APPENDIX: {peacesciencer}: An R Package for Quantitative Peace Science Research

Contents

Workflow Citations

The following tools and libraries were integral to the creation of the manuscript and I want to at least acknowledge them here. Wickham et al.'s (2019) `{tidyverse}` was the primary toolkit for most data transformation techniques outlined here, and indeed forms the implicit foundation of the package itself.[1] Regression tables were formatted in `{modelsummary}` (Arel-Bundock, 2021), which itself uses and suggests `{kableExtra}` as a back-end for presentation (Zhu, 2021). The document itself is a dynamic document using familiar R Markdown syntax (Xie, Dervieux & Riederer, 2020), knitted to various outputs (Xie, 2015), using `{bookdown}` for additional functionality (Xie, 2016).

I offer these citations first as an acknowledgment of these important contributions to our collective research productivity. Space restrictions in our journal preclude us from citing these important tools, even if they are truly ancillary to the product they help produce. No matter, I intend to put this appendix on my website to give Google Scholar the opportunity to find these citations and count them toward the *h*-indexes of the researchers responsible for these important tools.

Vignette: "Create Different Kinds of Data in `{peacesciencer}`

This tutorial is a companion to the user guide, which shows how to create different kinds of data in `{peacesciencer}`. However, space considerations (for ideal publication in a peer-reviewed journal) preclude the full "knitting" experience (i.e. giving the user a preview of what the data look like). What follows is a brief guide that expands on the tutorial section of that user guide for creating different kinds of data in `{peacesciencer}`.

This vignette will lean on the `{tidyverse}` package, which will be included in almost anything you should do (optimally) with `{peacesciencer}`. I will also load `{lubridate}`. Internal functions in `{peacesciencer}` use `{lubridate}`—it is a formal dependency of `{peacesciencer}`—but users may want to load it for doing some additional stuff outside of `{peacesciencer}`.

```
library(tidyverse)
library(peacesciencer)
library(lubridate)
```

State-Year Data

The most basic form of data `{peacesciencer}` creates is state-year, by way of `create_stateyears()`. `create_stateyears()` has two arguments: `system` and `mry`. `system` takes either "cow" or "gw", depending on whether the user wants Correlates of War state years or Gleditsch-Ward state-years. It defaults to "cow" in the absence of a user-specified override given the prominence of Correlates of War data in the peace science ecosystem. `mry` takes a logical (`TRUE` or `FALSE`), depending on whether the user wants the function to extend to the most recently concluded calendar year (2020). The Correlates of War state system data extend to the end of 2016 while the Gleditsch-Ward state system extend to the end of the 2017. This argument will allow the researcher to extend the data a few years, under the (reasonable) assumption there have been no fundamental

[1] The appendix will reference component packages of `{tidyverse}`, but follows the encouragement of Wickham et al. (2019) to cite the overall suite of packages outlined in Wickham et al. (2019) rather than reference the packages individually.

composition to the state system since these data sets were last updated. `mry` defaults to `TRUE` in the absence of a user-specified override.

This will create Correlates of War state-year data from 1816 to 2020.

```
create_stateyears()
#> # A tibble: 16,731 x 3
#>    ccode statenme                 year
#>    <dbl> <chr>                   <int>
#>  1     2 United States of America  1816
#>  2     2 United States of America  1817
#>  3     2 United States of America  1818
#>  4     2 United States of America  1819
#>  5     2 United States of America  1820
#>  6     2 United States of America  1821
#>  7     2 United States of America  1822
#>  8     2 United States of America  1823
#>  9     2 United States of America  1824
#> 10     2 United States of America  1825
#> # ... with 16,721 more rows
```

This will create Gleditsch-Ward state-year data from 1816 to 2017.

```
create_stateyears(system = "gw", mry = FALSE)
#> # A tibble: 17,767 x 3
#>    gwcode statename                year
#>     <dbl> <chr>                   <int>
#>  1      2 United States of America  1816
#>  2      2 United States of America  1817
#>  3      2 United States of America  1818
#>  4      2 United States of America  1819
#>  5      2 United States of America  1820
#>  6      2 United States of America  1821
#>  7      2 United States of America  1822
#>  8      2 United States of America  1823
#>  9      2 United States of America  1824
#> 10      2 United States of America  1825
#> # ... with 17,757 more rows
```

Dyad-Year Data

`create_dyadyears()` is one of the most useful functions in {peacesciencer}, transforming the raw Correlates of War state system data (`cow_states` in {peacesciencer}) or Gleditsch-Ward state system data (`gw_states`) into all possible dyad-years. It has three arguments. `system` and `mry` operate the same as they do in `create_stateyears()`. There is an additional argument—`directed`—that also takes a logical (`TRUE` or `FALSE`). The default here is `TRUE`, returning *directed* dyad-year data (useful for dyadic conflict analyses where the initiator/target distinction matters). `FALSE` returns *non-directed* dyad-year data, useful for cases where the

initiator/target distinction does not matter and the researcher cares more about the presence or absence of a conflict. The convention for non-directed dyad-year data is that `ccode2 > ccode1` and the underlying code of `create_dyadyears()` simply takes the directed dyad-year data and lops it in half with that rule.

Here are all Correlates of War dyad-years from 1816 to 2020.

```
create_dyadyears()
#> # A tibble: 2,063,610 x 3
#>    ccode1 ccode2  year
#>     <dbl>  <dbl> <int>
#>  1      2     20  1920
#>  2      2     20  1921
#>  3      2     20  1922
#>  4      2     20  1923
#>  5      2     20  1924
#>  6      2     20  1925
#>  7      2     20  1926
#>  8      2     20  1927
#>  9      2     20  1928
#> 10      2     20  1929
#> # ... with 2,063,600 more rows
```

Here are all Gleditsch-Ward dyad-years with the same temporal domain.

```
create_dyadyears(system = "gw")
#> # A tibble: 2,029,622 x 3
#>    gwcode1 gwcode2  year
#>      <dbl>   <dbl> <int>
#>  1       2      20  1867
#>  2       2      20  1868
#>  3       2      20  1869
#>  4       2      20  1870
#>  5       2      20  1871
#>  6       2      20  1872
#>  7       2      20  1873
#>  8       2      20  1874
#>  9       2      20  1875
#> 10       2      20  1876
#> # ... with 2,029,612 more rows
```

Dyadic Dispute-Year Data

Dyadic dispute-year data come pre-processed in `{peacesciencer}`. Another vignette show how these are transformed to true dyad-year data, but they are also available for analysis. For example, the (directed) dyadic dispute-year Gibler-Miller-Little (GML) MID data are available as `gml_dirdisp`. Here, we can add information to these dyadic dispute-years to identify contiguity relationships and Correlates of War major status.

```
gml_dirdisp %>% add_contiguity() %>% add_cow_majors()
#> # A tibble: 10,276 x 42
#>    dispnum ccode1 ccode2  year midongoing midonset sidea1 sidea2 revstate1
#>      <dbl>  <dbl>  <dbl> <dbl>      <dbl>    <dbl>  <dbl>  <dbl>     <dbl>
#>  1       2      2    200  1902          1        1      1      0         1
#>  2       2    200      2  1902          1        1      0      1         1
#>  3       3    300    345  1913          1        1      1      0         1
#>  4       3    345    300  1913          1        1      0      1         0
#>  5       4    200    339  1946          1        1      0      1         0
#>  6       4    339    200  1946          1        1      1      0         0
#>  7       7    200    651  1951          1        1      1      0         0
#>  8       7    200    651  1952          1        0      1      0         0
#>  9       7    651    200  1951          1        1      0      1         1
#> 10       7    651    200  1952          1        0      0      1         1
#> # ... with 10,266 more rows, and 33 more variables: revstate2 <dbl>,
#> #   revtype11 <dbl>, revtype12 <dbl>, revtype21 <dbl>, revtype22 <dbl>,
#> #   fatality1 <dbl>, fatality2 <dbl>, fatalpre1 <dbl>, fatalpre2 <dbl>,
#> #   hiact1 <dbl>, hiact2 <dbl>, hostlev1 <dbl>, hostlev2 <dbl>, orig1 <dbl>,
#> #   orig2 <dbl>, hiact <dbl>, hostlev <dbl>, mindur <dbl>, maxdur <dbl>,
#> #   outcome <dbl>, settle <dbl>, fatality <dbl>, fatalpre <dbl>, stmon <dbl>,
#> #   endmon <dbl>, recip <dbl>, numa <dbl>, numb <dbl>, ongo2010 <dbl>, ...
```

Users interested in the Correlates of War MID data will have this available for use as `cow_mid_dirdisps`. Future updates may change the object names for better standardization, but this is how it is now.

State-Day Data

{peacesciencer} comes with a `create_statedays()` function. This is admittedly more proof of concept as it is *really* difficult to conjure too many *daily* data sets in peace science, certainly with coverage into the 19th century. No matter, `create_statedays()` will create these data. It too has the same `system` and `mry` arguments (and same defaults) as `create_stateyears()`.

Here are all Correlates of War state-days from 1816 to 2020.

```
create_statedays()
#> # A tibble: 6,061,091 x 3
#>    ccode statenme                date
#>    <dbl> <chr>                   <date>
#>  1     2 United States of America 1816-01-01
#>  2     2 United States of America 1816-01-02
#>  3     2 United States of America 1816-01-03
#>  4     2 United States of America 1816-01-04
#>  5     2 United States of America 1816-01-05
#>  6     2 United States of America 1816-01-06
#>  7     2 United States of America 1816-01-07
#>  8     2 United States of America 1816-01-08
```

```
#>  9     2 United States of America 1816-01-09
#> 10     2 United States of America 1816-01-10
#> # ... with 6,061,081 more rows
```

Here are all Gleditsch-Ward state-days with the same temporal domain.

```
create_statedays(system = "gw")
#> # A tibble: 6,638,781 x 3
#>    gwcode statename                date
#>     <dbl> <chr>                    <date>
#>  1      2 United States of America 1816-01-01
#>  2      2 United States of America 1816-01-02
#>  3      2 United States of America 1816-01-03
#>  4      2 United States of America 1816-01-04
#>  5      2 United States of America 1816-01-05
#>  6      2 United States of America 1816-01-06
#>  7      2 United States of America 1816-01-07
#>  8      2 United States of America 1816-01-08
#>  9      2 United States of America 1816-01-09
#> 10      2 United States of America 1816-01-10
#> # ... with 6,638,771 more rows
```

I can conjure an application where a user may want to think of daily conflict episodes within the Gleditsch-Ward domain. The UCDP armed conflict data have more precise dates than, say, the Correlates of War MID data, making such an analysis possible. However, there are no conflict data before 1946 and you should reflect that with {peacesciencer} with something like this. This will require {lubridate}.

```
create_statedays(system = "gw") %>%
  filter(year(date) >= 1946)
#> # A tibble: 3,870,980 x 3
#>    gwcode statename                date
#>     <dbl> <chr>                    <date>
#>  1      2 United States of America 1946-01-01
#>  2      2 United States of America 1946-01-02
#>  3      2 United States of America 1946-01-03
#>  4      2 United States of America 1946-01-04
#>  5      2 United States of America 1946-01-05
#>  6      2 United States of America 1946-01-06
#>  7      2 United States of America 1946-01-07
#>  8      2 United States of America 1946-01-08
#>  9      2 United States of America 1946-01-09
#> 10      2 United States of America 1946-01-10
#> # ... with 3,870,970 more rows
```

