# Table of Contents

1 What you will learn ........................................... 3
2 Who can take it .................................................. 5
What you will learn

This textbook will guide you through an investigation of money in politics using data from the California Civic Data Coalition.

You will learn just enough of the Python computer programming language to work with the pandas library, a popular open-source tool for analyzing data. The course will teach you how to read, filter, join, group, aggregate and rank structured data.

You will also learn how to record, remix and republish your analysis using the Jupyter notebook, a browser-based application for writing code that is emerging as the standard for sharing reproducible research.

You will also learn how to plot and chart your data inside the notebook using the Altair data visualization library, which is on the cutting-edge of developing a simple, structured grammar for generating graphics.
This course is free. If you’ve tried Python once or twice, have good attitude and know how to take a few code crashes in stride, you are qualified. But before you begin be sure to review the prerequisites.

2.1 Prologue: Prerequisites

Your computer needs the following computer-programming tools to participate. Verify you have them working before you begin.

2.1.1 A command-line interface

Whether you know about it or not, there should be a way to open a window and directly issue commands to your operating system. Different operating systems give this tool slightly different names, but they all have some form of it.

On Windows this is called the “command prompt.” On MacOS it is called the “terminal.”

On Windows 10, we recommend you install the Windows Subsystem for Linux and select the Ubuntu distribution from the Windows Store. This will give you access to a generic open-source terminal without all the complications and quirks introduced by Windows. On MacOS, the standard terminal app will work fine.

2.1.2 Python 3.6 or higher

Python is a free and open-source computer programming language. It’s one of the most popular in the world and praised by its supporters as clear and easy to read.

That makes it ideal for beginners and is partly why it’s been adopted by professionals in many fields, ranging from engineering and web development to journalism and music.

You can check if Python is already installed on your computer by visiting your command line and entering the following:
Python --version

You should see something like this after you hit enter:

Python 3.6.10

If not, you’ll need to install Python on your system.
If you see a number starting with 2, like say …

Python 2.7.12

… then you have an outdated version of Python and will need to upgrade to a version starting with a three. You can probably complete the class without doing so, but the maintainers of Python are gradually phasing out version two and officially recommend you upgrade.

Instructions for both new installations and upgrades can be found here.

2.2 Chapter 1: Hello pipenv

Before we can start programming, we need to do a little housekeeping on our computer. It’s not required, but every organized Python project should have a system for managing two highly technical, but very important, issues. They are:

1. How to install and manage your programming tools
2. How to keep your code from conflicting with other projects

We will solve these problems with Pipenv. It handles both of the issues outlined above, hence the tool’s two-part name, which is a programming portmanteau.

2.2.1 The “pip” package manager

Whatever the aim of your project, you likely will rely on one or more Python packages that extend the language’s standard library. This allows you to import modules written by other trusty Python developers into your own code so that you can focus on the work that matters to you. The JupyterLab development environment, pandas analysis kit and Altair chart library covered in this class are all examples.

These third-party packages are available — for free — via the Python Package Index, where they are published largely by volunteers. To download and install them on your computer, you need a tool called a package manager.

Python’s default package manager is pip. It allows you to retrieve and unpack PyPi packages from your terminal. It goes something like this:

pip install jupyterlab

With pip, you can also document the exact version of each of package your project requires and store in a list that records everything necessary to run your code.

Typically these dependencies are specified in a requirements.txt file. This document makes it easier to sync your project’s requirements across multiple machines if, for instance, you are collaborating with other developers.

Lucky for us, all the functionality of pip is included in Pipenv, as well as much more.
2.2.2 The “env” environment manager

By default, Python’s third-party packages are all installed in a shared “global” folder somewhere in the depths of your computer. By default, every Python project on your computer draws from this same set of installed programs.

This approach is fine for your first experiments with Python, but it quickly falls apart when you start to get serious about coding.

For instance, say you develop a web application today with Flask version 1.1. What if, a year from now, you want to start a new project and use a newer version of Flask? Your old app is still live and requires occasional patches, but you don’t have time to re-write all of your old to make it compatible with the latest version of Flask.

Open-source projects are changing every day and such conflicts are common, especially when you factor in the sub-dependencies of your project’s direct dependencies, as well as the sub-dependencies of those sub-dependencies.

Programmers solve this problem by creating a virtual environment for each project that isolates them into discrete, independent containers that do not rely on code in the global environment.

Strictly speaking, working within a virtual environment is not required. At first, it might even feel like a hassle. But in the long run, you will be glad you did it. And you don’t have to take my word for it, you can read discussions on StackOverflow and Reddit.

Good thing Pipenv can do this too.

2.2.3 Installing Pipenv

Pipenv and its prerequisites are installed via your computer’s command-line interface. You can verify its there by typing the following into your terminal:

```
pipenv --version
```

If you have it installed, you should see the terminal respond with the version on your machine.

```
ipenv, version 2018.11.26
```

If you get an error, you will need to install it.

If you are on a Mac, Pipenv’s maintainers recommend installing via Homebrew:

```
brew install pipenv
```

If you are on Windows 10 and using the Windows Subsystem for Linux, you can install Linuxbrew and use it to install Pipenv.

If neither option makes sense for you, Pipenv’s docs recommend a user install via pip:

```
pip install --user pipenv
```

Whatever installation route you choose, you can confirm your success by testing for its version again:

```
pipenv --version
```

If you see that version number now, you know you’re okay.

