New Tools for Performing Financial Analysis Within the "Tidy" Ecosystem

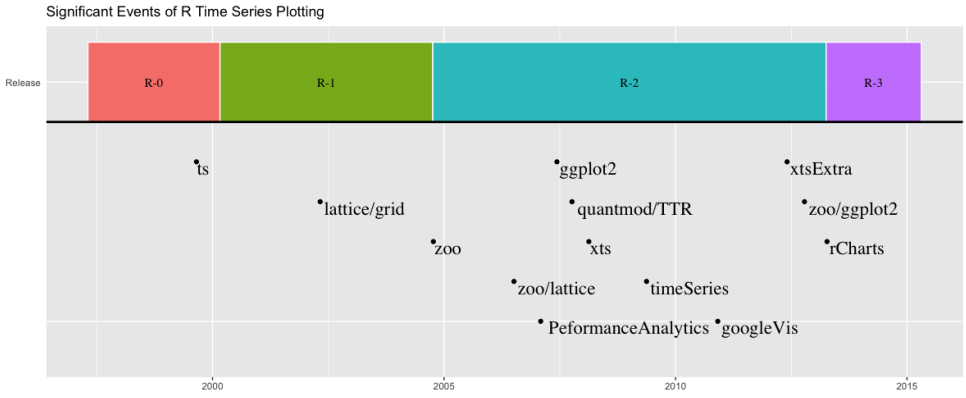
by Matt Dancho, Davis Vaughan

Abstract Financial analysis and data science in the R programming language have followed separate paths resulting in two important systems: "xts" and "tidyverse". The advantage of the "xts" system is in the management of time series financial data. The advantage of the "tidyverse" system is in the ability to implement a set of packages that have a uniform structure and data representation to incorporate within the data science workflow. Because of the separation in development, the two systems are difficult to use together limiting the full potential of financial analysis within R. The `tidyquant` package solves this problem by integrating several of the best financial analysis packages with the "tidy" ecosystem. The integration unlocks the data science workflow, which enables modeling and scaling analyses in a "tidy" way.

Three usage cases are discussed to illustrate the benefits of `tidyquant`. The first example uses the scaling capabilities to provide an answer for how the financial market values risk versus reward. The second example builds on the first by visualizing the returns over time and illustrating how the `purrr` package can be used to apply functions to data frames. The third example evaluates methods to scale the performance analysis of multiple portfolios using combinations of weighted asset blends. These examples only scratch the surface of what is possible with `tidyquant`.

Status of Financial Analysis Tools in R

The R programming language has seen immense growth in both popularity and tools over the past several years, primarily driven by the open-source nature of the R language and innovation in the field of data science. The sub-segment of financial analysis in R is no different. TimelyPortfolio maintains a timeline of the major advances in R time series graphics, which highlights the inception of several of the most influential R financial and time series packages. Several packages are worth describing in more detail as these create much of the current foundation of R in Finance.



Source: TimelyPortfolio

quantmod / TTR

The *Quantitative Financial Modelling & Trading Framework for R* (`quantmod`) package and the *Technical Trading Rules* (`TTR`) package include mechanisms to retrieve, compute, and visualize financial data using the most popular technical trading rules.

xts / zoo

The *Extensible Time Series* (`xts`) package along with the `zoo` package include mechanisms for the handling of time series data. Most importantly, the `xts` package implemented a cross-package method for handling the various time series data structures, in the process solving a major shortcoming by

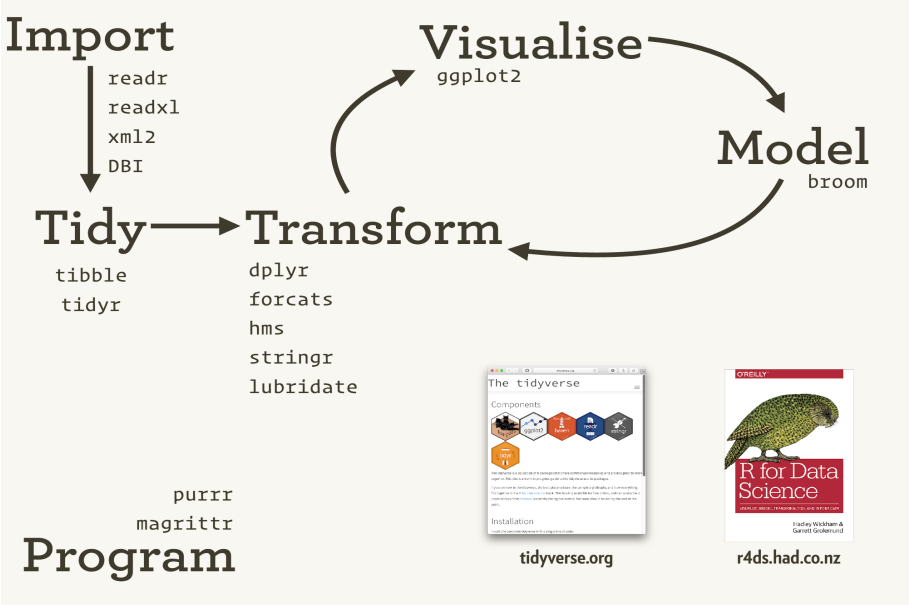
managing *all major R time series objects* [1] under one class, `xts`.

PerformanceAnalytics

The `PerformanceAnalytics` package includes a large collection of econometric functions for financial performance analysis, many of which are described in *Practical Portfolio Performance Measurement and Attribution* by Carl Bacon (Bacon, 2004). These functions enable analysis of individual asset and portfolio (the weighted aggregation of multiple assets) returns using popular statistical methods for measuring performance.

New Tools: tidyverse

In parallel with the progression in the R in Finance community, developers at *RStudio* have been creating useful tools for data science in R, namely the "tidyverse". The "tidyverse", or "tidy" ecosystem, is a collection of packages that fundamentally and philosophically work together utilizing a common "tidy" data representation, which is defined in *Tidy Data* (Wickham, 2014). To summarize, the "tidy" data representation reserves *each variable as a column, each observation as a row, and each value as a cell*.[2] The "tidyverse" packages all share a common API design and fit together seamlessly within the data science workflow. Additionally, the "tidyverse" packages are documented in the online text, *R for Data Science* (Wickham and Grolemund, 2017), which is the de facto manual for data scientists beginning with the R programming language. The "tidyverse" includes several packages worth describing in more detail.



Source: Wickham (2017)

dplyr / tidyr

The `dplyr` and `tidyr` packages provide tools to clean and manipulate data. Historically, the `plyr` package, the precursor to `dplyr`, was developed as a mechanism to implement the *split-apply-combine* framework popularized by Hadley Wickham in *The Split-Apply-Combine Strategy for Data Analysis* (Wickham, 2011). The `dplyr` package advanced this concept by implementing this framework internally along with providing many additional tools that work naturally within the framework. The `tidyr` package enables users to format the data in a "tidy" way. The major advances were threefold. First, the packages simplified and made consistent many of the most common data manipulation

[1] At least the following time series data objects exist in R to solve general and specific needs: 'fts', 'its', 'irts', 'timeSeries', 'ti', 'ts', 'mts', 'zoo', and 'xts'. The sheer volume of options results in a complex decision process to determine which to use.

