Set 14: Data Acquisition

Skill 14.1: Differentiate between primary data and secondary data
Skill 14.2: Differentiate between clean and raw data
Skill 14.3: Identify data file formats
Skill 14.4: Describe the purpose of an API
Skill 14.5: Acquire data through an API request
Skill 14.6: Make an API request in Python
Skill 14.7: Convert JSON data to CSV
Skill 14.8: Explore data using the *pandas* library

Skill 14.1: Differentiate between built-in and user-defined functions

Skill 14.1 Concepts

Data can be acquired from many different sources. Broadly, they can be categorized into **primary data** and **secondary data**.

Primary data is data collected by the individual or organization who will be doing the analysis. Examples include:

- Experiments (e.g., wet lab experiments like gene sequencing)
- Observations (e.g., surveys, sensors, *in situ* collection)
- Simulations (e.g., theoretical models like climate models)
- Scraping or compiling (e.g., webscraping, text mining)

Secondary data is data collected by someone else and is typically published for public use. Most data you will use falls into this category. Examples include:

- Any primary data that was collected by someone else
- Institutionalized data banks (e.g., census, gene sequences)

As you can imagine, collecting your own primary data can be time consuming. However, the closer you are to the data, the better you understand it and its nuances. On the other hand, secondary data is much easier to find. Even with secondary data, understanding how the data was created is essential in order to correctly utilize and analyze it. This includes reading any available data methodology or README files.

Skill 14.1 Exercise 1

Skill 14.2: Differentiate between cleaned and raw data

Skill 14.2 Concepts

There is another subcategory of secondary data that can be called “pre-cleaned” data. These are datasets published by institutions like Kaggle that are already cleaned, filtered, and ready to use.

While pre-cleaned data is undoubtedly easier to use, you lose some of the flexibility and control that working with unaltered, “raw” data offers. For example, if a cleaned dataset discarded certain fields or rows during the data processing step, there is no way of getting the information back. Nevertheless, pre-cleaned datasets are still popular, and that can be perfectly fine depending on the end goal.

Below is an example of raw data,

Raw Data (uncleaned)

ID	Age	City	Response Time	Favorite Color
1	17	seattle	3 min	Blu
2	-	Seatle	45 sec	blue
3	19	Seattle	2m	BLUE
4	0	seattl	90 seconds	bluu

Problems: inconsistent spelling, missing values, inconsistent units (“3 min” vs “45 sec”), impossible values (age 0), inconsistent capitalization.

Below is the same dataset that has been cleaned,

ID	Age	City	Response Time (sec)	Favorite Color
1	17	Seattle	180	Blue
2	—	Seattle	45	Blue
3	19	Seattle	120	Blue
4	—	Seattle	90	Blue

Fixes: standardized casing/spelling, consistent units, removed impossible ages, normalized color names.

Skill 14.2 Exercise 1

Skill 14.3: Identify data file formats

Skill 14.3 Concepts

Data can come in a variety of different file formats, depending on the type of data. Being able to open and convert between these file types opens a whole world of data that is otherwise inaccessible. Examples of common formats are summarized below,

Data Type	Definition	Examples / File Extensions
Tabular	Data organized in rows and columns	.csv (students.csv), .tsv (survey.tsv), .xlsx (sales.xlsx)
Non-Tabular	Data not in a table; text, markup, or documents	.txt (notes.txt), .rtf (report.rtf), .xml (data.xml)
Image	Visual data stored as pictures	.png (diagram.png), .jpg (photo.jpg), .tif (scan.tif)
Agnostic / Binary	Generic or binary data; requires software to interpret	.dat (experiment.dat)

Further, some file formats are proprietary and can only be opened by software developed by a particular company. Opening these in another program requires converting to a universal format (this has to do with how the characters are encoded). Proprietary formats include Excel or MS Access files that are designed to be opened by Microsoft Office applications, as opposed to more generic types like .csv files. There are also other file formats that store metadata, such as SPSS and STATA files that contain information on data labels.

It is best practice to store data in a way that is most easily accessible for everyone. Generally, this means storing data in a non-proprietary and openly documented format, using standard character encodings (utf-8), and keeping data files uncompressed, if space allows. There are various methods, including online tools, that can be used to convert between formats if necessary.

