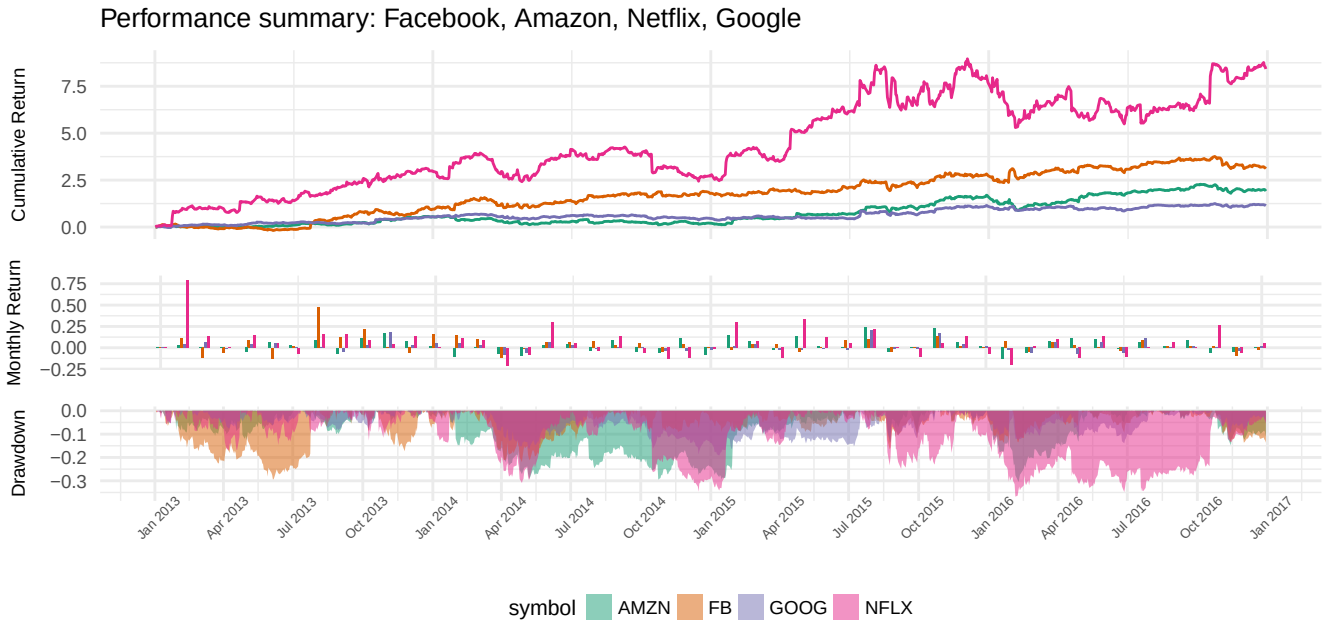
A time series platform for the tidyverse

Davis Vaughan and Matt Dancho

This version was compiled on January 29, 2018

The tidyverse ecosystem has evolved rapidly over the past few years, making it easier for the average R user to manipulate his or her data, but there is still not a robust solution for performing time series analysis directly inside that ecosystem. Specifically, there is room for a *time-based tibble*, similar in spirit to `xts`, but built with the intention of leveraging the benefits of the tidyverse and designed to work seamlessly with the packages that already exist within it. The `tibbletime` package aims to solve this problem by building directly off of the `tibble` package and incorporating a time-index into what is "known" about the data structure. By doing so, `tibbletime` functions as a platform for further development of packages that extend the tidyverse into the realm of time series analysis.



The current state of the world

Time series analysis for R in general is a well-developed field. There are CRAN Task Views for both Time Series and Finance, and an enormous amount of packages that build off of the zoo, Achim Zeileis (2018), and xts, Jeffrey A. Ryan (2017), infrastructure. The benefits of this are clear, with speed provided by `xts` and consistency with base R provided by `zoo`.

With that said, because the `xts` system is built off of matrices, there is little support for data frames. As the `tidyverse` grows, and the number of packages supporting manipulation of data frames continues to increase, we feel that there is room for a package that manipulates them in the way that `xts` and `zoo` manipulate matrices, specifically, with support for a time-index and specialized time-series manipulation functions. Like `zoo` attempts to be invisible to the base R user, this package would attempt to be invisible to the `tidyverse` user, continuing to allow access to packages such as `dplyr` and `tidyr` for general manipulation.