State-Month Data

State-months are simple aggregations of state-days. You can accomplish this with a few more extra commands after `create_statedays()`.

```
create_statedays(system = "gw") %>%
  mutate(year = year(date),
         month = month(date)) %>%
  distinct(gwcode, statename, year, month)
#> # A tibble: 218,194 x 4
#>    gwcode statename                year month
#>     <dbl> <chr>                   <dbl> <dbl>
#>  1      2 United States of America  1816     1
#>  2      2 United States of America  1816     2
#>  3      2 United States of America  1816     3
#>  4      2 United States of America  1816     4
#>  5      2 United States of America  1816     5
#>  6      2 United States of America  1816     6
#>  7      2 United States of America  1816     7
#>  8      2 United States of America  1816     8
#>  9      2 United States of America  1816     9
#> 10      2 United States of America  1816    10
#> # ... with 218,184 more rows
```

State-Quarter Data

There is some assumption about what a "quarter" would look like in a more general context, but it might look something like this. Again, this is an aggregation of `create_statedays()`.

```
create_statedays(system = "gw") %>%
  mutate(year = year(date),
         month = month(date)) %>%
  filter(month %in% c(1, 4, 7, 10)) %>%
  mutate(quarter = case_when(
    month == 1 ~ "Q1",
    month == 4 ~ "Q2",
    month == 7 ~ "Q3",
    month == 10 ~ "Q4"
  )) %>%
  distinct(gwcode, statename, year, quarter)
#> # A tibble: 72,687 x 4
#>    gwcode statename                year quarter
#>     <dbl> <chr>                   <dbl> <chr>
#>  1      2 United States of America  1816 Q1
#>  2      2 United States of America  1816 Q2
#>  3      2 United States of America  1816 Q3
```

```
#>  4      2 United States of America  1816 Q4
#>  5      2 United States of America  1817 Q1
#>  6      2 United States of America  1817 Q2
#>  7      2 United States of America  1817 Q3
#>  8      2 United States of America  1817 Q4
#>  9      2 United States of America  1818 Q1
#> 10      2 United States of America  1818 Q2
#> # ... with 72,677 more rows
```

Leader-Day (Leader-Month, Leader-Year) Data

{peacesciencer} has leader-level units of analysis as well, which can be easily created with the modified Archigos (archigos) data in {peacesciencer}. The data are version 4.1.

```
archigos
#> # A tibble: 3,409 x 11
#>    obsid  gwcode leadid   leader yrborn gender startdate  enddate    entry exit
#>    <chr>   <dbl> <chr>    <chr>   <dbl> <chr>  <date>     <date>     <chr> <chr>
#>  1 USA-1~      2 81dcc17~ Grant    1822 M      1869-03-04 1877-03-04 Regu~ Regu~
#>  2 USA-1~      2 81dcc17~ Hayes    1822 M      1877-03-04 1881-03-04 Regu~ Regu~
#>  3 USA-1~      2 81dcf24~ Garfi~   1831 M      1881-03-04 1881-09-19 Regu~ Irre~
#>  4 USA-1~      2 81dcf24~ Arthur   1829 M      1881-09-19 1885-03-04 Regu~ Regu~
#>  5 USA-1~      2 34fb155~ Cleve~   1837 M      1885-03-04 1889-03-04 Regu~ Regu~
#>  6 USA-1~      2 81dcf24~ Harri~   1833 M      1889-03-04 1893-03-04 Regu~ Regu~
#>  7 USA-1~      2 34fb155~ Cleve~   1837 M      1893-03-04 1897-03-04 Regu~ Regu~
#>  8 USA-1~      2 81dcf24~ McKin~   1843 M      1897-03-04 1901-09-14 Regu~ Irre~
#>  9 USA-1~      2 81dd231~ Roose~   1858 M      1901-09-14 1909-03-04 Regu~ Regu~
#> 10 USA-1~      2 81dd231~ Taft     1857 M      1909-03-04 1913-03-04 Regu~ Regu~
#> # ... with 3,399 more rows, and 1 more variable: exitcode <chr>
```

create_leaderdays() will create leader-day data from archigos.

```
create_leaderdays()
#> # A tibble: 5,298,380 x 5
#>    obsid    gwcode leader date       yrinoffice
#>    <chr>     <dbl> <chr>  <date>          <dbl>
#>  1 USA-1869      2 Grant  1869-03-04          1
#>  2 USA-1869      2 Grant  1869-03-05          1
#>  3 USA-1869      2 Grant  1869-03-06          1
#>  4 USA-1869      2 Grant  1869-03-07          1
#>  5 USA-1869      2 Grant  1869-03-08          1
#>  6 USA-1869      2 Grant  1869-03-09          1
#>  7 USA-1869      2 Grant  1869-03-10          1
#>  8 USA-1869      2 Grant  1869-03-11          1
#>  9 USA-1869      2 Grant  1869-03-12          1
#> 10 USA-1869      2 Grant  1869-03-13          1
```

```
#> # ... with 5,298,370 more rows
```

I do want to note one thing about the leader-level functions in this package. Whereas Correlates of War state system membership is often the default system for a lot of functions (prominently `create_stateyears()` and `create_dyadyears()`), the Gleditsch-Ward system is the default system because that is the state system around which the Archigos project created its leader data. Moreover, the leader data isn't exactly tethered to the Gleditsch-Ward state system for dates either (e.g. there are leader entries for Gleditsch-Ward states that aren't in the system yet). In a case like this, you can standardize these leader data to either the Correlates of War system or the Gleditsch-Ward system with the `standardize` argument. By default, the option here is "none" (i.e. return all available leader days recorded in the Archigos data). "cow" or "gw" standardizes the leader data to Correlates of War state system membership or Gleditsch-Ward state system membership, respectively.

```
create_leaderdays(standardize = "cow")
#> # A tibble: 4,824,967 x 5
#>    obsid    ccode leader date       yrinoffice
#>    <chr>    <dbl> <chr>  <date>          <dbl>
#>  1 USA-1869     2 Grant  1869-03-04          1
#>  2 USA-1869     2 Grant  1869-03-05          1
#>  3 USA-1869     2 Grant  1869-03-06          1
#>  4 USA-1869     2 Grant  1869-03-07          1
#>  5 USA-1869     2 Grant  1869-03-08          1
#>  6 USA-1869     2 Grant  1869-03-09          1
#>  7 USA-1869     2 Grant  1869-03-10          1
#>  8 USA-1869     2 Grant  1869-03-11          1
#>  9 USA-1869     2 Grant  1869-03-12          1
#> 10 USA-1869     2 Grant  1869-03-13          1
#> # ... with 4,824,957 more rows
```

The user may want to think about some additional post-processing on top of this, but this is enough to get started. From there, the same process that creates state-months can create something like leader-months.

```
create_leaderdays() %>%
  mutate(year = year(date),
         month = month(date)) %>%
  group_by(gwcode, obsid, year, month) %>%
  slice(1)
#> # A tibble: 177,128 x 7
#> # Groups:   gwcode, obsid, year, month [177,128]
#>    obsid    gwcode leader date       yrinoffice  year month
#>    <chr>     <dbl> <chr>  <date>          <dbl> <dbl> <dbl>
#>  1 USA-1869      2 Grant  1869-03-04          1  1869     3
#>  2 USA-1869      2 Grant  1869-04-01          1  1869     4
#>  3 USA-1869      2 Grant  1869-05-01          1  1869     5
#>  4 USA-1869      2 Grant  1869-06-01          1  1869     6
#>  5 USA-1869      2 Grant  1869-07-01          1  1869     7
#>  6 USA-1869      2 Grant  1869-08-01          1  1869     8
#>  7 USA-1869      2 Grant  1869-09-01          1  1869     9
```

```
#>  8 USA-1869      2 Grant  1869-10-01          1  1869    10
#>  9 USA-1869      2 Grant  1869-11-01          1  1869    11
#> 10 USA-1869      2 Grant  1869-12-01          1  1869    12
#> # ... with 177,118 more rows
```

And here are leader-years, which is pre-packaged as a `{peacesciencer}` function. The package also adds some information about leader gender, an approximation of the leader's age that year (i.e. `year - yrborn`), and a running count (starting a 1) for the leader's tenure (in years).

```
create_leaderyears()
#> # A tibble: 17,686 x 7
#>    obsid    gwcode leader gender  year yrinoffice leaderage
#>    <chr>     <dbl> <chr>  <chr>  <dbl>      <dbl>     <dbl>
#>  1 USA-1869      2 Grant  M       1869          1        47
#>  2 USA-1869      2 Grant  M       1870          2        48
#>  3 USA-1869      2 Grant  M       1871          3        49
#>  4 USA-1869      2 Grant  M       1872          4        50
#>  5 USA-1869      2 Grant  M       1873          5        51
#>  6 USA-1869      2 Grant  M       1874          6        52
#>  7 USA-1869      2 Grant  M       1875          7        53
#>  8 USA-1869      2 Grant  M       1876          8        54
#>  9 USA-1869      2 Grant  M       1877          9        55
#> 10 USA-1877      2 Hayes  M       1877          1        55
#> # ... with 17,676 more rows
```

Leader Dyad-Year Data

`{peacesciencer}` can also create leader dyad-year data by way of `create_leaderdyadyears()`. You can see some of the underlying code that is creating these data. It's a lot of code, it would take a lot of time to run from scratch, and the ensuing output is too large to store as an R data object in the package because CRAN hard-caps package size at 5 MB. Instead, users who want these data should first run `download_extdata()` when they *first* install or update the package. Therein, they can run `create_leaderdyadyears()` to create the full universe of leader dyad-year data.

```
# create_leaderdyadyears() is effectively doing this.
# Let's do the G-W leader dyad-year data for illustration's sake.
# Do note: `download_extdata()` will download these data and stick them in the package directo
# Thus, it is *not* downloading the data fresh each time.

readRDS(url("http://svmiller.com/R/peacesciencer/gw_dir_leader_dyad_years.rds")) %>%
  declare_attributes(data_type = "leader_dyad_year", system = "gw")
#> # A tibble: 2,336,990 x 11
#>     year obsid1   obsid2   gwcode1 gwcode2 gender1 gender2 leaderage1 leaderage2
#>    <int> <chr>    <chr>      <dbl>   <dbl> <chr>   <chr>        <dbl>      <dbl>
#>  1  1870 AFG-1868 AUH-1848     700     300 M       M               45         40
#>  2  1870 AFG-1868 BAV-1864     700     245 M       M               45         39
```

```
#>  3  1870 AFG-1868 BRA-1840     700     140 M       M                45         45
#>  4  1870 AFG-1868 CHN-1861     700     710 M       M                45         35
#>  5  1870 AFG-1868 COS-1870     700      94 M       M                45         39
#>  6  1870 AFG-1868 ECU-1869     700     130 M       M                45         49
#>  7  1870 AFG-1868 GMY-1858     700     255 M       M                45         73
#>  8  1870 AFG-1868 GRC-1863     700     350 M       M                45         25
#>  9  1870 AFG-1868 IRN-1848     700     630 M       M                45         39
#> 10  1870 AFG-1868 JPN-1868     700     740 M       M                45         18
#> # ... with 2,336,980 more rows, and 2 more variables: yrinoffice1 <dbl>,
#> #   yrinoffice2 <dbl>

# ^ compare with:
# download_extdata()
# create_leaderdyadyears()
```

Vignette: A Discussion of Various Joins in {peacesciencer}

```
library(tidyverse)
library(lubridate)
library(peacesciencer)
```

Users who may wish to improve their own data management skills in R by looking how {peacesciencer} functions are written will see that basic foundation of {peacesciencer}'s functions are so-called "join" functions. The "join" functions themselves come in {dplyr}, a critical dependency for {peacesciencer} and the effective engine of {tidyverse} (which I suggest for a basic workflow tool, and which the user may already be using). Users who have absolutely no idea what these functions are welcome to find more thorough texts about these different types of joins. Their functionality and terminology have a clear basis in SQL, a relational database management system that first appeared in 1974 for data management and data manipulation. My goal here is not to offer a crash course on all these potential "join" functions, though helpful visual primers are available in R and SQL. Instead, I will offer the basic principles these visual primers are communicating as they apply to {peacesciencer}.

