The WRDS PostgreSQL database

An example-based approach to doing research with R

Ian D. Gow

Contents

Chapter 1

Introduction

In recent years, we have heard much about the explosion of "big data" and the emergence of the data scientist. A casual perusal of the shelves of the section of the bookstore dedicated to topics related to data science suggests that data science is often done using R and Python.

In contrast, many researchers in finance and accounting tend to use the old standard of SAS, with Stata being a relative newcomer. This is particularly the case when it comes to data on the Wharton Research Data Services (WRDS) platform. First, the default format for WRDS data is the SAS binary format (`.sas7bdat`). Second, example programs provided by WRDS for dealing with CRSP, Compustat and IBES are usually provided as SAS code.

While there are many resources for learning R, for an academic researcher (*with a WRDS account*) in accounting, finance, and adjacent fields, this book offers an introduction using familiar data with very little set-up cost. Apart from the requirement for a WRDS account, the software used in this book is open-source, free, available on all major platforms, and easy to set up.

One challenge in learning R is the plethora of packages and inconsistent approaches across many of these. To minimize this, this book focuses on the "tidyverse" as the core set of packages for analysis. As discussed on its website, "the tidyverse is a collection of R packages that share common philosophies and are designed to work together. ... If you are new to the tidyverse, the best place to learn the complete philsophy and how everything fits together is the R for data science book."

1.1 PostgreSQL on WRDS

According to the official website, "PostgreSQL is a powerful, open source object-relational database system." Similar systems include MySQL and Oracle. If

you have used `PROC SQL` in SAS, then you can think of PostgreSQL as a system where SQL is the native language.[1] The critical thing for the purposes of this book is that WRDS has made available a PostgreSQL database server that gives you access to all the WRDS data you would otherwise access via SAS on WRDS's servers.

1.2 Setting up

To run the code in this book you will need the following:

1. **A WRDS ID.**
2. **R**: "R is a free software environment for statistical computing and graphics."[2] R has becaome *the* standard statistical programming language in many areas of statistics and is gaining popularity in other areas. It runs on all major platforms and the easiest way to get R is from this website.
3. **RStudio**: While not strictly necessary, if you are new to R, I recommend that you start with RStudio. "RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management."[3]
4. **Required R packages.** After installing R and RStudio, open RStudio, and type the following into the "Console" pane, then press enter:

```
install.packages(c("tidyverse", "RPostgres"))
```

In the examples that follow, I will assume that you have the certain enviroment variables set up. This can be done within RStudio by typing code like the folloowing into the Console pane:

```
Sys.setenv(PGHOST="wrds-pgdata.wharton.upenn.edu", PGPORT=9737L)
Sys.setenv(PGDATABASE="wrds", PGUSER="your_wrds_id")
```

Now, let's get started.

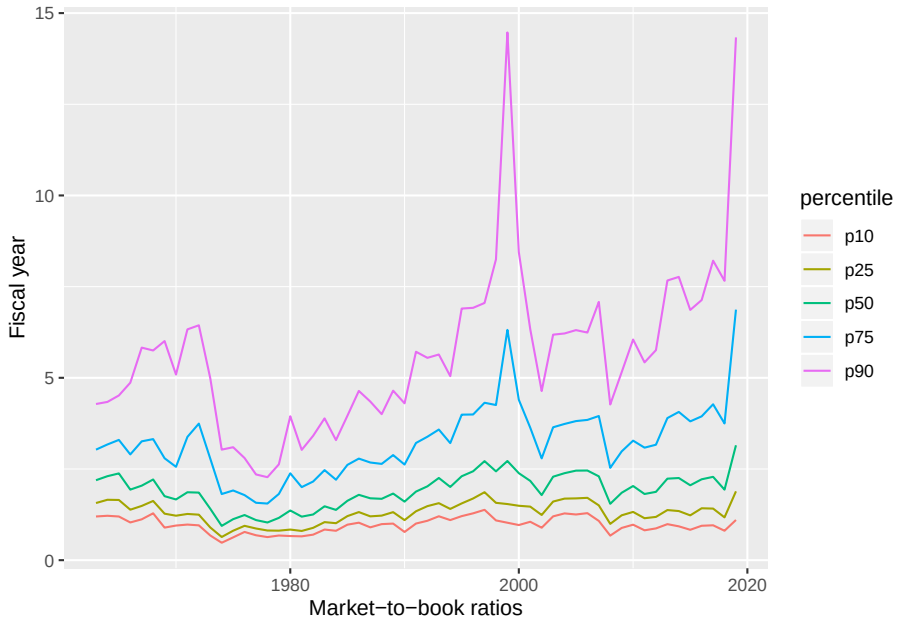
[1] And much of what you know from `PROC SQL` will carry over to PostgreSQL.
[2] See here
[3] See here.

