How to: Dynamic Reports and Reproducible Research[⋆]

Toghrul Aghbabali[a], Wan Soo Choi[a], Taihun Im[a], Dominik Rösch[a,*], Prince Roy[a]

[a]*University at Buffalo*

July 17, 2025

Abstract

We propose a framework to generate dynamic reports and make reproducible research easier. Motivated by growing concerns about the lack of reproducibility and replicability, nowadays, most journals ask authors to post their programming code. While beneficial to others, these policies seem costly to the original authors. Using a combination of freely available tools, we illustrate our framework with a case study in asset pricing. We demonstrate how research can be conducted with replicability in mind, significantly lowering costs and increasing benefits for the original authors. Specifically, we replicate and extend the main findings of Hendershott, Livdan, and Rösch (2020).

Keywords: Replication, Reproducible Research
JEL: A11, C18, I23

[⋆]This work is based on a seminar on Reproducible Research, taught at the University at Buffalo by Dominik Rösch in Spring, 2025.

[*]Corresponding author

Email addresses: `toghrula@buffalo.edu` (Toghrul Aghbabali), `wansooch@buffalo.edu` (Wan Soo Choi), `taihunim@buffalo.edu` (Taihun Im), `drosch@buffalo.edu` (Dominik Rösch), `princero@buffalo.edu` (Prince Roy)

In recent years, several papers argue that many research findings fail to reproduce or replicate.[1] While this issue is unlikely to be the main driver of the broader anti-science movement—which questions both scientific findings and, by extension, the scientific method itself, i.e., the rigorous construction and testing of hypotheses—it does little to reassure science skeptics.

Recent changes, such as data and code policies at journals, are trying to address these issues and help other authors to easily reproduce the original study. And while Brodeur, Mikola, and Cook (2024) find a rather high ratio of 85% of papers that reproduce, they find that 25% of all papers have coding errors.

Clearly, other authors benefit from these policy changes, but the benefits to authors of the original paper are less clear and are most likely marginal compared to the substantial costs associated with providing code for wider distribution (Whited, 2023).

In this paper, we propose a framework that both lowers the costs and increases the benefits for authors to write reproducible research. This framework is based on several common tools, used within computer science and the industry for several decades. While we briefly discuss these tools, our focus is on demonstrating their use through a case study in asset-pricing. In particular, we replicate, reproduce, and extend the main findings of Hendershott et al. (2020).

While there seems no clear consensus of how to distinguish between reproducibility and replicability (Duvendack et al., 2017), and while many questions regarding reproducible research remain unanswered (Vilhuber, Schmutte, Michuda, and Connolly, 2023), a commonly used distinction between reproducibility and replicability is whether one merely tries to reproduce the original study using the original data and code or to replicate it using new data or code. A similar distinction is also used in Brodeur et al. (2024), Dreber and Johannesson (2025), and Committee on Reproducibility and Replicability in Science (2019).

Similarly, Jensen et al. (2023) distinguish between internal and external validity. The distinction between internal and external validity mirrors the difference between reproducibility and replicability but the former often relates to theoretical and economic justifications, while the latter tends to involve more practical concerns.

[1] Ioannidis (2005); Duvendack, Palmer-Jones, and Reed (2017); Hou, Xue, and Zhang (2020); Hasso, Brosnan, Ali, and Chai (2024). Yet, Jensen, Kelly, and Pedersen (2023) argue that at least the "majority of asset pricing factors ... can be replicated;".

Other resources exist that try to evaluate and improve reproducibility and replication success rates, such as Sharma, Sarstedt, Ringle, Cheah, Herfurth, and Hair (2024), Dreber and Johannesson (2025), and Christopher Gandrud (2020). But these often focus on theory or are general in nature. Christopher Gandrud (2020) proposes a similar framework to ours and presents the various tools in much more details. A different approach is suggested by Hellum, Jensen, Kelly, and Pedersen (2025) using "the power of the common task framework".

Our focus is on providing examples of how this framework can be used to write (or replicate) actual finance papers using a case study. While reproducible research is associated with costs, we believe that adopting a similar framework early on, eventually saves costs. We believe that having reproducibility in mind, when starting a new project, can break the cycle of publishing incentives leading scientists to rush, leading to lower quality work and less reproducibility, as outlined by Hill and Stein (2025). As such, we believe, learning the various tools and the proposed framework is an investment worth making.

The paper proceeds as follows. In Chapter 1 we introduce reproducible research. In particular, in Section 1.1, we examine the differences between reproducibility and replicability in greater detail and explore the reasons why papers may fail to achieve either. In Section 1.2 we discuss several tools often used to help write reproducible research. We discuss our replication results of Hendershott et al. (2020) including the code in Chapter 2. The necessary code to reproduce Chapter 2 is in Appendix 1.

Chapter 1: Introduction to Reproducible Research

1.1 Why are papers not reproducible?

Many studies document the lack of reproducibility or replicability in vast areas of science. For example, Ioannidis (2005) claims that (in medicine) "Most Published Research Findings Are False" and the Open Science Collaboration (2015) finds that only 39% of published papers in the top three psychology journals could be replicated. More recently, these concerns are also expressed in other fields, such as in Finance and Economics, see, e.g., Menkveld and 342 co-authors (2024).

Why do papers not reproduce or replicate? The literature suggests several answers: low power, p-hacking, publication bias, HARKing, data errors, or low prior probabilities.[2]

Except for data errors, these explanations refer to issues of external validity and attempts to replicate papers. Ultimately, researchers aim for replicability. But replicability requires more than just statistical reasoning and data properties. Low power and p-hacking, referring to the quality of the statistical tests employed in the paper, likely play a minor role compared to the issue of low prior probabilities.

Ultimately, research can be replicated if the findings are genuinely correct, requiring inferential reasoning and incorporating priors.

Researchers (and non-researchers) often seem surprised about the reasonably low replication rates. Conceptually, this should not come as a surprise. At least in the social sciences, it seems incredibly more difficult to come up with a hypothesis "that is both true and non-trivial" than one that is wrong or trivial, as demonstrated by Samuelson taking 30 years to come up with the "Ricardian theory of comparative advantage" as a possible example (Samuelson, 1966, p.683). It is particularly challenging to test research questions in the social sciences in which the system under investigation is reflexive and adjusts to any potential findings (Diane Coyle, 2021, p. 72). Albeit, the above hypothesis, given the same logic, is likely wrong (or trivial) itself.

We provide an example of the difficulties of moving from statistical properties to inferential probabilities in Section 1. In Section 2 we discuss why papers often do not reproduce.

1.1.1 External Validity

While researchers are ultimately interested in inferential probabilities—the probability of a hypothesis given the empirical findings—statistical analysis can only yield the opposite: the probability of findings given a hypothesis. While several papers highlight the important difference between both probabilities,[3] the implications regarding replicability of individual research projects is rarely discussed.

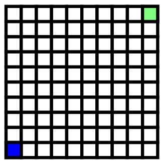
The following example should clarify the distinction between empirical tests, which

[2]see, among others, Ioannidis (2005), Dreber and Johannesson (2025), and Duvendack et al. (2017)
[3]see e.g., Harvey (2017) or Ziliak and McCloskey (2008).

can be incorporated into academic papers, and expert knowledge (priors), which is often mentioned implicitly but lies largely outside the scope of the authors and more within the domain of the academic community—such as journal editors and referees.

For example, if a market-maker is worried that returns for a specific stock are predictable, she could design a test for predictability. Assuming the test yields a positive result, what is the probability that the returns are actually predictable?

Figure 1 – Bayesian updating with low priors
This figure shows the results of 100 predictability tests (each represented by a square), assuming a base rate of predictability of 1 in 100. Each test correctly identifies predictable returns in 100 out of 100 cases (blue squares; lower left side) and incorrectly detects predictability in 1 out of 100 cases (green squares; upper right side). The ratio of truly predictable cases to positive test results is 1 to 2, indicating that a positive test implies predictability with a relatively low probability of just 50%.



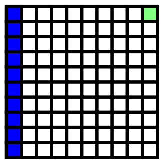
To answer this question, the market-maker needs to know the quality of the test in terms of its ability to avoid false positives and false negatives. In other words, she needs to know how many out of 100 tests correctly indicate a positive result if returns are predictable—the sensitivity of the test—and how many correctly indicate a negative result if returns are not predictable—the specificity of the test. For simplicity, let's assume that out of 100 tests, 100 correctly indicate a positive result if returns are predictable (i.e., the Type II error rate is 0%) and out of 100 tests, 99 correctly indicate a negative result if returns are not predictable (i.e., the Type I error rate is 1%).

Now, the most important question is, what is the base-rate, i.e., out of 100 (equivalent) stocks, how many have predictable returns? Lets assume that out of 100 stocks, only one has a predictable return. How would the test results look like? The one stock with predictable returns will be correctly identified with the test and the test will be positive. But out of the 99 other stocks, there will be around one other test which will be false positive. In this case, out of the 100 stocks two will test positive, but only one has predictable returns. The possibility that returns are predictable is one out of two, as indicated in Figure 1.

Unfortunately, the market-maker already lost significant amounts by providing liquidity in this stock, which increases the chance for having predictable returns. Let's assume instead that out of 100 stocks, 10 have predictable returns. As before, the 10 stocks with predictable returns will be correctly identified with the test and the test will be positive. And out of the 90 other stocks there will be around one other test which will be positive. In this case, out of the 100 stocks 11 will test positive, and 10 have predictable returns. The possibility that returns are predictable is 10 out of 11, as indicated in Figure 2.

This thought experiment shows two important aspects of any test, first, any test can err, second false positives are common when the base-rate is low, the so-called base rate fallacy. In other words, if a paper confirms well-accepted beliefs, it is likely to be correct, but it will change beliefs only marginally (as in Figure 2, where beliefs change by a factor of around 10, from 0.1 to around 1). On the other hand, if a paper is path-breaking, it will (or should) change beliefs significantly—but the results are still more likely to be incorrect (similar as in Figure 1, where beliefs change by a factor of 50, from 0.01 to 0.5).

Figure 2 – Bayesian updating with high priors
This figure shows the results of 100 predictability tests (each represented by a square), assuming a base-rate of predictability of 10 in 100 and that each test correctly identifies predictable returns in 100 out of 100 cases (blue squares; left side) and incorrectly detects predictability in 1 out of 100 cases (green square; upper, right side). The ratio of true predictability to positive tests is 10 to 11, indicating that a positive test indicates predictability with a high probability of almost 100%.



1.1.2 Internal Validity

External validity seems outside the scope of any individual paper and rather a goal of the whole literature.

On the other hand, internal validity depends on the reproducibility of individual papers. While reproducibility sounds like a rather low hurdle, in practice, often it is not. For example, the recent retraction at the Journal of Finance states that "the posted replication data and code do not reproduce key results" and that this is a "a major [reason]" for the retraction (Stefan Nagel, 2021). In general, retractions are increasing and are (or should be) about the lack of internal validity.

Why do papers not reproduce? In the strict sense, when the original authors make both data and all programming code available, papers that cannot be reproduced must contain programming errors. But up to a few years ago, open science and research transparency were uncommon and with it, data and code were rarely shared among researchers (Miguel, 2021). Even if code is shared, authors often do not share the code (or version number) of all statistical software and packages used in the analysis. Similarly, authors often do not make all data available, but rather refer to commonly used databases such as CRSP or TAQ. But these databases change over time; see, for example, the Internet Appendix of Brogaard, Ringgenberg, and Roesch (2025).

If code is not available, individual steps in the analysis must be inferred from the text of the paper. Given the many choices in any empirical analysis (often called the "garden of forking paths") it is unlikely that the reproducing/replicating authors take the same path as the original authors.[4] In other words, researchers that try to reproduce a paper, commonly use their own methodology, collect their own data and write their own code, blurring the lines between a true reproduction and a replication, which can explain the rather low reproducibility rate.

Working with large datasets, often from different vendors, over an extended period of many years, requires care and strong organizational skills. Luckily, many tools exist that can help researchers to write better code that is more easily reproduced. We will briefly discuss these tools in the next section and use them in our case study in Chapter 2.

1.2 Useful tools

What makes research reproducible? We propose three features that make papers or reports easier to reproduce. The paper (or report or website) ideally has been generated from only one file containing all text and code, which is under version control, and which is readable in a simple text editor.[5] The file:

- should include all text, figures, tables, and embedded programming code (ideally, based on a freely available programming language) to generate all results into one (PDF) document (e.g., using R knitr),
- should also include code to get the required data (e.g., from WRDS with ODBC),

[4]see, e.g., Menkveld and 342 co-authors (2024) or Mitton (2021).

[5]In larger projects it might be less feasible to only have one file, but having several files significantly decreases transparency and increases complexity, especially in making the project available to other researchers. For a discussion about how to handle projects with several files, we refer to Christopher Gandrud (2020).

- should also be in version control to allow syncing the file across platforms and to keep track of changes (e.g, Git and Github.com).

Overall, we believe that code should be easy to understand and transparent. But, researchers need to carefully evaluate advantages and disadvantages, for example, of using Latex versus various markup languages. The former adds complexity, but can create documents of publishable quality. Similar tradeoffs exist for writing own code at the risk of "reinventing the wheel" versus relying on publicly available packages. The former increases transparency, because the code is clearly visible, but the latter improves, e.g., re-usability. A guiding principle could be to rely on own code for anything core to the main contribution of the paper but use available packages when relying on commonly used techniques (like estimating regressions).

