HW-2 Tutorial: Creating a Stock Universe Database

February 17, 2025

1 Learning Outcomes

After completing the tutorial, students should be able to:

- Use R's `tidyverse`, `furrr`, and `NasdaqDataLink` packages to load, manipulate, and process financial data.
- Retrieve historical stock and market data in parallel while implementing robust error handling using `tryCatch`.
- Filter and validate the dataset by aligning stock data with the exact periods when companies were part of the S&P 500.
- Summarize raw insider trading data into meaningful daily indicators—such as the number of shares bought/sold and the count of unique insiders trading.
- Integrate supplementary market information (like market cap, P/E, and P/S ratios) to enhance the dataset for deeper analysis.

2 Introduction

This tutorial will guide you through the process of creating a historical database of stock data, specifically for the S&P 500 stocks, covering the last 10 years. The tutorial also includes adding financial sector information and summarizing insider trading data. The stock universe is constructed using data from the Sharadar Equities Prices dataset and supplemented with the Sharadar Core US Fundamentals Data dataset.

3 Data Collection

The first part of the task involves collecting stock data for the S&P 500. Adjusted open, high, low, close, and volume columns for each stock, covering the last 10 years are included from the data table.

3.1 Loading the Stock Symbols

The stock symbols are provided in a CSV file, "SP Tickers.csv," which contains the ticker symbols for all the companies in the S&P 500. The first step is to load these symbols into R.

```r
1  # Load necessary libraries
2  library(tidyverse)
3  library(furrr)
4  library(NasdaqDataLink)
5
6  # Set API key for Nasdaq Data Link
7  NasdaqDataLink.api_key("your-API-key")
8
9  # Read stock symbols from CSV file
10 symbols <- read_csv("SP Tickers.csv", col_names = c("symbol")) |> unique()
```

Lines 1-3: We load the required packages

- `tidyverse` is a collection of packages for data manipulation
- `furrr` is used for parallel processing
- `NasdaqDataLink` is used to fetch stock data from Nasdaq.

Line 7: The API key for Nasdaq Data Link is set to authenticate the requests.
Line 10 We read the stock symbols from the provided CSV file and remove duplicates, ensuring each symbol appears once.
Note: `|>` is called the **pipe** operator, that takes the output of one function and passes it into another function as the first argument, linking together the steps for data analysis with a clean syntax.

3.2 Fetching Stock Data

Next, we use the `future_map_dfr` function from the `furrr` package to fetch stock data for each symbol. The `NasdaqDataLink.datatable` function retrieves the historical data for each stock.

```r
 1 # Enable parallel processing
 2 plan(multisession, workers = 12)
 3 # Fetch stock data for each symbol
 4 stocks <- symbols$symbol |>
 5   future_map_dfr(function(symbol) {
 6     NasdaqDataLink.api_key("your-API-key")
 7     tryCatch(
 8       NasdaqDataLink.datatable(
 9         "SHARADAR/SEP",
10         ticker = symbol,
11         qopts.columns = c(
12           "ticker",
13           "date",
14           "open",
15           "high",
16           "low",
17           "close",
18           "volume"
19         )
20       ),
21       warning = function(w) {
22         print(w)
23         return(NULL)
24       },
25       error = function(e) {
26         print(e)
27         return(NULL)
28       },
29       finally = print(str_glue("Fetched data for {symbol}"))
30     )
31   }) |>
32   as_tibble()
```

Line 2: The `plan()` function is used to configure how parallel function calls from '`furrr`' are resolved. In this case, '`multisession`' will resolve the function in parallel using 12 R sessions running in the background on the same machine.
Lines 4-32: For each symbol, the '`tryCatch`' block ensures that any errors or warnings are

captured (e.g., if a symbol can't be fetched from the API), preventing the entire process from failing. The function '`NasdaqDataLink.datatable`' is called with the parameters that specify the columns to retrieve (open, high, low, close, volume). Then data is then stored as a 'tibble', `Tidyverse`'s analogue to a dataframe.