Left (Outer) Join

The first type of join is the most important type of join function in {peacesciencer}. Indeed, almost every function in this package that deals with adding variables to a type of data created in {peacesciencer} includes it. This is the "left join" (left_join() in {dplyr}), alternatively known as the "outer join" or "left outer join" in the SQL context, and is a type of "mutating join" in the {tidyverse} context. In plain English, the left_join() assumes two data objects—a "left" object (x) and a "right" object (y)—and returns all rows from the left object (x) with matching information in the right object (y) by a set of common matching keys (or columns in both x and y).

Here is a simple example of how this works in the {peacesciencer} context. Assume a simple state-year data set of the United States (ccode: 2), Canada (ccode: 20), and the United Kingdom (ccode: 200) for

all years from 2016 to 2020. Recreating this simple kind of data is no problem in R and will serve as our "left object" (x) for this simple example.

```
tibble(ccode = c(2, 20, 200)) %>%
  # rowwise() is a great trick for nesting sequences in tibbles
  # This parlor trick, for example, generates state-year data out of raw state
  # data in create_stateyears()
  rowwise() %>%
  # create a sequence as a nested column
  mutate(year = list(seq(2016, 2020))) %>%
  # unnest the column
  unnest(year) -> x

x
#> # A tibble: 15 x 2
#>    ccode  year
#>    <dbl> <int>
#>  1     2  2016
#>  2     2  2017
#>  3     2  2018
#>  4     2  2019
#>  5     2  2020
#>  6    20  2016
#>  7    20  2017
#>  8    20  2018
#>  9    20  2019
#> 10    20  2020
#> 11   200  2016
#> 12   200  2017
#> 13   200  2018
#> 14   200  2019
#> 15   200  2020
```

Let's assume we're building toward the kind of state-year analysis I describe in the manuscript accompanying this package. For example, the canonical civil conflict analysis by Fearon and Laitin (2003) has an outcome that varies by year, but several independent variables that are time-invariant and serve as variables for making state-to-state comparisons in their model of civil war onset (e.g ethnic fractionalization, religious fractionalization, terrain ruggedness). In a similar manner, we have a basic ranking of the United States, Canada, and the United Kingdom in our case. Minimally, the United States scores "low", Canada scores "medium", and the United Kingdom scores "high" on some metric. There is no variation by time in this simple example.

```
tibble(ccode = c(2, 20, 200),
       ranking = c("low", "medium", "high")) -> y

y
#> # A tibble: 3 x 2
#>   ccode ranking
```

```
#>   <dbl> <chr>
#> 1     2 low
#> 2    20 medium
#> 3   200 high
```

This is the "right object" (y) that we want to add to the "left object" that serves as our main data frame. Notice that x has no variable for the ranking information we want. It does, however, have matching observations for the state identifiers corresponding with the Correlates of War state codes for the U.S., Canada, and the United Kingdom. The left join (as `left_join()`) merges y into x, returning all rows of x with matching information in y based on columns they share in common (here: `ccode`).

```
# alternatively, as I tend to do it: x %>% left_join(., y)
left_join(x, y)
#> # A tibble: 15 x 3
#>    ccode  year ranking
#>    <dbl> <int> <chr>
#>  1     2  2016 low
#>  2     2  2017 low
#>  3     2  2018 low
#>  4     2  2019 low
#>  5     2  2020 low
#>  6    20  2016 medium
#>  7    20  2017 medium
#>  8    20  2018 medium
#>  9    20  2019 medium
#> 10    20  2020 medium
#> 11   200  2016 high
#> 12   200  2017 high
#> 13   200  2018 high
#> 14   200  2019 high
#> 15   200  2020 high
```

This is obviously a very simple example, but it scales well even if there is some additional complexity. For example, let's assume we added a simple five-year panel of Australia (`ccode: 900`) to the "left object" (x). However, we have no corresponding information about Australia in the "right object" (y). Here is what the left join would produce under these circumstances.

```
tibble(ccode = 900,
       year = c(2016:2020)) %>%
  bind_rows(x, .) -> x

x
#> # A tibble: 20 x 2
#>    ccode  year
#>    <dbl> <int>
#>  1     2  2016
```

```
#>  2     2  2017
#>  3     2  2018
#>  4     2  2019
#>  5     2  2020
#>  6    20  2016
#>  7    20  2017
#>  8    20  2018
#>  9    20  2019
#> 10    20  2020
#> 11   200  2016
#> 12   200  2017
#> 13   200  2018
#> 14   200  2019
#> 15   200  2020
#> 16   900  2016
#> 17   900  2017
#> 18   900  2018
#> 19   900  2019
#> 20   900  2020

left_join(x, y)
#> # A tibble: 20 x 3
#>    ccode  year ranking
#>    <dbl> <int> <chr>
#>  1     2  2016 low
#>  2     2  2017 low
#>  3     2  2018 low
#>  4     2  2019 low
#>  5     2  2020 low
#>  6    20  2016 medium
#>  7    20  2017 medium
#>  8    20  2018 medium
#>  9    20  2019 medium
#> 10    20  2020 medium
#> 11   200  2016 high
#> 12   200  2017 high
#> 13   200  2018 high
#> 14   200  2019 high
#> 15   200  2020 high
#> 16   900  2016 <NA>
#> 17   900  2017 <NA>
#> 18   900  2018 <NA>
#> 19   900  2019 <NA>
#> 20   900  2020 <NA>
```

Because have no ranking for Australia in this simple example, the left join returns NAs for Australia. The original dimensions of x under these conditions is unaffected.

What would happen if we had an observation in y that has no corresponding match in x? For example, let's assume our ranking data also included a ranking for Denmark (`ccode: 390`), though Denmark does not appear in x. Here is what would happen under these circumstances.

```
tibble(ccode = 390,
       ranking = "high") %>%
  bind_rows(y, .) -> y

y
#> # A tibble: 4 x 2
#>   ccode ranking
#>   <dbl> <chr>
#> 1     2 low
#> 2    20 medium
#> 3   200 high
#> 4   390 high

left_join(x, y)
#> # A tibble: 20 x 3
#>    ccode  year ranking
#>    <dbl> <int> <chr>
#>  1     2  2016 low
#>  2     2  2017 low
#>  3     2  2018 low
#>  4     2  2019 low
#>  5     2  2020 low
#>  6    20  2016 medium
#>  7    20  2017 medium
#>  8    20  2018 medium
#>  9    20  2019 medium
#> 10    20  2020 medium
#> 11   200  2016 high
#> 12   200  2017 high
#> 13   200  2018 high
#> 14   200  2019 high
#> 15   200  2020 high
#> 16   900  2016 <NA>
#> 17   900  2017 <NA>
#> 18   900  2018 <NA>
#> 19   900  2019 <NA>
#> 20   900  2020 <NA>
```

Notice the output of this left join is identical to the output above. Australia is in x, but not in y. Thus, the rows for Australia are returned but the absence of ranking information for Australia in y means the variable

is NA for Australia after the merge. Denmark is in y, but not x. Because the left join returns all rows in x with matching information in y, the absence of observations for Denmark in x means there is nowhere for the ranking information to go in the merge. Thus, Denmark's ranking is effectively ignored.

Why the Left Join, in Particular?

An interested user may ask what's so special about this kind of join that it appears everywhere in {peacesciencer}? One is a matter of taste. I could just as well be doing this vignette by reference to the "right join", the mirror join to "left join." The right join in {dplyr}'s right_join(x,y) returns all records from y with matching rows in x by common columns, though the equivalency would depend on reversing the order of x and y (i.e. left_join(x, y) produces the same information as right_join(y, x)). The arrangement of columns would differ in the left_join() and right_join() in this simple application even if the same underlying information is there. Ultimately, I tend to think "left-handed" when it comes to data management and instruct my students to do the same when I introduce them to data transformation in R. I like the intuition, especially in the pipe-based workflow, to start with a master data object at the top of the pipe and keep it "left" as I add information to it. It has the benefit of keeping the units of analysis (e.g. state-years in this simple setup) as the first columns the user sees as well. This is my preferred approach to data transformation and it recurs in {peacesciencer} as a result.

Beyond that matter of taste, the left join is everywhere in {peacesciencer} because the project endeavors hard to recreate the appropriate universe of cases of interest to the user and allow the user to add stuff to it as they see fit. create_stateyears() will create the entire universe of state-years from 1816 to the present for a state-year analysis. create_dyadyears() will create the entire universe of dyad-years from 1816 to the present for a dyad-year analysis. The logic here, as it is implemented in {peacesciencer}'s multiple functions, is the type of data the user wants to create has been created for them. The user does not want to expand the data any further than that, though the user may want to do something like reduce the full universe of 1816-2020 state-years to just 1946-2010. However, this is a universe discarded, not a universe that has been augmented or expanded.

With that in mind, every function's use of the left join assumes the data object it receives represents the full universe of cases of interest to the researcher. The left join is just adding information to it, based on matching information in one of its many data sets. When done carefully, the left join is a dutiful way of adding information to a data set without changing the number of rows of the original data set. The number of columns will obviously expand, but the number of rows is unaffected.

Potential Problems of the Left Join

"When done carefully" is doing some heavy-lifting in that last sentence. So, let me explain some situations where the left join will produce problems for the researcher (even if the join itself is doing what it is supposed to do from an operational standpoint).

The first is less of a problem, at least as I have implemented in {peacesciencer}, but more of a caution. In the above example, our panel consists of just the U.S., Canada, the United Kingdom, and Australia. We happen to have a ranking for Denmark, but Denmark wasn't in our panel of (effectively, exclusively) Anglophone states. Therefore, no row is created for Denmark. If it were that important that the left join create those rows for

Denmark, we should have had it in the first place. In this case, the left join is behaving as it should. We should have had Denmark in the panel before trying to match information to it.

