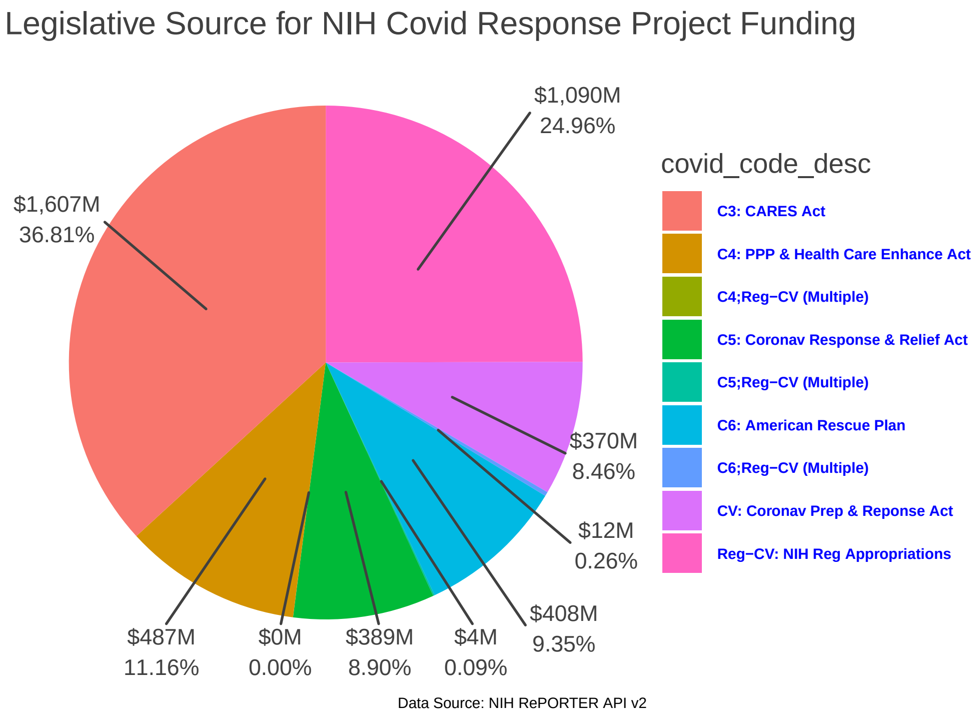
repoRter.nih: a convenient R interface to the NIH RePORTER Project API

Michael Barr, ACAS, MAAA, CPCU



Legislative Source for NIH Covid Response Project Funding

Data Source: NIH RePORTER API v2

Introduction

The US National Institute of Health (NIH) received funding of approximately $42 billion in fiscal year 2022; $31 billion (72%) of this was awarded by the NIH in the form of research grant funding to hospitals, medical colleges, non-profits, businesses, and other organizations based in the U.S. and abroad.[1] The NIH maintains a publicly available database called "RePORTER" to track this substantial flow of grant funding and makes it available to the public via a web-based query interface as well as an API.

> "The NIH RePORTER APIs is designed to programmatically expose relevant scientific awards data from both NIH and non-NIH federal agencies for the consumption of project teams or external 3rd party applications to support reporting, data analysis, data integration or to satisfy other business needs as deemed pertinent." –NIH RePORTER v2 API Documentation

This data can have significant value for many audiences, including researchers, investors, industry, watchdogs/public advocates, and R users. But constructing queries and retrieving results programmatically involves some coding overhead which can be a challenge for those not familiar with RESTful APIs and JSON; it

[1] https://nexus.od.nih.gov/all/2021/04/21/fy-2020-by-the-numbers-extramural-investments-in-research

takes some effort even for those who are. The `repoRter.nih` package aims to simplify this task for the typical analyst scripting in R.

Getting Started

Installation

This package (latest stable release) can be installed from CRAN the usual way:

```r
install.packages("repoRter.nih")
```

The current dev version can be installed from github, on the `dev` branch:

```r
devtools::install_github('bikeactuary/repoRter.nih@dev')
```

I welcome R developers more capable than myself to collaborate on improving the source code, documentation, and unit testing in this package.

Basic Workflow

```r
library(repoRter.nih)
```

The `make_req()` method is used to generate a valid JSON request object. The req can subsequently be passed to the RePORTER Project API and results retrieved via the `get_nih_data()` method.

Generating the request:

```r
# all projects funded by the Paycheck Protection Act, Coronavirus Response and
# Relief Act, and American Rescue Plan, in fiscal year 2021
req <- make_req(criteria =
                  list(fiscal_years = 2021,
                       covid_response = c("C4", "C5", "C6")))
#> This is your JSON payload:
#> {
#>     "criteria": {
#>         "fiscal_years": [
#>             2021
#>         ],
#>         "covid_response": [
#>             "C4",
#>             "C5",
#>             "C6"
#>         ],
#>         "use_relevance": false,
#>         "include_active_projects": false,
#>         "exclude_subprojects": false,
#>         "multi_pi_only": false,
#>         "newly_added_projects_only": false,
#>         "sub_project_only": false
#>     },
#>     "offset": 0,
#>     "limit": 500
#> }
#>
#> If you receive a non-200 API response, compare this formatting (boxes, braces, quotes, etc.) to
    the 'Complete Payload' schema provided here:
#> https://api.reporter.nih.gov/?urls.primaryName=V2.0#/Search/post_v2_projects_search
```

