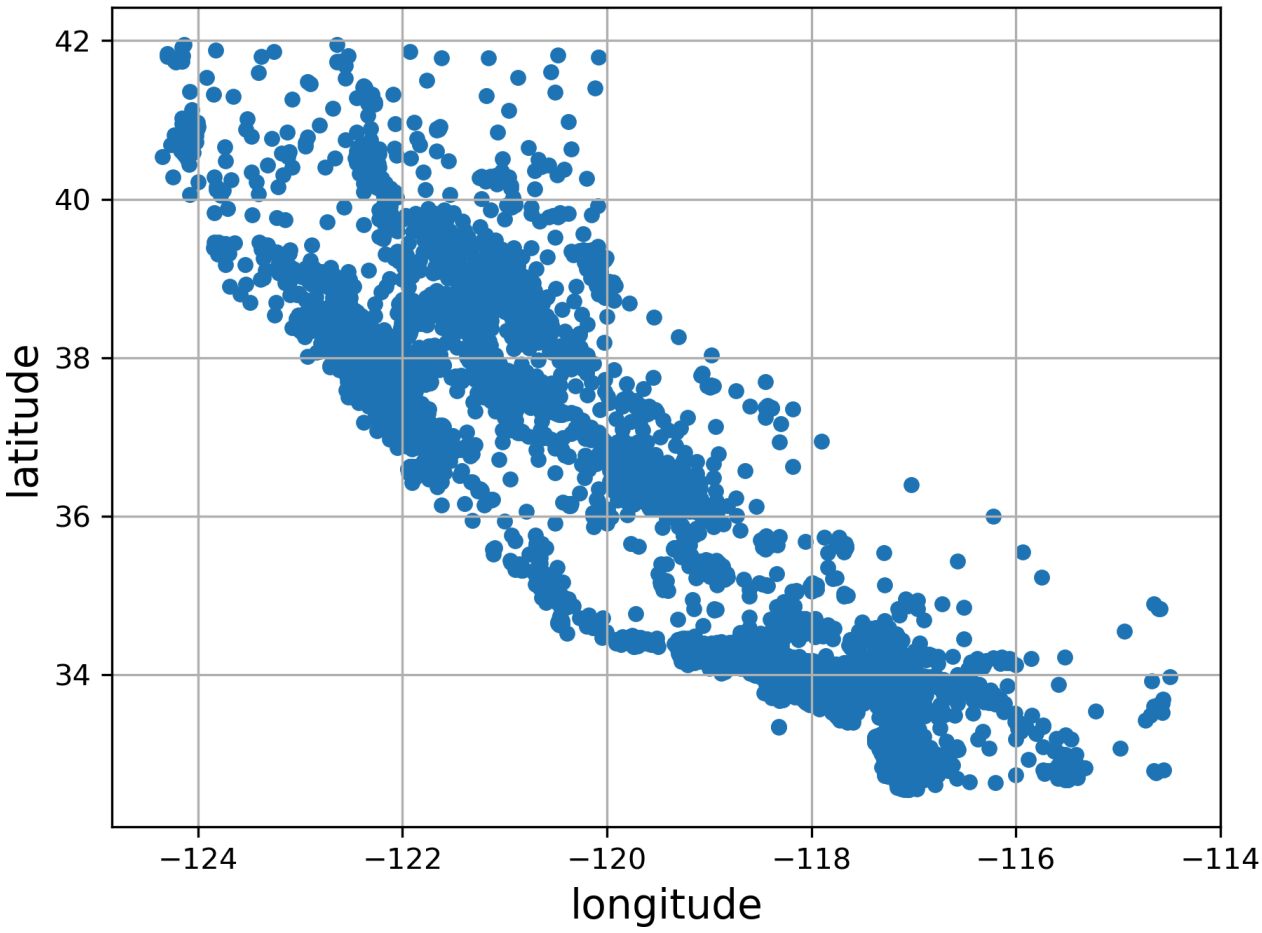

Figure 2-11. A geographical scatterplot of the data

This looks like California all right, but other than that it is hard to see any particular pattern. Setting the `alpha` option to `0.2` makes it much easier to visualize the places where there is a high density of data points (Figure 2-12):

```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True, alpha=0.2)
plt.show()
```

Now that's much better: you can clearly see the high-density areas, namely the Bay Area and around Los Angeles and San Diego, plus a long line of fairly high-density areas in the Central Valley (in particular, around Sacramento and Fresno).