Chapter 2

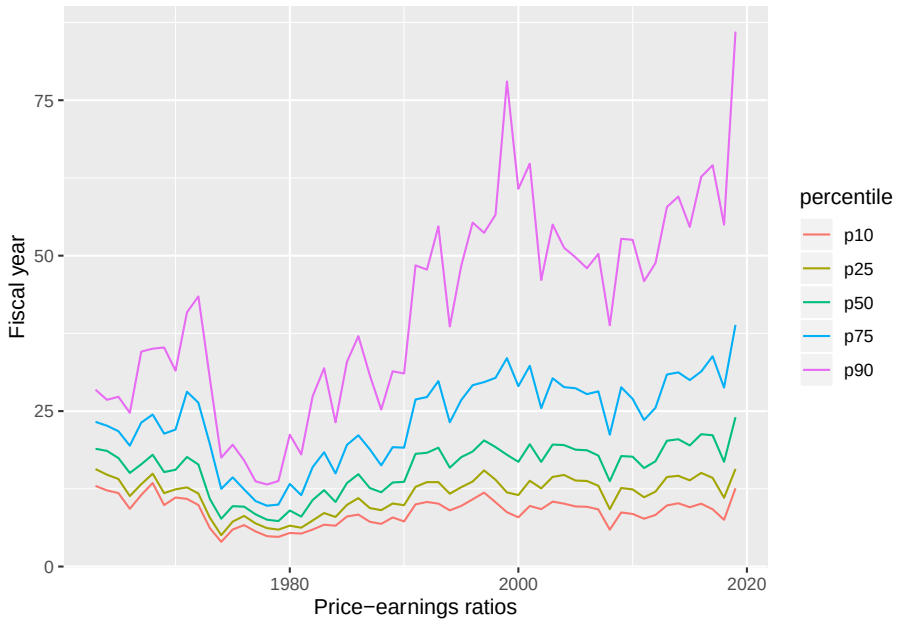
Performance over time

```
library(dplyr, warn.conflicts = FALSE)
library(tidyr)
library(DBI)

pg <- dbConnect(RPostgres::Postgres(), bigint = "integer")

funda <- tbl(pg, sql("SELECT * FROM comp.funda"))

comp <-
    funda %>%
    filter(indfmt =='INDL', datafmt =='STD', popsrc=='D', consol=='C') %>%
    mutate(mktcap = prcc_f*csho,
           mb = if_else(ceq > 0, prcc_f*csho/ceq, NA),
           pe = if_else(epspi >0, prcc_f/epspi, NA)) %>%
    filter(mktcap > 200, fyear>=1963) %>%
    select(gvkey, fyear, mktcap, mb, pe)

get_quantiles <- function(df, var, quantiles) {
  var_str <- substitute(var)
  perc_sql = paste0("percentile_cont(quantiles) ",
                    "WITHIN GROUP (ORDER BY ", var_str, ")")
  quantile_sql = paste0("ARRAY[", paste(quantiles, collapse = ", "), "]")
  df %>%
    mutate(quantiles = sql(quantile_sql)) %>%
    group_by(fyear, quantiles) %>%
    summarize(value = sql(perc_sql)) %>%
    mutate(percentile = unnest(quantiles)*100,
           value = unnest(value)) %>%
    select(-quantiles) %>%
```

```
    ungroup() %>%
    collect() %>%
    mutate(percentile = paste0("p", percentile),
           var = as.character(quote(var_str)))
}



quantiles <- c(0.1, 0.25, 0.5, 0.75, 0.9)


library(ggplot2)
get_quantiles(comp, mb, quantiles) %>%
  ggplot(aes(x = fyear, y = value, color = percentile)) +
  geom_line() +
  xlab("Market-to-book ratios") +
  ylab("Fiscal year")
```




```
get_quantiles(comp, pe, quantiles) %>%
  ggplot(aes(x = fyear, y = value, color = percentile)) +
  geom_line() +
  xlab("Price-earnings ratios") +
  ylab("Fiscal year")
```




```
rs <- dbDisconnect(pg)
```

Chapter 3

Variance-ratio test of asset prices

Here we follow BBK in replicating part of Lo and MacKinlay (1988).[1]