3.3 Excluding stocks not in the S&P 500

Next, we ensure that the data for each stock is only included for the period it was part of the S&P 500. This is done by filtering based on the dates provided in the "SP Additions.csv" and "SP Removals.csv" files.

```r
# Load additions data
additions <- read_csv(
  "SP Additions.csv",
  col_types = cols(
    symbol = col_character(),
    `date added` = col_date("%m/%d/%Y")
  )
)
# Load removals data
removals <- read_csv(
  "SP Removals.csv",
  col_types = cols(
    symbol = col_character(),
    `date removed` = col_date("%m/%d/%Y")
  )
)
```

Line 13-15: The 'SP Additions.csv' file contains the date each stock was added to the S&P 500.
Line 16-18: The 'SP Removals.csv' file contains the date each stock was removed from the S&P 500.

```r
# Filter stock data to only include dates
# when the stock was part of the S&P500
stocks <- stocks |>
  left_join(
    additions,
6      by = join_by(ticker == symbol),
7      relationship = "many-to-many"
8    ) |>
9    left_join(
10     removals,
11     by = join_by(ticker == symbol),
12     relationship = "many-to-many"
13   )
14
15 stocks <- stocks |>
16   filter(
17     date >= coalesce(`date added`, as_date("2000-12-31")) &
18     date < coalesce(`date removed`, as_date("2100-12-31"))
19   ) |>
20   select(ticker:volume) |>
21   arrange(ticker, desc(date))
```

Lines 2-12: After loading the additions and removals data, the `left_join` function combines this data with the stock data, matching rows in each table by ticker symbol.
Lines 14-20: Then, we filter the stock data to include only the period when the stock was in the S&P 500 by comparing each stock's 'date' with its 'date added' and 'date removed'. After filtering, the 'date added' and 'date removed' columns are discarded, and the data frame is sorted by ticker (ascending) and date (descending).

4 Q1: Incorporating Financial Sector Information

The next task is to add the financial sector (e.g., healthcare, energy) for each stock. This information is provided in the "sectors.csv" file.

```r
1 # Load sector data
2 sectors <- read_csv("sectors.csv")
3
4 # Join sector data with stock data
5 stocks <- stocks |> left_join(sectors, by = join_by(ticker == symbol))
```

Line 2: We load the sector data from the "sectors.csv" file, which contains the sector for each stock (e.g. "financial," "healthcare", etc.).
Line 5: The `left_join` function adds the sector information to the stock data by joining rows with the same ticker symbol.

5 Q2: Incorporating Insider Trading Data

The second task involves adding insider trading data for each stock and summarizing it into meaningful indicators. We fetch the insider trading data from Sharadar's “SF2" dataset.

```
1  # Fetch insider trading data for each stock
2  start_date <- "2015-01-01"
3  insider_data <- symbols$symbol |>
4    future_map_dfr(function(symbol) {
5      NasdaqDataLink.api_key("your-API-key")
6      tryCatch(
7        NasdaqDataLink.datatable(
8          "SHARADAR/SF2",
9          ticker = symbol,
10         filingdate.gte = start_date
11       ),
12       warning = function(w) {
13         print(w)
14         return(NULL)
15       },
16       error = function(e) {
17         print(e)
18         return(NULL)
19       },
20       finally = print(str_glue("Fetched insider data for {symbol}"))
21     )
22   }) |>
23   as_tibble()
```

Line 3-19: Insider trading data is fetched using ‘`NasdaqDataLink.datatable`’, with a filter to only include filings from January 1, 2015, onward (since we're only interested in the last 10 years of data).

5.1 Summarizing Insider Trading Data

The insider trading data is grouped by stock ticker and filing date. The goal is to summarize the data so that there is one observation per stock per day, containing meaningful indicators such as number of shares bought, sold by insiders, and the number of insiders involved in transactions.

```
1  # Group and summarize insider trading data
2  grouped_data <- insider_data |> group_by(ticker, filingdate)
3
4  insider_shares_sold <- grouped_data |>
5    filter(transactionshares < 0) |>
6    summarize(insider_shares_sold = sum(abs(transactionshares)))
7
8  insider_shares_bought <- grouped_data |>
9    filter(transactionshares >= 0) |>
10   summarize(insider_shares_bought = sum(transactionshares))
11
12 insiders_trading <- grouped_data |>
13   distinct(ownername) |>
14   summarize(insiders_trading = n())
```

The insider trading data is grouped by `ticker` and `filingdate`, ensuring that all transactions for a given stock on a specific date are processed together.