The purpose of this package is not to create a divide between the users of `xts` and those of the `tidyverse`, but to simply give the `tidyverse` users tools for working with time series in a format that they are already familiar with. The hope is that this can lower the barrier to entry to time series analysis for those users, while simultaneously taking full advantage of the tools of the `tidyverse`, of which grouped analysis and heterogeneous data structures are a few.

A time series platform: tibbletime

A new package, `tibbletime`, is our proposed solution. `tibbletime` is an extension of the tidyverse that allows for the creation of *time-aware tibbles* through the setting of a time-index column. It is built directly off of tibbles, Kirill Muller (2018), rather than matrices, directly inheriting all of the benefits (or consequences, according to some) that come with it.

The creation of such objects is easy, simply call `tbl_time()` on an existing `data.frame` or `tibble`, specifying a column to use as the `index`.

```
library(tibbletime)
data(FB)
FB$volume <- NULL

FB_time <- tbl_time(FB, index = date)
FB_time
```

```
#  # A time tibble: 1,008 x 7
#  # Index: date
#    symbol date        open  high   low close adjusted
#    <chr>  <date>     <dbl> <dbl> <dbl> <dbl>    <dbl>
#  1 FB     2013-01-02  27.4  28.2  27.4  28.0     28.0
#  2 FB     2013-01-03  27.9  28.5  27.6  27.8     27.8
#  3 FB     2013-01-04  28.0  28.9  27.8  28.8     28.8
#  4 FB     2013-01-07  28.7  29.8  28.6  29.4     29.4
#  5 FB     2013-01-08  29.5  29.6  28.9  29.1     29.1
#  # ... with 1,003 more rows
```

As `tbl_time` objects attempt to maintain seamless integration with the rest of the `tidyverse`, functions such as `mutate()` and `select()` work intelligently with `tbl_time` objects, retaining the class and attributes where expected.

```
library(dplyr)

# Index has not been modified / dropped, so this
# should still be a tbl_time object
mutate(FB_time, adjusted_return = adjusted / lag(adjusted) - 1)
```

```
#  # A time tibble: 1,008 x 8
#  # Index: date
#    symbol date        open  high   low close adjusted adjusted_return
#    <chr>  <date>     <dbl> <dbl> <dbl> <dbl>    <dbl>           <dbl>
#  1 FB     2013-01-02  27.4  28.2  27.4  28.0     28.0          NA
#  2 FB     2013-01-03  27.9  28.5  27.6  27.8     27.8        - 0.00821
#  3 FB     2013-01-04  28.0  28.9  27.8  28.8     28.8          0.0356
#  4 FB     2013-01-07  28.7  29.8  28.6  29.4     29.4          0.0229
#  5 FB     2013-01-08  29.5  29.6  28.9  29.1     29.1        - 0.0122
#  # ... with 1,003 more rows
```

```
# Index has been removed, this should no longer
# be a tbl_time object
select(FB_time, adjusted)
```

```
#  # A tibble: 1,008 x 1
#    adjusted
#       <dbl>
#  1     28.0
#  2     27.8
#  3     28.8
#  4     29.4
#  5     29.1
#  # ... with 1,003 more rows
```

There are a number of functions inside `tibbletime` that take advantage of the fact that it "knows" about the index column. One such function is `filter_time()`. Pass `filter_time()` a "time formula" represented as `from ~ to` and it returns rows inside that range, inclusive.

```
filter_time(FB_time, "2013-01-01" ~ "2013-01-04")
```

```
#  # A time tibble: 3 x 7
#  # Index: date
#    symbol date        open  high   low close adjusted
#    <chr>  <date>     <dbl> <dbl> <dbl> <dbl>    <dbl>
#  1 FB     2013-01-02  27.4  28.2  27.4  28.0     28.0
#  2 FB     2013-01-03  27.9  28.5  27.6  27.8     27.8
#  3 FB     2013-01-04  28.0  28.9  27.8  28.8     28.8
```

