# APIs and visualization

Tad Dallas

**Reading for this week:**

Two this week

> https://medium.com/@traffordDataLab/querying-apis-in-r-39029b73d5f1
>
> https://www.interaction-design.org/literature/article/guidelines-for-good-visual-information-representations

## What's the point of APIs?

Application programming interaces (or `APIs`) are used to describe the communication of one computer to a web-based source of data in a programmatic manner. The computer (client) makes a request for data in a well-documented and consistent way, and receives data from the server. This is incredibly useful for acquiring data in a programmatic way, allowing analyses to be re-run on dynamic data, creating a living analysis. This could be critical in situations such as an ongoing pandemic, where daily case count updates might change model parameters or predictions.

Many websites have APIs in order to allow users to make calls and develop apps on top of existing sites e.g., Facebook, Spotify, etc. all have APIs.

Why is this valuable? Contrast the API approach to pure web scraping. Web scraping refers to extracting data from html pages. This is different from querying an API, as APIs will have documentation about how the data are represented. For instance, what happens when a website decides to change a bit of its formatting? This would potentially break any web scraping code, but the API backend would be maintained.

In the context of biological data, many data repositories (figshare, data dryad, dataONE) have clear APIs. Further, it is becoming more common for large museum and government datasets to have developed APIs. I am a huge fan of this, as it is a departure from the longstanding "my data" mentality that has lead to a clear power dynamic (e.g., being associated with labs with lots of data gives them a clear advantage... but those data were likely gathered with a combination of taxpayer dollars and researcher sweat. The dollars trump the sweat, in my opinion.).

## How do I query an API?

APIs can be queried programmtically using a number of `R` packages. Some APIs will promote their own `R` packages that have functions specific to their data structures or API. However, at the core of these packages are a small number of low-level packages for querying APIs.

The general structure of an API call is the same relatively regardless of `R` package used. The parameters of an HTTP request are typically contained in the URL. For example, to return a map of Manchester using the Google Maps Static API we would submit the following request:

```
https://maps.googleapis.com/maps/api/staticmap?center=Manchester,England&zoom=13&size=600x300&maptype=r
```

The request contains:

- a URL to the API endpoint (https://maps.googleapis.com/maps/api/staticmap?) and;
- a query containing the parameters of the request (center=Manchester,England&zoom=13&size=600x300&maptype=roadm
  In this case, we have specified the location, zoom level, size and type of map.

Web service APIs use two key HTTP verbs to enable data requests: GET and POST. A GET request is submitted within the URL with each parameter separated by an ampersand (&). A POST request is submitted in the message body which is separate from the URL. The advantage of using POST over GET requests is that there are no character limits and the request is more secure because it is not stored in the browser's cache.

There are several types of Web service APIs (e.g. XML-RPC, JSON-RPC and SOAP) but the most popular is Representational State Transfer or REST. RESTful APIs can return output as XML, JSON, CSV and several other data formats. Each API has documentation and specifications which determine how data can be transferred.

```
install.packages(c("httr", "jsonlite"))
```

```
## Installing packages into '/home/tad/R/x86_64-pc-linux-gnu-library/3.6'
## (as 'lib' is unspecified)
```

After downloading the libraries, we will be able to use them in our `R` scripts or `RMarkdown` files.

```
library(httr)
library(jsonlite)
```

## A simple example of an API call

Some APIs require a token, in order to identify who is accessing the data (e.g., Spotify, Facebook, etc.). Developer access is pretty easy to get when you have an account, but it is a bit too much of a hassle for demonstration purposes. So we will use an open API from the Internet Archive's *Open Library*.

We can send a request for data by using the documented API put out by the data provider, which consists of a base url (http://openlibrary.org/search.json?) with an amended query for searching the database. We use the GET verb, which is in `R` but stems from HTTP (hypertext transfer protocol). It stores all the information in an object that is structured in JSON. Below, we search the Open Library for "Vonnegut", and receive a nested list which includes information on status codes, among other relevant information.

```
von <- httr::GET('http://openlibrary.org/search.json?q=vonnegut')
str(von)
```

```
## List of 10
##  $ url        : chr "http://openlibrary.org/search.json?q=vonnegut"
##  $ status_code: int 200
##  $ headers    :List of 9
##   ..$ server                      : chr "nginx/1.4.6 (Ubuntu)"
##   ..$ date                        : chr "Tue, 14 Jul 2020 19:30:46 GMT"
##   ..$ content-type                : chr "application/json"
##   ..$ transfer-encoding           : chr "chunked"
##   ..$ connection                  : chr "keep-alive"
##   ..$ access-control-allow-origin: chr "*"
##   ..$ access-control-allow-method: chr "GET, OPTIONS"
##   ..$ access-control-max-age      : chr "86400"
##   ..$ x-ol-stats                  : chr "\"SR 1 0.235 TT 0 0.607\""
##   ..- attr(*, "class")= chr [1:2] "insensitive" "list"
##  $ all_headers:List of 1
##   ..$ :List of 3
##   .. ..$ status : int 200
```

```
##   .. ..$ version: chr "HTTP/1.1"
##   .. ..$ headers:List of 9
##   .. .. ..$ server                      : chr "nginx/1.4.6 (Ubuntu)"
##   .. .. ..$ date                        : chr "Tue, 14 Jul 2020 19:30:46 GMT"
##   .. .. ..$ content-type                : chr "application/json"
##   .. .. ..$ transfer-encoding           : chr "chunked"
##   .. .. ..$ connection                  : chr "keep-alive"
##   .. .. ..$ access-control-allow-origin: chr "*"
##   .. .. ..$ access-control-allow-method: chr "GET, OPTIONS"
##   .. .. ..$ access-control-max-age      : chr "86400"
##   .. .. ..$ x-ol-stats                  : chr "\"SR 1 0.235 TT 0 0.607\""
##   .. .. ..- attr(*, "class")= chr [1:2] "insensitive" "list"
##  $ cookies    :'data.frame': 0 obs. of  7 variables:
##   ..$ domain    : logi(0)
##   ..$ flag      : logi(0)
##   ..$ path      : logi(0)
##   ..$ secure    : logi(0)
##   ..$ expiration: 'POSIXct' num(0)
##   ..$ name      : logi(0)
##   ..$ value     : logi(0)
##  $ content    : raw [1:322719] 7b 0a 20 22 ...
##  $ date       : POSIXct[1:1], format: "2020-07-14 19:30:46"
##  $ times      : Named num [1:6] 0 0.0444 0.1035 0.1036 0.8321 ...
##   ..- attr(*, "names")= chr [1:6] "redirect" "namelookup" "connect" "pretransfer" ...
##  $ request    :List of 7
##   ..$ method    : chr "GET"
##   ..$ url       : chr "http://openlibrary.org/search.json?q=vonnegut"
##   ..$ headers   : Named chr "application/json, text/xml, application/xml, */*"
##   .. ..- attr(*, "names")= chr "Accept"
##   ..$ fields    : NULL
##   ..$ options   :List of 2
##   .. ..$ useragent: chr "libcurl/7.58.0 r-curl/4.3 httr/1.4.1"
##   .. ..$ httpget  : logi TRUE
##   ..$ auth_token: NULL
##   ..$ output    : list()
##   .. ..- attr(*, "class")= chr [1:2] "write_memory" "write_function"
##   ..- attr(*, "class")= chr "request"
##  $ handle     :Class 'curl_handle' <externalptr>
##  - attr(*, "class")= chr "response"
```

This data structure is basically in JSON format, which is a different file format (e.g., xml, htm, etc.). To make it something that `R` can work with, we use the `jsonlite` package, and function `fromJSON`.

```
vonInfo <- jsonlite::fromJSON(content(von, "text"), simplifyVector = FALSE)
```

```
## No encoding supplied: defaulting to UTF-8.
```

This output is a list of length 4, each containing a nested list. It is not the cleanest output, but it rarely is. But all the information you could want is buried somewhere here.

