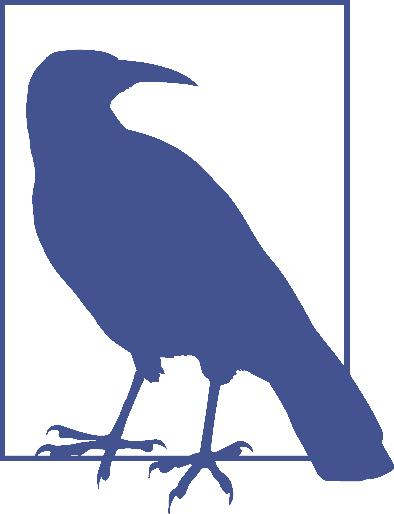
**CHAPTER 2**

**End-to-End Machine Learning Project**

**Dự án ML từ đầu đến cuối**



With Early Release ebooks, you get books in their earliest form— the author’s raw and unedited content as he or she writes—so you can take advantage of these technologies long before the official release of these titles. The following will be Chapter 2 in the final release of the book.



In this chapter, you will go through an example project end to end, pretending to be a recently hired data scientist in a real estate company.[1](#page1) Here are the main steps you will go through:

Trong chapter này, bạn sẽ trải qua một ví dự về dự án từ đầu đến cuối, giả định bạn là một nhà khoa học dữ liệu được thuê gần đây trong một công ty bất động sản. Đây là những bước chính bạn sẽ đi qua:

1. Look at the big picture./Quan sát một bức tranh lớn
2. Get the data./Lấy dữ liệu
3. Discover and visualize the data to gain insights./Khám phá và trực quan hóa dữ liệu để hiểu rõ hơn.
4. Prepare the data for Machine Learning algorithms./Chuẩn bị dữ liệu cho các thuật toán ML.
5. Select a model and train it./ Chọn một mô hình và huấn luyện nó.
6. Fine-tune your model./Tinh chỉnh mô hình
7. Present your solution./Thực hiện giải pháp của bạn
8. Launch, monitor, and maintain your system./ Khởi chạy, giám sát, bảo trì hệ thống của bạn.



1. The example project is completely fictitious; the goal is just to illustrate the main steps of a Machine Learning project, not to learn anything about the real estate business.



**37**

**Working with Real Data/Làm việc với dữ liệu thật**

When you are learning about Machine Learning it is best to actually experiment with real-world data, not just artificial datasets. Fortunately, there are thousands of open datasets to choose from, ranging across all sorts of domains. Here are a few places you can look to get data:

Khi bạn đang tìm hiểu về Machine Learning, tốt nhất là thử nghiệm thực tế với dữ liệu thực, không chỉ các bộ dữ liệu nhân tạo. May mắn thay, có hàng ngàn bộ dữ liệu mở để lựa chọn, trên tất cả các loại tên miền. Dưới đây là một vài nơi bạn có thể tìm để lấy dữ liệu:

* Popular open data repositories:/Kho dữ liệu mở phổ biến

— [UC Irvine Machine Learning Repository](http://archive.ics.uci.edu/ml/)

— [Kaggle datasets](https://www.kaggle.com/datasets)

— [Amazon’s AWS datasets](https://registry.opendata.aws/)

* Meta portals (they list open data repositories):/ Meta portals (chúng liệt kê các kho dữ liệu mở):

— [*http://dataportals.org/*](http://dataportals.org/)

— [*http://opendatamonitor.eu/*](http://opendatamonitor.eu/)

— [*http://quandl.com/*](http://quandl.com/)

* Other pages listing many popular open data repositories:/ Những danh sách trang có dữ liệu mở phổ biến khác:

— [Wikipedia’s list of Machine Learning datasets](https://homl.info/9)

— [Quora.com question](https://homl.info/10)

— [Datasets subreddit](https://www.reddit.com/r/datasets)

In this chapter we chose the California Housing Prices dataset from the StatLib repository[2](#page2) (see [Figure 2-1](#page3)). This dataset was based on data from the 1990 California census. It is not exactly recent (you could still afford a nice house in the Bay Area at the time), but it has many qualities for learning, so we will pretend it is recent data. We also added a categorical attribute and removed a few features for teaching purposes.

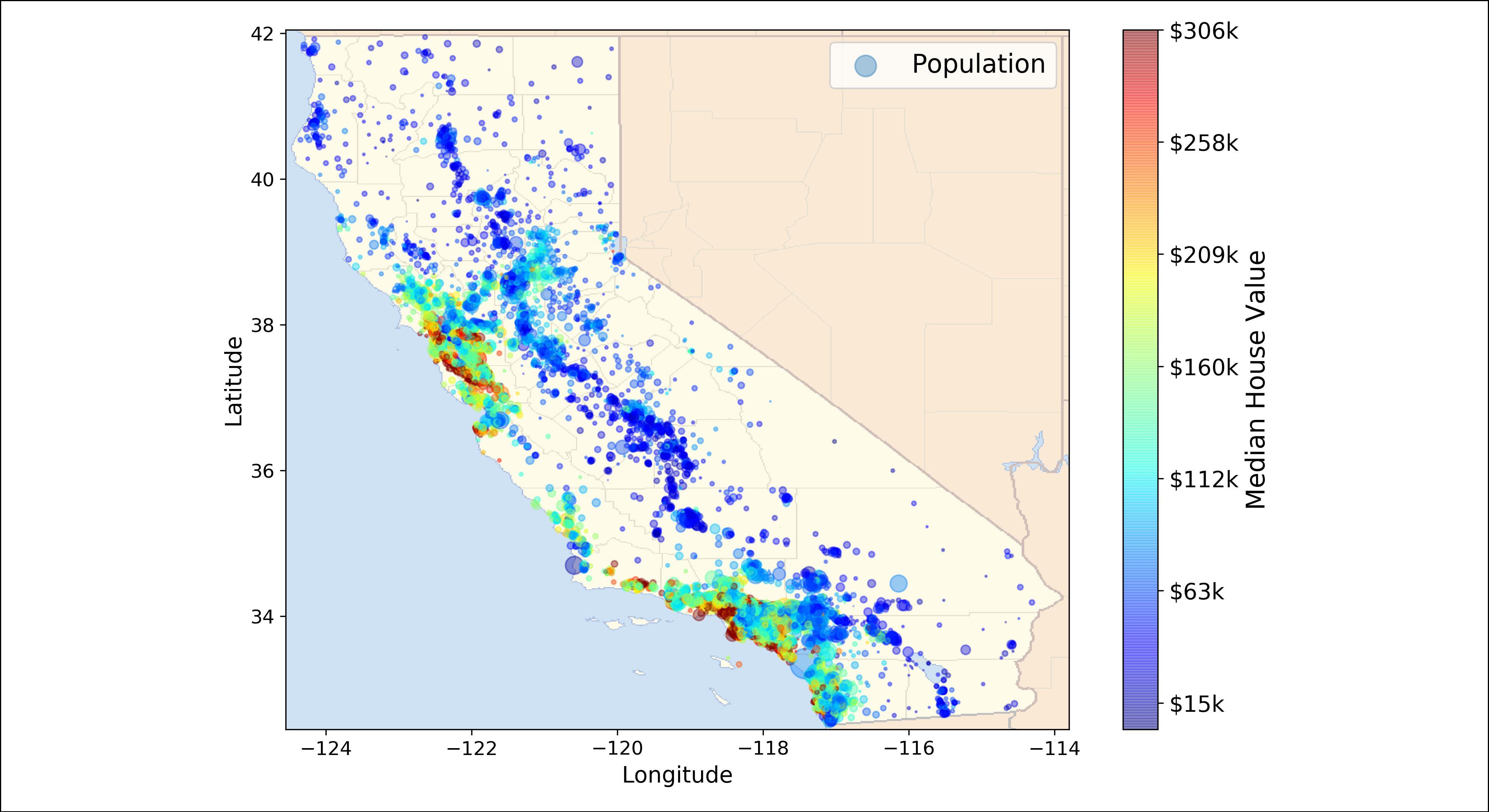
Trong Chaptre này chúng ta đã chọn bộ dữ liệu giá nhà ở California từ StatLib(hình 2-1). Tập dữ liệu này phụ thuộc trên dữ liệu từ dân số California năm 1990. Nó không chính xác gần đây(bạn vẫn có thể mua được một ngồi nhà tốt ở khu vực vịnh trong thời gian này), nhưng nó có nhiều thứ chất lượng để có thể học, vậy chúng ta coi nó là dữ liệu gần đây. Chúng tôi cũng đã thêm một thuộc tính phân loại và loại bỏ một vài tính năng cho mục đích giảng dạy.



1. The original dataset appeared in R. Kelley Pace and Ronald Barry, “Sparse Spatial Autoregressions,” *Statistics* *& Probability Letters* 33, no. 3 (1997): 291–297.



**38** **|** **Chapter 2: End-to-End Machine Learning Project**



*Figure 2-1. California housing prices/ Giá nhà California*

**Look at the Big Picture/Quan sát một bức tranh lớn**

Welcome to Machine Learning Housing Corporation! The first task you are asked to perform is to build a model of housing prices in California using the California census data. This data has metrics such as the population, median income, median housing price, and so on for each block group in California. Block groups are the smallest geographical unit for which the US Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people). We will just call them “districts” for short.

Chào mừng bạn đến với Machine Learning Housing Corporation! Nhiệm vụ đầu tiên bạn được yêu cầu thực hiện là xây dựng mô hình giá nhà ở California bằng dữ liệu điều tra dân số California. Dữ liệu này có các số liệu như dân số, thu nhập trung bình, giá nhà ở trung bình, v.v. cho từng nhóm khối ở California. Các nhóm khối là đơn vị địa lý nhỏ nhất mà Cục điều tra dân số Hoa Kỳ công bố dữ liệu mẫu (một nhóm khối thường có dân số từ 600 đến 3.000 người). Chúng tôi sẽ chỉ gọi họ là ”quận huyện” nhỏ.

Your model should learn from this data and be able to predict the median housing price in any district, given all the other metrics.

Mô hình của bạn nên học từ dữ liệu này và có thể dự đoán giá nhà ở trung bình ở bất kỳ quận nào, dựa trên tất cả các số liệu khác.



Since you are a well-organized data scientist, the first thing you do is to pull out your Machine Learning project checklist. You can start with the one in ???; it should work reasonably well for most Machine Learning projects but make sure to adapt it to your needs. In this chapter we will go through many checklist items, but we will also skip a few, either because they are self-explanatory or because they will be discussed in later chapters.

Vì bạn là một nhà khoa học dữ liệu của tổ chức nổi tiếng, điều đầu tiên bạn làm là rút ra danh sách kiểm tra dự án Machine Learning. Bạn có thể bắt đầu với một trong số chúng???; nó sẽ hoạt động tốt cho hầu hết các dự án Machine Learning nhưng đảm bảo điều chỉnh nó phù hợp với nhu cầu của bạn. Trong chapter này, chúng ta sẽ đi qua nhiều mục trong danh sách kiểm tra, nhưng chúng ta cũng sẽ bỏ qua một vài mục, vì chúng ta sẽ tự giải thích hoặc chúng ta sẽ được thảo luận trong các chương sau.

**Frame the Problem/Đóng khung vấn đề**

The first question to ask your boss is what exactly is the business objective; building a model is probably not the end goal. How does the company expect to use and benefit from this model? This is important because it will determine how you frame the problem, what algorithms you will select, what performance measure you will use to evaluate your model, and how much effort you should spend tweaking it.

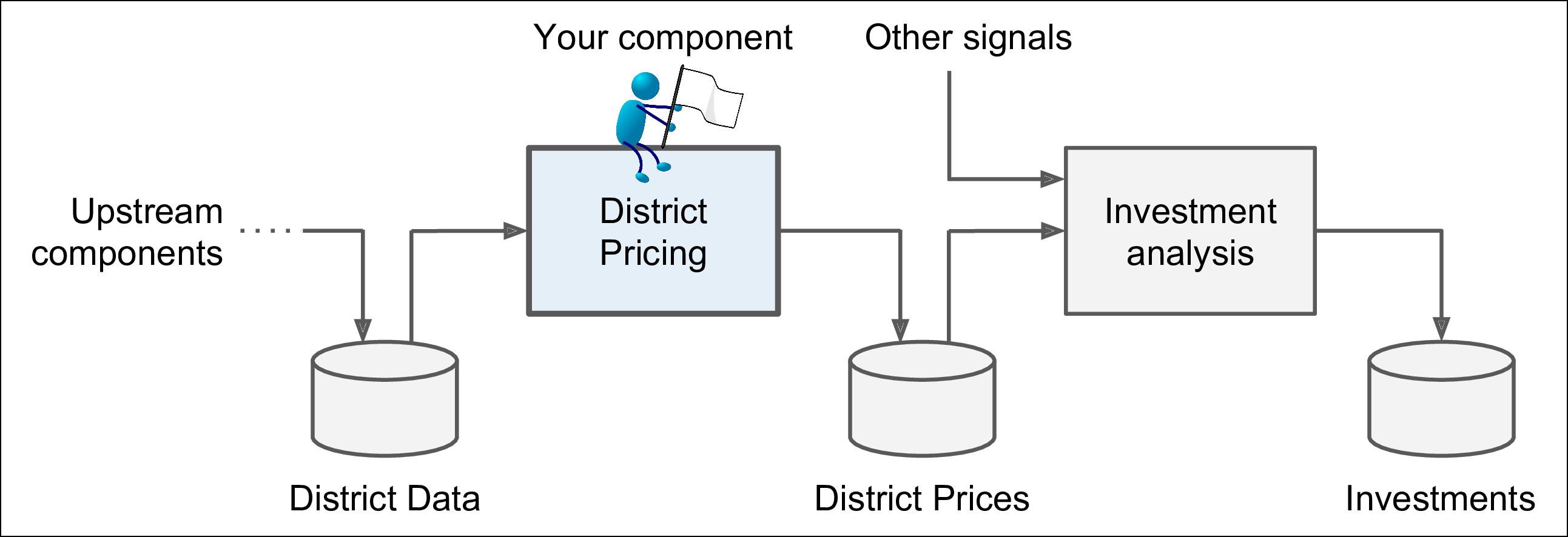
Câu hỏi đầu tiên để hỏi sếp là mục tiêu của doanh nghiệp là gì; xây dựng một mô hình có lẽ không phải mục đích cuối cùng. Làm thế nào để công ty mong được sử dụng và có lợi từ mô hình đó? Đây là điều quan trọng vì nó sẽ xác định cách bạn đóng khung vấn đề, bạn sẽ chọn thuật toán nào, bạn sẽ đánh giá đo lường hiệu suất mô hình như thế nào, và bạn nên dành bao nhiêu nỗ lực để điều chỉnh nó.



**Look at the Big Picture** **|** **39**

Your boss answers that your model’s output (a prediction of a district’s median housing price) will be fed to another Machine Learning system (see [Figure 2-2](#page4)), along with many other *signals*.[3](#page4) This downstream system will determine whether it is worth investing in a given area or not. Getting this right is critical, as it directly affects revenue.

Sếp của bạn trả lời rằng đầu ra của mô hình của bạn (dự đoán về giá nhà trung bình nhà ở của một quận) sẽ được đưa đến một hệ thống Machine Learning khác (xem Hình 2-2), cùng với nhiều tín hiệu khác. Hệ thống sẽ lấy nguồn dữ liệu này để xác định liệu nó có đáng để đầu tư trong một khu vực nhất định hay không. Điều này là rất quan trọng, vì nó ảnh hưởng trực tiếp đến doanh thu.



*Figure 2-2. A Machine Learning pipeline for real estate investments/Một quy trình ML cho đầu tư bất động sản*



**Pipelines/Quy trình**

A sequence of data processing *components* is called a data *pipeline*. Pipelines are very common in Machine Learning systems, since there is a lot of data to manipulate and many data transformations to apply.

Một chuỗi các thành phần xử lý dữ liệu được gọi là quy trình dữ liệu. Quy trình rất phổ biến trong các hệ thống Machine Learning, vì có rất nhiều dữ liệu để thao tác và nhiều dữ liệu biến đổi để áp dụng.

Components typically run asynchronously. Each component pulls in a large amount of data, processes it, and spits out the result in another data store, and then some time later the next component in the pipeline pulls this data and spits out its own output, and so on. Each component is fairly self-contained: the interface between components is simply the data store. This makes the system quite simple to grasp (with the help of a data flow graph), and different teams can focus on different components. Moreover, if a component breaks down, the downstream components can often continue to run normally (at least for a while) by just using the last output from the broken component. This makes the architecture quite robust.

Các thành phần thường chạy không đồng bộ. Mỗi thành phần lấy một lượng lớn dữ liệu, xử lý dữ liệu và tạo ra kết quả trong kho lưu trữ dữ liệu khác và sau đó một thời gian sau đó, thành phần tiếp theo trong quy trình sẽ kéo dữ liệu này và tạo ra đầu ra của chính nó, v.v. Mỗi thành phần sẽ được giữ lại: bố cục giữa các thành phần chỉ đơn giản là lưu trữ dữ liệu. Điều này làm cho hệ thống khá đơn giản để nắm bắt (với sự trợ giúp của biểu đồ luồng dữ liệu) và các nhóm khác nhau có thể tập trung vào các thành phần khác nhau. Ngoài ra, nếu một thành phần bị hỏng, các thành phần lấy dữ liệu thường có thể tiếp tục chạy bình thường (ít nhất là trong một thời gian) chỉ bằng cách sử dụng đầu ra cuối cùng từ thành phần bị hỏng. Điều này làm cho kiến ​​trúc khá mạnh mẽ.



1. A piece of information fed to a Machine Learning system is often called a *signal* in reference to Shannon’s information theory: you want a high signal/noise ratio.



**40** **|** **Chapter 2: End-to-End Machine Learning Project**

On the other hand, a broken component can go unnoticed for some time if proper monitoring is not implemented. The data gets stale and the overall system’s performance drops.

Mặt khác, một thành phần bị hỏng có thể không được chú ý trong một thời gian nếu việc giám sát thích hợp không được thực hiện. Dữ liệu trở nên cũ kỹ và hiệu suất toàn bộ hệ thống giảm dần.



The next question to ask is what the current solution looks like (if any). It will often give you a reference performance, as well as insights on how to solve the problem. Your boss answers that the district housing prices are currently estimated manually by experts: a team gathers up-to-date information about a district, and when they cannot get the median housing price, they estimate it using complex rules.

Câu hỏi tiếp theo là giải pháp hiện tại trông như thế nào (nếu có). Nó thường sẽ cung cấp cho bạn một hiệu suất tham khảo, cũng như hiểu biết sâu sắc về cách giải quyết vấn đề. Sếp của bạn trả lời rằng giá nhà ở quận hiện được các chuyên gia ước tính bằng tay: một nhóm thu thập thông tin cập nhật về một quận và khi họ không thể có được giá nhà ở trung bình, họ ước tính bằng cách sử dụng các quy tắc phức tạp.