Sending the request and retrieving results:

```
res <- get_nih_data(req)
#> Retrieving first page of results (up to 500 records)
class(res)
#> [1] "tbl_df"     "tbl"        "data.frame"
```

A tibble is returned containing 43 columns. This data is not flat - several columns are nested `data.frames` and `lists` (of variable length vectors and `data.frames` of varying height).

```
res %>% glimpse(width = getOption("cli.width"))
#> Rows: 251
#> Columns: 43
#> $ appl_id                  <int> 10255113, 10425707, 10403857, 10258548, 10439178, 10446500, ~
#> $ subproject_id            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
#> $ fiscal_year              <int> 2021, 2021, 2021, 2021, 2021, 2021, 2021, 2021, 2021, 2021, ~
#> $ project_num              <chr> "3P20GM104417-07S1", "3P20GM104417-08S1", "3R01ES028615-07S1~
#> $ project_serial_num       <chr> "GM104417", "GM104417", "ES028615", "ES028615", "DC019579", ~
#> $ organization             <df[,17]> <data.frame[31 x 17]>
#> $ award_type               <chr> "3", "3", "3", "3", "7", "3", "1", "3", "3", "1", "1", "~
#> $ activity_code            <chr> "P20", "P20", "R01", "R01", "U01", "R01", "R01", "U01", "U19~
#> $ award_amount             <int> 1115953, 681188, 300000, 1609765, 877287, 348242, 667277, 26~
#> $ is_active                <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, TRUE, TRUE, TRUE, TRUE,~
#> $ project_num_split        <df[,7]> <data.frame[31 x 7]>
#> $ principal_investigators  <list> [<data.frame[1 x 7]>], [<data.frame[1 x 7]>], [<data.frame[2~
#> $ contact_pi_name          <chr> "ADAMS, ALEXANDRA K.", "ADAMS, ALEXANDRA K.", "AL-HENDY, ~
#> $ program_officers         <list> [<data.frame[1 x 4]>], [<data.frame[1 x 4]>], [<data.frame[~
#> $ agency_ic_admin          <df[,3]> <data.frame[31 x 3]>
#> $ agency_ic_fundings       <list> [<data.frame[1 x 5]>], [<data.frame[1 x 5]>], [<data.frame[1~
#> $ cong_dist                <chr> "MT-00", "MT-00", "IL-01", "IL-01", "MA-08", "MA-08", "MO-0~
#> $ spending_categories      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
#> $ project_start_date       <chr> "2020-11-17T05:00:00Z", "2021-09-01T04:00:00Z", "2020-11-11~
#> $ project_end_date         <chr> "2022-08-31T04:00:00Z", "2022-10-31T04:00:00Z", "2022-07-31T~
#> $ organization_type        <df[,3]> <data.frame[31 x 3]>
#> $ full_foa                 <chr> "PA-20-135", "PAR-18-264", "PA-20-272", "PA-20-135", "RFA-OD~
#> $ full_study_section       <df[,6]> <data.frame[31 x 6]>
#> $ award_notice_date        <chr> "2020-11-17T05:00:00Z", "2021-09-21T04:00:00Z", "2021-08-31T~
#> $ is_new                   <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS~
#> $ mechanism_code_dc        <chr> "RC", "RC", "RP", "RP", "RP", "RP", "RP", "RP", "RP", "RP~
#> $ core_project_num         <chr> "P20GM104417", "P20GM104417", "R01ES028615", "R01ES028615", ~
#> $ terms                    <chr> "Adult ; 21+ years old ; Adult Human ; adulthood ; Affect~
#> $ pref_terms               <chr> "2019-nCoV;Adult;Affect;Agricultural Workers;American Indian~
#> $ abstract_text            <chr> "Project Summary\nThe COVID-19 pandemic has disproportionate~
#> $ project_title            <chr> "Center for American Indian and Rural Health Equity", "Cente~
#> $ phr_text                 <chr> "Project Narrative\nWorking with our Latino community partne~
#> $ spending_categories_desc <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
#> $ agency_code              <chr> "NIH", "NIH", "NIH", "NIH", "NIH", "NIH", "NIH", "NIH", "NIH~
#> $ covid_response           <list> "C4", "C6", "C6", "C4", "C4", "C6", "C6", "C6", "C6", "C4", ~
#> $ arra_funded              <chr> "N", "N", "N", "N", "N", "N", "N", "N", "N", "N", "N", "N", ~
#> $ budget_start             <chr> "2020-11-17T05:00:00Z", "2021-09-01T04:00:00Z", "2021-09-01T~
#> $ budget_end               <chr> "2022-08-31T04:00:00Z", "2022-10-31T04:00:00Z", "2022-07-31T~
#> $ cfda_code                <chr> "310", "859", "113", "310", "310", "855", "855", "855", "855~
#> $ funding_mechanism        <chr> "Research Centers", "Research Centers", "Non-SBIR/STTR", "N~
#> $ direct_cost_amt          <int> 1006607, 616778, 297064, 1560464, 569403, 207287, 473684, 15~
#> $ indirect_cost_amt        <int> 109346, 64410, 2936, 49301, 307884, 140955, 193593, 116250, ~
#> $ project_detail_url       <chr> "https://reporter.nih.gov/project-details/10255113", "https:~
```