2.2.4 Create a code directory to store all your work

Now let’s create a common folder where all you of your projects will be stored starting with this one. This is also where our virtualenv will be configured.
Open your command-line interface, which will start you off in your home directory. Enter the following command and press enter to see all of the folders there now.

```
ls
```

Next use the `mkdir` to create a new directory for your code. In the same style as the Desktop, Documents and Downloads folders included by most operating system, we will name this folder Code.

```
mkdir Code
```

To verify that worked, you can open in your file explorer and navigate to your home folder.

### 2.2.5 Create a project directory

Now let’s make a folder for your work in this class.

```
mkdir Code/first-python-notebook
```

Next use your terminal to navigate into the new directory with the `cd` command:

```
cd Code/first-python-notebook
```

### 2.2.6 Install your first package

Now let’s install a simple Python package to see Pipenv in action. We’ll choose yolk3k, a simple command-line tool that can list all your installed python packages.

We can add it to our project’s private virtual environment by typing its name after Pipenv’s install command.

```
pipenv install yolk3k
```

When you invoke Pipenv’s `install` command, it checks for an existing virtual environment connected to your project’s directory. Finding none, it creates one, then installs yolk3k into it.

As a result, two files are added to your project directory: Pipfile and Pipfile.lock. These are Pipenv’s alternative to the requirements.txt file mentioned earlier.

Open these files in a text editor (such as Sublime Text, Atom or Visual Studio Code), and you’ll see how they describe your project’s Python requirements.

In the Pipfile, you’ll see the name and exact version of any package we directed Pipenv to install. So far, we’ve only installed yolk3k, and we didn’t specify an exact version, so you’ll see:

```
[packages]
yolk3k = "*"
```

Pipfile.lock has a more complicated, nested structure that specifies the exact version of your project’s direct dependencies along with all their sub-dependencies.

Now that yolk is installed, we can execute it inside our environment using Pipenv’s run command. Let’s use its simple command for listing all of our currently installed tools.

```
pipenv run yolk -l
```

You should see the computer spit out everything you have installed. You’ll notice that yolk3k is on the list. You’ve completed the setup process for First Python Notebook. Now the real fun begins.
2.3 Chapter 2: Hello notebook

A Jupyter notebook is a browser application where you can write, run, remix and republish code. It is free software you can install and run like any other open-source library. It is used by scientists, scholars, investors and corporations to create and share their research.

It is also used by journalists to develop stories and show their work. Examples include:

- “The Tennis Racket” by BuzzFeed and the BBC
- “Machine bias” by ProPublica
- More than 30 different notebooks published by the Los Angeles Times

2.3.1 Navigate into your project directory

For starters, let’s check where we are in our computer’s file system. For this we’ll use a command called `pwd`, which stands for present working directory.

```
pwd
```

The output is the full path of your location in the file system, something like `/Users/palewire/Code/first-python-notebook`. If you aren’t currently in the project directory we created in chapter 1, you need to change directories.

First, jump into the code directory:

```
cd Code
```

Then, jump into project directory:

```
cd first-python-notebook
```

This is where you’ll store a local copy of all the code and files you create for this project.

**Note:** It isn’t necessary to change directories one level at a time. You can also specify the full path of directory you want to change into:

```
cd Code/first-python-notebook
```

2.3.2 Install JupyterLab

Now we will return to Pipenv and use it to install JupyterLab, the web-based interactive development environment for Jupyter notebooks, code and data.

```
pipenv install jupyterlab
```

2.3.3 Create your first notebook

Now we can use pipenv’s run command to start JupyterLab from your terminal.
pipenv run jupyter lab

That will open up a new tab in your default web browser that looks something like this:

Click the “Python 3” button in the middle panel and create a new Python 3 notebook.

### 2.3.4 Write Python in the notebook

Now you are all setup and ready to start writing Python code.

Do not stress. There is nothing too fancy about it. You can start by just doing a little simple math.

Type the following into the first box, then hit the play button in the toolbar above the notebook (or hit SHIFT+ENTER on your keyboard).

```
2+2
```

<table>
<thead>
<tr>
<th>In [1]:</th>
<th>2+2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out[1]:</td>
<td>4</td>
</tr>
</tbody>
</table>

There. You have just written your first Python code. You have entered two integers and added them together using the plus sign operator.
Not so bad, right?

**Note:** If you get an error after you run a cell, look carefully at your code and see that it exactly matches what’s been written in the example. Don’t worry.

Code crashes are a normal part of life for computer programmers. They’re usually caused by small typos that can be quickly corrected.

This to-and-fro of writing Python code in a notebook cell and then running it with the play button is the rhythm of working in a notebook. Over time you will gradually stack cells to organize an analysis that runs from top to bottom.

The cells can contain variables, functions and other Python tools.

A simple example would be storing your number in a variable in one cell . . .

```
number = 2
```

. . . then adding it to another number in the next.

```
number + 3
```

Run those two cells in succession and the notebook should output the number five. Change the number value to 3 and run both cells again and it should output six.

**Note:** If you’ve never written Python before, we recommend An Informal Introduction to Python and subsequent sections of python.org’s tutorial.

Once you’ve got the hang of making the notebook run, you’re ready to introduce pandas, the powerful Python analysis library that can do a whole lot more than add a few numbers together.

