

Data curation: The case of Call Reports

Ian D. Gow

4 February 2026

I recently (Gow, 2026) proposed an extended version of model of the data science “whole game” of in [R for Data Science](#) (Wickham et al., 2023). In Gow (2026), I used Australian stock price data to illustrate the data curation process and, in this note, I use “Call Report” data for United States banks as an additional illustration of my proposed extension.

My extension of the data science “whole game”—depicted in Figure 1 below—adds a *persist* step to the original version, groups it with *import* and *tidy* into a single process, which I call *Curate*. As a complement to the new *persist* step, I also add a *load* step to the *Understand* process.¹

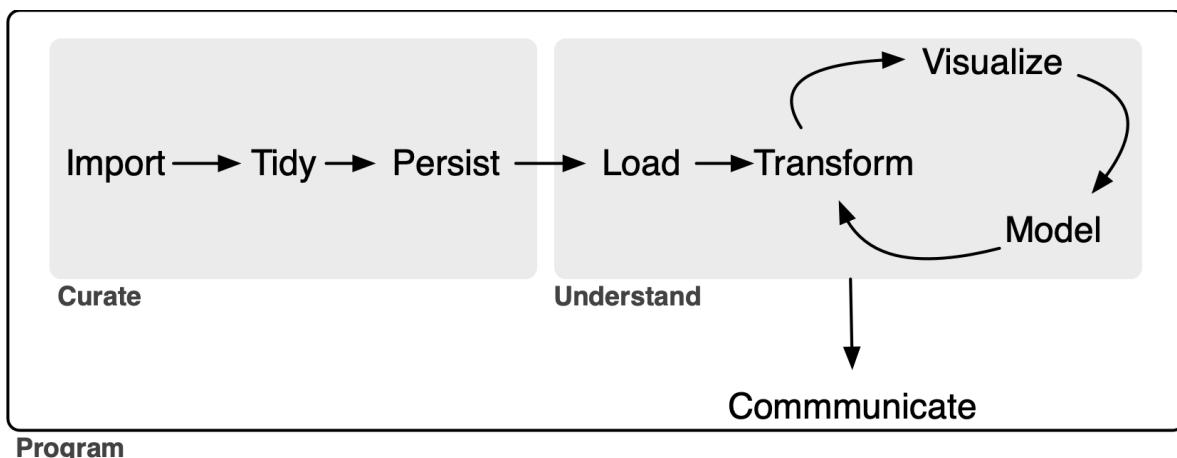


Figure 1: A representation of the data science workflow

In this note, as in Gow (2026), I focus on the data curation (*Curate*) process. My rationale for separating *Curate* from *Understand* is that I believe it clarifies certain best practices in the curation of data. In Gow (2026), I used the notion of a service-level agreement to delineate how the two processes can be delineated. My conception of *Curate* (Gow, 2026) encompasses some tasks that are included in the *transform* step (part of the *Understand* process) of Wickham et al. (2023).

¹As will be seen this, *load* step will not generally be an elaborate one. The inclusion of a separate *load* step serves more to better delineate the distinction between the *Curate* process and the *Understand* process.

While I will argue that even the sole analyst who will perform all three processes can benefit from thinking about *Curate* separate from *Understand*, it is perhaps easiest to conceive of the *Curate* and *Understand* processes as involving different individuals or organizational units of the “whole game” of a data analysis workflow. In Gow (2026), I used the idea of a service-level agreement to delineate the division of responsibilities between the *Curate* and *Understand* teams. In effect, I will act as a self-appointed, single-person, unpaid *Curate* team and I imagine potential users of call report data as my *Understand* clients.

💡 Tip

This note was written and rendered with [Quarto](#) using [RStudio](#), an integrated development environment (IDE) for working with R. The source code for this note is available [here](#) and the latest PDF version is available [here](#).

In writing the R portions of this note, I used the packages listed below.²

I have not submitted `ffiec.pq` to CRAN. First, in R install the `pak` package using `install.packages("pak")`. Then, use `pak` to install my package using `pak::pak("iangow/ffiec.pq")`.

```
library(ffiec.pq)
library(tidyverse)
library(farr)
library(dbplyr)
library(DBI)
```

1 Call Reports

According to [its website](#), the Federal Financial Institutions Examination Council (FFIEC) “is an interagency body … focused on promoting consistency in examination activities [related to United States financial institutions]. The FFIEC”does not regulate financial institutions [and its] jurisdiction is limited to prescribing uniform principles, standards, and report forms for the federal examination of financial institutions and making recommendations to promote uniformity in the supervision of financial institutions.”

One of the services provide by the FFEIC is its “Central Data Repository’s Public Data Distribution web site. Through [this site](#) you can obtain Reports of Condition and Income (Call Reports) for most FDIC-insured institutions. The information available on this site is updated to reflect the most recent data for both prior and current periods. The earliest data provided is from March 31, 2001.”

²Execute `install.packages(c("tidyverse", "farr", "DBI", "duckdb", "dbplyr"))` within R to install all the packages other than `ffiec.pq` that you will need to run the R code in this note.

Many academic studies use Call Report data. For example, Kashyap et al. (2002, p. 52) states “our data come from the ‘Call Reports,’ the regulatory filings that all commercial banks having insured deposits submit each quarter. The Call Reports include detailed information on the composition of bank balance sheets and some additional data on off-balance-sheet items. These data are reported at the level of the individual bank.”

While raw data are offered by the FFIEC, some work is required to arrange the data into a form amenable to further analysis. In other words, some curation of the data is required.

Several commercial data providers of the FFIEC data in a curated form, but accessing those requires an institutional or individual subscription. It appears that the Federal Reserve Bank of Chicago provided a curated data set on its [website](#), but no longer does so.

Other services provide “API wrappers” that allow users to access the data via the FFIEC API, but this approach is best suited to real-time feeds of small amounts of data. Collation of the data into a single repository using this approach seems unfeasible.

Regardless of the availability of paid curation services, in this note I will imagine that such sources either do not exist or are unavailable to my hypothetical *Understand* clients and illustrate how one can curate Call Reports data from the FFIEC source.

1.1 Getting the raw data

The FFIEC [Bulk Data Download site] provides the Call Report data in two forms. The first is as zipped tab-delimited data files, one for each quarter. The second is as zipped XBRL data files, one for each quarter. At the time of writing, getting the complete data archive amounts to point-and-clicking to download each of the 99 files for each format.

The FFIEC data sets are not as amenable to automated downloading as those offered by other government agencies such as the SEC (see my earlier [note on XBRL data](#)), the PCAOB (see my [note on Form AP data](#)), or even the Federal Reserve itself (I used data from the [MDRM site](#) in preparing this note). However, a group of individuals has contributed the Python package `ffiec_data_collector` that we can use to collect the data.³

To install this Python package, you first need to [install Python](#) and then install the `ffiec_data_collector` using a command like `pip install ffec_data_collector`.

As discussed in [Appendix E](#) of Gow and Ding (2024), I organize my raw and processed data in a repository comprising a single parent directory and several sub-directories corresponding to various data sources and projects. For some data sets, this approach to organization facilitates switching code from using (say) data sources provided by Wharton Research Data Services (WRDS) to using local data in my data repository. I will adopt that approach for the purposes of this note.

³I discovered this package after writing most of this note. In my case, I pointed-and-clicked to get many of the files and [Martien Lubberink](#) of Victoria University of Wellington kindly provided the rest.

As the location of my “raw data” repository is found in the the environment variable RAW_DATA_DIR, I can identify that location in Python easily. The following code specifies the download directory as the directory ffiec within my raw data repository.⁴

```
import os
from pathlib import Path
print(os.environ['RAW_DATA_DIR'])

/Users/igow/Dropbox/raw_data

download_dir = Path(os.environ['RAW_DATA_DIR']) / "ffiec"
```

Having specified a location to put the downloaded files, it is a simple matter to adapt a script provided on the [package website](#) to download the raw data files.

```
import ffiec_data_collector as fdc
import time

downloader = fdc.FFIECDownloader(download_dir=download_dir)

periods = downloader.select_product(fdc.Product.CALL_SINGLE)

results = []
for period in periods[:4]:

    print(f"Downloading {period.yyyymmdd}...", end=" ")
    result = downloader.download(
        product=fdc.Product.CALL_SINGLE,
        period=period.yyyymmdd,
        format=fdc.FileFormat.TSV
    )
    results.append(result)

    if result.success:
        print(f" ({result.size_bytes:,} bytes)")
    else:
        print(f" Failed: {result.error_message}")

# IMPORTANT: Be respectful to government servers
```

⁴I recommend that readers follow a similar approach if following along with note, as it makes subsequent steps easier to implement. A reader can simply specify `os.environ['RAW_DATA_DIR'] = "/Users/igow/Dropbox/raw_data"`, substituting a location where the data should go on his or her computer.

```

# Add delay between requests to avoid overloading the server
time.sleep(1) # 1 second delay - adjust as needed

Downloading 20250930... (5,686,532 bytes)
Downloading 20250630... (6,231,006 bytes)
Downloading 20250331... (5,704,421 bytes)
Downloading 20241231... (6,605,092 bytes)

# Summary
successful = sum(1 for r in results if r.success)
print(f"\nCompleted: {successful}/{len(results)} downloads")

Completed: 4/4 downloads

```

Note that the code above downloads just the most recent four files available on the site. Remove `[:4]` from the line `for period in periods[:4]:` to download *all* files. Note that the package website recommends using `time.sleep(5)` in place of `time.sleep(1)` to create a five-second delay and this may be a more appropriate choice if you are downloading all 99 files using this code. Note that the downloaded files occupy about 800 MB of disk space, so make sure you have that available if running this code.