Criteria-Field Translation

A dataset (`nih_fields`) is provided with this package to assist in translating between field names used in the payload `criteria`, column names in the return data, and field names used in the `include_fields`, `exclude_fields`, and `sort_field` arguments.

```
data("nih_fields")
nih_fields %>% print
#> # A tibble: 43 x 5
#>    payload_name             response_name      include_name     return_class mod_ind
#>    <chr>                    <chr>              <chr>            <chr>          <int>
#>  1 appl_ids                 appl_id            ApplId           integer            1
#>  2 <NA>                     subproject_id      SubprojectId     character          0
#>  3 fiscal_years             fiscal_year        FiscalYear       integer            1
#>  4 project_nums             project_num        ProjectNum       character          1
#>  5 serial_num               project_serial_num ProjectSerialNum character          1
#>  6 <NA>                     organization       Organization     data.frame         0
#>  7 award_types              award_type         AwardType        character          1
#>  8 activity_codes           activity_code      ActivityCode     character          1
#>  9 award_amount_range       award_amount       AwardAmount      integer            1
#> 10 include_active_projects is_active          IsActive         logical            1
#> # ... with 33 more rows
```

Some fields can not be used as filtering `criteria` - these will show `NA` in the `payload_name` column.

Generating Requests

Most of the detail (and function documentation) is around the many parameters available in RePORTER to filter/search project records. Let's get into some of the capabilities.

Default Request

If no arguments are supplied, the default behavior of `make_req()` is to generate a request for all projects funded in `fiscal_years = lubridate::year(Sys.Date())`. Limiting requests to a single year is often necessary (depending on additional filtering criteria used) due to a RePORTER restriction that a maximum of 10K records may be returned from any result set. There are currently ~2.6M projects in the database going back to fiscal year 1985, and each fiscal year tends to have 70-100K projects, so the 10K limit can be restrictive to the user wanting a broad search.

```
req <- make_req()
#> This is your JSON payload:
#> {
#>     "criteria": {
#>         "fiscal_years": [
#>             2022
#>         ],
#>         "use_relevance": false,
#>         "include_active_projects": false,
#>         "exclude_subprojects": false,
#>         "multi_pi_only": false,
#>         "newly_added_projects_only": false,
#>         "sub_project_only": false
#>     },
#>     "offset": 0,
#>     "limit": 500
#> }
#>
```

```
#> If you receive a non-200 API response, compare this formatting (boxes, braces, quotes, etc.) to
    the 'Complete Payload' schema provided here:
#> https://api.reporter.nih.gov/?urls.primaryName=V2.0#/Search/post_v2_projects_search
```

The method prints a helpful message to the console in addition to returning the JSON. Set `message = FALSE` if you wish to suppress this message.

Limiting Data Retrieved

You can limit both the width and height of the result set retrieved from the API.

Fields

We probably will not need to fetch every field every time. The `include_fields` argument is provided to specify a limited set of fields to be returned. Alternatively, fields may be excluded using `exclude_fields`.

Records (projects)

This package provides the ability to retrieve only a limited number of result pages via the `max_pages` argument. This can be useful for developing/testing your queries (and for reducing time to render package documentation). Each page has a record count equal to `limit` - so setting `max_pages = 5` with the default `limit = 500` (the maximum permitted by RePORTER) in `make_req()` will result in up to 2,500 total records returned.

Ex. 1 - Limiting results and selecting fields

```
data("nih_fields")
fields <- nih_fields %>%
  filter(response_name %in%
           c("appl_id", "subproject_id", "project_title", "fiscal_year",
             "award_amount", "is_active", "project_start_date")) %>%
  pull(include_name)

req <- make_req(include_fields = fields,
                limit = 500,
                message = FALSE) # default
res <- get_nih_data(query = req,
                    max_pages = 1)
#> Retrieving first page of results (up to 500 records)
#> max_pages set to 1 by user. Result set contains 32 pages. Only partial results will be
    retrieved.

res %>% glimpse(width = getOption("cli.width"))
#> Rows: 500
#> Columns: 7
#> $ appl_id            <int> 10335890, 10292904, 10400390, 10330448, 10356098, 10333175, 103370~
#> $ subproject_id      <chr> "5790", NA, NA, NA, NA, "7956", NA, NA, "8318", NA, NA, NA, NA, NA~
#> $ fiscal_year        <int> 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, ~
#> $ award_amount       <int> 244676, NA, 38717, 47003, 440201, 162193, 347296, 348376, 361634, ~
#> $ is_active          <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, ~
#> $ project_start_date <chr> "2020-02-28T05:00:00Z", "2018-10-01T04:00:00Z", "2022-01-01T05:00:~
#> $ project_title      <chr> "Project 2: Investigating the role of PAH exposures associated wit~
```

Some Vanilla Criteria

Many criteria are passed as vectors within the `criteria` list argument. We will cover some of the most useful examples:

Ex. 2 - Organization search

We can refine our query results by providing filtering criteria to `make_req()`, and by extension to the API. Suppose we want all currently active projects, funded in fiscal years 2017 through 2021, with a specific organization in mind (though we don't know exactly how its name will appear in RePORTER):

```
req <- make_req(criteria =
                  list(
                    fiscal_years = 2010:2011,
                    include_active_projects = TRUE,
                    org_names = c("Yale", "New Haven")
                  ),
                include_fields = c("Organization", "FiscalYear", "AwardAmount"),
                message = FALSE)
```

Here we are asking for any orgs containing the strings "yale" or "new haven" (ignoring case) - there are implied wildcards on either end of the strings we provide. This is the same as `org_name LIKE '%yale%' OR org_name LIKE '%new haven%'` in a SQL WHERE clause.

```
res <- get_nih_data(req, max_pages = 1)
#> Retrieving first page of results (up to 500 records)
#> max_pages set to 1 by user. Result set contains 8 pages. Only partial results will be retrieved.
res %>% glimpse(width = getOption("cli.width"))
#> Rows: 500
#> Columns: 3
#> $ fiscal_year  <int> 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, ~
#> $ organization <df[,17]> <data.frame[31 x 17]>
#> $ award_amount <int> 484894, 421533, 329765, 308745, 209375, 178489, 150000, 592057, 1776~
```

Notice the column `organization` is a nested data frame - it has 17 columns and always a single record. Setting `flatten_result = TRUE` in the call to `get_nih_data()` will flatten all such return fields, prefixing the original field name and returning with clean names (see `janitor::clean_names()`).

```
res <- get_nih_data(req,
                    max_pages = 1,
                    flatten_result = TRUE)
#> Retrieving first page of results (up to 500 records)
#> max_pages set to 1 by user. Result set contains 8 pages. Only partial results will be retrieved.

res %>% glimpse(width = getOption("cli.width"))
#> Rows: 500
#> Columns: 19
#> $ fiscal_year                    <int> 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, ~
#> $ award_amount                   <int> 484894, 421533, 329765, 308745, 209375, 178489, 150000~
#> $ organization_org_name          <chr> "YALE UNIVERSITY", "YALE UNIVERSITY", "YALE UNIVERSITY~
#> $ organization_city              <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
#> $ organization_country           <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
#> $ organization_org_city          <chr> "NEW HAVEN", "NEW HAVEN", "NEW HAVEN", "NEW HAVEN", "N~
#> $ organization_org_country       <chr> "UNITED STATES", "UNITED STATES", "UNITED STATES", "UN~
#> $ organization_org_state         <chr> "CT", "CT", "CT", "CT", "CT", "CT", "CT", "CT", "CT", ~
#> $ organization_org_state_name    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
#> $ organization_dept_type         <chr> "INTERNAL MEDICINE/MEDICINE", "INTERNAL MEDICINE/MEDIC~
#> $ organization_fips_country_code <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
#> $ organization_org_duns          <chr> "043207562", "043207562", "043207562", "043207562", "0~
#> $ organization_org_ueis          <chr> "FL6GV84CKN57", "FL6GV84CKN57", "FL6GV84CKN57", "FL6GV~
#> $ organization_primary_duns      <chr> "043207562", "043207562", "043207562", "043207562", "0~
#> $ organization_primary_uei       <chr> "FL6GV84CKN57", "FL6GV84CKN57", "FL6GV84CKN57", "FL6GV~
#> $ organization_org_fips          <chr> "US", "US", "US", "US", "US", "US", "US", "US", "US", ~
#> $ organization_org_ipf_code      <chr> "9420201", "9420201", "9420201", "9420201", "9420201",~
#> $ organization_org_zipcode       <chr> "065208327", "065208327", "065208327", "065208327", "0~
#> $ organization_external_org_id   <int> 9420201, 9420201, 9420201, 9420201, 9420201, 9420201, ~
```

Most users will prefer the flattened format above. It looks like Yale is busy, but it is not the only org matching our search.

```
res %>%
  group_by(organization_org_name) %>%
  summarise(project_count = n())
#> # A tibble: 2 x 2
#>   organization_org_name  project_count
#>   <chr>                           <int>
#> 1 UNIVERSITY OF NEW HAVEN             1
#> 2 YALE UNIVERSITY                   499
```

The `org_names_exact_match` criteria can be used as an alternative when we know the exact org name as it appears in RePORTER, if we want only that org's projects returned.

Ex. 3 - Geographic search

We can also filter projects by the geographic location (country/state/city) of the applicant organization.

```
## A valid request but probably not what we want
req <- make_req(criteria =
                  list(
                    fiscal_years = 2010:2011,
                    include_active_projects = TRUE,
                    org_cities = "New Haven",
                    org_states = "WY"
                  ),
                include_fields = c("Organization", "FiscalYear", "AwardAmount"),
                message = FALSE ## suppress printed message
)

res <- get_nih_data(req,
                    max_pages = 5,
                    flatten_result = TRUE)
#> Retrieving first page of results (up to 500 records)
#> Done - 0 records returned. Try a different search criteria.
```