This is costly and time-consuming, and their estimates are not great; in cases where they manage to find out the actual median housing price, they often realize that their estimates were off by more than 20%. This is why the company thinks that it would be useful to train a model to predict a district’s median housing price given other data about that district. The census data looks like a great dataset to exploit for this pur‐ pose, since it includes the median housing prices of thousands of districts, as well as other data.

Điều này là tốn kém và tốn thời gian, và ước tính của họ không quá đúng; trong trường hợp họ quản lý để tìm ra giá nhà trung bình thực tế, họ thường nhận ra rằng ước tính của họ đã giảm hơn 20%. Đây là lý do tại sao công ty nghĩ rằng sẽ rất hữu ích khi huấn luyện một mô hình để dự đoán giá nhà ở trung bình của quận với các dữ liệu khác về quận đó. Dữ liệu điều tra dân số trông giống như một bộ dữ liệu tuyệt vời để khai thác cho mục đích này, vì nó bao gồm giá nhà ở trung bình của hàng ngàn quận, hơn các dữ liệu khác.

Okay, with all this information you are now ready to start designing your system. First, you need to frame the problem: is it supervised, unsupervised, or Reinforce‐ ment Learning? Is it a classification task, a regression task, or something else? Should you use batch learning or online learning techniques? Before you read on, pause and try to answer these questions for yourself.

Okay, với tất cả thông tin này bạn bây giờ đã sẵn sàng để bắt đầu thiết kế hệ thống của mình. Đầu tiên, bạn cần khung các vấn đề: là nó học giám sát, không giám sát, hoặc học củng cố? Là nhiệm vụ phân loại, hồi quy, hoặc thứ gì đó? Bạn nên sử dụng kỹ thuật batch learning hay online learning? Trước khi bạn đọc tiếp, hãy tạm dừng và cố gắng trả lời các câu hỏi này cho chính mình.

Have you found the answers? Let’s see: it is clearly a typical supervised learning task since you are given *labeled* training examples (each instance comes with the expected output, i.e., the district’s median housing price). Moreover, it is also a typical regression task, since you are asked to predict a value. More specifically, this is a *multiple* *regression* problem since the system will use multiple features to make a prediction (itwill use the district’s population, the median income, etc.). It is also a *univariate* *regression* problem since we are only trying to predict a single value for each district.If we were trying to predict multiple values per district, it would be a *multivariate* *regression* problem. Finally, there is no continuous flow of data coming in the system,there is no particular need to adjust to changing data rapidly, and the data is small enough to fit in memory, so plain batch learning should do just fine.

Bạn đã tìm được các câu trả lời? Cùng xem nào, đây rõ rang là một task học giám sát điển hình vì bạn có các ví dụ gán nhãn (mỗi cá thể đi kèm với kết quả đầu ra, tức là, giá nhà ở trung bình của quận). Hơn nữa, nó cũng là một nhiệm vụ hồi quy điển hình, vì bạn được yêu cầu dự đoán một giá trị. Cụ thể hơn, đây là một vấn đề đa hồi quy vì hệ thống sẽ sử dụng nhiều đặc trưng để đưa ra dự đoán (nó sẽ sử dụng dân số quận, thu nhập trung bình, v.v.). Đây cũng là một vấn đề hồi quy đơn biến vì chúng tôi chỉ cố gắng dự đoán một giá trị duy nhất cho mỗi quận. Nếu chúng ta cố gắng dự đoán nhiều giá trị cho mỗi huyện, đó sẽ là một vấn đề hồi quy đa biến. Cuối cùng, không có luồng dữ liệu liên tục đến trong hệ thống, sẽ không cần điều chỉnh cụ thể để thay đổi dữ liệu nhanh chóng và dữ liệu đủ nhỏ để phù hợp với bộ nhớ, vì vậy batch learning đơn giản nên thực hiện tốt.



If the data was huge, you could either split your batch learning work across multiple servers (using the *MapReduce* technique), or you could use an online learning technique instead.

Nếu dữ liệu quá lớn, bạn có thẻ chia batch learning trên nhiều máy chủ( sử dụng MapReduce technique), hoặc bạn có thẻ sử dụng kỹ thuật online learning để thay thế.



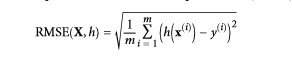
**Look at the Big Picture** **|** **41**

**Select a Performance Measure/ Chọn thước đo hiệu suất**

Your next step is to select a performance measure. A typical performance measure for regression problems is the Root Mean Square Error (RMSE). It gives an idea of how much error the system typically makes in its predictions, with a higher weight for large errors. [Equation 2-1](#page6) shows the mathematical formula to compute the RMSE.

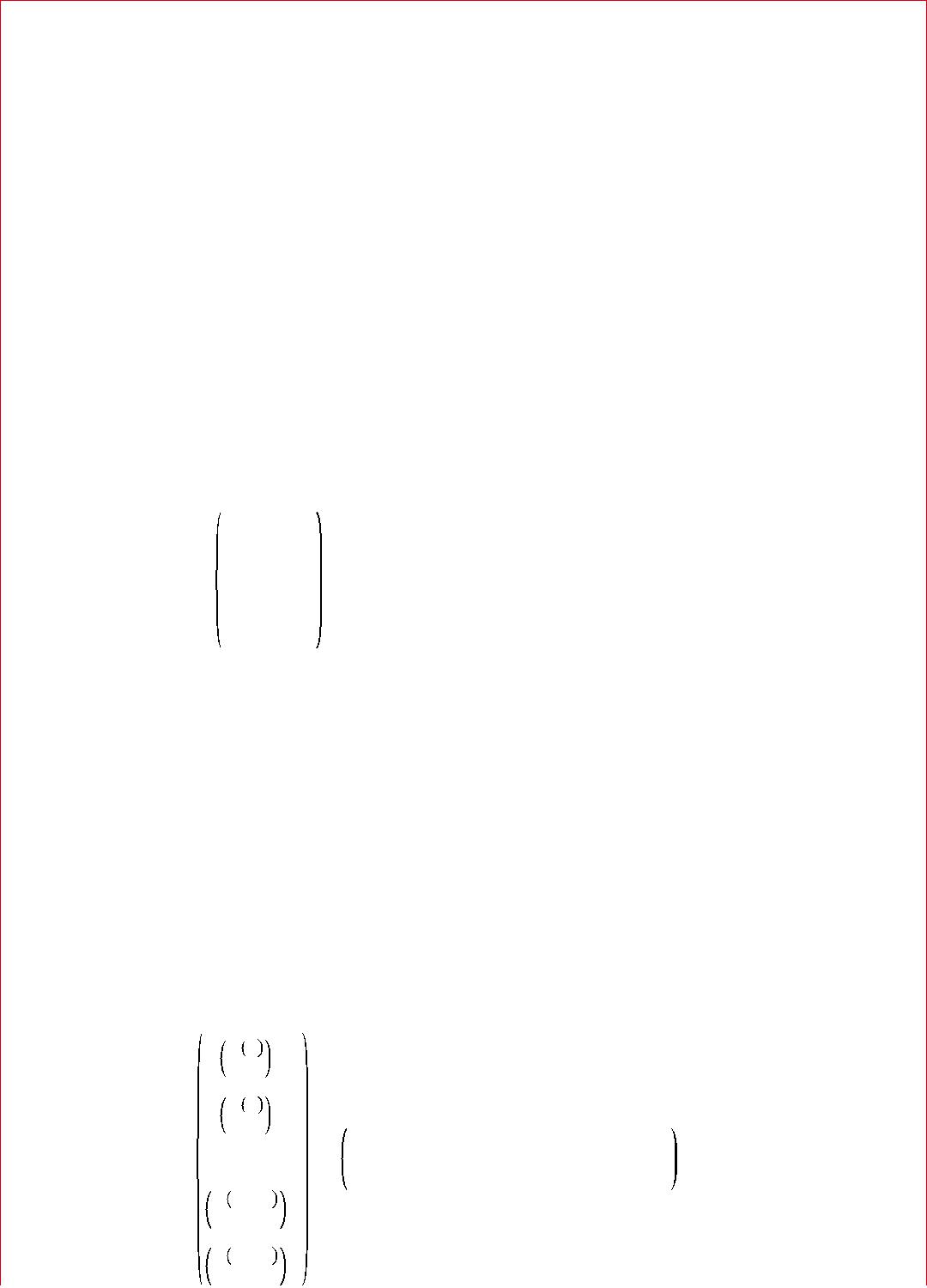
Bước tiếp theo của bạn là chọn thước đo hiệu suất. Một thước đo hiệu suất điển hình cho các vấn đề hồi quy là the Root Mean Square Error (RMSE). Nó đưa ra ý tưởng về mức độ lỗi mà hệ thống thường mắc phải trong các dự đoán của nó, với trọng số cao hơn cho các lỗi lớn. Công thức 2-1 cho thấy công thức toán học để tính toán RMSE.

*Equation 2-1. Root Mean Square Error (RMSE)*





**42** **|** **Chapter 2: End-to-End Machine Learning Project**

**Notations/Ký hiệu**

This equation introduces several very common Machine Learning notations that we will use throughout this book:

Phương trình này giới thiệu một số ký hiệu Machine Learning rất phổ biến mà chúng ta sẽ sử dụng trong suốt cuốn sách này:

* *m* is the number of instances in the dataset you are measuring the RMSE on.

*m* là số lượng cá thể trong bộ dữ liệu bạn đang đo bởi RMSE.

— For example, if you are evaluating the RMSE on a validation set of 2,000 districts, then *m* = 2,000.

Ví dụ, nếu bạn đánh giá RMSE trên tập xác nhận của 2,000 quận, *m* = 2,000

* **x***(i)*is a vector of all the feature values (excluding the label) of the*ith*instance inthe dataset, and *y(i)* is its label (the desired output value for that instance).

**x***(i)*là một vectơ của tất cả các giá trị đặc trưng (không bao gồm nhãn) của cá thể thứ i trong tập dữ liệu và *y(i)* là nhãn của nó (giá trị đầu ra mong muốn cho cá thể đó).

— For example, if the first district in the dataset is located at longitude –118.29°, latitude 33.91°, and it has 1,416 inhabitants with a median income of $38,372, and the median house value is $156,400 (ignoring the other features for now), then:

Ví dụ: nếu khu vực đầu tiên trong bộ dữ liệu được đặt tại kinh độ.118,29 °, vĩ độ 33,91 ° và hiện có 1.416 cư dân có thu nhập trung bình là 38.372 đô la và giá trị nhà trung bình là 156.400 đô la (bỏ qua các đặc trưng khác hiện tại), sau đó:



Và :

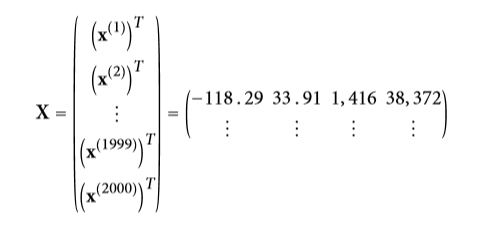
*y*1 = 156, 400

* **X** is a matrix containing all the feature values (excluding labels) of all instances inthe dataset. There is one row per instance and the *ith* row is equal to the transpose of **x***(i)*, noted (**x***(i)*)*T*.[4](#page7)

**X** là một ma trận chứa tất cả các giá trị đặc trưng (không bao gồm nhãn) của tất cả các cá thể trong bộ dữ liệu. Có một hàng cho mỗi đặc trưng và hàng thứ i bằng với chuyển vị của **x***(i)*, được ký hiệu(**x***(i)*)*T*.

— For example, if the first district is as just described, then the matrix **X** looks like this:

Ví dụ: nếu quận đầu tiên đúng như mô tả, thì ma trận **X** trông như thế này:





4 Recall that the transpose operator flips a column vector into a row vector (and vice versa).

Hãy nhớ rằng toán tử chuyển vị lật một vectơ cột thành một vectơ hàng (và ngược lại).



**Look at the Big Picture** **|** **43**

1

*k k* . ℓ0 just gives the number of non-zero ele‐

* *h* is your system’s prediction function, also called a *hypothesis*. When your systemis given an instance’s feature vector **x***(i)*, it outputs a predicted value *ŷ(i)* = *h*(**x***(i)*) for that instance (*ŷ* is pronounced “y-hat”).

*h* là hàm dự đoán của hệ thống của bạn, còn được gọi là *hypothesis*. Khi hệ thống của bạn đưa ra vecto đặc trưng **x***(i)*, nó sẽ đưa ra một giá trị dự đoán *ŷ(i)* = *h*(**x***(i)*) cho cá thể đó (*ŷ* được gọi là “y-hat”).

— For example, if your system predicts that the median housing price in the first district is $158,400, then *ŷ*(1) = *h*(**x**(1)) = 158,400. The prediction error for this district is *ŷ*(1) – *y*(1) = 2,000.

* RMSE(**X**,*h*) is the cost function measured on the set of examples using your hypothesis *h*.



We use lowercase italic font for scalar values (such as *m* or *y(i)*) and function names

(such as *h*), lowercase bold font for vectors (such as **x***(i)*), and uppercase bold font for matrices (such as **X**).

Even though the RMSE is generally the preferred performance measure for regression tasks, in some contexts you may prefer to use another function. For example, suppose that there are many outlier districts. In that case, you may consider using the *Mean* *Absolute Error* (also called the Average Absolute Deviation; see[Equation 2-2](#page8)):

*Equation 2-2. Mean Absolute Error*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | *m* | | |  |
| MAE **X**, *h* = | *i* ∑=1 |  | *h* **x** *i* − *y i* |  |
| *m* |  |
|  |  |  |

Both the RMSE and the MAE are ways to measure the distance between two vectors: the vector of predictions and the vector of target values. Various distance measures, or *norms*, are possible:

• Computing the root of a sum of squares (RMSE) corresponds to the *Euclidean* *norm*: it is the notion of distance you are familiar with. It is also called the ℓ2 *norm*, noted∥·∥2(or just∥·∥).

• Computing the sum of absolutes (MAE) corresponds to the ℓ1 *norm*, noted ∥ · ∥1. It is sometimes called the *Manhattan norm* because it measures the distance between two points in a city if you can only travel along orthogonal city blocks.

• More generally, the ℓ*k* *norm* of a vector **v** containing *n* elements is defined as

*k* = *v*0 *k* + *v*1 *k* +⋯+ *vn*

ments in the vector, and ℓ∞ gives the maximum absolute value in the vector.

* The higher the norm index, the more it focuses on large values and neglects small ones. This is why the RMSE is more sensitive to outliers than the MAE. But when



**44** **|** **Chapter 2: End-to-End Machine Learning Project**

outliers are exponentially rare (like in a bell-shaped curve), the RMSE performs very well and is generally preferred.

**Check the Assumptions/Kiểm tra các giả định**

Lastly, it is good practice to list and verify the assumptions that were made so far (by you or others); this can catch serious issues early on. For example, the district prices that your system outputs are going to be fed into a downstream Machine Learning system, and we assume that these prices are going to be used as such. But what if the downstream system actually converts the prices into categories (e.g., “cheap,” “medium,” or “expensive”) and then uses those categories instead of the prices themselves? In this case, getting the price perfectly right is not important at all; your system just needs to get the category right. If that’s so, then the problem should have been framed as a classification task, not a regression task. You don’t want to find this out after working on a regression system for months.

Cuối cùng, đây là bài thực hành tốt để liệt kê và xác minh các giả định đã được thực hiện cho đến nay (bởi bạn hoặc người khác); điều này có thể phát hiện những vấn đề nghiêm trọng sớm. Ví dụ: giá quận mà đầu ra hệ thống của bạn sẽ được đưa vào đầu vào hệ thống Machine Learning và chúng tôi giả định rằng các giá này sẽ được sử dụng như vậy. Nhưng điều gì sẽ xảy ra nếu hệ thống việc lấy dữ liệu thực sự chuyển đổi giá thành các loại (ví dụ: giá rẻ, giá rẻ trung bình, giá đắt) và sau đó sử dụng các danh mục đó thay vì giá của chúng? Trong trường hợp này, việc có được mức giá hoàn toàn đúng đắn không hề quan trọng; hệ thống của bạn chỉ cần có được danh mục đúng. Nếu điều đó như vậy, thì vấn đề nên được đóng khung như một nhiệm vụ phân loại, không phải là một nhiệm vụ hồi quy. Bạn không muốn tìm hiểu điều này sau khi làm việc trên một hệ thống hồi quy trong nhiều tháng.

Fortunately, after talking with the team in charge of the downstream system, you are confident that they do indeed need the actual prices, not just categories. Great! You’re all set, the lights are green, and you can start coding now!

May mắn thay, sau khi nói chuyện với đội ngũ phụ trách hệ thống lấy dữ liệu, bạn tự tin rằng họ thực sự cần giá thực tế, không chỉ các danh mục. Tuyệt quá! Bạn đã cài đặt xong, đèn màu xanh và bạn có thể bắt đầu viết code ngay bây giờ!

**Get the Data/Lấy dữ liệu**

It’s time to get your hands dirty. Don’t hesitate to pick up your laptop and walk through the following code examples in a Jupyter notebook. The full Jupyter note‐ book is available at [*https://github.com/ageron/handson-ml2*](https://github.com/ageron/handson-ml2).

Đừng ngần ngại lấy lap của bạn và xem code trong Jupyter notebook. Bản đầy đủ Jupyter notebook có sẵn tại [*https://github.com/ageron/handson-ml2*](https://github.com/ageron/handson-ml2).

**Create the Workspace/Tạo một không gian làm việc**

First you will need to have Python installed. It is probably already installed on your system. If not, you can get it at [*https://www.python.org/*](https://www.python.org/).[5](#page9)

Đầu tiên bạn cần cài đặt Python. Nó có lẽ đã được cài sẵn trong hệ thống của bạn. Nếu không, bạn có thể tìm thấy nó ở [*https://www.python.org/*](https://www.python.org/).[5](#page9)

Next you need to create a workspace directory for your Machine Learning code and datasets. Open a terminal and type the following commands (after the $ prompts):

Nếu bạn cần tạo một môi trường cho code ML và tập dữ liệu. Mở terminal và gõ các câu lệnh sau (sau dấu $)

|  |  |  |
| --- | --- | --- |
| $ | export ML\_PATH="$HOME/ml" | # You can change the path if you prefer |
| $ | mkdir -p $ML\_PATH |  |

You will need a number of Python modules: Jupyter, NumPy, Pandas, Matplotlib, and Scikit-Learn. If you already have Jupyter running with all these modules installed, you can safely skip to [“Download the Data” on page 49](#page13). If you don’t have them yet, there are many ways to install them (and their dependencies). You can use your system’s packaging system (e.g., apt-get on Ubuntu, or MacPorts or HomeBrew on MacOS), install a Scientific Python distribution such as Anaconda and use its packaging system, or just use Python’s own packaging system, pip, which is included by default with the Python binary installers (since Python 2.7.9).[6](#page10) You can check to see if pip is installed by typing the following command:

Bạn sẽ cần một số mô-đun Python: Jupyter, NumPy, Pandas, Matplotlib và Scikit-Learn. Nếu bạn đã cài đặt Jupyter với tất cả các mô-đun này, bạn có thể bỏ qua một cách an toàn để Tải xuống Dữ liệu trên trang 49. Nếu bạn không có chúng, có nhiều cách để cài đặt chúng (và phụ thuộc vào chúng). Bạn có thể sử dụng hệ thống package của hệ thống của mình (ví dụ: apt-get trên Ubuntu hoặc MacPorts hoặc HomeBrew trên MacOS), cài đặt bản phân phối Scientific Python như Anaconda và sử dụng các package hệ thống hoặc chỉ sử dụng các package riêng của Python, pip, đó là được bao gồm theo mặc định với trình cài đặt nhị phân Python (kể từ Python 2.7.9). Bạn có thể kiểm tra xem liệu pip có được cài đặt hay không bằng cách gõ lệnh sau:

$ python3 -m pip --version

pip 19.0.2 from [...]/lib/python3.6/site-packages (python 3.6)

You should make sure you have a recent version of pip installed. To upgrade the pip module, type: [7](#page10)

Bạn nên đảm bảo rằng phiên bản pip hiện tại được cài đặt. Nâng cấp pip, gõ:

$ python3 -m pip install --user -U pip Collecting pip

[...]