{peacesciencer} circumvents this issue by creating universal data (e.g. all state-years, all dyad-years, all available leader-years) that the user is free to subset as they see fit. Users should run one of the "create" functions (e.g. `create_stateyears()`, `create_dyadyears()`) at the top of their script before adding information to it because the left join, as implemented everywhere in this package, is building in an assumption that the universe of cases of interest to the user is represented in the "left object" for a left outer join. Basically, do not expect the left join to create new rows in x in a situation where there is a state represented in y but not in x. It will not. This type of join assumes the universe of cases of interest to the researcher already appear in the "left object."

The second situation is a bigger problem. Sometimes, often when bouncing between information denominated in Correlates of War states and Gleditsch-Ward states, there is an unwanted duplicate observation in the data frame to be merged into the primary data of interest to the user. Let's go back to our simple example of x and y here. Everything here performs nicely, though Australia (in x) has no ranking and Denmark (in y) is not in our panel of state-years because it wasn't part of the original universe of cases of interest to us.

```
x
#> # A tibble: 20 x 2
#>    ccode  year
#>    <dbl> <int>
#>  1     2  2016
#>  2     2  2017
#>  3     2  2018
#>  4     2  2019
#>  5     2  2020
#>  6    20  2016
#>  7    20  2017
#>  8    20  2018
#>  9    20  2019
#> 10    20  2020
#> 11   200  2016
#> 12   200  2017
#> 13   200  2018
#> 14   200  2019
#> 15   200  2020
#> 16   900  2016
#> 17   900  2017
#> 18   900  2018
#> 19   900  2019
#> 20   900  2020
y
#> # A tibble: 4 x 2
#>   ccode ranking
#>   <dbl> <chr>
#> 1     2 low
```

```
#> 2    20 medium
#> 3   200 high
#> 4   390 high
```

Let's assume, however, we mistakenly entered the United Kingdom twice into y. We know these data are supposed to be simple state-level rankings. Each state is supposed to be in there just once. The United Kingdom appears in there twice.

```
tibble(ccode = 200,
       ranking = "high") %>%
  bind_rows(y, .) -> y2
```

If we were to left join y2 into x, we get an unwelcome result. The United Kingdom is duplicated for all yearly observations.

```
left_join(x, y2) %>% data.frame
#>    ccode year ranking
#> 1      2 2016     low
#> 2      2 2017     low
#> 3      2 2018     low
#> 4      2 2019     low
#> 5      2 2020     low
#> 6     20 2016  medium
#> 7     20 2017  medium
#> 8     20 2018  medium
#> 9     20 2019  medium
#> 10    20 2020  medium
#> 11   200 2016    high
#> 12   200 2016    high
#> 13   200 2017    high
#> 14   200 2017    high
#> 15   200 2018    high
#> 16   200 2018    high
#> 17   200 2019    high
#> 18   200 2019    high
#> 19   200 2020    high
#> 20   200 2020    high
#> 21   900 2016    <NA>
#> 22   900 2017    <NA>
#> 23   900 2018    <NA>
#> 24   900 2019    <NA>
#> 25   900 2020    <NA>
```

It doesn't matter that the duplicate ranking in y2 for the UK was the same. It would be messier, sure, if the ranking was different for the duplicate observation, but it matters more here that it was duplicated. In a panel like this, a user who is not careful will have the effect of overweighting those observations that duplicate. In a simple example like this, subsetting to just complete cases (i.e. Australia has no ranking), the UK is 50% of all

observations despite the fact it should just be a third of observations. That's not ideal for a researcher.

`{peacesciencer}` goes above and beyond to make sure this doesn't happen in the data it creates. Functions are aggressively tested to make sure nothing duplicates, and various parlor tricks (prominently group-by slices) are used internally to cull those duplicate observations. The release of a function that makes prominent use of the left join is done with the assurance it doesn't create a duplicate. No matter, this is the biggest peril of the left join for a researcher who may want to duplicate what `{peacesciencer}` does on their own. Always inspect the data you merge, and the output.

Semi-Join

The "semi-join" (`semi_join()` in `{dplyr}`) returns all rows from the left object (`x`) that have matching values in the right object (`y`). It is a type of "filtering join", which affects the observations and not the variables. It appears just twice in `{peacesciencer}`, serving as a final join in `create_leaderdays()` and `create_leaderyears()`. In both cases, it serves as a means of standardizing leader data (denominated in the Gleditsch-Ward system, if not necessarily Gleditsch-Ward system dates) to the Correlates of War or Gleditsch-Ward system.

Here is a basic example of what a semi-join is doing in this context, with an illustration of the kind of difficulties that manifest in standardizing Archigos' leader data to the Correlates of War state system. Assume this simple state system that has just two states—"A" and "B"—over a two-week period at the start of 1975 (Jan. 1, 1975 to Jan. 14, 1975). In this simple system, "A" is a state for the full two week period (Jan. 1-Jan.14) whereas "B" is a state for just the first seven days (Jan. 1-Jan. 7) because, let's say, A occupied B and ended its statehood. We also happened to have some leader data for these two states. Over this two week period, our leader data suggests A had just one continuous leader—"Archie"—whereas B had three. "Brian" was the leader of B before he retired from office and was replaced by "Cornelius." However, he was deposed when A invaded B and was replaced by a puppet head of state, "Pete." Our data look like this.

```
tibble(code = c("A", "B"),
       stdate = make_date(1975, 01, 01),
       enddate = c(make_date(1975, 01, 14),
                   make_date(1975, 01, 07))) -> state_system

state_system
#> # A tibble: 2 x 3
#>   code  stdate     enddate
#>   <chr> <date>     <date>
#> 1 A     1975-01-01 1975-01-14
#> 2 B     1975-01-01 1975-01-07

tibble(code = c("A", "B", "B", "B"),
       leader = c("Archie", "Brian", "Cornelius", "Pete"),
       stdate = c(make_date(1975, 01, 01), make_date(1975, 01, 01),
                  make_date(1975, 01, 04), make_date(1975, 01, 08)),
       enddate = c(make_date(1975, 01, 14), make_date(1975, 01, 04),
                   make_date(1975, 01, 08), make_date(1975, 01, 14))) -> leaders
```

```
leaders
#> # A tibble: 4 x 4
#>   code  leader    stdate     enddate
#>   <chr> <chr>     <date>     <date>
#> 1 A     Archie    1975-01-01 1975-01-14
#> 2 B     Brian     1975-01-01 1975-01-04
#> 3 B     Cornelius 1975-01-04 1975-01-08
#> 4 B     Pete      1975-01-08 1975-01-14
```

We can use some basic `rowwise()` transformation to recast these data as daily, resulting in state-day data and leader-day data.

```
state_system %>%
  rowwise() %>%
  mutate(date = list(seq(stdate, enddate, by = '1 day'))) %>%
  unnest(date) %>%
  select(code, date) -> state_days

state_days %>% data.frame
#>    code       date
#> 1     A 1975-01-01
#> 2     A 1975-01-02
#> 3     A 1975-01-03
#> 4     A 1975-01-04
#> 5     A 1975-01-05
#> 6     A 1975-01-06
#> 7     A 1975-01-07
#> 8     A 1975-01-08
#> 9     A 1975-01-09
#> 10    A 1975-01-10
#> 11    A 1975-01-11
#> 12    A 1975-01-12
#> 13    A 1975-01-13
#> 14    A 1975-01-14
#> 15    B 1975-01-01
#> 16    B 1975-01-02
#> 17    B 1975-01-03
#> 18    B 1975-01-04
#> 19    B 1975-01-05
#> 20    B 1975-01-06
#> 21    B 1975-01-07

leaders %>%
  rowwise() %>%
  mutate(date = list(seq(stdate, enddate, by = '1 day'))) %>%
```

```
  unnest(date) %>%
  select(code, leader, date) -> leader_days

leader_days %>% data.frame
#>    code    leader       date
#> 1     A    Archie 1975-01-01
#> 2     A    Archie 1975-01-02
#> 3     A    Archie 1975-01-03
#> 4     A    Archie 1975-01-04
#> 5     A    Archie 1975-01-05
#> 6     A    Archie 1975-01-06
#> 7     A    Archie 1975-01-07
#> 8     A    Archie 1975-01-08
#> 9     A    Archie 1975-01-09
#> 10    A    Archie 1975-01-10
#> 11    A    Archie 1975-01-11
#> 12    A    Archie 1975-01-12
#> 13    A    Archie 1975-01-13
#> 14    A    Archie 1975-01-14
#> 15    B     Brian 1975-01-01
#> 16    B     Brian 1975-01-02
#> 17    B     Brian 1975-01-03
#> 18    B     Brian 1975-01-04
#> 19    B Cornelius 1975-01-04
#> 20    B Cornelius 1975-01-05
#> 21    B Cornelius 1975-01-06
#> 22    B Cornelius 1975-01-07
#> 23    B Cornelius 1975-01-08
#> 24    B      Pete 1975-01-08
#> 25    B      Pete 1975-01-09
#> 26    B      Pete 1975-01-10
#> 27    B      Pete 1975-01-11
#> 28    B      Pete 1975-01-12
#> 29    B      Pete 1975-01-13
#> 30    B      Pete 1975-01-14
```

If we wanted to standardize these leader-day data to the state system data, we would semi-join the leader-day data (the left object) with the state-day object (the right object), returning just the leader-day data with valid days in the state system data.

```
leader_days %>%
  semi_join(., state_days) %>%
  data.frame
#>    code    leader       date
#> 1     A    Archie 1975-01-01
#> 2     A    Archie 1975-01-02
```

```
#> 3     A    Archie 1975-01-03
#> 4     A    Archie 1975-01-04
#> 5     A    Archie 1975-01-05
#> 6     A    Archie 1975-01-06
#> 7     A    Archie 1975-01-07
#> 8     A    Archie 1975-01-08
#> 9     A    Archie 1975-01-09
#> 10    A    Archie 1975-01-10
#> 11    A    Archie 1975-01-11
#> 12    A    Archie 1975-01-12
#> 13    A    Archie 1975-01-13
#> 14    A    Archie 1975-01-14
#> 15    B     Brian 1975-01-01
#> 16    B     Brian 1975-01-02
#> 17    B     Brian 1975-01-03
#> 18    B     Brian 1975-01-04
#> 19    B Cornelius 1975-01-04
#> 20    B Cornelius 1975-01-05
#> 21    B Cornelius 1975-01-06
#> 22    B Cornelius 1975-01-07
```

Notice that Pete drops from these data because, in this simple example, Pete was a puppet head of state installed by Archie when state A invaded and occupied B. The semi-join here is simply standardizing the leader data to the state system data, which is effectively what's happening with the semi-joins in `create_leaderdays()` (and its aggregation function: `create_leaderyears()`).

Anti-Join

The anti-join is another type of filtering join, returning all rows from the left object (`x`) with*out* a match in the right object (`y`). This type of join appears just once in `{peacesciencer}`. Prominently, `{peacesciencer}` prepares and presents two data sets in this package—`false_cow_dyads` and `false_gw_dyads`—that represent directed dyad-years in the Correlates of War and Gleditsch-Ward systems that were active in the same year, but never at the same time on the same year.