Our brains are very good at spotting patterns in pictures, but you may need to play around with visualization parameters to make the patterns stand out.

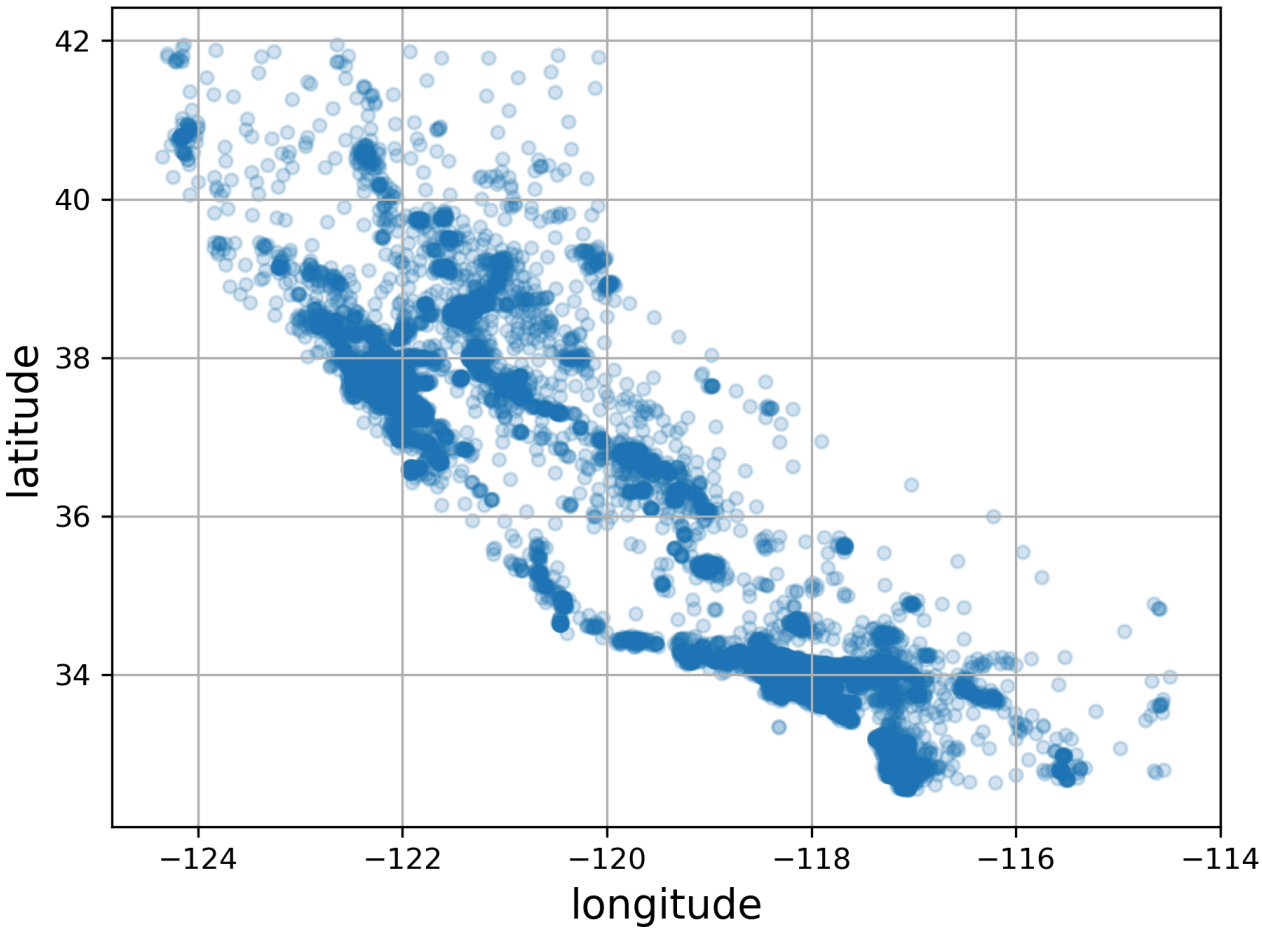

Figure 2-12. A better visualization that highlights high-density areas

Next, you look at the housing prices (Figure 2-13). The radius of each circle represents the district's population (option `s`), and the color represents the price (op-

tion `c`). Here you use a predefined color map (option `cmap`) called `jet`, which ranges from blue (low values) to red (high prices):[8]