This apply statement gets information on the title of each of the works returned from our search for "Vonnegut".

```
sapply(vonInfo[[4]], function(x){
  x$title
})
```

```
##   [1] "Vonnegut"
```

```
##   [2] "Galapagos CST"
##   [3] "Kurt Vonnegut"
##   [4] "Kurt Vonnegut"
##   [5] "Vonnegut Omnibus"
##   [6] "Kurt Vonnegut"
##   [7] "Kurt Vonnegut"
##   [8] "Kurt Vonnegut"
##   [9] "Kurt Vonnegut."
##  [10] "Vonnegut statement."
##  [11] "Kurt Vonnegut"
##  [12] "Kurt Vonnegut"
##  [13] "Kurt Vonnegut"
##  [14] "Kurt Vonnegut"
##  [15] "Kurt Vonnegut"
##  [16] "Kurt Vonnegut"
##  [17] "Kurt Vonnegut"
##  [18] "Kurt Vonnegut"
##  [19] "Kurt Vonnegut"
##  [20] "Kurt Vonnegut"
##  [21] "Kurt Vonnegut"
##  [22] "Kurt Vonnegut"
##  [23] "Slaughterhouse-Five"
##  [24] "Player Piano"
##  [25] "Mother Night"
##  [26] "The Sirens of Titan"
##  [27] "Cat's Cradle"
##  [28] "Jailbird"
##  [29] "Breakfast of Champions"
##  [30] "God bless you, Mr. Rosewater"
##  [31] "Slapstick"
##  [32] "Welcome to the Monkey House"
##  [33] "Palm Sunday"
##  [34] "Hocus Pocus"
##  [35] "Galapagos"
##  [36] "Deadeye Dick"
##  [37] "Bluebeard"
##  [38] "Timequake"
##  [39] "Fates Worse than Death"
##  [40] "The Eden express"
##  [41] "Bagombo Snuff Box"
##  [42] "Look at the birdie"
##  [43] "Wampeters, Foma & Granfalloons (opinions)"
##  [44] "Between time and Timbuktu"
##  [45] "Happy birthday, Wanda June"
##  [46] "Galápagos"
##  [47] "Canary in a cat house"
##  [48] "Armageddon in Retrospect"
##  [49] "Slaughterhouse-five, or, The children's crusade"
##  [50] "A man without a country"
##  [51] "God bless you, Dr. Kevorkian"
##  [52] "Breakfast of champions, or, Goodbye blue Monday!"
##  [53] "Wampeters, foma & granfalloons"
##  [54] "God bless you, Mr Rosewater; or, Pearls before swine"
##  [55] "Like shaking hands with God"
```

```
##  [56] "Gibier de potence"
##  [57] "Sun, moon, star"
##  [58] "Top producer"
##  [59] "Domestic goddesses"
##  [60] "Le Berceau du chat"
##  [61] "Slaughterhouse 5"
##  [62] "Between Time and Timbuktu Or Prometheus 5"
##  [63] "Zeitbeben"
##  [64] "Bagombo Snuff Box a Fmt (Export)"
##  [65] "Hocus Pocus Or, What's the Hurry, Son?"
##  [66] "X5 Slaughterhouse 5 Read Gu Exp"
##  [67] "Breakfast of Champions CD"
##  [68] "Welcome to the Monkey House CD"
##  [69] "Cat's Cradle CD"
##  [70] "Slaughterhouse-Five (or The Children's Crusade: A Duty Dance with Death)"
##  [71] "Essential Vonnegut Interviews CD"
##  [72] "Slaughterhouse-five ; The sirens of Titan ; Player-piano ; cat's cradle ; Breakfast of champi
##  [73] "Timequake"
##  [74] "Galapagos"
##  [75] "The trust"
##  [76] "Oasis"
##  [77] "Sirenene på Titan"
##  [78] "Slaughterhouse five, or, The childrens crusade, a duty dance with death"
##  [79] "Ghiaccio-nove"
##  [80] "The Sirens of Titan"
##  [81] "Nothing is lost save honor"
##  [82] "Welcome to the Monkey Hous"
##  [83] "OP Hocus Pocus"
##  [84] "God Bless You, Mr. Rosewater or Pearls Before Swine"
##  [85] "Bartlett's Words to Live By"
##  [86] "Das Nudelwerk. NIEDERschriften"
##  [87] "Op Timequake"
##  [88] "Galapagos"
##  [89] "Abracadabra"
##  [90] "Miss Temptation"
##  [91] "The Handicapper General"
##  [92] "Speaking of Slapstick"
##  [93] "Hapishane Kusu"
##  [94] "Cuna de Gato"
##  [95] "Barbazul"
##  [96] "Suche Traum, biete mich. Verstreute Kurzgeschichten"
##  [97] "Sireny Titana Kolibel Dlia Koshki"
##  [98] "Mitera nihta"
##  [99] "Breakfast Champions"
## [100] "Pajaro De Celda"
```

But Vonnegut as a search term is misleading, because many people have written books on Vonnegut, while we may want books *by* Vonnegut.

```
von2 <- httr::GET('http://openlibrary.org/search.json?author=vonnegut')
str(von2)
```

```
## List of 10
##  $ url        : chr "http://openlibrary.org/search.json?author=vonnegut"
##  $ status_code: int 200
```

```
##  $ headers    :List of 9
##   ..$ server                      : chr "nginx/1.4.6 (Ubuntu)"
##   ..$ date                        : chr "Tue, 14 Jul 2020 19:30:50 GMT"
##   ..$ content-type                : chr "application/json"
##   ..$ transfer-encoding           : chr "chunked"
##   ..$ connection                  : chr "keep-alive"
##   ..$ access-control-allow-origin: chr "*"
##   ..$ access-control-allow-method: chr "GET, OPTIONS"
##   ..$ access-control-max-age      : chr "86400"
##   ..$ x-ol-stats                  : chr "\"SR 1 1.378 TT 0 3.953\""
##   ..- attr(*, "class")= chr [1:2] "insensitive" "list"
##  $ all_headers:List of 1
##   ..$ :List of 3
##   .. ..$ status : int 200
##   .. ..$ version: chr "HTTP/1.1"
##   .. ..$ headers:List of 9
##   .. .. ..$ server                      : chr "nginx/1.4.6 (Ubuntu)"
##   .. .. ..$ date                        : chr "Tue, 14 Jul 2020 19:30:50 GMT"
##   .. .. ..$ content-type                : chr "application/json"
##   .. .. ..$ transfer-encoding           : chr "chunked"
##   .. .. ..$ connection                  : chr "keep-alive"
##   .. .. ..$ access-control-allow-origin: chr "*"
##   .. .. ..$ access-control-allow-method: chr "GET, OPTIONS"
##   .. .. ..$ access-control-max-age      : chr "86400"
##   .. .. ..$ x-ol-stats                  : chr "\"SR 1 1.378 TT 0 3.953\""
##   .. .. ..- attr(*, "class")= chr [1:2] "insensitive" "list"
##  $ cookies    :'data.frame': 0 obs. of  7 variables:
##   ..$ domain    : logi(0)
##   ..$ flag      : logi(0)
##   ..$ path      : logi(0)
##   ..$ secure    : logi(0)
##   ..$ expiration: 'POSIXct' num(0)
##   ..$ name      : logi(0)
##   ..$ value     : logi(0)
##  $ content    : raw [1:333720] 7b 0a 20 22 ...
##  $ date       : POSIXct[1:1], format: "2020-07-14 19:30:50"
##  $ times      : Named num [1:6] 0 0.000023 0.000024 0.000071 4.024886 ...
##   ..- attr(*, "names")= chr [1:6] "redirect" "namelookup" "connect" "pretransfer" ...
##  $ request    :List of 7
##   ..$ method    : chr "GET"
##   ..$ url       : chr "http://openlibrary.org/search.json?author=vonnegut"
##   ..$ headers   : Named chr "application/json, text/xml, application/xml, */*"
##   .. ..- attr(*, "names")= chr "Accept"
##   ..$ fields    : NULL
##   ..$ options   :List of 2
##   .. ..$ useragent: chr "libcurl/7.58.0 r-curl/4.3 httr/1.4.1"
##   .. ..$ httpget  : logi TRUE
##   ..$ auth_token: NULL
##   ..$ output    : list()
##   .. ..- attr(*, "class")= chr [1:2] "write_memory" "write_function"
##   ..- attr(*, "class")= chr "request"
##  $ handle     :Class 'curl_handle' <externalptr>
##  - attr(*, "class")= chr "response"
```