While this framework might seem impractical or even unrealistic, luckily, many tools have been developed to help and lower the costs to implement such a framework. In the following, we focus on one particular example for each tool—for example, using Git as a version control system. Many other version control systems exist, such as CVS or SVN, and we do not argue that our example is better than the others. While these tools are standard—especially in the industry and in computer science—and have been developed decades ago, the usage in Finance or Economics seems scant. Section 1.2 introduces these tools and a framework to help ensure replicability. Afterwards, in Chapter 2, we present a case study in asset pricing. We provide the underlying programming code in Appendix 3.1.

1.2.1 Version Control System: Git.

Version control systems allow programmers to keep track of changes when working with others or when working from different platforms. For example, a programmer might execute the code on her computer at home, at work, and on a server. To ensure that changes implemented on one platform propagate to the other platforms, it is common to use version control systems. As the name suggests, version control systems also allow tracking of textual changes. Another potential solution for this is to only store the files on a server, e.g., using Dropbox.

The main advantage of version control systems is that they also prevent conflicts and can help to merge "simultaneous" changes, when the same file has been modified simultaneously, but in different locations.

A popular Version Control System is Git accessed via Github.com, as explained on Wikipedia (2025):

> GitHub ... is a proprietary developer platform that allows developers to create, store, manage, and share their code. It uses Git to provide distributed version control and GitHub itself provides access control, bug tracking, software feature requests, task management, continuous integration, and wikis for every project.

We refer to the many excellent books and freely available resources to learn more about Git, Gihub, and Version Control Systems, more generally.

1.2.2 Databases and ODBC: PostgreSQL.

Often researchers are interested in aggregating data, for example, estimating the number of trades for all NYSE-listed stocks over a specific timeframe. These kind of database operations are best done with database queries (or stored procedure) that wholly execute on the database. One could download all trades, like:

```
in_stock_ticker <- c("IBM", "GME")
in_stock_ticker_list <- paste0("('", paste(in_stock_ticker, collapse = "', '"),
    "')")

query <- sprintf("SELECT date, date_trunc('minute', time_m) AS time,
                              CONCAT(sym_root, '_', sym_suffix) AS symbol,
                              price, tr_scond
                FROM
                    taqmsec.ctm_%s
                WHERE sym_root IN %s", in_date_yyyymmdd,
                    in_stock_ticker_list)
```

But that would require substantial bandwith and computer resources.

Instead, one can aggregate the data within the database:

```
query <- sprintf("SELECT date,
                            CONCAT(sym_root, '_', sym_suffix) AS symbol,
                            count(*) as trades
```

```
                FROM
                    taqmsec.ctm_%s
                WHERE sym_root IN %s group by date, symbol",
                    in_date_yyyymmdd, in_stock_ticker_list)
```

Databases provide their own programming language (SQL) and can be very powerful. For example, the following estimates the last trade price for each minute:

```
query <- sprintf("WITH minute_data AS (
                SELECT date, date_trunc('minute', time_m) AS time,
                      CONCAT(sym_root, '_', sym_suffix) AS symbol, price,
                          tr_scond
                FROM
                    taqmsec.ctm_%s
                WHERE (ex = 'N' OR ex = 'T' OR ex = 'Q' OR ex = 'A') AND
                    sym_root IN %s
                )
                SELECT date, time, symbol,
                      split_part(MAX(CONCAT(time, '_', price)), '_', -1) AS
                          close
                FROM
                    minute_data
                GROUP BY
                    date, time, symbol
                ORDER BY
                    symbol, date, time;", in_date_yyyymmdd,
                        in_stock_ticker_list)
```

The examples above refer to TAQ data accessed from WRDS. In the case-study presented in Chapter 2, we provide fully working examples of how to retrieve data from WRDS using ODBC, including how to set up the initial connection to the database.

Note that when working with large datasets—hundreds of terabytes of data—these aggregations often cannot be done on the fly. In this case, one needs to preprocess the data, for example, using GNU parallel (Tange, 2011) to aggregate data, e.g., for one stock-day per process, store the results in a database, and later retrieve these intermediate results for further analysis. This also often requires a better database, specifically designed for storing and analyzing tick-by-tick prices, such as Onetick. Note, that any intermediate analysis makes reproducible research much more difficult to achieve. For example, when running millions of processes to estimate stock-day variables, for various reasons, it may happen that not all processes end successfully. In this case, the number of stock-day observations might vary each time they are generated. Of course, one can avoid these issues by carefully logging start and end times of all processes and restarting all processes without an end time.

1.2.3 Dynamically analyze data: KnitR.

The idea of programs as literature, or "Literate Programming", is almost as old as programming itself and was proposed by Donal Knuth in 1984 (Knuth, 1984). In recent years, with the emphasis on reproducibility, it has gained further traction. Several tools exist to support this. We will rely on "knitR", which allows us to seamlessly integrate R (and Python) code with Latex.

1.2.4 RStudio to integrate

RStudio is a great software developing environment (IDE) for programming in R or in Python. It is free, open source, and has built-in support for many of the tools mentioned before, such as for version control and for using knitR or R-markdown documents. Several cloud providers (such as WRDS) offer access to RStudio, allowing clients to access RStudio from their web browser.[6] An alternative, with a focus on writing Latex code, is Overleaf. Overleaf also has built-in support for version control systems like Git and allows the execution of R-code.[7]

Note that by default, RStudio compiles PDFs using Sweave. We are using 'kintr' instead. The compiler can be changed in the settings: "Tools", "Project Options", "Sweave", and "Weave Rnw files using" setting to "knitr".

In the next Chapter, we use these tools to provide a case study of how research can be conducted with reproducibility in mind.

[6]RStudio at WRDS: `https://wrds-rstudio.wharton.upenn.edu`.

[7]See overleaf.com/learn/latex/Knitr

Chapter 2: Replication of Hendershott et al. (2020)

The Capital Asset Pricing Model (CAPM) developed by Lintner, Sharpe, and Treynor in the 1960's is an equilibrium model explaining asset prices. Till now, it is arguably the most influential theoretical model in asset pricing. Yet, since its beginnings the empirical failures in confirming the main conclusion of the CAPM—the idea that investors should be rewarded for taking undiversifiable risk (Beta)—gives rise to the so called Betting-Against-Beta anomaly.

While Hendershott et al. (2020) show that the Betting-Against-Beta anomaly largely disappears when using returns estimated over the night (close-to-open), till now, it is not clear what explains these patterns (Barroso, Detzel, and Maio, 2025).

In the following, we discuss the code to replicate the main results of Hendershott et al. (2020). Using the code provided in Appendix 1, it is straightforward to apply the same analysis in a different context. We can just copy and paste the code and then, verify that the results are robust in a more recent period, by changing:

```
    date_start <- "1992-01-01"
    date_end <- "2016-12-31"
```

to

```
    date_start <- "2017-01-01"
    date_end <- "2024-12-31"
```

Or we can verify that the results are robust for foreign stocks, by changing:

```
get_crsp_data <- function(wrds, start_date, end_date) {
    sql_query <- sprintf(" ... AND b.shrcd IN ('10', '11')" ...)
}
```

to

```
get_crsp_data <- function(wrds, start_date, end_date) {
    sql_query <- sprintf(" ... AND b.shrcd IN ('12', '30', '31')" ...)
}
```

After we change the code, we can click on "Compile PDF" in RStudio and retrieve a PDF similar to the one shown starting in the next section. When starting a new analysis, i.e., using a different timeframe or different assets, it is best to set the data directory to a newly created, empty directory:

```
data_dir <- "[NEWLY_CREATED_EMPTY_DIRECTORY]"
```

Note that if the generated PDF ends with a red code block, it indicates an error in the code of this block.

Figure 3 – US day and night returns for beta-sorted portfolios (2017–2024)
This figure shows average (equal-weighted) daily returns in percent against market betas for ten beta-sorted portfolios of all US publicly-listed common stocks. Portfolios are formed every month, with stocks sorted according to beta, estimated using daily night returns over a one-year rolling window. Portfolio returns are averaged, and post ranking betas are estimated over the whole sample. Each day, returns are measured during the day, from open-to-close (red), and during the night, from close-to-open (cyan). For both ways of measuring returns, a line is fit using ordinary least squares estimates. Data are from CRSP.

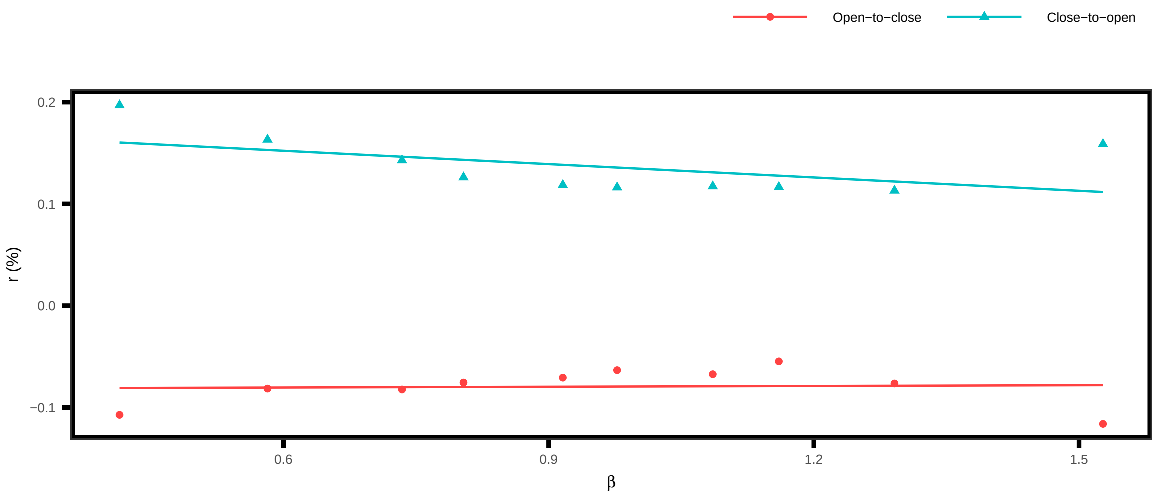
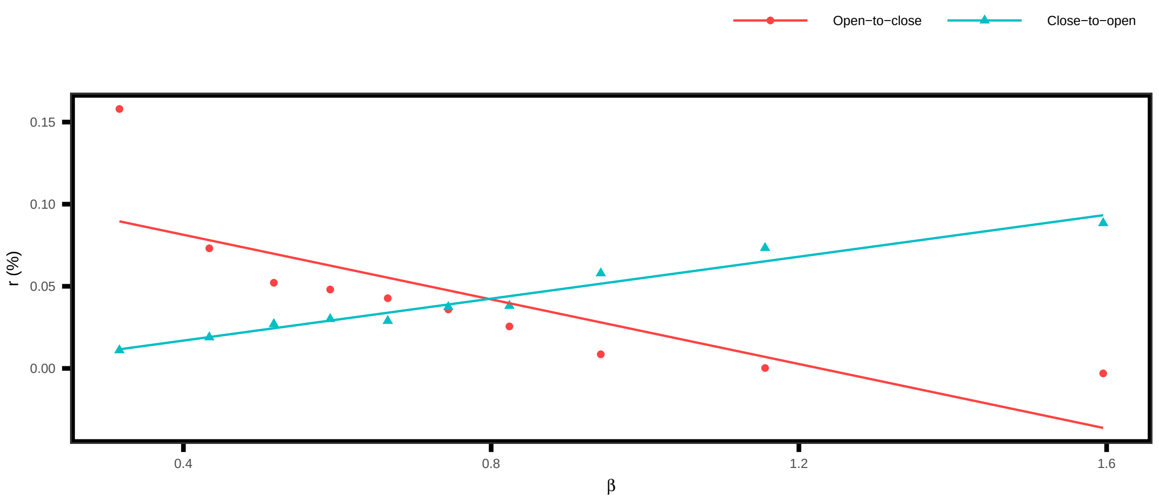


Figure 4 – Foreign stocks trading in the US day and night returns for beta-sorted portfolios (1992–2016)
This figure shows average (equal-weighted) daily returns in percent against market betas for ten beta-sorted portfolios of foreign stocks publicly-listed in the US. Portfolios are formed every month, with stocks sorted according to beta, estimated using daily night returns over a one-year rolling window. Portfolio returns are averaged, and post ranking betas are estimated over the whole sample. Each day, returns are measured during the day, from open-to-close (red), and during the night, from close-to-open (cyan). For both ways of measuring returns, a line is fit using ordinary least squares estimates. Data are from CRSP.



Figures 3 and 4 show the results of estimating the security market line as described in the following sections, but with the changes mentioned above. Figure 3 indicates that the security market line looks quite different from the estimates in Hendershott et al. (2020), potentially because of the pandemic starting in 2020. Figure 4 indicates that results in Hendershott et al. (2020) are robust when using foreign stocks.

