Applied Rest APIs I

Agenda - Schedule

1. Jupyter Notebook Preview
2. Requests Module
3. API Calls - Code Along
4. TLAB

JavaScript Object Notation (JSON) is an open-standard data format or interchange for semi-structured data. It is text-based and readable by humans and machines.
https://www.snowflake.com/guides/what-is-json

Agenda - Announcements

- Week 4 Pre-Class Quiz due 4/1
- TLAB #2 Due 4/21
- Office hours posted

Agenda - Goals

- Learn how to request data programmatically over the web
- Learn how to transform JSON files into structured data
- Get introduced to Jupyter notebooks

Fellowship Reminders

Fellowship Reminders

Now that we're on week 4, we just want to go through a few reminders related to what you should expect each week

- **Weekly Pre-Class Quizzes**
- **Slack Usage**
- **Requesting extensions**
- **Office Hours**

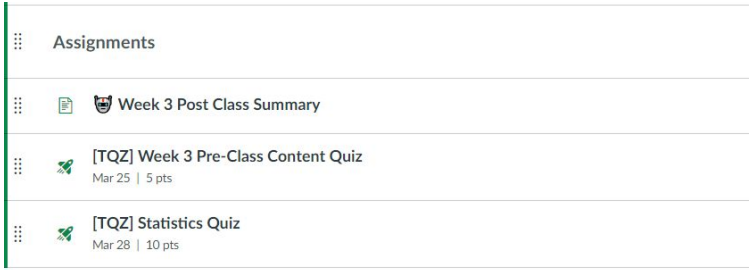
Weekly Pre-Class Quizzes

You will have **a pre-class quiz** each week that will be due on the following **Tuesday** of each week.

For example,the **Week 4 Pre-Class Quiz** was announced last Wednesday and will be due this **Tuesday**.

Unless you've requested a 48-extension on this quiz, it will close on **Tuesday 11:59 PM** and will not be able to be re-opened.

Look at the "assignments" tab of this week's module to see what's due.



We'll start trying to keep the Slack space organized by redirecting you if you send a message that might be out of this scope.

Slack Usage - General Usage

We want you to use Slack for **collaboration/communication**, but we should also try to direct messages to proper parties.

First let's iron out **general guidelines**:

- Don't use "**@here**" since it pings everyone that is currently on Slack. Generally staff will use this to make announcements.
- Any messages regarding your **grade**, your **scheduled office hours**, or anything that's directed to staff, should be sent directly to us
- You cannot ask for direct answers on your TLAB. This includes bugs. Don't treat errors as a problem, treat them as **challenges** you must solve yourself.

How to Reach out for Help

Remember, you will not **learn** a topic just by getting the answer. Treat your grade **as a secondary objective**. **At no point will an employer ask you for your TKH GPA.** We understand that stipend requirements have a GPA threshold, but you will make this threshold if you understand the topics.

Ask yourself 3 questions before reaching out to a human on coding help:

- *What does my error message tell me?*
- *Did I look at the notes/recordings for assistance?*
- *Did I read the documentation/instructions?*

We will ask you these questions as we go further into this fellowship during debugging sessions.

Slack Usage - Channel Purposes

For the remaining slack channels, please try to keep the messages focused on their intended usage:

- **data-science-fellowship-instructor-announcements:** *Announcements regarding content, labs, and due dates (don't miss messages from this channel)*
- **data-science-fellowship-all:** *Track wide collaboration, outreach between cohorts; group study sessions, installation help*
- **data-science-fellowship-a/b:** *Questions during class, support needed to get to right zoom session, study sessions, installation help*
- **random:** *memes, resources, etc*

Requesting Extensions

If you'd like to request an extension email **your student success specialist**, **your instructor**, and **your TA** a quick 1-2 sentence email.

Could be as simple as *"I would like to use my extension, thanks!"*

We will respond with a confirmation email. Going forward, **all requests must be made before the due date.**

Office Hours

Staff within your **own cohort** will only be able to hold office hours with you. This is to keep our availability consistent.

- **Cohort A**: Farukh & Malachi
- **Cohort B**: Anil & Maurice

Cohort A, please check out our office hours and book with **Malachi and me(Farukh)**.

Fellowship Non-Reminders

Lastly, we just want to end this batch of reminders with some words of encouragement:

- Only use **yourself as a reference point for progress**. If you start playing the comparison game with other people, it can quickly hinder your efforts.
- You are all **learning a lot in a short period of time**! As a self-taught learner, this progression of topics represents months if not years of content reduced down to a concentrated fellowship.

Jupyter Notebooks

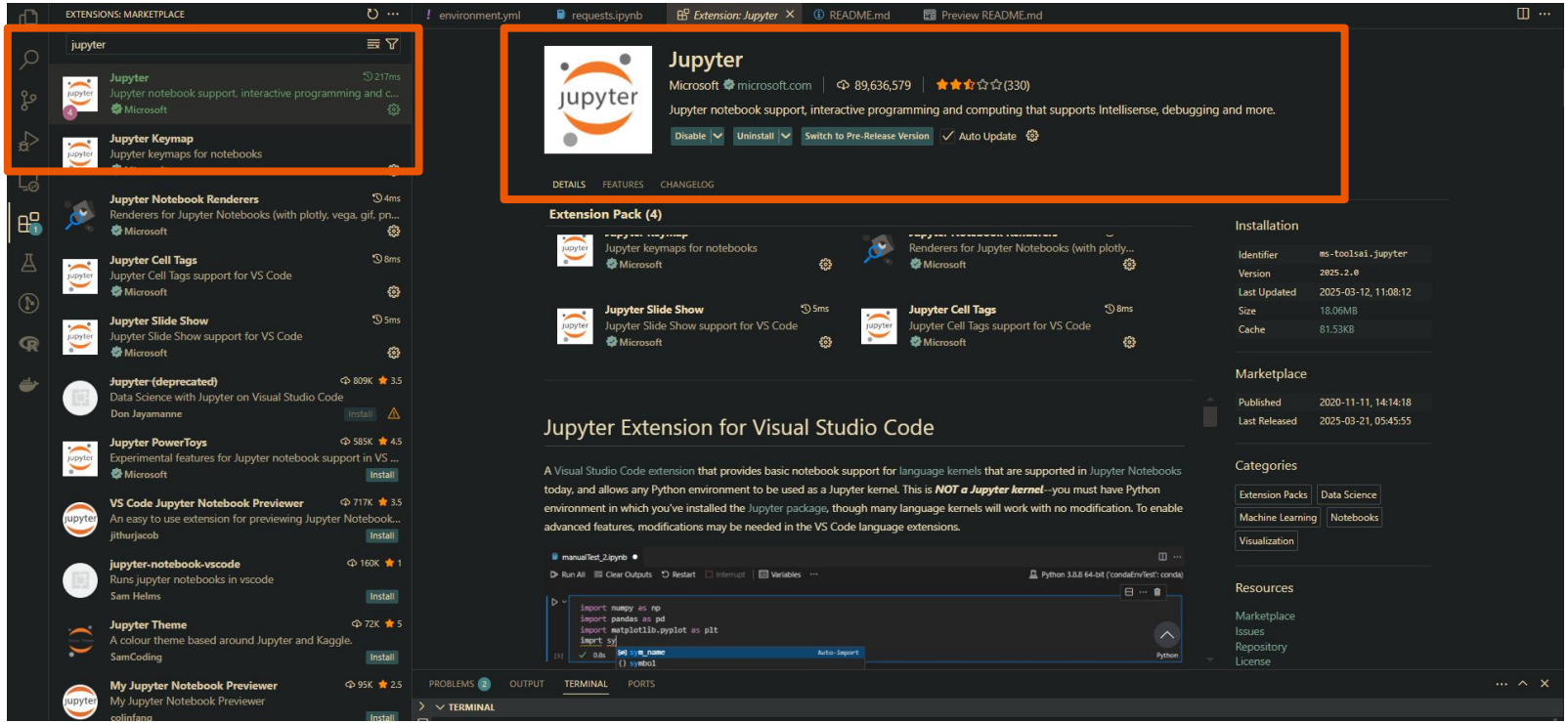
Intro to Jupyter Notebooks

As we get closer to performing **data analysis,** we should become familiar with tools that we can use to generate reports and observe intermediate output from Python code.

Jupyter Notebook is one of those tools. We can directly create jupyter notebooks in our VSCode in order to run code **piece by piece** and also present **visualizations**!

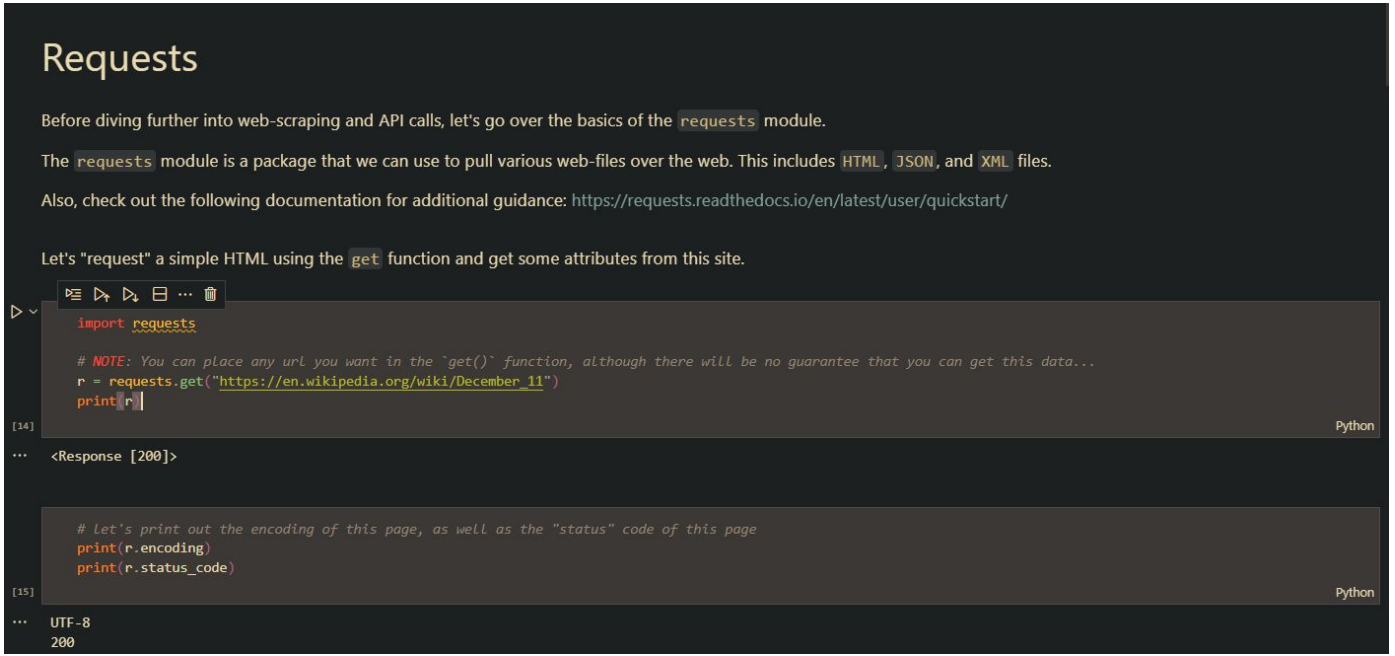
We will demonstrate how to begin working with jupyter notebooks. We won't be pausing for debugging help, so please just observe and then try this process yourself during lab (looking back to the slides).