Here are those dyads for context.

```
false_cow_dyads
#> # A tibble: 60 x 4
#>    ccode1 ccode2  year in_ps
#>     <dbl>  <dbl> <int> <dbl>
#>  1    115    817  1975     1
#>  2    210    255  1945     1
#>  3    211    255  1945     1
#>  4    223    678  1990     1
#>  5    223    680  1990     1
#>  6    255    210  1945     1
```

```
#>  7    255    211  1945     1
#>  8    255    260  1990     1
#>  9    255    265  1990     1
#> 10    255    290  1945     1
#> # ... with 50 more rows

false_gw_dyads
#> # A tibble: 38 x 4
#>    gwcode1 gwcode2  year in_ps
#>      <dbl>   <dbl> <int> <dbl>
#>  1      99     100  1830     1
#>  2      99     211  1830     1
#>  3     100      99  1830     1
#>  4     100     615  1830     1
#>  5     115     817  1975     1
#>  6     211      99  1830     1
#>  7     211     615  1830     1
#>  8     255     850  1945     1
#>  9     300     305  1918     1
#> 10     300     345  1918     1
#> # ... with 28 more rows
```

These were created by two scripts that, for each year in the respective state system data, creates every possible *daily* dyadic pairing and truncates the dyads to just those that had at least one day of overlap. This is a computationally demanding procedure compared to what {peacesciencer} does (which creates every possible dyadic pair in a given year, given the state system data supplied to it). However, it creates the possibility of same false dyads in a given year that showed no overlap.

Consider the case of Suriname (115) and the Republic of Vietnam (817) in 1975 as illustrative here.

```
check_both <- function(x) {

  gw_states %>%
    mutate(data = "G-W") %>%
    filter(gwcode %in% x) -> gwrows

  cow_states %>%
    mutate(startdate = ymd(paste0(styear,"/",stmonth, "/", stday)),
           enddate = ymd(paste0(endyear,"/",endmonth,"/",endday))) %>%
    select(stateabb:statenme, startdate, enddate) %>%
    mutate(data = "CoW") %>%
    rename(statename = statenme) %>%
    filter(ccode %in% x) -> cowrows

  dat <- bind_rows(gwrows, cowrows) %>%
    select(gwcode, ccode, stateabb, everything())
```

```
  return(dat)
}

check_both(c(115, 817))
#> # A tibble: 4 x 7
#>   gwcode ccode stateabb statename            startdate  enddate    data
#>    <dbl> <dbl> <chr>    <chr>                <date>     <date>     <chr>
#> 1    115    NA SUR      Surinam              1975-11-25 2017-12-31 G-W
#> 2    817    NA RVN      Vietnam, Republic of 1954-05-01 1975-04-30 G-W
#> 3     NA   115 SUR      Suriname             1975-11-25 2016-12-31 CoW
#> 4     NA   817 RVN      Republic of Vietnam  1954-06-04 1975-04-30 CoW
```

Notice both Suriname and Republic of Vietnam were both active in 1975. Suriname appears on Nov. 25, 1975 whereas the Republic of Vietnam exits on April 30, 1975. However, there is no daily overlap between the two because they did not exist at any point on the same day in 1975. These are false dyads. `anti_join()` is used in the `create_dyadyears()` function to remove these observations before presenting them to the user.

Here is a simple example of what an anti-join is doing with these examples in mind.

```
valid_dyads <- tibble(ccode1 = c(2, 20, 200),
                      ccode2 = c(20, 200, 900),
                      year = c(2016, 2017, 2018))

valid_dyads %>%
  bind_rows(., false_cow_dyads %>% select(ccode1:year)) -> valid_and_invalid

valid_and_invalid
#> # A tibble: 63 x 3
#>    ccode1 ccode2  year
#>     <dbl>  <dbl> <dbl>
#>  1      2     20  2016
#>  2     20    200  2017
#>  3    200    900  2018
#>  4    115    817  1975
#>  5    210    255  1945
#>  6    211    255  1945
#>  7    223    678  1990
#>  8    223    680  1990
#>  9    255    210  1945
#> 10    255    211  1945
#> # ... with 53 more rows

valid_and_invalid %>%
  # remove those invalid dyads-years
  anti_join(., false_cow_dyads)
#> # A tibble: 3 x 3
```

```
#>   ccode1 ccode2  year
#>    <dbl>  <dbl> <dbl>
#> 1      2     20  2016
#> 2     20    200  2017
#> 3    200    900  2018
```

Vignette: Various Parlor Tricks in {peacesciencer}

```
library(tidyverse)
library(peacesciencer)
```

This is a running list of various parlor tricks that you can do with the data and functions in {peacesciencer}. Space and time considerations, along with some rigidity imposed by CRAN guidelines, preclude me from including these as outright functions or belaboring them in greater detail in the user's guide. Again, {peacesciencer} can do a lot, but it can't do everything. Yet, some of its functionality may not also be obvious from the user's guide or documentation files because they're not necessarily core functions. Thus "parlor trick" is an appropriate descriptor here.

Create a "New State" Variable

The user's guide includes a partial replication of a state-year civil conflict analysis analogous to Fearon and Laitin (2003) and Gibler and Miller (2014). Both of those analyses include a "new state" variable, arguing that states within the first two years of their existence are more likely to experience a civil war onset. The partial replication does not include this. This is because the easiest way to create this variable is through a group_by() mutate based on the row number of the group, but group_by() has the unfortunate side effect of erasing any other attributes in the data (i.e. the ps_system and ps_type attributes). This would break the {peacesciencer} pipe. If you want this variable, I recommend creating and merging this variable after creating the bulk of the data.

Here's how you'd do it.

```
# Hypothetical main data
create_stateyears(system = 'gw') %>%
  filter(between(year, 1946, 2019)) %>%
  add_ucdp_acd(type = "intrastate") %>%
  add_peace_years() -> Data

# Add in new state variable after the fact
create_stateyears(system = 'gw') %>%
  group_by(gwcode) %>%
  mutate(newstate = ifelse(row_number() <= 2, 1, 0)) %>%
  left_join(Data, .) %>%
  select(gwcode:ucdponset, newstate, everything()) -> Data
```

```
# Proof of concept: Here's India
Data %>% filter(gwcode == 750)
#> # A tibble: 73 x 9
#>    gwcode statename  year ucdpongoing ucdponset newstate maxintensity
#>     <dbl> <chr>     <dbl>       <dbl>     <dbl>    <dbl>        <dbl>
#>  1    750 India      1947           0         0        1           NA
#>  2    750 India      1948           1         1        1            2
#>  3    750 India      1949           1         0        0            2
#>  4    750 India      1950           1         0        0            2
#>  5    750 India      1951           1         0        0            2
#>  6    750 India      1952           0         0        0           NA
#>  7    750 India      1953           0         0        0           NA
#>  8    750 India      1954           0         0        0           NA
#>  9    750 India      1955           0         0        0           NA
#> 10    750 India      1956           1         1        0            1
#> # ... with 63 more rows, and 2 more variables: conflict_ids <chr>,
#> #   ucdpspell <dbl>

# And here's Belize
Data %>% filter(gwcode == 80)
#> # A tibble: 39 x 9
#>    gwcode statename  year ucdpongoing ucdponset newstate maxintensity
#>     <dbl> <chr>     <dbl>       <dbl>     <dbl>    <dbl>        <dbl>
#>  1     80 Belize     1981           0         0        1           NA
#>  2     80 Belize     1982           0         0        1           NA
#>  3     80 Belize     1983           0         0        0           NA
#>  4     80 Belize     1984           0         0        0           NA
#>  5     80 Belize     1985           0         0        0           NA
#>  6     80 Belize     1986           0         0        0           NA
#>  7     80 Belize     1987           0         0        0           NA
#>  8     80 Belize     1988           0         0        0           NA
#>  9     80 Belize     1989           0         0        0           NA
#> 10     80 Belize     1990           0         0        0           NA
#> # ... with 29 more rows, and 2 more variables: conflict_ids <chr>,
#> #   ucdpspell <dbl>
```

Code Capabilities/Development/Militarization as Bremer (1992) Did

The user's guide includes a replication of Bremer's (1992) "dangerous dyads" design, albeit one that leverages newer/better data sources that were unavailable to Bremer at the time. For convenience's sake, the replication used other approaches to estimating Bremer's variables, including the "weak-link" mechanisms that Dixon (1994) introduced in his seminal work on democratic conflict resolution. If the user wanted to recreate some of the covariates as Bremer (1992) did it, here would be how to do it.

The covariates in question concern information grabbed from the Correlates of War national material capabil-

ities data set.[2] For example, the user guide recreates the "relative power" variable as a proportion of the lower composite index of national capabilities (CINC) variable over the higher one. Bremer opts for a different approach, defining a "relative power" variable as a three-part ordinal category where the more powerful side has a CINC score that is 1) 10 times higher than the less powerful side, 2) three times higher than the other side, or 3) less than three times higher than the other side. Here is the exact passage on p. 322.

> Based on these CINC scores, I computed the larger-to-smaller capability ratios for all dyad-years and classified them into three groups. If the capability ratio was less than or equal to three, then the dyad was considered to constitute a case of small power difference. If the ratio was larger than 10, then the power difference was coded as large, whereas a ratio between 3 and 10 was coded as a medium power difference. If either of the CINC scores was missing (or equal to zero) for a ratio calculation, then the power difference score for that dyad was coded as missing also.

This is an easy `case_when()` function, but it also would've consumed space and words in a user's guide with a hard word count of 6,000. There's added difficulty in making sure to identify which side in a non-directed dyad-year is more powerful.

```
cow_ddy %>% # built-in data set for convenience
  filter(ccode2 > ccode1) %>% # make it non-directed
  # add CINC scores
  add_nmc() %>%
  # select just what we want
  select(ccode1:year, cinc1, cinc2) -> Bremer

Bremer %>%
  # create a three-item ordinal relative power category with values 2, 1, and 0
  mutate(relpow = case_when(
    (cinc1 > cinc2) & (cinc1 > 10*cinc2) ~ 2,
    (cinc1 > cinc2) & ((cinc1 > 3*cinc2) & (cinc1 < 10*cinc2)) ~ 1,
    (cinc1 > cinc2) & (cinc1 <= 3*cinc2) ~ 0,
    # copy-paste, re-arrange
    (cinc2 > cinc1) & (cinc2 > 10*cinc1) ~ 2,
    (cinc2 > cinc1) & ((cinc2 > 3*cinc1) & (cinc2 < 10*cinc1))~ 1,
    (cinc2 > cinc1) & (cinc2 <= 3*cinc1) ~ 0,
    TRUE ~ NA_real_
  )) -> relpow_example

# Let's inspect the output.
relpow_example %>% na.omit %>%
  mutate(whichside = ifelse(cinc1 > cinc2, "ccode1 > ccode2", "ccode2 >= ccode1")) %>%
  group_split(whichside, relpow)
#> <list_of<
#>   tbl_df<
```