### 2.4 Chapter 3: Hello pandas

Lucky for us, Python is filled with functions to do pretty much anything you’d ever want to do with a programming language: navigate the web, parse data, interact with a database, run fancy statistics, build a pretty website and so much more.

Creative people have put these tools to work to get a wide range of things done in the academy, the laboratory and even in outer space.

Some of those tools are included in a toolbox that comes with the language, known as the standard library. Others have been built by members of Python’s developer community and need to be downloaded and installed from the web.

One that’s important for this class is called pandas. It is a tool invented at a financial investment firm that has become a leading open-source library for accessing and analyzing data in many different fields.

#### 2.4.1 Install pandas with pipenv

We’ll install pandas the same way we installed the JupyterLab earlier: Our new friend pipenv.

Let’s pick up where we left off at the end of chapter 2. Save your notebook by clicking the disk icon or selecting “save and checkpoint” from the file menu. Then switch to your command prompt and hit CTRL-C.

That will kill your notebook and return you to the command line. There we’ll install pandas:
pipenv install pandas

### 2.4.2 Import pandas

Now let’s restart our notebook and get back to work.

```bash
pipenv run jupyter lab
```

Reopen your notebook and create a new cell at the top of your Jupyter notebook. There we will import the pandas library for use in our script. Type in the following and hit the play button again.

```python
import pandas
```

If nothing happens, that’s good. It means you have pandas installed and ready.

**Note:** If you get an error message, return to the prerequisites section make sure you have everything installed properly.

If you do and it still doesn’t work, copy and paste the tail end of your error message into Google. Among the results there will almost certainly be others working through the same problem.

Return to the cell with the import and rewrite it like this.

```python
import pandas as pd
```

This will import the pandas library at the shorter variable name of pd. This is not required but it is standard practice in the pandas community and you will frequently see examples of pandas code online using it as shorthand. It’s not required, but it’s good to get in the habit so that your code will be understood by other computer programmers.

### 2.4.3 Conduct a simple data analysis

Those two little letters contain dozens of data analysis tools that we’ll use in future lessons.

They can import massive data files, compute advanced statistics, filter, sort, rank and just about anything else you’d want to do.

We’ll get to that soon, but let’s start out with something simple.

First let’s make a list of numbers in a new notebook cell. To keep things simple, I am going to enter all of the even numbers between zero and ten and press play.

```python
my_list = [2, 4, 6, 8]
```

If you’re a skilled Python programmer, you can do some cool stuff with any list. But hand it over to pandas instead, and you can analyze it without knowing much computer code at all.

In this case, it’s as simple as converting that plain Python list into what pandas calls a Series. Make it happen in your next cell.

```python
my_series = pd.Series(my_list)
```

Once the data becomes a Series, you can immediately run a wide range of descriptive statistics. Let’s try a few.

First, let’s sum all the numbers. Make a new cell and run this. It should spit out the total.
my_series.sum()

Then find the maximum value in the next.
my_series.max()

The minimum value in the next.
my_series.min()

How about the average (also known as the mean)? Keep adding cells and calculating new statistics.
my_series.mean()

The median?
my_series.median()

The standard deviation?
my_series.std()

And all of the above, plus a little more about the distribution, in one simple command.
my_series.describe()

With those simple techniques, we’re only scratching the surface of what pandas makes possible.

Substitute in a series of 10 million records at the top of the stack (or even just the odd numbers between zero and ten), and your notebook would calculate all those statistics again without you having to write any more code.

Once your data, however large or complex, is imported into pandas, there’s little limit to what you can do to filter, merge, group, aggregate, compute or chart using simple methods like the ones above.

In the next chapter we’ll get started doing just using data tracking the flow of money in California politics.

### 2.5 Chapter 4: Hello money in politics

In November 2016, California voters had a lot of decisions to make.

Millions of votes were cast across the state to choose who should be America’s next president, to send more than 50 members off to Congress, to select a new U.S. senator and to refill most of the seats in the Sacramento statehouse.

On top of all that, every ballot in the state included a list of propositions, yes or no questions that give voters the power to directly change the law.

They vary every election, and this time 17 different proposals were put to voters.

Should the state take out $9 billion in bonds to fund education? Should the cost of prescription drugs be limited? Should the cigarette tax be increased? Should recreational marijuana use be legalized? Should actors in pornographic films be required to wear condoms? Should the state stop administering the death penalty? Or should it instead speed up administering the death penalty?

### 2.5. Chapter 4: Hello money in politics
And that’s just the start. The election guide the state sends to every registered voter set a new record for length at 224 pages long.

### 2.5.1 Meet CAL-ACCESS

By election day, nearly $500 million dollars was spent by political campaigns that sought to influence voters to vote yes or no on those 17 propositions.

We know that because every dollar that is raised or spent by political campaigns in the state of California has to be disclosed. That’s thanks to the Political Reform Act of 1974, which was itself enacted by voters via a proposition.

Groups that support or oppose propositions, known as “committees,” must register with the secretary of state and file periodic reports. Those reports list the names, occupations and employers of donors, as well as how campaign funds are spent.

Those reports are stored in a public database maintained by the government. Almost every state has one like it. In California, the database is called CAL-ACCESS.

The CAL-ACCESS website offers tools to inspect recent filings, and a bulk download where you can access the raw data.

Unfortunately, the downloadable files are so jumbled, dirty and difficult that they are rarely used.