1) Type in "Jupyter" in your "Widgets" search bar in VSCode



2) Then install the Jupyter widget will enable you to create notebooks straight from VSCode

Since we've already created a conda environment, we do not need to worry about installing the **notebooks** package. Instead, we should first enable the Jupyter widget on VSCode.



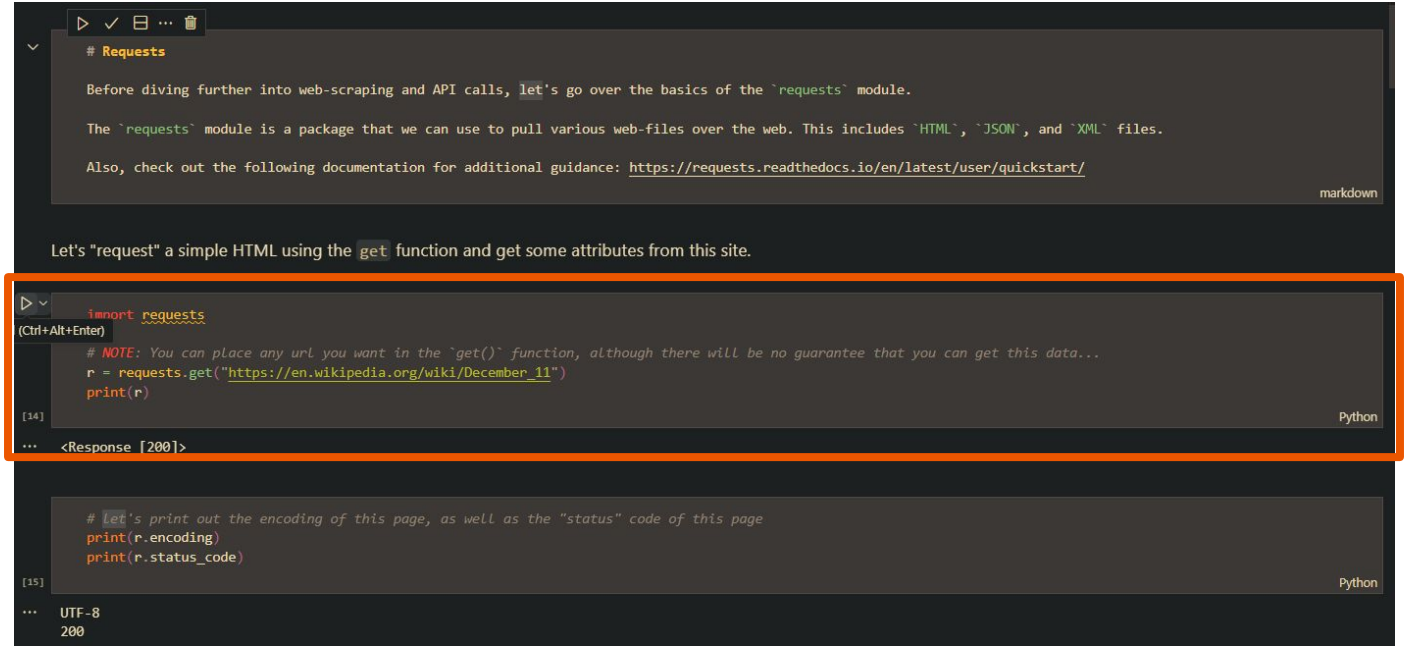
From there, we can create a Jupyter notebook by simply creating a new file in VSCode and give this an extension of **.ipynb.**

However since we're working off of **requests.ipynb** we can simply open this file.

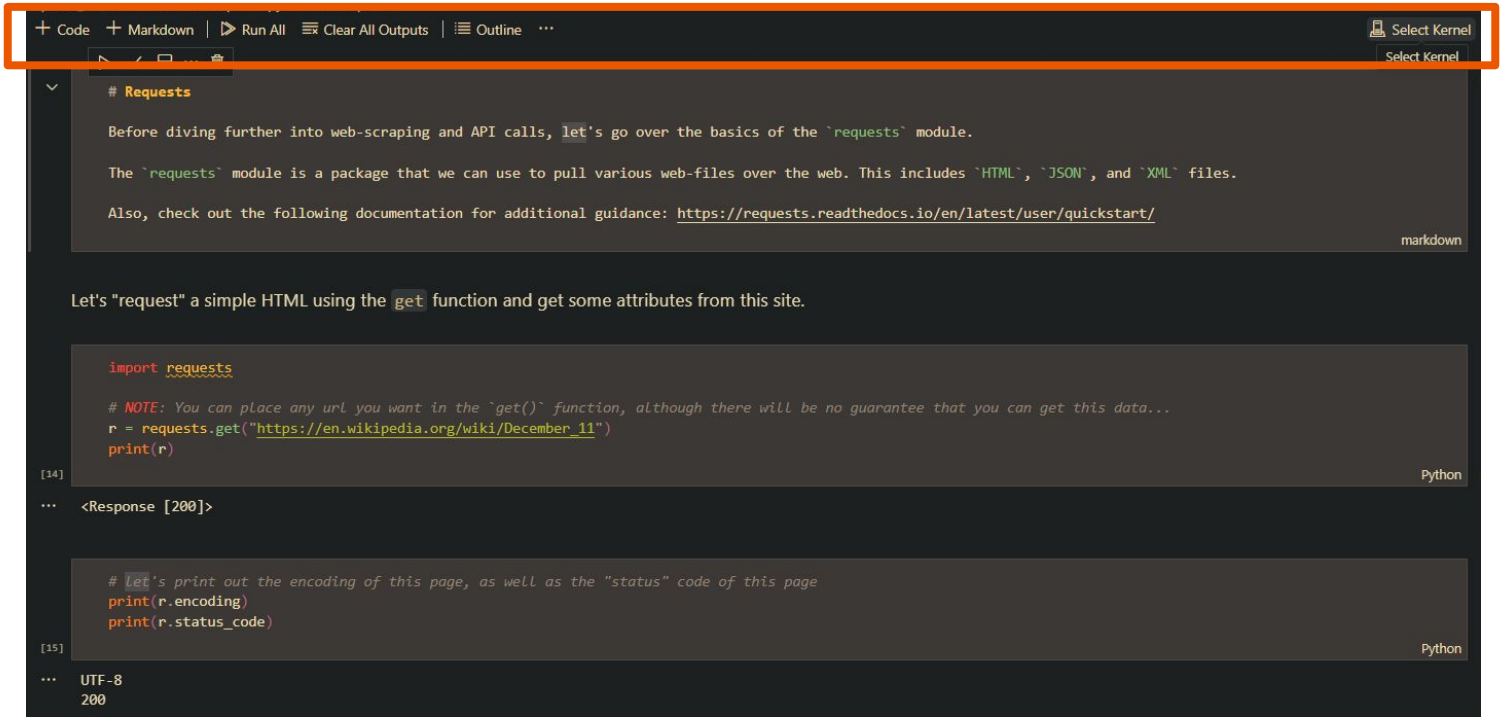
Before we go about running this code, let's note the structure. We can modify text in the markdown boxes by double clicking anywhere we see text.

This is where Jupyters power as a reporting tool is revealed. We can type in prescriptions/explanations for data analysis in these boxes.

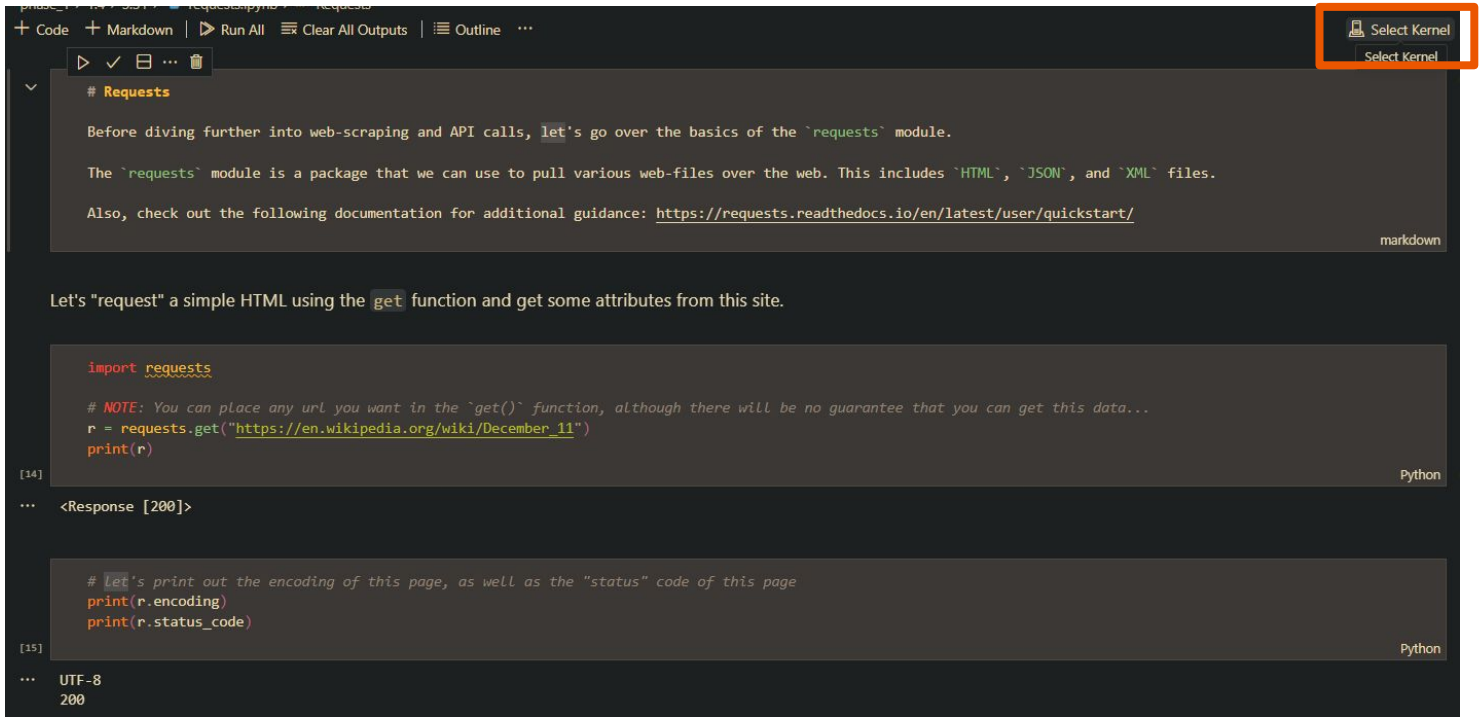
You must run code sequentially (i.e. run the code block by block) or else your notebook might not work



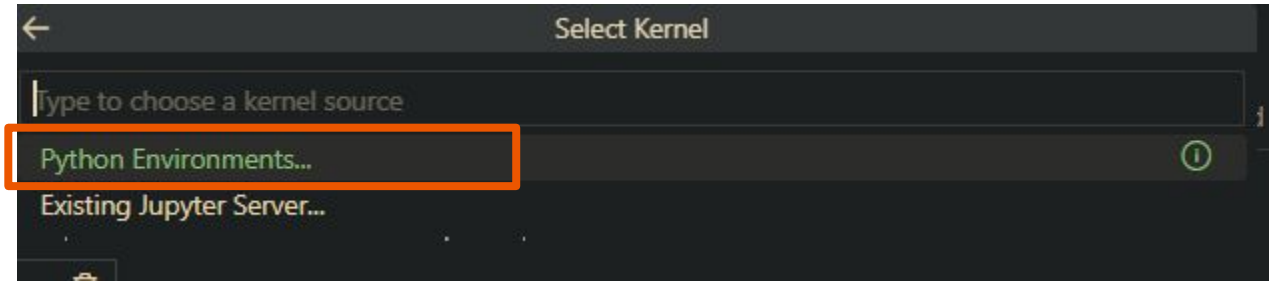
Below our markdown box we have a code-block. We can insert Python code into this code block, run our code by clicking on the “play button” on the left-hand side, and our **outputs will be presented in the space below**!



