First Python Notebook Documentation

Release 1.0

Ben Welsh and the California Civic Data Coalition

Mar 11, 2019
# Table of Contents

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

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 use pandas 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 in the sciences.
Who can take it

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 tools installed and working to participate. Visit each page to verify you are prepared. Installation instructions are provided.

2.1.1 Command-line interface

A command-line interface to interact with your computer is required.

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.

Windows

On Windows the command-line interface is called the “command prompt.” It is a perfectly fine piece of software, but to ease your experience with Python you should install cygwin, a free alternative. Here’s how:

1. Download the installer from cygwin.com.
2. Run the installer.
3. Verify it’s working.

Warning: When the class begins, all the examples will assume you are using cygwin instead of the standard command prompt. Please take the time to install it now. You’ll thank me later.
Mac OS X

On Apple’s Mac OS X operating system, a suitable command-line program is already installed. It’s called “Terminal.” You should try opening it up and typing in some basic commands to verify it’s working. Here’s how:

1. Click on the magnifying glass in the upper right hand corner.
2. Type in “Terminal”.
3. Press enter or select the first search result.
4. A command line prompt should appear.
5. Verify it works by typing a simple command like `whoami` and pressing enter.

2.1.2 Python

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 2.7.13
```

If not, you’ll need to install Python on your system.

Note: If you have a version of Python 3, don’t sweat it. It will work fine.

Windows

Here’s how to do so on Windows:

1. Verify Python is not already installed with the command above.
2. Download the Python 2.7 installer from `python.org <http://python.org>`.
3. Run the installer, and make sure to check the “Add python.exe to Path” setting is checked.
4. Verify that Python has been installed.

Mac OS X

If you’re using Mac OS X, it’s more likely that Python is already installed. Here’s how to find out what version you have.

You’ll start going to your terminal.

1. Go to your Terminal.
2. Type the `python --version` command described at the top of the page.
2.1.3 pip

The pip package manager makes it easy to install open-source libraries that expand what you’re able to do with Python. Later, we will use it to install everything needed to conduct a data analysis.

Verify pip is installed by typing the following into your command-line interface:

```
    pip --version
```

Windows

If you’ve followed the previous installation instructions for Python, pip should be already installed and ready to go. You can test it out by opening up the command prompt or cygwin and typing in the command above.

If you don’t have pip, carefully review the instructional video for Python and make sure you’ve followed all of the steps. If that doesn’t work, try following the official instructions from pip’s creators.

Mac OSX

In some cases, pip will be already be installed on Mac OSX computers. But to make sure you should take the following steps:

1. Check if pip is installed.
2. If not, install it using **easy_install**.
3. Verify pip is now available.

2.1.4 virtualenv

The virtualenv environment manager creates an isolated corner of your computer where all the Python tools you use to build an application are sealed off. We will use it to store all of the tools needed for the analysis in this class.

Windows

Here’s how to install it on Windows:

1. Verify pip and python are already installed as we did in the previous installers.
2. Check virtualenv is not already installed.

```
    virtualenv --version
```

3. Install virtualenv with pip.

```
    pip install virtualenv
```

4. Verify that virtualenv has been installed.

```
    virtualenv --version
```
Mac OS X

Here’s how to install it on Mac OS X:

1. Verify pip and python are already installed as we did in the previous installers.
2. Check virtualenv is not already installed.

```
virtualenv --version
```

3. Install virtualenv with pip. We’ll need to use `sudo` to install it in the system folders.

```
sudo pip install virtualenv
```

4. Verify that virtualenv has been installed.

```
virtualenv --version
```

2.1.5 Code compiler

A code compiler is a tool that lets your computer installed more advanced software. It is required to take advantage of the heavy-duty data analysis tools we will be using like the pandas library for Python.

Windows

Windows users will need to download and install the Microsoft Visual C++ Compiler for Python 2.7. Here’s a walk-through:

Mac OS X

If you are using Mac OS X, you will need to install Xcode. It’s a developer toolkit that includes a compiler. Here’s how:

You can make sure you’ve got it by running this on your command prompt.