In the relevant part of the paper, Lo and MacKinlay (1988) use weekly returns on the equal-weighted CRSP index going from one Wednesday to the next over a fixed period. One issue I discovered in doing this is that in 1968 there was a "paperwork crisis" that led to markets being closed on Wednesdays for several months and thus gaps in the sequence of index data on `crsp.dsic`. For those Wednesdays, I want to select the index on the next day (i.e., Thursday) as that seems to be what Lo and MacKinlay (1988). But how to do this? The easiest way is probably to get the full sequence of Wednesdays over the relevant time period and use an SQL `LEFT JOIN` with `crsp.dsic`. But how to get those dates? PostgreSQL has convenient functions for generating sequences, including sequences of dates.

```
library(dplyr, warn.conflicts = FALSE)
library(tidyr)
library(DBI)

pg <- dbConnect(RPostgres::Postgres())

first_date <- '1962-09-05'
last_date <- '1985-12-27'

get_dates <- function(first_date, last_date) {
     tbl(pg, sql(paste0("SELECT caldt::date FROM generate_series('",  first_date,
                        "'::date, '", last_date, "'::date, '1 day') AS caldt")))
```

[1] BBK is "Using SAS in Financial Research" by Boehmer, Broussar, and Kallunki.

```
}
all_days <- get_dates(first_date, last_date)
```

Here I set `first_date` and `last_date` to the dates used in the Lo and MacKinlay (1988) data set. This gives me a one-column data set I can `LEFT JOIN` onto the index data.

```
all_days
```

```
## # Source:   SQL [?? x 1]
## # Database: postgres [iangow@wrds-pgdata.wharton.upenn.edu:9737/wrds]
##    caldt
##    <date>
##  1 1962-09-05
##  2 1962-09-06
##  3 1962-09-07
##  4 1962-09-08
##  5 1962-09-09
##  6 1962-09-10
##  7 1962-09-11
##  8 1962-09-12
##  9 1962-09-13
## 10 1962-09-14
## # ... with more rows
```

I next get the index data. It takes some digging around WRDS to work out where these data are. The data in `crsp.dsic` seem to be closest to the data in BBK, but they're not exactly the same.

As you can see, I use a `left_join` and then `coalesce` the index values so that if there's no index value on given day, I grab the index value for the next day. I calculate a Boolean variable `is_wed` using the PostgreSQL function `date_part`. As discussed here, the first argument is the subfield of interest and `dow` means the day of the week.

```
dsic <- tbl(pg, sql("SELECT * FROM crsp.dsic"))

weekly <-
    all_days %>%
    left_join(dsic, by="caldt") %>%
    arrange(caldt) %>%
    mutate(ewindd = coalesce(ewindd, lead(ewindd)),
           is_wed = date_part('dow', caldt)==3) %>%
    filter(is_wed, caldt >= first_date) %>%
    select(caldt, ewindd) %>%
    collect()
```

```
rs <- dbDisconnect(pg)

weekly
```

```
## # A tibble: 1,217 x 2
##    caldt      ewindd
##    <date>      <dbl>
##  1 1962-09-05   19.1
##  2 1962-09-12   19.2
##  3 1962-09-19   19.1
##  4 1962-09-26   18.3
##  5 1962-10-03   18.1
##  6 1962-10-10   18.3
##  7 1962-10-17   18.1
##  8 1962-10-24   17.1
##  9 1962-10-31   17.6
## 10 1962-11-07   18.2
## # ... with 1,207 more rows
```

The next step is to calculate the ratio of variance of q-week returns to the variance of one-week returns. Lo and MacKinlay (1988) provide a test statistic that has an asymptotic standard normal distribution. If P_k is the natural logaritm of the index in period k, then returns are simply $P_k - P_{k-1}$.

Denoting mean returns as $\hat{\mu}$, we can calculate the following quantities

$

$$
\begin{aligned}
\hat{\mu} &= \frac{1}{n}\sum_{k=1}^{n}\left(P_k - P_{k-1}\right) \\
\overline{\sigma}_a^2 &= \frac{1}{n-1}\sum_{k=1}^{n}\left(P_k - P_{k-1} - \hat{\mu}\right)^2 \\
\overline{\sigma}_q^2 &= \frac{1}{m}\sum_{k=1}^{n}\left(P_k - P_{k-q} - \hat{\mu}\right)^2 \\
m &= q(n-q+1)\left(1 - \frac{q}{n}\right) \\
\overline{M}_r &= \frac{\sigma_q}{\sigma_a} - 1
\end{aligned}
$$