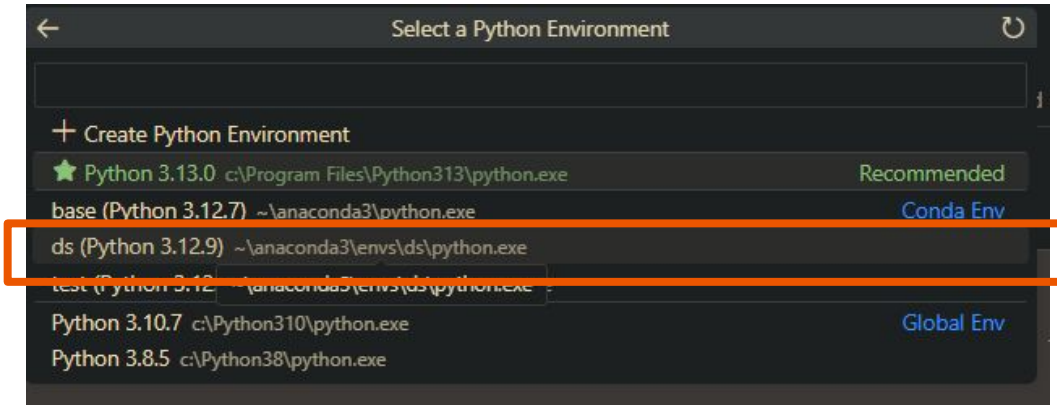
Note the top-menu as well. We invite you to explore these options. One of the best habits you can start forming is the **ability to experiment with novel systems.** Click around and see what happens, don't be afraid to ruin the file, we can always redownload it.



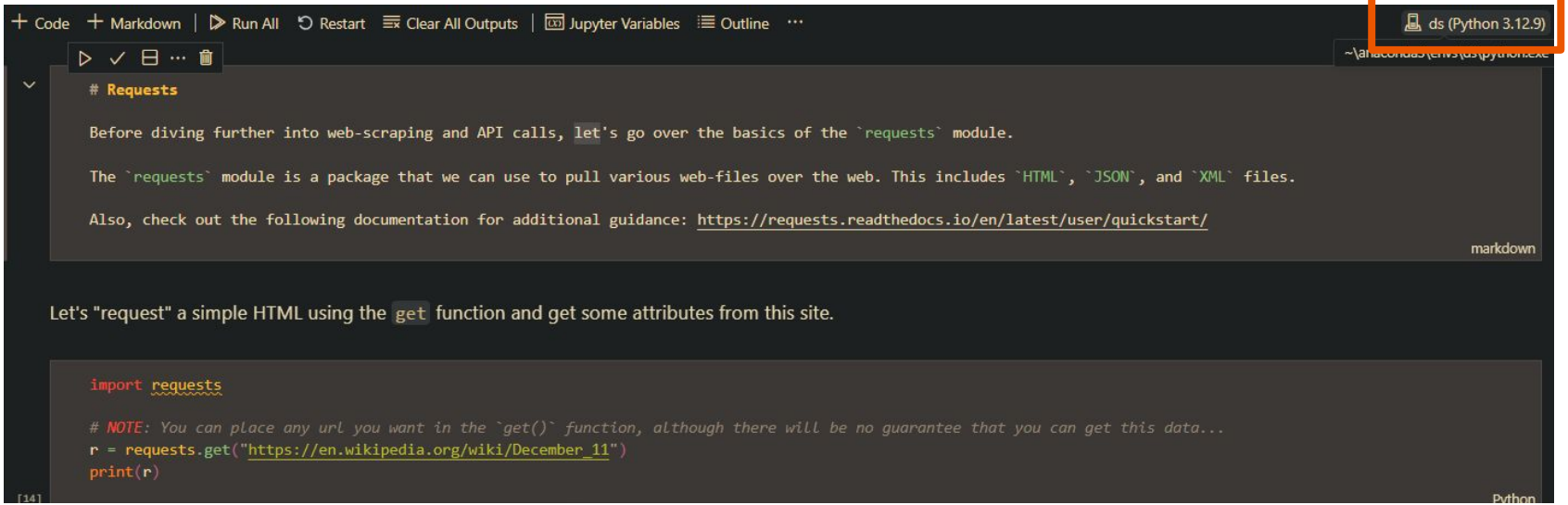
Let's point our attention to the button at the top-right that states "*Select Kernel*." Think of this as the **engine** of Python. Without a selected kernel your code will not work.



By clicking on “Select Kernel” we will be brought to a drop down menu that gives us two categories of environments. Let’s select “Python Environments.”



Finally we will be brought to a selection of conda and global environments. Select the environment that you've created last week "**ds**."



Verify that you see “ds” in the top-right and you’re done! Let’s get coding.

Choosing the Right Tool

You should not be creating notebooks unless you are doing data science (EDA, ML) which will be shared with us (or your future stakeholders).

Remember! Part of being a **good** technologist is being able to discern which tools are right for a specific task.

For now, we will guide you through what kind of files you should be using, but eventually (**Phase 2**) you will have to be able to make this selection yourself. Follow this rule of thumb:

- **Python Modules:** Any functionality that should be run **automatically** (on a schedule) or **integrated into a larger system**.
- **Jupyter Notebooks**: Making **tutorials, reports, or code** that you will share with a non-technologist stakeholder.

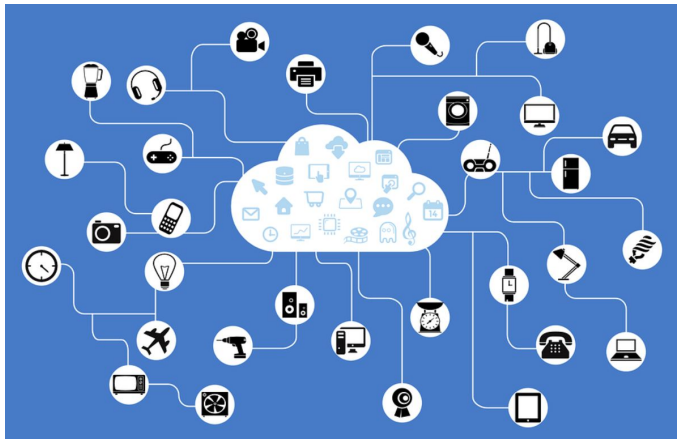
Requests Module

Requests Module

We've already discussed how we can access data that is **local** to our computer.

While this is important, this isn't the full picture.

Where do you think a large subset of data exists in the real world?

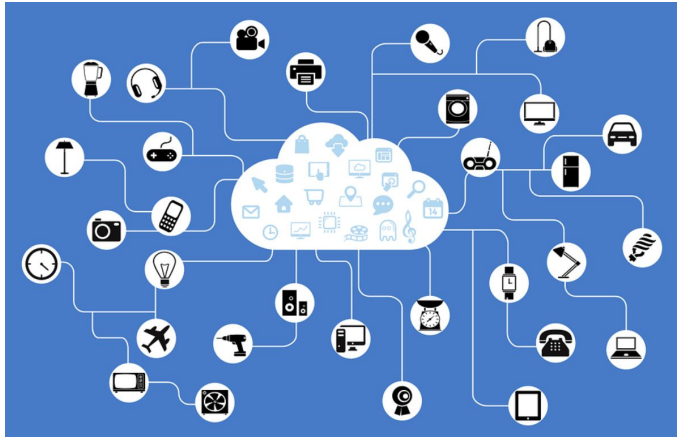


Requests Module

We've already discussed how we can access data that is **local** to our computer.

While this is important, this isn't the full picture.

Where do you think a large subset of data exists in the real world?



If it's not on a **private server**, then on the **internet.**

But how do we actually go about requesting data over the internet?

Requests Module

As it turns out, getting data over the internet is actually quite simple!

All we need is the appropriate package to handle **HTTP requests**. Access the `requests.ipynb` notebook and run the first code-block.

```
r = requests.get("https://www.scrapethissite.com/pages/simple/")
```

```
r = requests.get("https://www.scrapethissite.com/pages/simple/")

print(r.encoding)
print(r.status_code)
```

The ‘get’ function itself is an **HTTP method** that “gets” resources located in a **specific URL**.

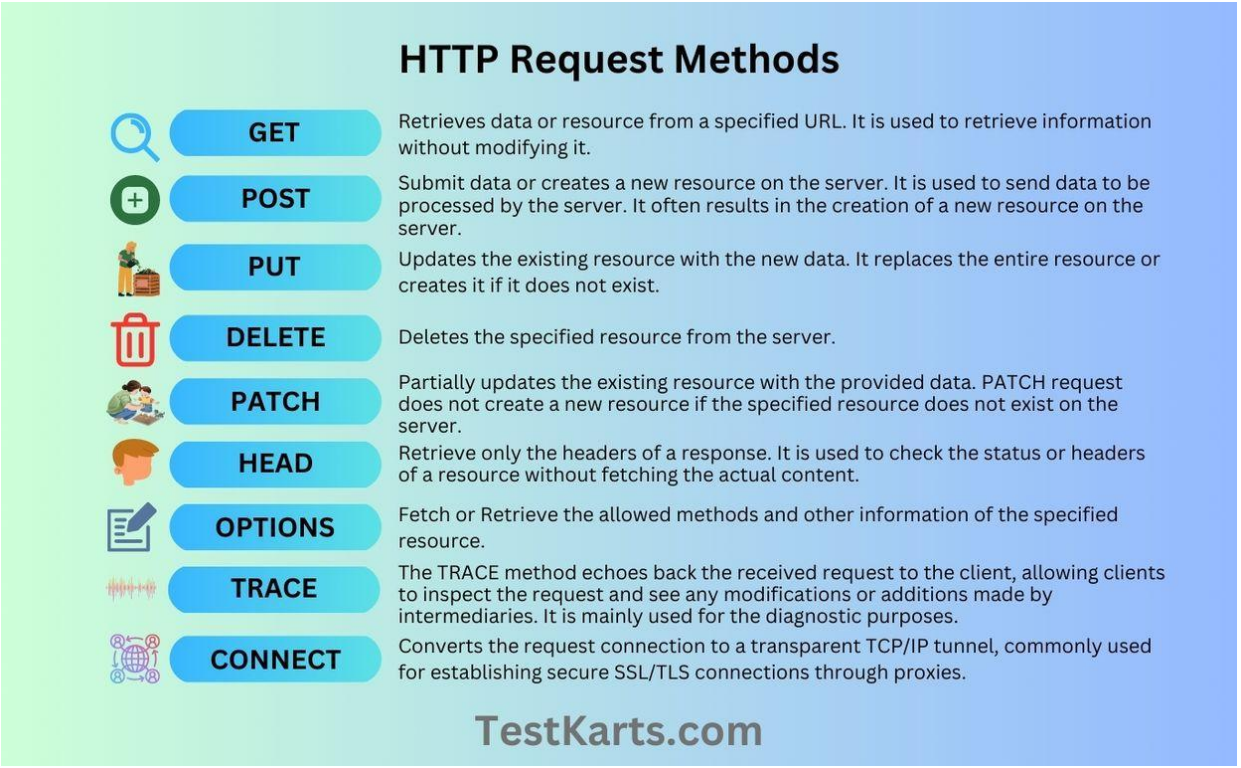
Note that this URL could be any resource that you want!

```
r = requests.get("https://en.wikipedia.org/wiki/Akira_Toriyama")

print(r.encoding)
print(r.status_code)
```

For example, if you're doing knowledge extraction on wikipedia, you could potentially throw in a wikipedia article.

```
r = requests.get("https://pokeapi.co/api/v2/pokemon/pikachu")

print(r.encoding)
print(r.status_code)
```

Or, as you will see later on, an **API resource.**



"Get" is **not the only method available to us**, however, for our purposes, we will strictly focus on this method as we are not able to modify resources that do not belong to us.

```
r = requests.get("https://www.scrapethissite.com/pages/simple/")

print(r.encoding)
print(r.status_code)
```

When running the following code-block, what do you notice gets printed?

```
r = requests.get("https://www.scrapethissite.com/pages/simple/")

print(r.encoding)
print(r.status_code)
```

In the first print statement we get some sort of encoding (*something akin to the language this website is in*).