[2] The "tidy" data format is often referred to as "long" format versus "wide" format because the observations within the data set extend row-wise versus column-wise.

and summarization tasks in R. Second, the packages implement the *split-apply-combine* framework internally allowing users to work with grouped and nested data sets. Third, the packages incorporated the use of the pipe (`%>%`) from the `magrittr` package enabling functional verbs to follow an easy, efficient, and human-readable workflow.

purrr

The `purrr` package provides tools for applying functions to data frames. Similar to the traditional `apply` function in base R, the `map` function enables "mapping" functions row-wise within a data frame. The major advance is the ability to scale analysis. An analysis can be performed with "nested" data frames allowing users to apply functions across many independent data sets, a common task in data science. Further, the resulting data frame is "tidy", each analysis result is kept alongside the data that generated it.

tibble

The `tibble` package extends the traditional `data.frame` object by providing useful tools for creating and coercing objects to "tibbles" or "tidy" data frames.

ggplot2

The `ggplot2` package provides mechanisms for creating complex visualizations using a layered approach called the "grammar of graphics", which is discussed in *A Layered Grammar of Graphics* (Wickham, 2010). The `ggplot2` package is the primary package for creating static graphics in the "tidy" ecosystem.

lubridate

The `lubridate` package includes functions to manage date and date-time objects in R, which is discussed in *Dates and Times Made Easy with lubridate* (Grolemund and Wickham, 2011). The combination of `lubridate` with `dplyr` enables easy coercion of character class to date and date-time, filtering and subsetting on dates, and many more complex tasks that are essential to financial analysis. Further, combining `lubridate` with `ggplot2` enables graphical visualization using dates and date-times.

Different Data Structures, Each with Advantages

All of the major financial packages work within the "xts" system, which is specifically designed for time series analysis. The system works very well. The extensible time series, "xts", data structure is similar to a numeric "matrix". The primary difference is that row names consist of the date or date-time information. The objects can be subset by date or transformed to different periodicity very easily using a wide variety of flexible functions built to operate within the "xts" time-based system. Because of its focus, flexibility and design, the "xts" system manages single assets of the numeric time-based data extremely well. Multiple assets are handled by merging columns for different groups in a "wide" format, which is different than the "tidy" data representation (i.e. "long" format). The system has been designed to manage this data representation.

Much of the recent innovation in data science has occurred within the "tidy" ecosystem. With the entrance of the "tidyverse", grouping and iteratively applying functions (i.e. scaling analysis) has become easy, efficient, and a core principle. Further, advances in date and date-time functionality have enabled management of time series data within the "tibble" data structure. As the data science field grows, more innovative functionality will be developed within the "tidy" ecosystem, much of which will be (and is already) useful to the field of financial analysis.

The two systems have advantages that are needed within the realm of financial analysis. The "xts" system has the infrastructure to perform financial analytics. The "tidyverse" has the functions and data representation to seamlessly scale analysis using the data science workflow. Unfortunately, the two systems do not work well together. The "xts" system has underlying dependencies on the "matrix" data structure whereas the "tidy" ecosystem is built on top of the "tibble" ("tidy" data frame) data structure. The "matrix" structure is *homogenous*, the entire matrix must hold the same data type. In R, coercion rules take over when data types are combined within a "matrix". For example, combining a character and a numeric class will result in a character only matrix. As a result, the "xts" system can not be formatted in a "tidy" structure with each variable as a column, each observation as a row, and each value in a cell. The solution is to merge data sets together keeping the column names as the

observation. This works well in the "xts" system, but presents problems in the "tidyverse" system since the data is not represented following the "tidy" principles. Conversely, the "tibble" structure is *heterogenous*, each column can contain a different data type. This works extremely well for grouping observations together. For example, a "tibble" can contain symbols in one column and daily prices in another column. This enables applying functions to groups (e.g. symbol) for scaling analyses. The end result is two excellent systems that do not communicate easily. Without communication between systems, the full potential of financial analysis within R is limited.

To solve this problem, the `tidyquant` package was developed to integrate many of the "xts" based financial analysis packages into the "tidyverse". The central motivation behind the integration is to enable the user to gain the benefits of both systems. The `tidyquant` package works with "tidy" input and output, fitting seamlessly within the "tidy" ecosystem allowing for analysis to follow the data science workflow discussed in detail in *R for Data Science* (Wickham and Grolemund, 2017). Internally, `tidyquant` leverages the "xts" system as the engine to perform financial computations. This enables almost all of the functionality of `quantmod`, `TTR`, `PerformanceAnalytics`, `xts` and `zoo` to be used integrally (without switching back and forth) within the "tidy" ecosystem.

In the next section, an analysis is performed with the "tidyverse" to illustrate the benefits in manipulating grouped financial data. Then, in the final section, a demonstration of the `tidyquant` package is performed to illustrate some of its key benefits within the realm of financial analysis.

Benefits of the "tidyverse"

One of the primary benefits of the "tidyverse" is the ability to work with grouped data sets. The core concept is to split a data set into groups, apply functions to independent groups, then recombine the results. The value in this approach is the ability to scale analyses from one group to many groups and then compare the results. A simple example using the `FANG` data set from the `tidyquant` package illustrates this framework.

Start with a question: *What is the historical performance for the FANG tech stocks?*

Load the "tidyverse" libraries. The `tidyquant` package is loaded to get the `FANG` data set and to load the `tidyverse` package. Only `tidyverse` functionality will be used in this example.

```
library(tidyquant)
data("FANG")
```

Next, view the data set. The data consists of the historical daily stock prices for the "FANG"[3] tech stocks ("FB", "AMZN", "NFLX", and "GOOG") spanning from the beginning of 2013 to the end of 2016.

```
head(FANG)
dim(FANG)
```

symbol	date	open	high	low	close	volume	adjusted
FB	2013-01-02	27.44	28.18	27.42	28.00	69846400	28.00
FB	2013-01-03	27.88	28.47	27.59	27.77	63140600	27.77
FB	2013-01-04	28.01	28.93	27.83	28.76	72715400	28.76
FB	2013-01-07	28.69	29.79	28.65	29.42	83781800	29.42
FB	2013-01-08	29.51	29.60	28.86	29.06	45871300	29.06
FB	2013-01-09	29.67	30.60	29.49	30.59	104787700	30.59

[1] 4032 8

Historical performance is a function of the period returns (i.e. percentage change between current price and previous price). A wealth index is often used to assess the investment growth, which is the product of the period returns and asset value during the previous period. The data workflow is split into two steps.

