- Get Twitter Data
- Tweet Word Frequency Analysis
- Tweet Word Bigrams and Network Analysis
- Tweet Sentiment Analysis
- CO Flood Tweets JSON

SECTION 8
EARTH DATA SCIENCE WORKFLOWS
Chapter 12: Design and Automate Data Workflows

SECTION 9
DATA STORIES
Chapter 20: Flood overview
Chapter 21: Intro to Lidar Data
Chapter 22: Wildfire Overview

OVERVIEW

Use Data for Earth and Environmental Science in Open Source Python Home

intermediate-earth-data-science-textbook Home

Lesson 1. Introduction to APIs

Leah Wasser, Carson Farmer, Max Joseph, Martha Morrissey, Jenny Palomino

Introduction to programmatic data access in Python - Intermediate earth data science textbook course module

Welcome to the first lesson in the **Introduction to programmatic data access in Python** module. In this module, you learn various ways to access, download and work with data programmatically. These methods include downloading text files directly from a website onto your computer and into Python, reading in data stored in text format from a website into a Pandas DataFrame, and accessing subsets of particular data using REST API calls in Python.

Learning Objectives

After completing this tutorial, you will be able to:

- Describe the difference between human and machine readable data structures.
- Describe the difference between data returned using an API compared to downloading a text file directly.
- Describe 2-3 components of a RESTful API call.
- List different ways to access data programmatically in Python.

What You Need

You will need a computer with internet access to complete this lesson.

Access Data Programmatically

This week, you will learn how to programmatically access data using:

1. Direct downloads / import of data.
2. Applied Programming Interfaces (APIs).

Up until this point, you have been downloading data from a website (in the case of this course, Figshare) independently. Then, you work with the data in `Python`. The data that you have downloaded are prepared specifically for this course.

However, independently downloading and unzipping data each week is not efficient and does not explicitly tie your data to your analysis.

You can automate the data download process using `Python`. Automation is particularly useful when:

- You want to download lots of data or particular subsets of data to support an analysis.
- There are programmatic ways to access and query the data online.

Link Data Access to Processing & Analysis

When you automate data access, download, or retrieval, and embed it in your code, you are **directly** linking your analysis to your data. Further, combined with Jupyter Notebooks, code comments and expressive coding techniques, you are better documenting your workflow.

In short - by linking data access and download to your analysis - you are not only reminding your future selves of your process - you are also reminding your future self where (and how) you got the data in the first place! Similarly, this allows your workflow to be easily reproduced by others.

Three Ways to Access Data

You can break up programmatic data access into three general categories:

1. Data that you download by calling a specific URL and using the `Pandas` function `read_table`, which takes in a url.
2. Data that you directly import into `Python` using the `Pandas` function `read_csv`.
3. Data that you download using an API, which makes a **request** to a data repository and **returns** requested data.

Two Key Formats

The data that you access programmatically may be returned in one of two main formats:

1. **Tabular Human-readable file:** Files that are tabular, including CSV files (Comma Separated Values) and even spreadsheets (Microsoft Excel, etc.). These files are organized into columns and rows and are "flat" in structure rather than hierarchical.
2. **Structured Machine-readable files:** Files that can be stored in a text format but are hierarchical and structured in some way that optimizes machine readability. JSON files are an example of structured machine-readable files.

Data Tip: There are non-text formatted hierarchical data structures that you will not learn in this module. One example of this is the HDF5 data model (structure).

Download Files Programmatically

Pandas function `read_csv()`

Note that you can use the `read_csv()` function from `Pandas` to import data directly into `Python` by providing a URL to the CSV file (e.g. `read_csv(URL)`).

When you programmatically read data into `Python` using `read_csv()` you are not saving a copy of your data locally, on your computer - you are importing the data directly into `Python`.

If you want a copy of that data to use for future analysis without directly importing it, you will need to export the data to your working directory using `write_csv()`.

Below is an example of directly importing data into `Python` using `read_csv()`. The data are average annual temperature in Canada from the World Bank.