But what about that code “200”, **what exactly does this indicate?**

HTTP Status Codes

Whenever we access a web-page, we receive a **status code** which signifies what was result of our request.

Since we got “200”, what does that indicate about our request?

200 indicates that the website honored our request, **and gave us back the information we wanted.**

If we ever get a **400** error, this usually **indicates that we were blocked**

A **500** error usually indicates that there's an **internal problem with your DNS provider or the website itself.**

Success

```
r = requests.get("https://www.scrapethissite.com/pages/simple/")

print(r.text)
```

Just like yesterday, we are not primarily interested in the metadata of this object, **but rather the data itself**.

To access this, we utilize the “text” attribute. What do you notice gets printed out?

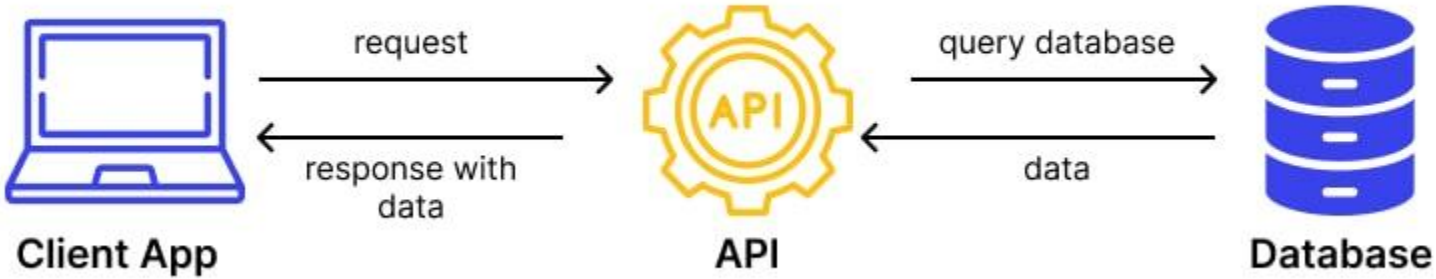
```
r = requests.get("https://www.scrapethissite.com/pages/simple/")
```

```
print(r.text)
```

```
'<!doctype html>\n<html lang="en">\n  <head>\n    <meta charset="utf-8">\n    <title>Countries of the World: A Simple Example
```

We get a string that represents the **websites HTML file**.

Note: this is semi-structured data! Accessing what we want will be **difficult**.
We will explore ways to parse this data next week.



Instead of trying to parse this HTML, let's shift our attention to API calls and utilize the **requests module** to programmatically access an external database.

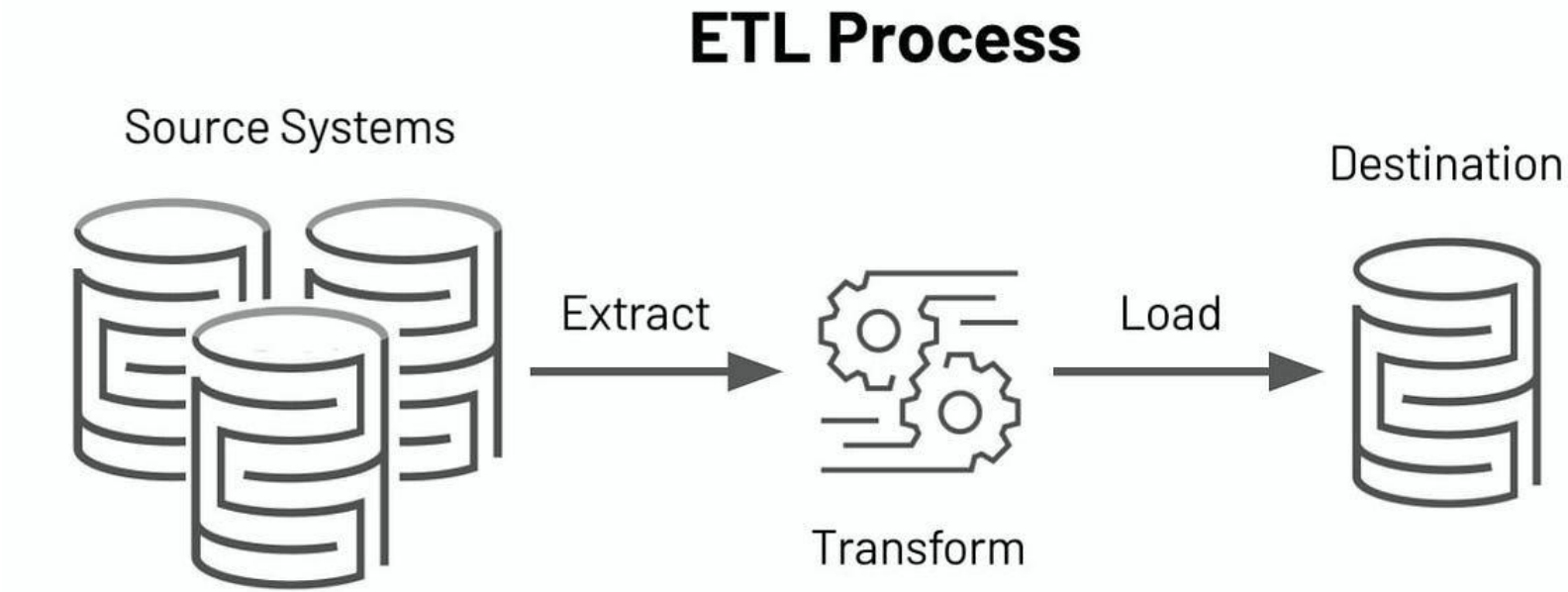
Requesting Data

Requests & API's - ETL/ELT Pipelines

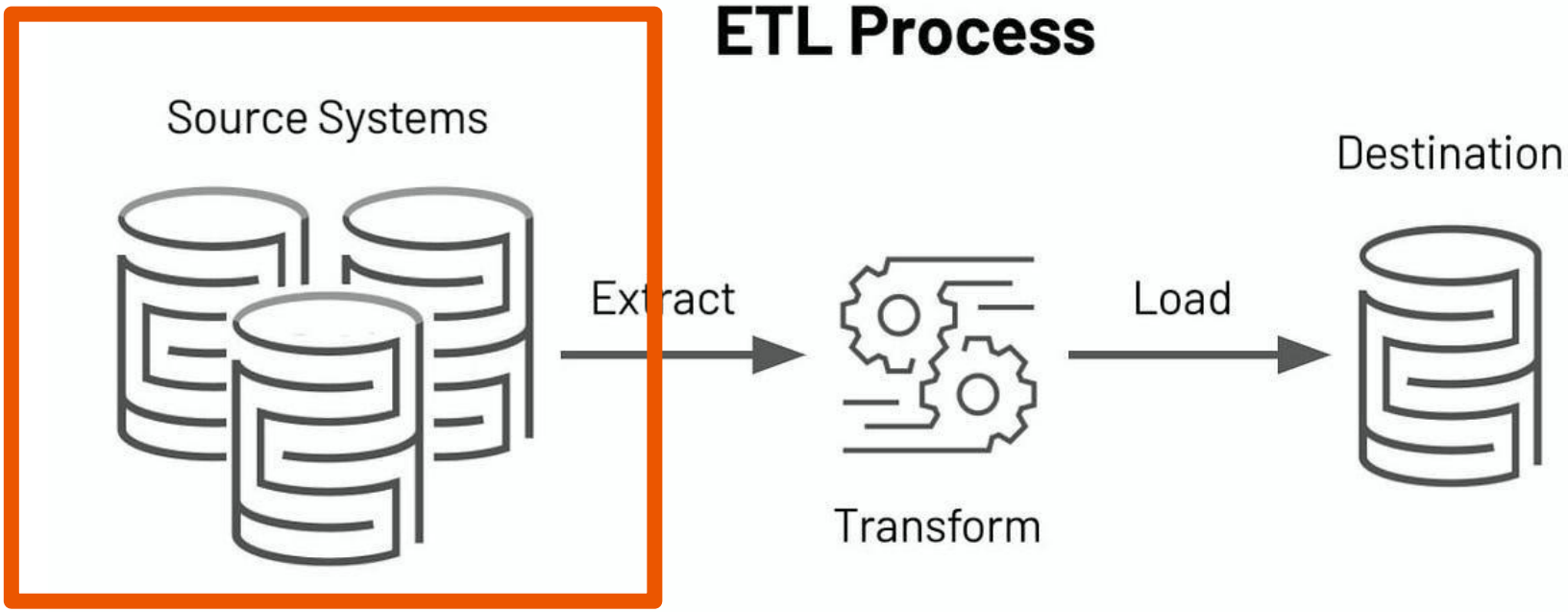
In a classic data analysis/engineering workflow, we usually set up pipelines in order to create a **database** for analysis/learning.

While classic pipeline tools are usually located on some **cloud environment**, we can still begin discussing the process of creating a **dataset from a local environment.**

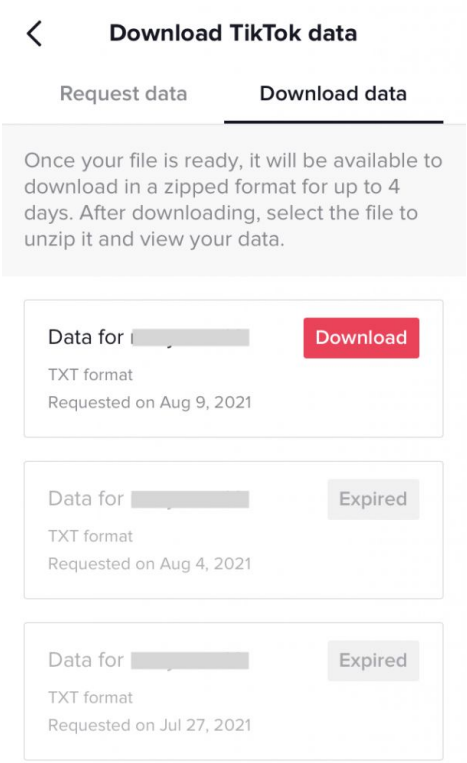
Remember, our goal is to get from **semi/unstructured data formats** to **structured formats**. Only then can we begin data analysis!