1.1.1 XBRL files

While this note does not use the XBRL files, you can download them using `ffiec_data_collector` by simply replacing TSV with XBRL in the code above. These zip files are larger than the TSV zip files, occupying about 6 GB of disk space. The `ffiec.pq` package does offer some rudimentary ability to process these files, but working with these is slow. To illustrate I just process one XBRL zip file.

```

zipfiles <- ffiec_list_zips(type = "xbrl")
ffiec_process_xbrls(zipfiles$zipfile[1]) |> system_time()

      user   system elapsed
207.576  69.438 278.471

# A tibble: 1 x 4
  zipfile           date_raw date       parquet
  <chr>             <chr>    <date>    <chr>
1 FFIEC CDR Call Bulk XBRL 03312001.zip 20010331 2001-03-31 xbrl_20010331.parqu~
```

1.2 Processing the data

With the raw data files in hand, the next task it to process these into files useful for analysis. For reasons I will discuss below, I will process the data into Parquet files. The Parquet format is described in *R for Data Science* (Wickham et al., 2023, p. 393) as “an open standards-based format widely used by big data systems.” Parquet files provide a format optimized for data analysis, with a rich type system. More details on the parquet format can be found in [Chapter 22](#) of Wickham et al. (2023) and every code example in Gow and Ding (2024) can be executed against Parquet data files created using my `db2pq` Python package as described in Appendix E of that book.

The easiest way to run the code I used to process the data is to install the `ffiec.pq` R package I have made available on GitHub. And the easiest way to run the package is to set the locations for the downloaded raw data files from above and for the processed data using the environment variables `RAW_DATA_DIR` and `DATA_DIR`, respectively. By default, the `ffiec.pq` package assumes that the raw data files can be found in a directory `ffiec` that is a subdirectory of `RAW_DATA_DIR`. Also, the `ffiec.pq` package will place the processed data it creates in a directory `ffiec` that is a subdirectory of `DATA_DIR`.

I already have these environment variables set:

```
Sys.getenv("RAW_DATA_DIR")
```

```
[1] "/Users/igow/Dropbox/raw_data"
```

```
Sys.getenv("DATA_DIR")
```

```
[1] "/Users/igow/Dropbox/pq_data"
```

But, even if I did not, I could set them within R using commands like the following. You should set `RAW_DATA_DIR` to match what you used above in Python and you should set `DATA_DIR` to point to the location where you want to put the processed files. The processed files will occupy about 3 GB of disk space, so make sure you have room for these there.

```
Sys.setenv(RAW_DATA_DIR="/Users/igow/Dropbox/raw_data")
Sys.setenv(DATA_DIR="/Users/igow/Dropbox/pq_data")
```

Having set these environment variables, I can load my package and run a single command `ffiec_process()` without any arguments to process *all* the raw data files. This takes four or five minutes for me:⁵

⁵Note I am using the argument `use_multicore = TRUE`, which gives me about a five-time improvement in performance.

```
results <- ffiec_process(use_multicore = TRUE) |> system_time()
```

```
user  system elapsed
10.518 7.787 309.870
```

Note that, behind the scenes, the `ffiec_process()` extracts the data in two phases. In the first phase, it processes the data for each schedule for each quarter into Parquet file. This results in 3674 Parquet files. In the second phase, `ffiec_process()` proceeds to organize the data in the 3674 Parquet files by variable type to facilitate working with the data. Once the data have been organized, the 3674 schedule-and-quarter-specific Parquet files are discarded.

The `results` table returned by the `ffiec_process()` function above reflects the outcome of the first phase, as that is when any problems arising from malformed data are expected to arise. If there were any issues in reading the data for a schedule in a quarter, then the variable `ok` for the corresponding row of `results` will be `FALSE`. We can easily confirm that all rows have `ok` equal to `TRUE`:

```
results |> count(ok)
```

```
# A tibble: 1 x 2
  ok      n
  <lgl> <int>
1 TRUE    3674
```

The results table also includes the field `repairs` that we can inspect to determine if any “repairs” were made to the data as it was processed. As can be seen, a minority of the 3674 files needed repairs. I discuss these repairs in more detail in Section 2.1.1.

```
results |>
  unnest(repairs) |>
  count(repairs)
```

```
# A tibble: 2 x 2
  repairs      n
  <chr>       <int>
1 newline-gsub 101
2 tab-repair    2
```

1.3 Using the data with R

So what have we just done? In a nutshell, we have processed each of the 99 zip files into six files.

The first file is the “[Panel of Reporters](#)” (POR) table, which provides details on the financial institutions filing in the respective quarter. We can use the `pq_file` argument to `ffiec_scan_pqs()` to load a single file into DuckDB.

```
por_20250930 <- ffiec_scan_pqs(db, pq_file="por_20250930.parquet")  
por_20250930
```

```
# A query: ?? x 13  
# Database: DuckDB 1.4.4-dev6 [root@Darwin 25.3.0:R 4.5.2/:memory:]  
IDRSSD fdic_certificate_number occ_charter_number ots_docket_number  
<int> <chr> <chr> <chr>  
1 37 10057 <NA> 16553  
2 242 3850 <NA> <NA>  
3 279 28868 <NA> 2523  
4 354 14083 <NA> <NA>  
5 457 10202 <NA> <NA>  
6 505 6959 1253 <NA>  
7 1155 17639 <NA> <NA>  
8 1351 9392 <NA> <NA>  
9 1454 19184 15359 <NA>  
10 1669 3384 13541 <NA>  
# i more rows  
# i 9 more variables: primary_aba_routing_number <chr>,  
#   financial_institution_name <chr>, financial_institution_address <chr>,  
#   financial_institution_city <chr>, financial_institution_state <chr>,  
#   financial_institution_zip_code <chr>,  
#   financial_institution_filing_type <chr>,  
#   last_date_time_submission_updated_on <dttm>, date <date>
```

But more often, it will be more convenient to just read all files in one step:

```
por <- ffiec_scan_pqs(db, "por")  
por
```

```
# A query: ?? x 13  
# Database: DuckDB 1.4.4-dev6 [root@Darwin 25.3.0:R 4.5.2/:memory:]  
IDRSSD fdic_certificate_number occ_charter_number ots_docket_number
```

```

<int> <chr>          <chr>          <chr>
1    37 10057           <NA>            16553
2    242 3850           12096           12627
3    279 28868           <NA>            2523
4    354 14083           <NA>            <NA>
5    439 16498           <NA>            <NA>
6    457 10202           <NA>            <NA>
7    505 6959            1253             <NA>
8    1146 17562           <NA>            8980
9    1155 17639           <NA>            13449
10   1351 9392           <NA>            <NA>
# i more rows
# i 9 more variables: primary_aba_routing_number <chr>,
#   financial_institution_name <chr>, financial_institution_address <chr>,
#   financial_institution_city <chr>, financial_institution_state <chr>,
#   financial_institution_zip_code <chr>,
#   financial_institution_filing_type <chr>,
#   last_date_time_submission_updated_on <dttm>, date <date>

```

Each of the other five files represents data from the schedules for that quarter for a particular data type, as shown in Table 1

Table 1: Table keys to Arrow types

Key	Arrow type
float	Float64
int	Int32
str	Utf8
date	Date32
bool	Boolean

We can use the data set `ffiec_items` to find out where a variable is located, based on its Arrow type.

ffiec_items

```

# A tibble: 5,141 x 5
  item      mnemonic item_code item_name          data_type
  <chr>     <chr>      <chr>     <chr>          <chr>
1 RCFA2170  RCFA       2170      Total assets      Float64
2 RCFA3128  RCFA       3128      Allocated transfer risk reserves  Float64