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

This image tells you that the housing prices are very much related to the location (e.g., close to the ocean) and to the population density, as you probably knew already. A clustering algorithm should be useful for detecting the main cluster and for adding new features that measure the proximity to the cluster centers. The ocean proximity attribute may be useful as well, although in Northern California the housing prices in coastal districts are not too high, so it is not a simple rule.

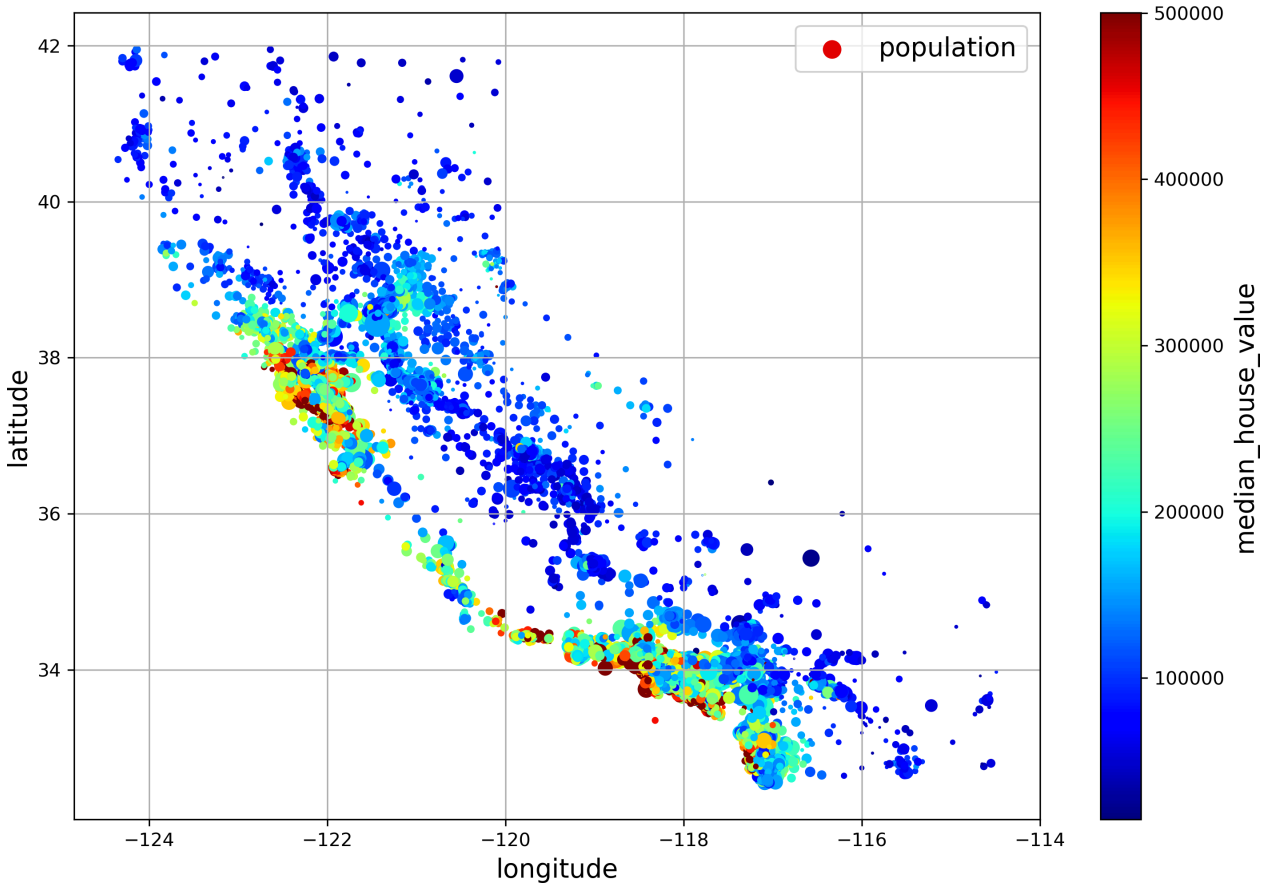

Figure 2-13. California housing prices: red is expensive, blue is cheap, larger circles indicate areas with a larger population

Look for Correlations

Since the dataset is not too large, you can easily compute the *standard correlation coefficient* (also called *Pearson's r*) between every pair of numerical attributes using the `corr()` method:

```
corr_matrix = housing.corr(numeric_only=True)
```

Now you can look at how much each attribute correlates with the median house value:

```
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value    1.000000
median_income         0.688380
total_rooms           0.137455
housing_median_age    0.102175
households            0.071426
total_bedrooms        0.054635
population           -0.020153
longitude            -0.050859
latitude             -0.139584
Name: median_house_value, dtype: float64
```