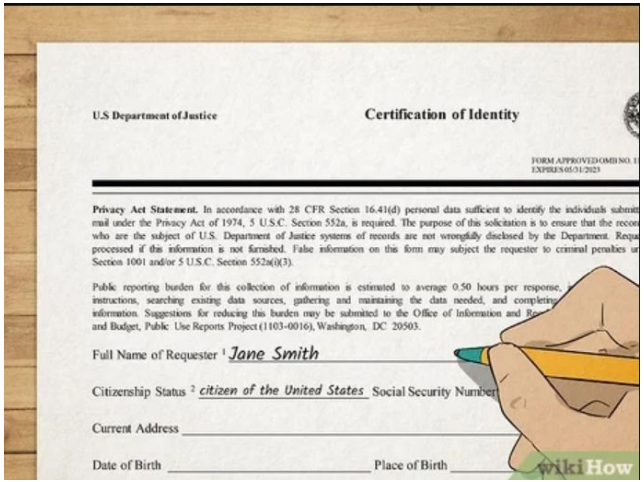


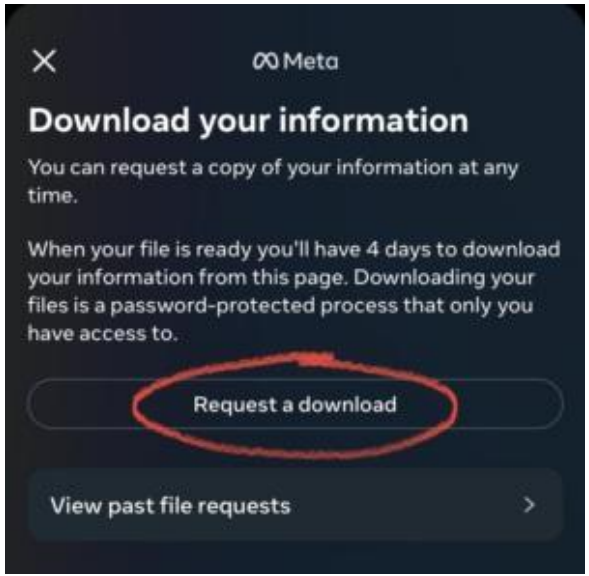
One classic format that we will learn about is the **extract-transform-load** (**ETL**) pipeline. We will skip details for now, but think of this as the process of **collecting** data, applying **transformations**, and finally, **saving it.**



This brings up a fundamental question however… how do we **extract data** outside of basic CSV files?







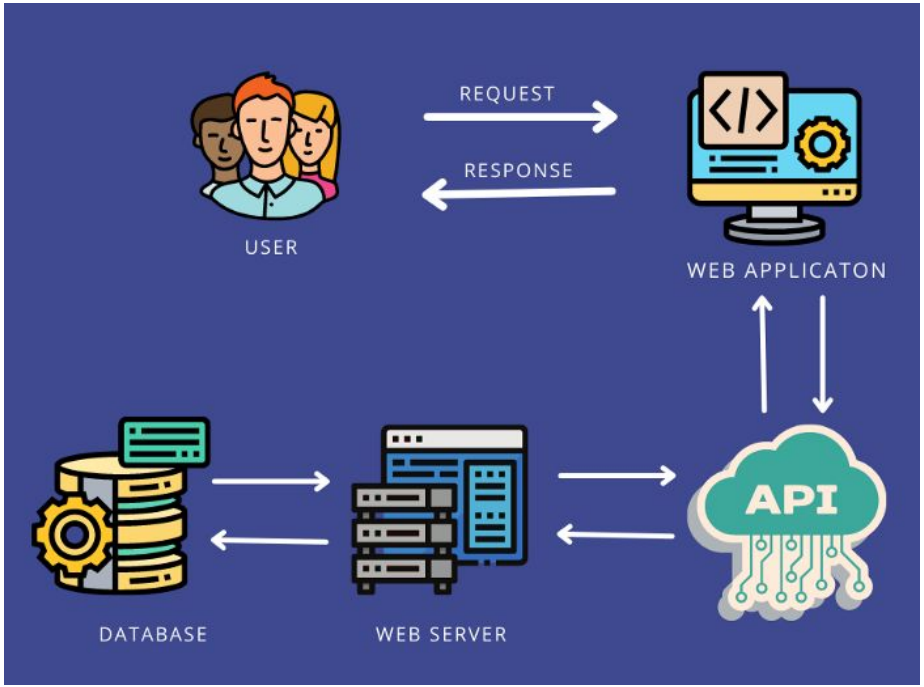
As we discussed yesterday, data is usually not readily available to us in a convenient and structured format. Instead, we usually have to **purchase/access/collect** our own data.

This data is usually accessed over the internet via **API calls**.

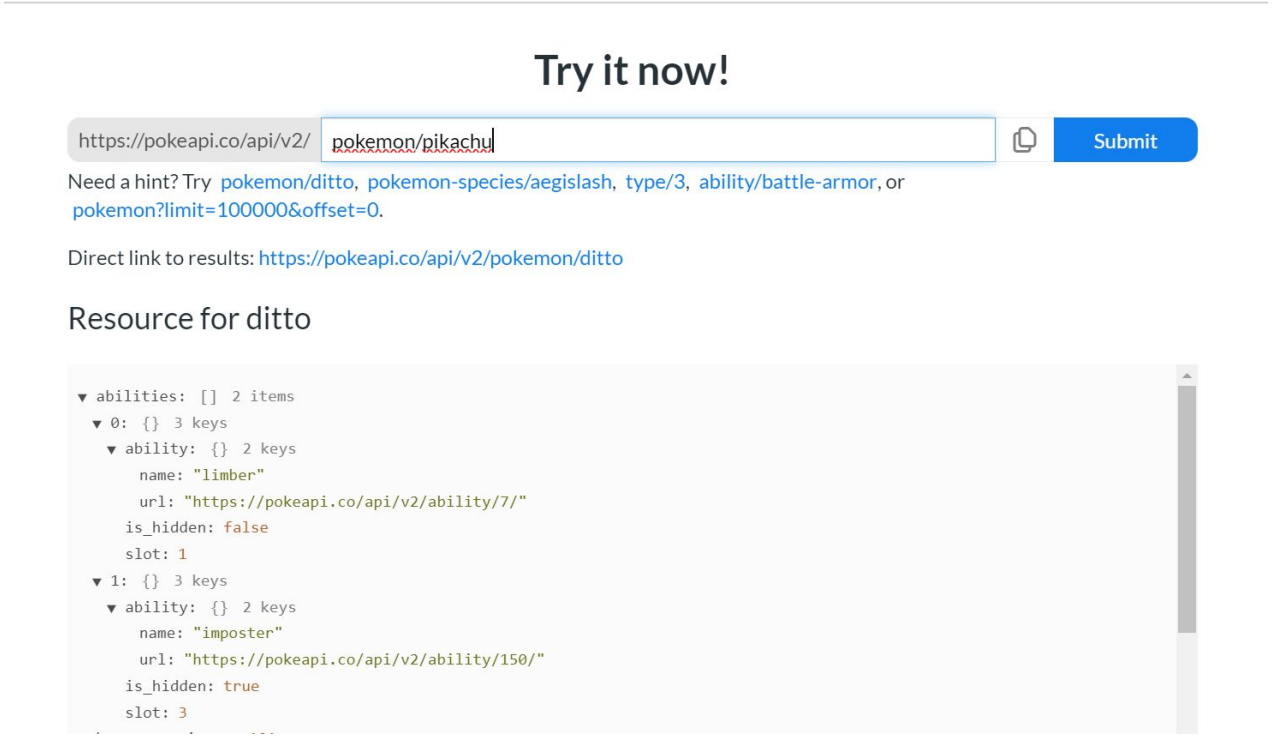
The term “API” can be used in multiple contexts:

pandas API:
An interface to use when coding in pandas

REST API:
An interface when requesting data over the web



More specifically, the **RESTful Web APIs.** This describes a programmatic way to “***ask respectfully***” for data.



Let's take a look at an example of a simple web API to request **Pokemon data**: https://pokeapi.co/.

Notice that we get some data about our pokemon located in a JSON object.

{"abilities":[{"ability":{"name":"static","url":"https://pokeapi.co/api/v2/ability/9/"},"is_hidden":false,"slot":1},{"
rod","url":"https://pokeapi.co/api/v2/ability/31/"},"is_hidden":true,"slot":3}],"base_experience":112,"cries":
{"latest":"https://raw.githubusercontent.com/PokeAPI/cries/main/cries/pokemon/latest/25.ogg","legacy":"https://raw.git
cy/25.ogg"},"forms":[{"name":"pikachu","url":"https://pokeapi.co/api/v2/pokemon-form/25/"}],"game_indices":[{"game_ind
{"name":"red","url":"https://pokeapi.co/api/v2/version/1/"}},{"game_index":84,"version":{"name":"blue","url":"https://
{"game_index":84,"version":{"name":"yellow","url":"https://pokeapi.co/api/v2/version/3/"}},{"game_index":25,"version":
{"name":"gold","url":"https://pokeapi.co/api/v2/version/4/"}},{"game_index":25,"version":{"name":"silver","url":"https
{"game_index":25,"version":{"name":"crystal","url":"https://pokeapi.co/api/v2/version/6/"}},{"game_index":25,"version"
{"name":"ruby","url":"https://pokeapi.co/api/v2/version/7/"}},{"game_index":25,"version":{"name":"sapphire","url":"htt
{"game_index":25,"version":{"name":"emerald","url":"https://pokeapi.co/api/v2/version/9/"}},{"game_index":25,"version"
{"name":"firered","url":"https://pokeapi.co/api/v2/version/10/"}},{"game_index":25,"version":{"name":"leafgreen","url"
{"game_index":25,"version":{"name":"diamond","url":"https://pokeapi.co/api/v2/version/12/"}},{"game_index":25,"version
{"name":"pearl","url":"https://pokeapi.co/api/v2/version/13/"}},{"game_index":25,"version":{"name":"platinum","url":"h
{"game_index":25,"version":{"name":"heartgold","url":"https://pokeapi.co/api/v2/version/15/"}},{"game_index":25,"versi
{"name":"soulsilver","url":"https://pokeapi.co/api/v2/version/16/"}},{"game_index":25,"version":{"name":"black","url":
{"game_index":25,"version":{"name":"white","url":"https://pokeapi.co/api/v2/version/18/"}},{"game_index":25,"version":
2","url":"https://pokeapi.co/api/v2/version/21/"}},{"game_index":25,"version":{"name":"white-2","url":"https://pokeapi
[{"item":{"name":"oran-berry","url":"https://pokeapi.co/api/v2/item/132/"},"version_details":[{"rarity":50,"version":
{"name":"ruby","url":"https://pokeapi.co/api/v2/version/7/"}},{"rarity":50,"version":{"name":"sapphire","url":"https:/
{"name":"emerald","url":"https://pokeapi.co/api/v2/version/9/"}},{"rarity":50,"version":{"name":"diamond","url":"https
{"rarity":50,"version":{"name":"pearl","url":"https://pokeapi.co/api/v2/version/13/"}},{"rarity":50,"version":
{"name":"platinum","url":"https://pokeapi.co/api/v2/version/14/"}},{"rarity":50,"version":{"name":"heartgold","url":"h

If you'd like to see the **raw JSON object**, access the following link:
https://pokeapi.co/api/v2/pokemon/pikachu

Notice the pattern of this link "https://apiname.co/api/v2/info"
We will see this applied to financial data later.

Requests & APIs

An API call is basically a website **URL** that **points us to a specific resource.**

While all API calls contain individual URL patterns they all follow a general pattern.