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
url = "http://climatedataapi.worldbank.org/climateweb/rest/v1/country/cru/tas/year/CAN.csv"
df = pd.read_csv(url)

df.head()
```

	year	data
0	1901	-7.672419
1	1902	-7.862711
2	1903	-7.910783
3	1904	-8.155729

	year	data
4	1905	-7.547311

Human Readable Data

Notice that the data that you downloaded above using are **tabular** and thus **human-readable**. The data are organized into a tabular structure with rows and columns that you can quickly understand. `Python` can import these data into a `Pandas DataFrame` and you can work with it programmatically.

However, what happens if your data structure is more complex? For example, what if you wanted to store more information about each measured precipitation data point? Your table could get very wide very quickly making is less readable but also more computationally intensive to process.

You will learn about structured machine readable data structures later in this module, which may be hard for humans to quickly digest when you look at them but are much more efficient to process - particular as your data get large.

What is an API?

An API (Application Programming Interface) is an interface that sits on top of a computer based system and simplifies certain tasks, such as extracting subsets of data from a large repository or database.

Using Web-APIs

Web APIs allow you to access data available via an internet web interface.

Often you can access data from web APIs using a URL that contains sets of parameters that specifies the type and particular subset of data that you are interested in.

If you have worked with a database such as `Microsoft SQL Server` or `PostgreSQL`, or if you've ever queried data from a GIS system like ArcGIS, then you can compare the set of parameters associated with a URL string to a SQL query.

> *Web APIs are a way to strip away all the extraneous visual interface that you don't care about and get the data that you want.*

Why You Use Web APIs

Among other things, APIs allow us to:

- Get information that would be time-consuming to get otherwise.
- Get information that you can't get otherwise.
- Automate an analytical workflows that require continuously updated data.
- Access data using a more direct interface.

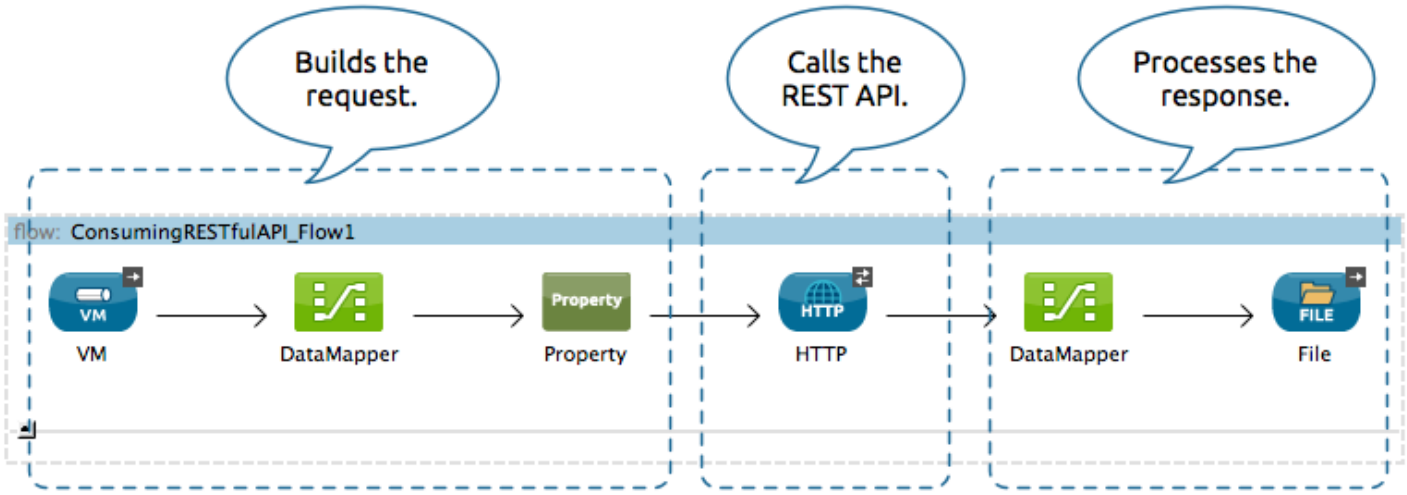
Three Parts of an API Request

When we talk about APIs, it is important to understand two key components: the request and the response. The third part listed below is the intermediate step where the request is PROCESSED by the remote server.

1. Data **REQUEST**: You try to access a `URL` in your browser that specifies a particular subset of data.
2. **Data processing:** A web server somewhere uses that url to query a specified dataset.
3. Data **RESPONSE**: That web server then sends you back some content.

The **response** may give you one of two things:

1. Some *data* or
2. An explanation of why your request *failed*



First, you create a request that queries the data repository for some data using the API. The repository gets the request and in turn processes it. Finally, it responds. If your request was a good one, the response will include the data that you are interested in. Image Source

API Endpoints

When we talk about an endpoint, we are referring to a datasource available through an API. These data may be census data, geospatial base maps, water quality or any type of data that has been made available through the API.

RESTful Web APIs

There are many different types of web APIs. One of the most common types is a `REST`, or `REST`ful, API. A *RESTful* API is a web API that uses URL arguments to specify what information you want returned through the API.

To put this all into perspective, next, you will explore a RESTful API interface example.

Colorado Population Projection Data

The Colorado Information Marketplace is a data warehouse that provides access to a wide range of Colorado-specific open datasets available via a **REST**ful API called the Socrata Open Data API (SODA)

There are lots of API *endpoints* or data sets available through this API.

- Check some of the data available on the site out in the browse section of the website.
- Check out the available demographic data.
- Colorado population projects data landing page.

One endpoint contains Colorado Population Projection data. If you click on the link to the CO Population projection data, you will see data returned in a `JSON` format.

`JSON` is a **structured, machine readable format**. You will learn more about it in the next lesson.

Population Data Request and Response

The CO population project data contain projected population estimates for *males* and *females* for every *county* in Colorado for every *year* from 1990 to 2040 for multiple *age* groups.

Phew! In the previous sentence, you just specified all of the variables stored within these data. These variables can be used to query the data! This is your data **request**.

Below, you see a small subset of the **response** that you get from a basic request with no URL parameters specified: https://data.colorado.gov/resource/tv8u-hswn.json. Notice that the **response** in this case is returned in `JSON` format.