The correlation coefficient ranges from –1 to 1. When it is close to 1, it means that there is a strong positive correlation; for example, the median house value tends to go up when the median income goes up. When the coefficient is close to –1, it means that there is a strong negative correlation; you can see a small negative correlation between the latitude and the median house value (i.e., prices have a slight tendency to go down when you go north). Finally, coefficients close to 0 mean that there is no linear correlation.

Another way to check for correlation between attributes is to use the Pandas `scatter_matrix()` function, which plots every numerical attribute against every other numerical attribute. Since there are now 9 numerical attributes, you would get $9^2 = 81$ plots, which would not fit on a page—so you decide to focus on a few

promising attributes that seem most correlated with the median housing value (Figure 2-14):

```
from pandas.plotting import scatter_matrix

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

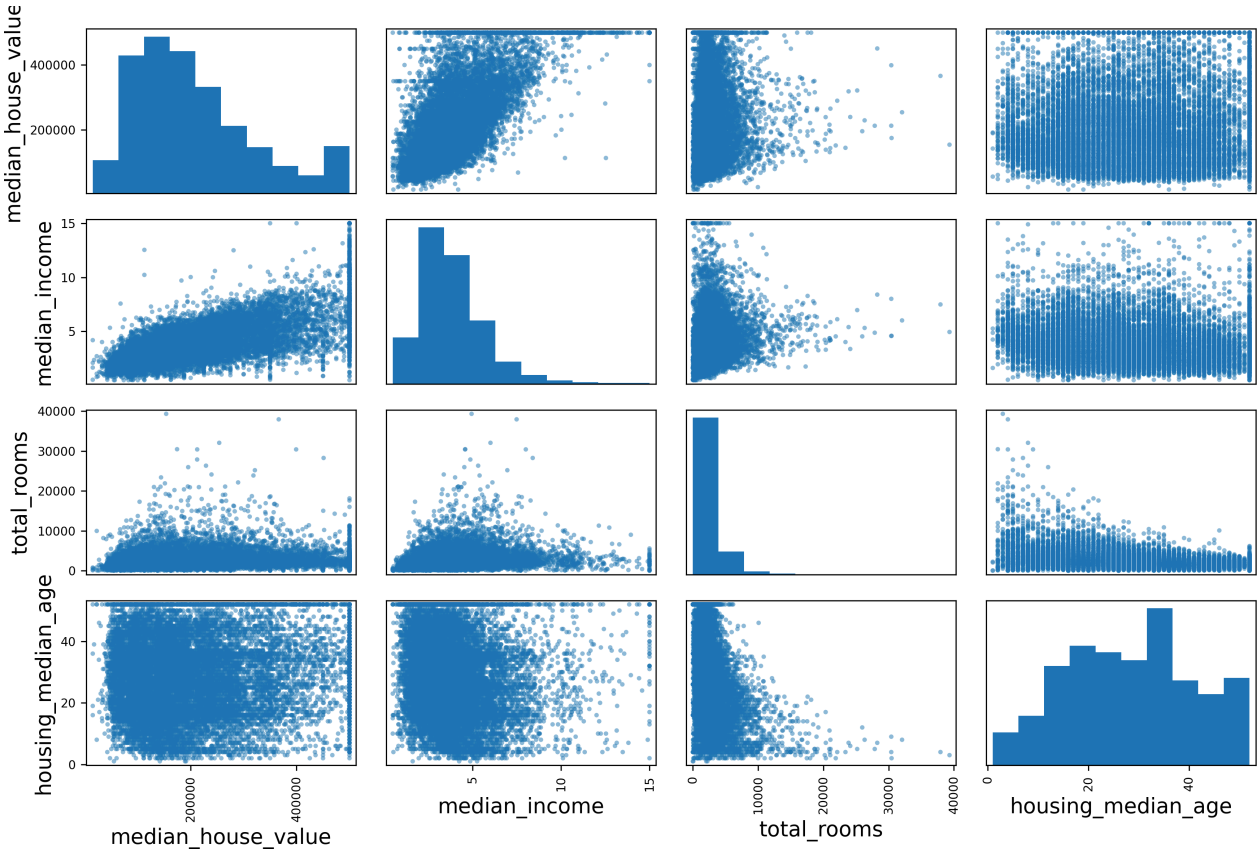

Figure 2-14. This scatter matrix plots every numerical attribute against every other numerical attribute, plus a histogram of each numerical attribute's values on the main diagonal (top left to bottom right)

The main diagonal would be full of straight lines if Pandas plotted each variable against itself, which would not be very useful. So instead, the Pandas displays a histogram of each attribute (other options are available; see the Pandas documentation for more details).

Looking at the correlation scatterplots, it seems like the most promising attribute to predict the median house value is the median income, so you zoom in on that

scatterplot (Figure 2-15):