Time formulas are intelligently parsed and expanded into appropriate date ranges, allowing for quick shorthand.

```
# Start of 2013 to the end of 2014
filter_time(FB_time, "2013" ~ "2014")
```

```
#  # A time tibble: 504 x 7
#  # Index: date
#    symbol date        open  high   low close adjusted
#    <chr>  <date>     <dbl> <dbl> <dbl> <dbl>    <dbl>
#  1 FB     2013-01-02  27.4  28.2  27.4  28.0     28.0
#  2 FB     2013-01-03  27.9  28.5  27.6  27.8     27.8
#  3 FB     2013-01-04  28.0  28.9  27.8  28.8     28.8
#  4 FB     2013-01-07  28.7  29.8  28.6  29.4     29.4
#  5 FB     2013-01-08  29.5  29.6  28.9  29.1     29.1
#  # ... with 499 more rows
```

Even less typing is required for common ranges, such as "every day in the second month of 2013", by using a one-sided time formula.

```
filter_time(FB_time, ~"2013-02")
```

```
#  # A time tibble: 19 x 7
#  # Index: date
#    symbol date        open  high   low close adjusted
#    <chr>  <date>     <dbl> <dbl> <dbl> <dbl>    <dbl>
#  1 FB     2013-02-01  31.0  31.0  29.6  29.7     29.7
#  2 FB     2013-02-04  29.1  29.2  28.0  28.1     28.1
#  3 FB     2013-02-05  28.3  29.0  28.0  28.6     28.6
#  4 FB     2013-02-06  28.7  29.3  28.7  29.0     29.0
#  5 FB     2013-02-07  29.1  29.2  28.3  28.6     28.6
#  # ... with 14 more rows
```

One of the main purposes of `tibbletime` is to enhance tools that already exist in the `tidyverse`. For example, a common task is to calculate monthly summaries from daily data. Traditionally, one might create separate year and month columns to group on, then summarise. `tibbletime` provides an easier and more general solution in `collapse_by()`. With `collapse_by()`, a `period` for collapsing is specified, and the index column is modified so that every row that falls in that interval shares a common date. This is easiest to show through an example.

```
# Before the collapse
select(FB_time, date)
```

```
#  # A time tibble: 1,008 x 1
#  # Index: date
#    date
#    <date>
#  1 2013-01-02
#  2 2013-01-03
#  3 2013-01-04
#  4 2013-01-07
#  5 2013-01-08
#  # ... with 1,003 more rows
```

```
# Collapse by year
FB_yearly <- collapse_by(FB_time, period = "year")

# After the collapse, every date in 2013 now shares a common date (2013-12-31)
# every date in 2014 shares (2014-12-31) and so on.
select(FB_yearly, date)
```

```
#  # A time tibble: 1,008 x 1
#  # Index: date
#    date
#    <date>
#  1 2013-12-31
#  2 2013-12-31
#  3 2013-12-31
#  4 2013-12-31
#  5 2013-12-31
#  # ... with 1,003 more rows
```

```
unique(FB_yearly$date)
```

```
#  [1] "2013-12-31" "2014-12-31" "2015-12-31" "2016-12-30"
```

This kind of manipulation is useful for bucketing your data into different time intervals that can be grouped on for further analysis.

```
FB_time %>%
  collapse_by("year") %>%
  group_by(date) %>%
  summarise(adjusted_mean = mean(adjusted))
```

```
#  # A time tibble: 4 x 2
#  # Index: date
#    date       adjusted_mean
#    <date>             <dbl>
#  1 2013-12-31          35.5
#  2 2014-12-31          68.8
#  3 2015-12-31          88.8
#  4 2016-12-30         117
```

The decision to split the collapsing of dates (the `collapse_by()`) from the actual manipulation of data (the `summarise()`) was intentional and serves two purposes. First, it lets users to continue to use tools they are familiar with. And second, as opposed to having a specific function that performed both the collapsing and summarising (such as `time_summarise()`), this approach eases the burden of the developer by having the user directly use existing tools rather than reimplementing every `tidyverse` function to have its own `time_*()` form. For instance, this approach allows for the direct use of scoped variants of `dplyr` functions such as `summarise_if()` for quick manipulation of multiple columns by period.

```
# Mean of every numeric column, calculated every 2 quarters
FB_time %>%
  collapse_by("2 quarter") %>%
  group_by(date) %>%
  summarise_if(is.numeric, mean)
```

```
#  # A time tibble: 8 x 6
#  # Index: date
#    date        open  high   low close adjusted
#    <date>     <dbl> <dbl> <dbl> <dbl>    <dbl>
#  1 2013-06-28  27.0  27.4  26.6  27.0     27.0
#  2 2013-12-31  43.6  44.4  43.0  43.7     43.7
#  3 2014-06-30  62.5  63.4  61.4  62.4     62.4
#  4 2014-12-31  74.8  75.7  74.0  74.9     74.9
#  5 2015-06-30  80.0  80.8  79.3  80.0     80.0
#  6 2015-12-31  97.2  98.3  96.0  97.2     97.2
#  7 2016-06-30 111   112   109   110      110
#  8 2016-12-30 124   124   122   123      123
```