Skill 14.3 Exercise 1

Skill 14.4: Describe the purpose of an API

Skill 14.4 Concepts

Whenever you develop a new research question you want to investigate, the very first thing you'll need to do is collect the data!

Here, we will explore an example of acquiring secondary data using an API (Application Programming Interface). An **API** (Application Programming Interface) is a set of rules that allows one piece of software to communicate with another.

An everyday analogy of an API is a restaurant menu,

- The menu (API) lists what the kitchen (software system) can do.
- You (the user or another program) choose what you want by placing an order (making an API request).
- The kitchen prepares it and returns it to you (API response).

APIs are useful because they enable the exchange of information without knowing the internal details.

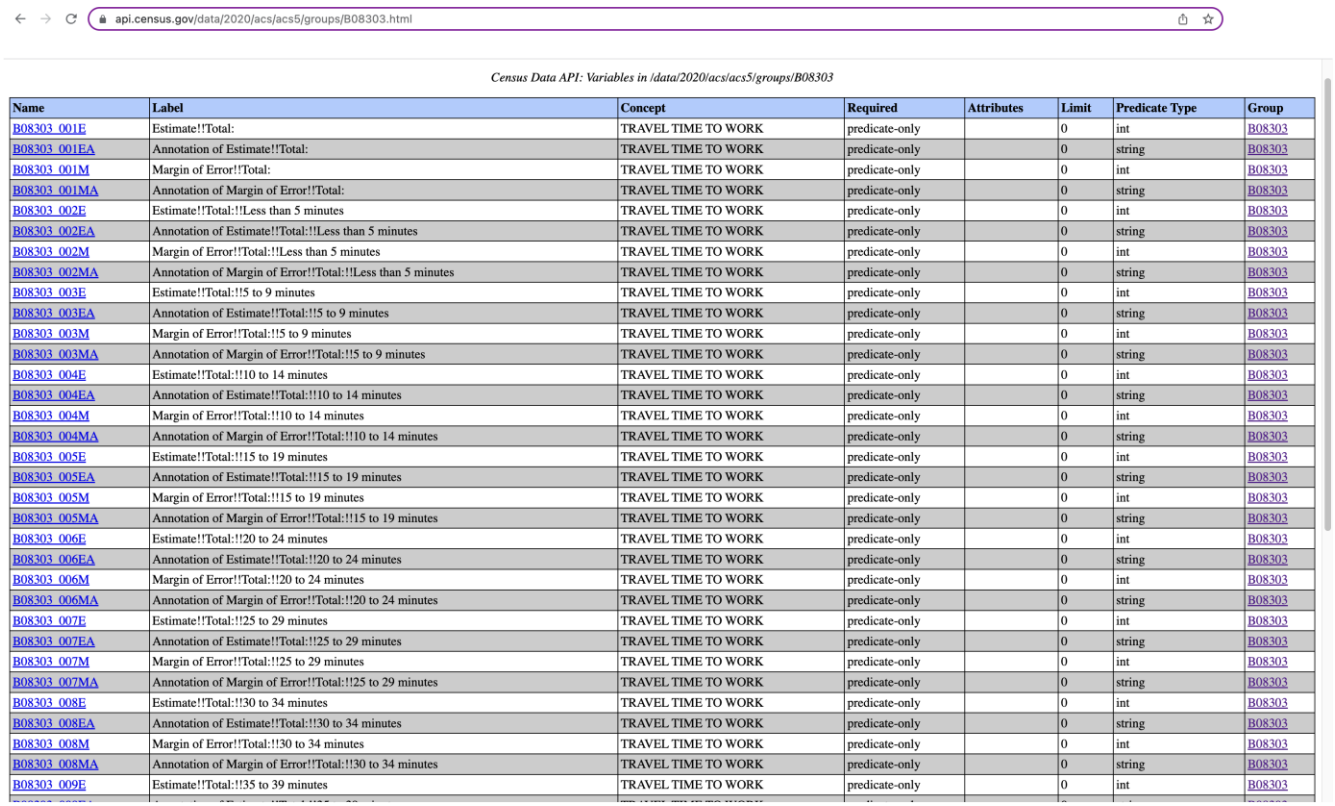
Skill 14.5: Acquire data through an API request

Skill 14.5 Concepts

Imagine you are curious about the average commute time to work in the U.S. and want to compare it across different regions.

To answer our question about commute time to work, we will be using Census data, specifically the American Community Survey (ACS) 5-year estimates. There is a group of variables that breaks down the population counts by travel time to work.