```
xcode-select --install
```

2.1.6 Git

Git is a version control program for saving the changes you make to files over time. It is useful when you’re working on your own, but quickly becomes essential with large software projects, especially when you work with others.

For this class you will need to have git installed and working from your command-line interface.

You can verify it’s working from your terminal by typing in the follow code and hitting the enter key:

```
git --version
```

If git is installed and working, you should see something like this:

```
git version 2.11.0
```

If you don’t have it installed, you’ll need to follow the instructions below.
Windows

Here’s how to install git on Windows.

1. Verify the git is not already installed on the command prompt.
2. Download the installer from git-for-windows.github.io.
3. Run the installer, accepting all of the default options.
4. Return to the command prompt and verify git is now installed.
5. Configure git with your identity with these two commands:

   ```
   git config --global user.email "your@email.com"
   git config --global user.name "your name"
   ```

Mac OSX

If you’ve followed the previous installation instructions for a code compiler, git should be already installed and ready to go. You can test it out by opening up the command prompt or cygwin and typing in the command above. Here it is again:

   ```
   git --version
   ```

If it’s not, visit git-scm.com and download the installer. Run it.

Either way, configure git with your identity using these two commands:

   ```
   git config --global user.email "your@email.com"
   git config --global user.name "your name"
   ```

**Warning:** You will likely need administrator privileges on your computer. Be sure you have them before you begin.

### 2.2 Chapter 1: Hello virtualenv

The virtualenv environment manager makes it possible to create an isolated corner of your computer where all the Python tools you use to build an application are sealed off.

Why do you need this?

By developing each of your Python projects inside a separate virtual environment, you can:

- Juggle different versions of the same Python libraries without a conflict.
- Easily install your project on another machine, as can your colleagues
- Quickly copy your code to a server that publishes pages on the Internet.

For those reasons, virtualenv has become one of the most popular ways to manage Python projects. Alternatives include its more complex cousins virtualenvwrapper and conda.
Note: All that said, 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.

Learn how to create your first virtualenv by following this video or the written instructions below.

### 2.2.1 Create a code directory to store all your work

Before we create your first virtualenv, the first step is to create a common folder where all you of your projects will be stored starting with this one.

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.

```bash
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.

```bash
mkdir Code
```

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

### 2.2.2 Create a virtualenv to store this project

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

```bash
cd Code
```

Now use the virtualenv command to create a new virtual environment. This is a one time thing necessary to initialize a new environment.

```bash
virtualenv first-python-notebook
```

If you inspect the new directory this command created in your file explorer, you will see that it has generated a set of folders inside. They are the basic tools that make the virtual environment work and include a complete copy of the Python programming language just for this project.

### 2.2.3 Activate the virtualenv

Now use cd jump into the directory that was created created.

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

Now the trickiest part. Each time you want to begin working on a virtualenv project, you need to start off by “activating” it inside your terminal.

The activation program is called activate. It was created inside the new folders in this directory. You will need to run it each and every time you start work in this environment.

On Mac OSX the program is inside a folder called bin. You can easily run it from your terminal by using the `source` command.
source bin/activate

Fun fact: The source command has a shorter nickname if you don’t want to type as much. It is simply a period.

. bin/activate

On Windows the activate script is inside a folder called Scripts. You will need to move into that folder, run the script, and then back out to the folder we are in now.

First move into the folder.

cd Scripts

Activate the environment. If you are in Cygwin, it will be something like this.

If you are in the default Windows command prompt, more like this.

activate

Then move back to where we were before.

cd ..