```

```

3 RCFA3792 RCFA      3792      Total qualifying capital allowable under regulation   Float64
4 RCFA5310 RCFA      5310      General loan and lease valuation allowance   Float64
5 RCFA5311 RCFA      5311      Tier 2 (supplementary) capital   Float64
6 RCFA7204 RCFA      7204      Tier 1 leverage capital ratio   Float64
7 RCFA7205 RCFA      7205      Total risk-based capital ratio   Float64
8 RCFA7206 RCFA      7206      Tier 1 risk-based capital ratio   Float64
9 RCFA8274 RCFA      8274      Tier 1 capital allowable under the risk-based capital ratio   Float64
10 RCFAA223 RCFA     A223      Risk-weighted assets (net of allowances)   Float64
# i 5,131 more rows

```

As might be expected, most variables have type `Float64` and will be found in the `ffiec_float` tables.

```
ffiec_items |> count(data_type, sort = TRUE)
```

```

# A tibble: 5 x 2
  data_type     n
  <chr>     <int>
1 Float64    4909
2 Int32       115
3 String       73
4 Boolean      42
5 Date32       2

```

1.3.1 Floating-point data items

We can access the data in `ffiec_float` files using `ffiec_scan_pqs()`. We can append a date in `yyyymmdd` form to load the file for just one quarter.

```
ffiec_float_20250930 <-
  ffiec_scan_pqs(db, pq_file = "ffiec_float_20250930.parquet")
```

But more often, we might just specify `ffiec_float`, so to scan all data in one step.

```
ffiec_float <- ffiec_scan_pqs(db, "ffiec_float")
ffiec_float |> head() |> collect()
```

```

# A tibble: 6 x 4
  IDRSSD date      item      value
  <int> <date>    <chr>    <dbl>
1        1 2025-09-30 RCFA3792  3792
2        1 2025-09-30 RCFA5310  5310
3        1 2025-09-30 RCFA5311  5311
4        1 2025-09-30 RCFA7204  7204
5        1 2025-09-30 RCFA7205  7205
6        1 2025-09-30 RCFA7206  7206

```

```

1 37 2001-03-31 RCON0010 1729
2 37 2001-03-31 RCON0030 0
3 37 2001-03-31 RCON0031 0
4 37 2001-03-31 RCON0032 0
5 37 2001-03-31 RCON0071 0
6 37 2001-03-31 RCON0081 1729

```

From the output above, we see that item RIAD4230 appears on both Schedule RI (ri) and Schedule RI-BII (ribii). The first question might be: What is RIAD4230? We can get the answer from ffiec_items:

```

ffiec_items |> filter(item == "RIAD4230")

# A tibble: 1 x 5
  item      mnemonic item_code item_name          data_type
  <chr>    <chr>     <chr>    <chr>            <chr>
1 RIAD4230 RIAD      4230     Provision for loan and lease losses Float64

```

Schedule RI is the income statement and “Provision for loan and lease losses” is an expense we would expect to see there for a financial institution. Schedule RI-BII is “Charge-offs and Recoveries on Loans and Leases” and provides detail on loan charge-offs and recoveries, broken out by loan category, for the reporting period. As part of processing the data, the ffiec.pq package confirms that the value for any given item for a specific IDRSSD and date is the same across schedules for *all* items in the data.

1.3.2 Integer data items

As can be seen by inspecting ffiec_items, most items with data_type of Int32 are counts of some kind, such as “number of loans” or “number of accounts”:

```

ffiec_items |>
  filter(data_type == "Int32") |>
  select(item, item_name)

# A tibble: 115 x 2
  item      item_name
  <chr>    <chr>
1 RCFD3561 Number of loans made to executive officers since the previous call ~
2 RCFD6165 Number of executive officers; directors; and principal shareholders~
3 RCFDB870 Personal trust and agency accounts - number of managed accounts
4 RCFDB871 Personal trust and agency accounts - number of non-managed accounts

```

```
5 RCFDB874 Employee benefit-defined contribution - number of managed accounts
6 RCFDB875 Employee benefit-defined contribution - number of non-managed accou~
7 RCFDB878 Employee benefit-defined benefit - number of managed accounts
8 RCFDB879 Employee benefit-defined benefit - number of non-managed accounts
9 RCFDB882 Other retirement accounts - number of managed accounts
10 RCFDB883 Other retirement accounts - number of non-managed accounts
# i 105 more rows
```

```
ffiec_int <- ffiec_scan_pqs(db, "ffiec_int")
```

If we want to add data from `ffiec_items` to tables from Parquet files, we need to copy the `ffiec_items`, a local data frame, to our DuckDB database. We can do this using the `copy = TRUE` argument when joining tables.

```
ffiec_int |>
  inner_join(ffiec_items, copy = TRUE) |>
  select(IDRSSD, date, item, value, item_name) |>
  head() |>
  collect()
```

Joining with `by = join_by(item)`

```
# A tibble: 6 x 5
  IDRSSD date      item      value item_name
  <int> <date>    <chr>    <int> <chr>
1     37 2001-03-31 RCON2722    84 Number of deposit accounts of more than $100~
2     37 2001-03-31 RCON3561     0 Number of loans made to executive officers s~
3     37 2001-03-31 RCON6165     0 Number of executive officers; directors; and~
4     37 2001-03-31 RCONA545     0 Legal title and FDIC certificate number of p~
5     37 2001-03-31 RIAD4150    27 Number of full time equivalent employees on ~
6    242 2001-03-31 RCON2722   16 Number of deposit accounts of more than $100~
```

1.3.3 Date data items

```
ffiec_date <- ffiec_scan_pqs(db, "ffiec_date")
```

Surprisingly, there are very few data items related to dates in the Call Report data.

```

ffiec_date |>
  count(item)

# A query: ?? x 2
# Database: DuckDB 1.4.4-dev6 [root@Darwin 25.3.0:R 4.5.2/:memory:]
  item      n
  <chr>    <dbl>
1 RCON9999 655810
2 RIAD9106 2707

ffiec_date |>
  inner_join(ffiec_items, by = "item", copy = TRUE) |>
  count(item, item_name) |>
  collect()

# A tibble: 2 x 3
  item      item_name          n
  <chr>    <chr>            <dbl>
1 RCON9999 Reporting date (cc;yr;mo;da) 655810
2 RIAD9106 If the reporting bank has restated its balance sheet as a res~ 2707

```

We can't quite see all of `item_name` for the first row above, so let's use `strwrap()` to fit it on the page:

```

ffiec_items |>
  filter(item == "RIAD9106") |>
  select(item_name) |>
  pull() |>
  strwrap(width = 78)

```

```

[1] "If the reporting bank has restated its balance sheet as a result of applying"
[2] "push down accounting this calendar year; report the date of the bank's"
[3] "acquisition"

```

1.3.4 Boolean data items

```

ffiec_items |>
  filter(data_type == "Boolean") |>
  select(item, item_name)

```

```
# A tibble: 42 x 2
  item      item_name
  <chr>    <chr>
1 RCFD4088 The entity's world wide web site is interactive?
2 RCFDA345 Fiduciary powers granted but not exercised
3 RCFDA346 Fiduciary powers granted and exercised but no dollar values to repo-
4 RCFDB569 Does the reporting bank sell private label or third party mutual fu-
5 RCFDB867 Does the bank have any fiduciary or related activity (in the form o-
6 RCFDK656 Banker's bank certification: does the reporting institution meet bo-
7 RCFDK659 Custodial bank certification: does the reporting institution meet t-
8 RCON4088 The entity's world wide web site is interactive?
9 RCON6860 Agricultural loans to small farms indicator
10 RCON6861 Estimate amount of uninsured deposits method indicator
# i 32 more rows
```

```
ffiec_bool <- ffiec_scan_pqs(db, "ffiec_bool")  
  
ffiec_bool |>  
  count(value) |>  
  collect()
```

```
# A tibble: 2 x 2
  value      n
  <lg1>    <dbl>
1 FALSE 6769662
2 TRUE  1893025
```

1.3.5 String data items

```
ffiec_str <-  
  ffiec_scan_pqs(db, "ffiec_str")  
  
ffiec_str |>  
  inner_join(ffiec_items, by = join_by(item), copy = TRUE) |>  
  count(mnemonic, sort = TRUE) |>  
  collect()
```

```
# A tibble: 14 x 2  
  mnemonic      n  
  <chr>        <dbl>
```

```

1 TEXT      2237174
2 RSSD     1988840
3 RCON     331490
4 TE01      36825
5 TE02      15525
6 TE03      9205
7 TE04      6167
8 TE05      4428
9 TE06      3494
10 RCFD     2584
11 TE07     1770
12 TE08     1384
13 TE09     1146
14 TE10     1007

