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

Check the Assumptions

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.

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!

Get the Data

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 notebook is available at *https://github.com/ageron/handson-ml*.

Create the Workspace

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/*.[7]

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):

```
$ 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 43. 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

7 The latest version of Python 3 is recommended. Python 2.7+ should work fine too, but it is deprecated.

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).[8] You can check to see if pip is installed by typing the following command:

```
$ pip3 --version
pip 9.0.1 from [...]/lib/python3.5/site-packages (python 3.5)
```

You should make sure you have a recent version of pip installed, at the very least >1.4 to support binary module installation (a.k.a. wheels). To upgrade the pip module, type:[9]

```
$ pip3 install --upgrade pip
Collecting pip
[...]
Successfully installed pip-9.0.1
```

Creating an Isolated Environment

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 by running the following pip command:

```
$ pip3 install --user --upgrade virtualenv
Collecting virtualenv
[...]
Successfully installed virtualenv
```

Now you can create an isolated Python environment by typing:

```
$ cd $ML_PATH
$ virtualenv env
Using base prefix '[...]'
New python executable in [...]/ml/env/bin/python3.5
Also creating executable in [...]/ml/env/bin/python
Installing setuptools, pip, wheel...done.
```

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

```
$ cd $ML_PATH
$ source env/bin/activate
```

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 site packages, you should create the environment

8 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.

9 You may need to have administrator rights to run this command; if so, try prefixing it with `sudo`.

using virtualenv's `--system-site-packages` option). Check out virtualenv's documentation for more information.

Now you can install all the required modules and their dependencies using this simple pip command:

```
$ pip3 install --upgrade 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:

```
$ 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:

```
$ 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).

Now create a new Python notebook by clicking on the New button and selecting the appropriate Python version[10] (see Figure 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.

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

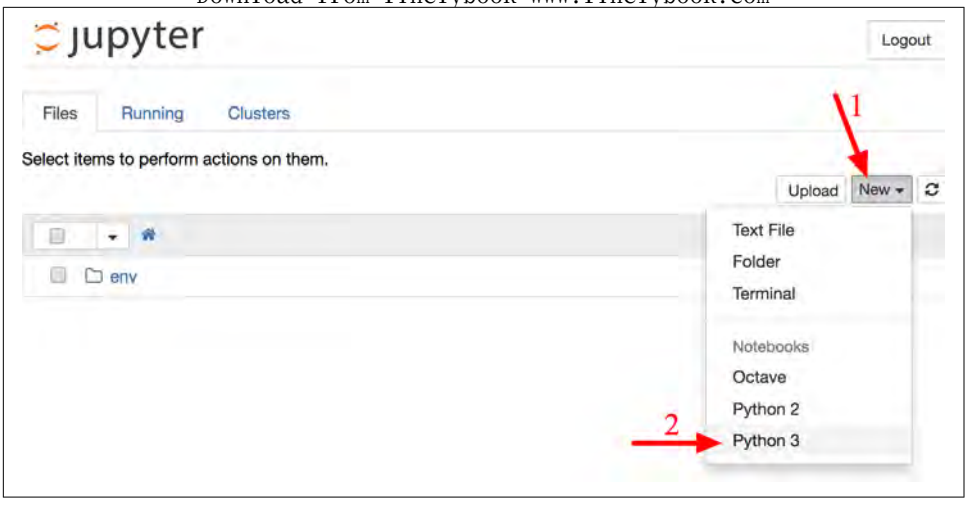

Figure 2-3. Your workspace in Jupyter

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) 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.

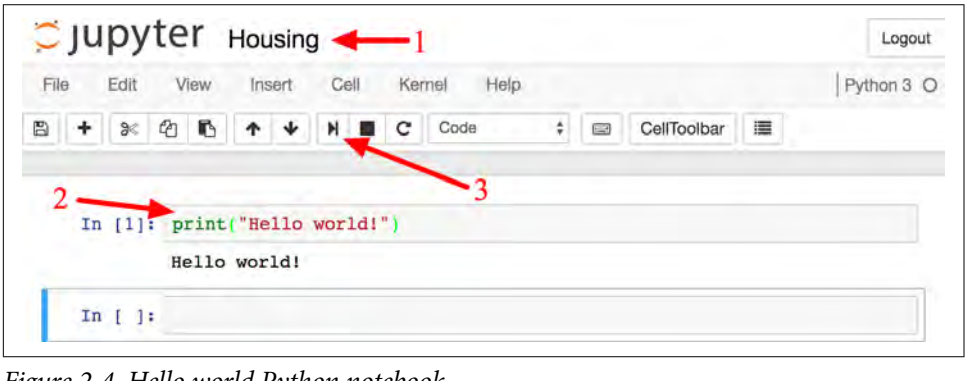

Figure 2-4. Hello world Python notebook

Download the Data

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