2.1 Setup

Before we run any code, we set up an error handler. This ensures that if the code breaks at any point, e.g., because the working directory is not yet set correctly, the output will stop and display the error message at the end of the document.

```r
# see https://stackoverflow.com/q/74097101
knitr::knit_hooks$set(error = function(x, options) {
  ERROR_GLOBAL <<- x
  knitr::knit_exit()
})

knit_chunk <- knitr::knit_hooks$get("chunk")

knitr::knit_hooks$set(chunk = function(x, options){

  out <- x
  if (exists('ERROR_GLOBAL', inherits = TRUE)) {
    out <- paste0(out,'\n**stopped with error**:',ERROR_GLOBAL,'\n\n')
    options$size       <- "large"
    options$background <- '#FFDDDD'
    options$tidy       <- 'T'
    options$tidy.opts  <- list(width.cutoff=50)
    options$results    <-'markup'
  }

  knit_chunk(out, options)

})
```

2.1.1 Prerequisites

In this section, we check whether the system can execute the analysis.

First, we ensure that the system has enough free memory to successfully run the analysis. Unfortunately, retrieving available memory is not straightforward and depends on factors such as the operating system and whether the code is running on a virtual machine. Because of this, we do not provide any actual code but instead ask users to manually verify that enough RAM is available. A lack of RAM often explains sudden termination without generating any output files.

```
#to do - we need to have 50GB
```

Next, we load all required R libraries. While this must be done at the beginning of the script, we evaluate the code here—without displaying it—to avoid disrupting the reading flow. The full code is shown at the end of the file, Section 4.

```
# The actual code is executed here, but shown only at the end of the file.
```

```
## [1] TRUE
```

Similarly, we ensure that we compile the document using tinytex and that all required Latex packages are available. If any package is missing we try to install it:

```
if(! tinytex::is_tinytex()) {
  stop("We use tinytex to install missing Latex packages, if you
  are using a different Latex version this might not be compatible.
  It might be easiest to run tinytex::install_tinytex()")
}

# if(! grepl("TinyTeX", Sys.which("pdflatex"))) {
# stop("It looks like you are using a different Latex version
# which might not be compatible.
# It might be easiest to switch to TinyTex, for example,
# by adding the PATH to pdflatex in TinyTex directory
# to PATH environmental variable. See the code in the Rnw file below.")
#
#   cat('Sys.setenv(PATH = paste0(tinytex::tinytex_root(),
#"\\bin\\windows;", Sys.getenv("PATH")))\n',
#     file = file.path(Sys.getenv("HOME"), ".Rprofile"),
#     append = TRUE)
# }
```

```r
latex_packages <- c('listings', 'caption', 'mathtools', 'floatrow',
                    'setspace', 'cmap', 'filecontents')

for (pkg in latex_packages) {
  tinytex::check_installed(pkg) || tinytex::tlmgr_install(pkg)
}
```

2.1.2 Setup variable names

This section contains various code blocks to set up the analysis such as variables determining how the compiled PDF will look (e.g., whether it will contain the underlying R code); environment variables that need to be adjusted (like where to store the data); and variables related to the analysis (such as start and end dates).

- First, the code contains variables that determine how the PDF will look, e.g., with ("echo=T") or without the underlying R code ("echo=F".)

  ```r
  knitr::opts_knit$set(progress = TRUE, verbose = TRUE)
  knitr::opts_knit$set(self.contained=T)

  # cache.lazy = F (helps error when caching large data sets)
  knitr::opts_chunk$set(echo=T,warning=F,message=F, error=T, tidy=T,
  tidy.opts=list(width.cutoff=60), results='markup',cache.lazy = FALSE,
  cache.rebuild=F)
  ```

- Second, the code sets important environment variables (which need to be adjusted by the user):

  ```r
  setwd("~/repositories/3r/")

  data_dir <- "/projects/academic/drosch/3r/"
  # data_dir <- '/projects/academic/drosch/3r/recent'
  # data_dir <- '/projects/academic/drosch/3r/adrs'
  ```

```
if (!file.exists(data_dir)) {
    stop(sprintf("Could not find data_dir '%s' to store data,
                 first create this directory.",
        data_dir))
}
```

- Third, we define variables specific to the analysis (which can be adjusted by the user):

```
date_start <- "1992-01-01"
date_end <- "2016-12-31"

# date_start <- '2017-01-01' date_end <- '2024-12-31'
```

- Fourth, we set up the username and password to download all required data from WRDS. We suggest two alternatives and opt for the second. As explained on the WRDS website[8] we can store the username and password in the following file (depending on the Operating System)

  - %APPDATA%/postgresql/pgpass.conf
  - /.pgpass

  and in the following way:

  - wrds-pgdata.wharton.upenn.edu:9737:wrds:wrds_username:wrds_password

```
# password_file_name <- '~/.pgpass'
# if(!file.exists(password_file_name)) {
# stop('Setup_wrds_password needs to run manually in
# Console to setup Your WRDS password:')
```

[8]https://wrds-www.wharton.upenn.edu/pages/support/programming-wrds/programming-r/r-from-your-computer/

```
# cat(paste0('wrds-pgdata.wharton.upenn.edu:9737:wrds:',
# wrds_username, ':', wrds_password),
# file=password_file_name, append=TRUE, sep = '\n') }
```

Instead, we use the R-package "keyring" to safely store the username and password. To do either, you need to execute the code directly in the console to set either the file (see code chunk above) or the keyring (see code below, within tryCatch).

```
tryCatch({
    keyring_create("credentials_wrds", password = "letmein")
    key_set("wrds_username", keyring = "credentials_wrds", prompt = "Your WRDS us
    key_set("wrds_password", keyring = "credentials_wrds", prompt = "Your WRDS pa
}, error = function(e) {
})

## are you sure you want to overwrite ~/.config/r-keyring/credentials_wrds.keyrin
## NULL

keyring_unlock("credentials_wrds", password = "letmein")
```

- Last, we establish a connection to WRDS. If required data cannot be found, we can automatically download the data from WRDS.

```
postregs <- ifelse(exists("Postgres"), Postgres, PostgreSQL)

tryCatch({
    wrds <- dbConnect(postregs(), host = "wrds-pgdata.wharton.upenn.edu",
        port = 9737, dbname = "wrds", user = key_get("wrds_username",
            keyring = "credentials_wrds"), password = key_get("wrds_password",
            keyring = "credentials_wrds"))
}, error = function(e) {
    stop("It is likely that wrds_username and
    wrds_password are not stored in keyring.
    For that, run code in setup_wrds_password directly
    in R console. An error occurred: ",
        e$message)
})
```

2.1.3 Download and Prepare Required Data

If the data are found in the previously defined data directory, read them; otherwise, download them. First, we define a function to download the stock data from WRDS.

```
get_stock_returns <- function(wrds, date_start, date_end) {
    sql_query <- sprintf("SELECT a.cusip, a.permno, a.date, a.prc, a.vol,
           a.ret, a.openprc, a.askhi, a.bidlo, b.ticker, a.shrout, b.shrcd
      FROM crsp.dsf AS a, crsp.dsenames AS b
      WHERE a.permno = b.permno
      AND a.date BETWEEN b.namedt AND b.nameendt
      AND a.date >= '%s' AND a.date <= '%s'
      AND b.shrcd IN ('10', '11')",
        date_start, date_end)
    crsp_data <- dbGetQuery(wrds, sql_query) %>%
        as.data.table()
    return(crsp_data)
}
```

Then we call the function above in case the relevant data file cannot be found.

```
data_file_name <- paste(data_dir, "data_wrds_crsp_complete.csv",
    sep = "/")

if (!file.exists(data_file_name)) {
    data_sample <- get_stock_returns(wrds, date_start, date_end)
    fwrite(data_sample, file = data_file_name)
}
```

```
data_sample <- fread(data_file_name)
```

We then define variables as in the reference paper. In particular, we estimate night returns from 24-hour, corporate action-adjusted returns. This ensures that our night returns are also calculated from corporate action adjusted prices:

```
data_sample[, `:=`(prc, abs(prc))]
data_sample[, `:=`(mcap, prc * shrout)]
data_sample[, `:=`(return_day, prc/openprc - 1)]
data_sample[, `:=`(return_night, (1 + ret)/(1 + return_day) -
    1)]
data_sample[, `:=`(date, as.Date(date))]
data_sample[, `:=`(yyyy, year(date))]
data_sample[, `:=`(mm, month(date))]
```

2.1.4 Filter and scale the data according to the reference paper

As in the reference paper, we drop stock-days without an opening price and stock-days with a daily return above 10.

```
data_sample <- data_sample[!is.na(openprc)]

stocks_drop_large_day_return <- count(data_sample[return_day >=
    10])

data_sample <- data_sample[return_day < 10]
```

Hendershott et al. (2020) drop 16 stock-days with a day return over 1,000%; in our sample, we drop 16 stock-days.

To ensure that we restrict the data to dates between ‘date_start’ and ‘date_end’, we filter accordingly. Note that if the data files contain fewer observations, the actual date range will be limited by the contents of the files. We test for this at the end of the script. In general, and to avoid such cases, it is best to set ‘data_dir’ to a new, empty directory when starting a new analysis.

```r
data_sample <- data_sample[date >= date_start & date <= date_end]
```

For readability, we measure returns in percent:

```r
data_sample[, `:=`(return_day, return_day * 100)]
data_sample[, `:=`(return_night, return_night * 100)]
```

2.1.5 Compute required variables

After filtering, we compute other variables that we need for the later analysis. Note that, in general, these variables depend on whether we construct them before or after we filter our sample. For each month and each stock, we compute the market capitalization based on the last trading day the month. We then use the market capitalization from the previous month to compute value-weighted portfolio returns:

```r
setorder(data_sample, permno, date)
data_sample[, `:=`(date_mktcap, .I == .I[.N]), by = .(permno,
    yyyy, mm)]

data_sample[date_mktcap == T, `:=`(mcap_mend, last(mcap)), by = .(permno,
    yyyy, mm)]
data_sample[, `:=`(mcap_mend_lag, shift(mcap_mend)), by = .(permno)]

data_sample[, `:=`(mcap_mend_lag, nafill(mcap_mend_lag, type = "locf")),
    by = .(permno)]
```

and we compute the market return. Following, Hendershott et al. (2020) “the market index is constructed as the value-weighted portfolio”. We use this market return to estimate how sensitive stock night-returns are to market night-returns, called the Beta of the stock.

```r
# data_sample[, return_market := mean(return_night, na.rm =
# TRUE), by = .(date)]
data_sample[, `:=`(return_market, weighted.mean(return_night,
    mcap/sum(mcap), na.rm = T)), by = .(date)]
```

We often need to compute market betas using a rolling window. For reusability, we define a function to do that:

```r
compute_rolling_beta <- function(data, y = "return_night",
                                 x ="return_market") {
  setorder(data, date)
  data$beta <- roll_lm(
    x = data[[x]],
    y = data[[y]],
    width = 252, # Approx. 12 months of trading days
    min_obs = 30 # Minimum of ~30 days of data to compute beta
  )$coef[, 2]

  return(data)
}
```

and then call the function to estimate betas. Because calculating betas is time consuming, we cache the results. Instead of writing the results to a file (as we do with the stock return data), we cache the code block. Beta estimations depend on previous blocks, such as how we filter the data. To ensure we recalculate betas whenever necessary, we use “dependson” in the code block. If any of the code blocks listed in “dependson” change, we will recalculate betas.

```r
data_beta <- data_sample[, compute_rolling_beta(.SD), by = permno][,
    c("permno", "date", "beta")]
```

To minimize the data cached when computing betas, we dropped all other data. We now merge back the rest of the data:

```r
data_sample <- merge(data_sample, data_beta, by = c("permno",
    "date"), all.x = TRUE)
```

As in Hendershott et al. (2020), we estimate monthly betas. We use the equally weighted average of the daily betas within each stock-month. The following function adds a new column to input “data” containing the average beta from the previous month:

```r
add_monthly_beta <- function(data, firm_id) {
    data_beta_aggregate <- data[, .(beta_month = mean(beta, na.rm = TRUE)),
        by = c(firm_id, "yyyy", "mm")]

    data_beta_aggregate <- data_beta_aggregate[!is.na(beta_month)]

    setorderv(data_beta_aggregate, cols = c(firm_id, "yyyy",
        "mm"))
    data_beta_aggregate[, `:=`(beta_month_lag, shift(beta_month)),
        by = c(firm_id)]
    data_beta_aggregate[, `:=`(beta_month, NULL)]

    data <- merge(data, data_beta_aggregate, by = c(firm_id,
        "yyyy", "mm"), all.x = TRUE)

    return(data)
}

data_sample <- add_monthly_beta(data_sample, firm_id = "permno")
```

2.2 Beta portfolios

One of the main predictions of the CAPM is that stocks with higher (undiversifiable) risk (beta) have higher expected returns. To test this, we look at the slope of the Security-Market Line (SML), i.e., the straight line that crosses the risk-free rate and the “market”. Empirically, we estimate the following regression:

$$Ret_{i,t} = \alpha + \beta PostBeta_{i,t} + \epsilon_{i,t}, \tag{1}$$

Where:

- $Ret_{i,t}$ is the return of test asset i and,
- $PostBeta_{i,t}$ is the post-ranking beta of test asset i.