This data structure is basically in JSON format, which is a different file format (e.g., xml, htm, etc.). To make it something that R can work with, we use the `jsonlite` package, and function `fromJSON`.

```
vonInfo2 <- jsonlite::fromJSON(content(von2, "text"), simplifyVector = FALSE)
```

```
## No encoding supplied: defaulting to UTF-8.
```

This is better, as it now provides information on books where Vonnegut was listed as an author.

```
sapply(vonInfo2[[4]], function(x){
  x$title
})
```

```
##   [1] "Slaughterhouse-Five"
##   [2] "Player Piano"
##   [3] "Mother Night"
##   [4] "The Sirens of Titan"
##   [5] "Cat's Cradle"
##   [6] "Jailbird"
##   [7] "Breakfast of Champions"
##   [8] "God bless you, Mr. Rosewater"
##   [9] "Slapstick"
##  [10] "Welcome to the Monkey House"
##  [11] "Palm Sunday"
##  [12] "Hocus Pocus"
##  [13] "Galapagos"
##  [14] "Deadeye Dick"
##  [15] "Bluebeard"
##  [16] "Timequake"
##  [17] "Fates Worse than Death"
##  [18] "Bagombo Snuff Box"
##  [19] "Look at the birdie"
##  [20] "Between time and Timbuktu"
##  [21] "Wampeters, Foma & Granfalloons (opinions)"
##  [22] "Galápagos"
##  [23] "Happy birthday, Wanda June"
##  [24] "Slaughterhouse-five, or, The children's crusade"
##  [25] "Armageddon in Retrospect"
##  [26] "A man without a country"
##  [27] "Canary in a cat house"
##  [28] "God bless you, Dr. Kevorkian"
##  [29] "Breakfast of champions, or, Goodbye blue Monday!"
##  [30] "Wampeters, foma & granfalloons"
##  [31] "Sun, moon, star"
##  [32] "God bless you, Mr Rosewater; or, Pearls before swine"
##  [33] "Gibier de potence"
##  [34] "Like shaking hands with God"
##  [35] "Galapagos"
##  [36] "Slapstick/Mother Night"
##  [37] "Le Berceau du chat"
##  [38] "Slaughterhouse-five ; The sirens of Titan ; Player-piano ; cat's cradle ; Breakfast of champi
##  [39] "X5 Slaughterhouse 5 Read Gu Exp"
##  [40] "Slaughterhouse 5"
##  [41] "Zeitbeben"
##  [42] "Vonnegut Omnibus"
##  [43] "Welcome to the Monkey House CD"
```

```
##  [44] "Slaughterhouse-Five (or The Children's Crusade: A Duty Dance with Death)"
##  [45] "Breakfast of Champions CD"
##  [46] "Bagombo Snuff Box a Fmt (Export)"
##  [47] "Cat's Cradle CD"
##  [48] "Essential Vonnegut Interviews CD"
##  [49] "Hocus Pocus Or, What's the Hurry, Son?"
##  [50] "Between Time and Timbuktu Or Prometheus 5"
##  [51] "Domestic goddesses"
##  [52] "The Sirens of Titan"
##  [53] "Welcome to the Monkey Hous"
##  [54] "Who Am I This Time"
##  [55] "Cats Cradle (Swc 1346)"
##  [56] "Pajaro De Celda"
##  [57] "Kurt Vonnegut's Slaughterhouse Five (Monarch Notes) A Critical Commentary"
##  [58] "Wampeters, Fomas & Granfalloons"
##  [59] "I folia tis gatas"
##  [60] "Easy Readers - English - Level 4"
##  [61] "Bluebeard Ltd"
##  [62] "Conversations with Kurt Vonnegut"
##  [63] "Nothing is lost save honor"
##  [64] "The Kurt Vonnegut, Jr Soundbook/4 Cassettes and Program Booklet"
##  [65] "Untitled"
##  [66] "The Barnhouse Effect"
##  [67] "Galapagos"
##  [68] "Di wu hao tu zai chang"
##  [69] "Galapagos. Roman"
##  [70] "Barbe-Bleue, ou, La vie et les oeuvres de Rabo Karabekian, 1916-1988"
##  [71] "Faces"
##  [72] "Teurastamo 5"
##  [73] "Long Walk to Forever"
##  [74] "Time & Timbuktu"
##  [75] "Bluebeard"
##  [76] "Man Without a Country, A"
##  [77] "OP Hocus Pocus"
##  [78] "God Bless You, Mr. Rosewater or Pearls Before Swine"
##  [79] "Bartlett's Words to Live By"
##  [80] "Das Nudelwerk. NIEDERschriften"
##  [81] "Op Timequake"
##  [82] "Cuna de Gato"
##  [83] "Barbazul"
##  [84] "Suche Traum, biete mich. Verstreute Kurzgeschichten"
##  [85] "Speaking of Slapstick"
##  [86] "Sireny Titana Kolibel Dlia Koshki"
##  [87] "Mitera nihta"
##  [88] "Armageddon in retrospect, and other new and unpublished writings on war and peace"
##  [89] "Hapishane Kusu"
##  [90] "Sirenene på Titan"
##  [91] "PT2 Fates Worse Than Death"
##  [92] "Fates Worse/death Ltd"
##  [93] "Kurt Vonnegut Jr Conversations With Writers (L077 Cassette)"
##  [94] "Welcome to the Monkey House"
##  [95] "Cat's Cradle"
##  [96] "The Kurt Vonnegut Jr. Audio Collection"
##  [97] "On Mark Twain, Lincoln, imperialist wars and the weather"
```

```
##  [98] "A Conversation with Kurt Vonnegut (Avid Reader)"
##  [99] "The Sirens of Titan"
## [100] "Madre Noche"
```

Side fun note: You can access Github info through API calls as well

```
tad <- GET('https://api.github.com/users/taddallas')
tadInfo <- jsonlite::fromJSON(content(tad, "text"), simplifyVector = FALSE)
```

## A more biological example

So this gives a good background about APIs, but it does not give a good biological example of when they are useful. The books that Vonnegut wrote are unlikely to change, so the search queries to this API are fairly static, and a bit removed from biology.

