Applied Business Research

Eric Weisbrod, with collaboration from ACCT 995 Students

2025-01-22

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

```
1 + 1
```

```
[1] 2
```

1 Introduction

This is a work in process website for a potential book on applied business research in R. The goal of the website is to provide tools and examples for reproducible and well-formatted research reports.

This is an example of a Quarto citation Knuth (1984) in a sentence.

2 Project Setup

This chapter will have links and explainers on how to get your quarto and other GitHub projects going...

The top finance/accounting/aio journals are about to be flooded with highly reproducible KU research.

2.1 Git vs. GitHub

Git is the underlying code that helps manage version control of your projects. You can find more information about the details of Git here. Information about how to install git on your machine can be found here.

GitHub is a web-based user interface that makes Git easier to work with by allowing "point-and-click" version control rather than typing git commands. You will want to set up a GitHub account. The rest of this chapter will reference the use of GitHub, although everything discussed can also be accomplished through the Git language.

2.2 Language of GitHub

It is important to remember that GitHub acts as a version control interface for your research projects. So, while we will discuss the verbiage of GitHub, at its core, all that it is doing is keeping track of the changes that you make to your code. It may be helpful to translate the Git language into words you regularly associate with project management.

Repository - A repository is where all the code for a specific project lives. You can think of it like the project folder where your code is stored. The benefit of a repository is that it is stored online, allowing you to easily access it from any machine. You will most likely want to make your repository private, so that only you and people you identify as co-authors can access your code.

Fork - "Forking" a repository is equivalent to making a copy of someone else's repository. If a repository is made public, then anyone can fork the repository to have their own copy. This is unlikely to be something that will commonly occur in your own research, as we will discuss

next. You can think of forking a repository as the same as copying someone else's code folder and pasting it onto your computer.

Clone - "Cloning" is where the power of GitHub really begins. Cloning a repository is the same as giving your local machine (e.g., computer) access to the code in the repository. Think of it like installing Dropbox on another computer. Now you have access to all the files stored on Dropbox. Cloning a repository is the exact same thing for code. Once you have cloned the repository, you can now work on the code from that machine. The power comes in by being able to clone the repository on multiple computers, and your coauthors doing the same, allowing you all to work on the same set of code. To clone a repository, you will need the cloning URL. To get this, go to the repository on GitHub, click the green "Code" button, and click on the copy button (two intersecting squares) next to the HTTPS URL.

Staging - Once you save the code you are working on to your local computer, a new row will appear under the "Git" tab in the top right panel of R Studio. In order to officially "save" your changes to your online GitHub repository, you need to first stage the changes. "Staging" is the same thing as "selecting" which changes you want to officially send to GitHub, which will be discussed next.

Commit - Once you have staged your changes, you are ready to begin the process of uploading them to GitHub. The first thing you need to do is to "commit" the changes. As the name implies, a "commit" means you are committed ;) to making the changes to the code. To do this, click the "Commit" button in the top right panel of R Studio. You will need to type a short message which summarizes the changes that you have made, which will be visible on GitHub so you can easily see the evolution of your code over time. Once you have typed your message, click "Commit".

Pull - "Pulling" is effectively the same as downloading the most recent version of your project's code onto your local computer. When you click "Pull" in the top right panel of R Studio, you are "pulling" the code from GitHub onto your local machine. **You should always pull before you push!**

Push - "Pushing" is the final step of saving your code through GitHub. After you have committed your changes and pulled the most recent version of the project to your machine, you can send your changes to GitHub by clicking "Push" in the top right panel of R Studio. Doing this officially sends the changes that you have made up to GitHub, effectively pushing the code from your local device onto GitHub. This can also be thought of as if you made changes to a file on your local computer and then saved that file onto Dropbox. The power of GitHub is that you can see both the old version and new version, and restore your code to older versions if you decide you don't like your changes.

2.3 Power of Projects

When you first begin working with R, one of the first things that you learn are the benefits of storing/saving code in R scripts. In theory, we could always write all of our code in the terminal, and just rewrite it every time we wanted to run a command (gross!). Using R scripts allows us to save our code, which can be run the exact same way every time. In a nutshell, what the R script is really doing is making our code more reproducible, as we can be sure that we are always running the same commands.

The next evolution of your R journey probably went something like this... You realized that trying to code up an entire research project in one R script was burdensome. There are thousands of lines of code, some of which you do not want to run every time. So, you break up your project into multiple R scripts, with each one serving a different function (e.g., data download, data cleaning, analysis, figures, etc.). While this likely initially occurred just by saving the different scripts in the Dropbox folder where you stored your project files (code, data, paper drafts, etc.), we will next discuss how we can integrate GitHub to help make working on a research project easier.

Projects in R can be thought of as self-contained folders, where all your code and associated documents can live. The power of these projects is that they can be integrated with GitHub, making your work more reproducible by keeping all your code for a project in one place along with the version history.[1] To start a new project, you will first create a repository on GitHub with this project's name. You then go into R Studio and click "File -> New Project". From there, you will click on "Version Control -> Git". This brings you to a screen which will allow you to clone (see above if this is unfamiliar) your GitHub repository to your local machine. You will copy and paste the URL from the repository into the "Repository URL" text box. The "Project directory name:" box will populate with a location on your local computer. This is where the project and subsequent R scripts will live. How you manage these comes down to personal preference, but a common way to store GitHub projects locally is to create a "git" folder on your computer's C-drive and then store each project in the git folder. Once you have set the project directory, you can select "Create Project" to create the local version of the repository on your computer.

The key thing to notice (and one of the many powers of working within projects) is located near the top right corner of R Studio. You will now see a blue box with the letter "R" inside of it, which is the image of an R project. Next to this, you will see the name of the project you just created, indicating that you are working "inside" this project. When you look at the "Files" in the bottom right panel, you will see that the directory has been changed to the location of your R project. This allows you to see all the scripts associated with this project, and quickly change between scripts. Importantly, any pushes that you make to GitHub will be pushed to that specific project.

[1] Projects and repositories are linked together. A project lives in R and is an R object, while a repository lives on GitHub. When you read "repository" think "project" or vice versa, however you prefer.

One of the nice features of working in projects is the ability to quickly access your project code and switch between projects. For example, if you were conducting a research project using just R scripts, you would need to save them somewhere (e.g., Dropbox) and navigate to that folder to open/access them. When using an R project, rather than navigating to the folder on your computer that contains your code, you can just open R Studio and click on the R project button in the top right corner (blue cube with "R" inside). This will open up a drop down menu of all the projects you have on your computer, allowing you to open the one you want. When you close the project, whatever R scripts that you have open at the time you close will automatically open again the next time you open the project.

2.4 Reproducible Workflow

All of the things we have discussed up to this point have been about making our code and research projects more reproducible and easier to understand. Beyond these tools, it is important to remember that a truly reproducible research project means that someone can take your project and with limited/no prior knowledge about what you were doing, replicate your findings and understand your code. To make this possible, it is important to try and make your code as clear and concise as possible, including plenty of documentation and comments in the code explaining what is happening. One of the best ways that I have found to keep my workflow easy to track is through developing a consistent naming convention between R scripts and data files.

When creating my R scripts, I always begin the script name with a number, beginning with "00". This is then followed by "-short-description.R" where the "short-description" is a short description of what the code is doing. The scripts are numbered based on when in the sequence they should be run. So, you would run the "00" scripts first, then "01", "02", and so on. Importantly, any data that I save from a script is named beginning with that script's numeric identifier. For example, if I save a data set called "new_data.parquet" from an R script titled "02-create-new-data.R", then I would save the parquet file with the name "02_new_data.parquet" in my Dropbox data folder. Doing this helps along two dimensions. First, it makes your data folder much more organized, with the data ordered in the chronological order that it was generated by your code. Second (and most importantly) it makes it immediately obvious which script generated the data set. That way, if someone has questions about a particular data set, you can immediately know what R script generated the data.

Below is an example project setup with the naming convention. The first level of bullet points are the R scripts (which would be located in the git folder) and the second level are the data sets (which would be saved on Dropbox).

- 00-Global-Parameters.R
- 01-Download-WRDS-Data.R
    - 01_compustat_data_raw.parquet

- 02-Create-Master-Data.R
  - 02_master_data.parquet
- 03-Main-Analyses.R
  - 03.01_table_1.tex
  - 03.02_table_2.tex
- 04-Figures.R
  - 04.01_figure_1.jpeg
  - 04.02_figure_2.jpeg

3 Installing Packages, Setting Global Parameters, and Creating Necessary Functions

In this chapter we will use some tricks to setup our project environment in such a way that can be easily shared. We will start with installing some packages, then work on setting a data path, and finally create some functions that can be used repeatedly in the project.

4 Install Necessary Packages

This script uses an R package called 'pacman' to check if the other packages used in this example are installed on your machine. If not, it installs them.

This line checks for the 'pacman' package, and installs it if needed.

```
if (!require("pacman")) install.packages("pacman")
```

```
Loading required package: pacman
```

This line uses 'pacman' to install all of the other packages in the project. Not all of the packages are crucial but we can put them all here anyways for future use.You may see a lot of warnings as the packages install but that is ok.

```
pacman::p_load(tidyverse,
               dbplyr,
               RPostgres,
               DBI,
               glue,
               arrow,
               haven,
               tictoc,
               lubridate,
               modelsummary,
               kableExtra,
               formattable,
               fixest,
               flextable,
               officer,
               corrplot,
               equatags,
               broom,
               usethis,
               scales,
               forcats
              )
```

4.1 Log into wrds to check

We will try to log into WRDS site here using Postgres. Just in case, check to make sure there is not already a connection open.

```
if(exists("wrds")){
  dbDisconnect(wrds)
}
```

Now log on by uncommenting the below code, it will prompt for your WRDS username and password. You can use the 'Keyring' in the prompt to save the username and password securely.

```
#wrds <- dbConnect(Postgres(),
 #                 host='wrds-pgdata.wharton.upenn.edu',
  #                port=9737,
   #               user=rstudioapi::askForSecret("WRDS user"),
    #              password=rstudioapi::askForSecret("WRDS pw"),
     #             sslmode='require',
      #            dbname='wrds')
```

Run the below line. If you are connected you should see something like the below output in your console:

```
#<PqConnection>wrds@wrds-pgdata.wharton.upenn.edu:9737
```

If the above two commands time out, you may be stuck behind a firewall etc. Remember to close the connection after testing

```
if(exists("wrds")){
  dbDisconnect(wrds)
}
```

5 Define Global Parameters

In this section, we will first setup a data folder path and create a project specific-parameter.

