Working with Form AP data

Ian D. Gow

16 May 2024

The United States Public Company Accounting Oversight Board (PCAOB) requires registered public accounting firms to file on Form AP for each audit report.[1] Data from these filings are made available by the PCAOB as part of its AuditorSearch page. The PCAOB describes AuditorSearch as "a public database of engagement partners and audit firms participating in audits of U.S. public companies."

These Form AP data recently featured in a tongue-in-cheek *Financial Times* article by George Steer that showed that the auditor Ben Borgers had used 14 different names—including the name "Ben F orgers"—on Form AP filings. The analysis of the Form AP data in the FT article had been conducted by independent researcher Stephen Walker and below I show how to reproduce Stephen's analysis.

The purpose of this note is to demonstrate how one can work with Form AP data using R. Working with these data is a nice exercise for developing some basic data science skills. All the software needed to produce the analyses in this note is free and can be installed in less than ten minutes. So it should be easy to work through everything I do here.[2] A fast-paced hands-on tutorial for R and some of the ideas here is provided in Chapter 2 of *Empirical Research in Accounting: Tools and Methods* here.[3]

In writing this note, I use the packages listed below.[4] This note was written using Quarto and compiled with RStudio, an integrated development environment (IDE) for working with R. The source code for this note is available here.

```
library(tidyverse)
library(DBI)
library(dbplyr)
```

[1]Discussion of the history of the PCAOB is provided by Gow and Kells (2018).
[2]An internet connection is required to download the data and software.
[3]*Empirical Research in Accounting: Tools and Methods* will be published in print form by CRC Press later in 2024 and will remain free online after publication.
[4]Execute `install.packages(c("tidyverse", "DBI", "dbplyr", "duckdb", "jsonlite", "arrow"))` within R to install all the packages you need to run the code in this note.

```r
library(jsonlite)
library(arrow)
```

Getting Form AP data

The Form AP data are made available by the PCAOB as a single compressed comma-separated value (CSV) file. The PCAOB requires use to provide an email address when downloading data using a script and we can set this using the `HTTPUserAgent` option as in the R code below.[5]

```r
options(HTTPUserAgent = "your_name@some_email.com")
```

Once we have set `HTTPUserAgent`, we can run the script available here by executing the following line in R. This script takes about 10 seconds to run for me, but it may take longer if you have a slower connection to the PCAOB website than I do.

```r
source("https://raw.githubusercontent.com/iangow/notes/main/get_form_aps.R")
```

This script downloads and processes the Form AP data into a parquet file. The parquet format is described in *R for Data Science* (Wickham, Çetinkaya-Rundel, and Grolemund 2023, 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 *R for Data Science* and a guide to creating a parquet data repository are provided here.

For reasons we will explain below, we use a DuckDB database connection to work with the data. We start by creating that connection and loading the parquet file we just created.

```r
db <- dbConnect(duckdb::duckdb())
form_aps <- tbl(db, sql("SELECT * FROM 'data/form_aps.parquet'"))
```

For the most part, the data in the Form AP file are straightforward. However, two sets of fields create more work than the others.

JSON data

The first set of fields in the Form AP data that require more effort are the three stored as JSON:

- `audit_fund_series`
- `audit_not_divided_percent_information`

[5]Use your actual email address here.

- `audit_divided_information`

JSON stands for "Javascript object notation" and is a common form of data on the internet. For example, many web APIs return data in JSON form. More on JSON can be found in Chapter 23 of *R for Data Science*.

The script we used to create `form_aps.parquet` does not process the JSON data; these are retained as simple text date. However, we could easily process these columns if we wanted to do so. While DuckDB has native functions for processing JSON, these are mostly predicated on JSON data being found in files, not in a single field of a table. In this case it is easier to bring the data into R (using the `collect()` function) and use the `parse_json()` function from the `jsonlite` library to parse these fields.