Successfully installed pip-19.0.2



1. The latest version of Python 3 is recommended. Python 2.7+ may work too, but it is now deprecated, all major scientific libraries are dropping support for it, so you should migrate to Python 3 as soon as possible.



**Get the Data** **|** **4**



**Creating an Isolated Environment/Tạo một môi trường độc lập**

If you would like to work in an isolated environment (which is strongly recommended so you can work on different projects without having conflicting library versions), install virtualenv[8](#page10) by running the following pip command (again, if you want virtualenv to be installed for all users on your machine, remove --user and run this command with administrator rights):

Nếu bạn muốn làm việc trong một môi trường biệt lập (được khuyến nghị mạnh mẽ để bạn có thể làm việc trên các dự án khác nhau mà không có các phiên bản thư viện xung đột), hãy cài đặt virtualenv bằng cách chạy lệnh pip sau (một lần nữa, nếu bạn muốn cài đặt virtualenv cho tất cả người dùng trên máy của bạn, xóa --user và chạy lệnh này với quyền quản trị viên):

$ python3 -m pip install --user -U virtualenv Collecting virtualenv

[...]

Successfully installed virtualenv

Now you can create an isolated Python environment by typing:

Bây giờ bạn có thể tạo một môi trường Python bị cô lập bằng cách gõ:

$ cd $ML\_PATH

$ virtualenv env

Using base prefix '[...]'

New python executable in [...]/ml/env/bin/python3.6 Also creating executable in [...]/ml/env/bin/python Installing setuptools, pip, wheel...done.



1. We will show the installation steps using pip in a bash shell on a Linux or MacOS system. You may need to adapt these commands to your own system. On Windows, we recommend installing Anaconda instead.
2. If you want to upgrade pip for all users on your machine rather than just your own user, you should remove the --user option and make sure you have administrator rights (e.g., by adding sudo before the whole com‐ mand on Linux or MacOSX).
3. Alternative tools include venv (very similar to virtualenv and included in the standard library), virtualenv‐ wrapper (provides extra functionalities on top of virtualenv), pyenv (allows easy switching between Python versions), and pipenv (a great packaging tool by the same author as the popular requests library, built on top of pip, virtualenv and more).



**46** **|** **Chapter 2: End-to-End Machine Learning Project**

Now every time you want to activate this environment, just open a terminal and type:

Bây giờ mỗi khi bạn muốn kích hoạt môi trường này, chỉ cần mở terminal và gõ:

$ cd $ML\_PATH

$ source env/bin/activate # on Linux or MacOSX $ .\env\Scripts\activate # on Windows

To deactivate this environment, just type deactivate. While the environment is active, any package you install using pip will be installed in this isolated environment, and Python will only have access to these packages (if you also want access to the system’s packages, you should create the environment using virtualenv’s --system-site-packages option).

Để hủy kích hoạt môi trường này, chỉ cần gõ deactivate. Khi môi trường hoạt động, mọi package bạn cài đặt bằng pip sẽ được cài đặt trong môi trường bị cô lập này và Python sẽ chỉ có quyền truy cập vào các gói này (nếu bạn cũng muốn truy cập vào các package của hệ thống, bạn nên tạo môi trường bằng virtualenv --system-site-packages).

Check out virtualenv’s documentation for more information.

Kiểm tra tà virtualenv để biết thêm thông tin.

Now you can install all the required modules and their dependencies using this simple pip command (if you are not using a virtualenv, you will need the --user option or administrator rights):

Bây giờ bạn có thể cài đặt tất cả các mô-đun cần thiết và các phụ thuộc của chúng bằng lệnh pip đơn giản này (nếu bạn không sử dụng virtualenv, bạn sẽ cần tùy chọn --user hoặc quyền quản trị viên):

$ python3 -m pip install -U jupyter matplotlib numpy pandas scipy scikit-learn Collecting jupyter

Downloading jupyter-1.0.0-py2.py3-none-any.whl Collecting matplotlib

[...]

To check your installation, try to import every module like this:

Để kiểm tra cài đặt của bạn, hãy thử nhập mọi mô-đun như thế này:

$ python3 -c "import jupyter, matplotlib, numpy, pandas, scipy, sklearn"

There should be no output and no error. Now you can fire up Jupyter by typing:

Không nên có ngoại lệ và lỗi. Bây giờ bạn có thể kích hoạt Jupyter bằng cách gõ:

$ jupyter notebook

[I 15:24 NotebookApp] Serving notebooks from local directory: [...]/ml [I 15:24 NotebookApp] 0 active kernels

[I 15:24 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/ [I 15:24 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).

A Jupyter server is now running in your terminal, listening to port 8888. You can visit this server by opening your web browser to *http://localhost:8888/* (this usually happens automatically when the server starts). You should see your empty workspace directory (containing only the *env* directory if you followed the preceding virtualenv instructions).

Một máy chủ Jupyter hiện đang chạy trong terminal của bạn, cổng 8888. Bạn có thể truy cập máy chủ này bằng cách mở trình duyệt web của mình tới http://localhost:8888/ (điều này thường tự động xảy ra khi máy chủ khởi động). Bạn sẽ thấy thư mục không gian làm việc trống của bạn (chỉ chứa thư mục env nếu bạn làm theo các hướng dẫn virtualenv trước đó).

Now create a new Python notebook by clicking on the New button and selecting the appropriate Python version[9](#page11) (see [Figure 2-3](#page12)).

Bây giờ hãy tạo Python notebook mới bằng cách nhấp vào nút Mới và chọn phiên bản Python thích hợp (xem Hình 2-3).

This does three things: first, it creates a new notebook file called *Untitled.ipynb* in your workspace; second, it starts a Jupyter Python kernel to run this notebook; and third, it opens this notebook in a new tab. You should start by renaming this notebook to “Housing” (this will automatically rename the file to *Housing.ipynb*) by clicking Untitled and typing the new name.

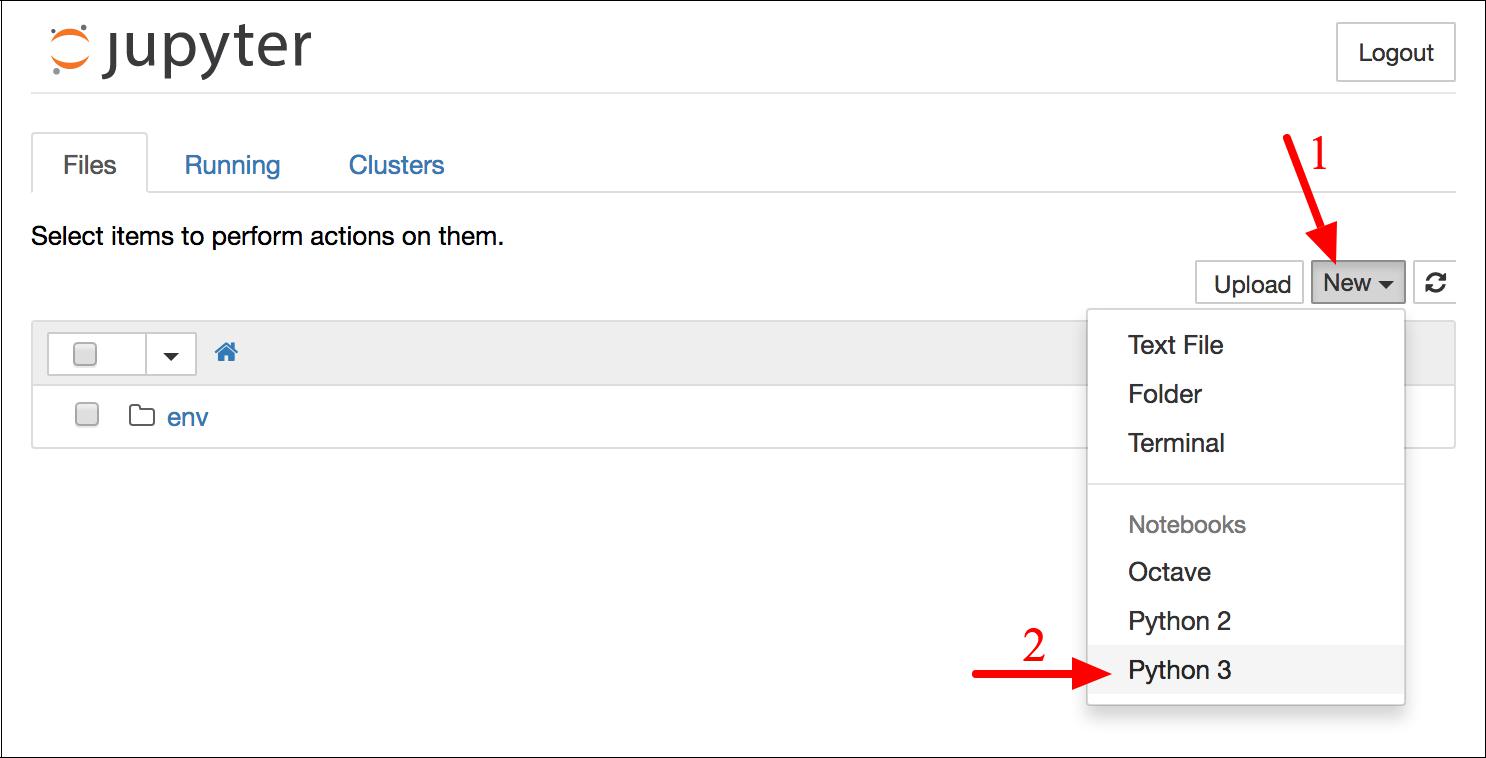
Điều này thực hiện ba điều: đầu tiên, nó tạo ra một notebook mới có tên Untitle.ipynb trong không gian làm việc của bạn; thứ hai, nó khởi động kernel Python Jupyter để chạy notebook này; và thứ ba, nó mở notebook này trong một tab mới. Bạn nên bắt đầu bằng cách đổi tên notebook này thành “Housing” (điều này sẽ tự động đổi tên tệp thành Housing.ipynb) bằng cách nhấp vào tiêu đề Untitled và nhập tên mới.



1. Note that Jupyter can handle multiple versions of Python, and even many other languages such as R or Octave.



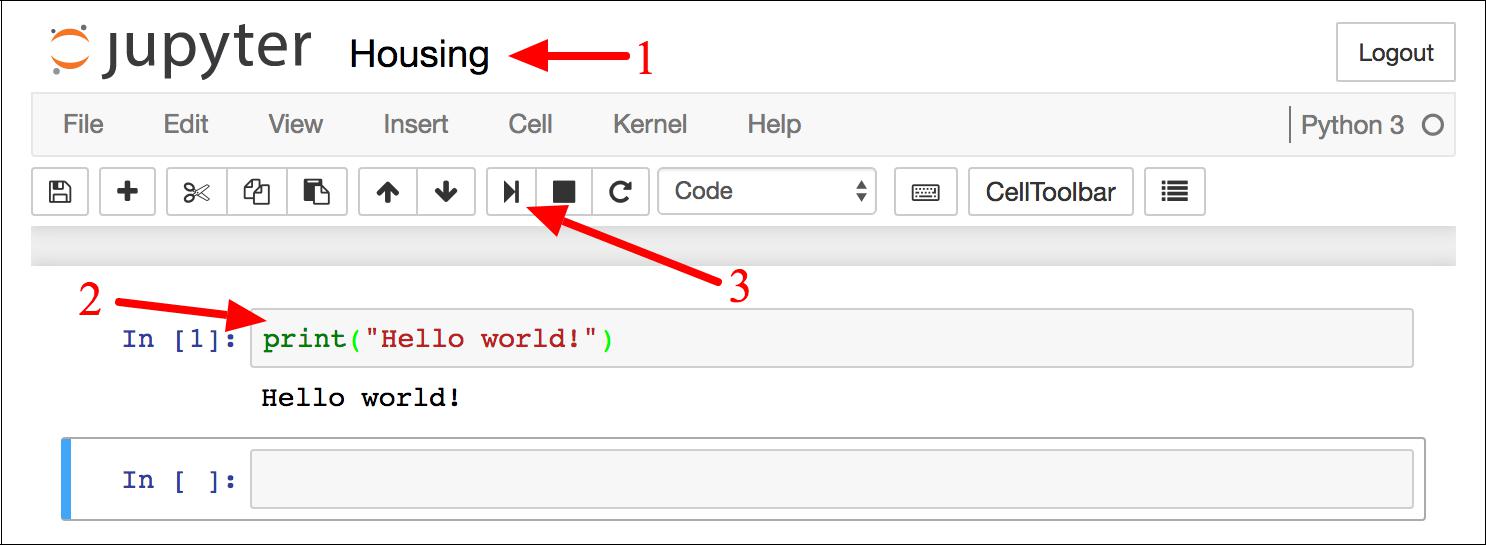
**Get the Data** **|** **47**



*Figure 2-3. Your workspace in Jupyter/Không gian làm việc của bạn trên Jupiter*

A notebook contains a list of cells. Each cell can contain executable code or formatted text. Right now the notebook contains only one empty code cell, labeled “In [1]:”. Try typing **print("Hello world!")** in the cell, and click on the play button (see [Figure 2-4](#page12)) or press Shift-Enter. This sends the current cell to this notebook’s Python kernel, which runs it and returns the output. The result is displayed below the cell, and since we reached the end of the notebook, a new cell is automatically created. Go through the User Interface Tour from Jupyter’s Help menu to learn the basics.

Một cuốn sổ chứa danh sách các ô. Mỗi ô có thể chứa code hoặc văn bản được định dạng. Ngay bây giờ sổ ghi chép chỉ chứa một ô code trống, có nhãn là In [1]:. Hãy thử gõ print ("Hello world!") Trong ô và nhấp vào nút play (xem Hình 2-4) hoặc nhấn Shift-Enter. Điều này sẽ gửi ô hiện tại đến kernel Python, nó chạy nó và trả về đầu ra. Kết quả được hiển thị bên dưới ô và vì chúng tôi đã đến cuối sổ ghi chép, một ô mới sẽ tự động được tạo. Đi qua the User Interface Tour của Jupyter để tìm hiểu những điều cơ bản.



*Figure 2-4. Hello world Python notebook/ Hello world trong Python notebook*



**48** **|** **Chapter 2: End-to-End Machine Learning Project**

**Download the Data/Tải dữ liệu**

In typical environments your data would be available in a relational database (or some other common datastore) and spread across multiple tables/documents/files. To access it, you would first need to get your credentials and access authorizations,[10](#page13) and familiarize yourself with the data schema. In this project, however, things are much simpler: you will just download a single compressed file, *housing.tgz*, which contains a comma-separated value (CSV) file called *housing.csv* with all the data.

Trong các môi trường điển hình, dữ liệu của bạn sẽ có sẵn trong cơ sở dữ liệu quan hệ (hoặc một số kho dữ liệu phổ biến khác) và trải rộng trên nhiều bảng / tài liệu / tệp. Để truy cập nó, trước tiên bạn cần có được thông tin đăng nhập và quyền truy cập, và tự làm quen với lược đồ dữ liệu. Tuy nhiên, trong dự án này, mọi thứ đơn giản hơn nhiều: bạn sẽ chỉ cần tải xuống một tệp nén duy nhất, housing.tgz, chứa tệp giá trị được phân tách bằng dấu phẩy (CSV) có tên là housing.csv với tất cả dữ liệu.

You could use your web browser to download it, and run tar xzf housing.tgz to decompress the file and extract the CSV file, but it is preferable to create a small function to do that. It is useful in particular if data changes regularly, as it allows you to write a small script that you can run whenever you need to fetch the latest data (or you can set up a scheduled job to do that automatically at regular intervals). Auto‐ mating the process of fetching the data is also useful if you need to install the dataset on multiple machines.

Bạn có thể sử dụng trình duyệt web của mình để tải xuống và chạy tar xzf housing.tgz để giải nén tệp và giải nén tệp CSV, nhưng tốt nhất là tạo một hàm nhỏ để thực hiện điều đó. Đặc biệt hữu ích nếu dữ liệu thay đổi thường xuyên, vì nó cho phép bạn viết một tập lệnh nhỏ mà bạn có thể chạy bất cứ khi nào bạn cần lấy dữ liệu mới nhất (hoặc bạn có thể thiết lập công việc được lên lịch để thực hiện tự động theo định kỳ). Tự động kết hợp quá trình tìm nạp dữ liệu cũng hữu ích nếu bạn cần cài đặt bộ dữ liệu trên nhiều máy.

Here is the function to fetch the data: [11](#page13)

Đây là hàm lấy dữ liệu:

**import os import tarfile**

**from six.moves 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()

Now when you call fetch\_housing\_data(), it creates a *datasets/housing* directory in your workspace, downloads the *housing.tgz* file, and extracts the *housing.csv* from it in this directory.

Bây giờ khi bạn gọi đến fetch\_housing\_data (), nó sẽ tạo một thư mục *datasets/housing* ở trong không gian làm việc của bạn, tải xuống tệp *housing*.tgz và giải nén tệp *housing*.csv từ thư mục này.

Now let’s load the data using Pandas. Once again you should write a small function to load the data:

Bây giờ, hãy để tải dữ liệu bằng cách sử dụng Pandas. Một lần nữa, bạn nên viết một hàm nhỏ để tải dữ liệu:

**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)

This function returns a Pandas DataFrame object containing all the data.

Hàm này trả về một đối tượng Pandas DataFrame chứa tất cả dữ liệu.



1. You might also need to check legal constraints, such as private fields that should never be copied to unsafe datastores.
2. In a real project you would save this code in a Python file, but for now you can just write it in your Jupyter notebook.