```
[{"age":"0","county":"Adams","femalepopulation":"2404.00","fipscode":"1","malepopulation":"2354.00","totalpopulation
,{"age":"1","county":"Adams","femalepopulation":"2375.00","fipscode":"1","malepopulation":"2345.00","totalpopulation":"
,{"age":"2","county":"Adams","femalepopulation":"2219.00","fipscode":"1","malepopulation":"2413.00","totalpopulation":"
,{"age":"3","county":"Adams","femalepopulation":"2261.00","fipscode":"1","malepopulation":"2321.00","totalpopulation":"
,{"age":"4","county":"Adams","femalepopulation":"2302.00","fipscode":"1","malepopulation":"2433.00","totalpopulation":"
...
]
```


URL Parameters

Using `URL` parameters, you can define a more specific request to limit what data you get back in **response** to your API request. For example, you can query the data to only return data for Boulder County, Colorado using the RESTful call.

Data Tip: Note the **?&county=Boulder** part of the url below. That is an important part of the API request that tells the API to only return a subset of the data - where county = Boulder. `https://data.colorado.gov/resource/tv8u-hswn.json?&county=Boulder`

Like this: https://data.colorado.gov/resource/tv8u-hswn.json?&county=Boulder.

Notice that when you visit the URL above and in turn **request** the data for Boulder County, you see that now the **response** is filtered to **only include Boulder County data**.


```
[{"age":"66","county":"Boulder","femalepopulation":"649","fipscode":"13","malepopulation":"596","totalpopulation":"1
,{"age":"78","county":"Boulder","femalepopulation":"427","fipscode":"13","malepopulation":"258","totalpopulation":"685"
,{"age":"85","county":"Boulder","femalepopulation":"265","fipscode":"13","malepopulation":"110","totalpopulation":"375"
,{"age":"74","county":"Boulder","femalepopulation":"516","fipscode":"13","malepopulation":"373","totalpopulation":"889"
...
]
```


Parameters associated with accessing data using this API are documented here.

Using the SODA RESTful API

The `SODA` RESTful API also allows us to specify more complex 'queries'. Here's the API `URL` for population projections for females who live in Boulder that are between the ages of 20–40 for the years 2016–2025:

`https://data.colorado.gov/resource/tv8u-hswn.json?$where=age between 20 and 40 and year between 2016 and 2025&county=Boulder&$select=year,age,femalepopulation`