We construct our test assets as one of ten decile portfolios, sorted by the pre-ranking beta, i.e., for each day, we now sort stocks by their betas from the previous month :

```
data_sample <- data_sample[!is.na(beta_month_lag), ]
data_sample[, `:=`(decile, cut(beta_month_lag, breaks = quantile(beta_month_lag,
    probs = seq(0, 1, by = 0.1), na.rm = TRUE), labels = 1:10,
    include.lowest = TRUE)), by = .(yyyy, mm)]
```

and then form equal-...

```
data_sample_decile <- data_sample[, .(return_day = mean(return_day,
    na.rm = T), return_night = mean(return_night, na.rm = T),
    return_market = mean(return_market, na.rm = T)), by = .(date,
    yyyy, mm, decile)]
```

and value-weighted decile portfolios as test assets.

```
data_sample[, mktcap_monthly_sum := sum(mcap_mend_lag),by =.(date, decile)]
data_sample[, mktcap_weight := mcap_mend_lag/mktcap_monthly_sum]

data_sample_decile_vw <- data_sample[, .(
                              return_day = weighted.mean(return_day,
                                                        mktcap_weight,
                                                          na.rm = T),
                              return_night = weighted.mean(return_night,
                                                          mktcap_weight,
                                                          na.rm = T),
                           return_market = mean(return_market, na.rm=T)),

                                     by = .(date, yyyy, mm, decile)]
```

2.2.1 Figure 1 of Hendershott et al. (2020)

Following Hendershott et al. (2020), we estimate post-ranking betas (the x-values in Figure 5 and the explanatory variable in Table 1) differently for Figure and Table.

For Figure, we estimate post-ranking betas across the whole sample period :

```
data_sample_decile[, `:=`(beta, coef(lm(return_night ~ return_market))[2]),
    by = decile]

data_sample_decile_agg <- data_sample_decile[, .(beta = mean(beta,
    na.rm = T), return_day = mean(return_day, na.rm = T), return_night = mean(return_n
    na.rm = T)), by = .(beta)]
```

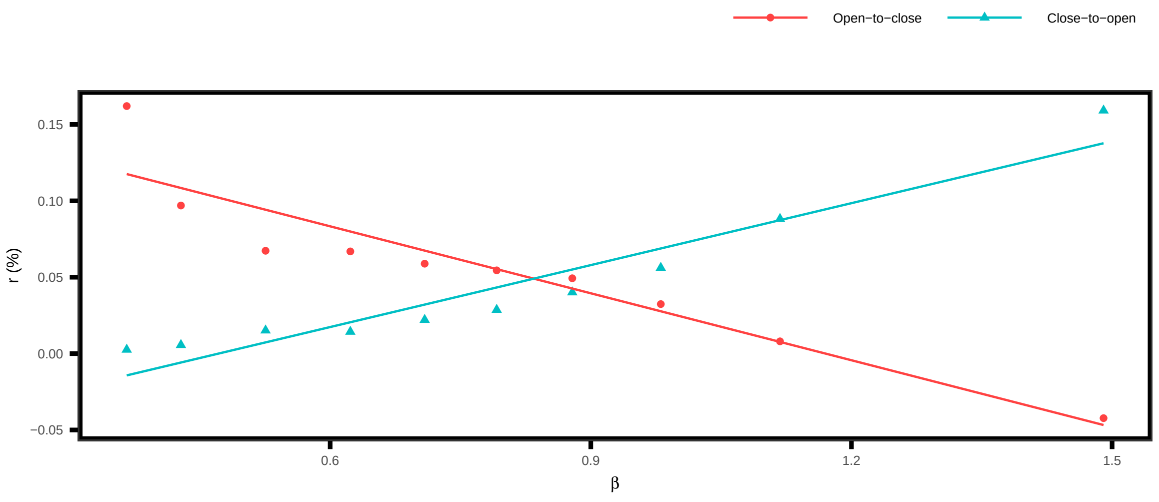
So that we can plot post-ranking betas (x-axis) against day- and night-returns, we convert the data from long format (with day and night returns in rows) to wide format (with day and night returns in columns).

```
dt_long <- melt(data_sample_decile_agg, id.vars = "beta", measure.vars = patterns("^re
    variable.name = "return", value.name = "value")

plot <- ggplot(dt_long, aes(x = beta, y = value)) + geom_point(aes(color = return,
    shape = return), size = 3) + geom_smooth(method = "lm", aes(color = return),
    se = FALSE) + scale_color_manual(values = c(return_day = "#ff4242",
    return_night = "#00bfc4"), labels = c(return_day = "Open-to-close",
    return_night = "Close-to-open")) + scale_shape_manual(values = c(return_day = 16,
    return_night = 17), labels = c(return_day = "Open-to-close",
    return_night = "Close-to-open")) + labs(x = expression(beta),
    y = "r (%)", color = "", shape = "") + guides(size = FALSE) +
    theme_report()
```

Figure 5 shows the resulting scatter plot and the fitted straight line. As in Hendershott et al. (2020), we see that the security market line is upward (downward) sloping when using night (day) returns.

Figure 5 – US day and night returns for beta-sorted portfolios (1992–2016)
This figure shows average (equal-weighted) daily returns in percent against market betas for ten beta-sorted portfolios of all US publicly listed common stocks. Portfolios are formed every month, with stocks sorted according to beta, estimated using daily night returns over a one-year rolling window. Portfolio returns are averaged, and post ranking betas are estimated over the whole sample period. Each day, returns are measured during the day, from open-to-close (red), and during the night, from close-to-open (cyan). For both ways of measuring returns, a line is fit using ordinary least squares estimates. Data are from CRSP.



2.2.2 Table 1 of Hendershott et al. (2020)

For the table, we estimate post-ranking betas each month using daily night returns over rolling 12-month windows. In other words, we estimate the post-ranking betas of the test assets (portfolios) in the same way we estimate the pre-ranking betas used to construct the test assets. Hence, we can reuse the same function that we used to estimate pre-ranking betas: compute_rolling_beta.

Empirically, Eq. 1 can be estimated in different ways. The literature commonly uses Fama–MacBeth regressions (i.e., cross-sectional regressions for each time period) and panel regressions (i.e., pooling all cross-sectional and time-series data into a single regression).

We define the following three functions to compute the results:

- For the Fama–MacBeth regressions, we use Newey–West standard errors to estimate t-statistics. To compute them, we define the following function:

```r
lags <- 10
fmb_nw <- function(x) {
    t(coeftest(lm(x ~ 1), vcov = NeweyWest(lm(x ~ 1), lag = lags,
        prewhite = FALSE))[, c("t value", "Pr(>|t|)")])
}
```

- The following function estimates the Fama-MacBeth regressions with Newey-West standard errors,

```r
sml_fmb <- function(data, return_column) {

    data_sample_decile_agg <- data[, compute_rolling_beta(.SD),
        by = decile]
    data_sample_decile_agg <- add_monthly_beta(data_sample_decile_agg,
        firm_id = "decile")

    data_sample_decile_agg <- data_sample_decile_agg[!is.na(beta_month_lag)]

    fmb_results <- data_sample_decile_agg[, {
        reg <- lm(get(return_column) ~ beta_month_lag)
        c(reg = as.list(coef(reg)), r_squared = summary(reg)$r.squared)
    }, by = .(date)]

    fmb_results[, `:=`(date, NULL)]

    fmb_coef <- colwise(mean)(fmb_results)
    fmb_t_stat <- colwise(fmb_nw)(fmb_results)

    return(list(coef = fmb_coef, t_stats = fmb_t_stat, r_squared = fmb_coef[["r_s
        100))
}
```

- The last function estimates the panel regression:

```r
sml_panel <- function(data) {

    data_sample_decile_agg <- data[, compute_rolling_beta(.SD),
        by = decile]
    data_sample_decile_agg <- add_monthly_beta(data_sample_decile_agg,
        firm_id = "decile")

    data_sample_decile_agg <- data_sample_decile_agg[!is.na(beta_month_lag)]

    data_sample_decile_agg_pool <- data_sample_decile_agg[, c("date",
        "decile", "beta_month_lag", "return_night", "return_day")]

    data_sample_decile_agg_pool <- melt(data_sample_decile_agg_pool,
        id.vars = c("date", "decile", "beta_month_lag"), variable.name = "return_
        value.name = "return")

    data_sample_decile_agg_pool[return_name == "return_day",
        `:=`(day_dummy, 1)]
    data_sample_decile_agg_pool[return_name == "return_night",
        `:=`(day_dummy, 0)]

    model <- feols(return ~ beta_month_lag + day_dummy + beta_month_lag *
        day_dummy | date, cluster = ~date, data = data_sample_decile_agg_pool)

    return(model)
}
```

Now we can call the functions defined above to estimate Eq. 1 in different ways. The results are shown in Table 1. Panels A and B use value-weighted and equal-weighted returns, respectively. In both panels, we present the results of the Fama–MacBeth and panel regressions.

```r
sml_fmb_day_ew <- sml_fmb(data_sample_decile, return_column = "return_day")
sml_fmb_day_vw <- sml_fmb(data_sample_decile_vw, return_column = "return_day")
sml_fmb_night_ew <- sml_fmb(data_sample_decile, return_column = "return_night")
sml_fmb_night_vw <- sml_fmb(data_sample_decile_vw, return_column = "return_night")

sml_panel_ew <- sml_panel(data_sample_decile)
sml_panel_ew_reg <- summary(sml_panel_ew)[["coeftable"]]
sml_panel_ew_r2 <- r2(sml_panel_ew)[["r2"]] * 100
sml_panel_vw <- sml_panel(data_sample_decile_vw)
sml_panel_vw_reg <- summary(sml_panel_vw)[["coeftable"]]
sml_panel_vw_r2 <- r2(sml_panel_vw)[["r2"]] * 100
```

Table 1 – US day and night returns (1992–2016)
This table reports results from the Fama-MacBeth and day fixed effect panel regressions of daily returns (in percent) on betas from ten beta-sorted test portfolios. Returns are measured during the day, from open-to-close, and during the night, from close-to-open. Portfolios are formed every month, with stocks sorted according to beta, estimated using daily night returns over a one-year rolling window. Panel A reports results from value-weighted portfolios based on market capitalization. Panel B reports results from equally weighted portfolios. *t*-statistics are reported in parentheses. Standard errors are based on Newey-West corrections, allowing for ten lags of serial correlation in Fama-MacBeth regressions. Standard errors are clustered at the day level for panel regressions. Statistical significance at the 1%, 5%, and 10% level is indicated by $***$, $**$, and *, respectively. Data are from CRSP.

Returns over	Fama-MacBeth regressions			Panel regressions			
	Intercept	Beta	Avg. R^2	Beta	Day	Day × Beta	R^2 [%]
Panel A: Value-weighted							
Night	-0.03***	0.06***	47.35	0.05***	0.11***	-0.12***	36.68
	(-7.43)	(8.93)		(5.34)	(9.53)	(-6.77)	
Day	0.08***	-0.07***	43.48				
	(9.27)	(-5.29)					
Panel B: Equal-weighted							
Night	-0.05***	0.14***	44.46	0.14***	0.25***	-0.32***	43.34
	(-9.70)	(13.65)		(11.27)	(20.82)	(-13.81)	
Day	0.16***	-0.14***	49.16				
	(19.36)	(-7.77)					

2.3 Test suite

```
test_that("test whether we have crsp data", {
    expect_true(file.exists(paste(data_dir, "data_wrds_crsp_complete.csv",
        sep = "/")))
})

## Test passed
```

```
date_min <- min(data_sample[, date])
date_max <- max(data_sample[, date])

test_that("test that we use data from date_start ", {
    expect_true(date_min + 5 > date_start)
})

## Test passed

test_that("test that we use data from date_start ", {
    expect_true(date_max - 5 < date_end)
})

## Test passed
```

```
test_data <- data.table(date = seq(as.Date("2010-01-01"), by = "day",
    length.out = 13), return_market = c(0.005539583, 0.006017547,
    0.002625172, 1.34571e-05, 0.000663921, 0.00752454, 0.002209173,
    0.009666029, 0.003485716, 0.008032367, 0.009132727, 0.005851332,
    0.00481022), return_night = c(0.003758466, 0.004434258, 0.001447537,
    0.001142291, 0.003993272, 0.001584714, 0.004837028, 0.003230552,
    0.005255025, 0.007339958, 0.001452744, 0.007059846, 0.000261878))
```

```r
test_that("compute_rolling_beta returns expected structure",
    {
        result <<- compute_rolling_beta(test_data)
        expect_s3_class(result, "data.table")
        expect_true("beta" %in% names(result))
    })

## Test passed

test_that("compute_rolling_beta returns NA because not enough observations",
    {
        expect_true(is.na(result[["beta"]][1]))
    })

## Test passed
```

```r
test_that("test whether we have beta for permno 14593 (AAPL) on 1995-12-27",
    {
        expect_equal(pretty_numbers(format_numeric, data_sample[permno ==
            14593 & date == "1995-12-27", beta]), "1.43")
    })

## Test passed
```