Here we illustrate one approach using the column `audit_fund_series`. We use `rowwise()` because `parse_json()` is design to work with one value at a time and `list()` to allow multiple values to be stored in a single row. To understand what the `unnest_longer()` and `unnest_wider()` functions are doing, it may be helpful to examine the input to and output from those functions. This can be achieved by selecting and executing the code preceding the pipes (i.e., `|>`) before and after the respective functions. More about pipes can be found in Chapter 4 of *R for Data Science*.

```
fund_series <-
  form_aps |>
  select(form_filing_id, audit_fund_series) |>
  filter(!is.na(audit_fund_series)) |>
  collect() |>
  rowwise() |>
  mutate(json = list(parse_json(audit_fund_series))) |>
  unnest_longer(json) |>
  unnest_wider(json) |>
  select(-audit_fund_series)
```

In effect, we process the data in this column into a new table, `fund_series`. This `fund_series` table could be copied to DuckDB and joined with `form_aps` as needed.

```
fund_series
```

```
# A tibble: 112,406 x 3
   form_filing_id FundSeriesID FundName
            <int> <chr>        <chr>
 1             27 S000000360   General Treasury Securities Money Market Fu~
 2             27 S000000359   General Government Securities Money Market ~
 3             28 S000000118   GENERAL CALIFORNIA MUNICIPAL MONEY MARKET F~
 4             29 S000000123   GENERAL NEW YORK AMT-FREE MUNICIPAL MONEY M~
 5             32 S000000121   General Municipal Money Market Funds, Inc.
 6             33 S000009025   Deutsche Municipal Income Trust
 7             35 S000000093   DREYFUS NEW YORK AMT-FREE MUNICIPAL BOND FU~
 8             37 S000006150   Deutsche Small Cap Value Fund
 9             37 S000006149   Deutsche Mid Cap Value Fund
10             37 S000006144   Deutsche Large Cap Value Fund
# i 112,396 more rows
```

Dates and times

The second set of columns that requires careful handling are the five columns containing dates or date-times.

- `audit_report_date`: Date of the audit report (`mm/dd/yyyy`)
- `fiscal_period_end_date`: The end date of the most recent period's financial statements identified in the audit report (`mm/dd/yyyy`)
- `signed_date`: Date of typed and manual signatures (`mm/dd/yyyy`)
- `audit_dual_date`: The date of the dual-date information, if any.
- `filing_date`: Date and time that the Form AP was filed.

Some clean-up is required for the first three fields, as these are not always in the indicated `mm/dd/yyyy` format. The `audit_dual_date` field can contain multiple dates and requires special handling. Finally, `filing_date` contains both a date and a time and I process it accordingly. According to the PCAOB-provided data dictionary, "dates and times are provided based on the U.S. Eastern time zone" and I process these fields using that assumption.

We can see from the following code that the Form AP data is updated frequently (on at least a daily basis) and with date-times for `filing_date` as recent as a couple of hours ago as I write.[6]

```
form_aps |>
  select(ends_with("date")) |>
  select(-audit_dual_date) |>
  summarize(across(everything(), \(x) max(x, na.rm = TRUE))) |>
  collect()
```

```
# A tibble: 1 x 4
  audit_report_date fiscal_period_end_date signed_date filing_date
  <date>            <date>                 <date>      <dttm>
1 2024-05-15        2024-05-13             2024-05-16  2024-05-16 13:30:37
```

[6]You will see more recent dates if you download after the date of writing.

The anonymous function is needed to exclude missing values (`NA` in R). More on missing values is found in Chapter 18 of *R for Data Science*.

Note that the `unnest()` function is needed to analyse `audit_dual_date`, as `audit_dual_date` contains a list of dates. More on list-columns can be found in Chapter 23 of *R for Data Science*.

```
form_aps |>
  select(audit_dual_date) |>
  mutate(audit_dual_date = unnest(audit_dual_date)) |>
  filter(!is.na(audit_dual_date)) |>
  summarize(max = max(audit_dual_date, na.rm = TRUE)) |>
  collect()
```

```
# A tibble: 1 x 1
  max
  <date>
1 2024-05-14
```