```
[{"age":"32","femalepopulation":"2007","year":"2024"}
,{"age":"35","femalepopulation":"1950","year":"2016"}
,{"age":"37","femalepopulation":"2039","year":"2019"}
,{"age":"30","femalepopulation":"2087","year":"2025"}
,{"age":"26","femalepopulation":"1985","year":"2019"}
,{"age":"22","femalepopulation":"3207","year":"2016"}
...
]
```


Click here to view the full API response.

Breaking Down an API String

Notice that the `colorado.data.gov` API URL above, starts with `data.colorado.gov` but then has various parameters attached to the end of the URL that specify the particular type of information that you are looking for.

A few of the parameters that you can see in the url are listed below:

- **The Data set itself:** /tv8u-hswn.json
- **AGE:** where=age between 20 and 40

- **YEAR:** year between 2016 and 2025
- **COUNTY:** county=Boulder
- **Columns to get:** select=year,age,femalepopulation

JSON Structured Text Format API Response

The response data that are returned from this API are in a text format, structured using `JSON`.

Data Tip: Many APIs allow you to specify the file format that you want to be returned. Learn more about how this works with the CO data warehouse here.

- JSON format
- GeoJSON format
- XML format
- CSV format

Notice that the first few rows of data returned via the query above with a `.csv` suffix look like this:

```
"age","femalepopulation","year"
"32","2007","2024"
"35","1950","2016"
"37","2039","2019"
"30","2087","2025"
"26","1985","2019"
...
```

Here is a different application of the same type of API. Here, the website developers have built a tabular viewer that we can use to look at and interact with the population data. These data are the same data that you can download using the **REST** API url string above. However, the developers have wrapped the API in a cool interface that allows us to view the data directly, online.

You will work with these data in `R` directly in the following lessons, but for now just notice how the API access works in this case.

1. Data **REQUEST**: You try to access a `URL` in your browser.
2. **Data processing:** A web server somewhere uses that url to query a specified dataset.
3. Data **RESPONSE**: That web server then sends you back some content.

Optional Challenge

Explore creating SODA API calls to the Colorado data warehouse. Go to the bottom of the page and check out each variable that you can query on.

Open JSON API Data Directly Using Pandas in Python

You can quickly open the JSON-formatted data using `read_json` in Pandas. However, notice that there are spaces in between some of the words in the URL. These spaces will yield an error. To address the spaces, you can replace them with the ascii value `%20` using `string.replace()`. The code below fixes the url so you can open the data using Pandas.

```
url = "https://data.colorado.gov/resource/tv8u-hswn.json?$where=age between 20 and 25 and year between 2020 and 2025
url = url.replace(" ", "%20")
url
```

```
## 'https://data.colorado.gov/resource/tv8u-hswn.json?$where=age%20between%2020%20and%2025%20and%20year%20between%20202
```

```
dem_data_20_25_female = pd.read_json(url)

dem_data_20_25_female.head()
```

	year	age	femalepopulation
0	2025	20	3960
1	2023	23	2673
2	2024	22	3361
3	2020	21	3649
4	2021	24	2215