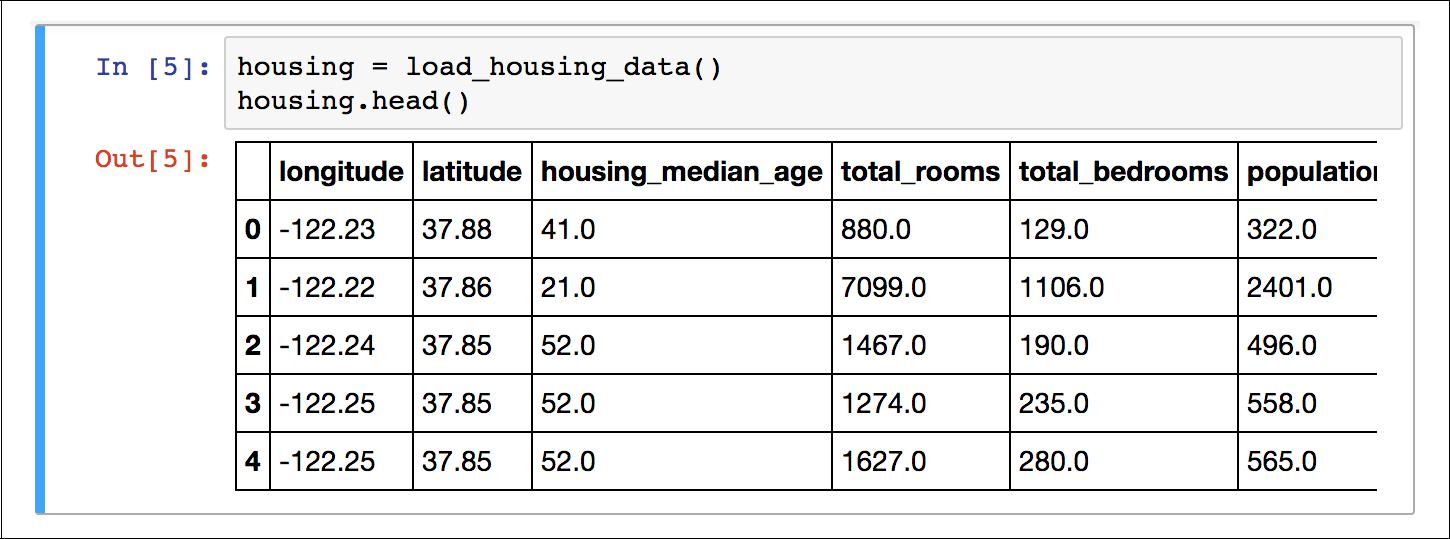


**Get the Data** **|** **49**

**Take a Quick Look at the Data Structure/Xem nhanh cấu trúc dữ liệu**

Let’s take a look at the top five rows using the DataFrame’s head() method (see [Figure 2-5](#page14)).

Hãy cùng nhìn 5 dòng đầu sử dựng hàm *head() của* DataFrame’s (như hình 2-5)



*Figure 2-5. Top five rows in the dataset/5 dòng đầu trong tập dữ liệu*

Each row represents one district. There are 10 attributes (you can see the first 6 in the screenshot): longitude, latitude, housing\_median\_age, total\_rooms, total\_bed

rooms, population, households, median\_income, median\_house\_value, and ocean\_proximity.

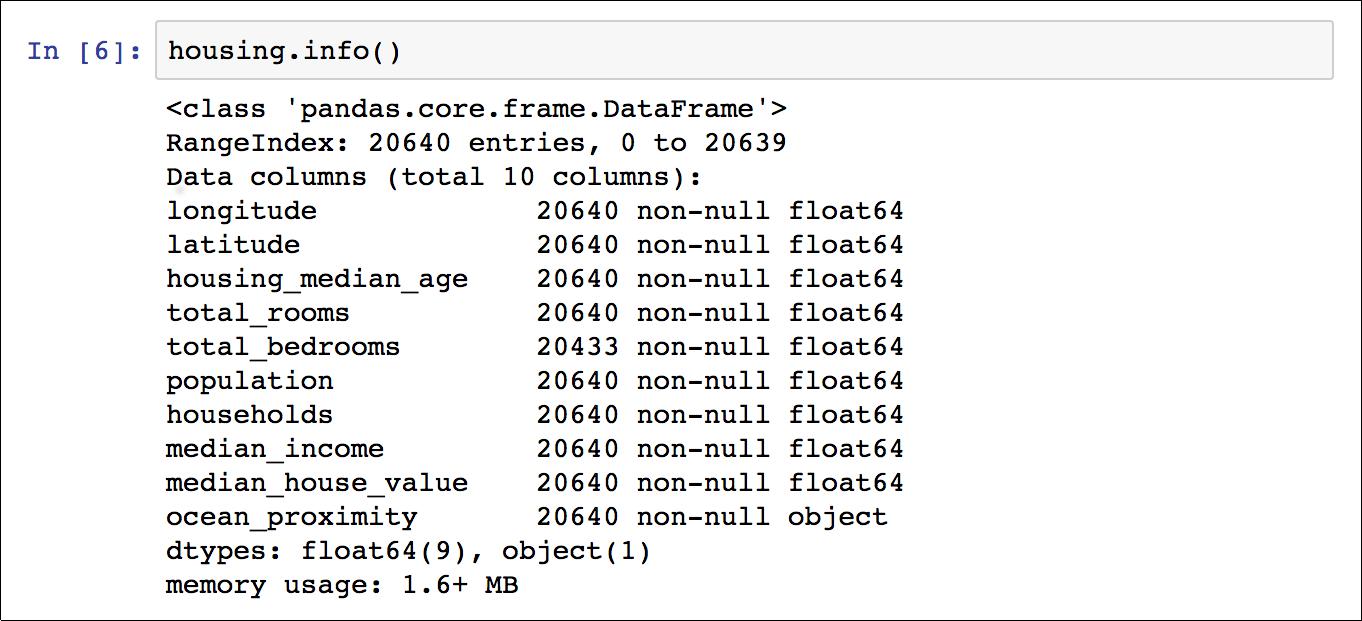
Mỗi dòng đại diện cho 1 quận. Có 10 thuộc tính(bạn có thể thấy 6 cái đầu tiên trong ảnh chụp màn hinh): kinh độ, vĩ độ, tuổi trung bình, số phòng, số phòng ngủ, dân số, số hộ gia đình, thu nhập trung bình, giá nhà trung bình, gần biển.

The info() method is useful to get a quick description of the data, in particular the total number of rows, and each attribute’s type and number of non-null values (see [Figure 2-6](#page15)).

Hàm info() rất hữu ích để có được một mô tả nhanh về dữ liệu, đặc biệt là tổng số hàng và từng loại thuộc tính và số giá trị không null (xem Hình 2-6).



**50** **|** **Chapter 2: End-to-End Machine Learning Project**



*Figure 2-6. Housing info/Thông tin nhà*

There are 20,640 instances in the dataset, which means that it is fairly small by Machine Learning standards, but it’s perfect to get started. Notice that the total\_bed rooms attribute has only 20,433 non-null values, meaning that 207 districts are missing this feature. We will need to take care of this later.

Có 20.640 cá thể trong bộ dữ liệu, điều đó có nghĩa là nó khá nhỏ theo tiêu chuẩn Machine Learning, nhưng nó hoàn hảo để bắt đầu. Lưu ý rằng thuộc tính total\_bed\_rooms chỉ có 20.433 giá trị không null, nghĩa là có 207 quận đang bỏ lỡ đặc trưng này. Chúng ta sẽ cần phải chăm tâm điều này sau.

All attributes are numerical, except the ocean\_proximity field. Its type is object, so it could hold any kind of Python object, but since you loaded this data from a CSV file you know that it must be a text attribute. When you looked at the top five rows, you probably noticed that the values in the ocean\_proximity column were repetitive, which means that it is probably a categorical attribute. You can find out what categories exist and how many districts belong to each category by using the value\_counts() method:

Tất cả các thuộc tính là số, ngoại trừ trường ocean\_proximity. Kiểu của nó là object, vì vậy nó có thể chứa bất kỳ loại đối tượng Python nào, nhưng vì bạn đã tải dữ liệu này từ tệp CSV, bạn biết rằng nó phải là một thuộc tính văn bản. Khi bạn nhìn vào năm hàng trên cùng, bạn có thể nhận thấy rằng các giá trị trong cột ocean\_proximity là lặp đi lặp lại, điều đó có nghĩa là nó có thể là một thuộc tính phân loại. Bạn có thể tìm hiểu những gì phân loại tồn tại và có bao nhiêu quận thuộc về mỗi loại bằng cách sử dụng hàm value\_counts ():

**>>>** housing["ocean\_proximity"].value\_counts()

|  |  |  |
| --- | --- | --- |
| <1H OCEAN | | 9136 |
| INLAND | | 6551 |
| NEAR | OCEAN | 2658 |
| NEAR | BAY | 2290 |
| ISLAND | | 5 |

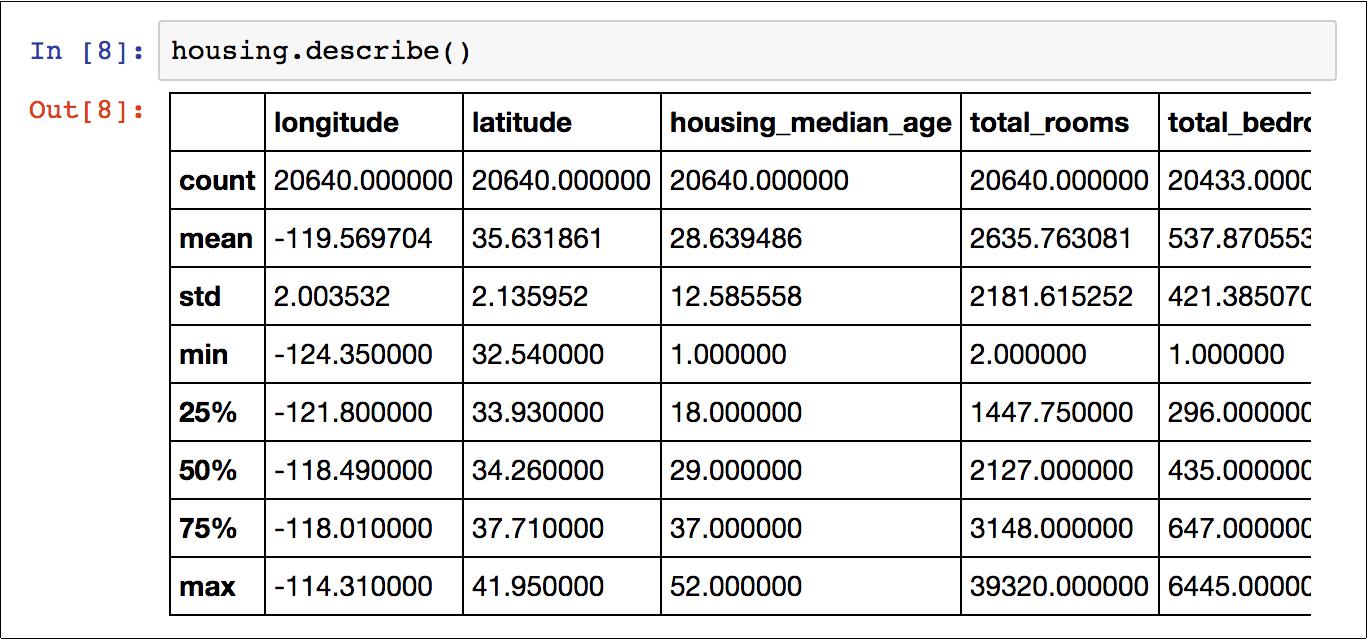
Name: ocean\_proximity, dtype: int64

Let’s look at the other fields. The describe() method shows a summary of the numerical attributes ([Figure 2-7](#page16)).

Hãy nhìn vào các trường khác. Hàm description () hiển thị tóm tắt các thuộc tính số (Hình 2-7).



**Get the Data** **|** **51**



*Figure 2-7. Summary of each numerical attribute*/*Hình 2-7. Tóm tắt từng thuộc tính số*

The count, mean, min, and max rows are self-explanatory. Note that the null values are ignored (so, for example, count of total\_bedrooms is 20,433, not 20,640). The std row shows the *standard deviation*, which measures how dispersed the values are.[12](#page16) The 25%, 50%, and 75% rows show the corresponding *percentiles*: a percentile indicates the value below which a given percentage of observations in a group of observations falls. For example, 25% of the districts have a housing\_median\_age lower than 18, while 50% are lower than 29 and 75% are lower than 37. These are often called the 25th percentile (or 1st *quartile*), the median, and the 75th percentile (or 3rd quartile).

Các số đếm, trung bình, tối thiểu và tối đa trên các dòng là các số biểu thị cho các giá trị. Lưu ý rằng các giá trị null được bỏ qua (vì vậy, ví dụ: tổng số total\_bedroom là 20,433, không phải 20,640). Hàng std hiển thị *standard deviation*, đo lường mức độ phân tán của các giá trị. Các hàng 25%, 50% và 75% hiển thị phân vị tương ứng: một phân vị chỉ ra giá trị dưới tỷ lệ phần trăm quan sát của một nhóm quan sát. Ví dụ: 25% các quận có housing\_median\_age thấp hơn 18, trong khi 50% thấp hơn 29 và 75% thấp hơn 37. Chúng thường được gọi là phân vị thứ 25 (phần tư thứ nhất), trung vị và phân vị thứ 75 (hoặc phần tư thứ 3).

Another quick way to get a feel of the type of data you are dealing with is to plot a histogram for each numerical attribute. A histogram shows the number of instances (on the vertical axis) that have a given value range (on the horizontal axis). You can either plot this one attribute at a time, or you can call the hist() method on the whole dataset, and it will plot a histogram for each numerical attribute (see [Figure 2-8](#page17)). For example, you can see that slightly over 800 districts have a median\_house\_value equal to about $100,000.

Một cách nhanh chóng khác để hiểu về loại dữ liệu bạn đang xử lý là vẽ biểu đồ cho mỗi thuộc tính số. Biểu đồ cho thấy số lượng cá thể (trên trục tung) có phạm vi giá trị nhất định (trên trục hoành). Bạn có thể vẽ một thuộc tính này tại một thời điểm hoặc bạn có thể gọi hàm hist () trên toàn bộ tập dữ liệu và nó sẽ vẽ biểu đồ cho mỗi thuộc tính số (xem Hình 2-8). Ví dụ: bạn có thể thấy rằng hơn 800 quận có một median\_house\_value tương đương với khoảng 100.000 đô la.

%matplotlib inline *# only in a Jupyter notebook* **import matplotlib.pyplot as plt** housing.hist(bins=50, figsize=(20,15)) plt.show()



1. The standard deviation is generally denoted σ (the Greek letter sigma), and it is the square root of the *variance*, which is the average of the squared deviation from the mean. When a feature has a bell-shaped *normal distribution* (also called a *Gaussian distribution*), which is very common, the “68-95-99.7” rule applies: about68% of the values fall within 1σ of the mean, 95% within 2σ, and 99.7% within 3σ.

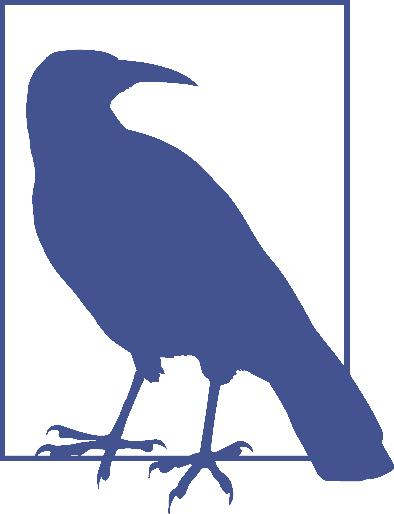
Độ lệch chuẩn thường được ký hiệu là σ (sigma chữ Hy Lạp) và nó là căn bậc hai của phương sai, là trung bình của độ lệch bình phương so với giá trị trung bình. Khi một đặc trưng có phân phối chuẩn hóa hình chuông (còn được gọi là phân phối Gaussian), rất phổ biến, áp dụng quy tắc “68-95-99.7”: khoảng 68% giá trị nằm trong khoảng 1σ giá trị trung bình, 95% trong phạm vi trung bình 2σ và 99,7% trong vòng 3σ.



**52** **|** **Chapter 2: End-to-End Machine Learning Project**

The hist() method relies on Matplotlib, which in turn relies on a user-specified graphical backend to draw on your screen. So before you can plot anything, you need to specify which backend Matplot‐ lib should use. The simplest option is to use Jupyter’s magic command %matplotlib inline. This tells Jupyter to set up Matplotlib so it uses Jupyter’s own backend. Plots are then rendered within the notebook itself. Note that calling show() is optional in a Jupyter notebook, as Jupyter will automatically display plots when a cell is executed.

Hàm hist () dựa trên Matplotlib, dựa trên phần phụ trợ đồ họa do người dùng chỉ định để vẽ trên màn hình của bạn. Vì vậy, trước khi bạn có thể vẽ bất cứ điều gì, bạn cần chỉ định Matplot‐ lib nào nên sử dụng. Tùy chọn đơn giản nhất là sử dụng Jupyter’s magic command, %matplotlib inline. Điều này nói với Jupyter để thiết lập Matplotlib để nó sử dụng backend của Jupyter’s backend. Các bản vẽ sau đó được hiển thị trong chính máy tính xách tay của bạn. Lưu ý rằng hàm show () là tùy chọn trong Jupyter notebook, vì Jupyter sẽ tự động hiển thị các bản vẽ khi một ô được thực thi.



*Figure 2-8. A histogram for each numerical attribute/Biều đồ cho mỗi thuộc tính số*

Notice a few things in these histograms:

Lưu ý một vài điều trong các biểu đồ này:

1. First, the median income attribute does not look like it is expressed in US dollars (USD). After checking with the team that collected the data, you are told that the data has been scaled and capped at 15 (actually 15.0001) for higher median incomes, and at 0.5 (actually 0.4999) for lower median incomes. The numbers represent roughly tens of thousands of dollars (e.g., 3 actually means about $30,000). Working with pre-processed attributes is common in Machine Learning, and it is not necessarily a problem, but you should try to understand how the data was computed.

Đầu tiên, thuộc tính thu nhập trung bình không biểu thị bằng đô la Mỹ (USD). Sau khi kiểm tra với nhóm đã thu thập dữ liệu, bạn được thông báo rằng dữ liệu đã được thu nhỏ và giới hạn ở mức 15 (thực tế là 15.0001) cho thu nhập trung bình cao hơn và ở mức 0,5 (thực tế là 0,4999) cho thu nhập trung bình thấp hơn. Các con số đại diện cho hàng chục ngàn đô la (ví dụ: 3 thực sự có nghĩa là khoảng 30.000 đô la). Làm việc với các thuộc tính được tiền xử lý là phổ biến trong Machine Learning và nó không nhất thiết là vấn đề, nhưng bạn nên cố gắng hiểu cách dữ liệu được tính toán.