You can verify that your virtualenv is running by using the which command to ask your computer what installation of Python it is currently using.

which python

If you are in your virtualenv, it should return a path leading to the same folder inside your virtualenv as activate. My looks like this:

/home/ben/Code/first-python-notebook/bin/python

2.2.4 Reactivate the virtualenv

You will need to remember to activate your virtualenv environment every time you log on to your computer and start work on this project. Before we move on, let’s take a moment to practice this routine.

Quit out of your command-line interface. Reopen it.

This new terminal will not be activated and working inside your virtual environment. You can verify this by using the which command again.

which python

This time, you are likely to see a path to your computer system’s global installation of Python, which we do not want to use on this project. Here’s what mine looks like (yours will be slightly different):

/usr/bin/python

We need to repeat the steps above to enter your new virtual environment and activate it.

First navigate into your code folder.

cd Code

Then into your virtualenv folder
Now activate your virtual environment with the source command. In you’re on Mac OSX, let’s use the shorter version this time.

```
source bin/activate
```

If you’re on Windows, here’s the routine again.

```
cd Scripts
. .\.activate
cd ..
```

Finally, verify the process has succeeded using the which command. It should now return a path leading to your virtual environment.

```
which python
```

That’s it for this chapter. You’ve successfully created your first virtual environment. Now let’s put it to use.

### 2.3 Chapter 2: Hello Jupyter

A [Jupyter Notebook](https://jupyter.org) 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
- “Fire officials were concerned about Westlake building where 5 died in a blaze” by the Los Angeles Times
- “Machine bias” by ProPublica

Learn how to install Jupyter and create your first notebook by following this [video](https://www.youtube.com/watch?v=example) or the written instructions below.

#### 2.3.1 Activate your virtual environment

To start off, open up your command-line interface and activate the virtual environment created for this project.

After the terminal is open, navigate to the code directory.

```
cd Code
```

Jump into virtual environment directory.

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

Activate the virtual environment, as we’ve done before.

```
# In Mac OSX ...
source bin/activate
# In Windows ...
cd Scripts
. .\.activate
cd ..
```
2.3.2 Install Jupyter Notebook with pip

Now that we’re in the virtual environment, we will use the `pip` command line tool to install the Jupyter Notebook software.

```
pip install jupyter
```

You can verify it’s been installed by running pip’s freeze command, which will list all of the installed Python libraries.

```
pip freeze
```

2.3.3 Create your first notebook

Now that Jupyter is installed, you can start its browser interface from your terminal.

```
jupyter notebook
```

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

![Jupyter Notebook Interface](image)

Click the “New” button in the upper right and create a new Python 2 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

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: The video at the top of this page introduces more of these Python fundamentals by writing and running cells of code in the notebook. If you’ve never written Python before, be sure to watch the clip before you advance to the next chapter.

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.

Learn how to install panda and conduct a simple data analysis by following this video or the written instructions below.

### 2.4.1 Install pandas with pip

We’ll install pandas the same way we installed the Jupyter Notebook earlier: Our friend **pip**.

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 using pip.

```
pip install pandas
```

### 2.4.2 Import pandas into a Jupyter Notebook

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

```
jupyter notebook
```

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

```
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.

```
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.

2.4. Chapter 3: Hello pandas
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.

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?

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:

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.
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
props.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
props.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
colls = 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.

```python
colls.head()
```

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.

```python
colls.info()
```

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 analyze a column from a DataFrame.

#### 2.7.1 Accessing a column

We’ll begin with the `prop_name` column where the proposition each committee is seeking 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 variable following a period.

```python
props.prop_name
```

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 in 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.

#### 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 any other method. All you need to do is add a second period after the column name and chain it on the tail end of your cell.

```python
props.prop_name.value_counts()
```

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
props.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 figure out the ones you need 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 runs pandas.

2.8 Chapter 7: Hello filters

Until last November, 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.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.
props.prop_name.value_counts()

From that result we can copy the full name of the proposition and place it between quotation marks to form the filter expression expected by pandas.

props.prop_name == 'PROPOSITION 064- MARIJUANA LEGALIZATION. INITIATIVE STATUTE.'