1. Calculate the period returns
2. Calculate the cumulative performance using a wealth index

Calculate the period returns. A lag is needed to line up the previous period return with the current period return. The difference is then taken, and from the difference the period return is calculated as the percentage change from the initial value.

[3]The acronym, "FANG", was popularized by Jim Cramer of the popular CNBC show, Mad Money. http://www.investopedia.com/terms/f/fang-stocks-fb-amzn.asp.

For a more granular look at the code, each of the functions do the following. Only the "symbol", "date" and "adjusted" price columns are needed to start, so a `select` statement removes the unnecessary columns. Then, the data is grouped by "symbol" using the `group_by` function. The `mutate` function adds calculated columns to a data frame. The "lag" is calculated using the `lag` function from `dplyr`. The "diff" is the "adjusted" closing price minus the "lag". The "period.returns" is the percentage change from the "adjusted", which is "diff" divided by "adjusted". The `replace_na` function is used to switch `NA` values with zero. Note that an ungrouped "tibble" is returned due to the `replace_na` function.

```
FANG_returns <- FANG %>%
    select(-(open:volume)) %>%
    group_by(symbol) %>%
    mutate(lag = lag(adjusted),
           diff = adjusted - lag,
           period.returns = diff / lag) %>%
    replace_na(list(period.returns = 0))
head(FANG_returns)
dim(FANG_returns)
```

symbol	date	adjusted	lag	diff	period.returns
FB	2013-01-02	28.00	NA	NA	0.0000000
FB	2013-01-03	27.77	28.00	-0.230000	-0.0082143
FB	2013-01-04	28.76	27.77	0.990000	0.0356500
FB	2013-01-07	29.42	28.76	0.660000	0.0229485
FB	2013-01-08	29.06	29.42	-0.360001	-0.0122366
FB	2013-01-09	30.59	29.06	1.530001	0.0526497

[1] 4032 6

Now, the wealth index can be calculated. The wealth index is the cumulative product of the period returns multiplied by the initial investment value. An initial investment of $10K USD is used as an example.

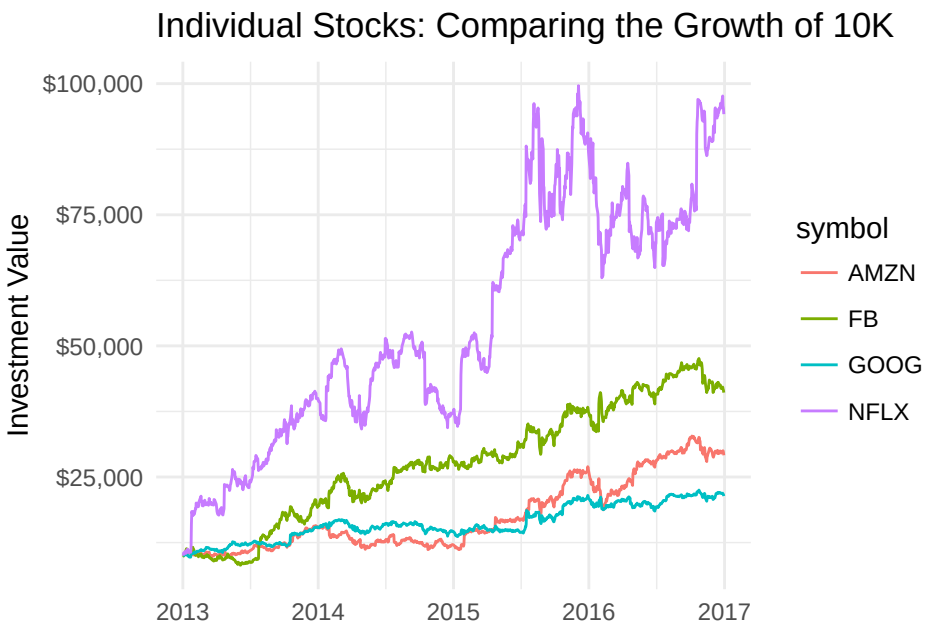
```
initial_investment <- 10000
FANG_performance <- FANG_returns %>%
    group_by(symbol) %>%
    mutate(wealth.index = initial_investment * cumprod(1 + period.returns))
head(FANG_performance)
dim(FANG_performance)
```

symbol	date	adjusted	lag	diff	period.returns	wealth.index
FB	2013-01-02	28.00	NA	NA	0.0000000	10000.000
FB	2013-01-03	27.77	28.00	-0.230000	-0.0082143	9917.857
FB	2013-01-04	28.76	27.77	0.990000	0.0356500	10271.429
FB	2013-01-07	29.42	28.76	0.660000	0.0229485	10507.143
FB	2013-01-08	29.06	29.42	-0.360001	-0.0122366	10378.571
FB	2013-01-09	30.59	29.06	1.530001	0.0526497	10925.000

[1] 4032 7

Lastly, the performance of the individual stocks using daily compounding of the returns are visualized using `ggplot2`.

```
FANG_performance %>%
    ggplot(aes(x = date, y = wealth.index, color = symbol)) +
    geom_line() +
    labs(title = "Individual Stocks: Comparing the Growth of 10K",
         x = "", y = "Investment Value") +
    scale_y_continuous(labels = scales::dollar) +
    theme_minimal()
```



The "tidy" ecosystem can be used to solve a wide range of complex problems because it enables grouping of data sets and applying a data science workflow. In the next section, examples are presented to illustrate the power of combining the financial analysis tools from the "xts" system with the "tidyverse" for financial applications.

Financial Applications with "tidyquant"

The financial analysis packages are difficult or impossible to use within the "tidy" ecosystem. A new tool is needed, `tidyquant`. The `tidyquant` package has one major advantage: it integrates the "xts" based financial packages with the "tidyverse" to enable the data science workflow and "tidy" ecosystem functionality to be applied to financial analysis. The innovation is relatively minor on a conceptual level, but the net effect is significant in that the tools within the "tidy" ecosystem are now unlocked for financial analysis. The following examples illustrate the new capability.

Example 1: Evaluating Risk vs Reward

Financial analysis is almost always a trade off between risk and reward. Reward is measured by the growth of an investment. Risk is typically associated with volatility. Often when beginning an analysis, one wishes to understand the components of a market basket of stocks on the basis of this risk-reward trade off in order to screen investments.