The full table of variables is accessible through the Census Data API and a graphic showing the first portion of the table is below,



Census Data API: Variables in /data/2020/acs/acs5/groups/B08303

Name	Label	Concept	Required	Attributes	Limit	Predicate Type	Group
B08303_001E	Estimate!!Total:	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_001EA	Annotation of Estimate!!Total:	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_001M	Margin of Error!!Total:	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_001MA	Annotation of Margin of Error!!Total:	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_002E	Estimate!!Total:!!Less than 5 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_002EA	Annotation of Estimate!!Total:!!Less than 5 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_002M	Margin of Error!!Total:!!Less than 5 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_002MA	Annotation of Margin of Error!!Total:!!Less than 5 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_003E	Estimate!!Total:!!5 to 9 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_003EA	Annotation of Estimate!!Total:!!5 to 9 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_003M	Margin of Error!!Total:!!5 to 9 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_003MA	Annotation of Margin of Error!!Total:!!5 to 9 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_004E	Estimate!!Total:!!10 to 14 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_004EA	Annotation of Estimate!!Total:!!10 to 14 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_004M	Margin of Error!!Total:!!10 to 14 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_004MA	Annotation of Margin of Error!!Total:!!10 to 14 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_005E	Estimate!!Total:!!15 to 19 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_005EA	Annotation of Estimate!!Total:!!15 to 19 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_005M	Margin of Error!!Total:!!15 to 19 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_005MA	Annotation of Margin of Error!!Total:!!15 to 19 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_006E	Estimate!!Total:!!20 to 24 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_006EA	Annotation of Estimate!!Total:!!20 to 24 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_006M	Margin of Error!!Total:!!20 to 24 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_006MA	Annotation of Margin of Error!!Total:!!20 to 24 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_007E	Estimate!!Total:!!25 to 29 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_007EA	Annotation of Estimate!!Total:!!25 to 29 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_007M	Margin of Error!!Total:!!25 to 29 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_007MA	Annotation of Margin of Error!!Total:!!25 to 29 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_008E	Estimate!!Total:!!30 to 34 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_008EA	Annotation of Estimate!!Total:!!30 to 34 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_008M	Margin of Error!!Total:!!30 to 34 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303
B08303_008MA	Annotation of Margin of Error!!Total:!!30 to 34 minutes	TRAVEL TIME TO WORK	predicate-only		0	string	B08303
B08303_009E	Estimate!!Total:!!35 to 39 minutes	TRAVEL TIME TO WORK	predicate-only		0	int	B08303

The Census Data API provides a fast and simple way to access its data. To query the ACS 5-year data, a request should be made to https://api.census.gov/data/2020/acs/acs5 specifying the variables to fetch and the geographic level you want.

For example, a call requesting the total population count of commuters (B08303_001E) at the state level is shown below,

Call
`https://api.census.gov/data/2020/acs/acs5?get=NAME,B08303_001E&for=state:*`
Description
Returns the total population of commuters from all states.
Output
[["NAME","B08303_001E","state"], ["Pennsylvania","5652158","42"], ["California","16710195","06"], ["West Virginia","697042","54"], ["Utah","1378826","49"], ["New York","8584828","36"], ["District of Columbia","332618","11"], ["Alaska","330203","02"], ["Florida","8817718","12"], ["South Carolina","2174285","45"], ["North Dakota","380227","38"], ["Maine","608861","23"], ["Georgia","4475685","13"], ["Alabama","2002359","01"], etc…

Let's break down the syntax of this call…

- The part of the URL after ? contains the query parameters each separated by an *&*
- The *get* parameter specifies a comma-separated list of variables we want to fetch
    - *NAME* is the name of the geographic level
    - *B08303_001E* is the number of commuters
- The *for* parameter specifies the geographic level
    - we are requesting state-level data
    - and we want all states, as indicated by the *.

Looking through the output above, you may have also noticed an extra column of data at the end. This is automatically generated and is the code for the geographic level of the data. In this case, the last column contains the 2-digit state code.