[2] Bremer has a different way of coding democracy (i.e. using a value of 5 or greater on the democracy scale in Polity), but this is so far removed from current practice that it's inadvisable to replicate. If you want to use the Polity data (using `add_democracy()` in this package), use the `polity2` variable that adds the autocracy and democracy indices together. Therein, use the weak-link specification *and* the distance between the more democratic and less democratic state.

```
#>     ccode1   : double
#>     ccode2   : double
#>     year     : double
#>     cinc1    : double
#>     cinc2    : double
#>     relpow   : double
#>     whichside: character
#>   >
#> >[6]>
#> [[1]]
#> # A tibble: 122,887 x 7
#>    ccode1 ccode2  year cinc1 cinc2 relpow whichside
#>     <dbl>  <dbl> <dbl> <dbl> <dbl>  <dbl> <chr>
#>  1      2    200  1892 0.173 0.173      0 ccode1 > ccode2
#>  2      2    200  1897 0.169 0.166      0 ccode1 > ccode2
#>  3      2    200  1898 0.197 0.157      0 ccode1 > ccode2
#>  4      2    200  1899 0.185 0.169      0 ccode1 > ccode2
#>  5      2    200  1900 0.188 0.178      0 ccode1 > ccode2
#>  6      2    200  1901 0.203 0.174      0 ccode1 > ccode2
#>  7      2    200  1902 0.208 0.161      0 ccode1 > ccode2
#>  8      2    200  1903 0.210 0.143      0 ccode1 > ccode2
#>  9      2    200  1904 0.205 0.135      0 ccode1 > ccode2
#> 10      2    200  1905 0.214 0.121      0 ccode1 > ccode2
#> # ... with 122,877 more rows
#>
#> [[2]]
#> # A tibble: 106,282 x 7
#>    ccode1 ccode2  year  cinc1   cinc2 relpow whichside
#>     <dbl>  <dbl> <dbl>  <dbl>   <dbl>  <dbl> <chr>
#>  1      2     70  1831 0.0420 0.00945      1 ccode1 > ccode2
#>  2      2     70  1832 0.0445 0.00963      1 ccode1 > ccode2
#>  3      2     70  1833 0.0481 0.00958      1 ccode1 > ccode2
#>  4      2     70  1834 0.0478 0.00971      1 ccode1 > ccode2
#>  5      2     70  1835 0.0485 0.00980      1 ccode1 > ccode2
#>  6      2     70  1836 0.0510 0.00941      1 ccode1 > ccode2
#>  7      2     70  1837 0.0535 0.00975      1 ccode1 > ccode2
#>  8      2     70  1838 0.0533 0.00966      1 ccode1 > ccode2
#>  9      2     70  1839 0.0508 0.00948      1 ccode1 > ccode2
#> 10      2     70  1840 0.0495 0.00898      1 ccode1 > ccode2
#> # ... with 106,272 more rows
#>
#> [[3]]
#> # A tibble: 181,735 x 7
#>    ccode1 ccode2  year cinc1   cinc2 relpow whichside
#>     <dbl>  <dbl> <dbl> <dbl>   <dbl>  <dbl> <chr>
```

```
#>  1      2     20  1920 0.290 0.0101       2 ccode1 > ccode2
#>  2      2     20  1921 0.253 0.0105       2 ccode1 > ccode2
#>  3      2     20  1922 0.256 0.00841      2 ccode1 > ccode2
#>  4      2     20  1923 0.272 0.00986      2 ccode1 > ccode2
#>  5      2     20  1924 0.254 0.00889      2 ccode1 > ccode2
#>  6      2     20  1925 0.254 0.00870      2 ccode1 > ccode2
#>  7      2     20  1926 0.263 0.00924      2 ccode1 > ccode2
#>  8      2     20  1927 0.239 0.00937      2 ccode1 > ccode2
#>  9      2     20  1928 0.240 0.00970      2 ccode1 > ccode2
#> 10      2     20  1929 0.240 0.00980      2 ccode1 > ccode2
#> # ... with 181,725 more rows
#>
#> [[4]]
#> # A tibble: 130,447 x 7
#>    ccode1 ccode2  year  cinc1 cinc2 relpow whichside
#>     <dbl>  <dbl> <dbl>  <dbl> <dbl>  <dbl> <chr>
#>  1      2    200  1861 0.144  0.258      0 ccode2 >= ccode1
#>  2      2    200  1862 0.176  0.251      0 ccode2 >= ccode1
#>  3      2    200  1863 0.179  0.251      0 ccode2 >= ccode1
#>  4      2    200  1864 0.193  0.243      0 ccode2 >= ccode1
#>  5      2    200  1865 0.135  0.256      0 ccode2 >= ccode1
#>  6      2    200  1866 0.0982 0.248      0 ccode2 >= ccode1
#>  7      2    200  1867 0.114  0.253      0 ccode2 >= ccode1
#>  8      2    200  1868 0.107  0.253      0 ccode2 >= ccode1
#>  9      2    200  1869 0.108  0.246      0 ccode2 >= ccode1
#> 10      2    200  1870 0.0990 0.242      0 ccode2 >= ccode1
#> # ... with 130,437 more rows
#>
#> [[5]]
#> # A tibble: 122,918 x 7
#>    ccode1 ccode2  year  cinc1 cinc2 relpow whichside
#>     <dbl>  <dbl> <dbl>  <dbl> <dbl>  <dbl> <chr>
#>  1      2    200  1816 0.0397 0.337      1 ccode2 >= ccode1
#>  2      2    200  1817 0.0358 0.328      1 ccode2 >= ccode1
#>  3      2    200  1818 0.0361 0.329      1 ccode2 >= ccode1
#>  4      2    200  1819 0.0371 0.317      1 ccode2 >= ccode1
#>  5      2    200  1820 0.0371 0.317      1 ccode2 >= ccode1
#>  6      2    200  1821 0.0342 0.317      1 ccode2 >= ccode1
#>  7      2    200  1822 0.0329 0.311      1 ccode2 >= ccode1
#>  8      2    200  1823 0.0331 0.318      1 ccode2 >= ccode1
#>  9      2    200  1824 0.0330 0.330      1 ccode2 >= ccode1
#> 10      2    200  1825 0.0342 0.331      1 ccode2 >= ccode1
#> # ... with 122,908 more rows
#>
#> [[6]]
```

```
#> # A tibble: 216,172 x 7
#>    ccode1 ccode2  year   cinc1  cinc2 relpow whichside
#>     <dbl>  <dbl> <dbl>   <dbl>  <dbl>  <dbl> <chr>
#>  1     20    200  1920 0.0101  0.128      2 ccode2 >= ccode1
#>  2     20    200  1922 0.00841 0.0945     2 ccode2 >= ccode1
#>  3     20    200  1923 0.00986 0.0990     2 ccode2 >= ccode1
#>  4     20    200  1924 0.00889 0.107      2 ccode2 >= ccode1
#>  5     20    200  1925 0.00870 0.0956     2 ccode2 >= ccode1
#>  6     20    200  1939 0.00909 0.0997     2 ccode2 >= ccode1
#>  7     20    255  1934 0.00891 0.0891     2 ccode2 >= ccode1
#>  8     20    255  1935 0.00874 0.103      2 ccode2 >= ccode1
#>  9     20    255  1936 0.00865 0.115      2 ccode2 >= ccode1
#> 10     20    255  1937 0.00893 0.118      2 ccode2 >= ccode1
#> # ... with 216,162 more rows
```

Next, the user's guide codes Bremer's (1992) development/"advanced economies" measure using the weak-link of the lower GDP per capita in the dyad using the simulations from Anders et al. (2020). In my defense, this is exactly the kind of data Bremer wishes he had available to him. He says so himself on footnote 26 on page 324.

> Under the most optimistic assumptions about data availability, I would estimate that the number of dyad-years for which the relevant data [GNP or GDP per capita] could be assembled would be less than 20% of the total dyad-years under consideration. A more realistic estimate might be as low as 10%. Clearly, our ability to test a generalization when 80% to 90% of the needed data are missing is very limited, and especially so in this case, because the missing data would be concentrated heavily in the pre-World War II era and less advanced states.

Given this limitation, Bremer uses this approach to coding the development/"advanced economies" measure.

> A more economically advanced state should be characterized by possessing a share of system-wide economic capability that is greater than its share of system-wide demographic capability. Hence, in years when this was found to be true, I classified a state as more advanced; otherwise, less advanced. The next step involved examining each pair of states in each year and assigning it to one of three groups: both more advanced (7,160 dyad-years), one more advanced (61,823 dyad-years), and both less advanced (128,939 dyad-years).

Replicating this approach is going to require group-by summaries of the raw national material capabilities data, which is outside of `{peacesciencer}`'s core functionality. Bremer's wording here is a little vague; he doesn't explain what variable, or variables, comprise "economic capability" and "demographic capability." Let's assume that "demographic capability" is just the total population variable whereas the "economic capability" variable*s* include iron and steel production and primary energy consumption. The variable would look something like this.

```
cow_nmc %>%
  group_by(year) %>%
  # calculate year proportions
  mutate(prop_tpop = tpop/sum(tpop, na.rm=T),
         prop_irst = irst/sum(irst, na.rm=T),
         prop_pec = pec/sum(pec, na.rm=T)) %>%
```

```
  ungroup() %>%
  # standardize an "economic capability" measure
  # then make an advanced dummy
  mutate(econcap = (prop_irst + prop_pec)/2,
         advanced = ifelse(econcap > prop_tpop, 1, 0)) %>%
  select(ccode, year, prop_tpop:ncol(.)) -> Advanced

Advanced
#> # A tibble: 15,171 x 7
#>    ccode  year prop_tpop prop_irst prop_pec econcap advanced
#>    <dbl> <dbl>     <dbl>     <dbl>    <dbl>   <dbl>    <dbl>
#>  1     2  1816    0.0398    0.0954  0.00966  0.0525        1
#>  2     2  1817    0.0404    0.0938  0.0103   0.0520        1
#>  3     2  1818    0.0411    0.102   0.0110   0.0564        1
#>  4     2  1819    0.0416    0.101   0.0104   0.0555        1
#>  5     2  1820    0.0422    0.113   0.0105   0.0617        1
#>  6     2  1821    0.0430    0.0927  0.0108   0.0518        1
#>  7     2  1822    0.0431    0.0950  0.0109   0.0530        1
#>  8     2  1823    0.0439    0.0933  0.0111   0.0522        1
#>  9     2  1824    0.0447    0.0861  0.0122   0.0491        1
#> 10     2  1825    0.0453    0.0891  0.0129   0.0510        1
#> # ... with 15,161 more rows
```