Start with a question: *How does the market value risk versus reward?*

This question is answerable by comparing the risk and reward characteristics for a basket of stocks viewed as a reasonable proxy for the "market". The observations within the market are stocks. Stocks also have observations, or historical prices, which can be obtained over time and can be used to evaluate various statistical qualities that relate to risk versus reward. The reward is the return performance (i.e. returns), which is the positive or negative percentage change between observations. When accumulated across a frequency such as daily, the large set of returns has statistical properties that can be measured. To simplify the question, the average and standard deviation of the returns can be used as a general proxy of risk versus reward. When pooled together, the stocks can thus be compared to expose general performance trends within the market. [4]

To start, load `tidyquant`, which loads all of the packages needed to evaluate risk versus reward.

```
# Loads tidyquant, tidyverse, lubridate, quantmod, TTR, xts, zoo, PerformanceAnalytics
library(tidyquant)
```

Next, collect ticker data. The question implies that a large sample of stock data is needed to evaluate how performance and risk is valued within the market. The S&P 500 index is a good place to

[4] Using the mean and standard deviation of stock returns has roots in Brownian motion, known as a Stochastic Process, which is often used in Monte Carlo simulation to predict the range of future returns within a confidence interval based on the properties of past returns.

start. `tidyquant` includes a function, `tq_index`, which returns the stock symbols and company names for every stock within an index.

```
sp500 <- tq_index("SP500")
head(sp500)
dim(sp500)
```

symbol	company
MMM	3M
ABT	ABBOTT LABORATORIES
ABBV	ABBVIE INC
ACN	ACCENTURE
ATVI	ACTIVISION BLIZZARD
AYI	ACUITY BRANDS

[1] 502 2

Next, the historical stock prices are needed for each stock within the S&P 500 index. The stock prices are easy to retrieve at scale using the function, `tq_get`, with the "get" option, `get = "stock.prices"`. `tq_get` is a wrapper for `quantmod::getSymbols`, which enables passing the underlying function parameters `from` and `to`. The input to `tq_get` is `data`, which can be a single stock symbol, a vector of stock symbols, or a tibble with stock symbols in the first column. The latter is passed to `tq_get` using the pipe (`%>%`). The function may take a few minutes to run because it is downloading the past ten years of daily open, high, low, close, volume, and adjusted stock prices for the entire S&P 500 index into one "tidy" data frame. Reviewing the results indicates that the prices were retrieved in entirety. The resulting "tibble" has 1,205,845 rows and 502 unique symbols.

```
sp500_stock_prices <- sp500 %>%
    tq_get(get  = "stock.prices",
           from = "2007-01-01",
           to   = "2017-01-01")
head(sp500_stock_prices)
dim(sp500_stock_prices)
```

symbol	company	date	open	high	low	close	volume	adjusted
MMM	3M	2007-01-03	77.53	78.85	77.38	78.26	3781500	60.31064
MMM	3M	2007-01-04	78.40	78.41	77.45	77.95	2968400	60.07174
MMM	3M	2007-01-05	77.89	77.90	77.01	77.42	2765200	59.66330
MMM	3M	2007-01-08	77.42	78.04	76.97	77.59	2434500	59.79431
MMM	3M	2007-01-09	78.00	78.23	77.44	77.68	1896800	59.86367
MMM	3M	2007-01-10	77.31	77.96	77.04	77.85	1787500	59.99468

[1] 1205845 9

Next, the prices are grouped by stock symbol to calculate the logarithmic daily returns for each stock. The transform is applied using `tq_transmute`, which is used in situations where periodicity changes (or can change). The `select` argument is use to select a column (or columns) and send these prices to the `periodReturn` function (`mutate_fun = periodReturn`). The additional arguments `period = "daily"` and `type = "log"` are passed to the mutating function, `periodReturn`. The daily log returns (DLR) for each of the 502 groups of stock symbols is generated below.

```
sp500_returns <- sp500_stock_prices %>%
    group_by(symbol) %>%
    tq_transmute(select = adjusted, mutate_fun = periodReturn,
                 period = "daily", type = "log", col_rename = "dlr")
head(sp500_returns)
dim(sp500_returns)
```

symbol	date	dlr
MMM	2007-01-03	0.0000000
MMM	2007-01-04	-0.0039691
MMM	2007-01-05	-0.0068224
MMM	2007-01-08	0.0021934
MMM	2007-01-09	0.0011593
MMM	2007-01-10	0.0021860

[1] 1205845 3

Next, the mean and standard deviation of the daily log returns are used to evaluate and compare the stocks. The easiest way is to use `tq_performance`, which applies the `PerformanceAnalytics` performance functions to "tidy" data frames of asset or portfolio returns. The `table.Stats` function returns the arithmetic mean and standard deviation along with a number of other useful statistics that characterize the returns.

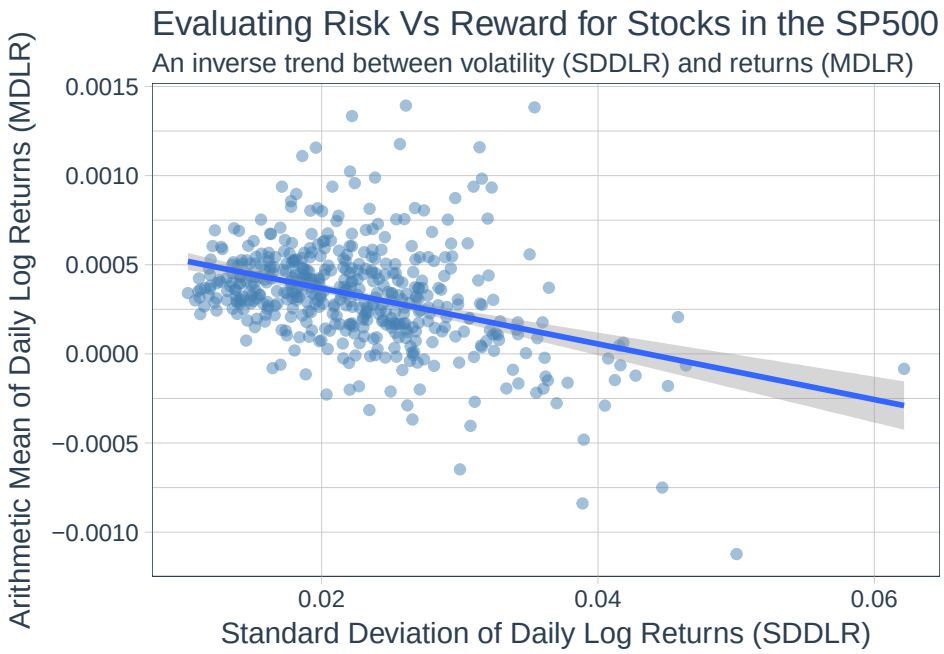
```
sp500_stats <- sp500_returns %>%
    tq_performance(Ra = dlr, performance_fun = table.Stats, ci = 0.95, digits = 6)
sp500_stats %>%
    select(1:5) %>%
    head()
dim(sp500_stats)
```

symbol	ArithmeticMean	GeometricMean	Kurtosis	LCLMean(0.95)
MMM	0.000431	0.000331	5.806040	-0.000120
ABT	0.000302	0.000215	6.294066	-0.000211
ABBV	0.000714	0.000565	3.868658	-0.000350
ACN	0.000543	0.000404	8.532566	-0.000108
ATVI	0.000604	0.000359	10.735599	-0.000263
AYI	0.000701	0.000420	6.011008	-0.000225