1. The housing median age and the median house value were also capped. The latter may be a serious problem since it is your target attribute (your labels). Your Machine Learning algorithms may learn that prices never go beyond that limit. You need to check with your client team (the team that will use your system’s output) to see if this is a problem or not. If they tell you that they need precise predictions even beyond $500,000, then you have mainly two options:

Tuổi đời ngôi nhà và giá trị nhà trung bình cũng được giới hạn. Cái sau có thể là một vấn đề nghiêm trọng vì đó là thuộc tính mục tiêu của bạn (nhãn của bạn). Các thuật toán Machine Learning của bạn có thể học được rằng giá không bao giờ vượt quá giới hạn đó. Bạn cần kiểm tra với nhóm khách hàng của mình (nhóm sẽ sử dụng kết quả đầu ra của hệ thống của bạn) để xem đây có phải là vấn đề hay không. Nếu họ nói với bạn rằng họ cần những dự đoán chính xác thậm chí vượt quá 500.000 đô la, thì bạn chủ yếu có hai lựa chọn:

* 1. a . Collect proper labels for the districts whose labels were capped.

Thu thập nhãn thích hợp cho các quận có nhãn được giới hạn.

* 1. b. Remove those districts from the training set (and also from the test set, since your system should not be evaluated poorly if it predicts values beyond $500,000).

Xóa các quận đó khỏi tập huấn luyện (và cả từ tập test, vì hệ thống của bạn không nên được đánh giá kém nếu dự đoán giá trị vượt quá 500.000 đô la).

1. These attributes have very different scales. We will discuss this later in this chapter when we explore feature scaling.

Các thuộc tính này có quy mô rất khác nhau. Chúng ta sẽ thảo luận điều này sau trong phần này khi chúng ta khám phá đặc trưng nhân rộng.

1. Finally, many histograms are *tail heavy*: they extend much farther to the right of the median than to the left. This may make it a bit harder for some Machine Learning algorithms to detect patterns. We will try transforming these attributes later on to have more bell-shaped distributions.

Cuối cùng, nhiều biểu đồ có *tail heavy*: chúng kéo dài ra xa hơn về phía bên phải của dải phân cách so với bên trái. Điều này có thể làm cho một số thuật toán Machine Learning khó phát hiện hơn một chút. Chúng tôi sẽ thử chuyển đổi các thuộc tính này sau để có nhiều bản phân phối hình chuông hơn.

Hopefully you now have a better understanding of the kind of data you are dealing with.

Hy vọng rằng bây giờ bạn đã hiểu rõ hơn về loại dữ liệu bạn đang xử lý.

Wait! Before you look at the data any further, you need to create a test set, put it aside, and never look at it.

Chờ đợi! Trước khi bạn xem xét mọi dữ liệu, bạn cần tạo một tập test, đặt nó sang một bên và không bao giờ nhìn vào nó.





**Get the Data** **|**

**Create a Test Set/Tạo một tập test**

It may sound strange to voluntarily set aside part of the data at this stage. After all, you have only taken a quick glance at the data, and surely you should learn a whole lot more about it before you decide what algorithms to use, right? This is true, but your brain is an amazing pattern detection system, which means that it is highly prone to overfitting: if you look at the test set, you may stumble upon some seemingly interesting pattern in the test data that leads you to select a particular kind of Machine Learning model. When you estimate the generalization error using the test set, your estimate will be too optimistic and you will launch a system that will not perform as well as expected. This is called *data snooping* bias.

Nghe có vẻ lạ khi tự nguyện dành một phần dữ liệu ở giai đoạn này. Rốt cuộc, bạn chỉ lướt qua dữ liệu nhanh chóng, và chắc chắn bạn nên tìm hiểu thêm rất nhiều về nó trước khi bạn quyết định sử dụng thuật toán nào, phải không? Điều này là đúng, nhưng bộ não của bạn là một hệ thống phát hiện mẫu tuyệt vời, điều đó có nghĩa là nó rất dễ bị overfitting: nếu bạn nhìn vào bộ test, bạn có thể vấp phải một số mẫu có vẻ thú vị trong dữ liệu test dẫn bạn chọn loại mô hình Machine Learning. Khi bạn ước tính lỗi tổng quát hóa bằng bộ test, ước tính của bạn sẽ quá lạc quan và bạn sẽ khởi chạy một hệ thống sẽ không hoạt động tốt như mong đợi. Điều này được gọi là *data snooping* bias.

Creating a test set is theoretically quite simple: just pick some instances randomly, typically 20% of the dataset (or less if your dataset is very large), and set them aside:

Tạo một bộ kiểm tra về mặt lý thuyết khá đơn giản: chỉ cần chọn ngẫu nhiên một số trường hợp, thường là 20% số liệu (hoặc ít hơn nếu tập dữ liệu của bạn rất lớn) và đặt chúng sang một bên:



**54** **|** **Chapter 2: End-to-End Machine Learning Project**

**import numpy as np**

**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]

You can then use this function like this: [13](#page19)

Sau đó bạn có thể sử dụng chức năng này như thế này

* train\_set, test\_set = split\_train\_test(housing, 0.2)
* len(train\_set)

16512

**>>>** len(test\_set)4128

Well, this works, but it is not perfect: if you run the program again, it will generate a different test set! Over time, you (or your Machine Learning algorithms) will get to see the whole dataset, which is what you want to avoid.

Chà, điều này hoạt động, nhưng nó không hoàn hảo: nếu bạn chạy lại chương trình, nó sẽ tạo ra một bộ thử nghiệm khác! Theo thời gian, bạn (hoặc thuật toán Machine Learning) sẽ thấy toàn bộ tập dữ liệu, đây là điều bạn muốn tránh.

One solution is to save the test set on the first run and then load it in subsequent runs. Another option is to set the random number generator’s seed (e.g., np.ran dom.seed(42))[14](#page19) before calling np.random.permutation(), so that it always generates the same shuffled indices.

Một giải pháp là lưu tập kiểm tra vào lần chạy đầu tiên và sau đó tải nó trong các lần chạy tiếp theo. Một tùy chọn khác là đặt hạt giống bộ tạo số ngẫu nhiên (ví dụ: np.ran dom.seed (42)) 14 trước khi gọi np.random.permuting () để nó luôn tạo ra các chỉ số được xáo trộn tương tự.

But both these solutions will break next time you fetch an updated dataset. A com‐ mon solution is to use each instance’s identifier to decide whether or not it should go in the test set (assuming instances have a unique and immutable identifier). For example, you could compute a hash of each instance’s identifier and put that instance in the test set if the hash is lower or equal to 20% of the maximum hash value. This ensures that the test set will remain consistent across multiple runs, even if you refresh the dataset. The new test set will contain 20% of the new instances, but it will not contain any instance that was previously in the training set. Here is a possible implementation:

Nhưng cả hai giải pháp này sẽ phá vỡ lần tới khi bạn lấy một tập dữ liệu cập nhật. Một giải pháp đơn giản là sử dụng từng mã định danh cá thể để xác định xem có nên đi trong tập kiểm tra hay không (giả sử các trường hợp có một định danh duy nhất và không thay đổi). Ví dụ: bạn có thể tính toán một hàm băm của từng cá thể Định danh và đưa trường hợp đó vào bộ kiểm tra nếu giá trị băm thấp hơn hoặc bằng 20% ​​giá trị băm tối đa. Điều này đảm bảo rằng bộ kiểm tra sẽ duy trì nhất quán trong nhiều lần chạy, ngay cả khi bạn làm mới tập dữ liệu. Bộ kiểm tra mới sẽ chứa 20% các trường hợp mới, nhưng nó sẽ không chứa bất kỳ trường hợp nào trước đây trong tập huấn luyện. Đây là một triển khai có thể:

**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]



1. In this book, when a code example contains a mix of code and outputs, as is the case here, it is formatted like in the Python interpreter, for better readability: the code lines are prefixed with >>> (or ... for indented blocks), and the outputs have no prefix.
2. You will often see people set the random seed to 42. This number has no special property, other than to be The Answer to the Ultimate Question of Life, the Universe, and Everything.



**Get the Data** **|** **55**

in\_test\_set = ids.apply(**lambda** id\_: test\_set\_check(id\_, test\_ratio)) **return** data.loc[~in\_test\_set],data.loc[in\_test\_set]

Unfortunately, the housing dataset does not have an identifier column. The simplest solution is to use the row index as the ID:

Thật không may, bộ dữ liệu nhà ở không có cột định danh. Giải pháp đơn giản nhất là sử dụng chỉ mục hàng làm ID:

|  |  |  |
| --- | --- | --- |
| 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") |

If you use the row index as a unique identifier, you need to make sure that new data gets appended to the end of the dataset, and no row ever gets deleted. If this is not possible, then you can try to use the most stable features to build a unique identifier. For example, a district’s latitude and longitude are guaranteed to be stable for a few million years, so you could combine them into an ID like so: [15](#page20)

Nếu bạn sử dụng chỉ mục hàng làm định danh duy nhất, bạn cần đảm bảo rằng dữ liệu mới sẽ được thêm vào cuối tập dữ liệu và không có hàng nào bị xóa. Nếu điều này là không thể, thì bạn có thể thử sử dụng các tính năng ổn định nhất để xây dựng một định danh duy nhất. Ví dụ: vĩ độ và kinh độ của huyện được đảm bảo ổn định trong vài triệu năm, vì vậy bạn có thể kết hợp chúng thành một ID như vậy:

housing\_with\_id["id"] = housing["longitude"] \* 1000 + housing["latitude"] train\_set, test\_set = split\_train\_test\_by\_id(housing\_with\_id, 0.2, "id")

Scikit-Learn provides a few functions to split datasets into multiple subsets in various ways. The simplest function is train\_test\_split, which does pretty much the same thing as the function split\_train\_test defined earlier, with a couple of additional features. First there is a random\_state parameter that allows you to set the random generator seed as explained previously, and second you can pass it multiple datasets with an identical number of rows, and it will split them on the same indices (this is very useful, for example, if you have a separate DataFrame for labels):

Scikit-Learn cung cấp một vài chức năng để phân chia bộ dữ liệu thành nhiều tập con theo nhiều cách khác nhau. Hàm đơn giản nhất là train\_test\_split, hoạt động khá giống với hàm split\_train\_test được xác định trước đó, với một vài tính năng bổ sung. Đầu tiên, có một tham số Random\_state cho phép bạn đặt hạt giống trình tạo ngẫu nhiên như đã giải thích trước đó và thứ hai bạn có thể truyền cho nó nhiều bộ dữ liệu với số lượng hàng giống nhau và nó sẽ phân chia chúng trên cùng một chỉ mục (điều này rất hữu ích, cho ví dụ: nếu bạn có một DataFrame riêng cho nhãn):

**from sklearn.model\_selection import** train\_test\_split

train\_set, test\_set = train\_test\_split(housing, test\_size=0.2, random\_state=42)

So far we have considered purely random sampling methods. This is generally fine if your dataset is large enough (especially relative to the number of attributes), but if it is not, you run the risk of introducing a significant sampling bias. When a survey company decides to call 1,000 people to ask them a few questions, they don’t just pick 1,000 people randomly in a phone book. They try to ensure that these 1,000 people are representative of the whole population. For example, the US population is com‐ posed of 51.3% female and 48.7% male, so a well-conducted survey in the US would try to maintain this ratio in the sample: 513 female and 487 male. This is called *strati‐* *fied sampling*: the population is divided into homogeneous subgroups called *strata*,and the right number of instances is sampled from each stratum to guarantee that the test set is representative of the overall population. If they used purely random sam‐ pling, there would be about 12% chance of sampling a skewed test set with either less than 49% female or more than 54% female. Either way, the survey results would be significantly biased.

Cho đến nay chúng tôi đã xem xét các phương pháp lấy mẫu hoàn toàn ngẫu nhiên. Điều này thường ổn nếu tập dữ liệu của bạn đủ lớn (đặc biệt liên quan đến số lượng thuộc tính), nhưng nếu không, bạn có nguy cơ đưa ra xu hướng lấy mẫu đáng kể. Khi một công ty khảo sát quyết định gọi 1.000 người để hỏi họ một vài câu hỏi, họ không chỉ chọn ngẫu nhiên 1.000 người trong một danh bạ điện thoại. Họ cố gắng đảm bảo rằng 1.000 người này là đại diện của toàn dân. Ví dụ, dân số Hoa Kỳ được xác định là 51,3% nữ và 48,7% nam, vì vậy một cuộc khảo sát được tiến hành tốt ở Mỹ sẽ cố gắng duy trì tỷ lệ này trong mẫu: 513 nữ và 487 nam. Điều này được gọi là lấy mẫu phân tầng: dân số được chia thành các nhóm nhỏ đồng nhất được gọi là tầng và số lượng cá thể đúng được lấy mẫu từ mỗi tầng để đảm bảo rằng bộ kiểm tra là đại diện cho tổng thể. Nếu họ sử dụng phương pháp lấy mẫu hoàn toàn ngẫu nhiên, sẽ có khoảng 12% cơ hội lấy mẫu thử nghiệm sai lệch với tỷ lệ nữ ít hơn 49% hoặc nữ hơn 54%. Dù bằng cách nào, kết quả khảo sát sẽ bị sai lệch đáng kể.



1. The location information is actually quite coarse, and as a result many districts will have the exact same ID, so they will end up in the same set (test or train). This introduces some unfortunate sampling bias.



**56** **|** **Chapter 2: End-to-End Machine Learning Project**

Suppose you chatted with experts who told you that the median income is a very important attribute to predict median housing prices. You may want to ensure that the test set is representative of the various categories of incomes in the whole dataset. Since the median income is a continuous numerical attribute, you first need to create an income category attribute. Let’s look at the median income histogram more closely (back in [Figure 2-8](#page17)): most median income values are clustered around 1.5 to 6 (i.e., $15,000–$60,000), but some median incomes go far beyond 6. It is important to have a sufficient number of instances in your dataset for each stratum, or else the estimate of the stratum’s importance may be biased. This means that you should not have too many strata, and each stratum should be large enough. The following code uses the pd.cut() function to create an income category attribute with 5 categories (labeled from 1 to 5): category 1 ranges from 0 to 1.5 (i.e., less than $15,000), category 2 from 1.5 to 3, and so on:

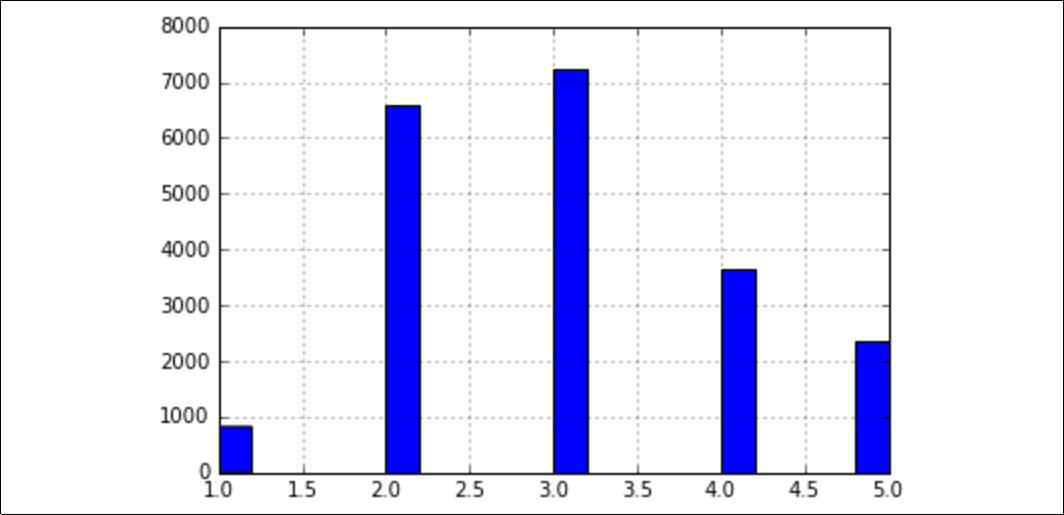
Giả sử bạn trò chuyện với các chuyên gia nói với bạn rằng thu nhập trung bình là một thuộc tính rất quan trọng để dự đoán giá nhà ở trung bình. Bạn có thể muốn đảm bảo rằng bộ kiểm tra là đại diện cho các loại thu nhập khác nhau trong toàn bộ tập dữ liệu. Vì thu nhập trung bình là một thuộc tính số liên tục, trước tiên bạn cần tạo thuộc tính loại thu nhập. Chúng ta hãy xem xét biểu đồ thu nhập trung bình chặt chẽ hơn (quay lại trong Hình 2-8): hầu hết các giá trị thu nhập trung bình được tập hợp trong khoảng 1,5 đến 6 (ví dụ: 15.000 đô la 60.000 đô la), nhưng một số thu nhập trung bình vượt xa 6. Điều quan trọng là phải có một số lượng đủ các trường hợp trong tập dữ liệu của bạn cho mỗi tầng hoặc nếu không thì ước tính tầm quan trọng của tầng có thể bị sai lệch. Điều này có nghĩa là bạn không nên có quá nhiều tầng lớp và mỗi tầng phải đủ lớn. Đoạn mã sau sử dụng hàm pd.cut () để tạo thuộc tính loại thu nhập với 5 loại (được gắn nhãn từ 1 đến 5): loại 1 nằm trong khoảng từ 0 đến 1,5 (tức là dưới 15.000 đô la), loại 2 từ 1,5 đến 3, và như thế:

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])

These income categories are represented in [Figure 2-9](#page21):

housing["income\_cat"].hist()



*Figure 2-9. Histogram of income categories*

Now you are ready to do stratified sampling based on the income category. For this you can use Scikit-Learn’s StratifiedShuffleSplit class:

**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]



**Get the Data** **|** **57**

Let’s see if this worked as expected. You can start by looking at the income category proportions in the test set:

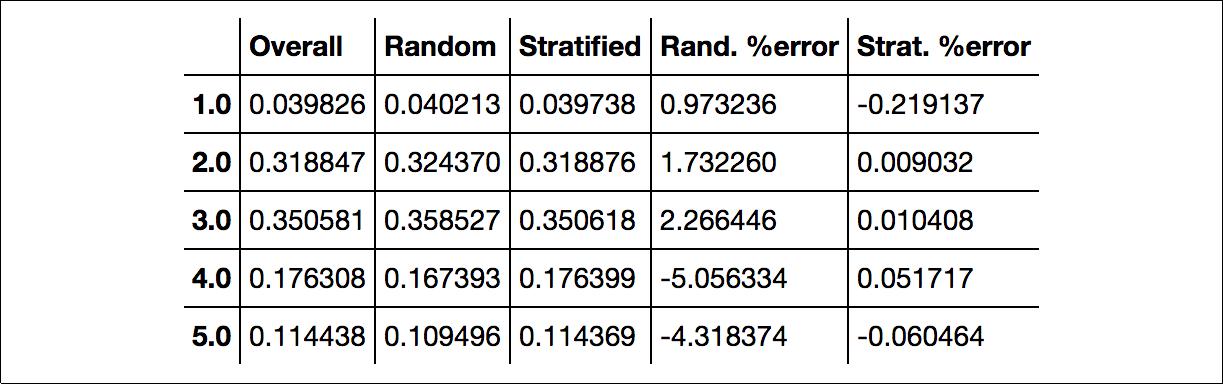
**>>>** strat\_test\_set["income\_cat"].value\_counts()/len(strat\_test\_set)

|  |  |
| --- | --- |
| 3 | 0.350533 |
| 2 | 0.318798 |

1. 0.176357
2. 0.114583
3. 0.039729

Name: income\_cat, dtype: float64

With similar code you can measure the income category proportions in the full data‐ set. [Figure 2-10](#page22) compares the income category proportions in the overall dataset, in the test set generated with stratified sampling, and in a test set generated using purely random sampling. As you can see, the test set generated using stratified sampling has income category proportions almost identical to those in the full dataset, whereas the test set generated using purely random sampling is quite skewed.