Multiple criteria are usually connected by logical "AND" - there are no orgs based in the city of New Haven in Wyoming state (because it doesn't exist.)

```
req <- make_req(criteria =
                  list(
                    fiscal_years = 2015,
                    include_active_projects = TRUE,
                    org_states = "WY"
                  ),
                include_fields = c("ApplId", "Organization", "FiscalYear", "AwardAmount"),
                sort_field = "AwardAmount",
                sort_order = "desc",
                message = FALSE)

res <- get_nih_data(req,
                    flatten_result = TRUE)
#> Retrieving first page of results (up to 500 records)
```

```
res %>% glimpse(width = getOption("cli.width"))
#> Rows: 98
#> Columns: 20
#> $ appl_id                        <int> 8884461, 8898483, 10147717, 10201479, 10216275, 101477~
#> $ fiscal_year                    <int> 2015, 2015, 2021, 2021, 2021, 2021, 2015, 2015, 2021, ~
#> $ award_amount                   <int> 4957554, 3521553, 3418046, 2638712, 2004078, 1668381, ~
#> $ organization_org_name          <chr> "WYOMING STATE DEPARTMENT OF HEALTH", "UNIVERSITY OF W~
#> $ organization_city              <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
#> $ organization_country           <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
#> $ organization_org_city          <chr> "CHEYENNE", "LARAMIE", "LARAMIE", "LARAMIE", "LARAMIE"~
#> $ organization_org_country       <chr> "UNITED STATES", "UNITED STATES", "UNITED STATES", "UN~
#> $ organization_org_state         <chr> "WY", "WY", "WY", "WY", "WY", "WY", "WY", "WY", "WY", ~
#> $ organization_org_state_name    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
#> $ organization_dept_type         <chr> NA, "PHARMACOLOGY", "PHARMACOLOGY", "VETERINARY SCIENC~
#> $ organization_fips_country_code <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
#> $ organization_org_duns          <chr> "809915796", "069690956", "069690956", "069690956", "0~
#> $ organization_org_ueis          <chr> "JP1QRJYYJG73", "FDR5YF2K32X5", "FDR5YF2K32X5", "FDR5Y~
#> $ organization_primary_duns      <chr> "809915796", "069690956", "069690956", "069690956", "0~
#> $ organization_primary_uei       <chr> "JP1QRJYYJG73", "FDR5YF2K32X5", "FDR5YF2K32X5", "FDR5Y~
#> $ organization_org_fips          <chr> "US", "US", "US", "US", "US", "US", "US", "US", "US", ~
#> $ organization_org_ipf_code      <chr> "9408801", "9412601", "9412601", "9412601", "9412601",~
#> $ organization_org_zipcode       <chr> "820020001", "820712000", "820712000", "820712000", "8~
#> $ organization_external_org_id   <int> 9408801, 9412601, 9412601, 9412601, 9412601, 9412601, ~
```

Why are there projects from more recent years than 2015? Because the `include_active_projects` flag adds in active projects that match all criteria aside from `fiscal_years` (this appears to be the intended behavior by RePORTER).

Ex. 3 - Coronavirus/Covid-19 research

We already provided one example of this search criteria above. Let's mix it up and request all Covid response projects.

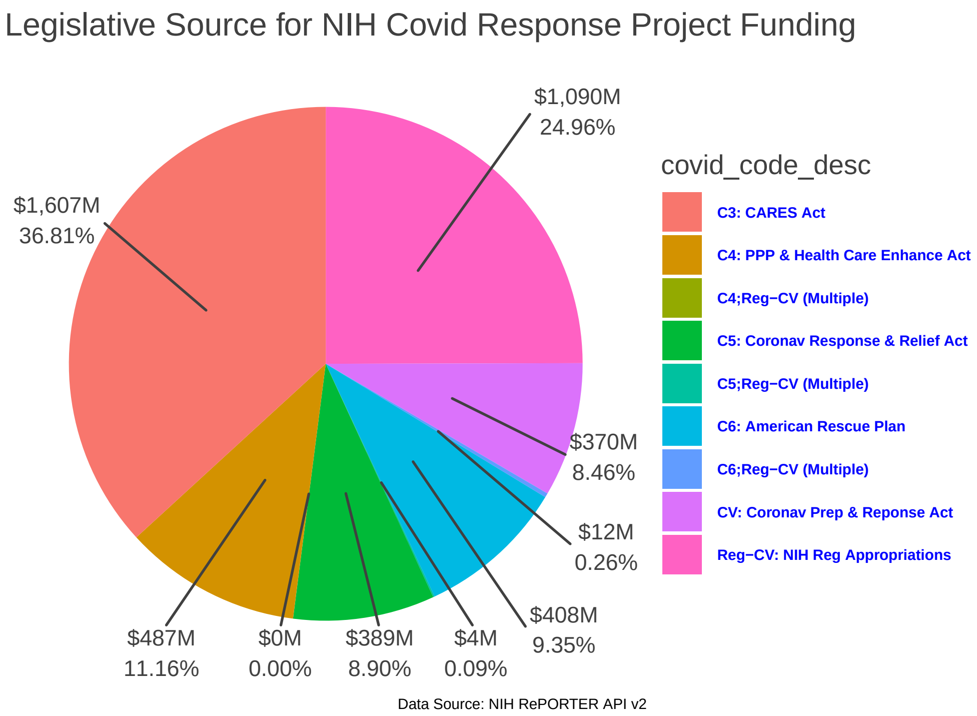
```
## all projects funded by the Paycheck Protection Act, Coronavirus Response and Relief Act,
## and American Rescue Plan, in fiscal year 2021
req <- make_req(criteria =
                  list(covid_response = c("All")),
                include_fields = nih_fields %>%
                  filter(payload_name %in% c("award_amount_range", "covid_response"))
                %>% pull(include_name))
#> This is your JSON payload:
#> {
#>     "criteria": {
#>         "covid_response": [
#>             "All"
#>         ],
#>         "use_relevance": false,
#>         "include_active_projects": false,
#>         "exclude_subprojects": false,
#>         "multi_pi_only": false,
#>         "newly_added_projects_only": false,
#>         "sub_project_only": false
#>     },
#>     "include_fields": [
#>         "AwardAmount",
#>         "CovidResponse"
#>     ],
#>     "offset": 0,
#>     "limit": 500
#> }
#>
#> If you receive a non-200 API response, compare this formatting (boxes, braces, quotes, etc.) to
    the 'Complete Payload' schema provided here:
#> https://api.reporter.nih.gov/?urls.primaryName=V2.0#/Search/post_v2_projects_search

res <- get_nih_data(req)
#> Retrieving first page of results (up to 500 records)
#> Retrieving results 501 to 1000 of 2572
#> Retrieving results 1001 to 1500 of 2572
#> Retrieving results 1501 to 2000 of 2572
#> Retrieving results 2001 to 2500 of 2572
#> Retrieving results 2501 to 2572 of 2572
res$covid_response %>% class()
#> [1] "list"
res$covid_response[[1]]
#> [1] "Reg-CV"
```