And finally, because this is directly in the `tidyverse`, it immediately takes advantage of existing tools for grouped analysis, allowing for scalability that users of the `tidyverse` are already familiar with.

```
data(FANG)

FANG_time <- FANG %>%
  group_by(symbol) %>%
  as_tbl_time(date)

FANG_time %>%
  collapse_by("year") %>%
  group_by(symbol, date) %>%
  summarise_all(median) %>%
  print(n = 12)
```

```
#  # A time tibble: 16 x 8
#  # Index:  date
#  # Groups: symbol [?]
#     symbol date        open  high   low close   volume adjusted
#     <chr>  <date>     <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl>
#   1 AMZN   2013-12-31 282   285   280   282    2556250    282
```

```
#   2 AMZN   2014-12-31 328   331   323   326    3498200    326
#   3 AMZN   2015-12-31 440   445   436   440    3245950    440
#   4 AMZN   2016-12-30 727   731   721   728    3594800    728
#   5 FB     2013-12-31  29.7  30.4  29.4  29.8 49838850     29.8
#   6 FB     2014-12-31  69.3  70.0  68.4  69.1 42264250     69.1
#   7 FB     2015-12-31  86.8  87.8  85.6  86.8 23898750     86.8
#   8 FB     2016-12-30 118   119   117   118   20993100    118
#   9 GOOG   2013-12-31 876   880   871   876    3838400    438
#  10 GOOG   2014-12-31 569   574   563   568    1877200    565
#  11 GOOG   2015-12-31 564   571   559   563    1817850    563
#  12 GOOG   2016-12-30 743   747   737   743    1587300    743
#  # ... with 4 more rows
```

Extensions

The end goal for `tibbletime` is to be an infrastructure package for other users of the `tidyverse` to use and extend as they see fit. It attempts to provide time-index support for a number of classes (`Date`, `POSIXct`, `yearmon`, `yearqtr`, and `hms`), and exposes functions that make working with that index column a bit easier.

As an example, an in-progress package with a temporary name of `tidyfinance` is the first of potentially many packages that extend the work of `tibbletime`. The below `calculate_return()` function leverages the infrastructure of `tibbletime` to make calculating returns on multiple columns and at multiple periods a simple task.

```
# devtools::install_github("DavisVaughan/tidyfinance")
library(tidyfinance)

# Daily return of the adjusted stock price of multiple companies
FANG_time %>%
  calculate_return(adjusted, period = "daily") %>%
  select(symbol, date, adjusted, adjusted_return)
```

```
#  # A time tibble: 4,032 x 4
#  # Index:  date
#  # Groups: symbol [4]
#    symbol date       adjusted adjusted_return
#    <chr>  <date>        <dbl>           <dbl>
#  1 FB     2013-01-02     28.0         0
#  2 FB     2013-01-03     27.8        -0.00821
#  3 FB     2013-01-04     28.8         0.0356
#  4 FB     2013-01-07     29.4         0.0229
#  5 FB     2013-01-08     29.1        -0.0122
#  # ... with 4,027 more rows
```

```
# Yearly return of multiple columns
FANG_time %>%
  calculate_return(open:low, adjusted, period = "year") %>%
  select(symbol, date, contains("return"))
```

```
#  # A time tibble: 20 x 6
#  # Index:  date
#  # Groups: symbol [4]
#    symbol date       open_return high_return low_return adjusted_return
#    <chr>  <date>           <dbl>       <dbl>      <dbl>           <dbl>
#  1 FB     2013-01-02     0           0         0               0
#  2 FB     2013-12-31     0.972       0.947     0.966           0.952
#  3 FB     2014-12-31     0.470       0.455     0.444           0.428
#  4 FB     2015-12-31     0.333       0.330     0.344           0.341
```

```
#  5 FB     2016-12-30      0.1000       0.100     0.0970          0.0993
#  # ... with 15 more rows
```

Combined with a few other other functions, namely `drawdown()` and `cumulative_return()`, one can quickly generate a performance summary report similar to `PerformanceAnalytics`.