Even the secretary of state himself, Alex Padilla, has condemned CAL-ACCESS as a Frankenstein monster of code.

### 2.5.2 Meet the California Civic Data Coalition

In 2014, a team of journalists, academics and developers formed to solve the problem. They call themselves the California Civic Data Coalition.

Their project, which is still in development, aims to create an open-source pipeline that converts the raw data published by CAL-ACCESS into refined data files that a beginner, like yourself, can easily pick up and analyze.

The coalition’s effort has drawn hundreds of contributions from developers and journalists at dozens of news organizations and was named a winner of the Knight News Challenge.

Experimental versions of the coalition’s data files enabled the Los Angeles Times to calculate the $500 million figure quoted earlier in this chapter.
It has also powered interactive graphics and several other investigations into the role of money in state politics.

You can review, install and contribute to the coalition’s open-source codebase on GitHub.

Currently, the coalition’s website archives the data published each day by the state and offers more complete documentation and easier access to the original files.

In the near future, the site will offer simplified files free to the public that make the data significantly easier to understand and interrogate. Until the coalition reaches that goal we will be working with experimental early versions created for this class.

In the next chapter, we will show how to import that data into pandas and your notebook to start an analysis.

### 2.6 Chapter 5: Hello data

Now it’s time to get our hands on some real data.

Our data source will be the California Civic Data Coalition, an open-source network of journalists and developers that maintains an archive of data tracking money in California politics.

The coalition has created simplified data files containing the disclosure forms that committees campaigning either for against one of the 17 propositions on the ballot in November 2016 filed with the state of California.

They are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>committees.csv</td>
<td>Committees active in the election linked to propositions supported or opposed</td>
</tr>
<tr>
<td>contributions.csv</td>
<td>Donors reported by each of the committees</td>
</tr>
</tbody>
</table>

The data are structured in rows of comma-separated values. This is known as a CSV file. It is the most common way you will find data published online.

#### 2.6.1 Creating a DataFrame

The pandas library is able to read in files from a variety formats, including CSV.

If it’s not currently running start up your Jupyter Notebook as described in chapter two.

Scroll down to the first open cell. There we will import the first CSV file listed above using the `read_csv` function included with pandas.

```python
pd.read_csv("https://first-python-notebook.readthedocs.io/_static/committees.csv")
```

**Warning:** You will need to precisely type in the URL to the file. Feel free to copy and paste it from the example above into your notebook.
After you run the cell, you should see a big table printed below.

It is a DataFrame where pandas has structured the CSV data into rows and columns, just like Excel or other spreadsheet software might.

The advantage here is that rather than manipulating the data through a haphazard series of clicks and keystrokes, we will be gradually grinding down the data using a computer programming script that is 100% transparent and reproducible.

### 2.6.2 Creating a variable

In order to do that, we need to store our DataFrame so it can be reused in subsequent cells. We can do this by saving in a "variable", which is a fancy computer programming word for a named shortcut where we save our work as we go.

Go back to your initial cell and change it to this. Then rerun it.

```python
committee_list = pd.read_csv("https://first-python-notebook.readthedocs.io/_static/committees.csv")
```

After you run it, you shouldn’t see anything. That’s a good thing. It means our DataFrame has been saved under the name props, which we can now begin interacting with in the cells that follow.

We can do this by calling “methods” that pandas has made available to all DataFrames.

You may not have known it at the time, but read_csv was one of these methods. There are dozens more that can do all sorts of interesting things. Let’s start with some easy ones that analysts use all the time.

### 2.6.3 Using the head method

First, to preview the first few rows of the dataset, try the `head` method. Hit the + button in the toolbar to add a new cell below the first one. Type this in it and hit the run button again.

```python
committee_list.head()
```

### 2.6.4 Using the info method

To get a look at all of the columns and what type of data they store, add another cell and try `info`.

```python
committee_list.info()
```

Look carefully at those results and you see we have more than 100 links between committees and propositions.

### 2.6.5 Creating another DataFrame

With that we’re ready to move on to a related, similar task: Importing all of the individual contributions reported to last year’s 17 ballot measures.

We’ll start by using the read_csv method to import the second CSV file linked above. Save it as a new variable just as we did before. Let’s call this one contribs.

```python
contrib_list = pd.read_csv("https://first-python-notebook.readthedocs.io/_static/contributions.csv")
```

Just as we did earlier, you can inspect the contents of this new file with the head method.
You should also inspect the columns using the info method. Running these two tricks whenever you open a new file is a good habit to develop so that you can carefully examine the data you’re about to work with.

Now that you’ve got some data imported, we’re ready to begin our analysis.

## 2.7 Chapter 6: Hello columns

In this chapter we’ll begin our analysis by learning how to inspect a column from a DataFrame.

### 2.7.1 Accessing a column

We’ll begin with the `prop_name` column where the proposition each committee sought to influence is stored. To see the contents of a column separate from the rest of the DataFrame, add the column’s name to the DataFrame’s variable following a period.

That will list the column out as a Series, just like the ones we created from scratch in chapter three. And, just as we did then, you can now start tacking on additional methods that will analyze the contents of the column. In this case, the column is filled with characters. So we don’t want to calculate statistics like the median and average, as we did before.

**Note:** You can also access columns a second way, like this:

This method isn’t as pretty, but it’s required if your column has a space in its name, which would break the simpler dot-based method.