`covid_response` is a nested list (with character vectors of variable length) within the return tibble. We can use `flatten_result = TRUE` here - elements of each vector will be collapsed to a single string delimited by ";", massaging the list to a single character vector.

```
## all projects funded by the Paycheck Protection Act, Coronavirus Response and Relief Act,
## and American Rescue Plan, in fiscal year 2021
req <- make_req(criteria =
                  list(covid_response = c("All")),
                message = FALSE)

res <- get_nih_data(req,
                    flatten_result = TRUE)
#> Retrieving first page of results (up to 500 records)
#> Retrieving results 501 to 1000 of 2572
#> Retrieving results 1001 to 1500 of 2572
#> Retrieving results 1501 to 2000 of 2572
#> Retrieving results 2001 to 2500 of 2572
#> Retrieving results 2501 to 2572 of 2572

unique(res$covid_response)
#> [1] "Reg-CV"    "CV"        "C3"        "C4"        "C6"        "C6;Reg-CV" "C5"
#> [8] "C5;Reg-CV" "C4;Reg-CV"
```

Some projects are being funded from multiple sources. Summarizing all Covid-related project awards:

```
library(ggplot2)

res %>%
  left_join(covid_response_codes, by = "covid_response") %>%
  mutate(covid_code_desc = case_when(!is.na(fund_src) ~ paste0(covid_response, ": ", fund_src),
                                     TRUE ~ paste0(covid_response, " (Multiple)"))) %>%
  group_by(covid_code_desc) %>%
  summarise(total_awards = sum(award_amount) / 1e6) %>%
  ungroup() %>%
  arrange(desc(covid_code_desc)) %>%
  mutate(prop = total_awards / sum(total_awards),
         csum = cumsum(prop),
         ypos = csum - prop/2 ) %>%
  ggplot(aes(x = "", y = prop, fill = covid_code_desc)) +
  geom_bar(stat="identity") +
  geom_text_repel(aes(label =
                        paste0(dollar(total_awards,
                                      accuracy = 1,
                                      suffix = "M"),
                               "\n", percent(prop, accuracy = .01)),
                      y = ypos),
                  show.legend = FALSE,
                  nudge_x = .8,
                  size = 3, color = "grey25") +
  coord_polar(theta ="y") +
  theme_void() +
  theme(legend.position = "right",
        legend.title = element_text(colour = "grey25"),
        legend.text = element_text(colour="blue", size=6,
                                   face="bold"),
        plot.title = element_text(color = "grey25"),
        plot.caption = element_text(size = 6)) +
  labs(caption = "Data Source: NIH RePORTER API v2") +
  ggtitle("Legislative Source for NIH Covid Response Project Funding")
```



A second dataset is provided to translate the `covid_response` codes; it includes both the long-form and a shorter version of the source name.

```
data("covid_response_codes")
covid_response_codes %>% print
#> # A tibble: 6 x 3
#>   covid_response funding_source                                          fund_src
#>   <chr>          <chr>                                                   <chr>
#> 1 Reg-CV         NIH regular appropriations funding                      NIH Reg Appropriations
#> 2 CV             Coronavirus Preparedness and Response Supplemental App~ Coronav Prep & Repons~
#> 3 C3             CARES Act (Coronavirus Aid, Relief, and Economic Secur~ CARES Act
#> 4 C4             Paycheck Protection Program and Health Care Enhancemen~ PPP & Health Care Enh~
#> 5 C5             Coronavirus Response and Relief Supplemental Appropria~ Coronav Response & Re~
#> 6 C6             American Rescue Plan Act of 2021                        American Rescue Plan
```

Some Rocky Road Criteria

Other criteria provide search and filtering capability on many of the nested data elements. These criteria are passed as lists and must include a value for each of the named elements within.