```
# set year as index
dem_data_20_25_female = dem_data_20_25_female.set_index("year")
dem_data_20_25_female.head()
```

	age	femalepopulation

year	age	femalepopulation

year		
2025	20	3960
2023	23	2673
2024	22	3361
2020	21	3649
2021	24	2215

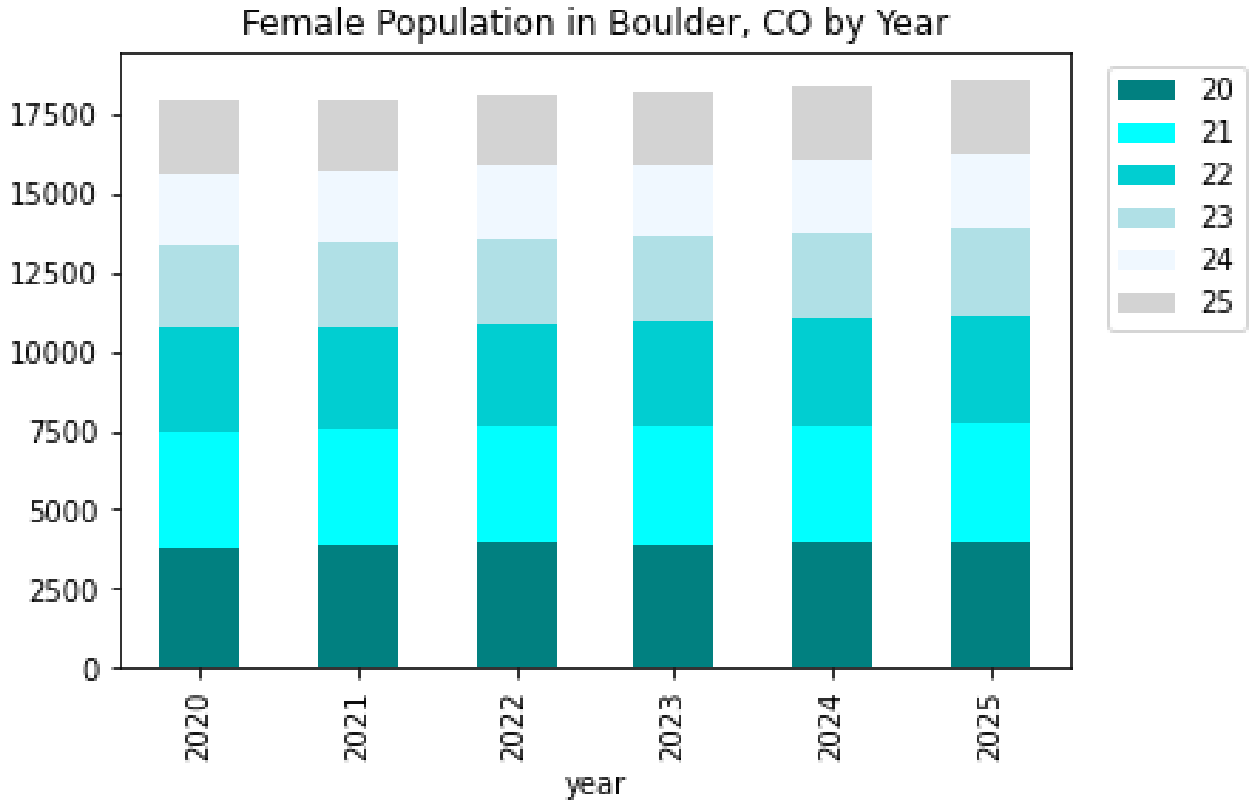
```
# Pivot the data for stacked plotting
dem_data_20_25_female_pivot = dem_data_20_25_female.pivot_table('femalepopulation',
                                                                ['year'], "age")
dem_data_20_25_female_pivot.head()
```

age	20	21	22	23	24	25
year						
2020	3831	3649	3290	2608	2263	2247
2021	3869	3666	3237	2705	2215	2265
2022	3924	3705	3255	2653	2314	2218
2023	3897	3761	3295	2673	2264	2319
2024	3937	3743	3361	2723	2294	2281

```
# Plot the data
my_colors = ["teal",
             "aqua",
             "darkturquoise",
             "powderblue",
             "aliceblue",
             "lightgrey"]

f, ax = plt.subplots()
dem_data_20_25_female_pivot.plot.bar(stacked=True,
                                     color=my_colors,
                                     ax=ax).legend(loc='upper right',
                                                   bbox_to_anchor=(1.2, 1.0))
ax.set(title="Female Population in Boulder, CO by Year")

plt.show()
```




```
# THIS will only work if you have plotly installed
# uncomment code to run - plotly is not  yet in our ea-environment but will be added
# conda install -c conda-forge plotly
# pd.options.plotting.backend = "plotly"

# # Plot the data
# my_colors = ["teal",
#              "aqua",
#              "darkturquoise",
#              "powderblue",
#              "aliceblue",
#              "lightgrey"]

# dem_data_20_25_female_pivot.plot.bar(title="Female Population in Boulder, CO by Year")
```

Additional Resources

More About JSON

- JSON tutorial

Using APIs

So, how do you learn more about APIs? Below are some resources ...

- Find APIs and read more about them
- Twitter API Documentation
- New York Times API Information
- Weather Underground API Information
- EnviroCar API Information

The documentation in the URL's above describes the different types of *requests* that you can make to the data provider. For each request `URL` you need to specify the parameters and consider the response.

Intro to JSON

Tags **Find and manage data:** apis
Updated: April 01, 2021

LEAVE A COMMENT