```

1.3.6 Schedules data

The final set of tables contain information about the

One field that warrants explanation is `schedule`, which is the list of schedules from which the data item in question was taken. As some values appear on multiple schedules even in a given quarter, `schedule` may contain more than one item.

```

ffiec_schedules <- ffiec_scan_pqs(db, "ffiec_schedules")

ffiec_schedules |>
  mutate(n_schedules = length(schedule)) |>
  filter(n_schedules > 1) |>
  mutate(schedule = unnest(schedule)) |>
  collect(n = 10)

# A tibble: 10 x 3
  item    schedule n_schedules
  <chr>   <chr>        <dbl>
1 RCFD0010 rca            2
2 RCFD0010 rcr            2
3 RCFD0426 rc             2
4 RCFD0426 rcm            2
5 RCFD1350 rc             2
6 RCFD1350 rcr            2
7 RCFD1754 rc             3
8 RCFD1754 rcb            3

```

```

9 RCFD1754 rcr          3
10 RCFD1773 rc          3

```

1.4 A small example

If we were experts in Call Report data, we might know that domestic total assets is reported as item RCFD2170 (on Schedule RC) for banks reporting on a consolidated basis and as item RCON2170 for banks reporting on an unconsolidated basis. We can make a balance sheet table using the following code.

```

bs_data <-
  ffiec_float |>
  ffiec_pivot(items = c("RCFD2170", "RCON2170")) |>
  mutate(total_assets = coalesce(RCFD2170, RCON2170)) |>
  system_time()

user   system elapsed
0.077   0.022   0.027

bs_data |> head() |> collect()

# A tibble: 6 x 5
  IDRSSD date      RCFD2170 RCON2170 total_assets
  <int> <date>    <dbl>    <dbl>      <dbl>
1 259554 2001-03-31     NA    38093      38093
2 264941 2001-03-31     NA   203024     203024
3 265340 2001-03-31     NA   89941      89941
4 265948 2001-03-31     NA   72813      72813
5 274829 2001-03-31     NA   357247     357247
6 490450 2001-03-31     NA   730843     730843

bs_data |>
  filter(is.na(RCON2170), !is.na(RCFD2170)) |>
  head() |>
  collect()

# A tibble: 6 x 5
  IDRSSD date      RCFD2170 RCON2170 total_assets
  <int> <date>    <dbl>    <dbl>      <dbl>
1 595869 2011-09-30 10483789     NA    10483789

```

```

2 1160161 2013-06-30 1071076      NA      1071076
3 214807 2013-09-30 53339000      NA      53339000
4 3382547 2020-06-30 322880      NA      322880
5 3382547 2021-03-31 734074      NA      734074
6 538802 2025-03-31 2394408      NA      2394408

```

```

bs_data |>
  filter(!is.na(RCON2170), !is.na(RCFD2170)) |>
  head() |>
  collect()

```

```

# A tibble: 6 x 5
  IDRSSD date      RCFD2170 RCON2170 total_assets
  <int> <date>    <dbl>    <dbl>      <dbl>
1 212465 2001-06-30 35419903 34232599    35419903
2 44134 2004-12-31 1677853 1671797     1677853
3 750864 2005-06-30 1873788 1640287     1873788
4 212465 2006-06-30 50054634 49601065    50054634
5 918918 2008-09-30 3777224 3554179     3777224
6 577137 2001-06-30 213682   185972     213682

```

```

bs_data |>
  filter(!is.na(RCON2170), !is.na(RCFD2170)) |>
  filter(RCFD2170 < RCON2170) |>
  collect()

```

```

# A tibble: 0 x 5
# i 5 variables: IDRSSD <int>, date <date>, RCFD2170 <dbl>, RCON2170 <dbl>,
#   total_assets <dbl>

```

Let's say we're interested in the top 10 banks by total assets on this date. This is easy enough to produce.

```

top_5 <-
  bs_data |>
  filter(date == "2025-09-30") |>
  window_order(desc(total_assets)) |>
  mutate(ta_rank = row_number()) |>
  filter(ta_rank <= 5) |>
  select(IDRSSD, date, total_assets, ta_rank) |>
  compute() |>
  system_time()

```

```
user  system elapsed
0.209  0.071   0.139
```

```
top_5 |> collect()
```

```
# A tibble: 5 x 4
  IDRSSD date      total_assets ta_rank
  <int> <date>        <dbl>    <dbl>
1 852218 2025-09-30  3813431000     1
2 480228 2025-09-30  2651090000     2
3 476810 2025-09-30  1844189000     3
4 451965 2025-09-30  1767105000     4
5 504713 2025-09-30  679293260      5
```

However, it's hard for us to know who these banks if we haven't committed the table of IDRSSD values to memory. Fortunately, we can get the information we need from por.

```
top_5_names <-
  top_5 |>
  inner_join(por |>
    select(IDRSSD, date, financial_institution_name),
    join_by(IDRSSD, date)) |>
  arrange(ta_rank) |>
  select(IDRSSD, financial_institution_name, ta_rank) |>
  rename(bank = financial_institution_name) |>
  compute() |>
  system_time()
```

```
user  system elapsed
0.038  0.048   0.035
```

```
top_5_names |> collect()
```

```
# A tibble: 5 x 3
  IDRSSD bank                      ta_rank
  <int> <chr>                    <dbl>
1 852218 JPMORGAN CHASE BANK, NATIONAL ASSOCIATION 1
2 480228 BANK OF AMERICA, NATIONAL ASSOCIATION      2
3 476810 CITIBANK, N.A.                 3
4 451965 WELLS FARGO BANK, NATIONAL ASSOCIATION      4
5 504713 U.S. BANK NATIONAL ASSOCIATION               5
```

```
bs_panel_data <-
  bs_data |>
  select(IDRSSD, date, total_assets) |>
  inner_join(top_5_names, join_by(IDRSSD)) |>
  collect() |>
  system_time()
```

```
user  system elapsed
4.766 0.948 0.883
```

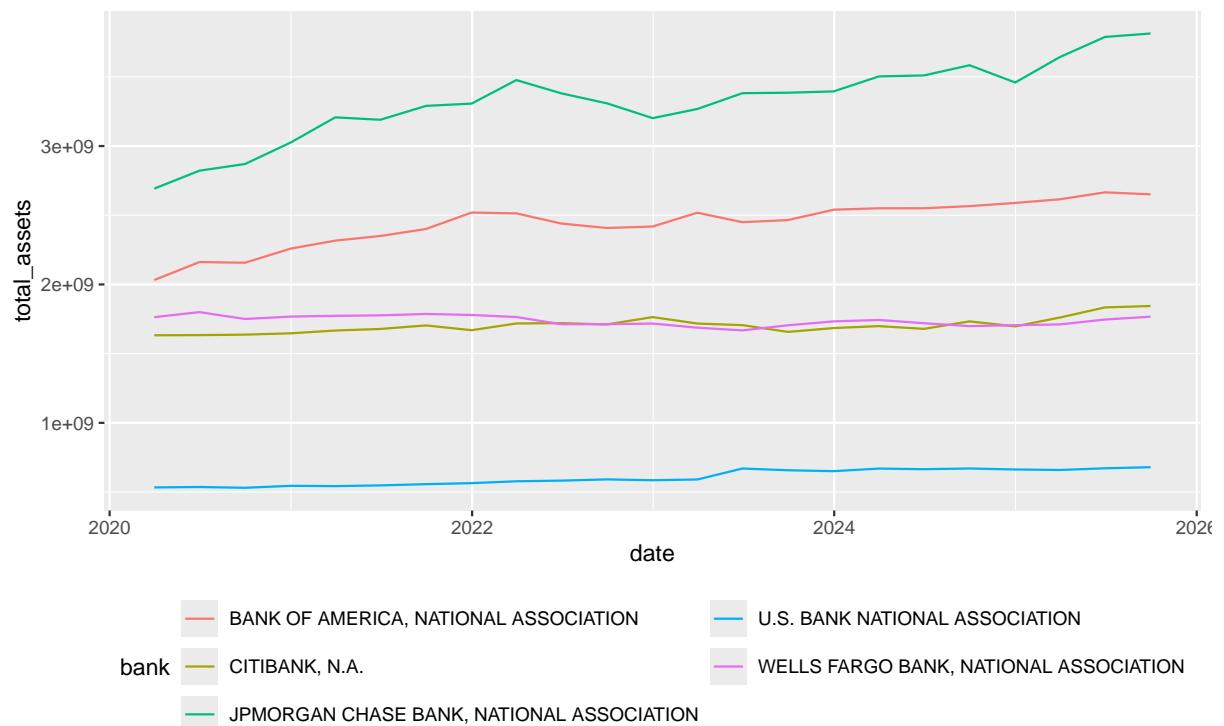


Figure 2: Total assets for top 5 banks: R version

1.4.1 Working with timestamps

Working with dates and times (**temporal data**) can be a lot more complicated than is generally appreciated. Broadly speaking we might think of temporal data as referring to points in time or *instants* or to time spans, which include durations, periods, and intervals.⁶

⁶See Wickham et al. (2023), p. 311-315, for discussion of time spans.