One good biology API that is open is the Global Biodiversity Information Facility (GBIF). We can use a similar structure to our Open Library query to obtain information on species names and occurrences.

```
giraffe <- httr::GET('https://www.gbif.org/developer/species/Giraffa+camelopardalis')
str(giraffe)
```

```
## List of 10
##  $ url        : chr "https://www.gbif.org/developer/species/Giraffa+camelopardalis"
##  $ status_code: int 404
##  $ headers    :List of 12
##   ..$ x-powered-by     : chr "Express"
##   ..$ x-request-id     : chr "87ba1a20-c608-11ea-9ecd-af6e9a0be978"
##   ..$ content-type     : chr "text/html; charset=utf-8"
##   ..$ etag             : chr "W/\"cb51-72GLVicQ6K8py7+o+756QTf8FQE\""
##   ..$ vary             : chr "Accept-Encoding"
##   ..$ content-encoding : chr "gzip"
##   ..$ date             : chr "Tue, 14 Jul 2020 19:30:52 GMT"
##   ..$ x-varnish        : chr "851742055"
##   ..$ age              : chr "0"
##   ..$ via              : chr "1.1 varnish (Varnish/5.2)"
##   ..$ transfer-encoding: chr "chunked"
##   ..$ connection       : chr "keep-alive"
##   ..- attr(*, "class")= chr [1:2] "insensitive" "list"
##  $ all_headers:List of 1
##   ..$ :List of 3
##   .. ..$ status : int 404
##   .. ..$ version: chr "HTTP/1.1"
##   .. ..$ headers:List of 12
##   .. .. ..$ x-powered-by     : chr "Express"
##   .. .. ..$ x-request-id     : chr "87ba1a20-c608-11ea-9ecd-af6e9a0be978"
##   .. .. ..$ content-type     : chr "text/html; charset=utf-8"
##   .. .. ..$ etag             : chr "W/\"cb51-72GLVicQ6K8py7+o+756QTf8FQE\""
##   .. .. ..$ vary             : chr "Accept-Encoding"
##   .. .. ..$ content-encoding : chr "gzip"
##   .. .. ..$ date             : chr "Tue, 14 Jul 2020 19:30:52 GMT"
##   .. .. ..$ x-varnish        : chr "851742055"
##   .. .. ..$ age              : chr "0"
##   .. .. ..$ via              : chr "1.1 varnish (Varnish/5.2)"
##   .. .. ..$ transfer-encoding: chr "chunked"
##   .. .. ..$ connection       : chr "keep-alive"
```

```
##   .. .. ..- attr(*, "class")= chr [1:2] "insensitive" "list"
##  $ cookies    :'data.frame': 0 obs. of  7 variables:
##   ..$ domain    : logi(0)
##   ..$ flag      : logi(0)
##   ..$ path      : logi(0)
##   ..$ secure    : logi(0)
##   ..$ expiration: 'POSIXct' num(0)
##   ..$ name      : logi(0)
##   ..$ value     : logi(0)
##  $ content    : raw [1:52049] 0a 3c 21 44 ...
##  $ date       : POSIXct[1:1], format: "2020-07-14 19:30:52"
##  $ times      : Named num [1:6] 0 0.0665 0.2226 1.0822 1.7929 ...
##   ..- attr(*, "names")= chr [1:6] "redirect" "namelookup" "connect" "pretransfer" ...
##  $ request    :List of 7
##   ..$ method    : chr "GET"
##   ..$ url       : chr "https://www.gbif.org/developer/species/Giraffa+camelopardalis"
##   ..$ headers   : Named chr "application/json, text/xml, application/xml, */*"
##   .. ..- attr(*, "names")= chr "Accept"
##   ..$ fields    : NULL
##   ..$ options   :List of 2
##   .. ..$ useragent: chr "libcurl/7.58.0 r-curl/4.3 httr/1.4.1"
##   .. ..$ httpget  : logi TRUE
##   ..$ auth_token: NULL
##   ..$ output    : list()
##   .. ..- attr(*, "class")= chr [1:2] "write_memory" "write_function"
##   ..- attr(*, "class")= chr "request"
##  $ handle     :Class 'curl_handle' <externalptr>
##  - attr(*, "class")= chr "response"
```

Before we dive too far down this hole, it is important to note that this wheel has already been invented and is being maintained by the ROpenSci collective, a group of `R` programmers who develop tools for the access and analysis of data resources. We will use this package, not because it enhances reproducibility (as it is yet another dependency), but because it is well-written, well-documented, and from a group of scientists committed to helping make open data available and analyses reproducible.

```
install.packages('rgbif')
```

```
## Installing package into '/home/tad/R/x86_64-pc-linux-gnu-library/3.6'
## (as 'lib' is unspecified)
```

```
giraffe <- rgbif::occ_search(scientificName = "Giraffa camelopardalis",
    limit = 5000)
```

> Depending on your operating system (OS) and previously installed packages, this is where we start to get into issues of dependencies. The `rgbif` package requires the installation of `rgeos`, which requires the `geos` library to be installed outside of `R`. Hopefully you will be able to successfully install the package, but it is an important note that handling these dependency and OS-specific issues is pretty central to making sure an analytical pipeline is reproducible.

So presumably we have successfully queried the GBIF API using the `rgbif` package, and now have a data.frame of giraffe occurrences (limited to 5000 occurrence points). The output data are structured as a list of lists, which is a bit confusing if we look at the `str()` of `giraffe`, as the authors of the package have created a `class` called `gbif` to hold all the data. This is common in `R` packages, and the authors have written in some great functionality in printing and working with these data, formatting the data as a `tibble` object. We will not go too much into `tibble`s, but they are pretty useful for keeping data in a "tidy" format, as columns in a tibble can be lists (e.g., what if we wanted to store spatial polygon information, vectors, etc.

alongside each row element? `tibble` handles this need effectively).

```
typeof(giraffe)
```

```
## [1] "list"
```

```
class(giraffe)
```

```
## [1] "gbif"
```

For simplicity sake, we can use some of the functionality within `rgbif` to just return the data we are interested in, specifying that we only want the raw data, not all the potential data (which includes images, etc.)

```
install.packages('rgbif')
```

```
## Installing package into '/home/tad/R/x86_64-pc-linux-gnu-library/3.6'
## (as 'lib' is unspecified)
```

```
library(rgbif)
```

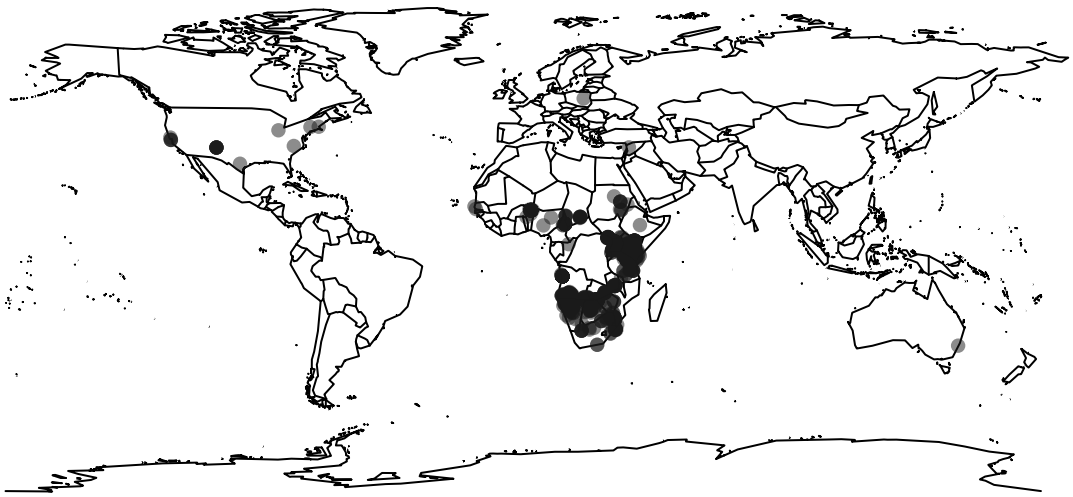
```
giraffe2 <- rgbif::occ_search(scientificName = "Giraffa camelopardalis",
    limit = 5000, return='data')[[3]]
```

We will quickly plot these data out just to show them in geographic space. We first create a base map using the `maps` package, which is a really easy way to plot geopolitical boundaries. Then we layer on the occurrence data using the `points` function.

```
install.packages('maps')
```

```
## Installing package into '/home/tad/R/x86_64-pc-linux-gnu-library/3.6'
## (as 'lib' is unspecified)
```

```
maps::map()
points(giraffe2$decimalLongitude, giraffe2$decimalLatitude, pch=16,
    col=grey(0.1,0.5))
```



Giraffes are in the US!!! These are almost certainly zoo records or errors. We will now shift into `R` visualization approaches, as we have some data, and the need to visualize it is present.