[1] 502 17

The data required to visualize risk versus reward is complete. A plot of the mean daily log returns (MDLR) versus the standard deviation of daily log returns (SDDLR) shows the relationship. Note that observations (stocks) with fewer than five years of trading days (5 years x 252 trade days per year) are removed from the visualization to yield a long-run perspective.

```
sp500_stats %>%
    filter(Observations >= 252 * 5) %>%
    ggplot(aes(x = Stdev, y = ArithmeticMean)) +
    geom_point(alpha = 0.5, col = "steelblue") +
    geom_smooth(method = "lm") +
    labs(title = "Evaluating Risk Vs Reward for Stocks in the SP500",
         subtitle = "An inverse trend between volatility (SDDLR) and returns (MDLR)",
         x = "Standard Deviation of Daily Log Returns (SDDLR)",
         y = "Arithmetic Mean of Daily Log Returns (MDLR)") +
    theme_tq()
```



By comparing an entire basket of stocks, reasonable trading strategies can be developed. For example, creating a portfolio that minimizes volatility may result in higher performance over the long run. This is easily seen in the trend line indicating an inverse trend between volatility (SDDLR) and returns (MDLR). Further, once trading strategies are developed, screening is easily implemented by filtering the data set.

```
sp500_high_performers <- sp500_stats %>%
    filter(Observations >= 252 * 5,
           Stdev <= 0.02,
           ArithmeticMean >= 0.0005) %>%
    arrange(desc(ArithmeticMean))
sp500_high_performers %>%
    select(1:5) %>%
    head()
```

symbol	ArithmeticMean	GeometricMean	Kurtosis	LCLMean(0.95)
TDG	0.00116	0.00097	7.96994	0.00039
FBHS	0.00111	0.00094	2.58643	0.00011
DLPH	0.00094	0.00079	4.21629	0.00000
ROST	0.00090	0.00073	5.15896	0.00019
ORLY	0.00086	0.00070	10.20942	0.00016
AGN	0.00082	0.00067	6.06114	0.00013

```
[1] 52 17
```

Example 2: Visualizing Return Consistency Among "High Performers"

Example 1 can be expanded upon using the notion that excessive volatility is a factor that the market tends to penalize. In this example, the goal is to visualize returns of the top ranked prospects from Example 1 to determine if the stock is consistently growing. This example will introduce a new package called `purrr`, which is used to map functions to "nested" data frames.

Start with a question: *Of the best stocks from Example 1, which offer the most consistent growth?*

The easiest way to answer this question is to visualize the mean daily log returns (MDLR) by year. Collecting the means by year can be accomplished in a number of ways. The `purrr` package offers a new approach by mapping functions to tibble groups similar to the `apply` function in base R. The nice thing about `map` is that if a function can be applied to one group of the data, it can be "mapped" to many groups. This example will proceed as follows:

1. Retrieve the symbols of the top n performers. The example uses the top 10, but any number can be selected.

2. Filter and "nest" the returns for the top performers.
3. Create a custom function to map to each of the nested "tibbles".
4. Map the custom function to the "tibble".
5. Unnest the data frame.
6. Visualize the results.

First, get the symbols of the top performers. Since the `sp_500_high_performers` data frame is arranged in decreasing MDLR, the `slice` function can be used to collect the top performers.

```
n <- 10
high_performers <- sp500_high_performers %>%
    ungroup() %>%
    select(symbol, ArithmeticMean, Stdev) %>%
    slice(1:n)
high_performers
```

symbol	ArithmeticMean	Stdev
TDG	0.001157	0.019580
FBHS	0.001110	0.018597
DLPH	0.000938	0.017123
ROST	0.000897	0.018166
ORLY	0.000857	0.017784
AGN	0.000825	0.017793
EW	0.000817	0.019710
DLTR	0.000803	0.019179
AZO	0.000753	0.015612
TJX	0.000708	0.017008

Next, filter the S&P 500 returns tibble to only the stocks of interest and "nest" the tibble. Nesting works on groups, so all of the data that is grouped on gets wrapped up into a list column.

```
high_performer_returns_nested <- sp500_returns %>%
    filter(symbol %in% high_performers$symbol) %>%
    nest()
high_performer_returns_nested
```

```
#> # A tibble: 10 × 2
#>    symbol                 data
#>     <chr>               <list>
#> 1     AGN <tibble [2,518 × 2]>
#> 2     AZO <tibble [2,518 × 2]>
#> 3    DLPH <tibble [1,288 × 2]>
#> 4    DLTR <tibble [2,518 × 2]>
#> 5      EW <tibble [2,518 × 2]>
#> 6    FBHS <tibble [1,332 × 2]>
#> 7    ORLY <tibble [2,518 × 2]>
#> 8    ROST <tibble [2,518 × 2]>
#> 9     TJX <tibble [2,518 × 2]>
#> 10    TDG <tibble [2,518 × 2]>
```

Once the tibble is nested, create a custom function to apply to each of the nested tibbles. The function `calculate_mdlr_by_year` transforms the daily log returns to mean daily log returns by year.

```
calculate_mdlr_by_year <- function(dlr) {
    dlr %>%
        mutate(year = year(date)) %>%
        group_by(year) %>%
        summarize(mdlr.by.year = mean(dlr))
}
```

Map the custom function, `calculate_mdlr_by_year`, to the nested data frame. This is done using `mutate` with `map`. The "data" column is no longer needed, so it is removed with `select`.

```
high_performer_returns_mapped <- high_performer_returns_nested %>%
    mutate(mdlr.by.year = map(data, calculate_mdlr_by_year)) %>%
    select(-data)
high_performer_returns_mapped
```

```
#> # A tibble: 10 × 2
#>    symbol      mdlr.by.year
#>     <chr>            <list>
#> 1     AGN <tibble [10 × 2]>
#> 2     AZO <tibble [10 × 2]>
#> 3    DLPH  <tibble [6 × 2]>
#> 4    DLTR <tibble [10 × 2]>
#> 5      EW <tibble [10 × 2]>
#> 6    FBHS  <tibble [6 × 2]>
#> 7    ORLY <tibble [10 × 2]>
#> 8    ROST <tibble [10 × 2]>
#> 9     TJX <tibble [10 × 2]>
#> 10    TDG <tibble [10 × 2]>
```

