

Download from finelybook www.finelybook.com
seems like an interesting attribute combination to look at. Let's create these new attributes:

```
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"]
```

And now let's look at the correlation matrix again:

```
>>> corr_matrix = housing.corr()
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value      1.000000
median_income           0.687170
rooms_per_household     0.199343
total_rooms             0.135231
housing_median_age      0.114220
households              0.064702
total_bedrooms          0.047865
population_per_household -0.021984
population              -0.026699
longitude               -0.047279
latitude                -0.142826
bedrooms_per_room       -0.260070
Name: median_house_value, dtype: float64
```

Hey, not bad! The new `bedrooms_per_room` attribute is much more correlated with the median house value than the total number of rooms or bedrooms. Apparently houses with a lower bedroom/room ratio tend to be more expensive. The number of rooms per household is also more informative than the total number of rooms in a district—obviously the larger the houses, the more expensive they are.

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 just doing this manually, you should write functions to do that, 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.

Download from [finelybook www.finelybook.com](http://finelybook.com)

- This will make it possible for you to easily try various transformations and see which combination of transformations works best.

But first let's revert to a clean training set (by copying `strat_train_set` once again), and let's separate the predictors and the labels since we don't necessarily want to apply the same transformations to the predictors and the target values (note that `drop()` creates a copy of the data and does not affect `strat_train_set`):

```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

Data Cleaning

Most Machine Learning algorithms cannot work with missing features, so let's create a few functions to take care of them. You noticed earlier that the `total_bedrooms` attribute has some missing values, so let's fix this. You have three options:

- Get rid of the corresponding districts.
- Get rid of the whole attribute.
- Set the values to some value (zero, the mean, the median, etc.).

You can accomplish these easily using `DataFrame`'s `dropna()`, `drop()`, and `fillna()` methods:

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

If you choose option 3, you should compute the median value on the training set, and use it to fill the missing values in the training set, but also don't forget to save the median value that you have computed. You will need it later to replace missing values in the test set when you want to evaluate your system, and also once the system goes live to replace missing values in new data.

Scikit-Learn provides a handy class to take care of missing values: `Imputer`. Here is how to use it. First, you need to create an `Imputer` instance, specifying that you want to replace each attribute's missing values with the median of that attribute:

```
from sklearn.preprocessing import Imputer

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

Since the median can only be computed on numerical attributes, we need to create a copy of the data without the text attribute `ocean_proximity`:

```
housing_num = housing.drop("ocean_proximity", axis=1)
```