2.4 Functions and Setup Code

```r
library(zoo)
library(plyr)
library(dplyr)
library(sqldf)
options(sqldf.driver = "SQLite")
```

```r
require(RPostgres) | require(RPostgreSQL)

library(reshape2)
library(ggplot2)
library(testthat)
# for na.locf
library(xts)

# for R tidy in knitr
library(formatR)

library(MatchIt)
library(plm)

# for fwrite, fread
library(data.table)
library(anytime)
library(lubridate)

# For panel regression
library(fixest)
library(lmtest)
library(roll)

# For Newey-West std.error
library(sandwich)
# for storing passwords
library(keyring)

# Setup default number layouts
format_numeric <- "%.2f"
format_integer <- "%.0f"
```

```r
format_percent <- "%.2f"
format_significance <- "(%.2f)"

pretty_numbers <- function(format, number, p_value = -1, t_value = -1,
    ...) {
    result <- sprintf(format, number, ...)

    if (format == format_integer || format == format_numeric) {
        result <- prettyNum(result, big.mark = ",", scietific = F)
    }

    if (p_value > 0) {

        stars <- ""
        if (p_value < 0.1) {
            stars <- "*"
        }
        if (p_value < 0.05) {
            stars <- "**"
        }
        if (p_value < 0.01) {
            stars <- "***"
        }
        result <- sprintf("%1.2f%2$-3s", number, stars)
    }
    if (t_value > 0) {
        result <- prettyNum(result, big.mark = ",", scietific = F)
        result <- paste0("(", result, ")")
    }
    return(result)
}
```

```
sanitize <- function(str) {
    result <- str
    result <- gsub("&", "\\&", result, fixed = TRUE)
    result <- gsub("_", "\\_", result, fixed = TRUE)

    return(result)
}

# Setup default charting layout

theme_report <- function(base_size = 30, base_family = "", ...) {
    theme_bw(base_size = base_size, base_family = base_family) +
        theme(line = element_line(colour = "black", size = 4,
            linetype = "solid"), legend.position = "top", legend.justification = c("ri
            "top"), legend.key.height = unit(3, "line"), legend.key.width = unit(3,
            "cm"), legend.title = element_text(size = base_size *
            0.4), legend.text = element_text(size = base_size *
            0.4), legend.background = element_blank(), legend.key = element_blank(),
            strip.text.x = element_text(size = base_size * 1,
                colour = "black"), strip.background = element_rect(color = "black",
                size = 4), text = element_text(colour = "black",
                size = base_size * 1), axis.title = element_text(size = base_size *
                0.5), axis.text = element_text(size = base_size *
                0.4), axis.ticks = element_line(colour = "black",
                size = 2), panel.grid = element_blank(), panel.grid.minor = element_bl
            panel.background = element_rect(fill = NA, colour = "black",
                size = 4))
}
```

```
sessionInfo()

## R version 4.2.0 (2022-04-22)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 24.04.2 LTS
##
## Matrix products: default
## BLAS/LAPACK: /cvmfs/soft.ccr.buffalo.edu/versions/2023.01/easybuild/software/avx512
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C         LC_TIME=C
##  [4] LC_COLLATE=C         LC_MONETARY=C        LC_MESSAGES=C
##  [7] LC_PAPER=C           LC_NAME=C            LC_ADDRESS=C
## [10] LC_TELEPHONE=C       LC_MEASUREMENT=C     LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] keyring_1.3.2     sandwich_3.0-1    roll_1.1.7        lmtest_0.9-40
##  [5] fixest_0.12.1     lubridate_1.8.0   anytime_0.3.10    data.table_1.14.2
##  [9] plm_2.6-4         MatchIt_4.3.4     formatR_1.12      xts_0.14.1
## [13] testthat_3.1.3    ggplot2_3.5.0     reshape2_1.4.4    RPostgreSQL_0.7-5
## [17] DBI_1.2.3         sqldf_0.4-11      RSQLite_2.2.12    gsubfn_0.7
## [21] proto_1.0.0       dplyr_1.0.8       plyr_1.8.7        zoo_1.8-10
## [25] knitr_1.50
##
## loaded via a namespace (and not attached):
##  [1] pkgload_1.2.4       splines_4.2.0       bit64_4.0.5
##  [4] RcppParallel_5.1.5  brio_1.1.3          Formula_1.2-4
##  [7] Rdpack_2.3          assertthat_0.2.1    highr_0.11
## [10] blob_1.2.3          yaml_2.3.5          numDeriv_2016.8-1.1
## [13] pillar_1.7.0        backports_1.4.1     lattice_0.20-45
## [16] glue_1.6.2          chron_2.3-62        digest_0.6.29
## [19] rbibutils_2.2.8     colorspace_2.0-3    Matrix_1.6-5
## [22] pkgconfig_2.0.3     purrr_0.3.4         scales_1.3.0
## [25] collapse_2.0.18     tibble_3.1.6        mgcv_1.8-40
## [28] farver_2.1.0        generics_0.1.3      ellipsis_0.3.2
## [31] cachem_1.0.6        withr_2.5.0         maxLik_1.5-2
## [34] cli_3.6.2           magrittr_2.0.3      crayon_1.5.1
## [37] memoise_2.0.1       evaluate_0.15       fansi_1.0.3
## [40] nlme_3.1-168        MASS_7.3-57         dreamerr_1.4.0
## [43] tools_4.2.0         lifecycle_1.0.4     stringr_1.4.0
## [46] munsell_0.5.0       sodium_1.3.2        compiler_4.2.0
## [49] rlang_1.1.3         grid_4.2.0          rstudioapi_0.13
## [52] miscTools_0.6-26    rappdirs_0.3.3      labeling_0.4.2
## [55] tcltk_4.2.0         waldo_0.4.0         gtable_0.3.0
## [58] R6_2.5.1            bdsmatrix_1.3-4     fastmap_1.1.0
## [61] bit_4.0.4           utf8_1.2.2          rprojroot_2.0.3
## [64] filelock_1.0.3      desc_1.4.1          stringmagic_1.1.2
## [67] stringi_1.7.6       parallel_4.2.0      Rcpp_1.0.8.3
## [70] vctrs_0.6.5         tidyselect_1.1.2    xfun_0.52
```

```
lapply(dbListConnections(PostgreSQL()), dbDisconnect)

## [[1]]
## [1] TRUE
```