Ex. 4 - Principal Investigator/Officer name search

The `pi_names` and `po_names` criteria allow the user to search for projects based on the first and last names of Principal Investigators and Principal Officers assigned. Each of these criteria must be provided as a list with three named character vector elements: `first_name`, `last_name`, and `any_name`. *Even if you only want to search on one of these name fields, you must provide the remaining elements as an empty string.* We will demonstrate with a search on PI name:

```
## projects funded in 2021 where the principal investigator first name
##  is "Michael" or begins with "Jo"
req <- make_req(criteria =
                  list(fiscal_years = 2021,
                       pi_names = list(first_name = c("Michael", "Jo*"),
                                       last_name = c(""), # must specify all pi_names elements
    always
                                       any_name = character(1))),
                include_fields = nih_fields %>%
                  filter(payload_name == "pi_names") %>%
                  pull(include_name),
                message = FALSE)

res <- get_nih_data(req,
                    max_pages = 1,
                    flatten_result = TRUE)
#> Retrieving first page of results (up to 500 records)
#> max_pages set to 1 by user. Result set contains 13 pages. Only partial results will be
    retrieved.

res %>% glimpse(width = getOption("cli.width"))
#> Rows: 500
#> Columns: 2
#> $ principal_investigators <list> [<data.frame[3 x 7]>], [<data.frame[3 x 7]>], [<data.frame[2~
#> $ contact_pi_name         <chr> "BURG, JONATHAN MICHAEL", "IX, JOACHIM H", "MOSHER, JOHN COMP~
```

Here we searched for any projects with a PI first-named "Michael" or beginning with "Jo" - the `"*"` is a wildcard operator.

Note that the first column in the return is a list of data frames of *variable height* (not a nested `data.frame`) - we leave such returned elements to the user to handle extraction/formatting - flattening is only performed for lists of atomic vectors and nested data frames.

Ex. 5 - Advanced text search

RePORTER allows users to search the project title, abstract, and tags for specific terms or phrases. You can access this capability with the `advanced_text_search` criteria - a named list with three elements:

- `operator` may be either "and", "or", or "advanced" - and/or will specify the logical operator connecting multiple search terms. "advanced" allows the user to pass a boolean search string directly;
- `search_field` can be any or multiple of "terms", "abstract", "projecttitle." To search all items, specify "all" or `""` (a length 1 vector with an empty string);
- `search_text` may be either a length 1 character vector of space-delimited search terms (when using "and" or "or" for the operator argument - the logical operator is inserted between all search terms); or it may be a boolean search string (when specifying "advanced" for the operator argument).

```
## using advanced_text_search with boolean search string
req <- make_req(criteria =
                  list(advanced_text_search =
                         list(operator = "advanced",
                              search_field = c("terms", "abstract"),
                              search_text = "(head AND trauma) OR \"brain damage\" AND NOT
    \"psychological\"")),
                include_fields = c("ProjectTitle", "AbstractText", "Terms") )
#> This is your JSON payload:
#> {
#>     "criteria": {
#>         "advanced_text_search": {
#>             "operator": "advanced",
#>             "search_field": "terms,abstract",
#>             "search_text": "(head AND trauma) OR \"brain damage\" AND NOT \"psychological\""
#>         },
#>         "use_relevance": false,
#>         "include_active_projects": false,
#>         "exclude_subprojects": false,
#>         "multi_pi_only": false,
#>         "newly_added_projects_only": false,
#>         "sub_project_only": false
#>     },
#>     "include_fields": [
#>         "ProjectTitle",
#>         "AbstractText",
#>         "Terms"
#>     ],
#>     "offset": 0,
#>     "limit": 500
#> }
#>
#> If you receive a non-200 API response, compare this formatting (boxes, braces, quotes, etc.) to
    the 'Complete Payload' schema provided here:
#> https://api.reporter.nih.gov/?urls.primaryName=V2.0#/Search/post_v2_projects_search

res <- get_nih_data(req, max_pages = 1)
#> Retrieving first page of results (up to 500 records)
#> max_pages set to 1 by user. Result set contains 37 pages. Only partial results will be
    retrieved.
```

Let's inspect the fields we searched from one of these results:

```
one_rec <- res %>%
  slice(5) %>%
  mutate(abstract_text = gsub("[\r\n]", " ", abstract_text))

one_rec %>% pull(project_title) %>% print
#> [1] "Delayed white matter loss in concussive head injuries and its treatment"
```

```
## substr to avoid LaTeX error exceeding char limit
one_rec %>% pull(abstract_text) %>% substr(1, 85) %>% print
#> [1] "Abstract. Half of all traumatic brain injuries cause changes in white matter integrit"
```

```
one_rec %>% pull(terms) %>% substr(1, 85) %>% print
#> [1] "Action Potentials ; Intranasal Drug Administration ; Intranasal Administration ; Anim"
```