### 2.7.2 Counting a column’s values

There’s another built-in pandas tool that will total up the frequency of values in a column. In this case that could be used to answer the question: Which proposition had the most committees? The method is called `value_counts` and it’s just as easy to use as sum, min or max. All you need to do is add a period after the column name and chain it on the tail end of your cell.

Run the code and you should see the lengthy proposition names ranked by their number of committees.
2.7.3 Resetting a DataFrame

You may have noticed that even though the result has two columns, pandas did not return a clean-looking table in the same way as head did for our DataFrame.

That’s because our column, a Series, acts a little bit different than the DataFrame created by read_csv.

In most instances, if you have an ugly Series generated by a method like value_counts and you want to convert it into a pretty DataFrame you can do so by tacking on the reset_index method onto the tail end.

```python
committee_list.prop_name.value_counts().reset_index()
```

Why do Series and DataFrames behave differently? Why does reset_index have such a weird name?

Like so much in computer programming, the answer is simply “because the people who created the library said so.”

That’s not worth stressing about in this case, but it’s important to learn that all open-source programming tools have their quirks. Over time you’ll learn pandas has more than a few.

As a beginner, you should just accept the oddities and roll with it. As you get more advanced, if there’s something about the system you think could be improved you should consider contributing to the Python code that operates the library you’d like to improve.

2.8 Chapter 7: Hello filters

Until November 2016, the use and sale of marijuana for recreational purposes was illegal in California. That changed when voters approved Proposition 64, which asked if the practice ought to be legalized.

A yes vote supported legalization. A no vote opposed it. In the final tally, 57% of voters said yes.

Our next mission is to use the DataFrames containing campaign committees and contributors to figure out the biggest donors both for and against the measure.

To do that, the first thing we need to do is isolate the fundraising committees active on Proposition 64, which are now buried among of the list of more than 100 groups active last November.

2.8.1 Filtering a DataFrame

The most common way to filter a DataFrame is to pass an expression as an “index” that can be used to decide which records should be kept and which discarded.

You write the expression by combining a column on your DataFrame with an “operator” like == or > or < and a value to compare the column against.

Note: If you are familiar with writing SQL to manipulate databases, pandas’ filtering system is somewhat similar to a WHERE query. The official pandas documentation offers direct translations between the two.

In our case, the column we want to filter against is prop_name. We only want to keep those records where the value there matches the full name of Proposition 64.

Where do we get that? Our friend value counts.

Running the command we learned before to list and count all of the proposition names will spit out the full name of all 17 measures.
From that result we can copy the full name of the proposition and place it between quotation marks in a variable in a new cell. This will allow us to reuse it later.

```python
my_prop = 'PROPOSITION 064- MARIJUANA LEGALIZATION. INITIATIVE STATUTE.'
```

In the next cell we will ask pandas to narrow down our list of committees to just those that match the proposition we’re interested in. We will create a filter expression that looks like this: `committee_list.prop_name == my_prop`, and place it between two flat brackets following the variable we want to filter. Place the following code in the next open cell in your notebook.

```python
committee_list[committee_list.prop_name == my_prop]
```

Run it and it outputs the filtered dataset, just those committees active on Proposition 64.

Now we should save the results of that filter into new variable separate from the full list we imported from the CSV file.

Since it includes only the committees for one proposition lets call it the singular prop.

```python
my_committees = committee_list[committee_list.prop_name == my_prop]
```

To check our work find out how many committees are left after the filter, let’s run the DataFrame inspection commands we learned earlier.

First head.

```python
my_committees.head()
```

Then info.

```python
my_committees.info()
```

### 2.9 Chapter 8: Hello merge

Our next job is to filter down the contributions list, which includes all disclosed contributions to all proposition campaigns, to just those linked to Proposition 64.

We could try to do this with a filter, as we did before with the committees.

But look carefully at the columns listed in the contribution file’s info output.

```python
contrib_list.info()
```

Now compare that to the committees file.

```python
committee_list.info()
```

You will notice that this file contains a field called `calaccess_committee_id` that is identical to the one found in the committee CSV.

That’s because these two files are drawn from a “relational database” that stores data in an array of tables linked together by common identifiers. In this case, the unique identifying codes of committees in one table can be expected to match those found in another.

We can therefore safely join the two files using the pandas `merge` method.
Note: Again, if you are familiar with traditional databases, you may recognize that the merge method in pandas is similar to SQL's `JOIN` statement. If you dig into `merge`'s documentation you will see it has many of the same options, such as the ability to conduct “inner” and “outer” joins.

### 2.9.1 Merging DataFrames

By default the merge method in pandas will return only those rows where a common identifier found in both tables, which is known as an “inner” join.

That means that if we merge the full contributions file to our filtered list of Proposition 64 committees, only the contributions to the Proposition 64 committees will remain. The result will be equivalent to a filter.

That’s exactly what we want. So let’s try it.

Merging two DataFrames is as simple as passing both to pandas built-in merge method and specifying which field we’d like to use to connect them together. We will save the result into another new variable.

```python
merged = pd.merge(my_committees, contrib_list, on="calaccess_committee_id")
```

That new DataFrame variable can be inspected like any other.

```python
merged.head()
```

By looking at the columns you can check how many rows survived the merged.

```python
merged.info()
```

You can also see that the DataFrame now contains all of the columns in both tables. Columns with the same name have had a suffix automatically appended to indicate whether they came from the first or second DataFrame submitted to the merge.