Chapter 3: Appendix

3.1 Programming code to reproduce Chapter 2

Listing 3.1 – Full Sweave Source

```
%
% This is a knitR file. To build it, it is easiest to use a local version of RStudio.
% By default, RStudio compiles PDF using Sweave. However, it is required to use `knitr`
% instead. To do this, update the setting in:
% "Tools" -> "Project Options" -> "Weave Rnw files using" -> "knitr"
%

\ifdefined\includemain

\else

\documentclass[authoryear,longnamesfirst,12pt]{article}\usepackage[]{graphicx}\usepackage[]{xcolor}
% maxwidth is the original width if it is less than linewidth
% otherwise use linewidth (to make sure the graphics do not exceed the margin)
\makeatletter
\def\maxwidth{ %
  \ifdim\Gin@nat@width>\linewidth
    \linewidth
  \else
    \Gin@nat@width
  \fi
}
\makeatother

\definecolor{fgcolor}{rgb}{0.345, 0.345, 0.345}
\newcommand{\hlnum}[1]{\textcolor[rgb]{0.686,0.059,0.569}{#1}}%
\newcommand{\hlsng}[1]{\textcolor[rgb]{0.192,0.494,0.8}{#1}}%
\newcommand{\hlcom}[1]{\textcolor[rgb]{0.678,0.584,0.686}{\textit{#1}}}%
\newcommand{\hlopt}[1]{\textcolor[rgb]{0,0,0}{#1}}%
\newcommand{\hldef}[1]{\textcolor[rgb]{0.345,0.345,0.345}{#1}}%
\newcommand{\hlkwa}[1]{\textcolor[rgb]{0.161,0.373,0.58}{\textbf{#1}}}%
\newcommand{\hlkwb}[1]{\textcolor[rgb]{0.69,0.353,0.396}{#1}}%
\newcommand{\hlkwc}[1]{\textcolor[rgb]{0.333,0.667,0.333}{#1}}%
\newcommand{\hlkwd}[1]{\textcolor[rgb]{0.737,0.353,0.396}{\textbf{#1}}}%
\let\hlipl\hlkwb

\usepackage{framed}
\makeatletter
\newenvironment{kframe}{%
 \def\at@end@of@kframe{}%
 \ifinner\ifhmode%
  \def\at@end@of@kframe{\end{minipage}}%
  \begin{minipage}{\columnwidth}%
 \fi\fi%
 \def\FrameCommand##1{\hskip\@totalleftmargin \hskip-\fboxsep
 \colorbox{shadecolor}{##1}\hskip-\fboxsep
     % There is no \\@totalrightmargin, so:
     \hskip-\linewidth \hskip-\@totalleftmargin \hskip\columnwidth}%
 \MakeFramed {\advance\hsize-\width
   \@totalleftmargin\z@ \linewidth\hsize
   \@setminipage}}%
 {\par\unskip\endMakeFramed%
 \at@end@of@kframe}
\makeatother

\definecolor{shadecolor}{rgb}{.97, .97, .97}
```

```
\definecolor{messagecolor}{rgb}{0, 0, 0}
\definecolor{warningcolor}{rgb}{1, 0, 1}
\definecolor{errorcolor}{rgb}{1, 0, 0}
\newenvironment{knitrout}{}{} % an empty environment to be redefined in TeX

\usepackage{alltt}

\begin{filecontents}{\jobname.bib}

@article{Hendershott2020,
title = {Asset pricing: A tale of night and day},
journal = {Journal of Financial Economics},
volume = {138},
pages = {635-662},
year = {2020},
author = {Terrence Hendershott and Dmitry Livdan and Dominik Rsch},
}

\end{filecontents}

\usepackage{cmap}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\newcommand{\q}[1]{`#1'}
\newcommand{\qq}[1]{``#1''}

\usepackage{natbib}
\usepackage{listings}  % Allows reading and displaying source files
\usepackage{xcolor}  % For color formatting (optional)

% To handle both R and Latex comments after line breaks in listing package:
% we add 1%%1# in front of each new line, which works both in R and Latex
% see https://tex.stackexchange.com/questions/376734/listings-make-linebroken-line-comment-a-comment
\makeatletter
\lst@AddToHook{AfterBeginComment}{%
    \CommentLinetrue%
}
\makeatother

%%%%%%%%%%%%%%%%%%%%%%


% to get URL formating
\usepackage[hidelinks]{hyperref}
\hypersetup{
    colorlinks=false,
}



% for table setup
\usepackage{tabularx}
\usepackage[margin=10pt,font=footnotesize,labelfont=bf,labelsep=endash]{caption}
\usepackage{booktabs}
\usepackage{subcaption}

\usepackage{mathtools}
\DeclarePairedDelimiter\abs{\lvert}{\rvert}

\usepackage{floatrow}
\floatsetup[table]{font={singlespacing,footnotesize}}
\floatsetup[table]{capposition=top}
\floatsetup[figure]{capposition=top}

% Set spacings
\usepackage[margin=3cm,includefoot]{geometry}
\usepackage{setspace}
% \doublespacing
```

```
\onehalfspacing
\IfFileExists{upquote.sty}{\usepackage{upquote}}{}
\begin{document}
% so that we can import into other file using standalone, these lines need to be after begin{document}

  \begin{center}
  \renewcommand{\thefootnote}{\fnsymbol{footnote}}
  \thispagestyle{empty}
  \begin{Large}
  \mbox{\textbf{Code for: Asset pricing: A tale of night and day}}
  \end{Large}
  \normalsize

  \vspace{7mm}
  \textbf{Toghrul Aghbabali}\\

  \vspace{5mm}
  \textbf{Wan Soo Choi}\\

  \vspace{5mm}
  \textbf{Taihun Im}\\

  \vspace{5mm}
  \textbf{Dominik R\"{o}sch}\\

  \vspace{5mm}
  \textbf{Prince Roy}\\

  \vspace{3mm}

  \vspace{10mm}
  \textbf{ABSTRACT}\\
  \vspace{-4mm}
  \end{center}
  \begingroup
  \leftskip2em
  \rightskip\leftskip
  This document provides code to replicate the main results of ``Asset pricing: A tale of night and day'' \cite{Hendershott2020} using
      data available through Wharton Research Data Services (WRDS).
  \\
  \par
  \endgroup

  \pagebreak

\fi

\newif\ifCommentLine%
\newcommand*{\CommentLineContinued}{\ifCommentLine 1\%\%1\#\space\fi}

\section{Setup}

Before we run any code, we set up an error handler.
This ensures that if the code breaks at any point, e.g., because the working directory is not yet set correctly, the output will stop
      and display the error message at the end of the document.
<<setup_error_handler,echo=TRUE>>=

# see https://stackoverflow.com/q/74097101
knitr::knit_hooks$set(error = function(x, options) {
  ERROR_GLOBAL <<- x
  knitr::knit_exit()
})

knit_chunk <- knitr::knit_hooks$get("chunk")

knitr::knit_hooks$set(chunk = function(x, options){

  out <- x
  if (exists('ERROR_GLOBAL', inherits = TRUE)) {
    out <- paste0(out,'\n**stopped with error**:',ERROR_GLOBAL,'\n\n')
    options$size       <- "large"
    options$background <- '#FFDDDD'
    options$tidy       <- 'T'
    options$tidy.opts  <- list(width.cutoff=50)
    options$results    <-'markup'
  }

  knit_chunk(out, options)

})

@

\subsection{Prerequisites}
In this section, we check whether the system can execute the analysis.

First, we ensure that the system has enough free memory to successfully run the analysis. Unfortunately, retrieving available memory is
      not straightforward and depends on factors such as the operating system and whether the code is running on a virtual machine.
      Because of this, we do not provide any actual code but instead ask users to manually verify that enough RAM is available. A lack
      of RAM often explains sudden termination without generating any output files.

<<setup_check_memory>>=

#to do - we need to have 50GB

@

Next, we load all required R libraries. While this must be done at the beginning of the script, we evaluate the code herewithout
      displaying itto avoid disrupting the reading flow. The full code is shown at the end of the file, Section~\ref{R:setup}.
<<setup_dummy>>=

# The actual code is executed here, but shown only at the end of the file.

@
<<setup, echo=FALSE,warning=F,message=F>>=

library(zoo)
library(plyr)
library(dplyr)
library(sqldf)
options(sqldf.driver = "SQLite")

require(RPostgres) | require(RPostgreSQL)

library(reshape2)
library(ggplot2)
library(testthat)
# for  na.locf
library(xts)

# for R tidy in knitr
library(formatR)

library(MatchIt)
library(plm)

# for fwrite, fread
library( data.table)
library(anytime)
library(lubridate)

# For panel regression
library(fixest)
library(lmtest)
library(roll)
```

```r
# For Newey-West std.error
library(sandwich)
# for storing passwords
library(keyring)

# Setup default number layouts
format_numeric    <- "%.2f";
format_integer    <- "%.0f";
format_percent    <- "%.2f";
format_significance <- "(%.2f)";

pretty_numbers <- function(format, number, p_value = -1, t_value = -1, ...)
{
    result <- sprintf(format, number, ...)

    if (format == format_integer || format == format_numeric)
    {
        result <- prettyNum(result, big.mark=",",  scietific=F)
    }

    if (p_value > 0) {

        stars <- ""
        if (p_value < 0.10) {  stars <- "*" }
        if (p_value < 0.05) {  stars <- "**" }
        if (p_value < 0.01) {  stars <- "***" }
        result <- sprintf("%1.2f%2$-3s", number, stars)
    }
    if (t_value >0) {
      result <- prettyNum(result, big.mark=",",  scietific=F)
      result <- paste0("(", result, ")")
    }
    return(result)
}

sanitize <- function(str) {
  result <- str
  result <- gsub("&", "\\&", result, fixed = TRUE)
  result <- gsub("_", "\\_", result, fixed = TRUE)

  return(result)
}

# Setup default charting layout

theme_report <- function(base_size = 30, base_family = "", ...) {
  theme_bw(base_size = base_size, base_family = base_family) +
    theme(
      line = element_line(colour = "black", size = 4, linetype = "solid"),
      legend.position = "top",
      legend.justification = c("right", "top"),
      legend.key.height = unit(3, "line"),
      legend.key.width = unit(3, "cm"),
      legend.title = element_text(size = base_size * 0.4),
      legend.text = element_text(size = base_size * 0.4),
      legend.background = element_blank(),
      legend.key = element_blank(),
      strip.text.x = element_text(size = base_size * 1.0, colour = "black"),
      strip.background = element_rect(color = "black", size = 4),
      text = element_text(colour = "black", size = base_size * 1.0),
      axis.title = element_text(size = base_size * 0.5),
      axis.text = element_text(size = base_size * 0.4),
      axis.ticks = element_line(colour = "black", size = 2),
      panel.grid = element_blank(),
      panel.grid.minor = element_blank(),
      panel.background = element_rect(fill = NA, colour = "black", size = 4)
    )
}
```

```
@

Similarly, we ensure that we compile the document using tinytex and that all required Latex packages are available.
If any package is missing we try to install it:
<<setup_latex_packages, cache=TRUE>>=

if(! tinytex::is_tinytex()) {
  stop("We use tinytex to install missing Latex packages, if you
  are using a different Latex version this might not be compatible.
  It might be easiest to run tinytex::install_tinytex()")
}

# if(! grepl("TinyTeX", Sys.which("pdflatex"))) {
# stop("It looks like you are using a different Latex version
# which might not be compatible.
# It might be easiest to switch to TinyTex, for example,
# by adding the PATH to pdflatex in TinyTex directory
# to PATH environmental variable. See the code in the Rnw file below.")
#
#   cat('Sys.setenv(PATH = paste0(tinytex::tinytex_root(),
#"\\bin\\windows;", Sys.getenv("PATH")))\n',
#     file = file.path(Sys.getenv("HOME"), ".Rprofile"),
#     append = TRUE)
# }

latex_packages <- c('listings', 'caption', 'mathtools', 'floatrow',
                    'setspace', 'cmap', 'filecontents')

for (pkg in latex_packages) {
  tinytex::check_installed(pkg) || tinytex::tlmgr_install(pkg)
}

@

\subsection{Setup variable names}
This section contains various code blocks to set up the analysis such as variables determining how the compiled PDF will look (e.g.,
      whether it will contain the underlying R code);
environment variables that need to be adjusted (like where to store the data); and variables related to the analysis (such as start and
      end dates).
\begin{itemize}

  \item  First, the code contains variables that determine how the PDF will look, e.g., with (``echo=T'') or without the underlying R
      code (``echo=F''.)
<<setup_variables_knitr,echo=TRUE>>=

knitr::opts_knit$set(progress = TRUE, verbose = TRUE)
knitr::opts_knit$set(self.contained=T)

# cache.lazy = F (helps error when caching large data sets)
knitr::opts_chunk$set(echo=T,warning=F,message=F, error=T, tidy=T,
tidy.opts=list(width.cutoff=60), results='markup',cache.lazy = FALSE,
cache.rebuild=F)

@

    \item  Second, the code sets important environment variables (which need to be adjusted by the user):
<<setup_variables_user,echo=T>>=

setwd("~/repositories/3r/")

data_dir <- "/projects/academic/drosch/3r/"
# data_dir <- "/projects/academic/drosch/3r/recent"
# data_dir <- "/projects/academic/drosch/3r/adrs"

 if(! file.exists(data_dir)) {
   stop(sprintf("Could not find data_dir '%s' to store data,
                first create this directory.", data_dir))
 }

@

 \item  Third, we define variables specific to the analysis (which can be adjusted by the user):
<<setup_variables_analysis>>=

date_start <- "1992-01-01"
date_end <- "2016-12-31"

# date_start <- "2017-01-01"
# date_end <- "2024-12-31"


@

    \item  Fourth, we set up the username and password to download all required data from WRDS.
    We suggest two alternatives and opt for the second.
    As explained on the WRDS website\footnote{
  https://wrds-www.wharton.upenn.edu/pages/support/programming-wrds/programming-r/r-from-your-computer/
}
we can store the username and password in the following file (depending on the Operating System)
\begin{itemize}
    \item \%APPDATA\%/postgresql/pgpass.conf
    \item  ~/.pgpass
\end{itemize}
and in the following way:
\begin{itemize}
    \item wrds-pgdata.wharton.upenn.edu:9737:wrds:wrds\_username:wrds\_password
\end{itemize}
<<setup_wrds_password_1>>=
# password_file_name <- "~/.pgpass"
# if(!file.exists(password_file_name)) {
#   stop('Setup_wrds_password needs to run manually in Console to setup Your WRDS password:')
#   cat(paste0("wrds-pgdata.wharton.upenn.edu:9737:wrds:", wrds_username, ":", wrds_password), file=password_file_name, append=TRUE,
      1%%1# sep = "\n")
# }
@


Instead, we use the R-package ``keyring'' to safely store the username and password. To do either, you need to execute the code
      directly in the console to set either the file (see code chunk above) or the keyring (see code below, within tryCatch).
<<setup_wrds_password_2>>=
tryCatch({
  keyring_create("credentials_wrds", password = 'letmein')
  key_set("wrds_username", keyring ="credentials_wrds"
      ,prompt = "Your WRDS username:")
  key_set("wrds_password", keyring ="credentials_wrds"
      ,prompt = "Your WRDS password:")
},  error = function(e) {}
)

keyring_unlock("credentials_wrds", password = 'letmein')

@


\item  Last, we establish a connection to WRDS. If required data cannot be found, we can automatically download the data from WRDS.
<<setup_db>>=



postregs <- ifelse(exists("Postgres"), Postgres, PostgreSQL)

tryCatch({
wrds <- dbConnect(postregs(),
        host='wrds-pgdata.wharton.upenn.edu',
        port=9737,
        dbname='wrds',
        user=key_get("wrds_username",
                     keyring ="credentials_wrds"),
        password =  key_get("wrds_password",
                     keyring ="credentials_wrds"))
},
  error = function(e) {
    stop("It is likely that wrds_username and
    wrds_password are not stored in keyring.
    For that, run code in setup_wrds_password directly
    in R console. An error occurred: ", e$message)
  }
)

@


\end{itemize}


\subsection{Download and Prepare Required Data}
If the data are found in the previously defined data directory, read them; otherwise, download them. First, we define a function to
      download the stock data from WRDS.
<<function_get_stock_returns>>=

get_stock_returns <- function(wrds, date_start, date_end) {
    sql_query <- sprintf(
    "SELECT a.cusip, a.permno, a.date, a.prc, a.vol,
          a.ret, a.openprc, a.askhi, a.bidlo, b.ticker, a.shrout, b.shrcd
      FROM crsp.dsf AS a, crsp.dsenames AS b
      WHERE a.permno = b.permno
      AND a.date BETWEEN b.namedt AND b.nameendt
      AND a.date >= '%s' AND a.date <= '%s'
      AND b.shrcd IN ('10', '11')",
      date_start, date_end
      )
  crsp_data <- dbGetQuery(wrds,sql_query) %>% as.data.table()
  return(crsp_data)
}

@

Then we call the function above in case the relevant data file cannot be found.
<<get_stock_returns>>=
data_file_name <- paste(data_dir, "data_wrds_crsp_complete.csv", sep = "/")

if(! file.exists(data_file_name)){
  data_sample <- get_stock_returns(wrds, date_start, date_end)
  fwrite(data_sample, file=data_file_name)
}

data_sample <- fread(data_file_name)
@

We then define variables as in the reference paper. In particular, we estimate night returns from 24-hour, corporate action-adjusted
      returns. This ensures that our night returns are also calculated from corporate action adjusted prices:
<<data_variables>>=
data_sample[, prc := abs(prc)]
data_sample[, mcap:=prc*shrout]
data_sample[, return_day   := prc/openprc -1]
data_sample[, return_night := (1+ret)/(1+return_day) - 1]
data_sample[,date:=as.Date(date)]
data_sample[,yyyy:=year(date)]
data_sample[,mm:=month(date)]
@

\subsection{Filter and scale the data according to the reference paper}
As in the reference paper, we drop stock-days without an opening price and stock-days with a daily return above 10.
```

```
<<data_filter>>=
data_sample <- data_sample[!is.na(openprc)]

stocks_drop_large_day_return <- count(data_sample[return_day >= 10])

data_sample <- data_sample[return_day < 10]
@
\cite{Hendershott2020} drop 16 stock-days with a day return over 1,000\%; in our sample, we drop \Sexpr{stocks_drop_large_day_return}
      1%%1# stock-days.

To ensure that we restrict the data to dates between `date\_start' and `date\_end', we filter accordingly.
Note that if the data files contain fewer observations, the actual date range will be limited by the contents of the files. We test for
      this at the end of the script. In general, and to avoid such cases, it is best to set `data\_dir' to a new, empty directory when
      starting a new analysis.

<<data_filter_2>>=
data_sample <- data_sample[date >= date_start & date <= date_end]
@

For readability, we measure returns in percent:
<<data_scale>>=
data_sample[, return_day   := return_day * 100]
data_sample[, return_night := return_night * 100]
@

\subsection{Compute required variables}

After filtering, we compute other variables that we need for the later analysis.
Note that, in general, these variables depend on whether we construct them before or after we filter our sample.
For each month and each stock, we compute the market capitalization based on the last trading day the month. We then use the market
      capitalization from the previous month to compute value-weighted portfolio returns:
<<calculate_mktcap>>=

setorder(data_sample, permno, date)
data_sample[, date_mktcap := .I == .I[.N], by = .(permno, yyyy, mm)]

data_sample[date_mktcap == T, mcap_mend := last(mcap), by = .(permno, yyyy,mm)]
data_sample[, mcap_mend_lag := shift(mcap_mend), by=.(permno)]

data_sample[, mcap_mend_lag := nafill(mcap_mend_lag, type = "locf"), by = .(permno)]

@

and we compute the market return. Following, \cite{Hendershott2020} ``the market index is constructed as the value-weighted portfolio''.
We use this market return to estimate how sensitive stock night-returns are to market night-returns, called the Beta of the stock.
<<calculate_mkt_return>>=

# data_sample[, return_market := mean(return_night, na.rm = TRUE), by = .(date)]
data_sample[, return_market := weighted.mean(return_night, mcap/sum(mcap), na.rm = T), by = .(date)]

@

We often need to compute market betas using a rolling window.
For reusability, we define a function to do that:
<<function_compute_rolling_beta>>=
  compute_rolling_beta <- function(data, y = "return_night",
                                   x ="return_market") {
    setorder(data, date)
    data$beta <- roll_lm(
      x = data[[x]],
      y = data[[y]],
      width = 252, # Approx. 12 months of trading days
      min_obs = 30 # Minimum of ~30 days of data to compute beta
    )$coef[, 2]

    return(data)
  }
```

```
@

and then call the function to estimate betas.
Because calculating betas is time consuming, we cache the results.
Instead of writing the results to a file (as we do with the stock return data), we cache the code block.
Beta estimations depend on previous blocks, such as how we filter the data.
To ensure we recalculate betas whenever necessary, we use ``dependson'' in the code block.
If any of the code blocks listed in ``dependson'' change, we will recalculate betas.
<<compute_rolling_beta, cache=T, dependson=c("function_compute_rolling_beta", "calculate_mkt_return","data_filter", "data_merge",
      "setup_variables_user", "function_get_stock_returns", "get_stock_returns", "setup_variables_analysis")>>=

  data_beta <- data_sample[, compute_rolling_beta(.SD), by = permno][, c("permno","date", "beta")]