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

Now you should remove the income\_cat attribute so the data is 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.

**Discover 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 a little bit more in depth.

First, make sure you have put the test set aside and you are only exploring the train‐ ing set. Also, if the training set is very large, you may want to sample an exploration



**58** **|** **Chapter 2: End-to-End Machine Learning Project**

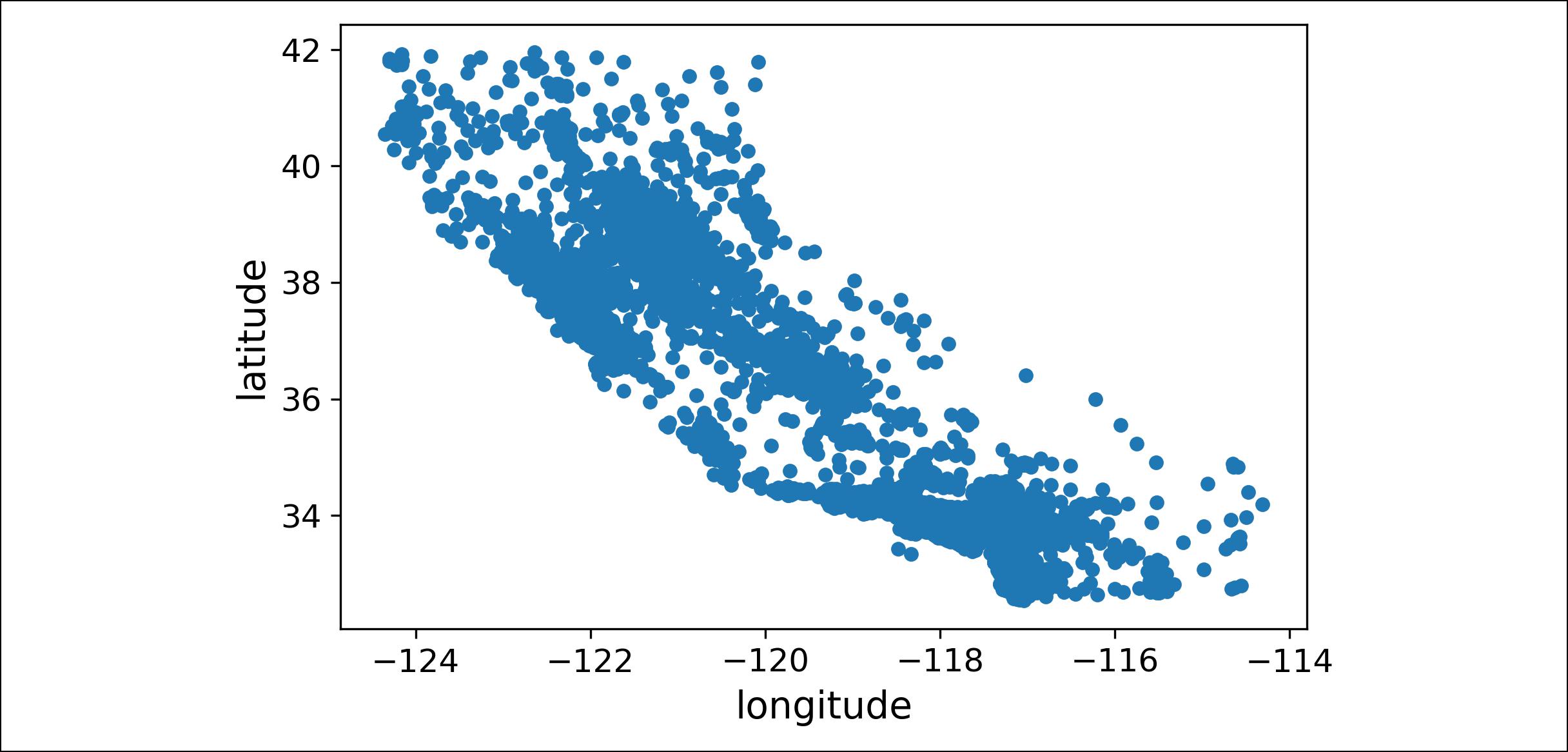
set, to make manipulations easy and fast. In our case, the set is quite small so you can just work directly on the full set. Let’s create a copy so you can play with it without harming the training set:

housing = strat\_train\_set.copy()

**Visualizing Geographical Data**

Since there is geographical information (latitude and longitude), it is a good idea to create a scatterplot of all districts to visualize the data ([Figure 2-11](#page23)):

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



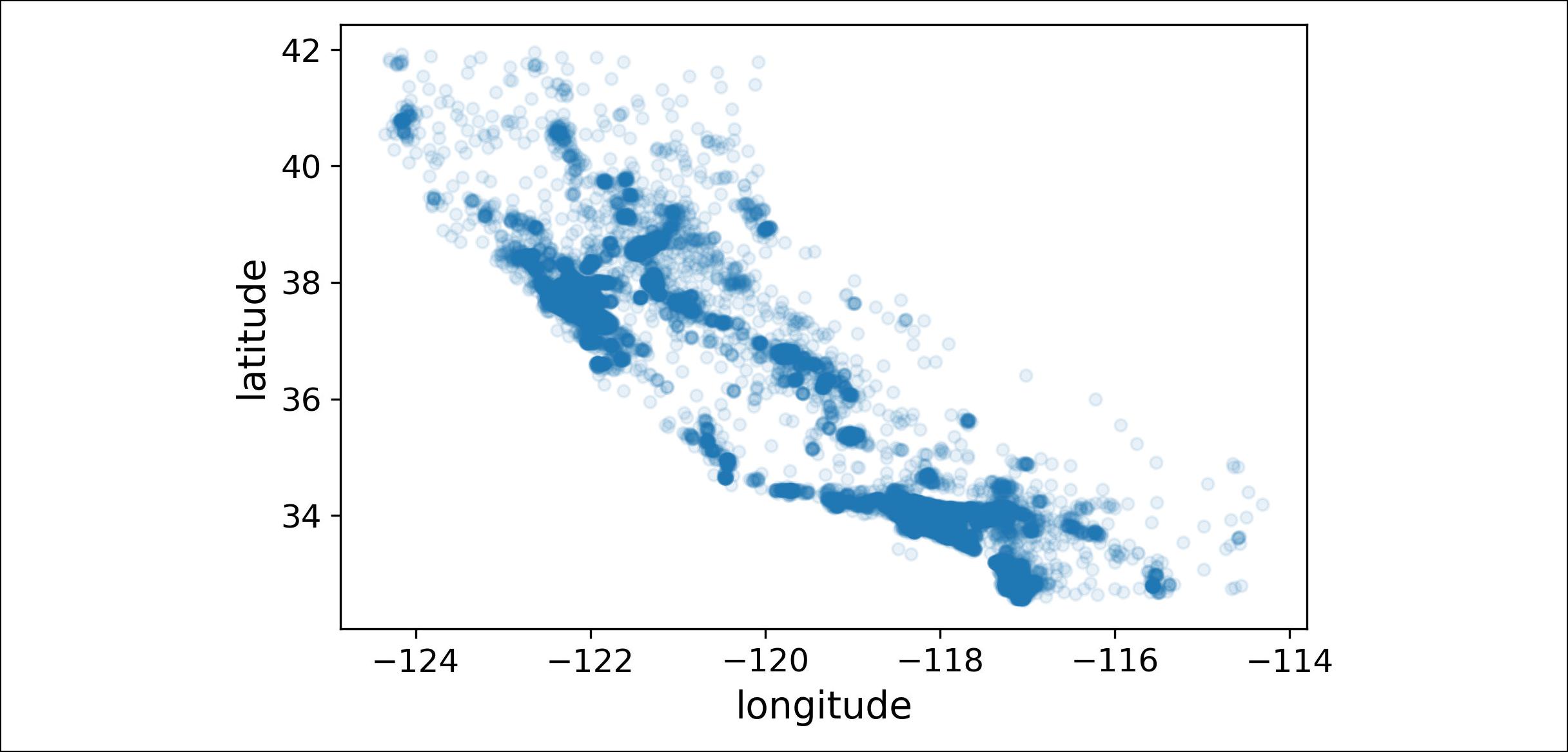
*Figure 2-11. A geographical scatterplot of the data*

This looks like California all right, but other than that it is hard to see any particular pattern. Setting the alpha option to 0.1 makes it much easier to visualize the places where there is a high density of data points ([Figure 2-12](#page24)):

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



**Discover and Visualize the Data to Gain Insights** **|** **59**



*Figure 2-12. A better visualization highlighting high-density areas*

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 in the Central Valley, in particular around Sacramento and Fresno.

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

Now let’s look at the housing prices ([Figure 2-13](#page25)). The radius of each circle represents the district’s population (option s), and the color represents the price (option c). We will use a predefined color map (option cmap) called jet, which ranges from blue (low values) to red (high prices):[16](#page24)

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,

)

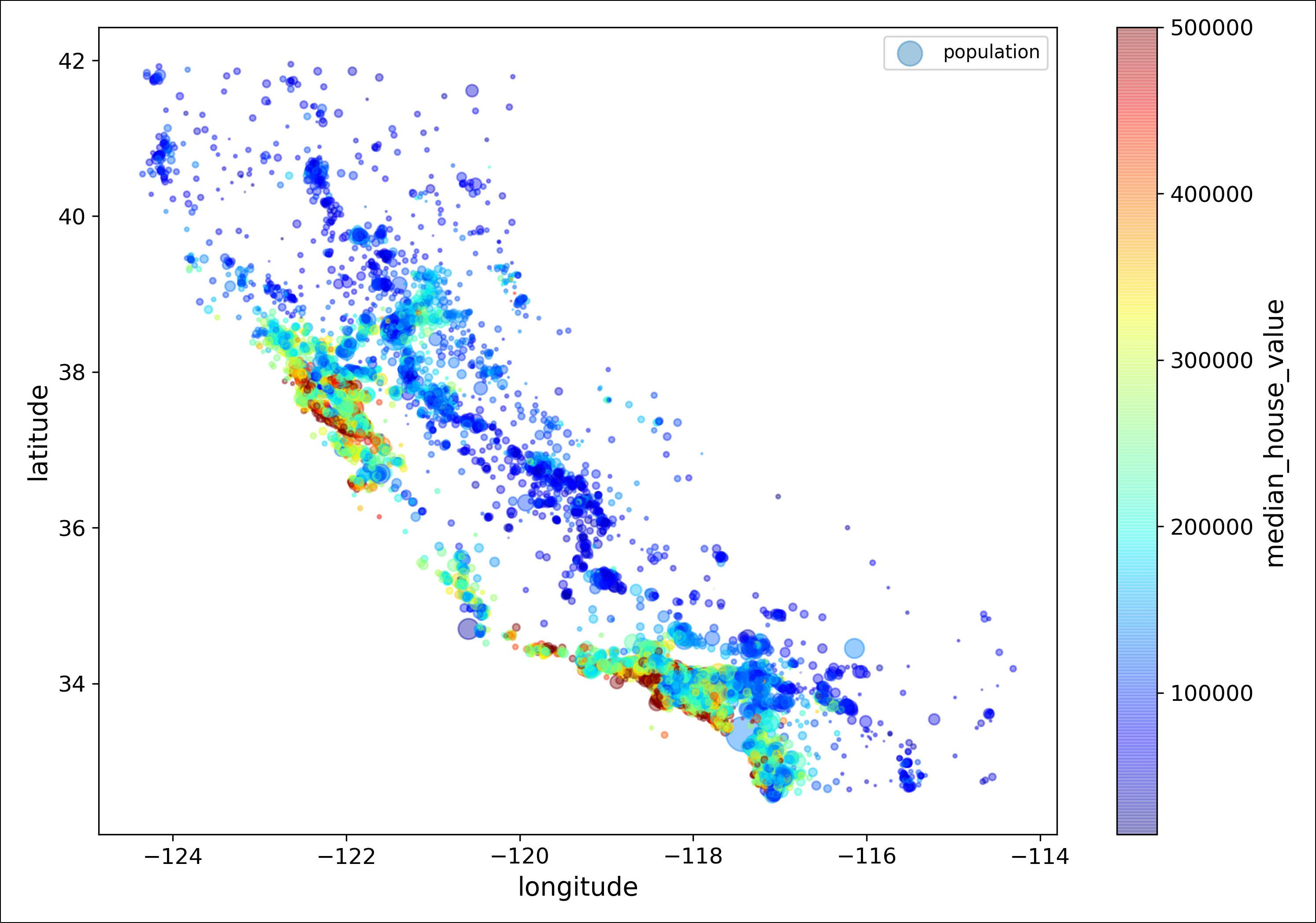
plt.legend()



1. If you are reading this in grayscale, grab a red pen and scribble over most of the coastline from the Bay Area down to San Diego (as you might expect). You can add a patch of yellow around Sacramento as well.



**60** **|** **Chapter 2: End-to-End Machine Learning Project**



*Figure 2-13. California housing prices*



**Discover and Visualize the Data to Gain Insights** **|** **61**

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. It will probably be useful to use a clustering algorithm to detect the main clusters, and add new features that measure the proximity to the cluster centers. The ocean prox‐ imity 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.

**Looking for Correlations**

Since the dataset is not too large, you can easily compute the *standard correlation* *coe‡cient* (also called *Pearson’s r*) between every pair of attributes using thecorr()method:

corr\_matrix = housing.corr()

Now let’s look at how much each attribute correlates with the median house value:

**>>>** corr\_matrix["median\_house\_value"].sort\_values(ascending=False)

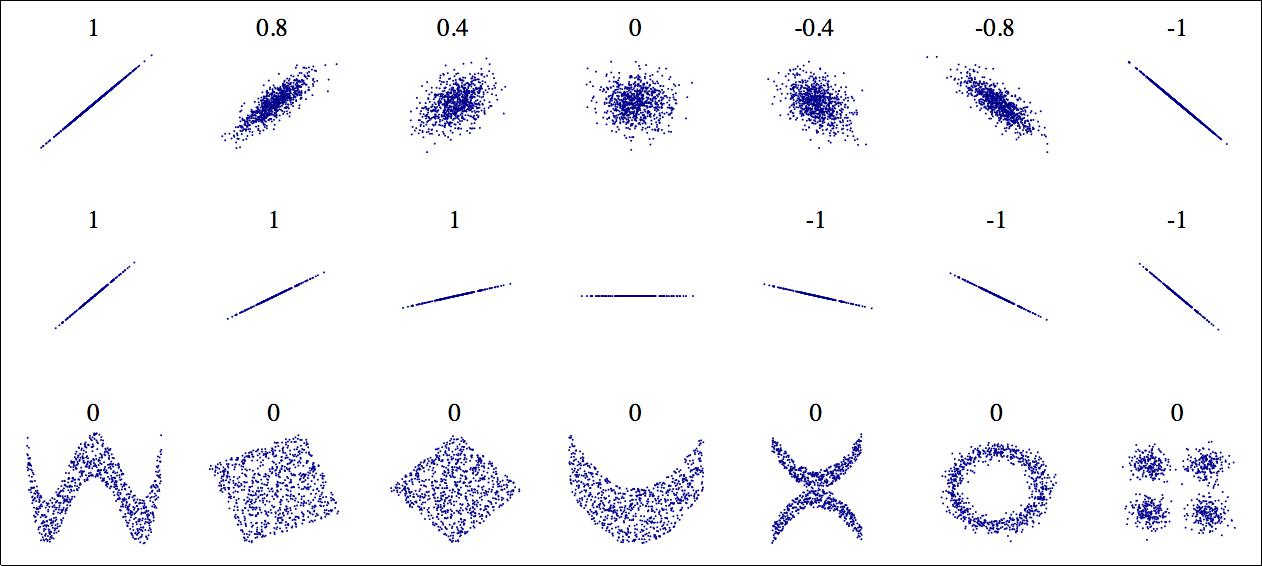
|  |  |
| --- | --- |
| median\_house\_value | 1.000000 |
| median\_income | 0.687170 |
| total\_rooms | 0.135231 |
| housing\_median\_age | 0.114220 |
| households | 0.064702 |
| total\_bedrooms | 0.047865 |
| population | -0.026699 |
| longitude | -0.047279 |
| latitude | -0.142826 |

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 zero mean that there is no linear correlation. [Figure 2-14](#page27) shows various plots along with the correlation coeffi‐ cient between their horizontal and vertical axes.



**62** **|** **Chapter 2: End-to-End Machine Learning Project**



*Figure 2-14. Standard correlation coe‡cient of various datasets (source: Wikipedia; public domain image)*

The correlation coefficient only measures linear correlations (“if *x* goes up, then *y* generally goes up/down”). It may completely miss out on nonlinear relationships (e.g., “if *x* is close to zero then *y* gen‐ erally goes up”). Note how all the plots of the bottom row have a correlation coefficient equal to zero despite the fact that their axes are clearly not independent: these are examples of nonlinear rela‐ tionships. Also, the second row shows examples where the correla‐ tion 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 correla‐ tion coefficient of 1 with your height in feet or in nanometers.



Another way to check for correlation between attributes is to use Pandas’ scatter\_matrix function, which plots every numerical attribute against every other numerical attribute. Since there are now 11 numerical attributes, you would get 112 = 121 plots, which would not fit on a page, so let’s just focus on a few promising attributes that seem most correlated with the median housing value ([Figure 2-15](#page28)):

**from pandas.plotting import** scatter\_matrix

attributes = ["median\_house\_value", "median\_income", "total\_rooms", "housing\_median\_age"]

scatter\_matrix(housing[attributes], figsize=(12, 8))



**Discover and Visualize the Data to Gain Insights** **|** **63**



*Figure 2-15. Scatter matrix*

The main diagonal (top left to bottom right) would be full of straight lines if Pandas plotted each variable against itself, which would not be very useful. So instead Pandas displays a histogram of each attribute (other options are available; see Pandas’ docu‐ mentation for more details).