We have now created a new dataset that includes only contributions supporting and opposing Proposition 64. We’re ready to move on from preparing our data. It’s time to interview it.

### 2.10 Chapter 9: Hello totals

In some ways, your database is no different from a human source. Getting a good story requires careful, thorough questioning. In this section we will move ahead by conducting an interview with pandas to pursue our quest of finding out the biggest donors to Proposition 64.

Using tricks we learned as far back as chapter three, we can start off by answering a simple question: What is the total sum of Proposition 64 contributions that have been reported?

#### 2.10.1 Summing a column

To answer that let’s start by getting our hands on `amount`, the column from the contributions DataFrame with the numbers in it. We can do that just as we did with other columns earlier.

```python
merged.amount
```

Now we can add up the column’s total using the pandas method `sum`, just as we did when we were first getting started with pandas.
And printed out below your cell, there’s our answer.
We’ve completed our first piece of analysis and discovered the total amount spent on this proposition.
Time to run off to Twitter and publish our results to the world, right?
Wrong.

2.10.2 How not to be wrong

The total we generated is not the overall total raised in the campaign, and it is guaranteed to be lower than the totals reported in the media and by the campaigns.

Why?
In California, campaigns are only required to disclose the names of donors who give over $100, so our data is missing all of the donors who gave less than that amount.
The cutoff varies, and there are some exceptions, but the same thing is true in other states and also at the federal level in races for Congress and the White House.
The overall totals are instead reported on cover sheets included with disclosure reports that lump together all the smaller contributions as part of a grand total. Those are the records most commonly cited to total up a campaign’s fundraising.
The result is that an itemized list of contributions, like the one we have, cannot be used to calculate a grand total. That’s true in California and virtually anywhere else you work with campaign data. Overlooking that limitation is a rookie mistake routinely made by analysts new to this field.
But that doesn’t mean our data is worthless. We just have to use it responsibly. In many cases, professional campaign reporters will refer to an analysis drawn from a list like ours as applying only to “large donors.”
Since large donors typically account for most of the money, the results are still significant. And the high level of detail included in each record — like the donor’s name, employer and occupation — makes the limitations worth working through.

2.10.3 Which side got more large donations?

Adding up a big total is all well and good. But we’re aiming for something more nuanced.
We want to separate the money spent supporting the proposition from the money opposing it. Then we want to find out who raised more.
To answer that question, let’s return to the filtering technique we learned in chapter seven.
First let’s look at the column we’re going to filter by, committee_position.

Now let’s filter our merged table down using that column and the pandas filtering method that combines a column, an operator and the value we want to filter by. Let’s stick the result in a variable.

Now let’s repeat all that for opposing contributions. First the filter into a new variable.
Now sum up the total disclosed contributions to each for comparison. First the opposition.

```python
oppose.amount.sum()
```

Then the supporters.

```python
support.amount.sum()
```

The support is clearly larger. But what percent is it of the overall disclosed total? We can find out by combining two sum calculations using the division operator.

```python
support.amount.sum() / merged.amount.sum()
```

2.11 Chapter 10: Hello sorting

Another simple but common technique for analyzing data is sorting.

What were the ten biggest contributions? We can find the answer by using the `sort_values` method to rearrange our list using the amount field.

```python
merged.sort_values("amount")
```

Note that returns the DataFrame resorted in ascending order from lowest to highest. That is pandas default way of sorting.

To answer our question you’ll need to reverse that, so that values are sorted in descending order from biggest to smallest. It’s a little tricky at first, but here’s how to do it with `sort_values`.

```python
merged.sort_values("amount", ascending=False)
```

You can limit the result to the top five by chaining the head method at the end.

```python
merged.sort_values("amount", ascending=False).head()
```

We can now use the new variable to rank the five biggest supporting contributions by using `sort_values` again.

```python
support.sort_values("amount", ascending=False).head()
```

And now how about the opposition.

```python
oppose.sort_values("amount", ascending=False).head()
```

2.12 Chapter 11: Hello groupby

To take the next step towards ranking the top contributors, we’ll need to learn a new trick. It’s called `groupby`.

It’s a pandas method that allows you to group a DataFrame by a column and then calculate a sum, or any other statistic, for each unique value. This is necessary when you want to rack up statistics on a long list of values, or about a combination of fields.
2.12.1 Grouping by one field

As we’ve been digging through the data, I’m sure a few questions have popped into mind. One interesting field in the contributions list is the home state of the contributor. A natural question follows: How much of the money came from outside of California?

If you scroll back up and look carefully as the info command we ran after merging out data, you will noticed it includes a column named `contributor_state`.