Skill 14.5 Exercises 1

The Census website documents all the supported geographic levels with examples. Below is a screen shot of a portion of the levels.

state	040	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=state:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=state:06&key=YOUR_KEY_GOES_HERE
state› county	050	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=county:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=county:*&in=state:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=county:037&in=state:06&key=YOUR_KEY_GOES_HERE
zip code tabulation area	860	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=zip%20code%20tabulation%20area:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=zip%20code%20tabulation%20area:77494&key=YOUR_KEY_GOES_HERE
state› school district (elementary)	950	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=school%20district%20(elementary):*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=school%20district%20(elementary):*&in=state:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=school%20district%20(elementary):99999&in=state:48&key=YOUR_KEY_GOES_HERE
state› school district (secondary)	960	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=school%20district%20(secondary):*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=school%20district%20(secondary):*&in=state:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=school%20district%20(secondary):99999&in=state:48&key=YOUR_KEY_GOES_HERE
state› school district (unified)	970	https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=school%20district%20(unified):*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=school%20district%20(unified):*&in=state:*&key=YOUR_KEY_GOES_HERE
		https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=school%20district%20(unified):99999&in=state:06&key=YOUR_KEY_GOES_HERE

As we've already seen, using * specifies all data at that geographic level. For example, to request county-level data for all counties, the API call would be,

Call
https://api.census.gov/data/2020/acs/acs5?get=NAME,B01001_001E&for=county:*
Description
Returns the total population of commuters from all counties
Output
[["NAME","B01001_001E","state","county"], ["Autauga County, Alabama","55639","01","001"], ["Baldwin County, Alabama","218289","01","003"], ["Barbour County, Alabama","25026","01","005"], ["Bibb County, Alabama","22374","01","007"], ["Blount County, Alabama","57755","01","009"], ["Bullock County, Alabama","10173","01","011"], ["Butler County, Alabama","19726","01","013"], etc…

We can also request specific geographic regions, like states, by specifying them in a comma-separated format. For example, to request data from California and Utah, we specify their 2-digit FIPS state code (California is 06 and Utah is 49),

Call
https://api.census.gov/data/2020/acs/acs5?get=NAME,B08303_001E&for=state:06,49
Description
Returns the total population of commuters from California and Utah
Output
[["NAME","B08303_001E","state"], ["California","16710195","06"], ["Utah","1378826","49"]]

The *for* parameter can also be used in conjunction with *in* to specify data within a certain geographic area, such as all unified school districts within specific states,

Call
https://api.census.gov/data/2020/acs/acs5?get=NAME,B08303_001E&for=school%20district%20(unified):*&in=state:06,49
Description
Returns the total population of commuters from all school districts in California and Utah
Output
[["NAME","B08303_001E","state","school district (unified)"], ["Big Valley Joint Unified School District, California","464","06","05010"], ["Biggs Unified School District, California","1388","06","05040"], ["Black Oak Mine Unified School District, California","4992","06","05240"], ["Bonita Unified School District, California","26256","06","05610"], ["Borrego Springs Unified School District, California","462","06","05700"], ["Brea-Olinda Unified School District, California","18859","06","05880"], ["Burbank Unified School District, California","45982","06","06450"], etc…

Skill 14.5 Exercises 2

Skill 14.6: Make an API request in Python

Skill 14.6 Concepts

Now that you have a pretty good idea of how the Census Data API works let's take a look at how to pull the data in Python.

To pull data into Python requires the *requests* library. Once this is imported, we can the *get()* method to return the data from our desired URL,

```
import requests

r = requests.get('https://api.census.gov/data/2020/acs/acs5?get=NAME,B08303_001E&for=state:*')
```

The result is a response object. Here we store it in a variable named *r*.
We can look at that response data by using the .text attribute. The text attribute turns the data into a string.

Code
`r = requests.get('https://api.census.gov/data/2020/acs/acs5?get=NAME,B08303_001E&for=state:*')` `print(r.text)`
Output
[["NAME","B08303_001E","state"], ["Pennsylvania","5652158","42"], ["California","16710195","06"], ["West Virginia","697042","54"], ["Utah","1378826","49"], ["New York","8584828","36"], etc…

We can also use the *.json()* method that can automatically decode data into the appropriate Python object. This is useful when working with JSON data, as in the case of the Census API, to have the data in a more intuitive data structure.