Partner names

We now combine the first, middle, and last names of each partner into a single name. This step is facilitated by our use of DuckDB, as missing values (e.g., cases where partners have no middle name) are quietly ignored.[7]

```
form_aps_names <-
  form_aps |>
  mutate(engagement_partner_name =
           str_c(engagement_partner_first_name,
                 engagement_partner_middle_name,
                 engagement_partner_last_name, sep = " ")) |>
  select(form_filing_id, engagement_partner_id, engagement_partner_name)
```

Note that `dplyr` is quietly translating our code into SQL, as can be seen by applying `show_query()` to `form_aps_names`. Here `str_c()` is translated into the SQL function `concat_ws()` (see here for details).

```
form_aps_names |>
  show_query()
```

[7]If we working with `dplyr` and native data frames, missing values would be "infectious" with `str_c()`. Such infectiousness is often the desired behaviour, but in this case, we want to quietly discard missing components of names. That is we want `str_c("Ian", NA, "Gow")` to be `"Ian Gow"` rather than the `NA` it would be if we used `str_c()` with native R data. More on missing values can be found in Chapter 18 of *R for Data Science*.

```
<SQL>
 SELECT form_filing_id, engagement_partner_id, CONCAT_WS(' ',
    engagement_partner_first_name, engagement_partner_middle_name,
    engagement_partner_last_name) AS engagement_partner_name FROM (SELECT * FROM
    'data/form_aps.parquet') q01
```

SQL is a specialized language for manipulating and retrieving tabular data used by almost all modern database systems. More on SQL is provided in an appendix to *Empirical Research in Accounting: Tools and Methods* and in Chapter 21 of *R for Data Science*.

In Table 1, we show the number of different names associated with a sample of engagement partners. While we could provide a table the number of names associated with each value of `engagement_partner_id`, `engagement_partner_id` has no real meaning for readers. Instead, for each value of `engagement_partner_id`, we identify a unique name (the most frequent value of `engagement_partner_name`) that we use for this purpose and store this value in `most_common_names`.[8]

```
most_common_names <-
  form_aps_names |>
  count(engagement_partner_id, engagement_partner_name, name = "n_forms") |>
  group_by(engagement_partner_id) |>
  window_order(desc(n_forms)) |>
  filter(row_number() == 1) |>
  ungroup() |>
  select(engagement_partner_id, engagement_partner_name)
```

We count the number of unique names for each `engagement_partner_id` using `n_distinct()` and then merge the result with `most_common_names` to create the data frame `names_df` that we can use to produce tables and figures.

```
names_df <-
  form_aps_names |>
  group_by(engagement_partner_id) |>
  summarize(n_names = n_distinct(engagement_partner_name)) |>
  inner_join(most_common_names, by = "engagement_partner_id")
```

We first produce Table 1, where we can already see that Ben F Borgers is an outlier.

[8] We use `group_by(engagement_partner_id)` then '`window_order(desc(n_forms))` to organize the data in descending order of the frequency a name appears for a given `engagement_partner_id` and then `filter(row_number() == 1)` to pick off the first row for each value of `engagement_partner_id`.

```r
names_df |>
  filter(n_names >= 6) |>
  select(engagement_partner_name, n_names) |>
  arrange(desc(n_names))
```

Table 1: Auditors with most reported spellings

engagement_partner_name	n_names
Ben F Borgers	14
OLAYINKA TEMITOPE OYEBOLA	9
Thomas Michael O'Neal	8
Yong Yun Lee	7
Louis V Esposito	7
Kristofer Heaton	6
Derek Webb	6
Corey Eric Fischer	6
SAU JONG LIM	6
Patrick Wong	6
Richard J Fleischman	6
Jaslyn Huynh	6

We next examine the various names under which Ben F Borgers has filed in Table 2. To be fair to "Ben F orgers", the vast majority of filings by Ben F Borgers use the name "Ben F Borgers" and most of the rest use the name "Ben Borgers".

```r
form_aps_names |>
  filter(engagement_partner_id == "504100001") |>
  count(engagement_partner_name) |>
  mutate(perc = round(n / sum(n, na.rm = TRUE) * 100, 2)) |>
  arrange(desc(n))
```