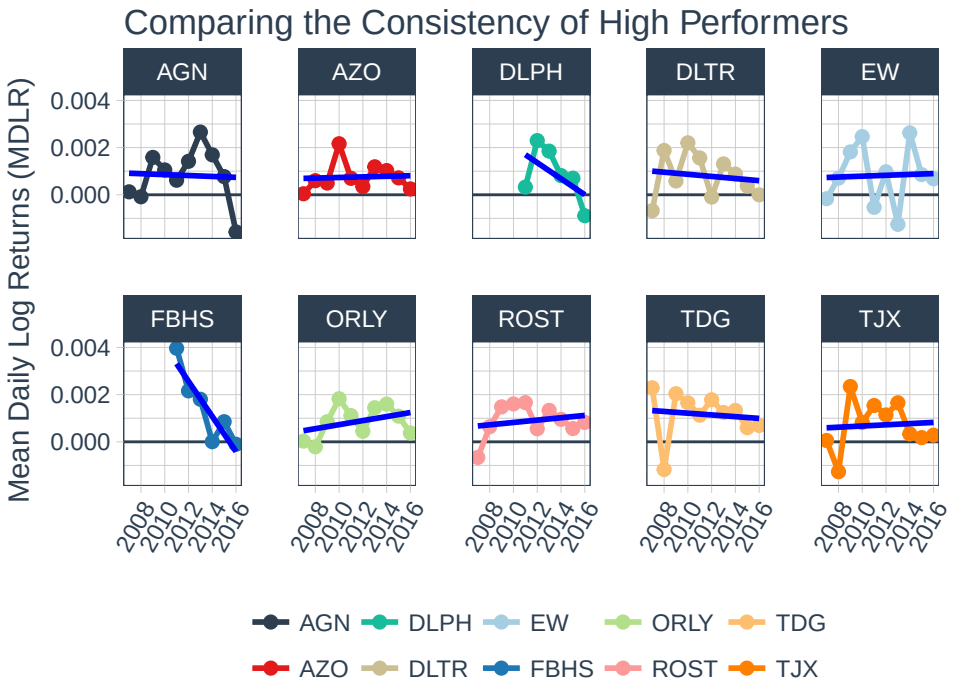
Now, `unnest` the "mdlr.by.year" column to return the MDLR by year for each of the high performing stocks.

```
high_performer_returns_mapped %>%
    unnest() %>%
    head()
```

symbol	year	mdlr.by.year
AGN	2007	0.0001253
AGN	2008	-0.0000839
AGN	2009	0.0015845
AGN	2010	0.0010532
AGN	2011	0.0006171
AGN	2012	0.0014174

Finally, visualize the results using `ggplot2`. The mean daily log returns when averaged by year yields a different picture then what the results of Example 1 portrayed. Some of the stocks exhibit consistent growth (e.g. ROST) while others have more volatile returns or negative returns in multiple years indicating a lack of consistency (e.g. DLPH).

```
high_performer_returns_mapped %>%
    unnest() %>%
    ggplot(aes(x = year, y = mdlr.by.year, color = symbol)) +
    geom_hline(yintercept = 0, color = palette_light()[[1]]) +
    geom_line(size = 1) +
    geom_point(size = 2) +
    geom_smooth(method = "lm", se = FALSE, color = "blue") +
    scale_color_tq() +
    theme_tq() +
    facet_wrap(~ symbol, ncol = 5) +
    labs(title = "Comparing the Consistency of High Performers",
         y = "Mean Daily Log Returns (MDLR)", color = "") +
    theme(axis.text.x = element_text(angle = 60, hjust = 1),
          axis.title.x = element_blank())
```



This analysis can be expanded even further. With the data science modeling workflow discussed in *R for Data Science* (Wickham and Grolemund, 2017), a financial analyst or engineer can apply prediction models to the data set (refer to the chapters on modeling using the `broom` package). Custom screening metrics can be developed such as using the coefficients of a regression model, counting the times an investment falls below zero, calculating standard deviation in MDLR by year, and more. Screening investments within an entire index or exchange becomes relatively easy, even with complex metrics. The blog article, *Russell 2000 Quantitative Stock Analysis in R: Six Stocks with Amazing, Consistent Growth* (Business Science), elaborates on the details behind using the `purrr` package to create complex screening metrics.

Example 3: Evaluating Performance of Multiple Portfolio Blends

Portfolio aggregation (the aggregation of assets using a weighted blend) is a useful technique to reduce risk while maintaining acceptable returns. The technique is typically used in portfolio attribution, comparing portfolios against a benchmark. In this example, the goal is to evaluate a few blended portfolios of FANG stocks. However, more important is the conceptual idea that the number of weighted portfolio blends and the number of underlying assets can easily be increased to scale the analysis. As the example progresses, consider how this approach could be applied to an entire index of assets and hundreds of weighted blends.

Throughout the past half decade, the FANG stocks have experienced a unique combination of high returns and high volatility, making them a good example to use in a blended portfolio that reduces downside risk but yields high returns.

Start with a question: *What portfolio blends reduce downside risk while maximizing return?*

Several portfolio blends will be evaluated to see the historical effect on returns since 2013: [5]

- Portfolio 1: 50% FB, 25% AMZN, 25% NFLX, 0% GOOG
- Portfolio 2: 0% FB, 50% AMZN, 25% NFLX, 25% GOOG
- Portfolio 3: 25% FB, 0% AMZN, 50% NFLX, 25% GOOG
- Portfolio 4: 25% FB, 25% AMZN, 0% NFLX, 50% GOOG

First, collect the data for the "FANG" stocks.

```
FANG <- c("FB", "AMZN", "GOOG", "NFLX") %>%
    tq_get(get = "stock.prices")
head(FANG)
dim(FANG)
```

[5] 2013 was the first full year of trading data for FB. Portfolio aggregation prior to 2013 using FB cannot be analyzed.

symbol	date	open	high	low	close	volume	adjusted
FB	2012-05-18	42.05	45.00	38.00	38.23	573576400	38.23
FB	2012-05-21	36.53	36.66	33.00	34.03	168192700	34.03
FB	2012-05-22	32.61	33.59	30.94	31.00	101786600	31.00
FB	2012-05-23	31.37	32.50	31.36	32.00	73600000	32.00
FB	2012-05-24	32.95	33.21	31.77	33.03	50237200	33.03
FB	2012-05-25	32.90	32.95	31.11	31.91	37149800	31.91

[1] 9049 8

Next, calculate monthly returns. Use `group_by` to group on the "symbol" column, and `tq_transmute` to transform the adjusted prices to monthly arithmetic returns. Note that "FB" was only actively traded for a full year beginning in 2013, so the investments will be compared starting from that year.

```
FANG_returns <- FANG %>%
    filter(date >= ymd("2013-01-01"),
           date <  ymd("2017-01-01")) %>%
    group_by(symbol) %>%
    tq_transmute(select     = adjusted,
                 mutate_fun = periodReturn,
                 period     = "monthly",
                 col_rename = "returns")
slice(FANG_returns, 1:2)
dim(FANG_returns)
```

symbol	date	returns
AMZN	2013-01-31	0.0318293
AMZN	2013-02-28	-0.0046328
FB	2013-01-31	0.1064286
FB	2013-02-28	-0.1204003
GOOG	2013-01-31	0.0448531
GOOG	2013-02-28	0.0602232
NFLX	2013-01-31	0.7958918
NFLX	2013-02-28	0.1382232