That expression is then placed between two flat brackets following the variable we want to filter. Place the following code in the next open cell in your notebook.

props[props.prop_name == 'PROPOSITION 064- MARIJUANA LEGALIZATION. INITIATIVE STATUTE.'][

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.

prop = props[props.prop_name == 'PROPOSITION 064- MARIJUANA LEGALIZATION. INITIATIVE STATUTE.'][

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.

prop.head()

Then info.

prop.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.

contribs.info()

Now compare that to the committees file.

props.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(prop, contribs, 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 merge.

```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 above.

```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.

```python
merged.committee_position.value_counts()
```

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.

```python
support = merged[merged.committee_position == 'SUPPORT']
```

Now let’s repeat all that for opposing contributions. First the filter into a new variable.
oppose = merged[merged.committee_position == 'OPPOSE']

Now sum up the total disclosed contributions to each for comparison. First the opposition.

oppose.amount.sum()

Then the supporters.

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.

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.

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`.

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

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

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.

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

And now how about the opposition.

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

We’ve encountered a lot of different committees as we’ve explored the data. A natural question follows: Which ones have raised the most money?

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 committee_name_x and committee_name_y.

That is because the field was present on both our committee list and our contributions list prior to joining them together. Rather than drop one of them, pandas is trained to keep them both and to append suffixes to the end.

That’s the field we want to group with here. Since the two columns are identical, it doesn’t matter which one we pick. Let’s go with x.

```
merged.groupby("committee_name_x").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.

```
merged.groupby("committee_name_x").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.

```
merged.groupby("committee_name_x").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 “committee_name_x” with a list like the one here:

```
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.

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

You’ve done it. Our brief interview is complete and you’ve answered the big question that started our inquiry. 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 percentage of donations came from people who live outside of California?
- What are the top employers of donors who gave for and against the measure?
- Which committees had the fewest donors?
2.13 Chapter 12: Hello git

In this chapter, we’ll step away from Python and the notebook to introduce another important tool: `git`.

Git is a command-line tool that tracks changes in files and makes it easier to collaborate with our programmers. It is widely used in open-source software and by most professional software development teams.

It will allow us to edit and delete our code without fear of losing past work. Ultimately it will enable us to publish our notebook for the entire world to read and remix.

2.13.1 Creating a new repository

The first step in working with git is to convert a directory on your computer into a repository that will have its contents tracked going forward.

You do that by returning to your terminal. If your notebook server is running, hit the `CTRL-C` key combination to return the standard command line. Then entering the following:

```
$ git init .
```

That will instruct git to initialize a new repository in your current folder, which is represented by the period.

If this is your first time using git, you should configure git with your name and email. This will ensure that your work is properly logged by the repository’s history file. Like the init command above, this is something that only needs to be done once.

```
$ git config --global user.email "your@email.com"
$ git config --global user.name "your name"
```

Note: If you followed along with our git configuration instructions in the prerequisites, you should have done this already. But it wouldn’t hurt anything to do it again either.

2.13.2 Committing your work

Now you’re ready to start logging your work. Changes to you code are logged by git in batches known as “commits.” It is not required but a good first step before committing any work is to run git’s status command, which will output the current state of your repository.

```
git status
```

Since your repository is brand new, all of the files will be listed as “untracked.” That means that while git sees that these files exist it is not monitoring them for changes.

The first step in logging your work is to ask git to start tracking your files using the `add` command.

In this repository, the only file we need to track are your notebooks. You can add all of them by running the command below, which uses the “wildcard” asterisk to start tracking all files with the notebook extension.

```
$ git add *.ipynb
```

Note: The `add` command isn’t only for when you are adding new files to your repository. You should run it each time a file has been changed and you’d like to commit the work.
Run the status command again and you’ll see something different. That’s git telling you that the notebooks in your repository have been staged and are ready to commit.

```bash
$ git status
```

Log its addition with git’s `commit` command. You must include a personalized message, which you can provide along with the command by adding on the `-m` flag along with a description of the work you’ve done.

```bash
$ git commit -m "First commit"
```

That’s it. You’ve made your first git commit.