For example, let's take a look at the Pokemon API URL.

https://pokeapi.co/api/v2/pokemon/pikachu

API Calls

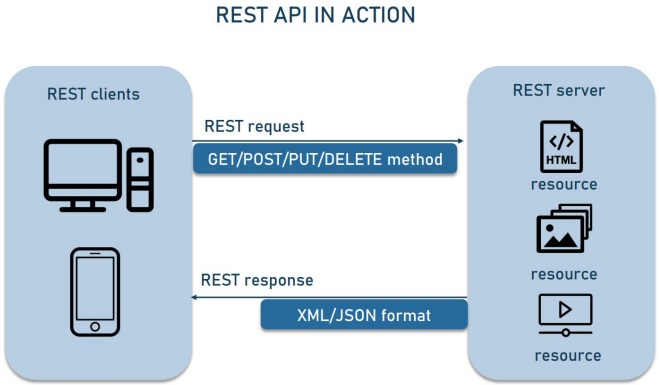
An Aside - The “REST” API

There are different types of API protocols and architectures, however, we will only focus on the **REST** (*Representational State Transfer*) API as its the most common architecture for requesting data over the web.

The REST architecture transfers a **representation** of the **state of the resource** to the **requester**.

This information comes in several formats: **JSON**, HTML, XML, or plain text.

A **client** is the party making the request, whereas the **server** is the party that resolves this request (ATM -> Bank; Computer -> Gmail)

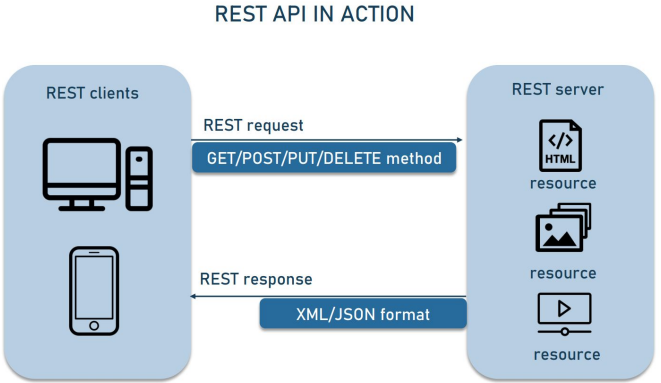


An Aside - The “REST” API

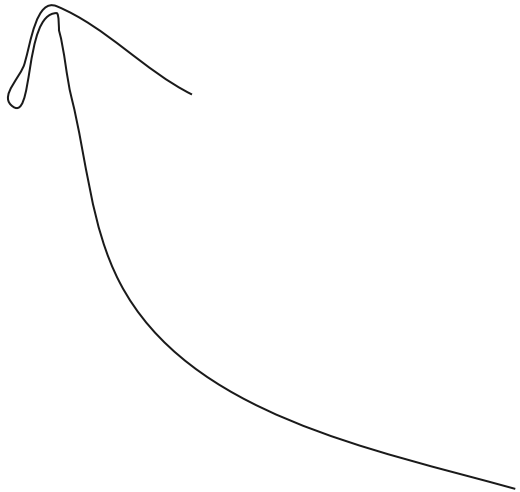
A **client** is the party making the request, whereas the **server** is the party that resolves this request (ATM -> Bank; Compter -> Gmail)

You can also design your own REST APIs! However, for it to truly be considered RESTful, it must conform to the following criteria:

- **Client-Server architecture** with requests managed through HTTP
- **Stateless communication**, meaning no client information is stored between requests
- **Cached data streamlines communication**
- **Uniform interface** that describes resources and errors
- **Layered system** to efficiently retrieve data



https://pokeapi.co/api/v2/pokemon/pikachu

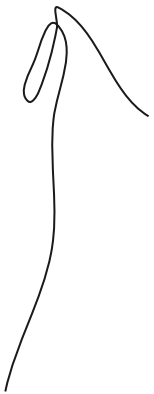


Notice that we **first specify the API** that we are requesting data from.

https://pokeapi.co/api/v2/pokemon/pikachu

Afterwards, we indicate what “kind” of resource we want. In this case, it is the data on specific pokemon.

https://pokeapi.co/api/v2/pokemon/pikachu



And finally, we specify which specific “entity” do we want information about.

Once again, **all API calls will have their own unique URLs** (just like all websites have their own URL), **however there is a pattern to this call!**

https://api.polygon.io/v1/open-close/AAPL/2022-11-08?adjusted=True

https://pokeapi.co/api/v2/pokemon/pikachu

While our stock data is a lot more granular, notice that it is fundamentally the same structure: **API/DB/resource**

For example, take a look at this stock data API where we can make **requests for live financial data**. Compare it to the pokemon API that we just used.

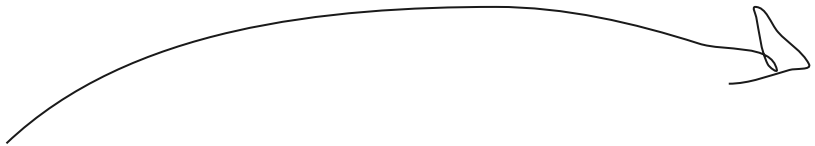
Notice the similarities!

{"abilities":[{"ability":{"name":"static","url":"https://pokeapi.co/api/v2/ability/9/"},"is_hidden":false,"slot":1},{"
rod","url":"https://pokeapi.co/api/v2/ability/31/"},"is_hidden":true,"slot":3}],"base_experience":112,"cries":
{"latest":"https://raw.githubusercontent.com/PokeAPI/cries/main/cries/pokemon/latest/25.ogg","legacy":"https://raw.git
cy/25.ogg"},"forms":[{"name":"pikachu","url":"https://pokeapi.co/api/v2/pokemon-form/25/"}],"game_indices":[{"game_ind
{"name":"red","url":"https://pokeapi.co/api/v2/version/1/"}},{"game_index":84,"version":{"name":"blue","url":"https://
{"game_index":84,"version":{"name":"yellow","url":"https://pokeapi.co/api/v2/version/3/"}},{"game_index":25,"version":
{"name":"gold","url":"https://pokeapi.co/api/v2/version/4/"}},{"game_index":25,"version":{"name":"silver","url":"https
{"game_index":25,"version":{"name":"crystal","url":"https://pokeapi.co/api/v2/version/6/"}},{"game_index":25,"version"
{"name":"ruby","url":"https://pokeapi.co/api/v2/version/7/"}},{"game_index":25,"version":{"name":"sapphire","url":"htt
{"game_index":25,"version":{"name":"emerald","url":"https://pokeapi.co/api/v2/version/9/"}},{"game_index":25,"version"
{"name":"firered","url":"https://pokeapi.co/api/v2/version/10/"}},{"game_index":25,"version":{"name":"leafgreen","url"
{"game_index":25,"version":{"name":"diamond","url":"https://pokeapi.co/api/v2/version/12/"}},{"game_index":25,"version
{"name":"pearl","url":"https://pokeapi.co/api/v2/version/13/"}},{"game_index":25,"version":{"name":"platinum","url":"h
{"game_index":25,"version":{"name":"heartgold","url":"https://pokeapi.co/api/v2/version/15/"}},{"game_index":25,"versi
{"name":"soulsilver","url":"https://pokeapi.co/api/v2/version/16/"}},{"game_index":25,"version":{"name":"black","url":
{"game_index":25,"version":{"name":"white","url":"https://pokeapi.co/api/v2/version/18/"}},{"game_index":25,"version":
2","url":"https://pokeapi.co/api/v2/version/21/"}},{"game_index":25,"version":{"name":"white-2","url":"https://pokeapi
[{"item":{"name":"oran-berry","url":"https://pokeapi.co/api/v2/item/132/"},"version_details":[{"rarity":50,"version":
{"name":"ruby","url":"https://pokeapi.co/api/v2/version/7/"}},{"rarity":50,"version":{"name":"sapphire","url":"https:/
{"name":"emerald","url":"https://pokeapi.co/api/v2/version/9/"}},{"rarity":50,"version":{"name":"diamond","url":"https
{"rarity":50,"version":{"name":"pearl","url":"https://pokeapi.co/api/v2/version/13/"}},{"rarity":50,"version":
{"name":"platinum","url":"https://pokeapi.co/api/v2/version/14/"}},{"rarity":50,"version":{"name":"heartgold","url":"h

Let's shift our focus back to the actual data however.

So now that we have a handle on how to "request" data using an API call, where do we actually put this URL? (**think back to the requests module)**

```
r = requests.get("https://pokeapi.co/api/v2/pokemon/pikachu")

data = r.text

print(data[0])
```



{

Notice that treating this like a "string" will not allow us to access individual objects!

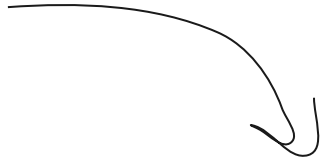
In this case we can simply extract this data format using the `requests` module!

Except this time, it would be insufficient to save this as a "text" file

```
r = requests.get("https://pokeapi.co/api/v2/pokemon/pikachu")

data = r.json()
```

Now we can access data programmatically.

```
print(data[“abilities”])
```



```
[{'ability': {'name': 'static', 'url': 'https://pokeapi.co/api/v2/ability/9/'}, 'is_hidden': False, 'slot': 1},
```

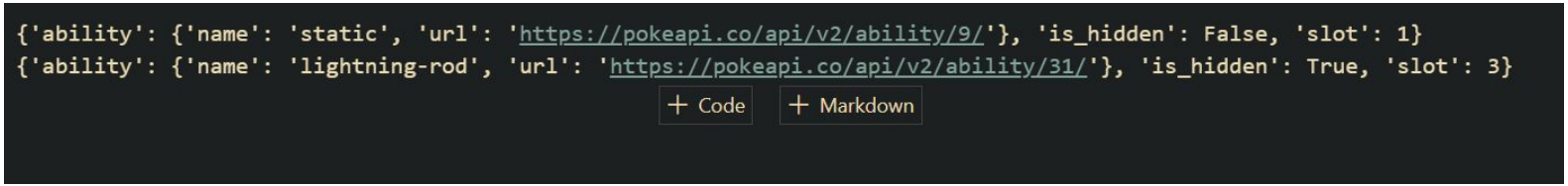
Instead, we utilize the `.json()` method in order to convert this into an object which we can then access like a regular Python data-structure.

```
r = requests.get("https://pokeapi.co/api/v2/pokemon/pikachu")

data = r.json()

for ab in data["abilities"]:
    print(ab)
```

Note that challenge of interacting with this object is a test of your ability to code.

Us just showing you lines of code on how to work with a JSON object will not be sufficient.

Instead let's take a look at a real exercise on some stock data.