@

To minimize the data cached when computing betas, we dropped all other data.
We now merge back the rest of the data:
<<data_merge_returns_betas>>=

  data_sample <- merge(data_sample, data_beta, by =c("permno","date"), all.x= TRUE)
@


As in \cite{Hendershott2020}, we estimate monthly betas.
We use the equally weighted average of the daily betas within each stock-month.
The following function adds a new column to input ``data'' containing the average beta from the previous month:
<<compute_monthly_beta>>=

add_monthly_beta <- function(data, firm_id) {
    data_beta_aggregate <- data[, .(beta_month = mean(beta,na.rm = TRUE)),by = c(firm_id,"yyyy","mm")]

    data_beta_aggregate <- data_beta_aggregate[!is.na(beta_month)]

    setorderv(data_beta_aggregate, cols = c(firm_id, "yyyy", "mm"))
    data_beta_aggregate[, beta_month_lag := shift(beta_month), by = c(firm_id)]
    data_beta_aggregate[, beta_month:=NULL]

    data <- merge(data, data_beta_aggregate, by = c(firm_id, "yyyy","mm"), all.x = TRUE)

    return(data)
  }

data_sample <- add_monthly_beta(data_sample,firm_id ="permno")

@


\section{Beta portfolios}

One of the main predictions of the CAPM is that stocks with higher (undiversifiable) risk (beta) have higher expected returns.
To test this, we look at the slope of the Security-Market Line (SML), i.e., the straight line that crosses the risk-free rate and the
      ``market''.
Empirically, we estimate the following regression:
\begin{equation}
\label{eq:sml}
  Ret_{i, t} = \alpha + \beta PostBeta_{i, t} + \epsilon_{i, t},
\end{equation}
Where:
\begin{itemize}
\item $Ret_{i, t}$ is the return of test asset $i$ and,
\item $PostBeta_{i, t}$ is the post-ranking beta of test asset $i$.
\end{itemize}

We construct our test assets as one of ten decile portfolios, sorted by the pre-ranking beta, i.e., for each day, we now sort stocks by
      their betas from the previous month :
<<create_beta_deciles>>=
data_sample <- data_sample[!is.na(beta_month_lag),]
data_sample[, decile := cut(beta_month_lag,
               breaks = quantile(beta_month_lag,
                      probs = seq(0, 1, by = 0.1),
                      na.rm = TRUE),
               labels = 1:10,
               include.lowest = TRUE), by = .(yyyy, mm)]
@

and then form equal-...
<<create_beta_deciles_ew>>=
data_sample_decile <- data_sample[, .(
                             return_day = mean(return_day, na.rm=T),
                             return_night = mean(return_night, na.rm=T),
                             return_market = mean(return_market, na.rm=T)
                                      ),
                                  by = .(date, yyyy, mm, decile)]
@

and value-weighted decile portfolios as test assets.
<<create_beta_deciles_vw>>=
data_sample[, mktcap_monthly_sum := sum(mcap_mend_lag),by =.(date, decile)]
data_sample[, mktcap_weight := mcap_mend_lag/mktcap_monthly_sum]

data_sample_decile_vw <- data_sample[, .(
                              return_day = weighted.mean(return_day,
                                                        mktcap_weight,
                                                           na.rm = T),
                              return_night = weighted.mean(return_night,
                                                            mktcap_weight,
                                                            na.rm = T),
                           return_market = mean(return_market, na.rm=T)),

                                  by = .(date, yyyy, mm, decile)]
@


\subsection{Figure 1 of \cite{Hendershott2020} }

Following \cite{Hendershott2020}, we estimate post-ranking betas (the x-values in Figure~\ref{fig:us_beta}} and the explanatory
      variable in Table~\ref{tab:us_portfolio_returns}) differently for Figure and Table.

For Figure, we estimate post-ranking betas across the whole sample period :
<<calculate_postranking_beta_whole_sample>>=
data_sample_decile[, beta := coef(lm(return_night ~ return_market))[2],
                   by = decile]

data_sample_decile_agg <- data_sample_decile[, .(
                    beta = mean(beta, na.rm=T),
                    return_day = mean(return_day,
                                      na.rm=T),
                    return_night = mean(return_night,
                                      na.rm=T)
                       ), by = .(beta) ]

@

So that we can plot post-ranking betas (x-axis) against day- and night-returns, we convert the data from long format (with day and
      night returns in rows) to wide format (with day and night returns in columns).
<<plot_beta_sorted_portfolio_returns_by_day_and_night>>=

dt_long <- melt(
  data_sample_decile_agg,
  id.vars = "beta",
  measure.vars = patterns("^return_"),
  variable.name = "return",
  value.name = "value"
)
```

```
plot <- ggplot(dt_long, aes(x = beta, y = value)) +
  geom_point(aes(color = return, shape = return), size = 3) +
  geom_smooth(method = "lm", aes(color = return), se = FALSE) +
  scale_color_manual(
    values = c("return_day" = "#ff4242", "return_night" = "#00bfc4"),
    labels = c("return_day" = "Open-to-close", "return_night" = "Close-to-open")
  ) +
  scale_shape_manual(
    values = c("return_day" = 16, "return_night" = 17),
    labels = c("return_day" = "Open-to-close", "return_night" = "Close-to-open")
  ) +
  labs(x = expression(beta), y = "r (%)", color = "", shape = "") +
  guides(size = FALSE) +
  theme_report()

@



Figure~\ref{fig:us_beta} shows the resulting scatter plot and the fitted straight line.
As in \cite{Hendershott2020}, we see that the security market line is upward (downward) sloping when using night (day) returns.
\begin{figure}[H]
\centering
<<plot_capm, fig=true, fig.width=15,fig.height=7,echo=FALSE>>=

print(plot)

@
\caption{
   \textbf{
   US day and night returns for beta-sorted portfolios (\Sexpr{year(date_start)}--\Sexpr{year(date_end)})} \\
This figure shows average (equal-weighted) daily
returns in percent against market betas for ten beta-sorted portfolios of
all US publicly listed common stocks. Portfolios are formed every month,
with stocks sorted according to beta, estimated using daily night returns
over a one-year rolling window. Portfolio returns are averaged, and
post ranking betas are estimated over the whole sample period. Each day, returns
are measured during the day, from open-to-close (red), and during the
night, from close-to-open (cyan). For both ways of measuring returns, a line
is fit using ordinary least squares estimates. Data are from CRSP.
}
\label{fig:us_beta}
\end{figure}



\subsection{Table 1 of \cite{Hendershott2020} }
For the table, we estimate post-ranking betas each month using daily night returns over rolling 12-month windows.
In other words, we estimate the post-ranking betas of the test assets (portfolios) in the same way we estimate the pre-ranking betas
      used to construct the test assets.
Hence, we can reuse the same function that we used to estimate pre-ranking betas: compute\_rolling\_beta.

Empirically, Eq.~\ref{eq:sml} can be estimated in different ways.
The literature commonly uses FamaMacBeth regressions (i.e., cross-sectional regressions for each time period) and panel regressions
      (i.e., pooling all cross-sectional and time-series data into a single regression).

We define the following three functions to compute the results:
\begin{itemize}
\item For the FamaMacBeth regressions, we use NeweyWest standard errors to estimate t-statistics. To compute them, we define the
      following function:
<<function_neweywest>>=

lags <- 10
fmb_nw <- function(x) {
    t(coeftest(lm(x ~ 1), vcov = NeweyWest(lm(x ~ 1), lag = lags, prewhite = FALSE))[,c("t value", "Pr(>|t|)")])
}

@
    \item The following function estimates the Fama-MacBeth regressions with Newey-West standard errors,
<<Fama-Macbeth>>=
sml_fmb <- function(data, return_column) {

  data_sample_decile_agg <- data[, compute_rolling_beta(.SD), by = decile]
  data_sample_decile_agg <- add_monthly_beta(data_sample_decile_agg, firm_id ="decile")

  data_sample_decile_agg <- data_sample_decile_agg[!is.na(beta_month_lag)]

  fmb_results <- data_sample_decile_agg[, {
    reg <- lm(get(return_column) ~ beta_month_lag)
    c(reg = as.list(coef(reg)), r_squared = summary(reg)$r.squared)
  }, by = .(date)]

  fmb_results[, date := NULL]

  fmb_coef   <- colwise(mean)(fmb_results)
  fmb_t_stat <- colwise(fmb_nw)(fmb_results)

  return(list(coef=fmb_coef, t_stats = fmb_t_stat, r_squared = fmb_coef[["r_squared"]] * 100))
}
@
  \item The last function estimates the panel regression:
<<Panel>>=
sml_panel <- function(data) {

  data_sample_decile_agg <- data[, compute_rolling_beta(.SD), by = decile]
  data_sample_decile_agg <- add_monthly_beta(data_sample_decile_agg, firm_id ="decile")

  data_sample_decile_agg <- data_sample_decile_agg[!is.na(beta_month_lag)]

  data_sample_decile_agg_pool <- data_sample_decile_agg[, c("date","decile", "beta_month_lag", "return_night", "return_day")]

  data_sample_decile_agg_pool <- melt(data_sample_decile_agg_pool,
                                      id.vars = c("date","decile", "beta_month_lag"),
                                      variable.name = "return_name", value.name = "return")

  data_sample_decile_agg_pool[return_name == "return_day",   day_dummy := 1]
  data_sample_decile_agg_pool[return_name == "return_night", day_dummy := 0]

  model <- feols(return~beta_month_lag + day_dummy + beta_month_lag * day_dummy | date,
                         cluster=~date, data=data_sample_decile_agg_pool)

  return(model)
}
@

\end{itemize}
Now we can call the functions defined above to estimate Eq.~\ref{eq:sml} in different ways.
The results are shown in Table 1.
Panels A and B use value-weighted and equal-weighted returns, respectively.
In both panels, we present the results of the FamaMacBeth and panel regressions.

<<table1>>=

sml_fmb_day_ew   <- sml_fmb(data_sample_decile,    return_column="return_day")
sml_fmb_day_vw   <- sml_fmb(data_sample_decile_vw, return_column="return_day")
sml_fmb_night_ew <- sml_fmb(data_sample_decile,    return_column="return_night")
sml_fmb_night_vw <- sml_fmb(data_sample_decile_vw, return_column="return_night")

sml_panel_ew <- sml_panel(data_sample_decile)
sml_panel_ew_reg <- summary(sml_panel_ew)[["coeftable"]]
sml_panel_ew_r2  <- r2(sml_panel_ew)[["r2"]] * 100
sml_panel_vw <- sml_panel(data_sample_decile_vw)
sml_panel_vw_reg <- summary(sml_panel_vw)[["coeftable"]]
sml_panel_vw_r2  <- r2(sml_panel_vw)[["r2"]] * 100

@
```

```
\begin{table}[!htb]
\caption{US day and night returns (1992--2016) \newline
{\protect\footnotesize This table reports results from the Fama-MacBeth and day
fixed effect panel regressions of daily returns (in percent) on betas from
ten beta-sorted test portfolios. Returns are measured during the day, from
open-to-close, and during the night, from close-to-open. Portfolios are
formed every month, with stocks sorted according to beta, estimated using
daily night returns over a one-year rolling window. Panel A reports
results from value-weighted portfolios based on market capitalization. Panel B reports
results from equally weighted portfolios. $t$-statistics are reported in
parentheses. Standard errors are based on Newey-West corrections, allowing
for ten lags of serial correlation in Fama-MacBeth regressions. Standard
errors are clustered at the day level for panel regressions. Statistical
significance at the 1\%, 5\%, and 10\% level is indicated by $***$, $**$, and *,
respectively. Data are from CRSP.}}
\label{tab:us_portfolio_returns}\renewcommand{\arraystretch}{1.5}%
\addtolength{\tabcolsep}{+2pt} \centering %
\begin{tabularx}{1\textwidth}{p{2.0cm} X@{} X@{} X@{} X@{} p{2.5cm} X@{}}
        \toprule
         Returns over&
         \multicolumn{3}{l}{Fama-MacBeth regressions}&
         \multicolumn{4}{l}{Panel regressions}
        \\
        &
         Intercept  &
         Beta &
        Avg. $R^2$ &
        Beta &
        Day &
        Day $\times$ Beta &
        $R^2$ [\%]
            \\
           \hline
    \multicolumn{5}{l}{Panel A: Value-weighted}
            \\
    \hspace{0.3cm} Night &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_night_vw[["coef"]][,"reg.(Intercept)"], p_value =
      sml_fmb_night_vw[["t_stats"]][["reg.(Intercept)"]][1,"Pr(>|t|)"])}  &
      \Sexpr{pretty_numbers(format_numeric, sml_fmb_night_vw[["coef"]][,"reg.beta_month_lag"], p_value =
      sml_fmb_night_vw[["t_stats"]][["reg.beta_month_lag"]][1,"Pr(>|t|)"])} &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_night_vw[["r_squared"]])} &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_vw_reg["beta_month_lag", "Estimate"], p_value =
      sml_panel_vw_reg["beta_month_lag", "Pr(>|t|)"])}  &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_vw_reg["day_dummy", "Estimate"], p_value = sml_panel_vw_reg["day_dummy",
      "Pr(>|t|)"])}  &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_vw_reg["beta_month_lag:day_dummy", "Estimate"], p_value =
      sml_panel_vw_reg["beta_month_lag:day_dummy", "Pr(>|t|)"])} &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_vw_r2)}
       \\[-3pt]
       &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_night_vw[["t_stats"]][["reg.(Intercept)"]][1,"t value"], t_value=1)} &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_night_vw[["t_stats"]][["reg.beta_month_lag"]][1,"t value"], t_value=1)} &
       &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_vw_reg["beta_month_lag", "t value"], t_value=1)}  &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_vw_reg["day_dummy", "t value"], t_value=1)}  &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_vw_reg["beta_month_lag:day_dummy", "t value"], t_value=1)} &

       \\
    \hspace{0.3cm} Day &
      \Sexpr{pretty_numbers(format_numeric, sml_fmb_day_vw[["coef"]][,"reg.(Intercept)"], p_value =
      sml_fmb_day_vw[["t_stats"]][["reg.(Intercept)"]][1,"Pr(>|t|)"])}  &
      \Sexpr{pretty_numbers(format_numeric, sml_fmb_day_vw[["coef"]][,"reg.beta_month_lag"], p_value =
      sml_fmb_day_vw[["t_stats"]][["reg.beta_month_lag"]][1,"Pr(>|t|)"])} &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_day_vw[["r_squared"]])} &
       &
       &
       &
       \\[-3pt]
       &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_day_vw[["t_stats"]][["reg.(Intercept)"]][1,"t value"], t_value=1)} &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_day_vw[["t_stats"]][["reg.beta_month_lag"]][1,"t value"], t_value=1)} &
       &
       &
       &
       &

       \\
              \\
     \multicolumn{5}{l}{Panel B: Equal-weighted}
              \\
     \hspace{0.3cm} Night &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_night_ew[["coef"]][,"reg.(Intercept)"], p_value =
      sml_fmb_night_ew[["t_stats"]][["reg.(Intercept)"]][1,"Pr(>|t|)"])}  &
      \Sexpr{pretty_numbers(format_numeric, sml_fmb_night_ew[["coef"]][,"reg.beta_month_lag"], p_value =
      sml_fmb_night_ew[["t_stats"]][["reg.beta_month_lag"]][1,"Pr(>|t|)"])} &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_night_ew[["r_squared"]])} &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_ew_reg["beta_month_lag", "Estimate"], p_value =
      sml_panel_ew_reg["beta_month_lag", "Pr(>|t|)"])}  &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_ew_reg["day_dummy", "Estimate"], p_value = sml_panel_ew_reg["day_dummy",
      "Pr(>|t|)"])}  &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_ew_reg["beta_month_lag:day_dummy", "Estimate"], p_value =
      sml_panel_ew_reg["beta_month_lag:day_dummy", "Pr(>|t|)"])} &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_ew_r2)}
       \\[-3pt]
       &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_night_ew[["t_stats"]][["reg.(Intercept)"]][1,"t value"], t_value=1)} &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_night_ew[["t_stats"]][["reg.beta_month_lag"]][1,"t value"], t_value=1)} &
       &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_ew_reg["beta_month_lag", "t value"], t_value=1)}  &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_ew_reg["day_dummy", "t value"], t_value=1)}  &
       \Sexpr{pretty_numbers(format_numeric, sml_panel_ew_reg["beta_month_lag:day_dummy", "t value"], t_value=1)} &

       \\
    \hspace{0.3cm} Day &
      \Sexpr{pretty_numbers(format_numeric, sml_fmb_day_ew[["coef"]][,"reg.(Intercept)"], p_value =
      sml_fmb_day_ew[["t_stats"]][["reg.(Intercept)"]][1,"Pr(>|t|)"])}  &
      \Sexpr{pretty_numbers(format_numeric, sml_fmb_day_ew[["coef"]][,"reg.beta_month_lag"], p_value =
      sml_fmb_day_ew[["t_stats"]][["reg.beta_month_lag"]][1,"Pr(>|t|)"])} &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_day_ew[["r_squared"]])} &
       &
       &
       &

       \\[-3pt]
       &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_day_ew[["t_stats"]][["reg.(Intercept)"]][1,"t value"], t_value=1)} &
       \Sexpr{pretty_numbers(format_numeric, sml_fmb_day_ew[["t_stats"]][["reg.beta_month_lag"]][1,"t value"], t_value=1)} &
       &
       &
       &
       &

       \\
    \bottomrule
    \end{tabularx}
\end{table}


\clearpage

\section{Test suite}

<<test_data_availability>>=
 test_that("test whether we have crsp data", {
     expect_true(file.exists(paste(data_dir, "data_wrds_crsp_complete.csv", sep="/")))
})

@

<<test_data_sample_dates>>=

  date_min <- min(data_sample[ ,date])
  date_max <- max(data_sample[ ,date])

test_that("test that we use data from date_start ", {
     expect_true(date_min + 5 > date_start )
})

test_that("test that we use data from date_start ", {
     expect_true(date_max - 5 < date_end )
})
@

<<test_function_compute_rolling_beta>>=

  test_data <- data.table(
    date = seq(as.Date("2010-01-01"), by = "day", length.out = 13),
    return_market = c(0.005539583, 0.006017547, 0.002625172, 1.34571E-05, 0.000663921,
                            0.00752454, 0.002209173, 0.009666029, 0.003485716, 0.008032367,
                            0.009132727, 0.005851332, 0.00481022),
    return_night = c(0.003758466, 0.004434258, 0.001447537, 0.001142291, 0.003993272,
                     0.001584714, 0.004837028, 0.003230552, 0.005255025, 0.007339958,
                     0.001452744, 0.007059846, 0.000261878)
  )

  test_that("compute_rolling_beta returns expected structure", {
    result <<- compute_rolling_beta(test_data)
    expect_s3_class(result, "data.table")
    expect_true("beta" %in% names(result))
  })

  test_that("compute_rolling_beta returns NA because not enough observations", {
    expect_true(is.na( result[["beta"]][1]) )
  })

@


<<test_data_sample_beta>>=

test_that("test whether we have beta for permno 14593 (AAPL) on 1995-12-27", {
     expect_equal(pretty_numbers(format_numeric,data_sample[permno == 14593 & date == "1995-12-27",beta]), "1.43")
})

@


\section{Functions and Setup Code}

\label{R:setup}
<<setup, eval=FALSE>>=
@


<<sessionInfo>>=
sessionInfo()
@

<<cleanup>>=
lapply(dbListConnections(PostgreSQL()), dbDisconnect)
 @
```

```
\ifdefined\includemain

% drop the following if included into other file

\else

  \addcontentsline{toc}{chapter}{\numberline{}References}
  \renewcommand{\bibname}{References}
  \bibliographystyle{chicago}
  \bibliography{\jobname}

  \clearpage
  \pagenumbering{gobble}

  \section{Appendix: Source Code}

  \lstset{
    basicstyle=\ttfamily\tiny,        % Reduce font size
    breaklines=true,                  % Enable line wrapping
    breakatwhitespace=true,           % Prefer breaking at spaces
    columns=fullflexible,             % Avoid extra spaces between characters
    frame=single,                     % Add a box around the code
    backgroundcolor=\color{gray!10},  % Light gray background
    xleftmargin=10pt,                 % Indent code for readability
    language=R,
    keepspaces=true,
    showstringspaces=false,
    morecomment=[l][\color{gray}]{\%},
    morecomment=[l][\color{gray}]{\#},
    postbreak={\CommentLineContinued},
  % postbreak={\CommentLineContinued},
    upquote=true
  }



  \lstinputlisting[caption=Full Sweave Source]{\jobname.Rnw}  % Auto-loads current file

   \end{document}
\fi
```