5.1 Setup the Data Folder Path

A useful trick for sharing codes with coauthors is to store the path to your local data folder in your project-level R environment file (.Renviron).

Step 1: Uncomment and run the below line to edit your .Renviron file.

```
#usethis::edit_r_environ('project')
```

Step 2: Uncomment and paste the below line into the .Renviron file. The .Renviron file should be open in a separate tab in Rstudio if you ran the above line correctly.

```
#DATA_PATH = 'E:/acct_995_data/abr'
```

Replace the quoted directory name in the example with the path to whatever folder you would like to store your data in. Notice the slashes might go the other way from Windows.

It is recommended to not store your data in the Git project folder. Github is designed for hosting code, not data. We can use a separate folder, usually in Dropbox if there is enough space on Dropbox.

Step 3: Comment out the code you ran to set up your Renviron and then restart R. You should only have to do these steps once any time you start a new project or download a project to a new computer.

The below line loads the data path from the project environment. The benefit of this is that you now only need one version of the code no matter which coauthor is running the code. The same code should work for all coauthors or work the same way whether you are on your laptop or desktop, etc. Essentially, anyone working on the same project can access the data without manually setting the data path each time.

```
data_path<- Sys.getenv('DATA_PATH')
```

If the above process is too complicated and you don't have coauthors you can just set data_path manually by deleting everything above and uncommenting the below:

```
#data_path<- "E:/acct_995_data/abr"
```

You would then replace "E:/acct_995_data/abr" with your own data path.

5.2 Setup any project-specific parameters

For example, you might want to define sample years here and then you can refer to them throughout the code as needed, there are many use-cases.

Example parameters: beg_year and end_year to define the sample period:

```
beg_year <- 1970
```

the assignment arrow is an R grammar style, but equal signs work too.

```
end_year = 2022
```

6 Utilities

In this section, we will create some useful utility functions that we can later use throughout the project repeatedly.

6.1 R Options

The below line will get rid of scientific notation which is unnecessary sometimes.

```
options(scipen=999)
```

This will print blanks for NAs in Kable documents.

```
options(knitr.kable.NA = '')
message("set R formatting options") #Check whether the R formatting options are applied.
```

```
set R formatting options
```

6.2 Parquet functions

This allows for use of reading and writing parquet files without library(arrow). This also sets default compression method for writing.

```
read_parquet <- arrow::read_parquet
write_parquet <- function(x, p) {
  arrow::write_parquet(x, p, compression = "gzip", compression_level = 5)
}
message("imported parquet functions") #Check whether the parquet function is defined
```

```
imported parquet functions
```

6.3 Industry functions

This section will use Fama-French industry classification to assign names and numbers to various industries. We will use the 12 and 49 industry classifications.

Assign FF12 industry name, given sic code.

```
assign_FF12 <- function(sic) {
  dplyr::case_when(
    sic >= 0100 & sic <= 0999 ~ "Consumer Nondurables",
    sic >= 2000 & sic <= 2399 ~ "Consumer Nondurables",
    sic >= 2700 & sic <= 2749 ~ "Consumer Nondurables",
    sic >= 2770 & sic <= 2799 ~ "Consumer Nondurables",
    sic >= 3100 & sic <= 3199 ~ "Consumer Nondurables",
    sic >= 3940 & sic <= 3989 ~ "Consumer Nondurables",

    sic >= 2500 & sic <= 2519 ~ "Consumer Durables",
    sic >= 2590 & sic <= 2599 ~ "Consumer Durables",
    sic >= 3630 & sic <= 3659 ~ "Consumer Durables",
    sic >= 3710 & sic <= 3711 ~ "Consumer Durables",
    sic >= 3714 & sic <= 3714 ~ "Consumer Durables",
    sic >= 3716 & sic <= 3716 ~ "Consumer Durables",
    sic >= 3750 & sic <= 3751 ~ "Consumer Durables",
    sic >= 3792 & sic <= 3792 ~ "Consumer Durables",
    sic >= 3900 & sic <= 3939 ~ "Consumer Durables",
    sic >= 3990 & sic <= 3999 ~ "Consumer Durables",

    sic >= 2520 & sic <= 2589 ~ "Manufacturing",
    sic >= 2600 & sic <= 2699 ~ "Manufacturing",
    sic >= 2750 & sic <= 2769 ~ "Manufacturing",
    sic >= 3000 & sic <= 3099 ~ "Manufacturing",
    sic >= 3200 & sic <= 3569 ~ "Manufacturing",
    sic >= 3580 & sic <= 3629 ~ "Manufacturing",
    sic >= 3700 & sic <= 3709 ~ "Manufacturing",
    sic >= 3712 & sic <= 3713 ~ "Manufacturing",
    sic >= 3715 & sic <= 3715 ~ "Manufacturing",
    sic >= 3717 & sic <= 3749 ~ "Manufacturing",
    sic >= 3752 & sic <= 3791 ~ "Manufacturing",
    sic >= 3793 & sic <= 3799 ~ "Manufacturing",
    sic >= 3830 & sic <= 3839 ~ "Manufacturing",
    sic >= 3860 & sic <= 3899 ~ "Manufacturing",

    sic >= 1200 & sic <= 1399 ~ "Energy",
```

```
    sic >= 2900 & sic <= 2999 ~ "Energy",

    sic >= 2800 & sic <= 2829 ~ "Chemicals",
    sic >= 2840 & sic <= 2899 ~ "Chemicals",

    sic >= 3570 & sic <= 3579 ~ "Business Equipment",
    sic >= 3660 & sic <= 3692 ~ "Business Equipment",
    sic >= 3694 & sic <= 3699 ~ "Business Equipment",
    sic >= 3810 & sic <= 3829 ~ "Business Equipment",
    sic >= 7370 & sic <= 7379 ~ "Business Equipment",

    sic >= 4800 & sic <= 4899 ~ "Telecommunications",

    sic >= 4900 & sic <= 4949 ~ "Utilities",

    sic >= 5000 & sic <= 5999 ~ "Retail",
    sic >= 7200 & sic <= 7299 ~ "Retail",
    sic >= 7600 & sic <= 7699 ~ "Retail",

    sic >= 2830 & sic <= 2839 ~ "Healthcare",
    sic >= 3693 & sic <= 3693 ~ "Healthcare",
    sic >= 3840 & sic <= 3859 ~ "Healthcare",
    sic >= 8000 & sic <= 8099 ~ "Healthcare",

    sic >= 6000 & sic <= 6999 ~ "Finance",

    TRUE ~ "Other"
  )
}
```

Assign FF12 industry number, given sic code

```
assign_FF12_num <- function(sic) {
  dplyr::case_when(
    sic >= 0100 & sic <= 0999 ~ 1,
    sic >= 2000 & sic <= 2399 ~ 1,
    sic >= 2700 & sic <= 2749 ~ 1,
    sic >= 2770 & sic <= 2799 ~ 1,
    sic >= 3100 & sic <= 3199 ~ 1,
    sic >= 3940 & sic <= 3989 ~ 1,

    sic >= 2500 & sic <= 2519 ~ 2,
```

```
    sic >= 2590 & sic <= 2599 ~ 2,
    sic >= 3630 & sic <= 3659 ~ 2,
    sic >= 3710 & sic <= 3711 ~ 2,
    sic >= 3714 & sic <= 3714 ~ 2,
    sic >= 3716 & sic <= 3716 ~ 2,
    sic >= 3750 & sic <= 3751 ~ 2,
    sic >= 3792 & sic <= 3792 ~ 2,
    sic >= 3900 & sic <= 3939 ~ 2,
    sic >= 3990 & sic <= 3999 ~ 2,

    sic >= 2520 & sic <= 2589 ~ 3,
    sic >= 2600 & sic <= 2699 ~ 3,
    sic >= 2750 & sic <= 2769 ~ 3,
    sic >= 3000 & sic <= 3099 ~ 3,
    sic >= 3200 & sic <= 3569 ~ 3,
    sic >= 3580 & sic <= 3629 ~ 3,
    sic >= 3700 & sic <= 3709 ~ 3,
    sic >= 3712 & sic <= 3713 ~ 3,
    sic >= 3715 & sic <= 3715 ~ 3,
    sic >= 3717 & sic <= 3749 ~ 3,
    sic >= 3752 & sic <= 3791 ~ 3,
    sic >= 3793 & sic <= 3799 ~ 3,
    sic >= 3830 & sic <= 3839 ~ 3,
    sic >= 3860 & sic <= 3899 ~ 3,

    sic >= 1200 & sic <= 1399 ~ 4,
    sic >= 2900 & sic <= 2999 ~ 4,

    sic >= 2800 & sic <= 2829 ~ 5,
    sic >= 2840 & sic <= 2899 ~ 5,

    sic >= 3570 & sic <= 3579 ~ 6,
    sic >= 3660 & sic <= 3692 ~ 6,
    sic >= 3694 & sic <= 3699 ~ 6,
    sic >= 3810 & sic <= 3829 ~ 6,
    sic >= 7370 & sic <= 7379 ~ 6,

    sic >= 4800 & sic <= 4899 ~ 7,

    sic >= 4900 & sic <= 4949 ~ 8,

    sic >= 5000 & sic <= 5999 ~ 9,
```

```
    sic >= 7200 & sic <= 7299 ~ 9,
    sic >= 7600 & sic <= 7699 ~ 9,

    sic >= 2830 & sic <= 2839 ~ 10,
    sic >= 3693 & sic <= 3693 ~ 10,
    sic >= 3840 & sic <= 3859 ~ 10,
    sic >= 8000 & sic <= 8099 ~ 10,

    sic >= 6000 & sic <= 6999 ~ 11,

    TRUE ~ 12
  )
}
```