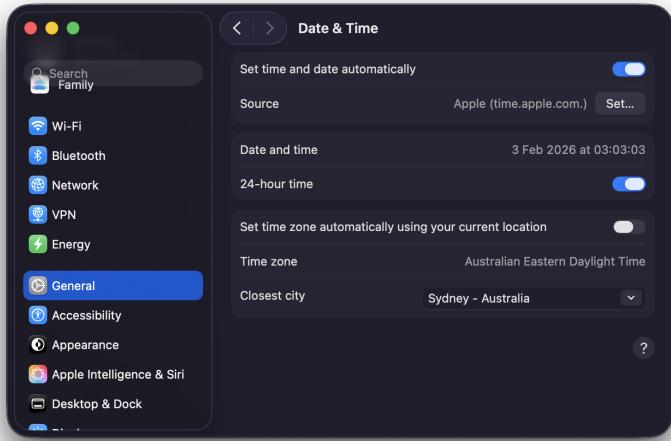


Figure 3: Setting my computer to a different time zone

```
Sys.timezone()
```

```
[1] "America/New_York"

"2026-01-13T10:13:21"

db <- dbConnect(duckdb::duckdb())

por <- ffiec_scan_pqS(db, "por")

por_default <-
  por |>
    filter(IDRSSD == 37, date == "2025-09-30") |>
    rename(dttm = last_date_time_submission_updated_on) |>
    mutate(dttm_text = as.character(dttm)) |>
    select(IDRSSD, date, dttm, dttm_text) |>
    collect()

dbDisconnect(db)

por_default

# A tibble: 1 x 4
```

```
IDRSSD date      dttm          dttm_text
<int> <date>    <dttm>        <chr>
1      37 2025-09-30 2026-01-13 15:13:21 2026-01-13 15:13:21+00
```

```
por_default$dttm[1]
```

```
[1] "2026-01-13 15:13:21 UTC"
```

```
tz <- "America/New_York"

db <- dbConnect(duckdb::duckdb(), timezone_out = tz)

rs <- dbExecute(db, "LOAD icu")
rs <- dbExecute(db, str_glue("SET TimeZone TO '{tz}'"))

por <- ffiec_scan_pqs(db, "por")
por_ny <-
  por |>
  filter(IDRSSD == 37, date == "2025-09-30") |>
  rename(dttm = last_date_time_submission_updated_on) |>
  mutate(dttm_text = as.character(dttm)) |>
  select(IDRSSD, date, dttm, dttm_text) |>
  collect()
```

```
por_ny
```

```
# A tibble: 1 x 4
IDRSSD date      dttm          dttm_text
<int> <date>    <dttm>        <chr>
1      37 2025-09-30 2026-01-13 10:13:21 2026-01-13 10:13:21-05
```

```
por_ny$dttm[1]
```

```
[1] "2026-01-13 10:13:21 EST"
```

```
western_states <- c("HI", "WA", "CA", "AK", "OR", "NV")

plot_data <-
  por |>
  rename(last_update = last_date_time_submission_updated_on) |>
```

```

mutate(
  q4 = quarter(date) == 4,
  offset = last_update - sql("last_update AT TIME ZONE 'UTC'"),
  offset = date_part("epoch", offset) / 3600,
  tzone = if_else(offset == -4, "EDT", "EST"),
  west = financial_institution_state %in% western_states,
  ref = sql(str_glue("TIMESTAMPTZ '2025-01-01 00:00:00 {tz}'")),
  sub_date = date_trunc('days', last_update)) |>
  mutate(time_adj = last_update - sub_date + ref) |>
  select(IDRSSD, date, last_update, time_adj, west, q4, offset, tzone) |>
  collect()