### Interfacing with local databases

Accessing data through APIs is fantastic, but there are plenty of times where you may need to access large data files locally. Note that this only is necessary if the data you are working with cannot fit into local

memory. Otherwise, there is no real advantage to using a database framework, as it will likely be more expensive in terms of time and potential frustration.

For this, we will have to include yet another dependency (5+ in this lecture alone), which of course has a bunch of other dependencies. This is not really ideal given the focus on reproducible research in the course, but it is sadly pretty much necessary.

```
install.packages('dbplyr')
```

```
## Installing package into '/home/tad/R/x86_64-pc-linux-gnu-library/3.6'
## (as 'lib' is unspecified)
```

When referring to "databases", we may think that there is a single type or schema, but this is definitely not the case. `R` has a way to accomodate for different database types, and that is to include a dependency for each database type (maybe not ideal). Below is a list describing the different ways `R` interfaces with different database types (taken from https://www.kaggle.com/anello/working-with-databases-in-r).

- RMySQL connects to MySQL and MariaDB
- RPostgreSQL connects to Postgres and Redshift.
- RSQLite embeds a SQLite database.
- odbc connects to many commercial databases via the open database connectivity protocol.
- bigrquery connects to Google's BigQuery.

We will focus on the use of RSQLite as a backend to connect to SQLite databases. This will sadly require another dependency.

```
install.packages('RSQLite')
```

```
## Installing package into '/home/tad/R/x86_64-pc-linux-gnu-library/3.6'
## (as 'lib' is unspecified)
```

```
dir.create("data_raw", showWarnings = FALSE)
download.file(url = "https://ndownloader.figshare.com/files/2292171",
              destfile = "data_raw/portal_mammals.sqlite", mode = "wb")
```

```
mammals <- DBI::dbConnect(RSQLite::SQLite(), "data_raw/portal_mammals.sqlite")
dbplyr::src_dbi(mammals)
```

```
## src:  sqlite 3.30.1 [/media/tad/EULER/Teaching/ReproResearch/Lectures/w7_rAndAPI/data_raw/portal_mamm
## tbls: plots, species, surveys
```

Just like a spreadsheet with multiple worksheets, a SQLite database can contain multiple tables. In this case three of them are listed in the tbls row in the output above:

- plots
- species
- surveys

Now that we know we can connect to the database, let's explore how to get the data from its tables into R.

### Querying the database with the SQL syntax

To connect to tables within a database, you can use the `tbl()` function from `dplyr`. This function can be used to send SQL queries to the database. To demonstrate this functionality, let's select the columns "year", "species_id", and "plot_id" from the surveys table:

```
dplyr::tbl(mammals, sql("SELECT year, species_id, plot_id FROM surveys"))
```

```
## # Source:   SQL [?? x 3]
## # Database: sqlite 3.30.1
```

```
## #   [/media/tad/EULER/Teaching/ReproResearch/Lectures/w7_rAndAPI/data_raw/portal_mammals.sqlite]
##     year species_id plot_id
##    <int> <chr>        <int>
##  1  1977 NL               2
##  2  1977 NL               3
##  3  1977 DM               2
##  4  1977 DM               7
##  5  1977 DM               3
##  6  1977 PF               1
##  7  1977 PE               2
##  8  1977 DM               1
##  9  1977 DM               1
## 10  1977 PF               6
## # ... with more rows
```

With this approach you can use any of the SQL queries we have seen in the database lesson.

### Querying the database with the dplyr syntax

One of the strengths of dplyr is that the same operation can be done using dplyr verbs instead of writing SQL. First, we select the table on which to do the operations by creating the surveys object, and then we use the standard dplyr syntax as if it were a data frame:

```
surveys <- dplyr::tbl(mammals, "surveys")
surveys %>%
    dplyr::select(year, species_id, plot_id)
```

```
## # Source:   lazy query [?? x 3]
## # Database: sqlite 3.30.1
## #   [/media/tad/EULER/Teaching/ReproResearch/Lectures/w7_rAndAPI/data_raw/portal_mammals.sqlite]
##     year species_id plot_id
##    <int> <chr>        <int>
##  1  1977 NL               2
##  2  1977 NL               3
##  3  1977 DM               2
##  4  1977 DM               7
##  5  1977 DM               3
##  6  1977 PF               1
##  7  1977 PE               2
##  8  1977 DM               1
##  9  1977 DM               1
## 10  1977 PF               6
## # ... with more rows
```

In this case, the surveys object behaves like a data frame. Several functions that can be used with data frames can also be used on tables from a database. For instance, the `head()` function can be used to check the first 10 rows of the table:

```
head(surveys, n = 10)
```

```
## # Source:   lazy query [?? x 9]
## # Database: sqlite 3.30.1
## #   [/media/tad/EULER/Teaching/ReproResearch/Lectures/w7_rAndAPI/data_raw/portal_mammals.sqlite]
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <int> <int> <int> <int>   <int> <chr>      <chr>           <int>  <int>
##  1         1     7    16  1977       2 NL         M                  32     NA
```

```
##  2         2     7    16  1977       3 NL         M                 33     NA
##  3         3     7    16  1977       2 DM         F                 37     NA
##  4         4     7    16  1977       7 DM         M                 36     NA
##  5         5     7    16  1977       3 DM         M                 35     NA
##  6         6     7    16  1977       1 PF         M                 14     NA
##  7         7     7    16  1977       2 PE         F                 NA     NA
##  8         8     7    16  1977       1 DM         M                 37     NA
##  9         9     7    16  1977       1 DM         F                 34     NA
## 10        10     7    16  1977       6 PF         F                 20     NA
```

```
surveys2 <- surveys %>%
  dplyr::filter(weight < 5) %>%
  dplyr::select(species_id, sex, weight)
```

R makes lazy calls to databases, until you make it not be lazy. That is, everything is held outside of memory until you tell R that some portion of the data should not be.

```
head(surveys2)
```

```
## # Source:   lazy query [?? x 3]
## # Database: sqlite 3.30.1
## #   [/media/tad/EULER/Teaching/ReproResearch/Lectures/w7_rAndAPI/data_raw/portal_mammals.sqlite]
##   species_id sex   weight
##   <chr>      <chr>  <int>
## 1 PF         M          4
## 2 PF         F          4
## 3 PF         <NA>       4
## 4 PF         F          4
## 5 PF         F          4
## 6 RM         M          4
```

To pull the data into memory and allow us to use the data, we must use the `collect` function.

```
surveys3 <- collect(surveys2)
head(surveys3)
```

```
## # A tibble: 6 x 3
##   species_id sex   weight
##   <chr>      <chr>  <int>
## 1 PF         M          4
## 2 PF         F          4
## 3 PF         <NA>       4
## 4 PF         F          4
## 5 PF         F          4
## 6 RM         M          4
```

This difference becomes clear when we try to index a specific column of data and work with it.

```
surveys2$sex
```

```
## NULL
```

```
surveys3$sex
```

```
##  [1] "M" "F" NA  "F" "F" "M" "F" "M" "M" "M" "M" "F" "M" "M" "M" "M" "M"
```

### Relating this back to the `dplyr` syntax of joins

We discussed joins in the data manipulation section. Joins are pretty key to working with databases, as much of the benefit of database structure is from nested data and the utility of key values. For instance, in our mammal sampling data, we have plot level data...

```
plots <- dplyr::tbl(mammals, "plots")
plots
```

```
## # Source:   table<plots> [?? x 2]
## # Database: sqlite 3.30.1
## #   [/media/tad/EULER/Teaching/ReproResearch/Lectures/w7_rAndAPI/data_raw/portal_mammals.sqlite]
##    plot_id plot_type
##      <int> <chr>
##  1       1 Spectab exclosure
##  2       2 Control
##  3       3 Long-term Krat Exclosure
##  4       4 Control
##  5       5 Rodent Exclosure
##  6       6 Short-term Krat Exclosure
##  7       7 Rodent Exclosure
##  8       8 Control
##  9       9 Spectab exclosure
## 10      10 Rodent Exclosure
## # ... with more rows
```

and survey-level data, where plot-level data provides what is essentially metadata for each survey.