Assign FF49 industry name, given sic code. Note that FF49 is just FF48 plus "Other"

```
assign_FF49 <- function(sic) {
  dplyr::case_when(
    sic >= 0100 & sic <= 0199 ~ "Agriculture",
    sic >= 0200 & sic <= 0299 ~ "Agriculture",
    sic >= 0700 & sic <= 0799 ~ "Agriculture",
    sic >= 0910 & sic <= 0919 ~ "Agriculture",
    sic >= 2048 & sic <= 2048 ~ "Agriculture",

    sic >= 2000 & sic <= 2009 ~ "Food Products",
    sic >= 2010 & sic <= 2019 ~ "Food Products",
    sic >= 2020 & sic <= 2029 ~ "Food Products",
    sic >= 2030 & sic <= 2039 ~ "Food Products",
    sic >= 2040 & sic <= 2046 ~ "Food Products",
    sic >= 2050 & sic <= 2059 ~ "Food Products",
    sic >= 2060 & sic <= 2063 ~ "Food Products",
    sic >= 2070 & sic <= 2079 ~ "Food Products",
    sic >= 2090 & sic <= 2092 ~ "Food Products",
    sic >= 2095 & sic <= 2095 ~ "Food Products",
    sic >= 2098 & sic <= 2099 ~ "Food Products",

    (sic >= 2064 & sic <= 2068) | (sic >= 2086 & sic <= 2086) | (sic >= 2087 & sic <= 2087)
    (sic >= 2080 & sic <= 2080) | (sic >= 2082 & sic <= 2082) | (sic >= 2083 & sic <= 2083)
    (sic >= 2100 & sic <= 2199) ~ "Tobacco Products",
    (sic >= 920 & sic <= 999) | (sic >= 3650 & sic <= 3651) | (sic >= 3652 & sic <= 3652) |
    (sic >= 7800 & sic <= 7829) | (sic >= 7830 & sic <= 7833) | (sic >= 7840 & sic <= 7841)
    (sic >= 2700 & sic <= 2709) | (sic >= 2710 & sic <= 2719) | (sic >= 2720 & sic <= 2729)
```

```
    (sic >= 2047 & sic <= 2047) | (sic >= 2391 & sic <= 2392) | (sic >= 2510 & sic <= 2519)
    (sic >= 2300 & sic <= 2390) | (sic >= 3020 & sic <= 3021) | (sic >= 3100 & sic <= 3111)
    (sic >= 8000 & sic <= 8099) ~ "Healthcare",
    (sic >= 3693 & sic <= 3693) | (sic >= 3840 & sic <= 3849) | (sic >= 3850 & sic <= 3851)
    (sic >= 2830 & sic <= 2830) | (sic >= 2831 & sic <= 2831) | (sic >= 2833 & sic <= 2833)
    (sic >= 2800 & sic <= 2809) | (sic >= 2810 & sic <= 2819) | (sic >= 2820 & sic <= 2829)
    (sic >= 3031 & sic <= 3031) | (sic >= 3041 & sic <= 3041) | (sic >= 3050 & sic <= 3053)
    (sic >= 2200 & sic <= 2269) | (sic >= 2270 & sic <= 2279) | (sic >= 2280 & sic <= 2284)
    (sic >= 800 & sic <= 899) | (sic >= 2400 & sic <= 2439) | (sic >= 2450 & sic <= 2459) |
    (sic >= 1500 & sic <= 1511) | (sic >= 1520 & sic <= 1529) | (sic >= 1530 & sic <= 1539)
    (sic >= 3300 & sic <= 3300) | (sic >= 3310 & sic <= 3317) | (sic >= 3320 & sic <= 3325)
    (sic >= 3400 & sic <= 3400) | (sic >= 3443 & sic <= 3443) | (sic >= 3444 & sic <= 3444)
    (sic >= 3510 & sic <= 3519) | (sic >= 3520 & sic <= 3529) | (sic >= 3530 & sic <= 3530)
    (sic >= 3600 & sic <= 3600) | (sic >= 3610 & sic <= 3613) | (sic >= 3620 & sic <= 3621)
    (sic >= 2296 & sic <= 2296) | (sic >= 2396 & sic <= 2396) | (sic >= 3010 & sic <= 3011)
    (sic >= 3720 & sic <= 3720) | (sic >= 3721 & sic <= 3721) | (sic >= 3723 & sic <= 3724)
    (sic >= 3730 & sic <= 3731) | (sic >= 3740 & sic <= 3743) ~ "Shipbuilding, Railroad Equi
    (sic >= 3760 & sic <= 3769) | (sic >= 3795 & sic <= 3795) | (sic >= 3480 & sic <= 3489)
    (sic >= 1040 & sic <= 1049) ~ "Precious Metals",
    (sic >= 1000 & sic <= 1009) | (sic >= 1010 & sic <= 1019) | (sic >= 1020 & sic <= 1029)
    (sic >= 1200 & sic <= 1299) ~ "Coal",
    (sic >= 1300 & sic <= 1300) | (sic >= 1310 & sic <= 1319) | (sic >= 1320 & sic <= 1329)
    (sic >= 4900 & sic <= 4900) | (sic >= 4910 & sic <= 4911) | (sic >= 4920 & sic <= 4922)
    (sic >= 4800 & sic <= 4800) | (sic >= 4810 & sic <= 4813) | (sic >= 4820 & sic <= 4822)
    (sic >= 7020 & sic <= 7021) | (sic >= 7030 & sic <= 7033) | (sic >= 7200 & sic <= 7200)
    (sic >= 2750 & sic <= 2759) | (sic >= 3993 & sic <= 3993) | (sic >= 7218 & sic <= 7218)
    (sic >= 3570 & sic <= 3579) | (sic >= 3680 & sic <= 3680) | (sic >= 3681 & sic <= 3681)
    (sic >= 7370 & sic <= 7372) | (sic >= 7375 & sic <= 7375) | (sic >= 7373 & sic <= 7373)
    (sic >= 3622 & sic <= 3622) | (sic >= 3661 & sic <= 3661) | (sic >= 3662 & sic <= 3662)
    (sic >= 3811 & sic <= 3811) | (sic >= 3820 & sic <= 3820) | (sic >= 3821 & sic <= 3821)
    (sic >= 2520 & sic <= 2549) | (sic >= 2600 & sic <= 2639) | (sic >= 2670 & sic <= 2699)
    (sic >= 2440 & sic <= 2449) | (sic >= 2640 & sic <= 2659) | (sic >= 3220 & sic <= 3221)
    (sic >= 4000 & sic <= 4013) | (sic >= 4040 & sic <= 4049) | (sic >= 4100 & sic <= 4100)
    (sic >= 5000 & sic <= 5000) | (sic >= 5010 & sic <= 5015) | (sic >= 5020 & sic <= 5023)
    (sic >= 5200 & sic <= 5200) | (sic >= 5210 & sic <= 5219) | (sic >= 5220 & sic <= 5229)
    (sic >= 5800 & sic <= 5819) | (sic >= 5820 & sic <= 5829) | (sic >= 5890 & sic <= 5899)
    (sic >= 6000 & sic <= 6000) | (sic >= 6010 & sic <= 6019) | (sic >= 6020 & sic <= 6020)
    (sic >= 6300 & sic <= 6300) | (sic >= 6310 & sic <= 6319) | (sic >= 6320 & sic <= 6329)
    (sic >= 6500 & sic <= 6500) | (sic >= 6510 & sic <= 6510) | (sic >= 6512 & sic <= 6512)
    (sic >= 6200 & sic <= 6299) | (sic >= 6700 & sic <= 6700) | (sic >= 6710 & sic <= 6719)
   TRUE ~ "Other"
  )
```

```
}
```