References

Barroso, P., A. Detzel, and P. Maio (2025). The volatility puzzle of the beta anomaly. *Journal of Financial Economics 165*, 103994.

Brodeur, A., D. Mikola, and N. Cook (2024). Mass reproducibility and replicability: A new hope. Technical report, IZA - Institute of Labor Economics.

Brogaard, J., M. C. Ringgenberg, and D. Roesch (2025). Does floor trading matter? *Journal of Finance 80*(1), 375–414.

Christopher Gandrud (2020). Reproducible Research with R and RStudio. `https://github.com/christophergandrud/Rep-Res-Book`. Accessed: 2025-04-07.

Committee on Reproducibility and Replicability in Science (2019). *Reproducibility and Replicability in Science.* National Academies of Sciences.

Diane Coyle (2021). *Cogs and Monsters: What Economics Is, and What It Should Be.* Princeton University Press.

Dreber, A. and M. Johannesson (2025). A framework for evaluating reproducibility and replicability in economics. *Economic Inquiry 63*(2), 338–356.

Duvendack, M., R. Palmer-Jones, and W. R. Reed (2017). What is meant by "replication" and why does it encounter resistance in economics? *American Economic Review 107*(5), 46–51.

Harvey, C. R. (2017). Presidential address: The scientific outlook in financial economics. *Journal of Finance 72*(4), 1399–1440.

Hasso, T., M. Brosnan, S. Ali, and D. Chai (2024). Perceived problems, causes, and solutions of finance research reproducibility and replicability: A pre-registered report. *Pacific-Basin Finance Journal*, 102564.

Hellum, O., T. I. Jensen, B. Kelly, and L. H. Pedersen (2025). The power of the common task framework. *working paper, Copenhagen Business School*.

Hendershott, T., D. Livdan, and D. Rösch (2020). Asset pricing: A tale of night and day. *Journal of Financial Economics 138*, 635–662.

Hill, R. and C. Stein (2025). Race to the bottom: Competition and quality in science*. *The Quarterly Journal of Economics 140*(2), 1111–1185.

Hou, K., C. Xue, and L. Zhang (2020). Replicating anomalies. *Review of Financial Studies 33*(5), 2019–2133.

Ioannidis, J. (2005). Why most published research findings are false. *PLOS Medicine 2*(124).

Jensen, T. I., B. Kelly, and L. H. Pedersen (2023). Is there a replication crisis in finance? *Journal of Finance 78*(5), 2465–2518.

Knuth, D. E. (1984). Literate programming. *Computer Journal 27*(2), 97–111.

Menkveld, A. J. and 342 co-authors (2024). Nonstandard errors. *Journal of Finance 79*, 2339–2390.

Miguel, E. (2021). Evidence on research transparency in economics. *Journal of Economic Perspectives 35*(3), 193–214.

Mitton, T. (2021). Methodological variation in empirical corporate finance. *Review of Financial Studies 35*(2), 527–575.

Open Science Collaboration (2015). Psychology. estimating the reproducibility of psychological science. *Science 349*(6251).

Samuelson, P. A. (1966). *The Collected Scientific Papers of Paul A. Samuelson*, Volume 3. Cambridge, MA: MIT Press.

Sharma, P. N., M. Sarstedt, C. M. Ringle, J.-H. Cheah, A. Herfurth, and J. F. Hair (2024). A framework for enhancing the replicability of behavioral mis research using prediction oriented techniques. *International Journal of Information Management 78*, 102805.

Stefan Nagel (2021). Answers to FAQ about the recent retraction of an article in the JF. `https://afajof.org/2021/07/answers-to-faq-about-the-recent-retraction-of-an-article-in-the-jf/`. Accessed: 2025-04-07.

Tange, O. (2011). Gnu parallel - the command-line power tool. *The USENIX Magazine*, 42–47.

Vilhuber, L., I. Schmutte, A. Michuda, and M. Connolly (2023, jul 27). Reinforcing Reproducibility and Replicability: An Introduction. *Harvard Data Science Review 5*(3). https://hdsr.mitpress.mit.edu/pub/l8dmf3cm.

Whited, T. (2023). Costs and Benefits of Reproducibility in Finance and Economics. *Harvard Data Science Review 5*(3).

Wikipedia (2025). GitHub. `https://en.wikipedia.org/wiki/GitHub/`. Accessed: 2025-04-07.

Ziliak, S. T. and D. N. McCloskey (2008). *The Cult of Statistical Significance: How the Standard Error Costs Us Jobs, Justice, and Lives.*