We can use joins to combine and manipulate these data, as we did before, but now in the context of the database structure.

```
survPlot <- dplyr::left_join(surveys, plots, by='plot_id')
```

---

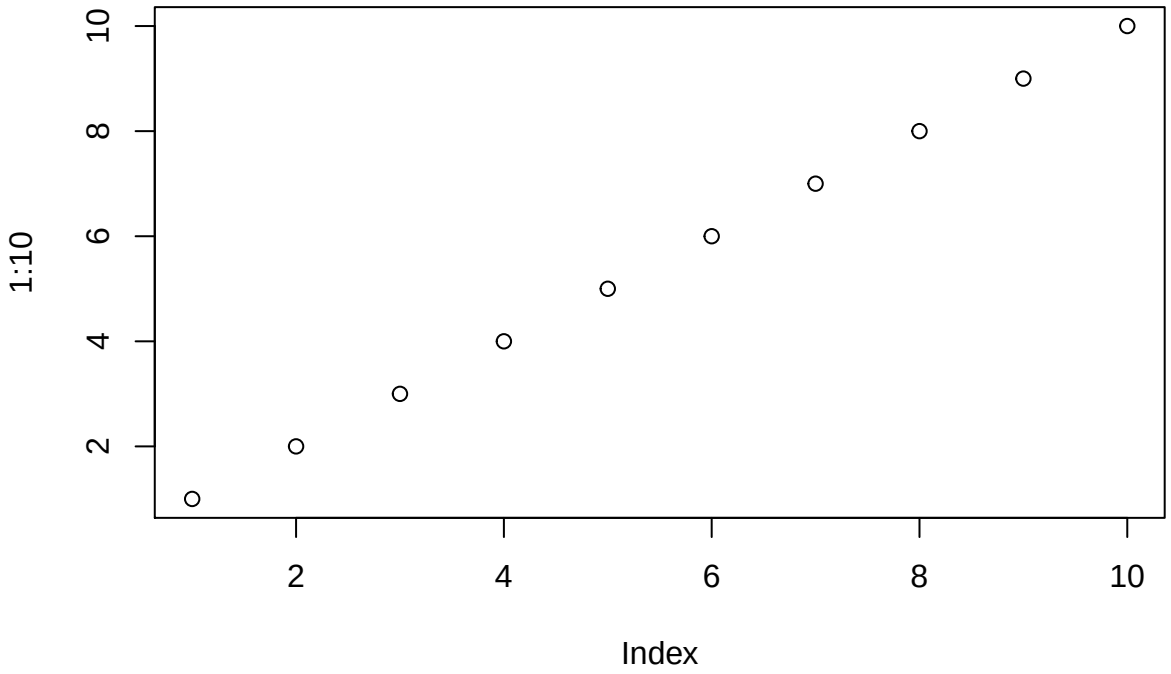
## R visualizations

Visualizing data is arguably the most fun thing to do in `R`, and is important to communicate the results of analyses, the structure of data, and the sign and strength of relationships. Also, I definitely recommend routinely plotting out your data in order to visually explore outliers, potential errors, data distributions, etc. This approach can generate new ideas, sanity check the data, and provides immediate feedback to the user on data quality/issues/size/etc.

There are multiple schools of though regarding generating visualizations in `R`. The loudest of these camps are the `ggplot2` users. Built using ideas from the Grammar of Graphics, `ggplot2` is integrated into the `tidyverse` and creates some decent-looking plots. I have some issues with the use of `ggplot2`, and will teach you how to make visualizations using base `R`. This is 1) because this course is on reproducibility, and `ggplot2` is liable to change and break your plotting code, and 2) `ggplot2` pretty much creates a language of its own for plotting (e.g., aes, geom_point, etc. etc.) and I would rather focus on things like look more like `R` code that give you the customizability (e.g., Google "how do I change axis label?" or something fairly simple and you will see the `ggplot2` folks struggling). There is also the `lattice` package, which is pretty solid for multi-panel plots and definitely has some cool functions (their `lattice::levelplot()` is a common go-to for making heatmaps for me). But base `R` can make everything you need.

The `plot` function

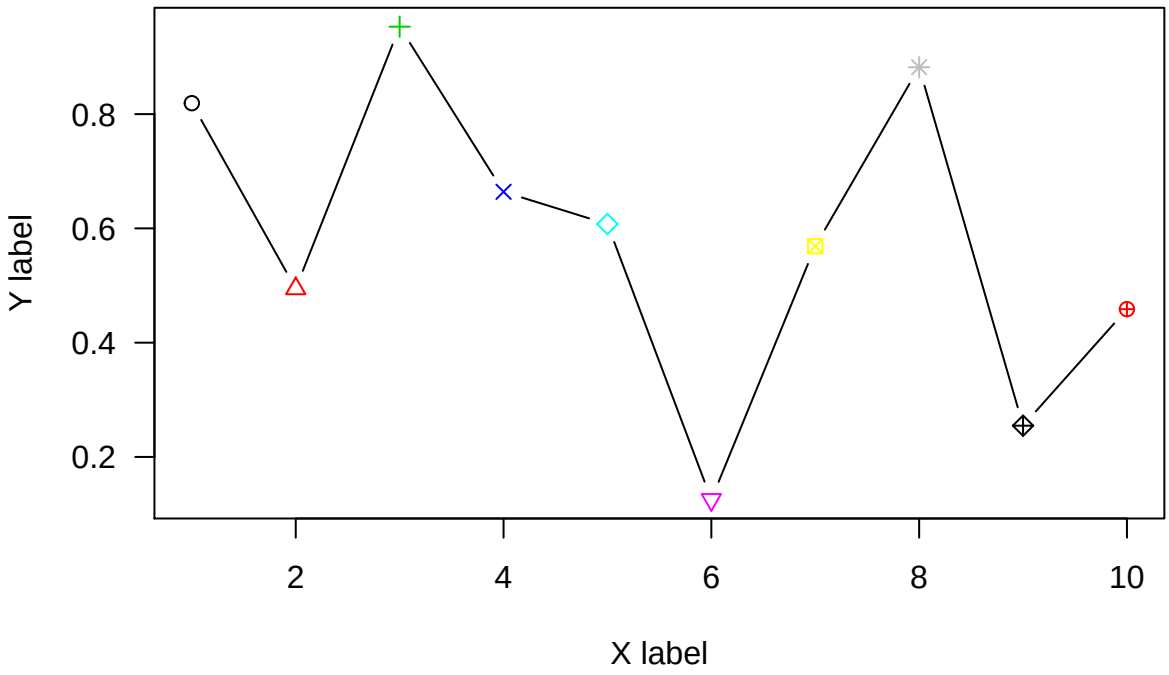
```
plot(1:10)
```




This is an ugly plot. But we can make it prettier. The full range of ways we can make it prettier is provided in the documentation of the `plot` function. For simplicity, I will only go over some basic ways to customize your plots here.

```
?plot
```

```
plot(x=1:10, y=runif(10),
  type='b',
  ylab='Y label',
  xlab='X label',
  pch=1:10,
  col=1:10,
  las=1
)
```




The `plot` function is general, and will putatively work with many different objects. Functions written in `R` libraries use base `R` plotting while specifying things specific to the data structure. But there are also a bunch of other base `R` functions for making plots.

```
plot()
hist()
barplot()
image()
```