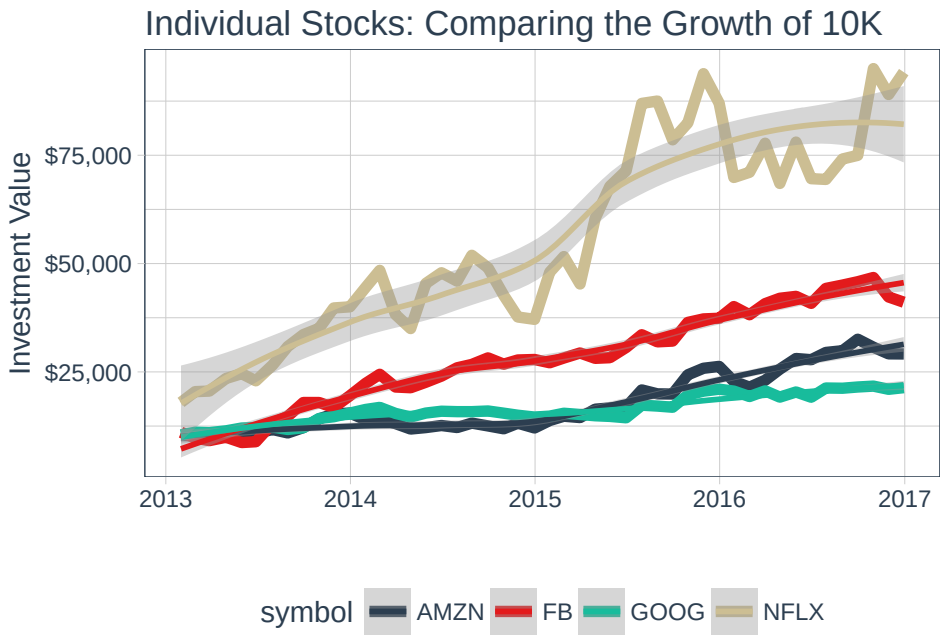
[1] 192 3

Individual Asset Performance

Before the portfolios are generated and evaluated, it is important to visualize and assess the performance of the individual asset returns. The first visualization uses a wealth index, which takes an initial investment as an input and returns a visualization showing how the investment would have grown over the specified time period. From the visualization, NFLX was the best performer, but it also experienced some significant drops along the way.

```
init_investment <- 10000
FANG_wealth <- FANG_returns %>%
    mutate(wealth.index = init_investment * cumprod(1 + returns))

FANG_wealth %>%
    ggplot(aes(x = date, y = wealth.index, color = symbol)) +
    geom_line(size = 2) +
    geom_smooth(method = "loess") +
    labs(title = "Individual Stocks: Comparing the Growth of 10K",
         x = "", y = "Investment Value") +
    theme_tq() +
    scale_color_tq() +
    scale_y_continuous(labels = scales::dollar)
```



Because of the volatility, the risk should be evaluated as well. One popular method to measure risk is using value at risk (VaR). VaR measures the worst expected loss over a given time interval under normal market conditions, at a given confidence level (Bacon, 2004). Implementing VaR can be done using `tq_performance` with the `PerformanceAnalytics` function, `VaR`, to estimate risk measures across the FANG stocks. While NFLX and AMZN returns are stellar, the VaR estimate is also double that of FB and GOOG.

```
VaR_FANG <- FANG_returns %>%
    tq_performance(Ra = returns, Rb = NULL, performance_fun = VaR, p = 0.95) %>%
    rename(VaR.monthly = VaR)
VaR_FANG
```

symbol	VaR.monthly
FB	-0.0564358
AMZN	-0.0968790
GOOG	-0.0566269
NFLX	-0.0977580

From the results, a portfolio consisting entirely of one of the assets could yield great results, but it's not for the risk averse. A better method might be to use a weighted aggregation within a portfolio.

Portfolio Performance

Weighted portfolio aggregation involves three steps:

1. Make a portfolio by repeating the stock returns table *n* times, once for each portfolio desired.
2. Create a weights table to map weights by asset and portfolio.
3. Aggregate the portfolios using `tq_portfolio`, a wrapper for `PerformanceAnalytics::Return.portfolio`.

Step 1: Repeat the stock returns table *n* times, once for each portfolio desired Use `tq_repeat_df` to repeat `n = 4` times. This function extends the tibble row-wise, adding an index column named "portfolio" and grouping by "portfolio". Four portfolio groups are generated.

```
FANG_returns_mult <- FANG_returns %>%
    tq_repeat_df(n = 4)
slice(FANG_returns_mult, 1:2)
dim(FANG_returns_mult)
```

portfolio	symbol	date	returns
1	FB	2013-01-31	0.1064286
1	FB	2013-02-28	-0.1204003
2	FB	2013-01-31	0.1064286
2	FB	2013-02-28	-0.1204003
3	FB	2013-01-31	0.1064286
3	FB	2013-02-28	-0.1204003
4	FB	2013-01-31	0.1064286
4	FB	2013-02-28	-0.1204003

[1] 768 4

Step 2: Create a Weights Table Construct a weights table using the portfolio blending parameters, which will be mapped to the stocks within each of the portfolios in the next step. The format of the weights table is critical. The table must have three columns, "portfolio", "stocks", and "weights". Make sure to group by the "portfolio" column.

- Portfolio 1: 50% FB, 25% AMZN, 25% NFLX, 0% GOOG
- Portfolio 2: 0% FB, 50% AMZN, 25% NFLX, 25% GOOG
- Portfolio 3: 25% FB, 0% AMZN, 50% NFLX, 25% GOOG
- Portfolio 4: 25% FB, 25% AMZN, 0% NFLX, 50% GOOG