That’s the field we want to group with here. Here’s how you do it.

```python
merged.groupby("contributor_state")
```

A nice start. But you’ll notice you don’t get much back. The data’s been grouped by state, but we haven’t chosen what to do with it yet. We want totals by state, so we can sum the `amount` field the same way we did earlier for the entire DataFrame.

```python
merged.groupby("contributor_state") .amount.sum()
```

Again our data has come back as an ugly Series. To reformat it as a pretty DataFrame use the `reset_index` method again.

```python
merged.groupby("contributor_state") .amount.sum().reset_index()
```

Sorting the biggest totals to the top is as easy as appending the `sort_values` trick we already know to the end. And voila there’s our answer.

```python
merged.groupby("contributor_state") .amount.sum().reset_index().sort_values("amount", ascending=False)
```

2.12.2 Grouping by multiple fields

Finding the top contributors is almost as easy, but since the first and last names are spread between two fields we’ll need to submit them to groupby as a list. Copy the last line above, and replace “contributor_state” with a list like the one here:

```python
merged.groupby(["contributor_firstname", "contributor_lastname"])["amount"].sum() .reset_index() .sort_values("amount", ascending=False)
```

Note: You should be noticing that several of the top contributors appear to be the same person with their name entered in slightly different ways. This is another important lesson of campaign contributions data. Virtually none of the data is standardized by the campaigns or the government. The onus is on the analyst to show caution and responsibly combine records where the name fields refer to the same person.

To find out if each contributor supported or opposed the measure, you simple add that field to our groupby method.

```python
merged.groupby(["contributor_firstname", "contributor_lastname", "committee_position"])["amount"].sum() .reset_index() .sort_values("amount", ascending=False)
```

If you wanted just the top supporters or opponents alone, you could run those same commands with the support and oppose datasets we filtered down to earlier. Everything else about the commands would be the same as the first one above.

For the supporters:
2.12.3 How not to be wrong

You’ve done it. Our brief interview is complete and you’ve answered the big question that started our inquiry. Or so you think! Look again at our rankings above. Now compare them against the ranking we looked at earlier in our sorting lesson.

Study it closely and you’ll see an important difference. All of the contributors without a first name are dropped from our groupby lists. And some of them gave a lot of money.

This is happening because if another pandas quirk. Empty fields are read in by pandas as null values, which is a mathematical representation of nothing. In pandas a null is called a NaN, an abbreviation for “not a number” commonly used in computer programming.

And, guess what, pandas’ groupby method will drop any rows with nulls in the grouping fields. So all those records without a first name were silently excluded from our analysis. Yikes!

Whatever our opinion of pandas’ default behavior, it’s something we need to account for, and a reminder that we should never assume we know what computer programming tools are doing under the hood. As with human sources, everything you code tells you should be viewed skeptically and verified.

The solution to this problem is easy. We need to replace those NaN first names with empty strings, which pandas won’t drop. We can do that by using pandas’ fillna method ahead of the group.

Now we’ve finally got a ranking we can work with. Congratulations, you’ve finished our analysis.

2.12.4 Extra credit

If you’re interested in continuing the interview, see if you can answer a few more questions on your own. Here are some ideas:

- What are the top employers of donors for and against the measure?
- Which committees had the fewest donors?
- What was the average size of donations both for and against?

2.13 Chapter 12: Hello remix

Now here’s where things get fun. Your entire analysis is scripted top to bottom, which means it can be rerun and reproduced. It also be remixed.

Remember this line earlier?
my_prop = 'PROPOSITION 064- MARIJUANA LEGALIZATION. INITIATIVE STATUTE.'

That’s where we set which proposition we wanted to filter on. It was a key fork in the road, which shaped all the analysis that followed.

That means that if we substituted a different proposition name from the value_counts list just above it we could rerun our notebook and conduct an identical analysis of another proposition, without writing another line of code.

Let’s try it. I picked the death penalty ban that was on the same ballot and changed that cell of code to this:

my_prop = 'PROPOSITION 062- DEATH PENALTY. INITIATIVE STATUTE.'

Now I go to the Run menu at the top of the notebook and selected “Run All Cells.” Wait a few seconds and, boom, you’ll have a whole new of donors plotted out.

## 2.14 Chapter 13: Hello charts

Python has a number of charting tools that can work hand-in-hand with pandas. Altair is a relative newbie, but it’s got good documentation and can display charts right in your Jupyter Notebook — plus it can export to lots of other formats.

Let’s take it for a spin.

Before we start, we need to make sure it is installed. Head back to your terminal and practice that pipenv install process.

$ pipenv install altair

After that completes, once again restart your notebook.

$ pipenv run jupyter notebook

Now you can head back to your notebook and add Altair to your imports. We’ll usually import it with the alias alt so we don’t have to type out the whole thing every time we make a chart.

```python
import altair as alt
```

Now rerun the entire notebook, as we learned above. You will need to do this when you halt and restart your notebook on the command line. Reminder, you can do this by pulling down the Cell menu at the top of the notebook and selecting the Run all option.

Let’s pick up where we last left off in the previous chapter. If we want to chart out how much the top supporters of the proposition spent, we first need to select them from the dataset. Using the grouping and sorting tricks we learned earlier, the top 10 can returned like this:

```python
top_supporters = support.fillna('').groupby(['contributor_firstname', 'contributor_lastname']).amount.sum().reset_index().sort_values('amount', ascending=False).head(10)
```

We can then view them with a trick you may remember by now.

```python
top_supporters.head(10)
```
Now that we have `altair` imported, we can pop that dataframe into a quick chart. Let’s step through the building blocks of a chart.

```python
alt.Chart(top_supporters).mark_bar().encode(
    x="contributor_lastname",
    y="amount"
)
```

Look at that chart!
Here’s an idea — maybe we want to do horizontal, not vertical bars. How would you rewrite this chart code to reverse those bars?

```python
alt.Chart(top_supporters).mark_bar().encode(
    x="amount",
    y="contributor_lastname"
)
```

What if we wanted to focus on the top five records? We can use that `head` command we already know.

```python
alt.Chart(top_supporters.head(5)).mark_bar().encode(
    x="amount",
    y="contributor_lastname"
)
```

Okay, but what if I want to combine the first and last name? We have the data we need in two separate columns, which we can put together simply by inventing a new field on our data frame and, just like a variable, setting it equal to a combination of the other fields.

```python
top_supporters['contributor_fullname'] = top_supporters.contributor_firstname + " " + top_supporters.contributor_lastname
```

Now we can use that column instead of ‘`contributor_lastname`’ in our chart.
Notice how the sort order changed when we changed the contributor column? This chart is sorted alphabetically by y-axis value, and it’s making everything look pretty sloppy and hard to parse. Let’s fix that.