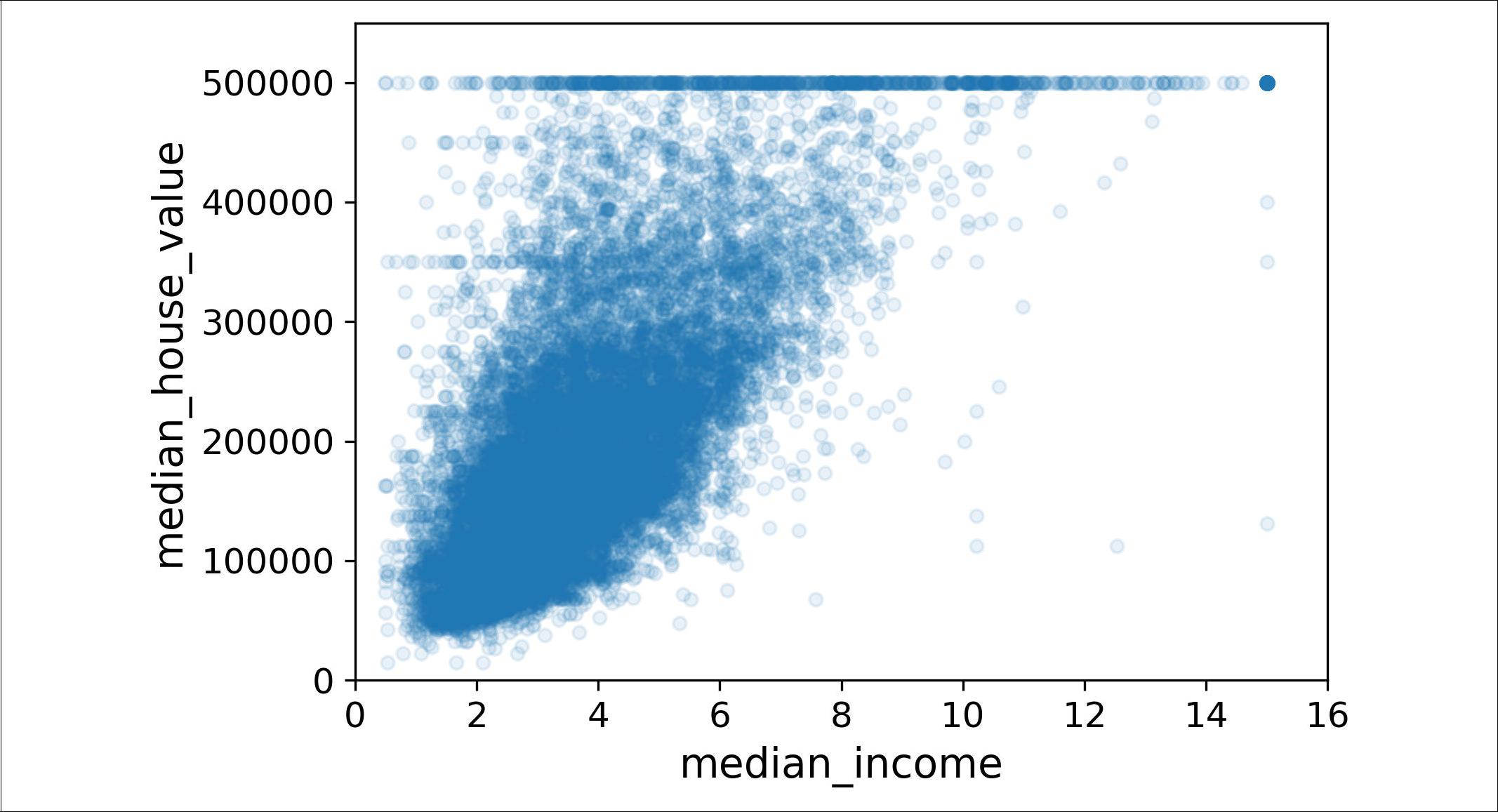
The most promising attribute to predict the median house value is the median income, so let’s zoom in on their correlation scatterplot ([Figure 2-16](#page29)):

housing.plot(kind="scatter", x="median\_income", y="median\_house\_value", alpha=0.1)

This plot reveals a few things. First, the correlation is indeed very strong; you can clearly see the upward trend and the points are not too dispersed. Second, the price cap that we noticed earlier is clearly visible as a horizontal line at $500,000. But this plot 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.



**64** **|** **Chapter 2: End-to-End Machine Learning Project**



*Figure 2-16. Median income versus median house value*

**Experimenting 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 tail-heavy distribution, so you may want to trans‐ form them (e.g., by computing their logarithm). 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 actually 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 number of rooms. And the population per household also 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



**Discover and Visualize the Data to Gain Insights** **|** **65**

|  |  |
| --- | --- |
| 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

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 rea‐ sonably 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.
* 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()



**66** **|** **Chapter 2: End-to-End Machine Learning Project**

**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() | *#* | *option* | *3* |

housing["total\_bedrooms"].fillna(median, inplace=True)

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: SimpleImputer. Here is how to use it. First, you need to create a SimpleImputer instance, specifying that you want to replace each attribute’s missing values with the median of that attribute:

**from sklearn.impute import** SimpleImputer

imputer = SimpleImputer(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)

Now you can fit the imputer instance to the training data using the fit() method:

imputer.fit(housing\_num)

The imputer has simply computed the median of each attribute and stored the result in its statistics\_ instance variable. Only the total\_bedrooms attribute had missing values, but we cannot be sure that there won’t be any missing values in new data after the system goes live, so it is safer to apply the imputer to all the numerical attributes:

**>>>** imputer.statistics\_

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



**Prepare the Data for Machine Learning Algorithms** **|** **67**

**>>>** housing\_num.median().values

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

Now you can use this “trained” imputer to transform the training set by replacing missing values by the learned medians:

X = imputer.transform(housing\_num)

The result is a plain NumPy array containing the transformed features. If you want to put it back into a Pandas DataFrame, it’s simple:

housing\_tr = pd.DataFrame(X, columns=housing\_num.columns)



**Scikit-Learn Design**

Scikit-Learn’s API is remarkably well designed. The [main design principles](https://homl.info/11) are:[17](#page32)

* **Consistency**. All objects share a consistent and simple interface:

— *Estimators*. Any object that can estimate some parameters based on a dataset is called an *estimator* (e.g., an imputer is an estimator). The estimation itself is performed by the fit() method, and it takes only a dataset as a parameter (or two for supervised learning algorithms; the second dataset contains the labels). Any other parameter needed to guide the estimation process is con‐ sidered a hyperparameter (such as an imputer’s strategy), and it must be set as an instance variable (generally via a constructor parameter).

— *Transformers*. Some estimators (such as an imputer) can also transform a dataset; these are called *transformers*. Once again, the API is quite simple: the transformation is performed by the transform() method with the dataset to transform as a parameter. It returns the transformed dataset. This transforma‐ tion generally relies on the learned parameters, as is the case for an imputer. All transformers also have a convenience method called fit\_transform() that is equivalent to calling fit() and then transform() (but sometimes fit\_transform() is optimized and runs much faster).

— *Predictors*. Finally, some estimators are capable of making predictions given a dataset; they are called *predictors*. For example, the LinearRegression model in the previous chapter was a predictor: it predicted life satisfaction given a country’s GDP per capita. A predictor has a predict() method that takes a dataset of new instances and returns a dataset of corresponding predictions. It also has a score() method that measures the quality of the predictions given



1. For more details on the design principles, see “API design for machine learning software: experiences from the scikit-learn project,” L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Müller, et al. (2013).



**68** **|** **Chapter 2: End-to-End Machine Learning Project**

a test set (and the corresponding labels in the case of supervised learning algorithms).[18](#page33)



* **Inspection**. All the estimator’s hyperparameters are accessible directly via publicinstance variables (e.g., imputer.strategy), and all the estimator’s learned parameters are also accessible via public instance variables with an underscore suffix (e.g., imputer.statistics\_).
* **Nonproliferation of classes**. Datasets are represented as NumPy arrays or SciPysparse matrices, instead of homemade classes. Hyperparameters are just regular Python strings or numbers.
* **Composition**. Existing building blocks are reused as much as possible. Forexample, it is easy to create a Pipeline estimator from an arbitrary sequence of transformers followed by a final estimator, as we will see.
* **Sensible defaults**. Scikit-Learn provides reasonable default values for mostparameters, making it easy to create a baseline working system quickly.

**Handling Text and Categorical Attributes**

Earlier we left out the categorical attribute ocean\_proximity because it is a text attribute so we cannot compute its median:

* housing\_cat = housing[["ocean\_proximity"]]
* housing\_cat.head(10)

|  |  |
| --- | --- |
|  | 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 |

Most Machine Learning algorithms prefer to work with numbers anyway, so let’s con‐ vert these categories from text to numbers. For this, we can use Scikit-Learn’s Ordina lEncoder class[19](#page33):

* **from sklearn.preprocessing import** OrdinalEncoder
* ordinal\_encoder = OrdinalEncoder()



1. Some predictors also provide methods to measure the confidence of their predictions.
2. This class is available since Scikit-Learn 0.20. If you use an earlier version, please consider upgrading, or use Pandas’ Series.factorize() method.



**Prepare the Data for Machine Learning Algorithms** **|** **69**

* housing\_cat\_encoded = ordinal\_encoder.fit\_transform(housing\_cat)
* housing\_cat\_encoded[:10]

array([[0.],

[0.],

[4.],

[1.],

[0.],

[1.],

[0.],

[1.],

[0.],

[0.]])

You can get the list of categories using the categories\_ instance variable. It is a list containing a 1D array of categories for each categorical attribute (in this case, a list containing a single array since there is just one categorical attribute):

**>>>** ordinal\_encoder.categories\_

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

One issue with this representation is that ML algorithms will assume that two nearby values are more similar than two distant values. This may be fine in some cases (e.g., for ordered categories such as “bad”, “average”, “good”, “excellent”), but it is obviously not the case for the ocean\_proximity column (for example, categories 0 and 4 are clearly more similar than categories 0 and 1). To fix this issue, a common solution is to create one binary attribute per category: one attribute equal to 1 when the category is “<1H OCEAN” (and 0 otherwise), another attribute equal to 1 when the category is “INLAND” (and 0 otherwise), and so on. This is called *one-hot encoding*, because only one attribute will be equal to 1 (hot), while the others will be 0 (cold). The new attributes are sometimes called *dummy* attributes. Scikit-Learn provides a OneHotEn coder class to convert categorical values into one-hot vectors[20](#page34):

* **from sklearn.preprocessing import** OneHotEncoder
* cat\_encoder = OneHotEncoder()
* housing\_cat\_1hot = cat\_encoder.fit\_transform(housing\_cat)
* housing\_cat\_1hot

<16512x5 sparse matrix of type '<class 'numpy.float64'>' with 16512 stored elements in Compressed Sparse Row format>

Notice that the output is a SciPy *sparse matrix*, instead of a NumPy array. This is very useful when you have categorical attributes with thousands of categories. After one-hot encoding we get a matrix with thousands of columns, and the matrix is full of zeros except for a single 1 per row. Using up tons of memory mostly to store zeros would be very wasteful, so instead a sparse matrix only stores the location of the non‐



1. Before Scikit-Learn 0.20, it could only encode integer categorical values, but since 0.20 it can also handle other types of inputs, including text categorical inputs.



**70** **|** **Chapter 2: End-to-End Machine Learning Project**

zero elements. You can use it mostly like a normal 2D array,[21](#page35) but if you really want to convert it to a (dense) NumPy array, just call the toarray() method:

**>>>** housing\_cat\_1hot.toarray()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.]])

Once again, you can get the list of categories using the encoder’s categories\_ instance variable:

**>>>** cat\_encoder.categories\_

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



If a categorical attribute has a large number of possible categories (e.g., country code, profession, species, etc.), then one-hot encod‐ ing will result in a large number of input features. This may slow down training and degrade performance. If this happens, you may want to replace the categorical input with useful numerical features related to the categories: for example, you could replace the ocean\_proximity feature with the distance to the ocean (similarly, a country code could be replaced with the country’s population and GDP per capita). Alternatively, you could replace each category with a learnable low dimensional vector called an *embedding*. Each category’s representation would be learned during training: this is an example of *representation learning* (see Chapter 13 and ??? for more details).

**Custom Transformers**

Although Scikit-Learn provides many useful transformers, you will need to write your own for tasks such as custom cleanup operations or combining specific attributes. You will want your transformer to work seamlessly with Scikit-Learn func‐ tionalities (such as pipelines), and since Scikit-Learn relies on duck typing (not inher‐ itance), all you need is to create a class and implement three methods: fit() (returning self), transform(), and fit\_transform(). You can get the last one for free by simply adding TransformerMixin as a base class. Also, if you add BaseEstima tor as a base class (and avoid \*args and \*\*kargs in your constructor) you will get two extra methods (get\_params() and set\_params()) that will be useful for auto‐



21 See SciPy’s documentation for more details.



**Prepare the Data for Machine Learning Algorithms** **|** **71**

matic hyperparameter tuning. For example, here is a small transformer class that adds the combined attributes we discussed earlier:

**from sklearn.base import** BaseEstimator,TransformerMixin

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,y=None):

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 this example the transformer has one hyperparameter, add\_bedrooms\_per\_room, set to True by default (it is often helpful to provide sensible defaults). This hyperpara‐ meter will allow you to easily find out whether adding this attribute helps the Machine Learning algorithms or not. More generally, you can add a hyperparameter to gate any data preparation step that you are not 100% sure about. The more you automate these data preparation steps, the more combinations you can automatically try out, making it much more likely that you will find a great combination (and sav‐ ing you a lot of time).

**Feature Scaling**

One of the most important transformations you need to apply to your data is *feature* *scaling*. With few exceptions, Machine Learning algorithms don’t perform well whenthe input numerical attributes have very different scales. This is the case for the hous‐ ing data: the total number of rooms ranges from about 6 to 39,320, while the median incomes only range from 0 to 15. Note that scaling the target values is generally not required.

There are two common ways to get all attributes to have the same scale: *min-max* *scaling* and *standardization*.

Min-max scaling (many people call this *normalization*) is quite simple: values are shifted and rescaled so that they end up ranging from 0 to 1. We do this by subtract‐ ing the min value and dividing by the max minus the min. Scikit-Learn provides a



**72** **|** **Chapter 2: End-to-End Machine Learning Project**

transformer called MinMaxScaler for this. It has a feature\_range hyperparameter that lets you change the range if you don’t want 0–1 for some reason.

Standardization is quite different: first it subtracts the mean value (so standardized values always have a zero mean), and then it divides by the standard deviation so that the resulting distribution has unit variance. Unlike min-max scaling, standardization does not bound values to a specific range, which may be a problem for some algo‐ rithms (e.g., neural networks often expect an input value ranging from 0 to 1). How‐ ever, standardization is much less affected by outliers. For example, suppose a district had a median income equal to 100 (by mistake). Min-max scaling would then crush all the other values from 0–15 down to 0–0.15, whereas standardization would not be much affected. Scikit-Learn provides a transformer called StandardScaler for stand‐ ardization.

As with all the transformations, it is important to fit the scalers to the training data only, not to the full dataset (including the test set). Only then can you use them to transform the training set and the test set (and new data).



**Transformation Pipelines**

As you can see, there are many data transformation steps that need to be executed in the right order. Fortunately, Scikit-Learn provides the Pipeline class to help with such sequences of transformations. Here is a small pipeline for 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)

The Pipeline constructor takes a list of name/estimator pairs defining a sequence of steps. All but the last estimator must be transformers (i.e., they must have a fit\_transform() method). The names can be anything you like (as long as they are unique and don’t contain double underscores “\_\_”): they will come in handy later for hyperparameter tuning.

When you call the pipeline’s fit() method, it calls fit\_transform() sequentially on all transformers, passing the output of each call as the parameter to the next call, until it reaches the final estimator, for which it just calls the fit() method.



**Prepare the Data for Machine Learning Algorithms** **|** **73**

The pipeline exposes the same methods as the final estimator. In this example, the last estimator is a StandardScaler, which is a transformer, so the pipeline has a trans form() method that applies all the transforms to the data in sequence (and of course also a fit\_transform() method, which is the one we used).

So far, we have handled the categorical columns and the numerical columns sepa‐ rately. It would be more convenient to have a single transformer able to handle all col‐ umns, applying the appropriate transformations to each column. In version 0.20, Scikit-Learn introduced the ColumnTransformer for this purpose, and the good news is that it works great with Pandas DataFrames. Let’s use it to apply all the transforma‐ tions to the housing data:

**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)

Here is how this works: first we import the ColumnTransformer class, next we get the list of numerical column names and the list of categorical column names, and we construct a ColumnTransformer. The constructor requires a list of tuples, where each tuple contains a name[22](#page38), a transformer and a list of names (or indices) of columns that the transformer should be applied to. In this example, we specify that the numer‐ ical columns should be transformed using the num\_pipeline that we defined earlier, and the categorical columns should be transformed using a OneHotEncoder. Finally, we apply this ColumnTransformer to the housing data: it applies each transformer to the appropriate columns and concatenates the outputs along the second axis (the transformers must return the same number of rows).

Note that the OneHotEncoder returns a sparse matrix, while the num\_pipeline returns a dense matrix. When there is such a mix of sparse and dense matrices, the Colum nTransformer estimates the density of the final matrix (i.e., the ratio of non-zero cells), and it returns a sparse matrix if the density is lower than a given threshold (by default, sparse\_threshold=0.3). In this example, it returns a dense matrix. And that’s it! We have a preprocessing pipeline that takes the full housing data and applies the appropriate transformations to each column.



22 Just like for pipelines, the name can be anything as long as it does not contain double underscores.



**74** **|** **Chapter 2: End-to-End Machine Learning Project**

Instead of a transformer, you can specify the string "drop" if you want the columns to be dropped. Or you can specify "pass through" if you want the columns to be left untouched. By default, the remaining columns (i.e., the ones that were not listed) will be dropped, but you can set the remainder hyperparameter to any transformer (or to "passthrough") if you want these columns to be handled differently.

If you are using Scikit-Learn 0.19 or earlier, you can use a third-party library such as sklearn-pandas, or roll out your own custom transformer to get the same function‐ ality as the ColumnTransformer. Alternatively, you can use the FeatureUnion class which can also apply different transformers and concatenate their outputs, but you cannot specify different columns for each transformer, they all apply to the whole data. It is possible to work around this limitation using a custom transformer for col‐ umn selection (see the Jupyter notebook for an example).

**Select and Train a Model**

At last! You framed the problem, you got the data and explored it, you sampled a training set and a test set, and you wrote transformation pipelines to clean up and prepare your data for Machine Learning algorithms automatically. You are now ready to select and train a Machine Learning model.

**Training and Evaluating on the Training Set**

The good news is that thanks to all these previous steps, things are now going to be much simpler than you might think. Let’s first train a Linear Regression model, like we did in the previous chapter:

**from sklearn.linear\_model import** LinearRegression

lin\_reg = LinearRegression() lin\_reg.fit(housing\_prepared, housing\_labels)

Done! You now have a working Linear Regression model. Let’s try it out on a few instances from the training set:

* 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: [ 210644.6045 317768.8069 210956.4333 59218.9888 189747.5584]

**>>> print**("Labels:",list(some\_labels))

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



**Select and Train a Model** **|** **75**

It works, although the predictions are not exactly accurate (e.g., the first prediction is off by close to 40%!). Let’s measure this regression model’s RMSE on the whole train‐ ing set using Scikit-Learn’s mean\_squared\_error function:

* **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

68628.19819848922

Okay, this is better than nothing but clearly not a great score: most districts’ median\_housing\_values range between $120,000 and $265,000, so a typical predic‐ tion error of $68,628 is not very satisfying. This is an example of a model underfitting the training data. When this happens it can mean that the features do not provide enough information to make good predictions, or that the model is not powerful enough. As we saw in the previous chapter, the main ways to fix underfitting are to select a more powerful model, to feed the training algorithm with better features, or to reduce the constraints on the model. This model is not regularized, so this rules out the last option. You could try to add more features (e.g., the log of the popula‐ tion), but first let’s try a more complex model to see how it does.