```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1, grid=True)
plt.show()
```

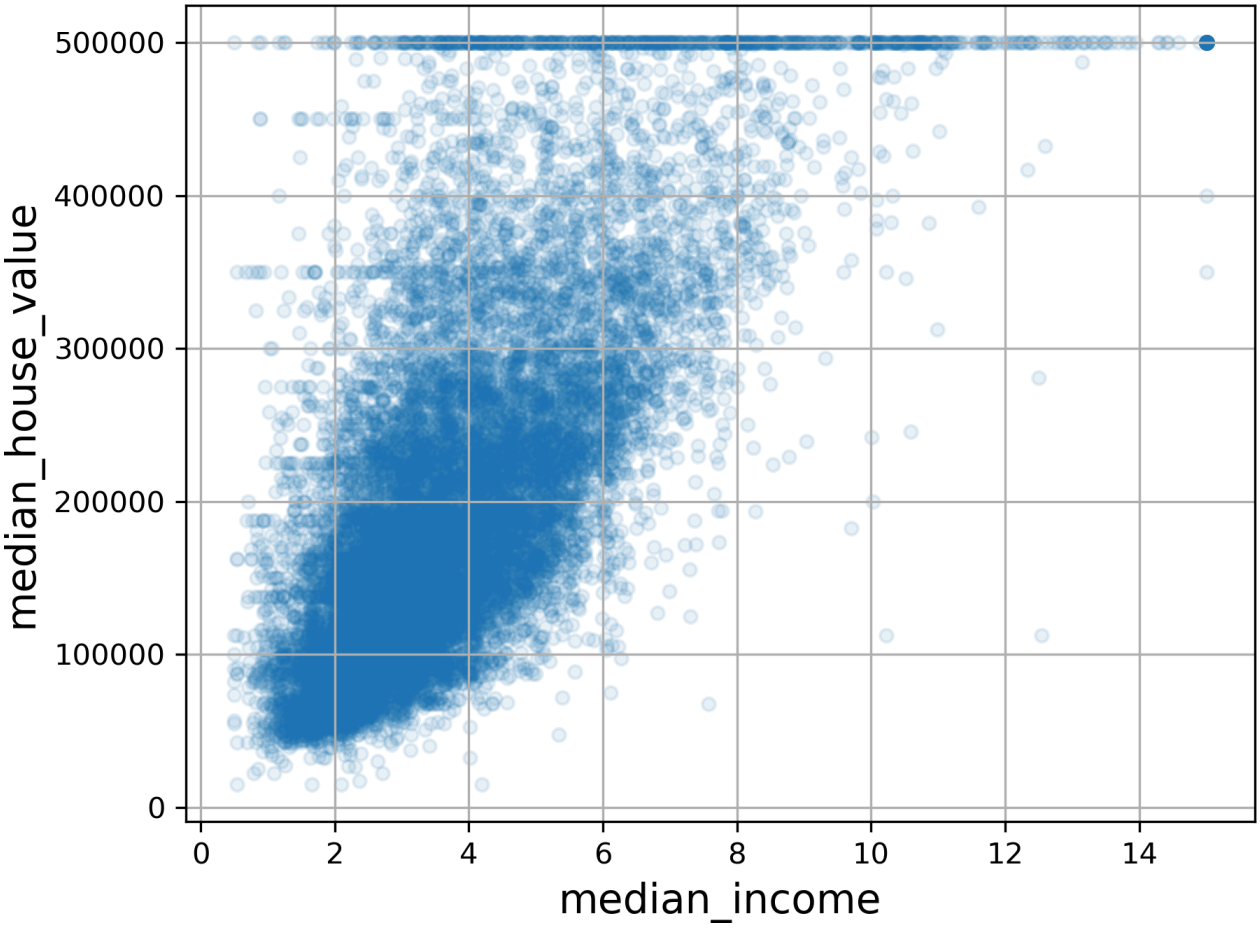

Figure 2-15. Median income versus median house value

This plot reveals a few things. First, the correlation is indeed quite strong; you can clearly see the upward trend, although the data is noisy. Second, the price cap you noticed earlier is clearly visible as a horizontal line at $500,000. But the plot also reveals other less obvious straight lines: a horizontal line around $450,000, another around $350,000, perhaps one around $280,000, and a few more below that. You may want to try removing the corresponding districts to prevent your algorithms from learning to reproduce these data quirks.

WARNING

The correlation coefficient only measures linear correlations ("as *x* goes up, *y* generally goes up/down"). It may completely miss out on nonlinear relationships (e.g., "as *x* approaches 0, *y* generally goes up"). Figure 2-16 shows a variety of datasets along with their correlation coefficient. Note how all the plots of the bottom row have a correlation coefficient equal to 0, despite the fact that their axes are clearly *not* independent: these are examples of nonlinear relationships. Also, the second row shows examples where the correlation coefficient is equal to 1 or –1; notice that this has nothing to do with the slope. For example, your height in inches has a correlation coefficient of 1 with your height in feet or in nanometers.