Now, let's merge this into the `Bremer` data frame we created. I'll make this an ordinal variable as well with the same 2, 1, 0 ordering scheme.

```
Bremer %>%
  left_join(., Advanced %>% select(ccode, year, advanced) %>% rename(ccode1 = ccode, advanced1
  left_join(., Advanced %>% select(ccode, year, advanced) %>% rename(ccode2 = ccode, advanced2
  mutate(advancedcat = case_when(
    advanced1 == 1 & advanced2 == 1 ~ 2,
    (advanced1 == 1 & advanced2 == 0) | (advanced1 == 0 & advanced2 == 1) ~ 1,
    advanced1 == 0 & advanced2 == 0 ~ 0
  )) -> Bremer

# Let's inspect the output
Bremer %>% na.omit %>%
  group_split(advancedcat)
#> <list_of<
#>   tbl_df<
#>     ccode1     : double
#>     ccode2     : double
#>     year       : double
#>     cinc1      : double
#>     cinc2      : double
#>     advanced1  : double
```

```
#>     advanced2  : double
#>     advancedcat: double
#>   >
#> >[3]>
#> [[1]]
#> # A tibble: 492,967 x 8
#>    ccode1 ccode2  year     cinc1   cinc2 advanced1 advanced2 advancedcat
#>     <dbl>  <dbl> <dbl>     <dbl>   <dbl>     <dbl>     <dbl>        <dbl>
#>  1     31     40  1986 0.0000349 0.00326         0         0            0
#>  2     31     40  1987 0.0000349 0.00328         0         0            0
#>  3     31     40  1988 0.000046  0.00334         0         0            0
#>  4     31     40  1989 0.0000584 0.00335         0         0            0
#>  5     31     40  1990 0.0000511 0.00324         0         0            0
#>  6     31     40  1991 0.0000432 0.00330         0         0            0
#>  7     31     40  1992 0.0000444 0.00278         0         0            0
#>  8     31     40  1993 0.0000479 0.00272         0         0            0
#>  9     31     40  1994 0.0000365 0.00204         0         0            0
#> 10     31     40  1995 0.0000355 0.00161         0         0            0
#> # ... with 492,957 more rows
#>
#> [[2]]
#> # A tibble: 315,588 x 8
#>    ccode1 ccode2  year cinc1     cinc2 advanced1 advanced2 advancedcat
#>     <dbl>  <dbl> <dbl> <dbl>     <dbl>     <dbl>     <dbl>        <dbl>
#>  1      2     31  1986 0.135 0.0000349         1         0            1
#>  2      2     31  1987 0.134 0.0000349         1         0            1
#>  3      2     31  1988 0.134 0.000046          1         0            1
#>  4      2     31  1989 0.148 0.0000584         1         0            1
#>  5      2     31  1990 0.141 0.0000511         1         0            1
#>  6      2     31  1991 0.137 0.0000432         1         0            1
#>  7      2     31  1992 0.148 0.0000444         1         0            1
#>  8      2     31  1993 0.154 0.0000479         1         0            1
#>  9      2     31  1994 0.146 0.0000365         1         0            1
#> 10      2     31  1995 0.140 0.0000355         1         0            1
#> # ... with 315,578 more rows
#>
#> [[3]]
#> # A tibble: 50,500 x 8
#>    ccode1 ccode2  year cinc1   cinc2 advanced1 advanced2 advancedcat
#>     <dbl>  <dbl> <dbl> <dbl>   <dbl>     <dbl>     <dbl>        <dbl>
#>  1      2     20  1920 0.290 0.0101          1         1            2
#>  2      2     20  1921 0.253 0.0105          1         1            2
#>  3      2     20  1922 0.256 0.00841         1         1            2
#>  4      2     20  1923 0.272 0.00986         1         1            2
#>  5      2     20  1924 0.254 0.00889         1         1            2
```

```
#>  6      2     20  1925 0.254 0.00870         1         1         2
#>  7      2     20  1926 0.263 0.00924         1         1         2
#>  8      2     20  1927 0.239 0.00937         1         1         2
#>  9      2     20  1928 0.240 0.00970         1         1         2
#> 10      2     20  1929 0.240 0.00980         1         1         2
#> # ... with 50,490 more rows
```

Finally, the user's guide creates a militarization measure that is a weak-link that uses the data on military personnel and total population. Bremer opts for an approach similar to the development indicator he uses.

> Instead, I relied on the material capabilities data set discussed above, and classified a state as more militarized if its share of system-wide military capabilities was greater than its share of system-wide demographic capabilities. I classified it less militarized if this was not true. The classification of each dyad-year was then based on whether both, one, or neither of the two states making up the dyad were more militarized in that year.

It reads like this is what he's doing, while again reiterating that I'm assuming he's using just the total population variable to measure "demographic capability."

```
cow_nmc %>%
  group_by(year) %>%
  # calculate year proportions
  mutate(prop_tpop = tpop/sum(tpop, na.rm=T),
         prop_milex = milex/sum(milex, na.rm=T),
         prop_milper = milper/sum(milper, na.rm=T)) %>%
  ungroup() %>%
  # standardize a "military capability" measure
  # then make an advanced dummy
  mutate(militcap = (prop_milper + prop_milex)/2,
         militarized = ifelse(militcap > prop_tpop, 1, 0)) %>%
  select(ccode, year, prop_tpop:ncol(.)) -> Militarized

Militarized
#> # A tibble: 15,171 x 7
#>    ccode  year prop_tpop prop_milex prop_milper militcap militarized
#>    <dbl> <dbl>     <dbl>      <dbl>       <dbl>    <dbl>       <dbl>
#>  1     2  1816    0.0398     0.0682     0.00859   0.0384           0
#>  2     2  1817    0.0404     0.0451     0.00827   0.0267           0
#>  3     2  1818    0.0411     0.0370     0.00832   0.0227           0
#>  4     2  1819    0.0416     0.0449     0.00709   0.0260           0
#>  5     2  1820    0.0422     0.0310     0.00733   0.0192           0
#>  6     2  1821    0.0430     0.0345     0.00486   0.0197           0
#>  7     2  1822    0.0431     0.0249     0.00417   0.0146           0
#>  8     2  1823    0.0439     0.0249     0.00534   0.0151           0
#>  9     2  1824    0.0447     0.0295     0.00474   0.0171           0
#> 10     2  1825    0.0453     0.0321     0.00511   0.0186           0
#> # ... with 15,161 more rows
```

Let's merge this into the Bremer data we created and inspect the output.

```
Bremer %>%
  left_join(., Militarized %>% select(ccode, year, militarized) %>% rename(ccode1 = ccode, mili
  left_join(., Militarized %>% select(ccode, year, militarized) %>% rename(ccode2 = ccode, mili
  mutate(militcat = case_when(
    militarized1 == 1 & militarized2 == 1 ~ 2,
    (militarized1 == 1 & militarized2 == 0) | (advanced1 == 0 & militarized2 == 1) ~ 1,
    militarized1 == 0 & militarized2 == 0 ~ 0
  )) -> Bremer

Bremer %>% select(ccode1:year, militarized1:ncol(.)) %>%
  na.omit %>%
  group_split(militcat)
#> <list_of<
#>   tbl_df<
#>     ccode1      : double
#>     ccode2      : double
#>     year        : double
#>     militarized1: double
#>     militarized2: double
#>     militcat    : double
#>   >
#> >[3]>
#> [[1]]
#> # A tibble: 285,667 x 6
#>    ccode1 ccode2  year militarized1 militarized2 militcat
#>     <dbl>  <dbl> <dbl>        <dbl>        <dbl>    <dbl>
#>  1      2     20  1923            0            0        0
#>  2      2     20  1925            0            0        0
#>  3      2     20  1926            0            0        0
#>  4      2     20  1927            0            0        0
#>  5      2     20  1928            0            0        0
#>  6      2     20  1929            0            0        0
#>  7      2     20  1930            0            0        0
#>  8      2     20  1931            0            0        0
#>  9      2     20  1932            0            0        0
#> 10      2     20  1933            0            0        0
#> # ... with 285,657 more rows
#>
#> [[2]]
#> # A tibble: 316,008 x 6
#>    ccode1 ccode2  year militarized1 militarized2 militcat
#>     <dbl>  <dbl> <dbl>        <dbl>        <dbl>    <dbl>
#>  1      2     20  1920            1            0        1
#>  2      2     20  1921            1            0        1
```

```
#>  3      2     20  1922            1            0        1
#>  4      2     20  1924            1            0        1
#>  5      2     20  1947            1            0        1
#>  6      2     20  1948            1            0        1
#>  7      2     20  1949            1            0        1
#>  8      2     20  1950            1            0        1
#>  9      2     20  1971            1            0        1
#> 10      2     20  1973            1            0        1
#> # ... with 315,998 more rows
#>
#> [[3]]
#> # A tibble: 104,554 x 6
#>    ccode1 ccode2  year militarized1 militarized2 militcat
#>     <dbl>  <dbl> <dbl>        <dbl>        <dbl>    <dbl>
#>  1      2     20  1942            1            1        2
#>  2      2     20  1943            1            1        2
#>  3      2     20  1944            1            1        2
#>  4      2     20  1945            1            1        2
#>  5      2     20  1946            1            1        2
#>  6      2     20  1951            1            1        2
#>  7      2     20  1952            1            1        2
#>  8      2     20  1953            1            1        2
#>  9      2     20  1954            1            1        2
#> 10      2     20  1955            1            1        2
#> # ... with 104,544 more rows
```

If we wanted to perfectly recreate the data as Bremer (1992) did it almost 30 years ago, here's how you'd do it in `{peacesciencer}` (albeit with newer data). Still, I think the data innovations that have followed Bremer (1992) merit the approach employed in the user's guide.

Get Multiple Peace Years in One Fell Swoop

`add_peace_years()` is designed to work generally, based on the other data/functions included in the package. For example, assume you wanted to a dyad-year analysis comparing the Correlates of War (CoW) Militarized Interstate Dispute (MID) with the Gibler-Miller-Little conflict data. Just add both in the pipe and ask for peace-years.

```
cow_ddy %>%
  # non-directed, politically relevant, for convenience
  filter(ccode2 > ccode1) %>%
  filter_prd() %>%
  add_cow_mids(keep = NULL) %>%
  add_gml_mids(keep = NULL) %>%
  add_peace_years() -> NDY
```

```
# Here's a snapshot of U.S-Cuba from 1980-89 for illustration sake.
NDY %>%
  filter(ccode1 == 2 & ccode2 == 40) %>%
  select(ccode1:year, cowmidongoing, gmlmidongoing, cowmidspell:gmlmidspell) %>%
  filter(year >= 1980)
#> # A tibble: 37 x 7
#>    ccode1 ccode2  year cowmidongoing gmlmidongoing cowmidspell gmlmidspell
#>     <dbl>  <dbl> <dbl>         <dbl>         <dbl>       <dbl>       <dbl>
#>  1      2     40  1980             0             0           0           0
#>  2      2     40  1981             1             1           1           1
#>  3      2     40  1982             0             0           0           0
#>  4      2     40  1983             1             1           1           1
#>  5      2     40  1984             0             0           0           0
#>  6      2     40  1985             0             0           1           1
#>  7      2     40  1986             0             1           2           2
#>  8      2     40  1987             1             1           3           0
#>  9      2     40  1988             0             0           0           0
#> 10      2     40  1989             0             0           1           1
#> # ... with 27 more rows
```