**Note:** There’s no rule about when to commit your work, but disciplined developers get in the habit of doing it frequently. Whenever you’ve reached a small milestone or a breaking point in your work, it’s a good idea to make a commit.

To get some practice, save a change to your notebook and try to log another.

It might seen a little burdensome at first, but it is correct to run the status, add and commit commands each time. Take it slowly and carefully. You’ll quickly get the hang of it.

### 2.14 Chapter 13: Hello Markdown

If you have looked at notebooks published by others, you’ve probably noticed that the authors have helpfully summarized and annotated their code by inserting text, links and images between code blocks.

Here’s an example from a recent project by the Los Angeles Times.
This is accomplished by adding new cells to your notebook and converting them from the default output, Python code, to an alternative called Markdown. Markdown is a markup language that formats text. It’s a common, lightweight alternative to HTML.

### 2.14.1 Writing Markdown

To create and print a new Markdown cell in your notebook, start up your notebook again from the terminal.

```
$ jupyter notebook
```

Now open your notebook file. At the top, add a new cell by clicking the plus button and hitting the up arrow button to move it to the top slot.

Click on the box and use your mouse to pull down the option menu that current reads “Code” from the toolbar. Replace it with “Markdown.”
Now click into the cell and type the following:

```plaintext
# First Python Notebook

Hello world!
```

Hit the play button you will see the result. The first line has been turned into a header because that is how Markdown formats # at the front of lines. To learn more Markdown rules refer to its documentation.
Now try adding more cells to your document lower down and annotating individual lines of code before they are run.

After you’ve finished, save your notebook and return to your terminal so we can commit your work and push it to GitHub. Again, open the terminal and hit the `CTRL-C` key combination to halt the notebook.

You can see git has noticed the changes you’ve made by running the status command.

```bash
$ git status
```

Now tell git to log your notebook file changes using add.

```bash
$ git add *.ipynb
```

Now log your changes with commit.

```bash
$ git commit -m "Markdown"
```
2.15 Chapter 14: Hello GitHub

In this section, we will publish your notebook to the Internet using GitHub, a social network for sharing and collaborating on code. GitHub is a platform frequently used by journalists and others to publish their notebooks. Examples include:

- “The Tennis Racket” by BuzzFeed and the BBC
- “California H-2A visas analysis” by the Los Angeles Times
- “Machine bias” by ProPublica

GitHub is an online extension of the git command-line tool we installed in a previous chapter. In may help to think of GitHub as a social network that publishes and connects git code repositories.

GitHub also offers free hosting to open-source software projects. It is where many of the world’s largest are developed, including pandas, the Jupyter Notebook and the flavor of Python they favor.

**Warning:** If you’ve set up your GitHub account with two-factor authentication, you will need to take the extra step of generating an SSH key that allows you to publish your work.

If you’ve just made a GitHub account for the first time, do not worry about this. If you have turned on two-factor authentication, but don’t have SSH keys, read more how to set it up in GitHub’s documentation.

2.15.1 Creating a GitHub repository

Visit GitHub and create a new public repository named first-python-notebook. Don’t check “Initialize with README.” You want to start with a blank repository.

This will create a new repository page. It needs to be synchronized with the local repository we’ve already created.

You can connect your local directory to GitHub’s site by returning to the terminal and using git’s “remote add” command to connect it with GitHub.

```
$ git remote add origin https://github.com/<yourusername>/first-python-notebook.git
```

**Note:** The tradition among developers is to label your default remote repository as the “origin”, which is why you see it included in the command above. You could substitute a different nickname for GitHub there if you wanted, but it’s probably best to stick with the standard terminology.