Code

```
r = requests.get('https://api.census.gov/data/2020/acs/acs5?get=NAME,B08303_001E&for=state:*')
print(r.json())
```

Output

[['NAME', 'B08303_001E', 'state'], ['Pennsylvania', '5652158', '42'], ['California', '16710195', '06'], ['West Virginia', '697042', '54'], ['Utah', '1378826', '49'], ['New York', '8584828', '36'], ['District of Columbia', '332618', '11'], ['Alaska', '330203', '02'], ['Florida', '8817718', '12'], etc…

It may not be evident from the examples above, but the *json()* method has a clear advantage over the *text* attribute because it converts the response into native Python data structures. See the examples below for the API request above,

Code

```
r_text = r.text
print(r_text[1])#converts the data into a String


r_json = r.json()#converts the data into a usable Python data structure
print(r_json[1])
```

Output

[
['Pennsylvania', '5652158', '42']

Skill 14.6 Exercises 1

Skill 14.7: Convert JSON data to CSV

Skill 14.7 Concepts

While JSON is a great universal format for data interchange, it might not be the ideal format in other aspects, such as readability. Instead, having the data in a tabular format (like a CSV) can make it much more human-readable and accessible. Therefore, being able to convert between file types is essential.

There are several libraries in Python to work with different data formats. For example, to convert the Census data from JSON to CSV we can use the built-in *csv* library.

As a refresher, visit https://api.census.gov/data/2020/acs/acs5?get=NAME,B08303_001E&for=state:* in your browser to view the total commuters count for all states.

The JSON data we got from the Census is a list of lists in Python, where each inner list corresponds to a single row of data.

To convert from JSON to CSV, we want to write each sublist as a comma-separated row of data to file. This bit of code is a little complicated. We will use the *writerows()* from the *csv* library,

Code

```
import requests
import csv
r = requests.get('https://api.census.gov/data/2020/acs/acs5?get=NAME,B08303_001E&for=state:*')
r_json = r.json()#converts the data into a usable Python data structure
# writes the data to a file called census.csv
with open('census.csv', mode='w', newline='') as file:
  writer = csv.writer(file)
  writer.writerows(r_json)
```

Output (census.csv)

census.csv > data

```
1   NAME,B08303_001E,state
2   Pennsylvania,5652158,42
3   California,16710195,06
4   West Virginia,697042,54
5   Utah,1378826,49
6   New York,8584828,36
7   District of Columbia,332618,11
8   Alaska,330203,02
9   Florida,8817718,12
10  South Carolina,2174285,45
11  North Dakota,380227,38
12  Maine,608861,23
13  Georgia,4475685,13
```

Below is an explanation for what happened in the example above,

- The *with* keyword is a code block within which we can safely process the file.
- `open('census.csv', mode='w')` creates or overwrites the *census.csv* file with `mode='w'` for *writing* mode and `newline=''` to ensure that newlines are always interpreted correctly.
- Next, we use the *writer()* function from the csv library to make a writer object (don't worry about what this is right now).
- We then use the *writerows()* method to write each row of data into comma-separated format.

Skill 14.7 Exercises 1

Skill 14.8: Explore data using the *pandas* library

Skill 14.8 Concepts

Now that we have our data saved into an easy-to-work-with format, we can investigate it! We'll use Python's *pandas* library, which is designed for working with data.

Below illustrates how we can use the *read_csv()* function from the *pandas* library to read csv data into a *DataFrame* object called *census_df*.

```
import pandas

census_df = pandas.read_csv("census.csv")
```

By default, the first row of the CSV file is read in as the header row. We can use the *head()* method to preview the first few rows of the DataFrame.

Code

```
import pandas

census_df = pandas.read_csv("census.csv")

print(census_df.head())
```

Output

```
           NAME  B08303_001E  state
0  Pennsylvania      5652158     42
1    California     16710195      6
2  West Virginia      697042     54
3          Utah      1378826     49
4      New York      8584828     36
```

Sometimes columns have ambiguous or confusing names (like Census codes). We might also want to rename those columns. We can use the *columns* attribute to rename the column headings if needed. Notice the difference between the output above compared to the output below,

Code

```
import pandas

census_df.columns = ['name', 'total_commuters', 'state']

print(census_df.head())
```

Output

```
           name  total_commuters  state
0  Pennsylvania          5652158     42
1    California         16710195      6
2  West Virginia          697042     54
3          Utah          1378826     49
4      New York          8584828     36
```

The *pandas* library offers a lot more functionality for exploring and manipulating tabular data. We will learn more about *pandas* later!

Skill 14.8 Exercises 1