Assign FF49 industry number, given sic code.

```
assign_FF49_num <- function(sic) {
  dplyr::case_when(
    sic >= 0100 & sic <= 0199 ~ 1,
    sic >= 0200 & sic <= 0299 ~ 1,
    sic >= 0700 & sic <= 0799 ~ 1,
    sic >= 0910 & sic <= 0919 ~ 1,
    sic >= 2048 & sic <= 2048 ~ 1,

    sic >= 2000 & sic <= 2009 ~ 2,
    sic >= 2010 & sic <= 2019 ~ 2,
    sic >= 2020 & sic <= 2029 ~ 2,
    sic >= 2030 & sic <= 2039 ~ 2,
    sic >= 2040 & sic <= 2046 ~ 2,
    sic >= 2050 & sic <= 2059 ~ 2,
    sic >= 2060 & sic <= 2063 ~ 2,
    sic >= 2070 & sic <= 2079 ~ 2,
    sic >= 2090 & sic <= 2092 ~ 2,
    sic >= 2095 & sic <= 2095 ~ 2,
    sic >= 2098 & sic <= 2099 ~ 2,

    (sic >= 2064 & sic <= 2068) | (sic >= 2086 & sic <= 2086) | (sic >= 2087 & sic <= 2087)
    (sic >= 2080 & sic <= 2080) | (sic >= 2082 & sic <= 2082) | (sic >= 2083 & sic <= 2083)
    (sic >= 2100 & sic <= 2199) ~ 5,
    (sic >= 920 & sic <= 999) | (sic >= 3650 & sic <= 3651) | (sic >= 3652 & sic <= 3652) |
    (sic >= 7800 & sic <= 7829) | (sic >= 7830 & sic <= 7833) | (sic >= 7840 & sic <= 7841)
    (sic >= 2700 & sic <= 2709) | (sic >= 2710 & sic <= 2719) | (sic >= 2720 & sic <= 2729)
    (sic >= 2047 & sic <= 2047) | (sic >= 2391 & sic <= 2392) | (sic >= 2510 & sic <= 2519)
    (sic >= 2300 & sic <= 2390) | (sic >= 3020 & sic <= 3021) | (sic >= 3100 & sic <= 3111)
    (sic >= 8000 & sic <= 8099) ~ 11,
    (sic >= 3693 & sic <= 3693) | (sic >= 3840 & sic <= 3849) | (sic >= 3850 & sic <= 3851)
    (sic >= 2830 & sic <= 2830) | (sic >= 2831 & sic <= 2831) | (sic >= 2833 & sic <= 2833)
    (sic >= 2800 & sic <= 2809) | (sic >= 2810 & sic <= 2819) | (sic >= 2820 & sic <= 2829)
    (sic >= 3031 & sic <= 3031) | (sic >= 3041 & sic <= 3041) | (sic >= 3050 & sic <= 3053)
    (sic >= 2200 & sic <= 2269) | (sic >= 2270 & sic <= 2279) | (sic >= 2280 & sic <= 2284)
    (sic >= 800 & sic <= 899) | (sic >= 2400 & sic <= 2439) | (sic >= 2450 & sic <= 2459) |
    (sic >= 1500 & sic <= 1511) | (sic >= 1520 & sic <= 1529) | (sic >= 1530 & sic <= 1539)
    (sic >= 3300 & sic <= 3300) | (sic >= 3310 & sic <= 3317) | (sic >= 3320 & sic <= 3325)
    (sic >= 3400 & sic <= 3400) | (sic >= 3443 & sic <= 3443) | (sic >= 3444 & sic <= 3444)
```

```
    (sic >= 3510 & sic <= 3519) | (sic >= 3520 & sic <= 3529) | (sic >= 3530 & sic <= 3530)
    (sic >= 3600 & sic <= 3600) | (sic >= 3610 & sic <= 3613) | (sic >= 3620 & sic <= 3621)
    (sic >= 2296 & sic <= 2296) | (sic >= 2396 & sic <= 2396) | (sic >= 3010 & sic <= 3011)
    (sic >= 3720 & sic <= 3720) | (sic >= 3721 & sic <= 3721) | (sic >= 3723 & sic <= 3724)
    (sic >= 3730 & sic <= 3731) | (sic >= 3740 & sic <= 3743) ~ 25,
    (sic >= 3760 & sic <= 3769) | (sic >= 3795 & sic <= 3795) | (sic >= 3480 & sic <= 3489)
    (sic >= 1040 & sic <= 1049) ~ 27,
    (sic >= 1000 & sic <= 1009) | (sic >= 1010 & sic <= 1019) | (sic >= 1020 & sic <= 1029)
    (sic >= 1200 & sic <= 1299) ~ 29,
    (sic >= 1300 & sic <= 1300) | (sic >= 1310 & sic <= 1319) | (sic >= 1320 & sic <= 1329)
    (sic >= 4900 & sic <= 4900) | (sic >= 4910 & sic <= 4911) | (sic >= 4920 & sic <= 4922)
    (sic >= 4800 & sic <= 4800) | (sic >= 4810 & sic <= 4813) | (sic >= 4820 & sic <= 4822)
    (sic >= 7020 & sic <= 7021) | (sic >= 7030 & sic <= 7033) | (sic >= 7200 & sic <= 7200)
    (sic >= 2750 & sic <= 2759) | (sic >= 3993 & sic <= 3993) | (sic >= 7218 & sic <= 7218)
    (sic >= 3570 & sic <= 3579) | (sic >= 3680 & sic <= 3680) | (sic >= 3681 & sic <= 3681)
    (sic >= 7370 & sic <= 7372) | (sic >= 7375 & sic <= 7375) | (sic >= 7373 & sic <= 7373)
    (sic >= 3622 & sic <= 3622) | (sic >= 3661 & sic <= 3661) | (sic >= 3662 & sic <= 3662)
    (sic >= 3811 & sic <= 3811) | (sic >= 3820 & sic <= 3820) | (sic >= 3821 & sic <= 3821)
    (sic >= 2520 & sic <= 2549) | (sic >= 2600 & sic <= 2639) | (sic >= 2670 & sic <= 2699)
    (sic >= 2440 & sic <= 2449) | (sic >= 2640 & sic <= 2659) | (sic >= 3220 & sic <= 3221)
    (sic >= 4000 & sic <= 4013) | (sic >= 4040 & sic <= 4049) | (sic >= 4100 & sic <= 4100)
    (sic >= 5000 & sic <= 5000) | (sic >= 5010 & sic <= 5015) | (sic >= 5020 & sic <= 5023)
    (sic >= 5200 & sic <= 5200) | (sic >= 5210 & sic <= 5219) | (sic >= 5220 & sic <= 5229)
    (sic >= 5800 & sic <= 5819) | (sic >= 5820 & sic <= 5829) | (sic >= 5890 & sic <= 5899)
    (sic >= 6000 & sic <= 6000) | (sic >= 6010 & sic <= 6019) | (sic >= 6020 & sic <= 6020)
    (sic >= 6300 & sic <= 6300) | (sic >= 6310 & sic <= 6319) | (sic >= 6320 & sic <= 6329)
    (sic >= 6500 & sic <= 6500) | (sic >= 6510 & sic <= 6510) | (sic >= 6512 & sic <= 6512)
    (sic >= 6200 & sic <= 6299) | (sic >= 6700 & sic <= 6700) | (sic >= 6710 & sic <= 6719)
    TRUE ~ 49
  )
}
message("imported industry functions") #Check whether all industry functions are defined
```

```
imported industry functions
```

6.4 Variable transformation functions

Below is a general function to standardize a variable to mean zero, and standard deviation of one. This is sometimes handy to compare different time series.

```r
standardize <- function(x){
  (x - mean(x, na.rm=TRUE)) / sd(x, na.rm=TRUE)
}
```

Below is a general function to winsorize a variable in a mutate statement. Note that, winsorization does not remove the extreme values, it sets the extreme values to some specified values (e.g. 1st and 99th percentile).

```r
winsorize_x = function(x, cuts = c(0.01,0.01)) {
  cut_point_top <- quantile(x, 1 - cuts[2], na.rm = T)
  cut_point_bottom <- quantile(x, cuts[1], na.rm = T)
  i = which(x >= cut_point_top)
  x[i] = cut_point_top
  j = which(x <= cut_point_bottom)
  x[j] = cut_point_bottom
  return(x)
}
```

Below is a general function to truncate a variable in a mutate statement. Unlike winsorization, truncation does remove the extreme values.

```r
truncate_x = function(x, cuts = c(0.01,0.01)) {
  cut_point_top <- quantile(x, 1 - cuts[2], na.rm = T)
  cut_point_bottom <- quantile(x, cuts[1], na.rm = T)
  i = which(x >= cut_point_top)
  x[i] = NA_real_
  j = which(x <= cut_point_bottom)
  x[j] = NA_real_
  return(x)
}

message("imported transformation functions") #Check whether all transformation functions are
```

```
imported transformation functions
```

7 Obtaining and Merging Data

Parquet files offer significant advantages when dealing with large datasets that need to be retrieved, stored, and merged efficiently. Their columnar storage format allows reading only the necessary columns instead of scanning the entire dataset, making data retrieval much faster compared to traditional formats like CSV or Stata. Additionally, Parquet optimizes input/output operations, reducing the amount of data that needs to be read from disk, which speeds up processing times when merging multiple datasets.

Another key benefit of Parquet is its efficient storage and compression capabilities. Unlike CSV or Stata files, which store data in a row-based format, Parquet applies built-in compression algorithms such as Gzip or Snappy, significantly reducing file size. This not only saves disk space but also speeds up data transfer, especially when obtaining datasets from remote servers like WRDS, AWS, or PostgreSQL databases.

Parquet files also ensure schema consistency and data type preservation, which is crucial when merging datasets from different sources. In contrast to CSV files, where numerical values may sometimes be interpreted as text, Parquet maintains strict data types. It also efficiently handles missing values, reducing potential errors and inconsistencies when performing joins and merges.

When merging large datasets, Parquet's columnar format and filtering capabilities help reduce memory usage and processing time. Instead of loading entire datasets into memory, users can select and load only the relevant columns before merging, significantly optimizing resource allocation. Additionally, Parquet is highly compatible with modern data science tools, supporting parallel computing and batch processing with frameworks like Spark, Dask, and DuckDB.

Parquet's cross-platform support makes it ideal for working in R, Python (pandas, pyarrow, Polars), SQL databases, and cloud platforms such as AWS S3 and Google BigQuery. This ensures seamless integration with existing workflows, whether working on local machines or in cloud-based data pipelines. Given its speed, storage efficiency, and compatibility, Parquet is a preferred format for obtaining and merging large financial datasets from sources like Compustat, CRSP, and other WRDS databases.