Table 2: The many names of Ben F orgers

engagement_partner_name	n	perc
Ben F Borgers	864	91.82
Ben Borgers	61	6.48
Benjamin Borgers	3	0.32
Borgers F Ben	2	0.21
Ben F Borger	2	0.21
Borgers Ben	1	0.11

Table 2: The many names of Ben F orgers

engagement_partner_name	n	perc
Ben F Vonesh	1	0.11
Ben F orgers	1	0.11
Blake F Borgers	1	0.11
Ben F Brgers	1	0.11
Ben Borges	1	0.11
Borges F Ben	1	0.11
Ben f Borgers	1	0.11
Ben F Brogers	1	0.11

I next produce Figure 1, which is almost identical to the plot in the FT article cited above.

```
names_df |>
  count(n_names, name = "count") |>
  ggplot(aes(x = n_names, y = count)) +
  geom_col()
```

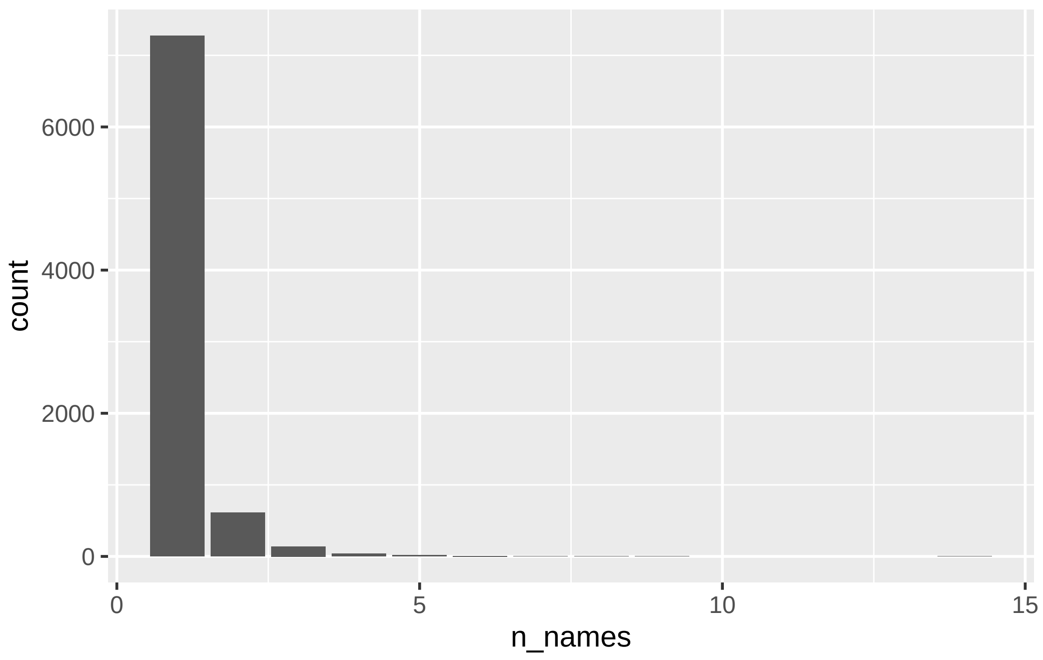

Figure 1: Distribution of auditors by number of reported spellings

A weakness of Figure 1 is that it is difficult to discern any information for values of `n_names` greater than 5. One approach to address this is to use a log-transformation. Figure 2 depicts the result of transforming the y-axis using $log(1 + y)$.

```r
names_df |>
  count(n_names, name = "count") |>
  ggplot(aes(x = n_names, y = count)) +
  geom_col() +
  scale_y_continuous(transform = "log1p")
```