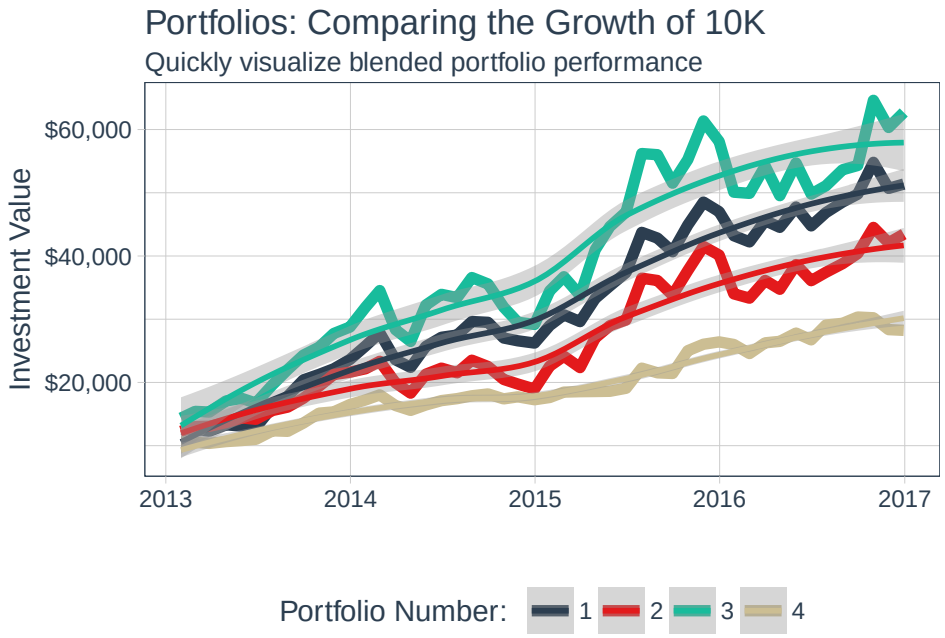
```
weights <- c(0.50, 0.25, 0.25, 0.00,
             0.00, 0.50, 0.25, 0.25,
             0.25, 0.00, 0.50, 0.25,
             0.25, 0.25, 0.00, 0.50)
weights_table <- tibble(stocks = c("FB", "AMZN", "NFLX", "GOOG")) %>%
    tq_repeat_df(n = 4) %>%
    bind_cols(tibble(weights)) %>%
    group_by(portfolio)
weights_table
```

portfolio	stocks	weights
1	FB	0.50
1	AMZN	0.25
1	NFLX	0.25
1	GOOG	0.00
2	FB	0.00
2	AMZN	0.50
2	NFLX	0.25
2	GOOG	0.25
3	FB	0.25
3	AMZN	0.00
3	NFLX	0.50
3	GOOG	0.25
4	FB	0.25
4	AMZN	0.25
4	NFLX	0.00
4	GOOG	0.50

Step 3: Aggregate the Portfolios with tq_portfolio Aggregate the portfolios using `tq_portfolio`, a wrapper for `PerformanceAnalytics::Return.portfolio`. The `Return.portfolio` function also has additional arguments to create a wealth index. Setting `wealth.index = TRUE` and multiplying the result by the initial investment value of $10,000 returns a wealth index. The performance of the various blended portfolios is visualized using `ggplot2` with the aesthetic argument, `color = factor(portfolio)`.

```
# Aggregate portfolio with tq_portfolio. Pass wealth.index = TRUE
init_investment <- 10000
FANG_portfolio_wealth <- FANG_returns_mult %>%
    tq_portfolio(assets_col = symbol, returns_col = returns,
                 weights = weights_table, wealth.index = TRUE,
                 col_rename = "wealth.index") %>%
    mutate(wealth.index = wealth.index * init_investment)

FANG_portfolio_wealth  %>%
    ggplot(aes(x = date, y = wealth.index, color = factor(portfolio))) +
    geom_line(size = 2) +
    geom_smooth(method = "loess") +
    labs(title = "Portfolios: Comparing the Growth of 10K",
         subtitle = "Quickly visualize blended portfolio performance",
         x = "", y = "Investment Value",
         color = "Portfolio Number: ") +
    theme_tq() +
    scale_color_tq() +
    scale_y_continuous(labels = scales::dollar)
```



Finally, the risk is assessed in the same manner as the individual assets. The portfolio aggregation is performed without the `wealth.index` option, which aggregates uncompounded returns. The returns are "piped" to the `tq_performance` function, which returns the VaR.

```
VaR_portfolio <- FANG_returns_mult %>%
    tq_portfolio(assets_col = symbol, returns_col = returns,
                 weights = weights_table, col_rename = "returns") %>%
    tq_performance(Ra = returns, Rb = NULL, performance_fun = VaR) %>%
    rename(VaR.monthly = VaR)
VaR_portfolio
```

portfolio	VaR.monthly
1	-0.0853386
2	-0.1055108
3	-0.1077418
4	-0.0526966

The two tables below summarize the results of the individual and portfolio analyses, respectively. The Portfolio 1 blend is an attractive combination of growth with lower risk: over 5.1X return with over

a 1% decrease in the VaR of both NFLX and AMZN. To further optimize, one could run algorithms to identify the "best" minimum variance portfolio.

symbol	value.end	VaR.monthly
AMZN	29142.67	-0.0968790
FB	41089.29	-0.0564358
GOOG	21364.41	-0.0566269
NFLX	94185.41	-0.0977580

portfolio	value.end	VaR.monthly
1	51376.66	-0.0853386
2	43458.79	-0.1055108
3	62706.13	-0.1077418
4	28240.19	-0.0526966

Conclusion

With the creation of the "xts" and "tidyverse" systems, high performance tools are available to perform financial analysis and to scale data science in R. Because of the separation in development of the two systems, the ability to scale financial analysis is limited. `tidyquant` unlocks the data science workflow for financial applications, increasing the scale and modeling capability of financial analyses.

As scale and data science functionality increases, the vast array of possibilities becomes limited only by the data storage and processing capabilities of the computers on which the computations are performed. With advances in cloud technology, parallel processing, and optimization algorithms, the tools are available to scale analyses to terabytes worth of financial data in real time. `tidyquant` is being built with extendibility in mind, allowing for potential integration with Apache Spark™ (large-scale data processing), `multidplyr` (localized multicore parallel processing), and even Bloomberg Terminal® integration for access to better performance and higher quality data. Further, with advances in artificial intelligence and machine learning, the potential for financial analysis at scale is in its infancy. This is an exciting time. As `tidyquant`, complimentary R software packages, and technology in general evolves, financial analysis will certainly benefit from the powerful speed, scale, and autonomous learning that is rapidly becoming available.

Bibliography

C. Bacon. *Practical Portfolio Performance Measurement and Attribution*. John Wiley and Sons Ltd, West Sussex, England, 2004. [p2, 14]

Business Science. URL http://www.business-science.io/investments/2016/11/30/Russell2000_Analysis.html. [p12]

G. Grolemund and H. Wickham. Dates and times made easy with lubridate. *Journal of Statistical Software*, 40(3):1—25, 2011. URL http://www.jstatsoft.org/v40/i03/. [p3]

TimelyPortfolio. URL https://timelyportfolio.github.io/rCharts_time_series/history.html. [p1]

H. Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1): 3–28, 2010. doi: 10.1198/jcgs.2009.07098. [p3]

H. Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1): 1—29, 2011. URL http://www.jstatsoft.org/v40/i01/. [p2]

H. Wickham. Tidy data. *The Journal of Statistical Software*, 59, 2014. URL http://www.jstatsoft.org/v59/i10/. [p2]

H. Wickham, 2017. URL https://www.rstudio.com/resources/videos/data-science-in-the-tidyverse/. [p2]

H. Wickham and G. Grolemund. *R for Data Science*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA, 2017. URL http://r4ds.had.co.nz/. [p2, 4, 12]

Matt Dancho
Business Science

mdancho@business-science.io

Davis Vaughan
Business Science

dvaughan@business-science.io