```

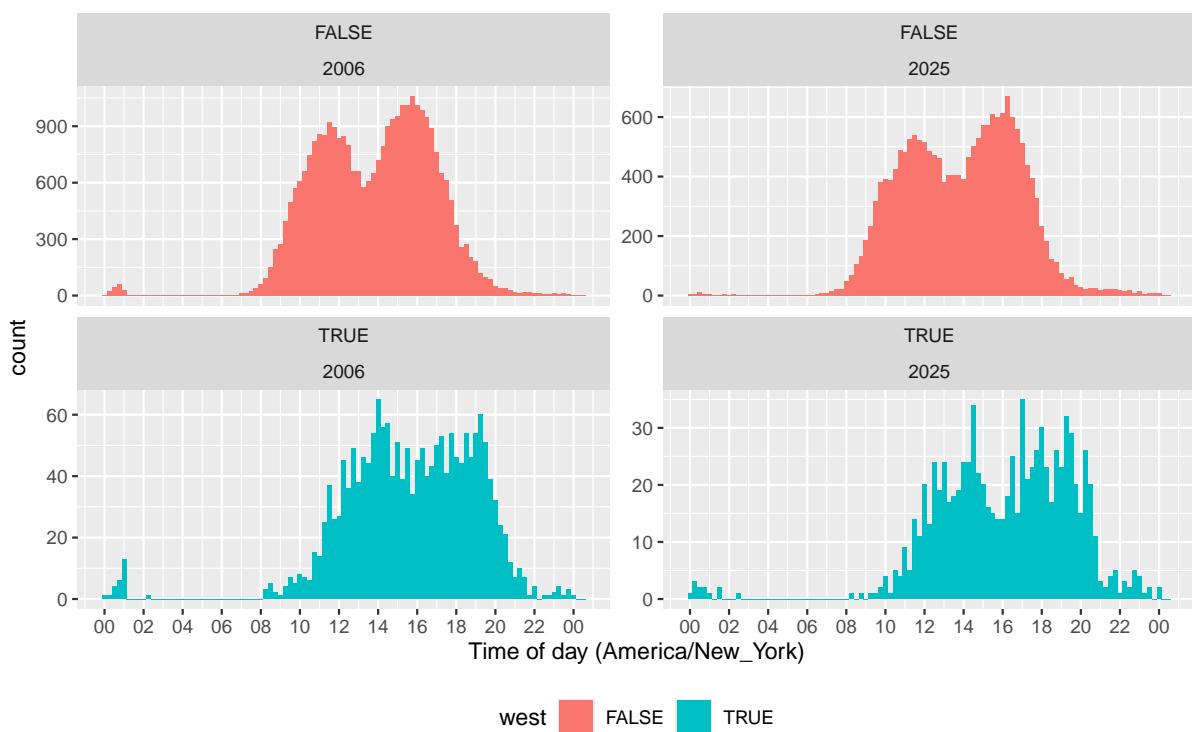


Figure 4: Submission times by year and region

1.5 Using the data with Python

```

from pathlib import Path
import polars as pl
import os

```

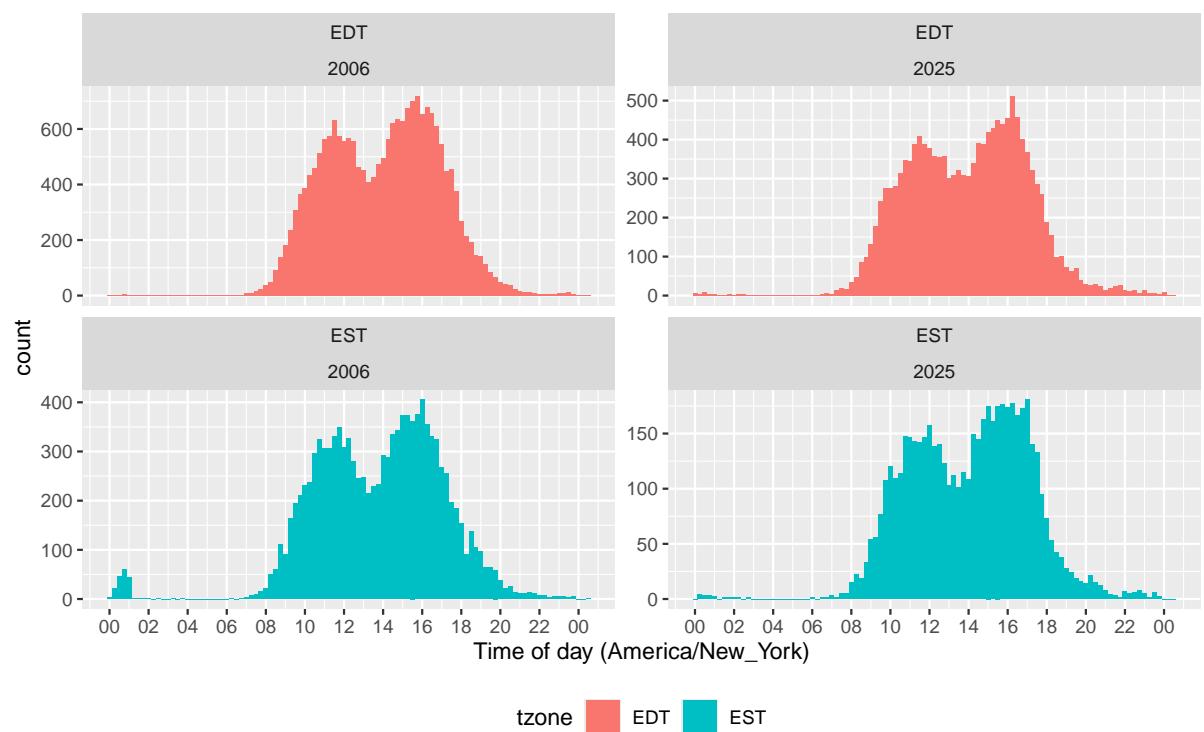


Figure 5: Submission times by year and US/Eastern time zone

```

def ffiec_scan_pqs(schedule=None, *,
                    schema="ffiec", data_dir=None):
    if data_dir is None:
        data_dir = Path(os.environ["DATA_DIR"]).expanduser()

    path = data_dir / schema if schema else data_dir

    if schedule is None:
        raise ValueError("You must supply `schedule`.")
    files = list(path.glob(f"{schedule}/*.parquet"))
    if not files:
        raise FileNotFoundError(
            f"No Parquet files found for schedule '{schedule}' in {path}"
        )

    return pl.concat([pl.scan_parquet(f) for f in files])

ffiec_float = ffiec_scan_pqs("ffiec_float")
por = ffiec_scan_pqs(schedule="por")

cols = ["RCFD2170", "RCON2170"]

bs_data = (
    ffiec_float
    .filter(pl.col("item").is_in(cols))
    .pivot(
        on = "item",
        on_columns = cols,
        index = ["IDRSSD", "date"],
        values = "value")
    .with_columns(
        total_assets = pl.coalesce(pl.col("RCFD2170"), pl.col("RCON2170"))
    )
)
bs_data.head(5).collect()

```

shape: (5, 5)

IDRSSD	date	RCFD2170	RCON2170	total_assets
---	---	---	---	---
i32	date	f64	f64	f64

864341	2014-12-31	null	23364.0	23364.0
2325882	2013-09-30	null	129814.0	129814.0
870650	2013-06-30	null	437650.0	437650.0
362856	2007-03-31	null	39746.0	39746.0
732758	2018-12-31	null	364805.0	364805.0

```

top_5 = (
    bs_data
    .filter(pl.col("date") == pl.date(2025, 9, 30))
    .sort("total_assets", descending=True)
    .with_row_index("ta_rank", offset=1)
    .filter(pl.col("ta_rank") <= 5)
)

top_5_names = (
    top_5
    .join(
        por.select(["IDRSSD", "date", "financial_institution_name"]),
        on=["IDRSSD", "date"],
        how="inner",
    )
    .sort("ta_rank")
    .select(["IDRSSD", "financial_institution_name", "ta_rank"])
    .rename({"financial_institution_name": "bank"})
)

bs_panel_data = bs_data.join(top_5_names, on="IDRSSD", how="inner")

pdf = (
    bs_panel_data
    .filter(pl.col("date") >= pl.date(2020, 1, 1))
    .collect()
    .to_pandas()
)

bank_names = {
    476810: "Citibank",
    504713: "US Bank",
    852218: "JPMorgan Chase",
    451965: "Wells Fargo",
}

```

```

480228: "Bank of America",
}

pdf["bank"] = pdf["IDRSSD"].map(bank_names)

```

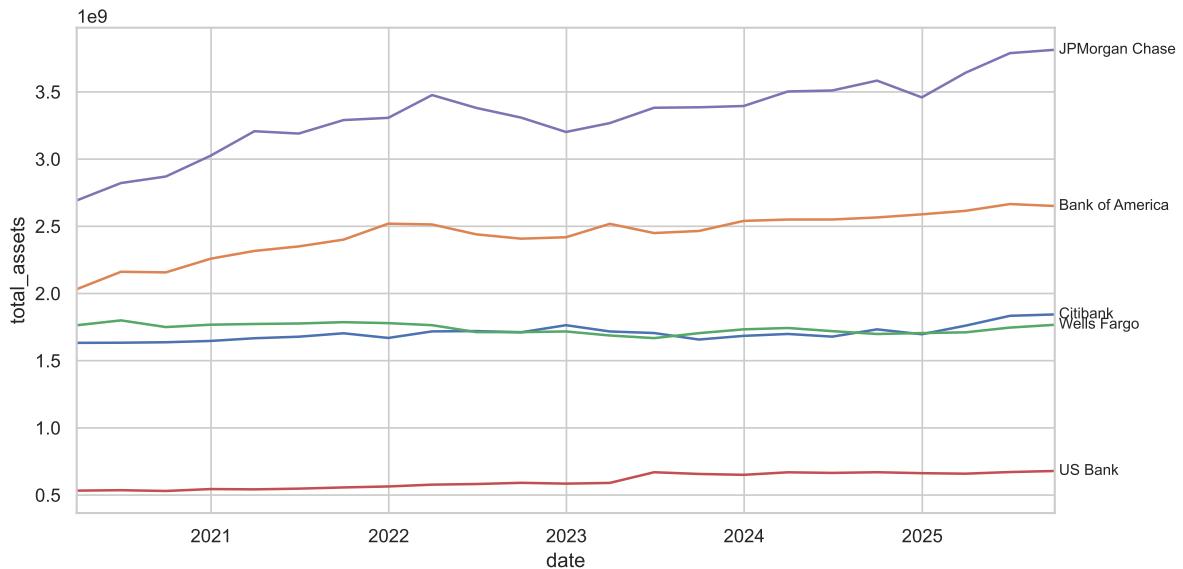


Figure 6: Total assets for top 5 banks: Python version

2 The boring details

2.1 Reading the data

Each quarter's zip file (`zipfile`) actually contains dozens of text files (`.txt`) in TSV ("tab-separated values") form. The TSV is a close relative of the CSV ("comma-separated values") and the principles applicable to one form apply to the other.

I have seen code that just imports from these individual files (what I call `inner_file`) in some location on the user's hard drive. This approach is predicated on the user having downloaded the zip files and unzipped them. While we have downloaded all the zip files—assuming you followed the steps outlined in Section 1.1—I don't want to be polluting my hard drive (or yours) with thousands of `.txt` files that won't be used after reading them once.

Instead, R allows me to do simply say:

```
con <- unzip(zipfile, inner_file)
```

The resulting `con` object is a temporary read-only connection to a single text file (`inner_file`) stored inside the zip file `zipfile`. The object allows R to stream the file's contents directly from the zip archive, line by line, without extracting it. Given `con`, the core function used to read the data into R has the following basic form, where `read_tsv()` comes from the `readr` package, part of the Tidyverse:

```
df <- read_tsv(  
  con,  
  col_names = cols,  
  col_types = colspec,  
  skip = skip,  
  quote = "",  
  na = c("", "CONF"),  
  progress = FALSE,  
  show_col_types = FALSE  
)
```

2.1.1 Handling embedded newlines and tabs

Experienced uses of the `readr` package might wince a little at the `quote = ""` argument above. What this means is that the data are not quoted.

Wickham et al. (2023, p. 101) points out that “sometimes strings in a CSV file contain commas. To prevent them from causing problems, they need to be surrounded by a quoting character, like `"` or `'`. By default, `read_csv()` assumes that the quoting character will be `"`.”

Adapting this to our context and expanding it slightly, I would say: “sometimes strings in a TSV file contain tabs (`\t`) and newline characters (`\n`).⁷ To prevent them from causing problems, they need to be surrounded by a quoting character, like `"` or `'`.” While this is a true statement, the TSV files provided on the FFIEC Bulk Data website are *not* quoted, which means that tabs and newlines characters **embedded** in strings *will* cause problems.

The approach taken by the `ffiec.pq` package is to attempt to read the data using a call like that above, which I term the “fast path” (in part because it is indeed fast). Before making that call, the code has already inspected the first row of the file to determine the column names (stored in `cols`) and used those column names to look up the appropriate type for each column (stored in `colspecs`). Any anomaly caused by embedded newlines or embedded tabs will almost certainly cause this first `read_tsv()` call to fail. But if there are no issues, then we pretty much have the

⁷Tabs and newlines are what are sometimes called **invisibles** because their presence is not apparent from viewing their usual representation as text (for example, a tab might look the same as a series of spaces). The `\t` and `\n` representations are quite standard ways of making these characters visible to humans.

data as we want them and can return `df` to the calling function.⁸ Fortunately, over 95% of files can be read successfully on the “fast path”.