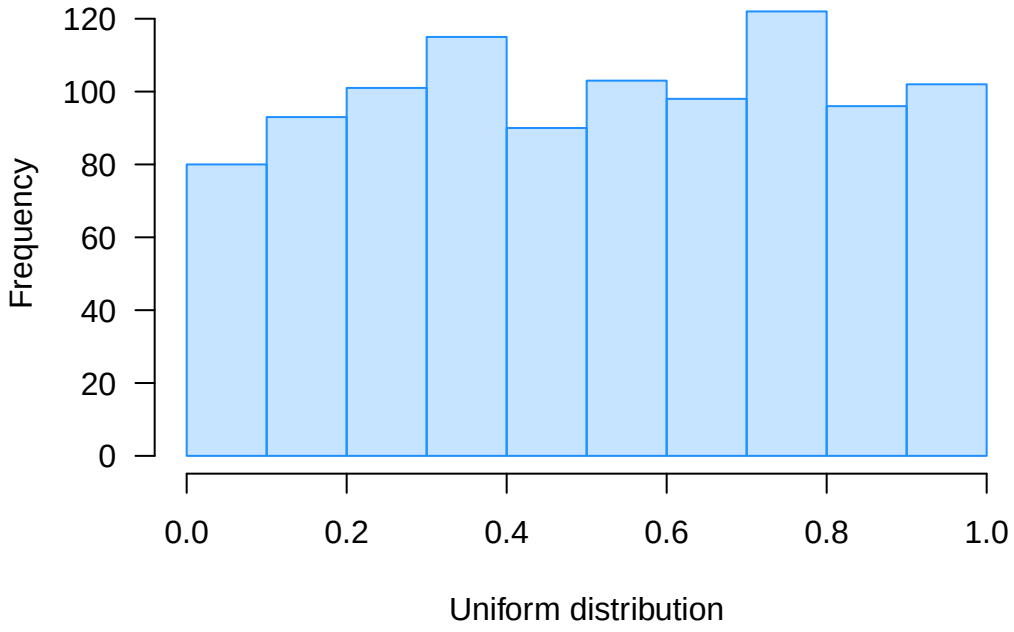
## par is your friend

`par` is used to set and query your graphical parameters. This is incredibly useful, as it allows you to customize the plotting space itself (e.g., setting margins). If you call `par()` with no specifications, it provides you a long list of all the default parameters. Every single one of these can be changed, giving the user the freedom to design the plotting space exactly to their needs.

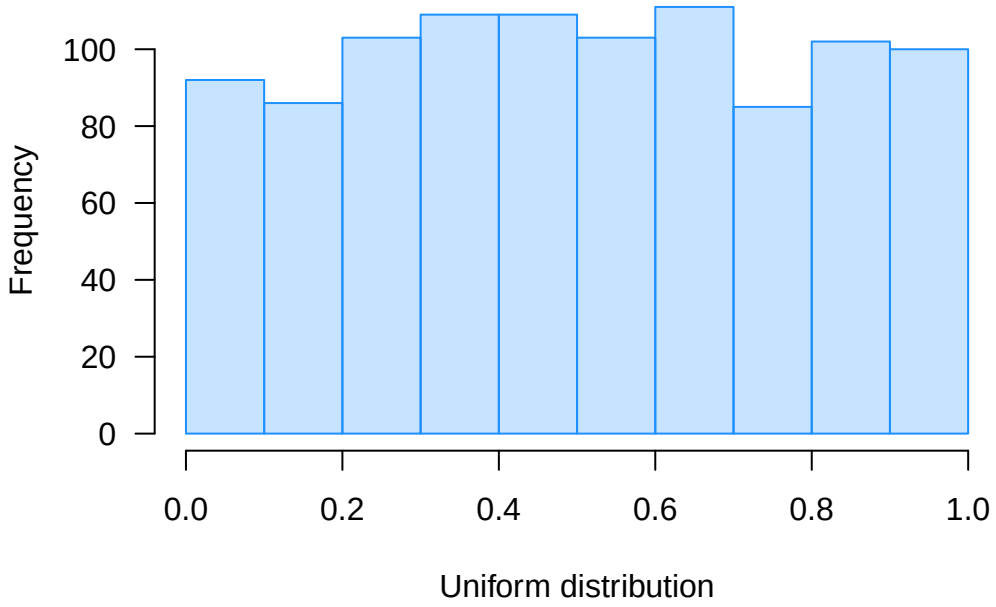
```
par()
```

One of the most useful arguments to `par` in my opinion is setting the plotting margins (`mar` to set inner margins, or `oma` to set outer margins). I mostly worry about the inner margins (`mar`).

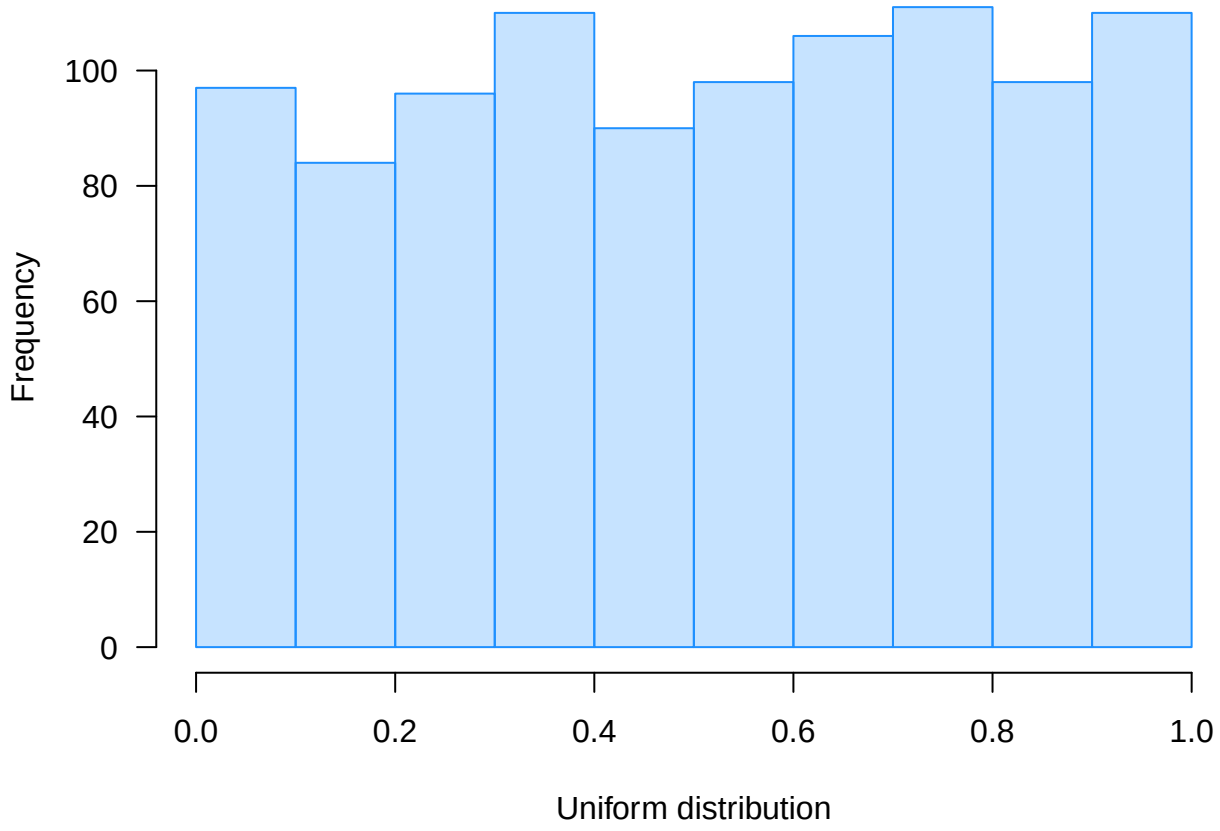
```
par(mar=c(5,5,5,5))
hist(runif(1000),
    main='', border='dodgerblue', las=1,
    xlab='Uniform distribution',
    col=adjustcolor('dodgerblue', 0.25))
```




```
par(mar=c(10,10,0.5,0.5))
hist(runif(1000),
    main='', border='dodgerblue', las=1,
    xlab='Uniform distribution',
    col=adjustcolor('dodgerblue', 0.25))
```




```
par(mar=c(4,4,0.5,0.5))
hist(runif(1000),
    main='', border='dodgerblue', las=1,
    xlab='Uniform distribution',
    col=adjustcolor('dodgerblue', 0.25))
```