Let's first learn how to obtain data from WRDS. The code below shows how to connect to WRDS.

```
# Setup ----------------------------------------------------------------------

# Load Libraries [i.e., packages]
library(dbplyr)
library(RPostgres)
library(DBI)
library(glue)
library(arrow)
```

```
Attaching package: 'arrow'

The following object is masked from 'package:utils':

    timestamp
```

```
library(haven)
library(tictoc) #very optional timer, mostly as a teaching example
library(tidyverse) # I like to load tidyverse last to avoid package conflicts
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4

-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x lubridate::duration() masks arrow::duration()
x dplyr::filter()       masks stats::filter()
x dplyr::ident()        masks dbplyr::ident()
x dplyr::lag()          masks stats::lag()
x dplyr::sql()          masks dbplyr::sql()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```
#load helper scripts
#similar to "include" statement in SAS.
source("E:/acct_995_data/abr/-Global-Parameters.R")
```

```
v Setting active project to "E:/e679w418/_git/abr".
[ ] Edit '.Renviron'.
[ ] Restart R for changes to take effect.
```

```
source("E:/acct_995_data/abr/utils.R")
```

```
set R formatting options
imported parquet functions
imported industry functions
imported transformation functions
```

```
# Log into wrds ------------------------------------------------------------

if(exists("wrds")){
  dbDisconnect(wrds)  # because otherwise WRDS might time out
}
wrds <- dbConnect(Postgres(),
                  host = 'wrds-pgdata.wharton.upenn.edu',
                  port = 9737,
                  user = keyring::key_get("WRDS_user"),
                  password = keyring::key_get("WRDS_pw"),
                  sslmode = 'require',
                  dbname = 'wrds')
```

The R script below retrieves and processes financial data from the **Compustat** database using **PostgreSQL** on the **WRDS server**. It begins by listing available tables within a specified schema and then loads the necessary datasets (`funda` and `company`). Applying standard filters, it selects key financial variables, merges company-level information, and constructs industry classifications. Additional variables, such as market value of equity (MVE) and earnings before special items, are derived. The dataset is then filtered to retain **U.S. firms from 1967 onward** while excluding financial and utility firms.

```
# See a list of tables in a schema -----------------------------------------
# Just an example to play with the Postgres server

# List all of the tables in Compustat (comp)
wrds |>
  DBI::dbListObjects(DBI::Id(schema = 'comp')) |>
  dplyr::pull(table) |>
  purrr::map(~slot(.x, 'name'))  |>
  dplyr::bind_rows()  |>
```

```
  View()
# can replace "comp" with any schema such as "crsp" "ibes" etc.
# schemas on the Postgres server are similar to WRDS SAS libraries

# Load table references and download data -------------------------------------

# Load funda as a tbl
comp.funda <- tbl(wrds,in_schema("comp", "funda"))
comp.company <- tbl(wrds,in_schema("comp", "company"))

# Optional line:
#if you want to see how long a block of code takes you can start a tictoc timer
#it will tell you how long it takes between when you run tic() and when you run
# toc()
tictoc::tic()

# Get some raw Compustat data from funda
raw_funda <-
  comp.funda |>
  #Apply standard Compustat filters
  filter(indfmt=='INDL', datafmt=='STD', popsrc=='D' ,consol=='C') |>
  #Select the variables we want to download
  #the pattern for inline renaming is
  #new_name = old_name
  select(conm, gvkey, datadate, fyear, fyr, cstat_cusip=cusip, #inline renaming
         cik, cstat_ticker= tic, sich, ib, spi, at, xrd, ceq, sale,
         csho, prcc_f
  ) |>
  #Merge with the Compustat Company file for header SIC code and GICs code
  inner_join(select(comp.company, gvkey, sic, fic, gind), by="gvkey") |>
  #Use historical sic [sich] when available. Otherwise use header sic [sic]
  mutate(sic4 = coalesce(sich, as.numeric(sic))) |>
  #Calculate two digit sic code
  mutate(sic2 = floor(sic4/100)) |>
  #Delete financial and utility industries
  #For some research projects this is common to remove highly regulated firms
  #with unique accounting practices
  filter(!between(sic2,60,69),
         sic2 != 49) |>
  # replace missings with 0 for defined vars
  mutate(across(c(spi, xrd),
           ~ coalesce(., 0))) |>
```

```
  # create a few additional variables
  mutate(
    # Some example code to align the data in June calendar time.
    # Some papers use June of each year and assume a 3 month reporting lag.
    # Effectively this is coded as aligning datadate as of March each year.
    # See, for example, Hou, Van Dijk, and Zhang (2012 JAE) figure 1
    # This example also demonstrates injecting sql into dplyr code
    calyear = if_else( fyr > 3,
                       sql("extract(year from datadate)")+1,
                       sql("extract(year from datadate)")),
    # mve is market value of equity
    mve = csho * prcc_f,
    # define earnings (e) as earnings before special items
    e= ib-spi,
  ) |>
  # filter to fiscal years after 1955, not much in Compustat before that
  filter(1967 < fyear) |>
  # filter to US companies
  filter(fic=="USA") |>
  # everything above manupulates the data inside the WRDS postgres server
  #behind the scenes it generates efficient sql code
  # below line downloads to local machine RAM
  collect()
  #if you comment out the above collect() and instead run below command
  # you can see the behind the scenes sql
  #show_query()

#stop the tictoc timer
tictoc::toc()
```

```
2.19 sec elapsed
```

```
# Save the data to disk ------------------------------------------------------

# saving to Stata is convenient for working with coauthors
# glue package allows for dynamic file paths
# then each coauthor can specify their own local data folder
tic()
write_dta(raw_funda,glue("{data_path}/raw-data-R.dta"))
toc()
```

```
0.72 sec elapsed
```

```
#looks like about 162 MB on my machine

# if the data will stay in R or another advanced/modern language like Python
# then Parquet files are a nice open-source file format for data science
# they are fast and small and have some other advanced features as well

# in this example, we have customized the write_parquet function a bit to
# default to a high level of gzip compression to save space
# therefore, the write_parquet function is using the function defined in the
# utils script
tic()
write_parquet(raw_funda,glue("{data_path}/raw-data-R.parquet"))
toc()
```

```
1.92 sec elapsed
```

```
# the parquet operations are faster and the file is only 32MB on my machine
```