It turns out that if the “fast path” read fails, the most likely culprit is **embedded newlines**. Let’s say the table we’re trying to read has seven columns and the text field that is the fourth field in the file contains, in some row of the data, an embedded newline, because the data submitted by the reporting financial institution contained `\n` in that field. Because `read_tsv()` processes the data line by line and lines are “delimited” by newline characters (`\n`), it will see the problematic line as terminating part way through the fourth column and, because `cols` tells `read_tsv()` to expect seven columns, `read_tsv()` will issue a warning.

When a warning occurs on the “fast path”, the read function in `ffiec.pq` moves to what I call (unimaginatively) the “slow path”. A “feature” (it turns out) of the TSV files provided on the FFIEC Bulk Data website is that each line ends with not just `\n`, but `\t\n`. This means we can assume that any `\n` not preceded by `\t` is an embedded newline, not a line-terminating endline.⁹ So I can read the data into the variable `txt` using `readLines()` and use a regular expression to replace embedded newlines with an alternative character. The alternative I use is a space and I use the `gsub()` function to achieve this: `gsub("(?<!\\t)\\\\n", " ", txt, perl = TRUE)` The regular expression here is `(?<!\\t)\\\\n` is equivalent to `(?<!\\t)\\n` in Perl or `r"(?<!\\t)\\n"` in Python.¹⁰ In words, the regular expression literally means “any newline character that is not immediately preceded by a tab” and the `gsub()` function will replace such characters with spaces (“ ”) because that is the second argument to `gsub()`.

This fix addresses almost all the problematic files. “Almost all” means “all but two”. The issue with the remaining two files is (as you might have guessed) **embedded tabs**. Unfortunately, there’s no easy “identify the embedded tabs and replace them” fix, because there’s no easy way to distinguish embedded tabs from delimiting tabs. However, we can detect the presence of embedded tabs because there will one too many tabs in any affected row.

For one of the two “bad” files, there is only one text field, so once we detect the presence of the embedded tab, we can assume that the extra tab belongs in that field: Problem solved. For the other “bad” file, there are several text fields and, while there is just one bad row, we cannot be sure which text field has the embedded tab. The `ffiec.pq` package just assumes that the embedded tab belongs in the last field and moves on. This means that the textual data for one row (i.e., one financial institution) for one schedule for one quarter cannot be guaranteed to be completely correct.¹¹ Such is life.

Even without addressing the issue, given that only two files are affected, it’s possible to measure the “damage” created by embedded tabs.

⁸In practice, there’s a little clean-up to be done before returning `df`, as I will explain shortly.

⁹Unfortunately, the `read_tsv()` function does not allow use to specify an alternative to the default for line-terminating characters.

¹⁰There are extra backslashes in the R version to specify that the backslashes represent backslashes to be passed along to `gsub()`, not special characters to be interpreted by R itself.

¹¹This is not a completely insoluble problem in that we could inspect the XBRL data to determine the correct form of the data in this case. This is “above my pay grade” in this setting. (I’m not being paid to do this!)

Let's look at the earlier file, which turns out to affect the Schedule RIE data for The Traders National Bank (IDRSSD of 490937) for June 2004.¹² Traders National Bank in Tennessee was "Tullahoma's second oldest bank", until changing its name to Wellworth Bank (seemingly after some mergers), and it listed total assets of \$117,335,000 in the affected Call Report.

Let's look at the textual data we have in our Parquet files for this case:¹³

```
ffiec_str <- ffiec_scan_pqs(db, schedule = "ffiec_str")

ffiec_str |>
  filter(IDRSSD == "490937", date == "2004-06-30") |>
  select(IDRSSD, item, value) |>
  filter(!is.na(value)) |>
  collect() |>
  system_time()

# A tibble: 6 x 3
  IDRSSD item      value
  <int> <chr>    <chr>
1 490937 TEXT3549 IENC SECURITIES
2 490937 TEXT3550 IENC CD'S
3 490937 TEXT4464 ATM Fees, Exam Fees, Dues & Chairitable Cont
4 490937 TEXT4467 Telephone, BanClub, Federal Res Fees, NSF and Other Loss
5 490937 TEXT4468 Courier, Audit Tax, Deff Comp, Other
6 490937 TEXT4469 ns Exp, Bus Dev, P
```

We can compare this with what we see in the Call Report in Figure 7, where we see that the value of TEXT4468 should be something like "Courier, Audit Tax, Deff Comp, Other\tns Exp, Bus Dev, P". The embedded tab has split this into "Courier, Audit Tax, Deff Comp, Other" for TEXT4468 and "ns Exp, Bus Dev, P" for TEXT4469, which should be NA. If the values for TEXT4468 and TEXT4469 for Traders National Bank in June 2004 are important to your analysis, you could fix this "by hand" easily enough.

Looking at the later file, there were embedded tabs in two rows of Schedule NARR for December 2022. I compared the values in the Parquet file with those in the Call Reports for the two affected

¹²Note that it would be somewhat faster to use `ffiec_text <- load_parquet(db, "ffiec_str_20040630", "ffiec")`, but this code doesn't take too long to run.

¹³Only the textual data will have embedded tabs and, because such data is arranged after numerical data, only text data will be affected by embedded tabs.

(TEXT4464) ATM Fees, Exam Fees, Dues & Chairitable Cont

(TEXT4467) Telephone, BanClub, Federal Res Fees, NSF and Other Loss

(TEXT4468) Courier, Audit Tax, Deff Comp, Other ns Exp, Bus Dev, P

Figure 7: Extract from June 2004 Call Report for The Traders National Bank

banks and the values in the Parquet file match perfectly.¹⁴ Because Schedule NARR (“Optional Narrative Statement Concerning the Amounts Reported in the Consolidated Reports of Condition and Income”) has just one text column (TEXT6980), the fix employed by the `ffiec.pq` package will work without issues.¹⁵

2.1.2 Handling missing-value sentinels

Users familiar with both `readr` and Call Report data might also have noticed the use of `na = c("", "CONF")` in the call to `read_tsv()` above. The default value for this function is `na = c("", "NA")` means that empty values and the characters NA are treated as missing values. As I saw no evidence that the export process for FFIEC Bulk Data files used "NA" to mark NA values, I elected not to treat "NA" as NA. However, a wrinkle is that the reporting firms sometimes populate text fields—but not numerical fields—with the value "NA".¹⁶ While the most sensible interpretation of such values is as NA, without further investigation it is difficult to be sure that "NA" is the canonical form in which firms reported NA values rather than "N/A" or "Not applicable" or some other variant.

This approach seems validated by the fact that I see the value "NR" in text fields of PDF versions of Call Reports and these values show up as empty values in the TSV files, suggesting that "NR", not "NA" is the FFIEC's canonical way of representing NA values in these files, while "NA" is literally the text value "NA", albeit perhaps one intended *by the reporting firm* to convey the idea of NA. Users of the FFIEC data created by the `ffiec.pq` package who wish to use textual data should be alert to the possibility that values in those fields may be intended by the reporting firm to convey the idea of NA, even if they are not treated as such by the FFIEC's process for creating the TSV files.

The other value in the `na` argument used above is "CONF", which denotes that the the reported value is confidential and therefore not publicly disclosed. Ideally, we might distinguish between

¹⁴See [here](#) for the gory details. Interestingly, the WRDS Call Report data have the [same issue](#) with the earlier case and have incorrect data for one of the banks in the latter case. This seems to confirm that WRDS uses the TSV data itself in creating its Call Report data sets.

¹⁵Recall that the “fix” assumes that embedded tab belongs in last available text column.

¹⁶If "NA" appeared in a numeric field, my code would report an error. As I detected no errors in importing the data, I know there are no such values.

NA, meaning “not reported by the firm to the FFIEC” or “not applicable to this firm” or things like that, from “CONF”, meaning the FFIEC has the value, but we do not. Unfortunately, the value “CONF” often appears in numeric fields and there is no simple way to ask `read_tsv()` to record the idea that “this value is confidential”, so I just read these in as NA.¹⁷

I say “no simple way” because there are probably workarounds that allow “CONF” to be distinguished from true NAs. For example, I could have chosen to have `read_tsv()` read all numeric fields as character fields and then convert the value CONF in such fields to a **sentinel value** such as Inf (R’s way of saying “infinity” or ∞).¹⁸ This would not be terribly difficult, but would have the unfortunate effect of surprising users of the data who (understandably) didn’t read the manual and starting finding that the mean values of some fields are Inf. Perhaps the best way to address this would allow the user of `ffiec.pq` to *choose* that behaviour as an option, but I did not implement this feature at this time.

In addition to these missing values, I discovered in working with the data that the FFIEC often used specific values as **sentinel values** for NA. For example, “0” is used for some fields, while “00000000” is used to mark dates as missing, and “12/31/9999 12:00:00 AM” is used for timestamps. I recoded such sentinel values as NA in each case.