You can do this with state-year data as well. For example, you can compare how CoW and UCDP code civil wars differently since 1946. Do note, however, that the nature of different state systems used in these data sets means we'll treat one as a master and merge other codes into it.

```
create_stateyears(system = 'gw') %>%
  filter(between(year, 1946, 2019)) %>%
  add_ccode_to_gw() %>%
  add_ucdp_acd(type = "intrastate", only_wars = TRUE) %>%
  add_cow_wars(type = "intra") %>%
  # select just a few things
  select(gwcode, ccode, year, statename, ucdpongoing, ucdponset,
         cowintraongoing, cowintraonset) %>%
  add_peace_years() %>%
  select(gwcode:statename, ucdpspell, cowintraspell, everything()) %>%
  # India is illustrative of how the two differ.
  # UCDP has an intra-state conflict to the level of war early into its existence. CoW does no
  filter(gwcode == 750)
#> # A tibble: 73 x 10
#>    gwcode ccode  year statename ucdpspell cowintraspell ucdpongoing ucdponset
#>     <dbl> <dbl> <dbl> <chr>         <dbl>         <dbl>       <dbl>     <dbl>
#>  1    750   750  1947 India             0             0           0         0
#>  2    750   750  1948 India             1             1           1         1
#>  3    750   750  1949 India             0             2           1         0
#>  4    750   750  1950 India             0             3           1         0
#>  5    750   750  1951 India             0             4           1         0
#>  6    750   750  1952 India             0             5           0         0
```

```
#>  7    750   750  1953 India              1              6           0         0
#>  8    750   750  1954 India              2              7           0         0
#>  9    750   750  1955 India              3              8           0         0
#> 10    750   750  1956 India              4              9           0         0
#> # ... with 63 more rows, and 2 more variables: cowintraongoing <dbl>,
#> #   cowintraonset <dbl>
```

Measure Leader Tenure in Days

`create_leaderyears()`, by default, returns an estimate of leader-tenure as the unique calendar year for the leader. I think of this is a reasonable thing to include, and benchmarking to years is doing some internal lifting elsewhere in the function that generates leader-year data from leader-day data in Archigos. However, it can lead some peculiar observations that may not square with how we knee-jerk think about leader tenure.

I will illustrate what I mean by this with the case of Jimmy Carter from leader-year data standardized to Correlates of War state system membership.

```
leader_years <- create_leaderyears(standardize = 'cow')

leader_years %>% filter(obsid == "USA-1977")
#> # A tibble: 5 x 7
#>   obsid    ccode leader gender  year yrinoffice leaderage
#>   <chr>    <dbl> <chr>  <chr>  <dbl>      <dbl>     <dbl>
#> 1 USA-1977     2 Carter M       1977          1        53
#> 2 USA-1977     2 Carter M       1978          2        54
#> 3 USA-1977     2 Carter M       1979          3        55
#> 4 USA-1977     2 Carter M       1980          4        56
#> 5 USA-1977     2 Carter M       1981          5        57
```

Jimmy Carter took office in January 1977 (year 1) and had a tenure through 1978 (year 2), 1979 (year 3), 1980 (year 4), and exited office in January 1981 (year 5). We know presidents in the American context have four-year terms. This output suggests five years.

If this is that problematic for the research design, especially one that may be interested in what happens to leader behavior after a certain amount of time in office, a user can do something like generate estimates of leader tenure in a given year to the day. Basically, once the core leader-year are generated, the user can use the `create_leaderdays()` function and summarize leader tenure in the year as the minimum number of days the leader was in office in the year and the maximum number of days the leader was in office in the year.

```
# don't standardize the leader-days for this use, just to be safe.
create_leaderdays(standardize = 'none') %>%
  # extract year from date
  mutate(year = lubridate::year(date)) %>%
  # group by leader
  group_by(obsid) %>%
  # count days in office, for leader tenure
  mutate(daysinoffice = seq(1:n())) %>%
```

```
  # group-by leader and year
  group_by(obsid, year) %>%
  # how long was the minimum (maximum) days in office for the leader in the year?
  summarize(min_daysoffice = min(daysinoffice),
            max_dayoffice = max(daysinoffice)) %>%
  #practice safe group-by, and assign to object
  ungroup() -> leader_tenures

# add this information to our data
leader_years %>%
  left_join(., leader_tenures) -> leader_years
```

Here's what this would look like in the case of Jimmy Carter.

```
leader_years %>% filter(obsid == "USA-1977")
#> # A tibble: 5 x 9
#>   obsid    ccode leader gender  year yrinoffice leaderage min_daysoffice
#>   <chr>    <dbl> <chr>  <chr>  <dbl>      <dbl>     <dbl>          <int>
#> 1 USA-1977     2 Carter M       1977          1        53              1
#> 2 USA-1977     2 Carter M       1978          2        54            347
#> 3 USA-1977     2 Carter M       1979          3        55            712
#> 4 USA-1977     2 Carter M       1980          4        56           1077
#> 5 USA-1977     2 Carter M       1981          5        57           1443
#> # ... with 1 more variable: max_dayoffice <int>
```

This measure might be more useful. Basically, Jimmy Carter was a new leader in 1977 (`min_daysoffice = 1`). By 1978, he had almost a year under his belt (i.e. Jan. 1, 1978 was his 347th day in office). By time he left office in 1981, he had completed 1,462 days on the job.

`create_leaderyears()` elects to not create this information for the user. No matter, it does not take much effort for the user to create it if this is the kind of information they wanted.

Vignette: {peacesciencer} Data Versions

```
library(tidyverse)
library(peacesciencer)
library(kableExtra)
```

These are the data versions available in {peacesciencer}. Do note the user can find specific data versions with more targeted use of the `ps_version()` function in this package. Without an argument, `ps_version()` produces a data frame of all the data versions in this package.

```
ps_version() %>%
  kbl(., caption = "Data Versions in `{peacesciencer}`",
      align =c("c", "l", "c", "c"),
      booktabs = TRUE, longtable = TRUE)  %>%
```

```
kable_styling(position = "center", full_width = F,
              bootstrap_options = "striped") %>%
row_spec(0, bold=TRUE)
```

Table A.1: Data Versions in 'peacesciencer'

category	data	version	bibtexkey
states	Correlates of War State System Membership	2016	cowstates2016
leaders	LEAD	2015	ellisetal2015lead
leaders	Archigos	4.1	goemansetal2009ia
alliance	ATOP	5	leedsetal2002atop
democracy	Polity	2017	marshalletal2017p
democracy	{QuickUDS}	0.2.3	marquez2016qme
democracy	V-Dem	10	coppedgeetal2020vdem
capitals	{peacesciencer}	2020	peacesciencer-package
contiguity	Correlates of War Direct Contiguity	3.2	stinnettetal2002cow
igo	Correlates of War IGOs	3	pevehouseetal2020tow
majors	Correlates of War	2016	cowstates2016
conflict_interstate	Correlates of War Militarized Interstate Disputes	5	palmeretal2021mid5
distance	{Cshapes}	2	schvitz2021mis
capabilities	Correlates of War National Material Capabilities	5	singer1987rcwd
gdp	SDP	2020	andersetal2020bbgb
sdp	SDP	2020	andersetal2020bbgb
population	SDP	2020	andersetal2020bbgb
trade	Correlates of War Trade	4	barbierietal2009td
conflict_intrastate	Correlates of War Intra-State War	4.1	dixonsarkees2016giw
conflict_interstate	Correlates of War Inter-State War	4	sarkeeswayman2010rw
fractionalization	CREG	2012	nardulli2012creg
polarization	CREG	2012	nardulli2012creg
conflict_interstate	Gibler-Miller-Little (GML)	2.2.1	gibleretal2016amid
states	Gleditsch-Ward	2017	gledtischward1999rlis
leaders	Leader Willingness to Use Force	2020	cartersmith2020fml
terrain	Ruggedness	2012	nunnpuga2012r
terrain	% Mountainous	2014	giblermiller2014etts
rivalries	Thompson and Dreyer	2012	thompsondreyer2012hir
conflict_intrastate	UCDP Armed Conflicts	20.1	gleditschetal2002ac
conflict_intrastate	UCDP Onsets	19.1	pettersson2019ov
dyadic_similarity	FPSIM	2	haege2011cc

Every data set that is used by a function in this package is included in this table (and with the `ps_version()` function). Users can see a category of the type of data (which can be used for more careful searches of the underlying data), a description of the data set, a version number associated with that data set in `{peacesciencer}`,

along with a BibTeX key. Users should interpret data versions as years as instances where the data are not formally versioned, per se, and the year corresponds with a year of a last update or a year of publication. For example, Anders et al. (2020) released their state-year simulations of population, surplus domestic product, and gross domestic product in 2020. The data are not formally versioned and the year corresponds with, in this case, the publication.

Users can use the BibTeX key (`bibtexkey`) to search the dataframe of citations in this package. For example, we can use `ps_cite()` to get a full citation for Carter and Smith's (2020) estimates of leader willingness to use force.

```
ps_cite("cartersmith2020fml", column = "bibtexkey")
#> @Article{cartersmith2020fml,
#>   Author = {Jeff Carter and Charles E. Smith},
#>   Journal = {American Political Science Review},
#>   Number = {4},
#>   Pages = {1352--1358},
#>   Title = {A Framework for Measuring Leaders' Willingness to Use Force},
#>   Volume = {114},
#>   Year = {2020},
#>   Keywords = {lwuf, add_lwuf()}
#> }
```

```
create_dyadyears(system = 'gw')
#> # A tibble: 2,029,622 x 3
#>    gwcode1 gwcode2  year
#>      <dbl>   <dbl> <int>
#>  1       2      20  1867
#>  2       2      20  1868
#>  3       2      20  1869
#>  4       2      20  1870
#>  5       2      20  1871
#>  6       2      20  1872
#>  7       2      20  1873
#>  8       2      20  1874
#>  9       2      20  1875
#> 10       2      20  1876
#> # ... with 2,029,612 more rows
```

References

Anders, Therese, Christopher J Fariss & Jonathan N Markowitz (2020) Bread before guns or butter: Introducing surplus domestic product (SDP). *International Studies Quarterly* 64(2): 392–405.

Arel-Bundock, Vincent (2021) *Modelsummary: Summary Tables and Plots for Statistical Models and Data: Beautiful, Customizable, and Publication-Ready* (https://CRAN.R-project.org/package=modelsummary).

Carter, Jeff & Charles E Smith (2020) A framework for measuring leaders' willingness to use force. *American Political Science Review* 114(4): 1352–1358.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, Max Kuhn, Thomas Lin Pedersen, Evan Miller, Stephan Milton Bache, Kirill Müller, Jeroen Ooms, David Robinson, Dana Paige Seidel, Vitalie Spinu, Kohske Takahashi, Davis Vaughan, Claus Wilke, Kara Woo & Hiroaki Yutani (2019) Welcome to the tidyverse. *Journal of Open Source Software* 4(43): 1686.

Xie, Yihui (2015) *Dynamic Documents with R and Knitr*. Chapman; Hall/CRC.

Xie, Yihui (2016) *Bookdown: Authoring Books and Technical Documents with R Markdown*. Boca Raton, Florida: Chapman; Hall/CRC (https://bookdown.org/yihui/bookdown).

Xie, Yihui, Christophe Dervieux & Emily Riederer (2020) *R Markdown Cookbook*. Boca Raton, Florida: Chapman; Hall/CRC (https://bookdown.org/yihui/rmarkdown-cookbook).

Zhu, Hao (2021) *kableExtra: Construct Complex Table with "Kable" and Pipe Syntax* (https://CRAN.R-project.org/package=kableExtra).