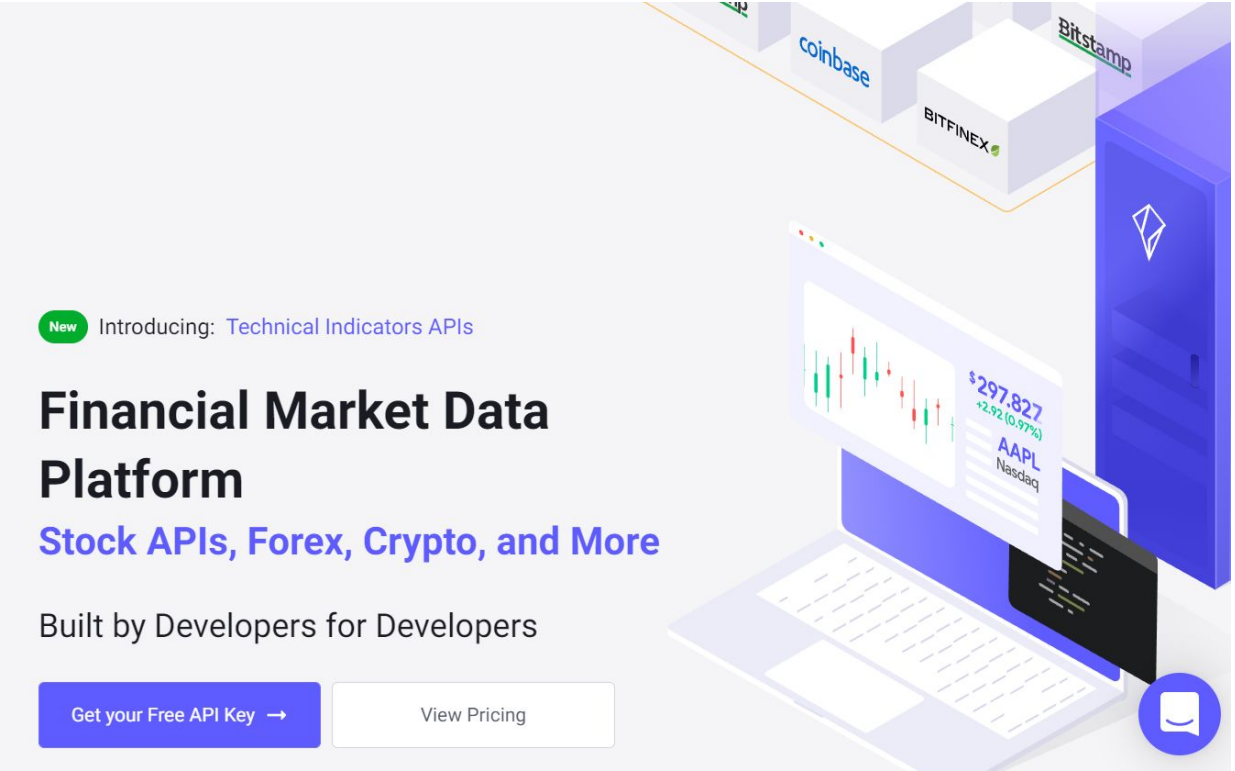
API Access Exercise

API Exercise

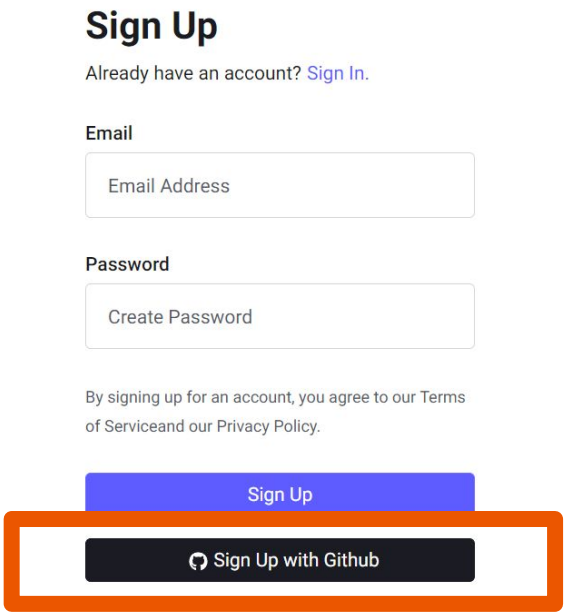
More often than not, API requests are only authorized via some API-key that was either **bought** for you through your company or was offered **freely**.

Let's go through an exercise where we request "**privileged**" information using an API key.

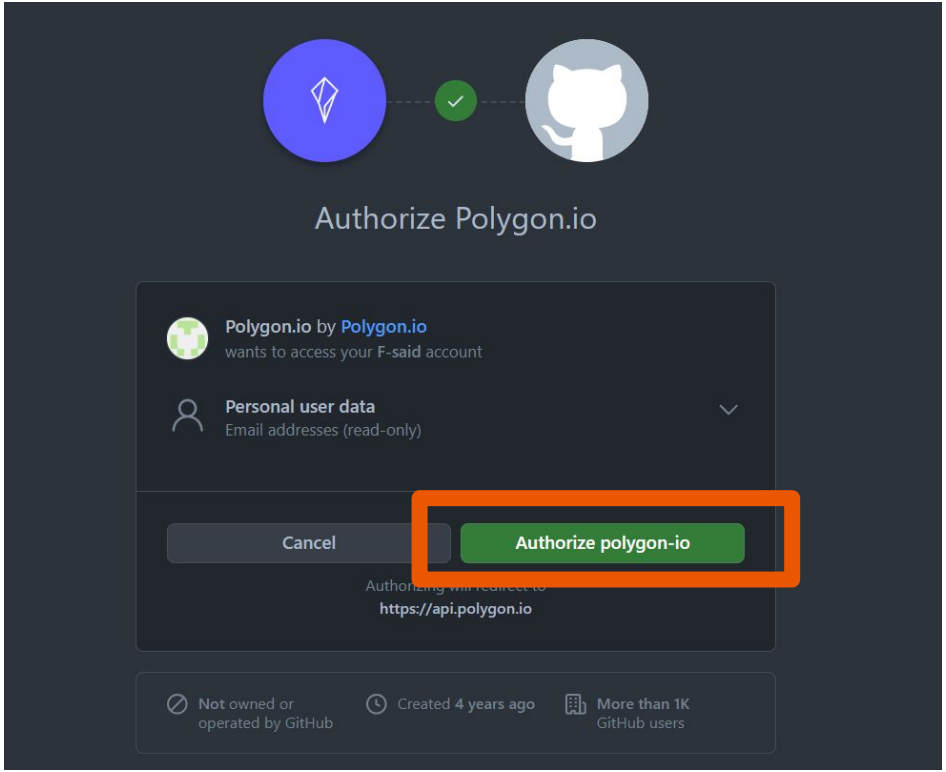
Specifically, we will be using a website that collects financial data: https://polygon.io



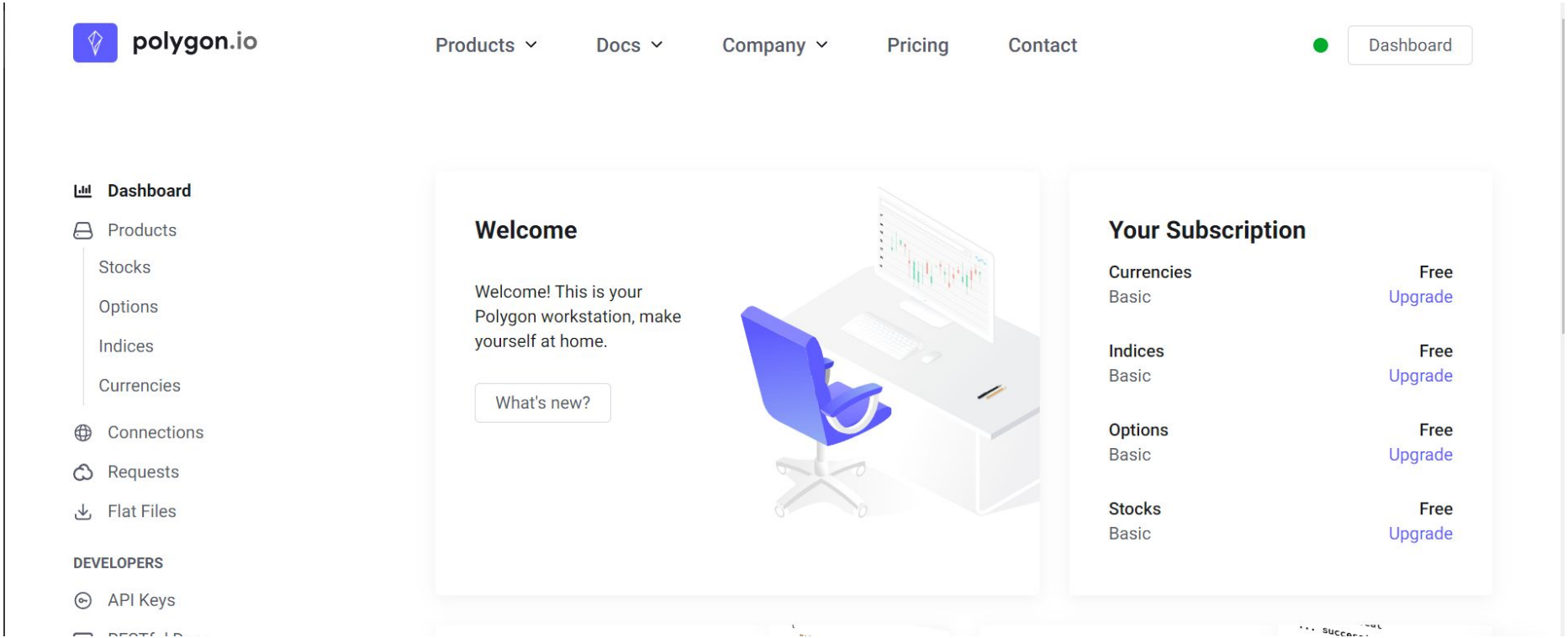
Before beginning this exercise, let's sign up for a **free API key** in order to request financial data from the https://polygon.io/ API



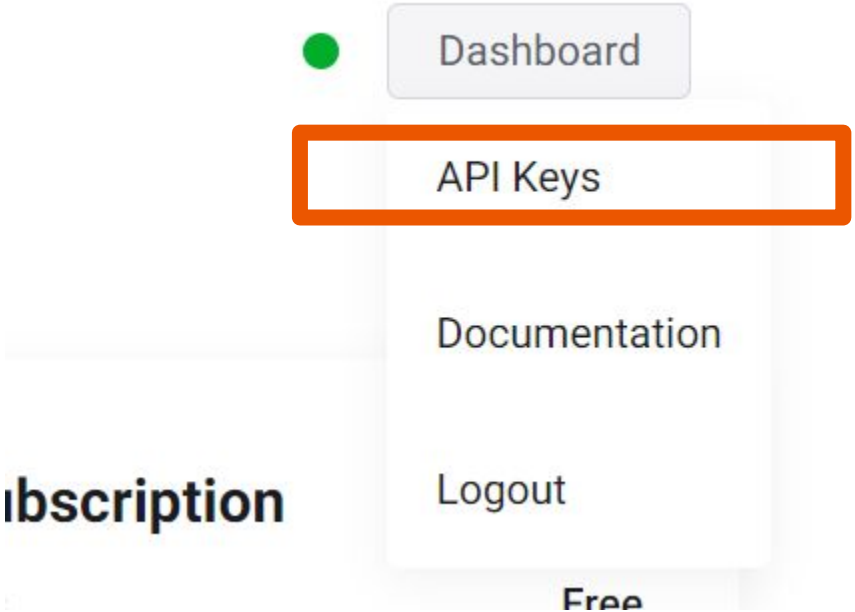
By clicking on “**Sign Up**”, you will be redirected to a page where you can use your GitHub account to sign up for a free account.