Then we play with transforming data. The R script below processes financial panel data by cleaning, transforming, and preparing it for analysis. It begins by loading necessary libraries and reading in raw data, applying filtering criteria to ensure data quality. Key financial variables are computed, including return on assets, R&D intensity, and industry classifications based on Fama-French groupings. To analyze earnings persistence, the script generates lead earnings variables while ensuring continuity in fiscal periods. Exploratory analysis is conducted to summarize industry-level characteristics and visualize loss frequencies. Winsorization is applied to key financial metrics to mitigate the influence of outliers. Finally, the processed dataset is saved in Stata format for further statistical modeling and analysis.

```
# Setup ------------------------------------------------------------------------

# Load Libraries [i.e., packages]
library(lubridate)
library(glue)
library(arrow)
library(haven)
library(tidyverse) # I like to load tidyverse last to avoid package conflicts



source("E:/acct_995_data/abr/-Global-Parameters.R")
```

```
[ ] Edit '.Renviron'.

[ ] Restart R for changes to take effect.
```

```
source("E:/acct_995_data/abr/utils.R")
```

```
set R formatting options

imported parquet functions

imported industry functions

imported transformation functions
```

```
# read in the data from the previous step -------------------------------------

#let's work with the parquet version
data1 <- read_parquet(glue("{data_path}/raw-data-R.parquet"))

#note: if you choose to collect your raw data in SAS or Stata
# these could easily be read in using haven::read_dta() or haven::read_sas()


# Some quick peeks at the data ------------------------------------------------

#since the data is structured as a dplyr tibble, just calling its name
#will preview the first 10 rows (similar to a head function)
data1
```

```
# A tibble: 302,836 x 25
   conm  gvkey datadate   fyear   fyr cstat_cusip cik   cstat_ticker  sich    ib
   <chr> <chr> <date>     <int> <int> <chr>       <chr> <chr>        <int> <dbl>
 1 A & ~ 0010~ 1968-12-31  1968    12 000032102   <NA>  AE.2            NA 0.347
 2 A & ~ 0010~ 1969-12-31  1969    12 000032102   <NA>  AE.2            NA 1.84
 3 A & ~ 0010~ 1970-12-31  1970    12 000032102   <NA>  AE.2            NA 1.88
 4 A & ~ 0010~ 1971-12-31  1971    12 000032102   <NA>  AE.2            NA 0.138
 5 A & ~ 0010~ 1972-12-31  1972    12 000032102   <NA>  AE.2            NA 1.55
 6 A & ~ 0010~ 1973-12-31  1973    12 000032102   <NA>  AE.2            NA 1.86
 7 A & ~ 0010~ 1974-12-31  1974    12 000032102   <NA>  AE.2            NA 1.56
 8 A & ~ 0010~ 1975-12-31  1975    12 000032102   <NA>  AE.2            NA 2.28
```

```
 9 A & ~ 0010~ 1976-12-31  1976    12 000032102   <NA>  AE.2            NA 3.43
10 A & ~ 0010~ 1977-12-31  1977    12 000032102   <NA>  AE.2            NA 1.93
# i 302,826 more rows
# i 15 more variables: spi <dbl>, at <dbl>, xrd <dbl>, ceq <dbl>, sale <dbl>,
#   csho <dbl>, prcc_f <dbl>, sic <chr>, fic <chr>, gind <chr>, sic4 <dbl>,
#   sic2 <dbl>, calyear <dbl>, mve <dbl>, e <dbl>
```

```
#can also glimpse
glimpse(data1)
```

```
Rows: 302,836
Columns: 25
$ conm         <chr> "A & E PLASTIK PAK INC", "A & E PLASTIK PAK INC", "A & E ~
$ gvkey        <chr> "001000", "001000", "001000", "001000", "001000", "001000~
$ datadate     <date> 1968-12-31, 1969-12-31, 1970-12-31, 1971-12-31, 1972-12-~
$ fyear        <int> 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 197~
$ fyr          <int> 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 1~
$ cstat_cusip  <chr> "000032102", "000032102", "000032102", "000032102", "0000~
$ cik          <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, "0000723576", "00~
$ cstat_ticker <chr> "AE.2", "AE.2", "AE.2", "AE.2", "AE.2", "AE.2", "AE.2", "~
$ sich         <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ ib           <dbl> 0.347, 1.835, 1.878, 0.138, 1.554, 1.863, 1.555, 2.284, 3~
$ spi          <dbl> 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, -0.580, 0.000, ~
$ at           <dbl> 5.922, 28.712, 33.450, 29.330, 19.907, 21.771, 25.638, 23~
$ xrd          <dbl> 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0~
$ ceq          <dbl> 2.571, 10.210, 10.544, 8.381, 7.021, 8.567, 9.843, 10.240~
$ sale         <dbl> 7.400, 37.392, 45.335, 47.033, 34.362, 37.750, 50.325, 51~
$ csho         <dbl> 0.372, 2.582, 2.446, 2.995, 2.902, 2.840, 2.150, 2.098, 2~
$ prcc_f       <dbl> NA, NA, 10.000, 5.750, 5.125, 1.750, 2.125, 4.375, 5.750,~
$ sic          <chr> "3089", "3089", "3089", "3089", "3089", "3089", "3089", "~
$ fic          <chr> "USA", "USA", "USA", "USA", "USA", "USA", "USA", "USA", "~
$ gind         <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, "253010", "253010~
$ sic4         <dbl> 3089, 3089, 3089, 3089, 3089, 3089, 3089, 3089, 3089, 308~
$ sic2         <dbl> 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 58, 58, 58, 58, 5~
$ calyear      <dbl> 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 197~
$ mve          <dbl> NA, NA, 24.460000, 17.221250, 14.872750, 4.970000, 4.5687~
$ e            <dbl> 0.347, 1.835, 1.878, 0.138, 1.554, 1.863, 2.135, 2.284, 3~
```

```
#or summarize
summary(data1)
```

```
     conm               gvkey              datadate             fyear
```

```
Length:302836      Length:302836      Min.   :1968-06-30   Min.   :1968
Class :character   Class :character   1st Qu.:1984-09-30   1st Qu.:1984
Mode  :character   Mode  :character   Median :1995-12-31   Median :1995
                                      Mean   :1996-05-11   Mean   :1995
                                      3rd Qu.:2007-06-30   3rd Qu.:2007
                                      Max.   :2025-01-31   Max.   :2024

      fyr          cstat_cusip            cik            cstat_ticker
Min.   : 1.000   Length:302836      Length:302836      Length:302836
1st Qu.: 6.000   Class :character   Class :character   Class :character
Median :12.000   Mode  :character   Mode  :character   Mode  :character
Mean   : 9.471
3rd Qu.:12.000
Max.   :12.000

      sich             ib                 spi                 at
Min.   : 100     Min.   :-56121.90   Min.   :-51066.20   Min.   :      0.0
1st Qu.:2842     1st Qu.:    -2.07   1st Qu.:    -0.24   1st Qu.:     11.7
Median :3823     Median :     0.65   Median :     0.00   Median :     62.7
Mean   :4549     Mean   :    73.48   Mean   :   -15.54   Mean   :   1655.3
3rd Qu.:5961     3rd Qu.:    10.73   3rd Qu.:     0.00   3rd Qu.:    396.0
Max.   :9998     Max.   :104821.00   Max.   :120517.00   Max.   :1153881.0
NA's   :106837   NA's   :18740                           NA's   :17557
      xrd               ceq               sale              csho
Min.   :   -0.65   Min.   :-86154.0   Min.   : -1965.0   Min.   :     0.0
1st Qu.:    0.00   1st Qu.:     3.0   1st Qu.:     9.4   1st Qu.:     2.9
Median :    0.00   Median :    23.1   Median :    63.1   Median :     9.7
Mean   :   34.19   Mean   :   570.8   Mean   :  1387.7   Mean   :    73.2
3rd Qu.:    2.11   3rd Qu.:   151.9   3rd Qu.:   406.7   3rd Qu.:    32.7
Max.   :88544.00   Max.   :649368.0   Max.   :680985.0   Max.   :500796.1
                   NA's   :18295      NA's   :18818      NA's   :21330
    prcc_f             sic                fic                gind
Min.   :     0.00   Length:302836      Length:302836      Length:302836
1st Qu.:     2.40   Class :character   Class :character   Class :character
Median :     8.62   Mode  :character   Mode  :character   Mode  :character
Mean   :    23.46
3rd Qu.:    21.88
Max.   :141600.00
NA's   :62081
      sic4            sic2           calyear           mve
Min.   : 100   Min.   : 1.00   Min.   :1969   Min.   :      0
1st Qu.:2836   1st Qu.:28.00   1st Qu.:1985   1st Qu.:     13
Median :3714   Median :37.00   Median :1996   Median :     66
```

```
 Mean   :4426   Mean   :43.83   Mean   :1996   Mean   :   2337
 3rd Qu.:5812   3rd Qu.:58.00   3rd Qu.:2008   3rd Qu.:    439
 Max.   :9998   Max.   :99.00   Max.   :2025   Max.   :3522211
                                               NA's   :64795
       e
 Min.   :-39656.00
 1st Qu.:    -1.38
 Median :     0.86
 Mean   :    90.04
 3rd Qu.:    13.01
 Max.   :100834.00
 NA's   :18740
```

```
# Manipulate a few variables ----------------------------------------------

#many of the below steps could be combined into one. They also could have been
#done on the WRDS server
#I just separate them for teaching purposes

data2 <- data1 |>
  #filter based on the global parameters for the sample period that we set in
  # the global-parameters script.
  filter(calyear >= beg_year,
         calyear <= end_year) |>
  #I am going to scale by total assets (at) so I am going to set a minimum at
  # to avoid small denominators
  filter(at >= 10) |>
  mutate(
    #use the FF utility functions to assign fama french industries
    FF12 = assign_FF12(sic4),
    ff12num = assign_FF12_num(sic4),
    FF49 = assign_FF49(sic4),
    ff49num = assign_FF49_num(sic4),
    # code a loss dummy, I like 1/0 but true/false is also fine
    loss = if_else(e < 0 , 1, 0),
    # scale e by ending total assets
    # FSA purists would probably use average total assets, but just an example
    roa = e / at ,
    # scale r&d by ending total assets
    rd = xrd / at
  ) |>
  # let's do an earnings persistence regression with lead earnings as y
```

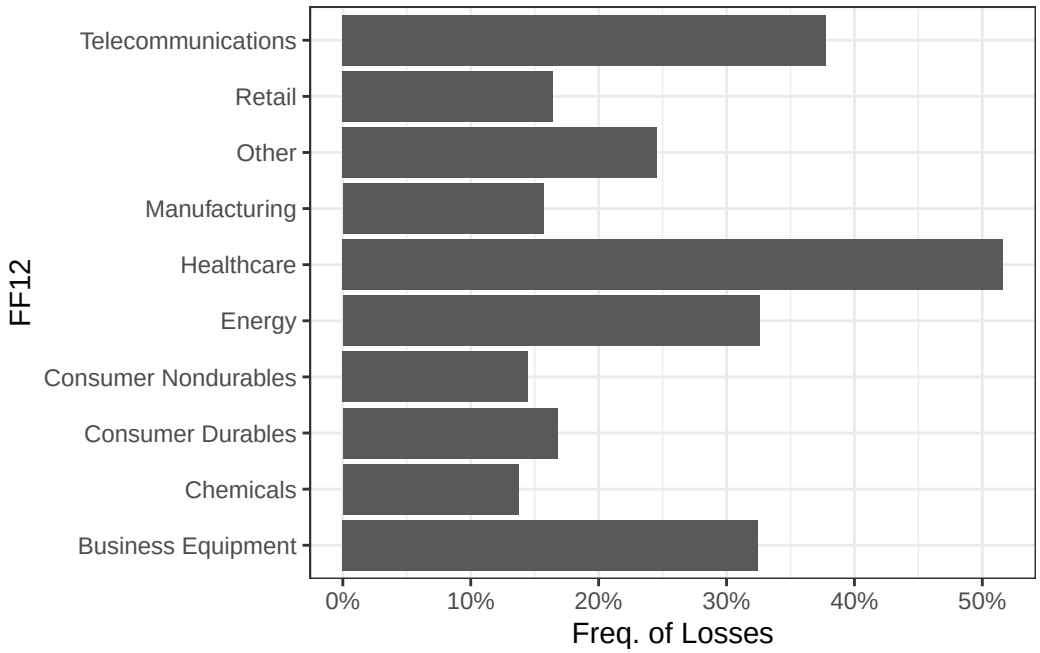
```
  # so for each gvkey we need the next earnings for that gvkey
  # first make sure the data is sorted properly
  arrange(gvkey,datadate) |>
  # then group by firm (gvkey)
  # this will restrict the lead function to only look at the next obs
  # for the same firm
  group_by(gvkey) |>
  mutate(roa_lead_1 = lead(roa,1L),
         datadate_lead_1 = lead(datadate,1L)) |>
  #check to make sure no gaps or fiscal year changes
  filter(month(datadate_lead_1) == month(datadate),
         year(datadate_lead_1) == year(datadate) + 1) |>
  #not a bad idea to ungroup once you are finished
  ungroup() |>
  #Filter multiple variables to require non-missing values
  filter(if_all(c(at, mve,rd,ff12num,starts_with("roa")), ~ !is.na(.x)))


# Play around ------------------------------------------------------------------

#how many observations in each FF12 industry?
data2 |>
  group_by(FF12) |>
  count()
```

```
# A tibble: 10 x 2
# Groups:   FF12 [10]
   FF12                    n
   <chr>               <int>
 1 Business Equipment  30461
 2 Chemicals            5303
 3 Consumer Durables    6201
 4 Consumer Nondurables 14146
 5 Energy               8802
 6 Healthcare          17835
 7 Manufacturing       27260
 8 Other               24743
 9 Retail              23568
10 Telecommunications   4979
```

```r
#percentage of losses by industry?
data2 |>
  group_by(FF12) |>
  summarize(pct_loss = sum(loss, na.rm = T)/n())
```

```
# A tibble: 10 x 2
   FF12                 pct_loss
   <chr>                   <dbl>
 1 Business Equipment      0.325
 2 Chemicals               0.137
 3 Consumer Durables       0.168
 4 Consumer Nondurables    0.145
 5 Energy                  0.326
 6 Healthcare              0.516
 7 Manufacturing           0.157
 8 Other                   0.246
 9 Retail                  0.164
10 Telecommunications      0.378
```

```r
#as a quick figure?
data2 |>
  group_by(FF12) |>
  summarize(pct_loss = sum(loss, na.rm = T)/n()) |>
  ggplot(aes(x = FF12, y= pct_loss)) +
  scale_y_continuous(name = "Freq. of Losses", labels = scales::percent) +
  geom_col() +
  coord_flip() +
  theme_bw()
```




```
# Winsorize the data ------------------------------------------------------

#check the tail values as an example
quantile(data2$roa, probs = c(0,.01,.99,1))
```

```
         0%          1%         99%        100%
-39.6179784  -0.8033140   0.2430401   2.5709079
```

```
#default winsorization
data3 <- data2 |>
  #default is 1% / 99 % , this winsorizes rd and all roa vars at that cut
  mutate(across(c(mve,at,rd,starts_with("roa")), winsorize_x))


#check the winsorized tail values
quantile(data3$roa, probs = c(0,.01,.99,1))
```

```
         0%          1%         99%        100%
-0.8033140 -0.8033083  0.2430382  0.2430401
```

```
#alternate version, if we want to change the tails
data3b <- data2 |>
  #winsorize 2.5% / 97.5 %
  mutate(
    across(c(rd,starts_with("roa")), ~ winsorize_x(.x,cuts = c(0.025,0.025)))
  )

#check
quantile(data2$roa, probs = c(0,.025,.975,1))
```

```
         0%        2.5%       97.5%        100%
-39.6179784  -0.5113552   0.1891310   2.5709079
```

```
quantile(data3b$roa, probs = c(0,.025,.975,1))
```

```
        0%       2.5%      97.5%       100%
-0.5113552 -0.5113345  0.1891293  0.1891310
```

```
# Save the winsorized data  ----------------------------------------------------

# just saving to Stata format this time for brevity
write_dta(data3,glue("{data_path}/regdata-R.dta"))
```