Below, we again calculate daily returns, but also append the drawdown and cumulative returns of each of the four stocks onto the data frame as well.

```
library(ggplot2)

FANG_return <- FANG_time %>%
  select(symbol, date, adjusted) %>%
  calculate_return(adjusted, period = "daily") %>%
  mutate(drawdown = drawdown(adjusted_return),
         cum_ret  = cumulative_return(adjusted_return))

FANG_return
```

```
#  # A time tibble: 4,032 x 6
#  # Index:  date
#  # Groups: symbol [4]
#    symbol date       adjusted adjusted_return drawdown  cum_ret
#    <chr>  <date>        <dbl>           <dbl>    <dbl>    <dbl>
#  1 FB     2013-01-02     28.0         0        0        0
#  2 FB     2013-01-03     27.8        -0.00821 -0.00821 -0.00821
#  3 FB     2013-01-04     28.8         0.0356   0        0.0271
#  4 FB     2013-01-07     29.4         0.0229   0        0.0507
#  5 FB     2013-01-08     29.1        -0.0122  -0.0122   0.0379
#  # ... with 4,027 more rows
```

Using `collapse_by()` from `tibbletime` along with `total_return()` we can convert daily returns into monthly returns easily. Again, notice the separation of the specification of the period that the data is collapsed at, and the actual computation done at each period. This allows `total_return()` to be as simple as possible, while retaining the ability to calculate it at any generic period.

```
FANG_return_monthly <- FANG_return %>%
  collapse_by("month") %>%
  group_by(symbol, date) %>%
  summarise(monthly_return = total_return(adjusted_return))

FANG_return_monthly
```

```
#  # A time tibble: 192 x 3
#  # Index:  date
#  # Groups: symbol [?]
#    symbol date       monthly_return
#    <chr>  <date>              <dbl>
#  1 AMZN   2013-01-31        0.0318
#  2 AMZN   2013-02-28       -0.00463
#  3 AMZN   2013-03-28        0.00840
#  4 AMZN   2013-04-30       -0.0476
#  5 AMZN   2013-05-31        0.0606
#  # ... with 187 more rows
```

The next few chunks of code use the results above to create three images that are then combined using a new (currently only on GitHub) package, `patchwork`, by Thomas Lin Pedersen. The resulting image is a summary that was inspired by the work of `PerformanceAnalytics`.

Chart 1 - Cumulative returns.

```
plot_cum_ret <- FANG_return %>%
  ggplot(aes(x = date, y = cum_ret, color = symbol)) +
  geom_line() +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.text.x  = element_blank(),
        axis.ticks.x = element_blank()) +
  labs(
   y     = "Cumulative Return",
   title = "Performance summary: Facebook, Amazon, Netflix, Google") +
  theme(legend.position="none") +
  scale_color_brewer(palette = "Dark2")
```

Chart 2 - Monthly returns.

```
plot_month_ret <- FANG_return %>%
  calculate_return(adjusted, period = "monthly") %>%
  ggplot(aes(x = date, y = adjusted_return, fill = symbol)) +
  geom_col(width = 15, position = position_dodge()) +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.text.x  = element_blank(),
        axis.ticks.x = element_blank()) +
  labs(y = "Monthly Return") +
  theme(legend.position="none") +
  scale_fill_brewer(palette = "Dark2")
```