Large Result Sets

The RePORTER API provides no direct way to obtain complete result sets when searches yield over 10,000 records. `get_nih_data()` provides the `return_meta` argument which is defaulted to `FALSE`. When set to `TRUE` and combined with a little programming, you can easily obtain full result sets well beyond the 10K limit. One approach may be the following:

1. Obtain a sample from your full result set by making the query you desire and calling `get_nih_data()` with `max_pages = 1` (or some small number of pages); also set `return_meta = TRUE` in order to determine the total number of records in the full result set
2. Calculate quantiles for the sample distribution of the `award_amount` column
   - Set the # of quantiles such that you can confidently infer that the number of records within each range from the full result set will contain <10K records
3. Iterate over your quantiles making separate requests, passing the endpoints of each quantile to `award$award_amount_range` criteria
   - Wait until the end to flatten results since some columns may flatten differently on smaller individual result sets, causing problems in combining them
4. Bind your list of results together
5. Flatten the complete result set, if desired

Below is an implementation of the above logic:

```
all_res <- list()
for(y in 2017:2021) { ## five years to loop over, each year is ~80K records
  ## We only need the AwardAmount for quantiles
  req_sample <- make_req(criteria = list(fiscal_years = y),
                         include_fields = "AwardAmount")

  ## get a sample of the result set - 1000 records should be enough
  ## return the metadata
  res_sample <- get_nih_data(req_sample, max_pages = 2, return_meta = TRUE)

  res_sample$meta$total %>% print ## there are 73,142 project records

  ## deciles of award amount - each decile should contain ~7,314.2 records, approximately
  qtiles <- res_sample$records %>% pull(award_amount) %>% quantile(na.rm = TRUE, probs = seq(.1,
    1, .1))

  ## list for qtile results (full year)
  this_res <- list()
  ## for each qtile
  for (i in 1:length(qtiles)) {
    if (i == 1) {
      award_min <- 0
    } else {
      award_min <- ceiling(qtiles[i-1])+.01
    }
    if (i == length(qtiles)) {
      award_max <- 1e9 ## arbitrarily huge
    } else {
      award_max <- ceiling(qtiles[i])
    }
    req <- make_req(criteria = list(fiscal_years = y,
                                    award = list(award_notice_date = "",
                                                 award_notice_opr = "",
                                                 award_amount_range = list(min_amount = award_min,
                                                                           max_amount =
    award_max))))
    ## result set for quantile
    this_res[[i]] <- get_nih_data(req, flatten_result = FALSE)
  }
  ## list of result sets for the year
  yr_res[[y %>% as.character()]] <- this_res
}

## shape it up
all_res <- unlist(yr_res, recursive = FALSE) %>%
  bind_rows() %>%
  flatten(recursive = FALSE) %>%
  clean_names()

## pull out everything that is flat
flat_columns <- all_res %>%
  select_if(is.atomic)

## everything that isn't
annoying_columns <- all_res %>%
  select_if(!is.atomic)
```

Additional Resources

- The RePORTER web interface and official API documentation are useful for getting familiar with available search parameters
- …and the homepage with further examples/documentation is here
- Information on NIH study sections, IRGs, etc. is here
- h/t to Chris whose code on github was all I could find existing in R and served as a starting point for this work

Session Information

The version number of R and packages loaded for generating the vignette were:

```
sessionInfo()
#> R version 4.1.0 (2021-05-18)
#> Platform: x86_64-pc-linux-gnu (64-bit)
#> Running under: Ubuntu 20.04.2 LTS
#> 
#> Matrix products: default
#> BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
#> LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
#> 
#> locale:
#>  [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8       
#>  [4] LC_COLLATE=C           LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8   
#>  [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C          
#> [10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C   
#> 
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base     
#> 
#> other attached packages:
#> [1] repoRter.nih_0.1.1 tufte_0.12         scales_1.1.1       dplyr_1.0.7       
#> [5] ggrepel_0.9.1      ggplot2_3.3.5      tibble_3.1.3      
#> 
#> loaded via a namespace (and not attached):
```

```
#>  [1] Rcpp_1.0.7        highr_0.9         pillar_1.6.2      compiler_4.1.0    tools_4.1.0
#>  [6] digest_0.6.27     jsonlite_1.7.2    lubridate_1.7.10  evaluate_0.14     lifecycle_1.0.0
#> [11] gtable_0.3.0      pkgconfig_2.0.3   rlang_0.4.11      rstudioapi_0.13   cli_3.0.1
#> [16] DBI_1.1.1         curl_4.3.2        yaml_2.2.1        xfun_0.24         withr_2.4.2
#> [21] stringr_1.4.0     httr_1.4.2        knitr_1.33        janitor_2.1.0     generics_0.1.0
#> [26] vctrs_0.3.8       grid_4.1.0        tidyselect_1.1.1  glue_1.4.2        snakecase_0.11.0
#> [31] R6_2.5.0          fansi_0.5.0       rmarkdown_2.11    farver_2.1.0      purrr_0.3.4
#> [36] magrittr_2.0.1    codetools_0.2-18  ellipsis_0.3.2    htmltools_0.5.1.1 assertthat_0.2.1
#> [41] colorspace_2.0-2  labeling_0.4.2    utf8_1.2.2        stringi_1.7.3     munsell_0.5.0
#> [46] crayon_1.4.1
```