This is my first time working on a Quarto book. So, this first post will be very rough for now. I will try to provide a few different examples of ways to obtain and merge data in R, and a few tips of things to keep in mind.

We already know how to obtain data from WRDS. Let's use this to obtain some returns for the S&P 500. We could use the formal index data, but let's take a shortcut and just use the popular SPY ETF that tracks the S&P 500. To do this, we need to find the CRSP identifier (PERMNO) for the ticker "SPY." We can look in the WRDS stocknames file for this, and then use the SPY PERMNO to pull data from the CRSP monthly stock file.

```
# Load Libraries [i.e., packages]
library(dbplyr)
library(RPostgres)
library(DBI)
library(glue)
library(arrow)
library(haven)
```

```
library(tictoc) #very optional timer, mostly as a teaching example
library(tidyverse) # I like to load tidyverse last to avoid package conflicts

#I have done this in a separate chunk with the options
# results: FALSE
# message: FALSE
#because I don't need to see the messages from loading the packages.
```

```
# Log in to WRDS -----------------------------------------------------------

#before running this block, I used these commands to securely store my WRDS username and pass
# keyring::key_set("WRDS_user")
# keyring::key_set("WRDS_pw")

if(exists("wrds")){
  dbDisconnect(wrds)  # because otherwise WRDS might time out
}

wrds <- dbConnect(Postgres(),
                  host = 'wrds-pgdata.wharton.upenn.edu',
                  port = 9737,
                  user = keyring::key_get("WRDS_user"),
                  password = keyring::key_get("WRDS_pw"),
                  sslmode = 'require',
                  dbname = 'wrds')


# Create WRDS Table References ---------------------------------------------
crsp.msf <- tbl(wrds,in_schema("crsp","msf"))
stocknames <- tbl(wrds,in_schema("crsp","stocknames"))

#I am collecting this data locally to play with duplicates
spy_permnos <- stocknames |> filter(ticker == "SPY") |> collect()
```

Notice that there are six observations in the stocknames table that all share the same ticker "SPY." I am going to use this as a toy example to play with duplicates. My goal is for this data to be unique at the level of ticker-permno links. First, I can check whether this is true.

```
#check whether there are duplicates
#this simple logic is useful in general
#group by the level I want to make unique,
#count within each group
```

```
#sort by descending count so that if there are duplicates
#they will show up at the top.
spy_permnos |>
  group_by(ticker,permno) |>
  count() |>
  arrange(-n)
```

```
# A tibble: 3 x 3
# Groups:   ticker, permno [3]
  ticker permno     n
  <chr>   <int> <int>
1 SPY     84398     3
2 SPY     33910     1
3 SPY     60716     1
```

There are multiple permnos connected to the SPY ticker and some duplicate entries for permno 84398 so I better just look at the data. Also this tells me that there are only a few rows so it doesn't hurt to just print the data.

```
#| #note that we can use the kable commmand to embed a simple table in the quarto document
knitr::kable(spy_permnos)
```

permno	namedt	nameenddt	shrcd	exchcd	siccd	ncusip	ticker	comnam	shrcls	permco	hexcd	cusip	st_date	end_date	namedum
33910	1962-07-02	1966-05-24	10	2	2893		SPY	SPEEDRY CHEMICAL PRODS INC	A	2751	3	55914210	1962-07-02	1979-01-22	2
60716	1978-10-03	1987-07-01	10	1	3811	84756710	SPY	SPECTRA PHYSICS INC		4215	1	84756710	1972-12-14	1987-07-01	2
84398	1993-01-29	2009-02-23	73	2	6726	78462F10	SPY	SPDR TRUST		46699	4	78462F10	1993-01-29	2024-12-31	2
84398	2009-02-24	2010-01-26	73	4	6726	78462F10	SPY	SPDR TRUST		46699	4	78462F10	1993-01-29	2024-12-31	2

permno	namedt	nameenddt	shrcd	exchcd	siccd	ncusip	ticker	comnam	shrcls	permco	hexcd	cusip	st_date	end_date	nameendum
84398	2010-01-27	2024-12-31	73	4	6726	78462F10	SPY	SPDR S & P 500 E T F TRUST	46699	4	78462F10	1993-01-29	2024-12-31	2

Looking at the data, the company name for permno 84398 matches the SPDR S&P 500 ETF I am looking for. It looks like the duplicate entries might have to do with a change in the listing exchange for the ETF (exchcd) and then a slight name change in 2010 to make the name of the trust more descriptive. Let's keep using this toy example to demonstrate some other functions for dealing with duplicates:

```
#if I want to just collapse the duplicates, I can use "distinct" across the groups that I ca

spy_permnos |>
  select(ticker,permno) |>
  distinct()
```

```
# A tibble: 3 x 2
  ticker permno
  <chr>   <int>
1 SPY     33910
2 SPY     60716
3 SPY     84398
```

Now there are only three observations,which is what I asked for, but sometimes it might matter which of the duplicate observations I keep. For example, perhaps what I should do is keep the most recent observation from the spy_permno dataset, in terms of nameenddt.

```
#select the max data within each group as more advanced way to keep one obs per
#group
spy_permnos |>
  group_by(ticker,permno) |>
  filter(nameenddt==max(nameenddt))
```

```
# A tibble: 3 x 16
# Groups:   ticker, permno [3]
  permno namedt     nameenddt  shrcd exchcd siccd ncusip   ticker comnam  shrcls
   <int> <date>     <date>     <int>  <int> <int> <chr>    <chr>  <chr>   <chr>
1  33910 1962-07-02 1966-05-24    10      2  2893 <NA>     SPY    SPEEDR~ A
```

```
2  60716 1978-10-03 1987-07-01    10      1  3811 84756710 SPY    SPECTR~ <NA>
3  84398 2010-01-27 2024-12-31    73      4  6726 78462F10 SPY    SPDR S~ <NA>
# i 6 more variables: permco <int>, hexcd <int>, cusip <chr>, st_date <date>,
#   end_date <date>, namedum <dbl>
```

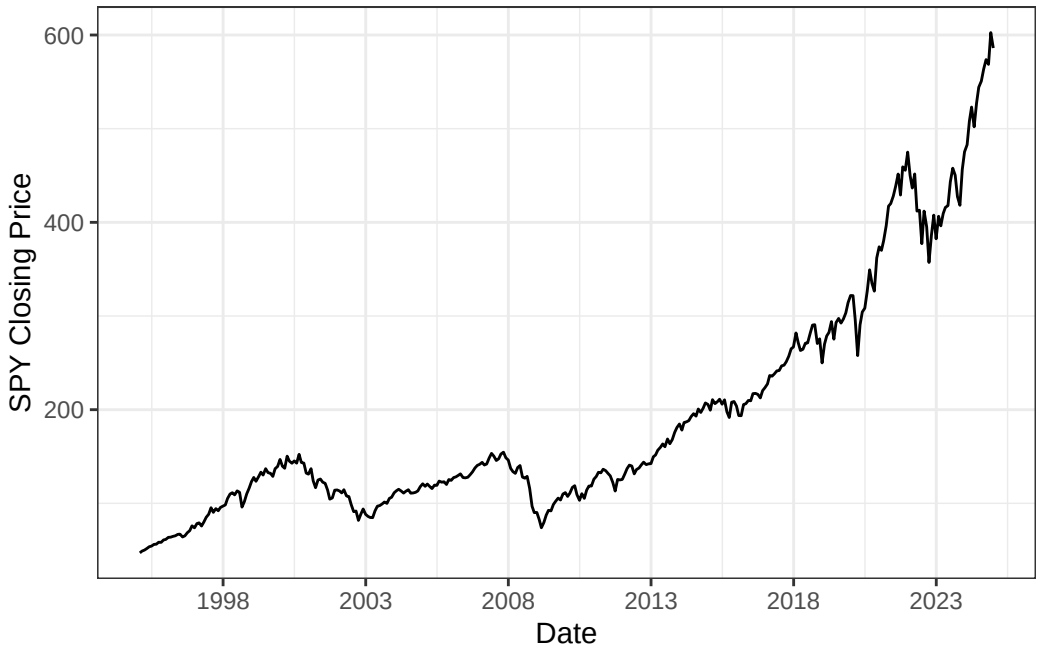
```
#ultimately we can assign the permno of the current observation, which we already know from r

spy_permno <- spy_permnos |>
  group_by(ticker,permno) |>
  filter(nameenddt==max(nameenddt)) |>
  ungroup() |>
  filter(nameenddt==max(nameenddt)) |>
  select(permno) |>
  as.numeric()

spy_permno
```

```
[1] 84398
```