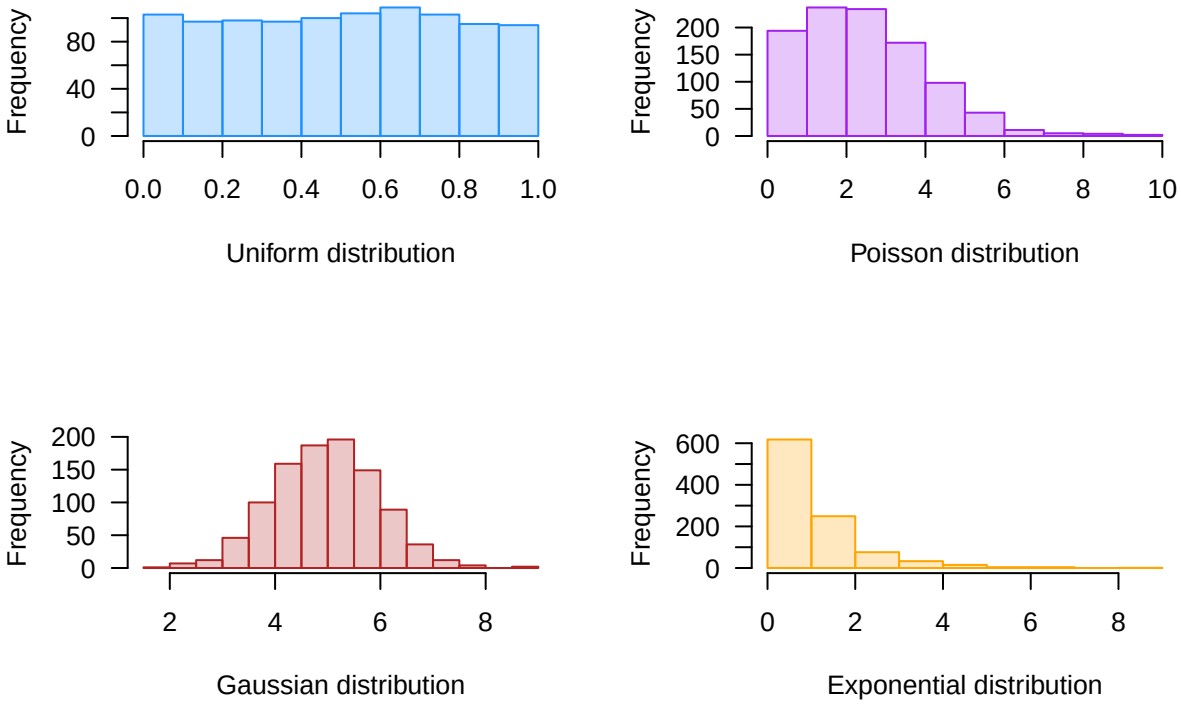

**Using the layout function to make panel plots**

```
layout(matrix(1:4, ncol=2))
hist(runif(1000),
    main='', border='dodgerblue', las=1,
    xlab='Uniform distribution',
    col=adjustcolor('dodgerblue', 0.25))

hist(rnorm(1000, 5,1),
    main='', border='firebrick', las=1,
    xlab='Gaussian distribution',
    col=adjustcolor('firebrick', 0.25))

hist(rpois(1000, 3),
    main='', border='purple', las=1,
    xlab='Poisson distribution',
    col=adjustcolor('purple', 0.25))

hist(rexp(1000, 1),
    main='', border='orange', las=1,
    xlab='Exponential distribution',
    col=adjustcolor('orange', 0.25))
```




Modifying base plots (building from scratch). The code below uses the `type='n'` argument, which creates a graphical window, but does not plot the data. We also set `axes=FALSE`, which removes x and y axes. This creates a blank slate that we can layer additional information on.

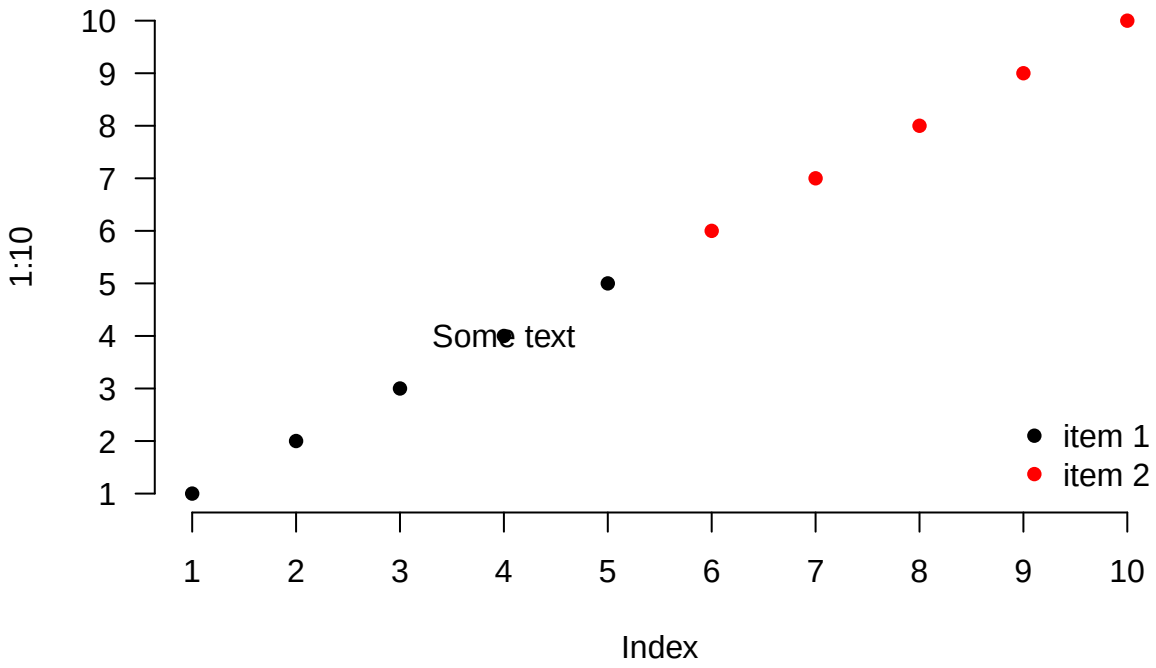
```
plot(1:10,
    type='n',
    axes=FALSE)

# plot the points
points(x=1:5, y=1:5, pch=16, col=1)
points(x=6:10, y=6:10, pch=16, col=2)

# set up the axes
axis(1, at=1:10)
axis(2, at=1:10, las=1)

#Add some text to the plot.
text(4, 4, 'Some text')


# Include a legend
legend('bottomright',
    legend=c('item 1', 'item 2'),
    pch=16, col=1:2, bty='n')
```




From this exercise, we can see that we have complete control over the base `R` plotting. This is not really the case with `ggplot2`, or at least not quite as easily.

### Some other side considerations

Color is important while plotting, but we want to make sure that the use of color is informative and accessible to all. Color palettes which are color-blind friendly and easily convertable to greyscale for black-and-white printers. This can be done by Googling color palettes and providing them to `R` as a vector of hex codes (e.g., c("#999999", "#E69F00", "#56B4E9")) or html color names (e.g., 'c('dodgerblue', 'firebrick', 'purple')). I like hex codes because they are unlikely to change, and are a good way to specify the exact color you want.

If you want to use existing color palettes, there is no shortage of `R` packages that are just color palettes. My feelings are mixed on this. One of the most developed of these packages is `RColorBrewer`, which has a nice set of palettes to use in different situations (sequential, paired, diverging, etc.), and is really easily accessible and straightforward to use.

```
install.packages('RColorBrewer')
library(RColorBrewer)
display.brewer.all(colorblindFriendly = T)
```