Next we’ll try “pushing” the latest commit from your repository up to GitHub. Assuming all of your work has been properly logged to your local repository, here’s what it takes.

```
$ git push origin master
```

**Note:** That might seem like a long weird command (it is!) but it’s one you’ll need to memorize to continue working with GitHub.

If you pick it apart, it’s not that complicated. The first two terms, git push, tell git you’d like to copy your code from your local directory to a remote repository. The third, origin, indicates the destination you’d like to update. Remember, we gave our GitHub repository the alias origin. The fourth tells it which branch of your code you’d like to synchronize. We haven’t covered it yet in this class, but the standard branch of code in any new git repository is called the master branch. If you’re interested, you can read more about branches in the git user manual.
Now reload the GitHub page in your browser and you should see your code online. Congratulations, you’ve published you work.

To see how your notebook appears on the web, click on the ipynb file in the browser and it should render much like it does in the live notebook on your localhost server. This URL can now be used to share your work with the entire world.

As a parting challenge, try saving a change to your local notebook. Then add and commit it to your repository with git. Finally push it back up to GitHub to see your published the change arrive online.

### 2.16 Chapter 15: Hello charts

Python has a number of charting tools that can work hand-in-hand with pandas. The most popular is matplotlib. It isn’t the prettiest thing in the world, but it offers straightforward tools for exploring your data by making quick charts. And, best of all, it can display in your Jupyter Notebook.

Before we start, we’ll need to make sure matplotlib is installed. Return to your terminal and try installing it with our buddy pip, as we installed other things before.

```bash
$ pip install matplotlib
```

After that completes, once again restart your notebook.

```bash
$ jupyter notebook
```

Now you can open your notebook and add a new cell below the imports that lets the system know you plan to make some charts and that it’s okay to surface them in the notebook.

```python
%matplotlib inline
```

Let’s return to where we set our proposition filter at the top and restore it our initial interest, Proposition 64.

```python
prop = props[props.prop_name == 'PROPOSITION 064- MARIJUANA LEGALIZATION. INITIATIVE STATUTE. ']
```

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.

Then scroll down to the bottom of the notebook and pick up where we last left off in the previous chapter.

If we want to chart out the top supporters of the proposition, 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.groupby('contributor_firstname', 'contributor_lastname').amount.sum().reset_index().sort_values('amount', ascending=False).head(10)
```

We can then view them with a trick I bet you remember by now.
Now that matplotlib is installed, making a simple chart is as simple as stringing the `plot` method onto the end of your `DataFrame`.

```
top_supporters.amount.plot.bar()
```

You can rotate the bar chart so that it is horizontal by substituting in the `barh` method.

```
top_supporters.amount.plot.barh()
```

The chart can be limited to the first five records by slipping in the `head` command.
What are those y axis labels? Those are the row number (pandas calls them indexes) of each row. We don’t want that. We want the names. We can swap them in by saving our chart to a variable and then using another matplotlib option, set_yticklabels to instruct the system which field to use.

```python
chart = top_supporters.head(5).amount.plot.barh()
chart.set_yticklabels(top_supporters.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
```

We can see the results right here.