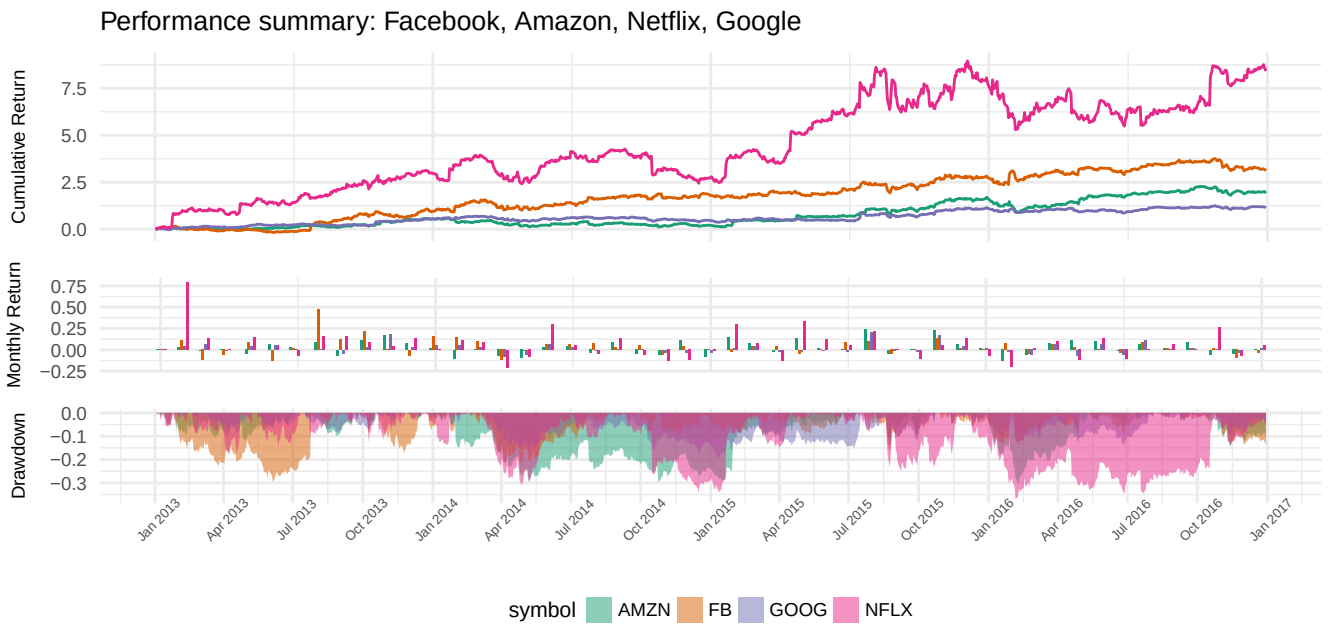
Chart 3 - Drawdown.

```
plot_drawdown <- FANG_return %>%
  ggplot(aes(x = date, y = drawdown, fill = symbol)) +
  geom_area(position = position_identity(), alpha = .5) +
  theme_minimal() +
  scale_x_date(
    date_breaks = "3 months",
    date_labels = "%b %Y") +
  labs(x = "", y = "Drawdown") +
  scale_fill_brewer(palette = "Dark2") +
  theme(axis.text.x = element_text(angle = 45, size = 5)) +
  theme(legend.position = "bottom", legend.margin = margin(t = -10))
```

Performance summary with patchwork.

```
# For performance summary plots
# This will install the dev version of ggplot2
# devtools::install_github("thomasp85/patchwork")
library(patchwork)

plot_cum_ret +
  plot_month_ret +
  plot_drawdown +
  plot_layout(ncol = 1, heights = c(2, 1, 1))
```



Conclusion

Time series analysis is a valuable part of the R ecosystem, but up until now the development of time-based manipulation of data frames has been limited. We hope that `tibbletime` can be the start of that development, and can provide tools to R users that commonly work with this kind of data, whether that be in finance, business, meterology, or any other field that relies on time-based data. By attempting to design the package as a platform, we hope that other R users will extend the package in their own creative ways, creating new tools for all of us to benefit from.

Acknowledgments. This package builds upon the data frame infrastructure of tibble, along with inherting time-series data structure ideas from xts.

References

Achim Zeileis Gabor Grothendieck JAR (2018). *An S3 class with methods for totally ordered indexed observations. It is particularly aimed at irregular time series of numeric vectors/matrices and factors. zoo's key design goals are independence of a particular index/date/time class and consistency with ts and base R by providing methods to extend standard generics.* R package version 1.8-1, URL https://CRAN.R-project.org/package=zoo.

Jeffrey A Ryan Joshua M Ulrich RB (2017). *Provide for uniform handling of R's different time-based data classes by extending zoo, maximizing native format information preservation and allowing for user level customization and extension, while simplifying cross-class interoperability.* R package version 0.10-1, URL https://CRAN.R-project.org/package=xts.

Kirill Muller Hadley Wickham RF (2018). *Provides a tbl df class, the tibble, that provides stricter checking and better formatting than the traditional data frame.* R package version 1.4.2, URL https://CRAN.R-project.org/package=tibble.