- **Lines 4-6**: The dataset is filtered to include only transactions where insiders sold shares (`transactionshares < 0`). The absolute values of shares sold are summed to calculate the total number of shares sold by insiders on that date.
- **Lines 8-10**: Similarly, transactions where insiders purchased shares (`transactionshares >= 0`) are filtered. The total number of shares bought by insiders on the given date is computed.
- **Lines 12-14**: The number of unique insiders involved in transactions for a given stock on a particular date is determined using `distinct(ownername)`. This calculates the number of different insiders trading on that day.

```
1  # Merge insider trading data into stock data
2  insider_summary <- insider_shares_sold |>
3    full_join(insider_shares_bought, by = join_by(ticker, filingdate)) |>
4    full_join(insiders_trading, by = join_by(ticker, filingdate))
5
6  stocks <- stocks |>
7    left_join(insider_summary, by = join_by(ticker, date == filingdate)) |>
8    mutate(across(where(is.numeric), ~coalesce(.x, 0)))
```

Lines 2-4: Merge all the insider trading data together into a single tibble.
Lines 6-8: Merge the insdier trading data together with the original stock data. We use the function '`colaesce`' to replace `NA` observations with 0.

6 Q3: Integrating the Daily Dataset from the Sharadar Core US Fundamentals Data

In this part of the assignment, we enhance our daily stock repository by merging additional data from Sharadar's Daily dataset. The goal is to enrich each stock–day observation with key market metrics—specifically, market capitalization (`marketcap`), price-to-earnings ratio (`pe`), and price-to-sales ratio (`ps`). Such metrics are valuable because:

- **Market Capitalization:** Provides an indication of a company's size and liquidity.
- **PE and PS Ratios:** Offer insight into valuation, helping to inform trading decisions and portfolio risk assessment.

By integrating these data points, we can perform more in-depth analyses such as valuation checks, liquidity assessments, or even develop more refined trading signals.

```r
# Fetch Data and Set the API key for Nasdaq Data Link
daily_data <- symbols$symbol |> future_map_dfr(function(symbol) {
  NasdaqDataLink.api_key("your-API-key")
```

- We begin with our list of stock symbols and then using `future_map_dfr`, we concurrently fetch each stock's daily data and merge the results into one data frame. We also set the API key inside each call, because the '`furrr`' package does not properly capture the API key from the global context.

```r
# Fetch daily data with error handling
tryCatch(
  # Retrieve specified columns from the Sharadar DAILY dataset
  NasdaqDataLink.datatable(
    "SHARADAR/DAILY",
    ticker = symbol,
    qopts.columns = c("ticker", "date", "marketcap", "pe", "ps")
  ),
  # Warning handler
10    warning = function(w) {
11      print(w)
12      return(NULL)
13    },
14    # Error handler
15    error = function(e) {
16      print(e)
17      return(NULL)
18    },
19    # Finally close and output conversion
20    finally = print(str_glue("Fetched daily data for {symbol}"))
21  )
22  }) |> as_tibble()
```

- **Lines 4-8** We retrieve the required columns from the Sharadar DAILY Dataset by defining them in the datatable.
- **Lines 10-13** A warning handler is defined in case of any warnings issued during execution and returns a NULL value
- **Lines 15-18** Similarly to handle errors in execution, we define an error handler to print out errors and return NULL value if found
- **Line 20** We print a confirmation message indicating that data for the current symbol has been processed. This is helpful for tracking progress and subsequently we convert the output into a tibble for further processing. Tibble format makes manipulation with tidyverse functions easier and more consistent.

```
# Merge fetched data
stocks <- stocks |> left_join(daily_data, by = join_by(ticker, date))
```

- Finally, we combine the fetched `daily_data` with our main `stocks` dataset using a left join on the `ticker` and `date` columns.

  This adds useful market information to our stock data. Now that we have `marketcap`, we can also calculate the number of shares by dividing it by the closing price.

  This shows how extra data from Sharadar can be used to make better trading decisions as required in the question.