We want to sort the y-axis values by their corresponding x values. We’ve been using the shorthand syntax to pass in our axis columns so far, but to add more customization to our chart we’ll have to switch to the longform way of defining the y axis.

That will look something like the way we define the chart in the first place: `alt.Y(column_name, arg="value")`. There are lots of options that you might want to pass in, like ones that will sum your data on the fly or define the number range you want your axis to display. In this case, we’ll just be using the `sort` command.

And we can’t have a chart without context. Let’s throw in a title for good measure.
Yay, we made a chart!

Now, we have a good idea of who spent the most in support of Prop. 64. What if we wanted to see who spent money on both sides?

Add a new cell and a new dataframe, `top_contributors`, summing up the top contributors in our whole merged dataframe. We're going to repeat a lot of the pandas functions we've stepped through before, all in one go this time.

```python
import pandas as pd

merged = pd.read_csv('merged_data.csv')

# Select only the top contributors for each position
top_contributors = merged.fillna('').groupby(['contributor_firstname', 'contributor_lastname', 'committee_position']).amount.sum().reset_index().sort_values('amount', ascending=False).head(10)
```

And once again, we're going to want a `contributor_fullname` column that combines our first and last name columns.

```python
top_contributors['contributor_fullname'] = top_contributors['contributor_firstname'] + ' ' + top_contributors['contributor_lastname']
```

Now pop `top_contributors` into a chart, just like we did before. Remember that sort function!

```python
alt.Chart(top_contributors.head(5)).mark_bar().encode(
    x='amount',
    y=alt.Y('contributor_fullname', sort='-x'),
).
```

What facet of the data is this chart not showing? How might we add additional context?

We have that `committee_position` column in our dataframe now. Let's try an altair option that we haven't used yet: color. Can you guess where we should add that in?

```python
alt.Chart(top_contributors.head(5)).mark_bar().encode(
    x='amount',
    y=alt.Y('contributor_fullname', sort='-x'),
    color='committee_position'
).
```
Hey now! That wasn’t too hard, was it?
To be fair, none of these charts are ready to pop into a news story quite yet. There are lots of additional formatting and design options that you can start digging into in the Altair docs — you can even create Altair themes to specify default color schemes and fonts.

But you may not want to do all that tweaking in code, especially if you’re just working on a one-off graphic. If you wanted to hand this chart off to a graphics department, all you’d have to do is head to the top right corner of your chart.

See those three dots? Click on that, and you’ll see lots of options. Downloading the file as an SVG will let anyone with graphics software like Adobe Illustrator take this file and tweak the design.

Want to recreate this chart in a tool like Chartbuilder or Datawrapper? In that case, you’ll want to export this data into a spreadsheet.

Guess what? It’s this easy.

```python
top_supporters.head(5).to_csv("top_supporters.csv")
```

### 2.15 About the authors

Ben Welsh is the editor of the Los Angeles Times Data and Graphics Department, a team of reporters and computer programmers in the newsroom that works to collect, organize, analyze and present large amounts of information. He is originally from Swisher, Iowa.

James Gordon is a designer and developer at the Reynolds Journalism Institute. He also teaches data journalism at the Missouri School of Journalism. He was previously lead developer for the California Civic Data Coalition.

Andrea Suozzo is the data editor at Seven Days newspaper in Vermont. She works with reporters to develop data-driven stories, creates data visualizations for stories and builds bots and databases to support newsgathering.

Cheryl Phillips is the Hearst Professional in Residence at Stanford. She is a founder of Big Local News, the Stanford Open Policing Project and the California Civic Data Coalition. Previously, Phillips worked at The Seattle Times for 12 years in a variety of reporting and editing roles with the investigations team and across the newsroom.

### 2.15.1 About this class

This course was first developed by Welsh for an October 2016, “watchdog workshop” organized by Investigative Reporters and Editors at San Diego State University’s school of journalism.
It was revised for a February 2017 hands-on training of students at Stanford’s journalism school and expanded into a six-hour class at the annual conference of the National Institute for Computer-Assisted Reporting in March 2017.

It was expanded into its current form for a massive open online course offered by the Knight Center for Journalism in the Americas in May 2017.

Since then, the course has been taught more than a dozen times in classrooms and at conferences across America and around the world.

2.15.2 About the data

The course is based on data provided by the California Civic Data Coalition, an open-source network of journalists and computer programmers working to ease access to the state’s jumbled, dirty and difficult database tracking money in politics.

The goal of the coalition’s work is to make the data those reporters used easier to access, understand and analyze. Learn more about the status of the project and the data you can download at californiacivicdata.org.