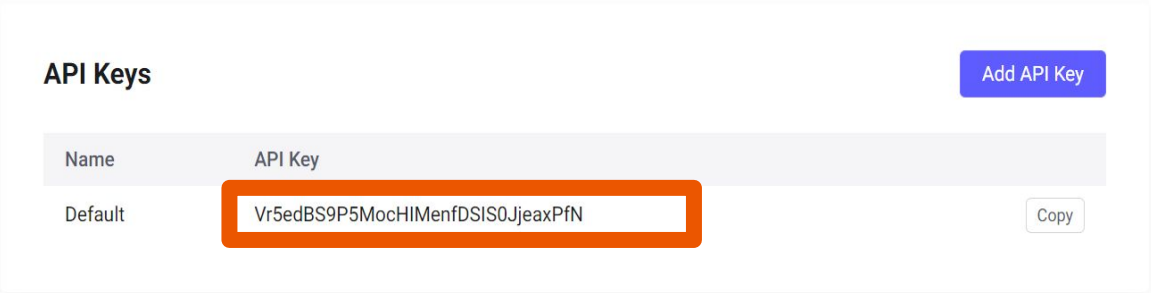
It will ask you to authorize your account (just basically view your email address), go ahead and click "**Authorize**"

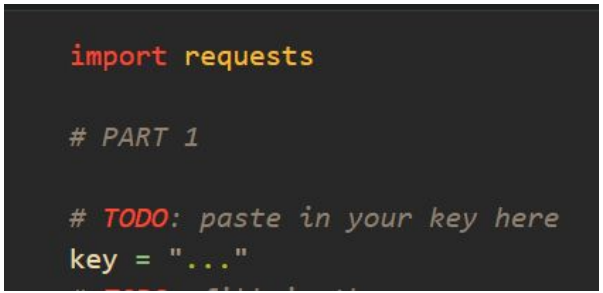


We will then be redirected to a “dashboard” page where we can request data. By default we will be in the **free tier.**



By hovering over the “**Dashboard**” menu you will see the “**API Keys**” button. Go ahead and click on this.





Finally, you will observe some randomly generated key. **Copy and paste this code into the following string**

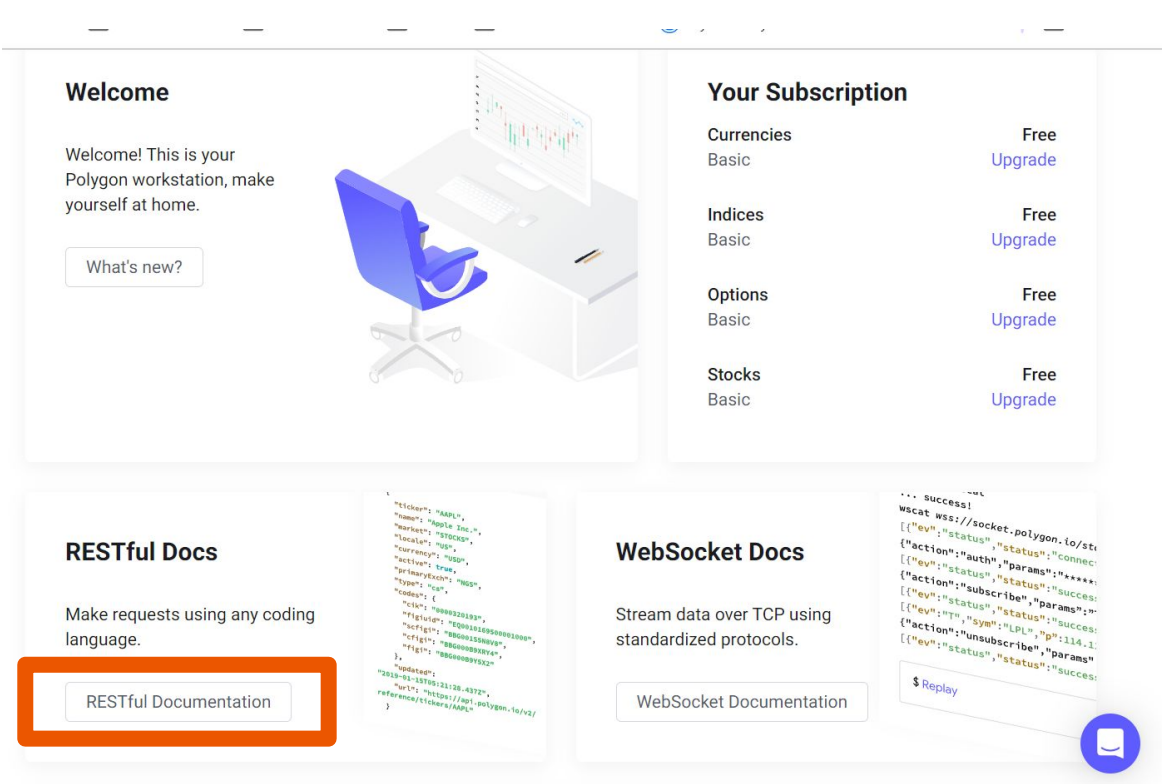
38TB of data accidentally exposed by Microsoft AI researchers

Wiz Research found a data exposure incident on Microsoft's AI GitHub repository, including over 30,000 internal Microsoft Teams messages – all caused by one misconfigured SAS token

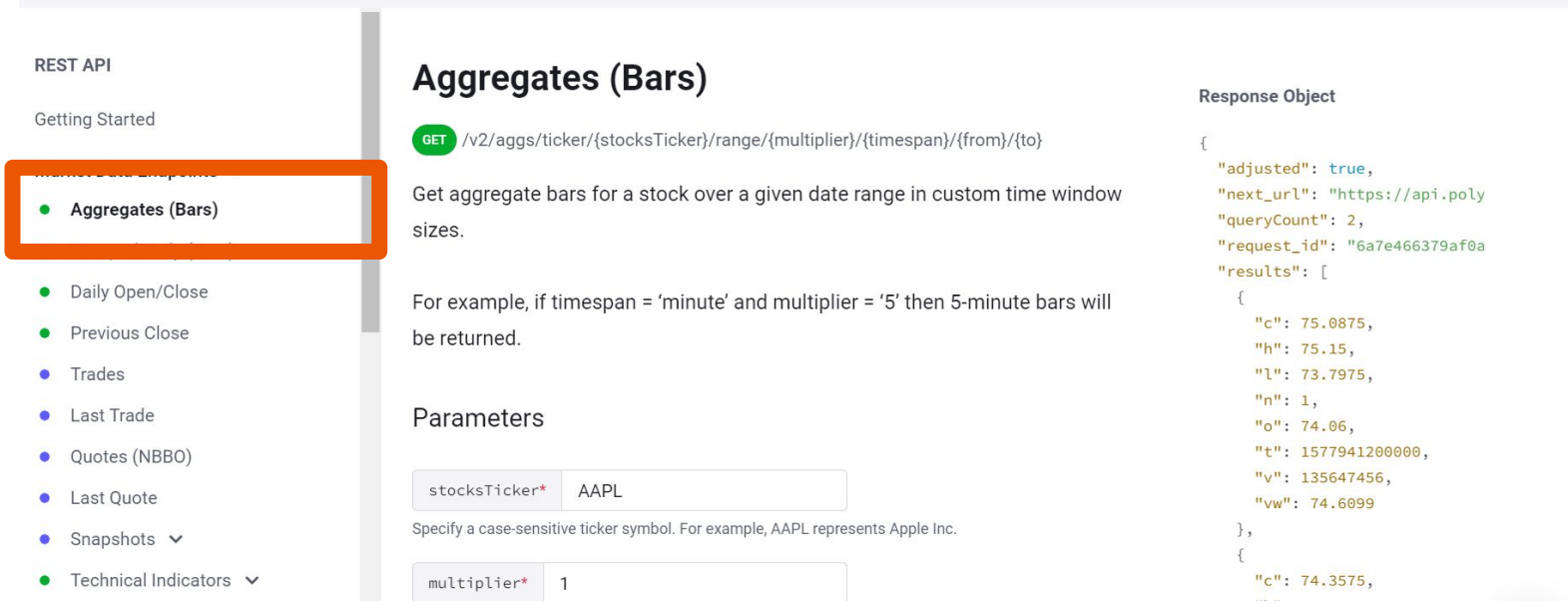
Hillai Ben-Sasson, Ronny Greenberg
September 18, 2023

10 minutes read

NOTE: You should always keep your key a secret. For now, we are working with the free tier so this isn't too important, however for future reference, exposing your key is basically the same as exposing sensitive company information.



Once we get this API key copied, let's access our respective dashboard and click on "**RESTful Documentation**"



From there, we will access the "**Aggregates**" menu option and scroll down to the respective URL.

timestamp.

`adjusted` true

Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.

`sort` asc

Sort the results by timestamp. `asc` will return results in ascending order (oldest at the top), `desc` will return results in descending order (newest at the top).

`limit` 120

Limits the number of base aggregates queried to create the aggregate results. Max 50000 and Default 5000. Read more about how limit is used to calculate aggregate results in our article on Aggregate Data API Improvements.

```
https://api.polygon.io/v2/aggs/ticker/AAPL/range/1/day/2023-01-09/2023-01-09?adjusted=true&sort=asc&limit=120&apiKey=Vr5edBS9P5MocHIMenfDSIS0JjeaxPfN
```

Copy

JSON Run Query


```
{
    "ticker": "AAPL",
    "queryCount": 1,
    "resultsCount": 1,
    "adjusted": true,
    "results": [
        {
            "v": 70790813,
            "vw": 131.6292,
            "o": 130.465,
            "c": 130.15,
            "h": 133.41,
            "l": 129.89,
            "t": 1673240400000,
            "n": 645365
        }
    ],
    "status": "OK",
    "request_id": "4e1b02a1af6d2ce53792e635c515835f",
```


Copy

Notice that we are given a **respective URL** with specified parameters that will give us a JSON object.

That is, if we just use this URL in our `requests.get()` method, we will get back this object!

Stock API Exercise

Follow along with the listed directions below to implement some basic data engineering via the `requests`, `json`, and `pandas` packages.

By the end of this analysis, we should be able to convert a JSON object into a pandas DataFrame.

We will then use these dataframes for some basic analysis.

```
import requests

# PART 1

# TODO: paste in your key here
key = "..."

# TODO: Using the "aggregates" endpoint, make a URL that will request stock data from BA (Boeing) from the date
# ranges of 2024-01-01 to 2024-03-11
# Notice that most of the URL is filled in for you. You just need to fill int the `BA` stock ticker after `ticker`
# As well as `2024-01-01` after `day`, and then `2024-03-11`
# URL: https://polygon.io/docs/stocks/get_v2_aggs_ticker__stocksticker__range__multiplier___timespan___from___to

# url = f"https://api.polygon.io/v2/aggs/ticker/.../range/1/day/.../...?adjusted=true&sort=asc&limit=120&apiKey={key}"
url = "https://api.polygon.io/v2/aggs/ticker/BA/range/1/day/2024-01-01/2024-03-11?adjusted=true&sort=asc&limit=120&apiKey=Vr5edBS9P5MocHIMenfDSIS0JeaxPfN"

r = requests.get(url)

# TODO: print out the request status of this GET request
r.status_code
```

41 ✓ 0.3s

Using this idea, complete the following API exercise as a group. Before beginning be sure to set you kernel!

Wrap-Up

Lab (Due 04/21)

Vancouver, Canada

You are a growth analyst at a Vancouver-based consulting firm called Monica Group. Your manager is spearheading the completion of a a new analytical tool which will automatically label if a review is positive, neutral, negative, or irrelevant.

You will be kicking off completion of this milestone by independently implementing a minimal-viable-product. **This will be a Python pipeline that ingests a text-file of review data and interfaces with the Open AI API in order to automatically label each review.**

We will release API keys on 4/1

Tuesday

Tuesday will entail:

- More API calls!
- Interacting with the OpenAI API

Jupyter: scratchpad of the data scientist

If you understand what you're doing, you're not learning anything. - Anonymous