Let’s train a DecisionTreeRegressor. This is a powerful model, capable of finding complex nonlinear relationships in the data (Decision Trees are presented in more detail in Chapter 6). The code should look familiar by now:

**from sklearn.tree import** DecisionTreeRegressor

tree\_reg = DecisionTreeRegressor() tree\_reg.fit(housing\_prepared, housing\_labels)

Now that the model is trained, let’s evaluate it on the training set:

* housing\_predictions = tree\_reg.predict(housing\_prepared)
* tree\_mse = mean\_squared\_error(housing\_labels, housing\_predictions)
* tree\_rmse = np.sqrt(tree\_mse)
* tree\_rmse

0.0

Wait, what!? No error at all? Could this model really be absolutely perfect? Of course, it is much more likely that the model has badly overfit the data. How can you be sure? As we saw earlier, you don’t want to touch the test set until you are ready to launch a model you are confident about, so you need to use part of the training set for train‐ ing, and part for model validation.

**Better Evaluation Using Cross-Validation**

One way to evaluate the Decision Tree model would be to use the train\_test\_split function to split the training set into a smaller training set and a validation set, then



**76** **|** **Chapter 2: End-to-End Machine Learning Project**

train your models against the smaller training set and evaluate them against the vali‐ dation set. It’s a bit of work, but nothing too difficult and it would work fairly well.

A great alternative is to use Scikit-Learn’s *K-fold cross-validation* feature. The follow‐ ing code randomly splits the training set into 10 distinct subsets called *folds*, then it trains and evaluates the Decision Tree model 10 times, picking a different fold for evaluation every time and training on the other 9 folds. The result is an array con‐ taining the 10 evaluation scores:

**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)

Scikit-Learn’s cross-validation features expect a utility function (greater is better) rather than a cost function (lower is better), so the scoring function is actually the opposite of the MSE (i.e., a neg‐ ative value), which is why the preceding code computes -scores before calculating the square root.



Let’s look at the results:

* **def** display\_scores(scores):

**...print**("Scores:",scores)

**...print**("Mean:",scores.mean())

**...print**("Standard deviation:",scores.std())

**...**

* display\_scores(tree\_rmse\_scores)

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

Mean: 71407.68766037929

Standard deviation: 2439.4345041191004

Now the Decision Tree doesn’t look as good as it did earlier. In fact, it seems to per‐ form worse than the Linear Regression model! Notice that cross-validation allows you to get not only an estimate of the performance of your model, but also a measure of how precise this estimate is (i.e., its standard deviation). The Decision Tree has a score of approximately 71,407, generally ±2,439. You would not have this information if you just used one validation set. But cross-validation comes at the cost of training the model several times, so it is not always possible.

Let’s compute the same scores for the Linear Regression model just to be sure:

**>>>** 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)



**Select and Train a Model** **|** **77**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Scores: [66782.73843989 66960.118071 | | | 70347.95244419 74739.57052552 | |
| 68031.13388938 | 71193.84183426 | 64969.63056405 | | 68281.61137997 |
| 71552.91566558 | 67665.10082067] |  |  |  |

Mean: 69052.46136345083

Standard deviation: 2731.674001798348

That’s right: the Decision Tree model is overfitting so badly that it performs worse than the Linear Regression model.

Let’s try one last model now: the RandomForestRegressor. As we will see in Chap‐ ter 7, Random Forests work by training many Decision Trees on random subsets of the features, then averaging out their predictions. Building a model on top of many other models is called *Ensemble Learning*, and it is often a great way to push ML algo‐ rithms even further. We will skip most of the code since it is essentially the same as for the other models:

* **from sklearn.ensemble import** RandomForestRegressor
* forest\_reg = RandomForestRegressor()
* forest\_reg.fit(housing\_prepared, housing\_labels)
* [...]
* forest\_rmse

18603.515021376355

**>>>** display\_scores(forest\_rmse\_scores)

Scores: [49519.80364233 47461.9115823 50029.02762854 52325.28068953 49308.39426421 53446.37892622 48634.8036574 47585.73832311 53490.10699751 50021.5852922 ]

Mean: 50182.303100336096

Standard deviation: 2097.0810550985693

Wow, this is much better: Random Forests look very promising. However, note that the score on the training set is still much lower than on the validation sets, meaning that the model is still overfitting the training set. Possible solutions for overfitting are to simplify the model, constrain it (i.e., regularize it), or get a lot more training data. However, before you dive much deeper in Random Forests, you should try out many other models from various categories of Machine Learning algorithms (several Sup‐ port Vector Machines with different kernels, possibly a neural network, etc.), without spending too much time tweaking the hyperparameters. The goal is to shortlist a few (two to five) promising models.



**78** **|** **Chapter 2: End-to-End Machine Learning Project**

You should save every model you experiment with, so you can come back easily to any model you want. Make sure you save both the hyperparameters and the trained parameters, as well as the cross-validation scores and perhaps the actual predictions as well. This will allow you to easily compare scores across model types, and compare the types of errors they make. You can easily save Scikit-Learn models by using Python’s pickle module, or using sklearn.externals.joblib, which is more efficient at serializing large NumPy arrays:

**from sklearn.externals import** joblib

joblib.dump(my\_model, "my\_model.pkl")

*# and later...*

my\_model\_loaded = joblib.load("my\_model.pkl")

**Fine-Tune Your Model**

Let’s assume that you now have a shortlist of promising models. You now need to fine-tune them. Let’s look at a few ways you can do that.

**Grid Search**

One way to do that would be to fiddle with the hyperparameters manually, until you find a great combination of hyperparameter values. This would be very tedious work, and you may not have time to explore many combinations.

Instead you should get Scikit-Learn’s GridSearchCV to search for you. All you need to do is tell it which hyperparameters you want it to experiment with, and what values to try out, and it will evaluate all the possible combinations of hyperparameter values, using cross-validation. For example, the following code searches for the best combi‐ nation of hyperparameter values for the RandomForestRegressor:

**from sklearn.model\_selection import** GridSearchCV

param\_grid = [

{'n\_estimators': [3, 10, 30], 'max\_features': [2, 4, 6, 8]},

{'bootstrap': [False], 'n\_estimators': [3, 10], 'max\_features': [2, 3, 4]},

]

forest\_reg = RandomForestRegressor()

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)



**Fine-Tune Your Model** **|** **79**

When you have no idea what value a hyperparameter should have, a simple approach is to try out consecutive powers of 10 (or a smaller number if you want a more fine-grained search, as shown in this example with the n\_estimators hyperparameter).

This param\_grid tells Scikit-Learn to first evaluate all 3 × 4 = 12 combinations of n\_estimators and max\_features hyperparameter values specified in the first dict (don’t worry about what these hyperparameters mean for now; they will be explained in Chapter 7), then try all 2 × 3 = 6 combinations of hyperparameter values in the second dict, but this time with the bootstrap hyperparameter set to False instead of True (which is the default value for this hyperparameter).

All in all, the grid search will explore 12 + 6 = 18 combinations of RandomForestRe gressor hyperparameter values, and it will train each model five times (since we are using five-fold cross validation). In other words, all in all, there will be 18 × 5 = 90 rounds of training! It may take quite a long time, but when it is done you can get the best combination of parameters like this:

**>>>** grid\_search.best\_params\_{'max\_features': 8, 'n\_estimators': 30}



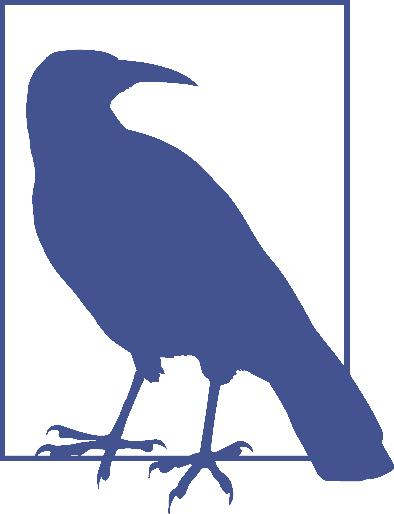
Since 8 and 30 are the maximum values that were evaluated, you should probably try searching again with higher values, since the score may continue to improve.

You can also get the best estimator directly:

**>>>** grid\_search.best\_estimator\_

RandomForestRegressor(bootstrap=True, criterion='mse', max\_depth=None, max\_features=8, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_impurity\_split=None, min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=30, n\_jobs=None, oob\_score=False, random\_state=None, verbose=0, warm\_start=False)

If GridSearchCV is initialized with refit=True (which is the default), then once it finds the best estimator using cross-validation, it retrains it on the whole training set. This is usually a good idea since feeding it more data will likely improve its perfor‐ mance.



And of course the evaluation scores are also available:

* cvres = grid\_search.cv\_results\_
* **for** mean\_score,params **in** zip(cvres["mean\_test\_score"],cvres["params"]):



**80** **|** **Chapter 2: End-to-End Machine Learning Project**

**...** **print**(np.sqrt(-mean\_score),params)

**...**

63669.05791727153 {'max\_features': 2, 'n\_estimators': 3}

55627.16171305252 {'max\_features': 2, 'n\_estimators': 10}

53384.57867637289 {'max\_features': 2, 'n\_estimators': 30}

60965.99185930139 {'max\_features': 4, 'n\_estimators': 3}

52740.98248528835 {'max\_features': 4, 'n\_estimators': 10}

50377.344409590376 {'max\_features': 4, 'n\_estimators': 30}

58663.84733372485 {'max\_features': 6, 'n\_estimators': 3}

52006.15355973719 {'max\_features': 6, 'n\_estimators': 10}

50146.465964159885 {'max\_features': 6, 'n\_estimators': 30}

57869.25504027614 {'max\_features': 8, 'n\_estimators': 3}

51711.09443660957 {'max\_features': 8, 'n\_estimators': 10}

49682.25345942335 {'max\_features': 8, 'n\_estimators': 30}

62895.088889905004 {'bootstrap': False, 'max\_features': 2, 'n\_estimators': 3} 54658.14484390074 {'bootstrap': False, 'max\_features': 2, 'n\_estimators': 10} 59470.399594730654 {'bootstrap': False, 'max\_features': 3, 'n\_estimators': 3} 52725.01091081235 {'bootstrap': False, 'max\_features': 3, 'n\_estimators': 10} 57490.612956065226 {'bootstrap': False, 'max\_features': 4, 'n\_estimators': 3} 51009.51445842374 {'bootstrap': False, 'max\_features': 4, 'n\_estimators': 10}

In this example, we obtain the best solution by setting the max\_features hyperpara‐ meter to 8, and the n\_estimators hyperparameter to 30. The RMSE score for this combination is 49,682, which is slightly better than the score you got earlier using the default hyperparameter values (which was 50,182). Congratulations, you have suc‐ cessfully fine-tuned your best model!



Don’t forget that you can treat some of the data preparation steps as hyperparameters. For example, the grid search will automatically find out whether or not to add a feature you were not sure about (e.g., using the add\_bedrooms\_per\_room hyperparameter of your CombinedAttributesAdder transformer). It may similarly be used to automatically find the best way to handle outliers, missing fea‐ tures, feature selection, and more.

**Randomized Search**

The grid search approach is fine when you are exploring relatively few combinations, like in the previous example, but when the hyperparameter *search space* is large, it is often preferable to use RandomizedSearchCV instead. This class can be used in much the same way as the GridSearchCV class, but instead of trying out all possible combi‐ nations, it evaluates a given number of random combinations by selecting a random value for each hyperparameter at every iteration. This approach has two main bene‐ fits:



**Fine-Tune Your Model** **|** **81**

* If you let the randomized search run for, say, 1,000 iterations, this approach will explore 1,000 different values for each hyperparameter (instead of just a few val‐ ues per hyperparameter with the grid search approach).
* You have more control over the computing budget you want to allocate to hyper‐ parameter search, simply by setting the number of iterations.

**Ensemble Methods**

Another way to fine-tune your system is to try to combine the models that perform best. The group (or “ensemble”) will often perform better than the best individual model (just like Random Forests perform better than the individual Decision Trees they rely on), especially if the individual models make very different types of errors. We will cover this topic in more detail in Chapter 7.

**Analyze the Best Models and Their Errors**

You will often gain good insights on the problem by inspecting the best models. For example, the RandomForestRegressor can indicate the relative importance of each attribute for making accurate predictions:

* feature\_importances = grid\_search.best\_estimator\_.feature\_importances\_
* feature\_importances

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])

Let’s display these importance scores next to their corresponding attribute names:

* extra\_attribs = ["rooms\_per\_hhold", "pop\_per\_hhold", "bedrooms\_per\_room"]
* cat\_encoder = full\_pipeline.named\_transformers\_["cat"]
* cat\_one\_hot\_attribs = list(cat\_encoder.categories\_[0])
* attributes = num\_attribs + extra\_attribs + cat\_one\_hot\_attribs
* sorted(zip(feature\_importances, attributes), reverse=True) [(0.3661589806181342, 'median\_income'),

(0.1647809935615905, 'INLAND'), (0.10879295677551573, 'pop\_per\_hhold'), (0.07334423551601242, 'longitude'), (0.0629090704826203, 'latitude'), (0.05641917918195401, 'rooms\_per\_hhold'), (0.05335107734767581, 'bedrooms\_per\_room'), (0.041143798478729635, 'housing\_median\_age'), (0.014874280890402767, 'population'), (0.014672685420543237, 'total\_rooms'), (0.014257599323407807, 'households'), (0.014106483453584102, 'total\_bedrooms'), (0.010311488326303787, '<1H OCEAN'), (0.002856474637320158, 'NEAR OCEAN'),



**82** **|** **Chapter 2: End-to-End Machine Learning Project**

(0.00196041559947807, 'NEAR BAY'), (6.028038672736599e-05, 'ISLAND')]

With this information, you may want to try dropping some of the less useful features (e.g., apparently only one ocean\_proximity category is really useful, so you could try dropping the others).

You should also look at the specific errors that your system makes, then try to under‐ stand why it makes them and what could fix the problem (adding extra features or, on the contrary, getting rid of uninformative ones, cleaning up outliers, etc.).

**Evaluate Your System on the Test Set**

After tweaking your models for a while, you eventually have a system that performs sufficiently well. Now is the time to evaluate the final model on the test set. There is nothing special about this process; just get the predictors and the labels from your test set, run your full\_pipeline to transform the data (call transform(), *not* fit\_transform(), you do not want to fit the test set!), and evaluate the final model on the test set:

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) | *# =>* | *evaluates to 47,730.2* |

In some cases, such a point estimate of the generalization error will not be quite enough to convince you to launch: what if it is just 0.1% better than the model cur‐ rently in production? You might want to have an idea of how precise this estimate is. For this, you can compute a 95% *confidence interval* for the generalization error using

scipy.stats.t.interval():

* **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)))

**...**

array([45685.10470776, 49691.25001878])

The performance will usually be slightly worse than what you measured using cross-validation if you did a lot of hyperparameter tuning (because your system ends up fine-tuned to perform well on the validation data, and will likely not perform as well



**Fine-Tune Your Model** **|** **83**

on unknown datasets). It is not the case in this example, but when this happens you must resist the temptation to tweak the hyperparameters to make the numbers look good on the test set; the improvements would be unlikely to generalize to new data.

Now comes the project prelaunch phase: you need to present your solution (high‐ lighting what you have learned, what worked and what did not, what assumptions were made, and what your system’s limitations are), document everything, and create nice presentations with clear visualizations and easy-to-remember statements (e.g., “the median income is the number one predictor of housing prices”). In this Califor‐ nia housing example, the final performance of the system is not better than the experts’, but it may still be a good idea to launch it, especially if this frees up some time for the experts so they can work on more interesting and productive tasks.

**Launch, Monitor, and Maintain Your System**

Perfect, you got approval to launch! You need to get your solution ready for produc‐ tion, in particular by plugging the production input data sources into your system and writing tests.

You also need to write monitoring code to check your system’s live performance at regular intervals and trigger alerts when it drops. This is important to catch not only sudden breakage, but also performance degradation. This is quite common because models tend to “rot” as data evolves over time, unless the models are regularly trained on fresh data.

Evaluating your system’s performance will require sampling the system’s predictions and evaluating them. This will generally require a human analysis. These analysts may be field experts, or workers on a crowdsourcing platform (such as Amazon Mechanical Turk or CrowdFlower). Either way, you need to plug the human evalua‐ tion pipeline into your system.

You should also make sure you evaluate the system’s input data quality. Sometimes performance will degrade slightly because of a poor quality signal (e.g., a malfunc‐ tioning sensor sending random values, or another team’s output becoming stale), but it may take a while before your system’s performance degrades enough to trigger an alert. If you monitor your system’s inputs, you may catch this earlier. Monitoring the inputs is particularly important for online learning systems.

Finally, you will generally want to train your models on a regular basis using fresh data. You should automate this process as much as possible. If you don’t, you are very likely to refresh your model only every six months (at best), and your system’s perfor‐ mance may fluctuate severely over time. If your system is an online learning system, you should make sure you save snapshots of its state at regular intervals so you can easily roll back to a previously working state.



**84** **|** **Chapter 2: End-to-End Machine Learning Project**

**Try It Out!**

Hopefully this chapter gave you a good idea of what a Machine Learning project looks like, and showed you some of the tools you can use to train a great system. As you can see, much of the work is in the data preparation step, building monitoring tools, setting up human evaluation pipelines, and automating regular model training. The Machine Learning algorithms are also important, of course, but it is probably preferable to be comfortable with the overall process and know three or four algo‐ rithms well rather than to spend all your time exploring advanced algorithms and not enough time on the overall process.

So, if you have not already done so, now is a good time to pick up a laptop, select a dataset that you are interested in, and try to go through the whole process from A to Z. A good place to start is on a competition website such as [*http://kaggle.com/*](http://kaggle.com/): you will have a dataset to play with, a clear goal, and people to share the experience with.

**Exercises**

Using this chapter’s housing dataset:

1. Try a Support Vector Machine regressor (sklearn.svm.SVR), with various hyper‐ parameters such as kernel="linear" (with various values for the C hyperpara‐ meter) or kernel="rbf" (with various values for the C and gamma hyperparameters). Don’t worry about what these hyperparameters mean for now. How does the best SVR predictor perform?
2. Try replacing GridSearchCV with RandomizedSearchCV.
3. Try adding a transformer in the preparation pipeline to select only the most important attributes.
4. Try creating a single pipeline that does the full data preparation plus the final prediction.
5. Automatically explore some preparation options using GridSearchCV.

Solutions to these exercises are available in the online Jupyter notebooks at [*https://*](https://github.com/ageron/handson-ml2) [*github.com/ageron/handson-ml2*](https://github.com/ageron/handson-ml2).



**Try It Out!** **|** **85**