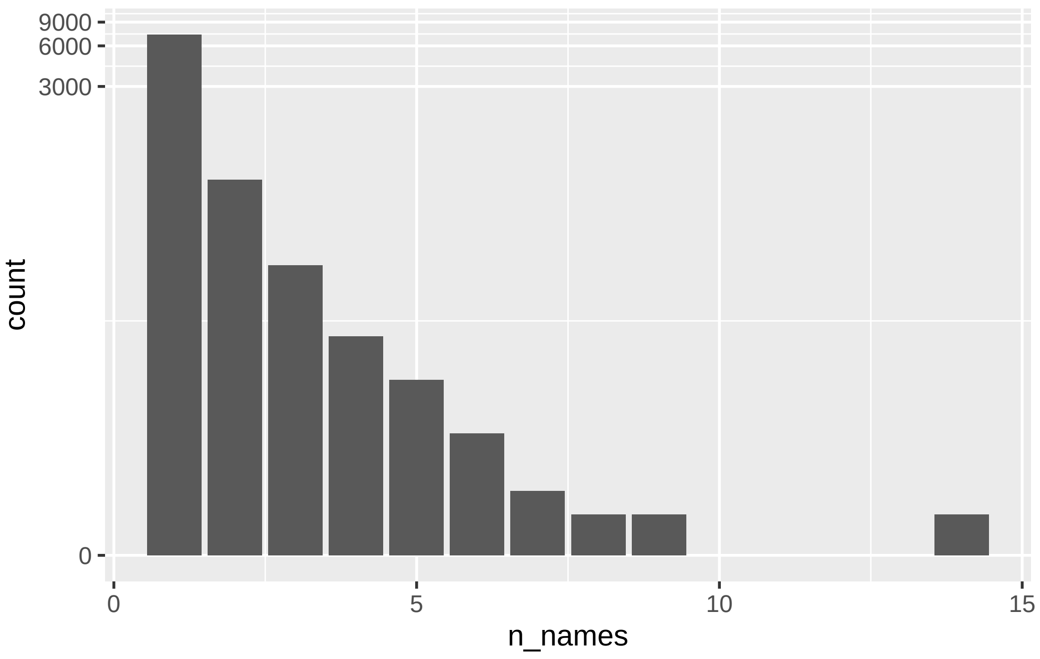

Figure 2: Distribution of auditors by number of reported spellings ($log(1 + y)$)

Busy auditors

One possible red flag with the case of Ben F Borgers is the sheer number of audit engagements he has been involved in. Is Borgers an outlier in this regard too?

```r
form_aps |>
  group_by(engagement_partner_id) |>
  summarize(n_audits = n(), .groups = "drop") |>
  inner_join(most_common_names, by = "engagement_partner_id") |>
  arrange(desc(n_audits)) |>
  select(engagement_partner_name, n_audits) |>
  collect(n = 10)
```

Table 3: Auditors with the most audits

engagement_partner_name	n_audits
Daniel Timothy Rooney	2155
Yossi Daniel Jayinski	1698
Paul Michael Gallagher	1398
Lawrence Raymond Depp	1311
Joseph D'Introno	1278
Amy Jo Huber	1164
Frank Michael Mleko	991
Kristine Marie Obrecht	987
Robert Allen Achtstatter	948
Ben F Borgers	941

From Table 3, we see that Borgers barely cracks the top ten. What's going on here?

In Table 4, I add information about the value of `audit_report_type`. It turns out that the "top auditors" are generally auditing investment companies. It seems plausible that such audits are quite perfunctory and a single engagement covers many companies (e.g., audits of ETFs or mutual funds).

```
form_aps |>
  mutate(audit_report_type = str_replace(audit_report_type, ",.*$", "")) |>
  group_by(engagement_partner_id, audit_report_type) |>
  summarize(n_audits = n(), .groups = "drop") |>
  inner_join(most_common_names, by = "engagement_partner_id") |>
  arrange(desc(n_audits)) |>
  select(engagement_partner_name, audit_report_type, n_audits) |>
  collect(n = 10)
```

Table 4: Auditors with the most audits by type

engagement_partner_name	audit_report_type	n_audits
Daniel Timothy Rooney	Investment Company	2155
Yossi Daniel Jayinski	Investment Company	1698
Paul Michael Gallagher	Investment Company	1396
Lawrence Raymond Depp	Investment Company	1311
Joseph D'Introno	Investment Company	1266
Amy Jo Huber	Investment Company	1164
Kristine Marie Obrecht	Investment Company	981
Frank Michael Mleko	Investment Company	979

Table 4: Auditors with the most audits by type

engagement_partner_name	audit_report_type	n_audits
Robert Allen Achtstatter	Investment Company	943
Ben F Borgers	Issuer	941