```python
top_supporters.head()
```

Now using that in the chart is as simple as substituting in the set_yticklabels method we used above.
That's all well and good, but this chart is still pretty ugly. If you wanted to hand this data off to your graphics department, or try your hand at a simple chart yourself using something like Chartbuilder, you'd need to export this data into a spreadsheet.

Guess what? It’s this easy.

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

2.17 Chapter 16: Hello cleaning

You’ve probably noticed that top two supporters are the same person, **Sean Parker**. However, due to variations in how the `contributor_lastname` and `contributor_firstname` fields were filled out on the disclosure forms his contributions were not combined during our earlier grouping.

A common approach to correcting this issue is to create a new column where the cleaned up version of names are stored alongside the raw values from the public data. If two rows with different raw names are given an identical name in the clean column, that field can then be used to group the data and aggregate their contributions together.

There are several different approaches to making that happen. In this introductory lesson we’re going to take a simplified approach that will also teach you a valuable skill: How to use panda’s `apply` method to generate a new column.

The first step is to write a Python **function**. A function is a fundamental tool shipped with Python that allows you to define a chunk of code that you can rerun later. A typical function will take an input and return a result based on what’s passed in each time it’s run.
Functions are useful when you have a series of statements you want to run again and again. In our case, we’re going to write a function that can inspect any row in our data frame. If that row’s name field starts with the phrase SEAN PARKER we’re going to return a common clean name that all Sean Parker rows can share. If it doesn’t, we’re just going to return the raw name value.

That looks like the code below. Create a new cell and run it there.

```python
def combine_names(row):
    if row.contributor_fullname.startswith('SEAN PARKER'):
        return 'SEAN PARKER'
    return row.contributor_fullname
```

You know it’s a function because it starts with `def`. The name we’re giving the function, `combine_names` is what follows. Then a parenthesis that defines the input the function will expect each time it’s run. Since we’re going to run this function on every row in a DataFrame, we will only have one input and we will name it `row`.

Now that we have our function, we’ll want to run it across our DataFrame and store the result for each row in a new column. In pandas this can be done for a DataFrame with the `apply` method. To run the method row by row, pandas requires you pass the number 1 into the `axis` option.

```python
top_supporters.apply(combine_names, axis=1)
```

That will spit out a Series with the returned value for each row. Notice that the two Sean Parker rows are now identical.

```
444  SEAN PARKER
443  SEAN PARKER
102  DANIEL LEWIS
180  HENRY VAN AMERINGEN
374  NICHOLAS PRITZKER
399  PETER THIEL
421  ROGER JENKINS
22   ANGELA HOWARD
297  LAWRENCE HESS
423  ROGER B. MCBRIDE

dtype: object
```

That Series can now be saved into a new column on the DataFrame by assigning it with a new name.

```
top_supporters['contributor_cleanname'] = top_supporters.apply(combine_names, axis=1)
```

This same approach can be used to add new columns that do all kinds of things. One common trick is to use the apply method and a function to divide one column into another, or some other mathematical computation.

Now you can regroup the data using this new column and total the columns again, exactly as we did before.

```
top_supporters.groupby('contributor_cleanname').amount.sum().reset_index().sort_values('amount', ascending=False).head(10)
```

The money previously split between two variations on Sean Parker’s name are now combined. You could now remake the charts above without the duplication.
And that’s it. You’ve completed this class. If you want to keep working, try inventing your own questions to ask and answer with the database, or find more information to analyze at californiacivicdata.org.

If you have any questions or feedback on the class, please contact me at ben.welsh@gmail.com.

2.18 About the author

Ben Welsh is the editor of the Los Angeles Times Data Desk, a team of reporters and computer programmers in the newsroom that works to collect, organize, analyze and present large amounts of information.

He is also a cofounder of the California Civic Data Coalition, an open-source network of developers working to open up public data, and the creator of PastPages, an archive dedicated to the preservation of online news.

Ben has worked at the Los Angeles Times since 2007. Before working at The Times, Ben conducted data analysis for investigative projects at The Center for Public Integrity in Washington DC.

Projects he has contributed to have been awarded the Pulitzer Prize, the Library of Congress’ Innovation Award and numerous other prizes for investigative reporting, digital design and online journalism.

Ben graduated from DePaul University in 2004. During his time there, he worked with Carol Marin and Don Moseley at the DePaul Documentary Project. He later earned a master’s degree from the Missouri School of Journalism — where he served as a graduate assistant at the National Institute for Computer-Assisted Reporting.

He is originally from Swisher, Iowa.

2.18.1 About this class

This course was first developed 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.

### 2.18.2 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.