HTTP and Web APIs

HTTP (Hypertext Transfer Protocol) is a set of standards describing a language for how different computers can communicate with one another over the internet. It is how the vast majority of the internet communicates; for example, your web browser uses HTTP to communicate with other websites.

An **API** is a generic term for a specific interface that two machines can communicate across. These have a number of different use-cases and different protocols based on the domain in which they are used. However, Web-based APIs almost always use HTTP.

In summary, **HTTP** is a single, generic language that computers use to communicate. A **Web-based API** is a specific implementation, which most often makes use of the HTTP language.

T. ARNOLD

Accessing Data through an API

In this class, we are going to use Web-based APIs to request various kinds of data from several different websites. While the data that can be returned from an API technically can be in any format, it is most commonly formatted in JSON (the next most common in XML, which we will see shortly).

This is not a course about networking, so note that in the following slides I am simplifying and ignoring some of the details that will not be important for our work as data scientists.

In order to get data from a Web-based API, we need to make an HTTP request. Note: your browser, email client, and many cell-phone applications also operate by making HTTP requests.

Structure of an HTTP Request

In order to make an HTTP request, we need these elements:

1. method	for us, this will always be **GET**
2. protocol (or scheme)	for us, this will be either **http** or **https**
3. authority (or hostname)	something that looks like **www.richmond.edu**
4. path	part after the authority; usually describes the specific API endpoint that we want to use from various options
5. query strings	these are a set of name value pairs that specify the specific data we are looking for

HTTP requests also have some other elements—such as cookies and a user agent—but we won't cover those here because we will not need to change nor set these values.

Example of HTTP Request

For example, here are the elements of an API that looks up the current time in a timezone:

1. method	**GET**
2. protocol	**https**
3. authority	**www.timeapi.io**
4. path	**api/Time/current/zone**
5. query strings	**{ "timeZone" : "Europe/Amsterdam" }**

We will see how to actually make this request in a moment.

Structure of an HTTP Response

Like the request, the response from the server HTTP has several parts but we only need to worry about a few of these:

1. status	an integer code giving the status of the response
2. headers	a set of name value pairs; some optional and some required we will use the fields "content-type" and date
3. body	this is the actual data as JSON, XML, text, etc.

There are a large number of status codes; we will mostly use helper functions to split them into good (we can assume the data is okay to use) and bad (we should stop and check before proceeding).

HTTP Request in R

To make HTTP request in R, we will use the excellent package **httr.** All of the components of the HTTP request are collapsed into a string called a URL. We can create a request using the modify url function by starting with the protocol, authority, and path and adding query parameters:

```
url_str <- modify_url("https://www.timeapi.io/api/Time/current/zone",
                      query = list("timeZone" = "Europe/Amsterdam"))
```

Then, we can make a GET request to the server by using the function GET:

```
res <- GET(url_str)
```

The result is a special R object that contains all of the contents of the HTTP response.

HTTP Response in R

We can get different parts of the response status and header information by using helper functions. Here are the helper functions that we will find most useful:

```
> http_type(res)
[1] "application/json"

> http_status(res)
$category
[1] "Success"

$reason
[1] "OK"

$message
[1] "Success: (200) OK »

> stop_for_status(res)      # does nothing if the request succeeds
```

HTTP Response in R

To get the actual content, we need to use the function content(), which can return the response in a variety of formats:

```
> content(res, type = "text", encoding = "utf-8")
[1]
"{\"year\":2021,\"month\":10,\"day\":2,\"hour\":2,\"minute\":52,\"seconds\":53,\"milliSecond
s\":302,\"dateTime\":\"2021-10-
02T02:52:53.3025582\",\"date\":\"10/02/2021\",\"time\":\"02:52\",\"timeZone\":\"Europe/A
msterdam\",\"dayOfWeek\":\"Saturday\",\"dstActive\":true} »
```

The data here is a text string. We can parse it using the type "application/json":

```
> content(res, type = "application/json")
$year
[1] 2021

[truncated]
```

Caching HTTP Requests

Making HTTP requests can be relatively slow and many servers will block and throttle you when making a large number of requests. For this reason, it can be useful to cache the results of the requests. I made a small helper function to do this in R:

```
res <- dsst_cache_get(url_str, cache_dir = "my_cache")
```

The object res contains a HTTP request object, loaded locally if it has already been accessed and grabbed from the server otherwise.