So, in Table 5, I exclude audits of investment companies and there it can be seen that Borgers is once again an outlier.

```r
form_aps |>
  mutate(audit_report_type = str_replace(audit_report_type, ",.*$", ""),
         firm_name = str_replace(firm_name, ",.*$", "")) |>
  filter(audit_report_type != "Investment Company") |>
  group_by(engagement_partner_id, firm_name, audit_report_type) |>
  summarize(n_audits = n(), .groups = "drop") |>
  inner_join(most_common_names, by = "engagement_partner_id") |>
  arrange(desc(n_audits)) |>
  select(engagement_partner_name, firm_name, audit_report_type, n_audits) |>
  collect(n = 10)
```

Table 5: Auditors with the most audits (excluding investment companies)

engagement_partner_name	firm_name	audit_report_type	n_audits
Ben F Borgers	B F Borgers CPA PC	Issuer	941
Edward Francis Hackert	Marcum LLP	Issuer	502
Steven Vertucci	MaloneBailey	Issuer	349
Natalie Verbanac	Marcum LLP	Issuer	344
Carl John Scheuten	WithumSmith+Brown	Issuer	307
Michael Bruce Tenny	WithumSmith+Brown	Issuer	305
Jay Christopher Shepulski	WithumSmith+Brown	Issuer	298
Mark John Deters	WithumSmith+Brown	Issuer	291
Marc Howard Silverman	WithumSmith+Brown	Issuer	267
John Edward Klenner	Marcum LLP	Issuer	202

Trump Media's new auditor

According to the *Wall Street Journal*, Semple, Marchal & Cooper was appointed as the new financial auditor for Trump Media & Technology Group early in May 2024.

We can search `form_aps` to identify the `firm_id` for this new audit firm.

```
djt_new_auditor <-
  form_aps |>
  filter(str_detect(firm_name, "^Semple")) |>
  distinct(firm_id, firm_name)

djt_new_auditor |>
  collect()
```

```
# A tibble: 1 x 2
  firm_id firm_name
    <int> <chr>
1     178 Semple, Marchal & Cooper, LLP
```

Table 6 suggests that Semple, Marchal & Cooper, LLP, has a much more manageable client roster than Ben F Borgers had.

```
form_aps |>
  semi_join(djt_new_auditor, join_by(firm_id, firm_name)) |>
  mutate(fiscal_year = year(fiscal_period_end_date)) |>
  group_by(issuer_name) |>
  summarize(n_audits = n(),
            first_year = min(fiscal_year, na.rm = TRUE),
            last_year = max(fiscal_year, na.rm = TRUE)) |>
  arrange(desc(last_year), desc(n_audits)) |>
  collect()
```

Table 6: Audit clients of Semple, Marchal & Cooper, LLP

issuer_name	n_audits	first_year	last_year
Quest Resource Holding Corp.	5	2019	2023
CISO Global, Inc.	1	2023	2023
Cerberus Cyber Sentinel Corp.	5	2018	2022
Adamas One Corp.	2	2021	2022
RVeloCITY, Inc.	2	2021	2022
Snoogoo Corp.	2	2016	2021
Item 9 Labs Corp.	2	2019	2020
Adamas One Corp	1	2020	2020
Quest Resource Holding Corp	3	2016	2018
Abtech Holdings, Inc.	2	2016	2017
El Capitan Precious Metals, Inc.	1	2017	2017
Modern Round Entertainment Corporation	1	2016	2016

Having completed our analysis, we can disconnect from our in-memory DuckDB database.

```r
dbDisconnect(db)
```

References

Gow, Ian D., and Stuart S. Kells. 2018. *The Big Four: The Curious Past and Perilous Future of the Global Accounting Monopoly*. Berrett-Koehler Publishers. https://books.google.com/books?id=VuReDwAAQBAJ.

Wickham, Hadley, Mine Çetinkaya-Rundel, and Garrett Grolemund. 2023. *R for Data Science*. Sebastopol, CA: O'Reilly Media. https://books.google.com/books?id=TiLEEAAAQBAJ.