$

Lo and MacKinlay show that the test statistic

$$
z^* = \frac{\sqrt{n}\overline{M}_r}{\sqrt{\hat{\theta}}} \rightarrow N(0,1)
$$

where

$$\hat{\theta} = \sum_{j=1}^{q-1} \left(\left(\frac{2(q-j)}{q} \right)^2 \hat{\delta}(j) \right)$$

and

$$\hat{\delta}(j) = \frac{n \sum_{k=j+1}^{n} \left(P_k - P_{k-1} - \hat{\mu}\right)^2 \left(P_k - P_{k-j-1} - \hat{\mu}\right)^2}{\left(\sum_{k=j+1}^{n} \left(P_k - P_{k-1} - \hat{\mu}\right)^2\right)^2}$$

BBK first provide code for $q = 2$ and then carefully adapt the code for $q = 4$. BBK direct the reader to a later chapter to show how the SAS macro facility could be used to produce a general function for any value of $q \geq 2$.

But in R, it is easy enough to make a function right from the beginning. While BBK's SAS code involves multiple steps and data sets, everything we need is a function of the index values (which become $\{P_k : k \in (1, n)\}$ after taking logs) and the value q.

Most of the calculations above are straightforward except for $\hat{theta}$, which involves summing the product of two terms over values of j ranging from 1 to $q - 1$. One approach would be to use a loop:

```
theta <- 0
for (j in 1:(q-1)) {
    theta <- theta + (2 * (q - j)/q)^2 + delta(j)
}
```

But an alternative approach is to use vectors. Taking $q = 4$, we can calculate the first term of the product as follows:

```
q <- 4
j <- 1:(q-1)
(2 * (q - j)/q)^2
```

```
## [1] 2.25 1.00 0.25
```

Doing the same for $\hat{\delta} j$ is a little more complicated, but quite feasible using the `lapply` functional. The basic idea of `lapply` is that it applies a function (its second argument) to the list given as its first argument and returns a list of results. Here we create a function `f` and apply it to the list of numbers from 1 to 6. As you can see, the result (`res`) is a list.

```
f <- function(x) 3 * x^2 + 2 * x + 6
res <- lapply(1:6, f)
res
```

```
## [[1]]
## [1] 11
##
## [[2]]
## [1] 22
```

```
##
## [[3]]
## [1] 39
##
## [[4]]
## [1] 62
##
## [[5]]
## [1] 91
##
## [[6]]
## [1] 126
```

We can turn the result into a vector by using `unlist`. Also, we can skip the step of assigning the function to `f`. Combining these two ideas gives:

```
res <- unlist(lapply(1:6, function(x) 3 * x^2 + 2 * x + 6))
res
```

```
## [1]  11  22  39  62  91 126
```

We can sum `res` by calling `sum(res)`:

```
sum(res)
```

```
## [1] 351
```

We apply this idea in calculating `deltop` (the numerator or "top" of the expression for $\hat{\delta}(j)$) below and then summing to calculate $\hat{\theta}$. The function `get_varrat` takes index values (`p`) and a value of q (`q`) and returns output much like that on pp.12–13 of BBK.

```
get_varrat <- function(p, q) {

    p <- log(p)
    ret <- p - lag(p)
    ret_q  <- p - lag(p, q)

    muhat <- mean(ret, na.rm=TRUE)
    n <- sum(!is.na(ret))

    sqr_dev <- (ret -  muhat)^2
    siga <- sum(sqr_dev, na.rm = TRUE)/(n-1)

    m = q * (n - q + 1) * (1 - q / n)
    sigc <- sum((ret_q - q * muhat)^2, na.rm = TRUE)/m
    varrat <- sigc/siga

    j <- 1:(q-1)
```

```
    delbot <- sum(sqr_dev, na.rm = TRUE)^2
    deltop <- n * unlist(lapply(j, function(x) sum(sqr_dev * lag(sqr_dev, x), na.rm = T

    delta <- deltop/delbot
    theta <- sum(((2 * (q - j)/q)^2) * delta)
    z <- sqrt(n) * (varrat - 1)/sqrt(theta)

    cat(sprintf("Number of weekly returns: %d\n", n))
    cat(sprintf("Variance ratio:           %3.5f\n", varrat))
    cat(sprintf("Test statistic:           %3.5f\n", z))
}
```

Now we have our function, we can reproduce Output 2.1 and Output 2.2 from BBK.

```
get_varrat(weekly$ewindd, 2)
```

```
## Number of weekly returns: 1216
## Variance ratio:           1.29518
## Test statistic:           7.51377
```

```
get_varrat(weekly$ewindd, 4)
```

```
## Number of weekly returns: 1216
## Variance ratio:           1.64118
## Test statistic:           8.88605
```