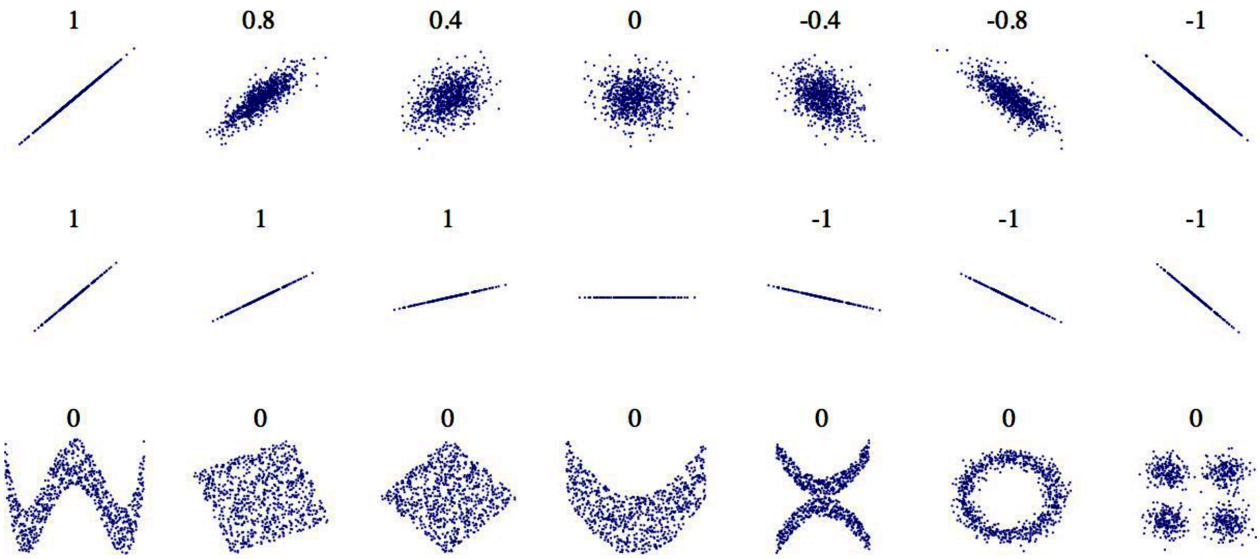

Figure 2-16. Standard correlation coefficient of various datasets (source: Wikipedia; public domain image)

Experiment with Attribute Combinations

Hopefully the previous sections gave you an idea of a few ways you can explore the data and gain insights. You identified a few data quirks that you may want to clean up before feeding the data to a machine learning algorithm, and you found interesting correlations between attributes, in particular with the target attribute. You also noticed that some attributes have a skewed-right distribution, so you may want to transform them (e.g., by computing their logarithm or square root). Of course, your mileage will vary considerably with each project, but the general ideas are similar.

One last thing you may want to do before preparing the data for machine learning algorithms is to try out various attribute combinations. For example, the total number of rooms in a district is not very useful if you don't know how many households there are. What you really want is the number of rooms per household. Similarly, the total number of bedrooms by itself is not very useful: you probably want to compare it to the total number of rooms. And the population per household also seems like an interesting attribute combination to look at. You create these new attributes as follows:

```
housing["rooms_per_house"] = housing["total_rooms"] / housing["households"]
housing["bedrooms_ratio"] = housing["total_bedrooms"] / housing["total_rooms"]
housing["people_per_house"] = housing["population"] / housing["households"]
```

And then you look at the correlation matrix again:

```
>>> corr_matrix = housing.corr(numeric_only=True)
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value    1.000000
median_income         0.688380
rooms_per_house       0.143663
total_rooms           0.137455
housing_median_age    0.102175
households            0.071426
total_bedrooms        0.054635
population           -0.020153
people_per_house     -0.038224
longitude            -0.050859
latitude             -0.139584
bedrooms_ratio       -0.256397
Name: median_house_value, dtype: float64
```

Hey, not bad! The new `bedrooms_ratio` attribute is much more correlated with the median house value than the total number of rooms or bedrooms. It's a strong negative correlation, so it looks like houses with a lower bedroom/room ratio tend to be more expensive. The number of rooms per household is also more informa-

tive than the total number of rooms in a district—obviously the larger the houses, the more expensive they are.

WARNING

When creating new combined features, make sure they are not too linearly correlated with existing features: *collinearity* can cause issues with some models, such as linear regression. In particular, avoid simple weighted sums of existing features.

This round of exploration does not have to be absolutely thorough; the point is to start off on the right foot and quickly gain insights that will help you get a first reasonably good prototype. But this is an iterative process: once you get a prototype up and running, you can analyze its output to gain more insights and come back to this exploration step.

Prepare the Data for Machine Learning Algorithms

It's time to prepare the data for your machine learning algorithms. Instead of doing this manually, you should write functions for this purpose, for several good reasons:

- This will allow you to reproduce these transformations easily on any dataset (e.g., the next time you get a fresh dataset).
- You will gradually build a library of transformation functions that you can reuse in future projects.
- You can use these functions in your live system to transform the new data before feeding it to your algorithms.
- This will make it possible for you to easily try various transformations and see which combination of transformations works best.

But first, revert to a clean training set (by copying `strat_train_set` once again). You should also separate the predictors and the labels, since you don't necessarily