3 The service-level agreement

Nothing from here up has been updated from original note

It is perhaps helpful to think of dividing the data science workflow into processes with different teams being responsible for the different processes. From this perspective, the *Curate* team manufactures data that are delivered the *Understand* team. While I won’t discuss transfer pricing (i.e., how much the *Understand* team needs to pay the *Curate* team for the data), we might consider the analogy of a **service-level agreement** between the two teams.

One template for a service-level agreement would specify that data from a particular source will be delivered to the *Understand* team with the following conditions:

1. The data will be presented as a set of tables in a modern storage format.
2. The division into tables will adhere to a pragmatic version of good database principles.
3. The **primary key** of each table will be identified and validated.
4. Each variable (column) of each table will be of the correct type.
5. There will be no manual steps that cannot be reproduced.
6. A process for updating the curated data will be established.
7. The entire process will be documented in some way.
8. Some process for version control of data will be maintained.

¹⁷This is the kind of situation where SAS’s approach to coding missing values would be helpful.

¹⁸In a sense, this would be doing the opposite of what the Python package pandas did in treating `np.NAN` as the way of expressing what later became `pd.NA`; I’d be using `Inf` to distinguish different kinds of missing values.

3.1 Storage format

In principle, the storage format should fairly minor detail determined by the needs of the *Understand* team. For example, if the *Understand* team works in Stata or Excel, then perhaps they will want the data in some kind of Stata format or as Excel files. However, I think it can be appropriate to push back on notions that data will be delivered in form that involves downgrading the data or otherwise compromises the process in a way that may ultimately add to the cost and complexity of the task for the *Curate* team. For example, “please send the final data as an Excel file attachment as a reply email” might be a request to be resisted because the process of converting to Excel can entail the degradation of data (e.g., time stamps or encoding of text).¹⁹ Instead it may be better to choose a more robust storage format and supply a script for turning that into a preferred format.

One storage format that I have used in the past would deliver data as tables in a (PostgreSQL) database. The *Understand* team could be given access data from a particular source organized as a **schema** in a database. Accessing the data in this form is easy for any modern software package. One virtue of this approach is that the data might be curated using, say, Python even though the client will analyse it using, say, Stata.²⁰

I chose to use parquet files for `ffiec.pq`.

3.2 Good database principles

I included the word “pragmatic” because I think it’s not necessary in most cases to get particularly fussy about **database normalization**. That said, it’s probably bad practice to succumb to requests for One Big Table that the *Understand* team might make. It is reasonable to impose some obligation to merge tables that are naturally different tables on the client *Understand* team.

In general, I think one wants to be fairly conservative in considering database principles with a data package. If the data are workable and make sense in the form they come in, then it may make most sense to keep them in that form.

3.3 Primary keys

In Gow (2026), I suggested that “the *Curate* team should communicate the primary key of each table to the *Understand* team. A primary key of a table will be a set of variables that can be used to uniquely identify a row in that table. In general a primary key will have no missing values. Part of data curation will be confirming that a proposed primary key is in fact a valid primary key.”

¹⁹I discuss some of the issues with Excel as a storage format below.

²⁰One project I worked on involved Python code analysing text and putting results in a PostgreSQL database and a couple of lines of code were sufficient for a co-author in a different city to load these data into Stata.

For all of the schedules, IDRSSD and date combine to make a valid primary key. To facilitate checking this, I include the function `ffiec_check_pq_keys()`, which checks that a proposed primary key valid for the table formed by combining the associated files for all quarters for a specified schedule. I first apply to it the Schedule RC table (`schedule = "rc"`):

We can see that `c("IDRSSD", "date")` is a valid primary key for Schedule RC because there are no rows in `null_violations` and no rows in `pk_violations` (checks for uniqueness) and `ok` is `TRUE`.

We can use `ffiec_list_pqs()` to identify all schedules and pass that to `ffiec_check_pq_keys()`:

As can be seen, IDRSSD and date form a primary key for all schedules (including the POR data).

3.4 Data types

In Gow (2026), I proposed that “each variable of each table should be of the correct type. For example, dates should be of type DATE, variables that only take integer values should be of INTEGER type. Date-times should generally be given with TIMESTAMP WITH TIME ZONE type. Logical columns should be supplied with type BOOLEAN.”²¹

This element is (to the best of my knowledge) satisfied with one exception. The Parquet format is a bit like the Model T Ford: it supports time zones, and you can use any time zone you want, so long as it is UTC.²² As discussed above, there is only one timestamp in the whole set-up, `last_date_time_submission_updated_on` on the POR files and I discussed this field above.

3.5 No manual steps

When data vendors are providing well-curated data sets, much about the curation process will be obscure to the user. This makes some sense, as the data curation process has elements of trade secrets. But often data will be supplied by vendors in an imperfect state and significant data curation will be performed by the *Curate* team working for or within the same organization as the *Understand* team.

Focusing on the case where the data curation process transforms an existing data set—say, one purchased from an outside vendor—into a curated data set in sense used here, there are a few ground rules regarding manual steps.

²¹Gow (2026) is referring to PostgreSQL types. The `ffiec.pq` package uses (logical) Parquet types DATE, INT32, `TIMESTAMP(isAdjustedToUTC = true)`, and BOOLEAN types, respectively for these types. Floating-point numbers are stored as FLOAT64 and strings as STRING.

²²Henry Ford famously said of the Model T that “any customer can have a car painted any color that he wants so long as it is black.” Strictly speaking, the Parquet format supports **timezone-aware timestamps**, but only as UTC instants, as other time zones are not supported.

“First, the original data files should not be modified in any way.” Correct. The `ffiec.pq` package does not modify the the FFIEC Bulk Data files after downloading them. I do make some corrections to the `item_name` variable in the `ffiec_items` package, but these “manual steps [are] extensively documented and applied in a transparent, automated fashion.” The code for these steps can be found on [the GitHub page](#) for the `ffiec.pq` package.

3.6 Documentation

“The process of curating the data should be documented sufficiently well that someone else could perform the curation steps should the need arise.” I regard that the `ffiec.pq` package do all the work of processing the data satisfies this requirement.

A important idea here is that the code for processing the data is documentation in its own right. Beyond that the document you are reading now is a form of documentation, as is the documentation in the `ffiec.pq` package.

3.7 Update process

If a new zip file appears on the FFIEC Bulk Data website, you can download it using the process outlined in Section 1.1. Just changing the `[:4]` to `[0]` and the script downloads the latest file.

Then run the following code and the data will be updated.

```
results <-
  ffiec_list_zips() |>
  filter(date == max(date)) |>
  select(zipfile) |>
  pull() |>
  ffiec_process() |>
  system_time()
```

```
user  system elapsed
13.975   1.806  12.019
```

```
results |> count(date, ok)
```

```
# A tibble: 1 x 3
  date      ok     n
  <date>    <lgl> <int>
1 2025-09-30 TRUE     39
```

3.8 Data version control

Welch (2019) argues that, to ensure that results can be reproduced, “the author should keep a private copy of the full data set with which the results were obtained.” This imposes a significant cost on the *Understand* team to maintain archives of data sets that may run to several gigabytes or more and it would seem much more efficient for these obligations to reside with the parties with the relevant expertise. Data version control is a knotty problem and one that even some large data providers don’t appear to have solutions for.

Given that I am delivering the Call Report data not as the data files, but as an R package along with instructions for obtaining the zip files from the FFIEC Bulk Data website. That said, if a user retains the downloaded zip files, the application of the `ffiec.pq` functions to process these into Parquet files should provide a high degree of reproducibility of the data for individual researcher.²³

For my own purposes, I achieve a modest level of data version control by using Dropbox, which offers the ability to restore previous versions of data files.

References

- Gow, I.D., 2026. [Data curation and the data science workflow](#).
- Gow, I.D., Ding, T., 2024. Empirical research in accounting: Tools and methods. Chapman & Hall/CRC. <https://doi.org/10.1201/9781003456230>
- Kashyap, A.K., Rajan, R., Stein, J.C., 2002. Banks as liquidity providers: An explanation for the co-existence of lending and deposit-taking. *Journal of Finance* 57, 33–63. <https://doi.org/10.1111/1540-6261.00415>
- Welch, I., 2019. Editorial: An opinionated FAQ. *Critical Finance Review* 8, 19–24. <https://doi.org/10.1561/104.00000077>
- Wickham, H., Çetinkaya-Rundel, M., Gromlund, G., 2023. [R for data science](#). O'Reilly Media, Sebastopol, CA.

²³Note that a researcher might need to use a specific version of the `ffiec.pq` package to achieve full reproducibility, but the `pak` package allows for that.