Now we can use the SPY permno to pull monthly returns for SPY:

```
# Pull CRSP MSI Data -------------------------------------------------------

#Data seems to begin in feb 1993, lets start in 1995 as a nice round number
#notice that this implicitly feeds the permno I calculated locally back up to WRDS in my crsp
mkt_index <- crsp.msf |>
  filter(date >= "1995-01-01",
         permno == spy_permno) |>
  select(date,ret,prc) |>
  collect() |>
  mutate(month = month(date),
         year = year(date))
```

Then I can plot them, note that if you look at the source code for this page, I do this in a chunk with echo=false so that I only see the output and not the code. This would be useful for creating an actual paper rather than coding examples:



This plot would look nice with recessions shaded. We can get recession dates from FRED. FRED data can be accessed from an API, there is a custom package to work with FRED data in R called fredr. First you need to obtain a FRED API key by signing up here: https://fred.stlouisfed.org/docs/api/api_key.html

```
#load the fredr package
library(fredr)

#Unblock the below and run to set your password
#keyring::key_set("fred_api_key")

#set my API key which is saved in keyring
fredr_set_key(keyring::key_get("fred_api_key"))

#collect the data from the series USRECD
# https://fred.stlouisfed.org/series/USRECD

fred_data<-fredr(series_id = "USRECD",
                 observation_start = as.Date("1995-01-01"),
                 observation_end = as.Date("2024-12-31"),
                 frequency = "m") |>
  #I am going to add month and year variables because I think this is
  #easier for linking
```

```
  mutate(month = month(date),
         year = year(date))

# show the first few rows which has a value of 0 or 1 where 1 is recession
fred_data |> head() |> knitr::kable()
```

date	series_id	value	realtime_start	realtime_end	month	year
1995-01-01	USRECD	0	2025-03-21	2025-03-21	1	1995
1995-02-01	USRECD	0	2025-03-21	2025-03-21	2	1995
1995-03-01	USRECD	0	2025-03-21	2025-03-21	3	1995
1995-04-01	USRECD	0	2025-03-21	2025-03-21	4	1995
1995-05-01	USRECD	0	2025-03-21	2025-03-21	5	1995
1995-06-01	USRECD	0	2025-03-21	2025-03-21	6	1995

Now we need to merge the SPY data with the recession data.

```
merged_data <- mkt_index |>
  #I am going to select only the columns I need from   #the FRED data
  inner_join(fred_data |>
               select(month,year,recession=value),
             by=join_by(month,year))

# check to make sure it is still unique by month
merged_data |>
  group_by(month,year) |>
  count() |>
  arrange(-n)
```

```
# A tibble: 360 x 3
# Groups:   month, year [360]
   month  year     n
   <dbl> <dbl> <int>
 1     1  1995     1
 2     1  1996     1
 3     1  1997     1
 4     1  1998     1
 5     1  1999     1
 6     1  2000     1
 7     1  2001     1
 8     1  2002     1
```

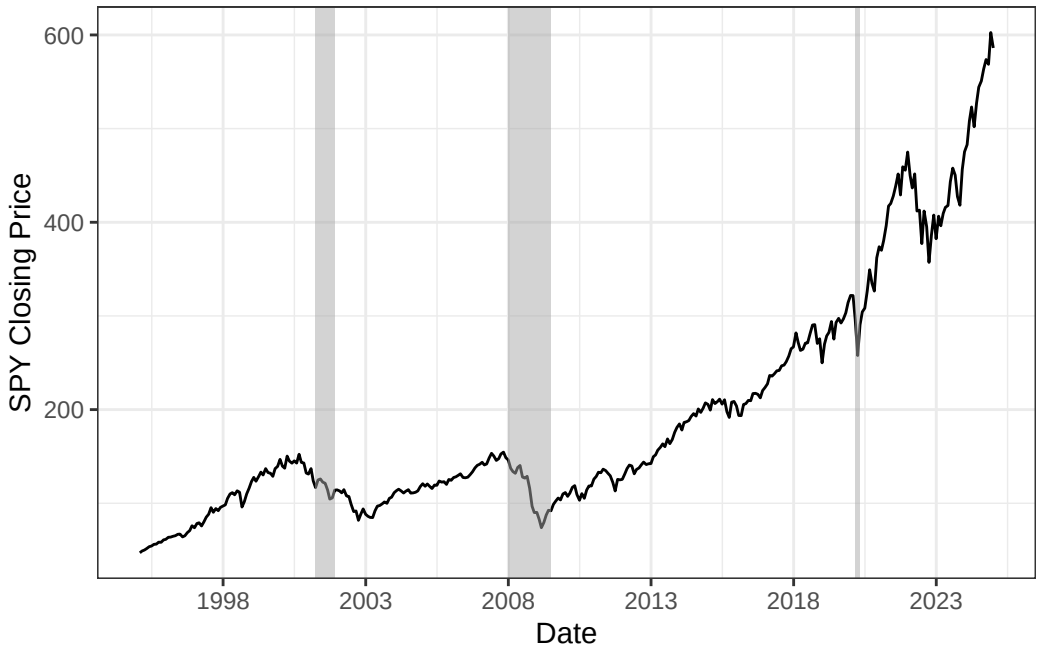
```
 9     1  2003     1
10     1  2004     1
# i 350 more rows
```

Now we can make the plot with shades for recession months

```
#turns out the merged data was not the preferred way to do this kind of plot




#here is some code I found online to reshape the recession data and add it to the plot

#rename/assign fred data to recession because
#that was the name in the example I found
recession<-fred_data

#load a package they used
library(ecm)

#reshape the recession data for the way
#geom_rect likes the data shaped
recession$diff<-recession$value-lagpad(recession$value,k=1)
  recession<-recession[!is.na(recession$diff),]
  recession.start<-recession[recession$diff==1,]$date
  recession.end<-recession[recession$diff==(-1),]$date

  if(length(recession.start)>length(recession.end))
  {recession.end<-c(recession.end,Sys.Date())}
  if(length(recession.end)>length(recession.start))
  {recession.start<-c(min(recession$date),recession.start)}

  recs<-as.data.frame(cbind(recession.start,recession.end))
  recs$recession.start<-as.Date(as.numeric(recs$recession.start),origin=as.Date("1970-01-01")
  recs$recession.end<-as.Date(recs$recession.end,origin=as.Date("1970-01-01"))

#look at the reshaped data
recs
```

```
  recession.start recession.end
1      2001-04-01    2001-12-01
2      2008-01-01    2009-07-01
3      2020-03-01    2020-05-01
```

```r
#plot the new plot with recession bars
merged_data |>
  ggplot(aes(x=date,y=abs(prc))) +
  geom_line() +
  scale_x_date(name = "Date",
               date_breaks= "5 years",
               date_labels = "%Y") +
  scale_y_continuous(name = "SPY Closing Price") +
  geom_rect(data=recs, inherit.aes=F,
                          aes(xmin=recession.start, xmax=recession.end, ymin=-Inf, ymax=+Inf)
                fill="darkgrey", alpha=0.5)+
  theme_bw()
```



Some other useful materials:

https://cran.r-project.org/web/packages/fredr/vignettes/fredr.html

https://iangow.github.io/far_book/web-data.html

https://iangow.github.io/far_book/identifiers.html

8 Merging Data (In Progress)

This chapter will go over how to merge in data from various sources. This will include examples of how to merge in firm-level data from various sources, and time-series data from FRED/Fama-French Factors/CRSP Index Files.

8.1 Common Sources of Firm-Level Data

Below is a table of common sources and firm identifiers that allow researchers to link data from different data sources:

Data Source	Identifiers	Other Firm Identifiers	Can Be Linked To	Notes
Compustat	GVKEY	Ticker, CIK	CRSP, Audit Analytics	
CRSP	PERMNO	Ticker, CUSIP	Compustat, IBES	To link to Compustat data, researchers should use the CRSP-Compustat link file for PERMNO-GVKEY mapping.
I/B/E/S	TICKER	OFTIC, CUSIP	CRSP	The "TICKER" variable is the I/B/E/S firm identifier **not** the trading symbol. The trading symbol is the "OFTIC" variable.
TAQ	SYMBOL			

Data Source	Identifiers	Other Firm Identifiers	Can Be Linked To	Notes
TRACE	CUSIP		CRSP	
Audit Analytics	CIK		Compustat	
XBRL	CIK		Compustat	
RavenPack	RP_ENTITY_ID	CUSIP	CRSP, IBES	

Blocking out for next attempt

9 Regression Tables

Test of embedding a regression in Quarto.

Table 9.1

	Base	No FE	Year FE	Two-Way FE	With Controls
ROA_t	0.839***	0.756***	0.769***	0.639***	0.624***
	(62.731)	(48.158)	(48.625)	(38.635)	(35.599)
$LOSS$		-0.030***	-0.028***	-0.015***	-0.017***
		(-7.949)	(-7.755)	(-7.560)	(-8.117)
$ROA_t \times LOSS$		0.032	0.012	-0.285***	-0.294***
		(1.470)	(0.535)	(-13.307)	(-12.620)
Year FE			X	X	X
Firm FE				X	X
Controls					X
N	163,298	163,298	163,298	161,635	161,635
R^2	0.594	0.597	0.603	0.707	0.707
R^2 Within			0.580	0.184	0.186

10 Summary

In summary, this book has no content whatsoever.

```
1 + 1
```

```
[1] 2
```

References

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.