New Tools for Performing Financial Analysis Using the "Tidy" Split-Apply-Combine Framework

Matt Dancho
Davis Vaughan

February 23, 2017

Abstract

Financial analysis and data science in the R programming language have followed two separate yet innovative paths resulting in two different but important systems: "xts" and "tidyverse". Because of the separation in development, the two systems are difficult to use together, which limits the full potential of financial analysis within R. The `tidyquant` package solves this problem by integrating best financial analysis packages in the "xts" system with the "tidy" ecosystem, which unlocks the benefits of the *"split-apply-combine"* framework and enables scaling financial analyses.

Two usage cases are discussed to illustrate the potential. The first example uses the scaling capabilities to provide an answer to how the "market" values risk versus reward. The second example evaluates the performance of multiple portfolios using weighted blends. These examples just scratch the surface of the full potential as technology develops, and the future possibilities are briefly addressed.

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 plotting, which highlights the inception of several of the most influential R financial and time series packages. Several of the influential packages are worth describing in more detail as these create much of the foundation of R in Finance currently.

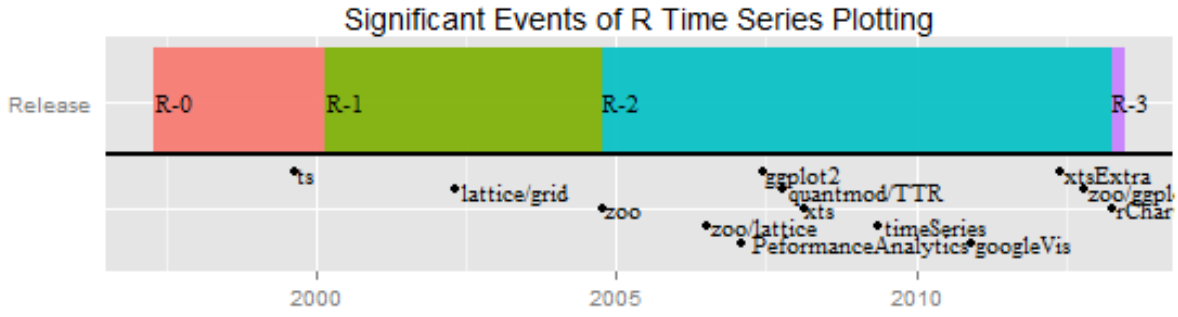


Figure 1: Significant Events of R Time Series Plotting

quantmod / TTR

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

xts / zoo

The *"Extensible Time Series"* or `xts` package along with the `zoo` package includes 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* [FOOTNOTE] under one object class, `xts`.

```
# FOOTNOTE
At least the following time series data objects exist in R to solve general and specific needs: `fts`,
```

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 [REF]. These functions enable analysis of individual asset and portfolio (the weighted aggregation of multiple assets) returns using popular statistical methods for measuring performance.

```
# REF
@Book{Bacon2004,
  author = {Carl Bacon},
  title = {Practical Portfolio Performance Measurement and Attribution},
  publisher = {John Wiley & Sons Ltd},
  year = {2004},
  address = {West Sussex, England}
}
```

New Tools: tidyverse

In parallel with the progression within the R in Finance community, developers at *RStudio* have been developing 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 "tidy" data, which is defined in *"Tidy Data"* [INSERT REF]. Further, the "tidyverse" packages and the data analysis workflow are documented in the online text, *"R for Data Science"* [REF], 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.

```
# REF
@Article{tidy-data,
  author = {Hadley Wickham},
  issue = {10},
  journal = {The Journal of Statistical Software},
  selected = {TRUE},
  title = {Tidy data},
  url = {http://www.jstatsoft.org/v59/i10/},
  volume = {59},
  year = {2014},
```

```
  bdsk-url-1 = {http://www.jstatsoft.org/v59/i10/},
}
```

```
# REF - Add website and fix authors
@Book{R4DS2017,
  author = {Hadley Wickham and Garrett Grolemund},
  title = {R for Data Science},
  publisher = {O'Reilly Media, Inc.},
  year = {2017},
  address = {1005 Gravenstein Highway North, Sebastopol, CA},
  url = "http://r4ds.had.co.nz/"
}
```

dplyr / tidyr

The `dplyr` and `tidyr` packages provide tools to clean and manipulate data using the *"split-apply-combine"* framework popularized by Hadley Wickham in *"The Split-Apply-Combine Strategy for Data Analysis"* [REF]. The major advances were threefold. First, the packages simplified and made consistent many of the most common data manipulation and summarization tasks in R. Second, the packages enable the *"split-apply-combine"* framework on grouped data sets. Third, the packages incorporated the use of the pipe (`%>%`) from the `magrittr` package enabling the functional verbs to follow an easy, efficient, and human-readable workflow.

```
# REF
@Article{plyr,
  author = {Hadley Wickham},
  journal = {Journal of Statistical Software},
  number = {1},
  pages = {1--29},
  selected = {TRUE},
  title = {The split-apply-combine strategy for data analysis},
  url = {http://www.jstatsoft.org/v40/i01/},
  volume = {40},
  year = {2011},
  bdsk-url-1 = {http://www.jstatsoft.org/v40/i01/},
}
```

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, which is a common task in data science. Further, the resulting data frame is "tidy", meaning 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"* [INSERT REF]. The `ggplot2` package is the primary package for creating static graphics in the "tidy" ecosystem.

```
# REF
@Article{layered-grammar,
  author = {Hadley Wickham},
  doi = {10.1198/jcgs.2009.07098},
  journal = {Journal of Computational and Graphical Statistics},
  number = {1},
  pages = {3-28},
  selected = {TRUE},
  title = {A layered grammar of graphics},
  volume = {19},
  year = {2010},
  bdsk-url-1 = {http://dx.doi.org/10.1198/jcgs.2009.07098},
}
```

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"* [INSERT REF]. The combination of using `lubridate` with `dplyr` enables coercing 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.

```
@Article{lubridate,
  author = {Garrett Grolemund and Hadley Wickham},
  journal = {Journal of Statistical Software},
  number = {3},
  pages = {1--25},
  title = {Dates and Times Made Easy with lubridate},
  url = {http://www.jstatsoft.org/v40/i03/},
  volume = {40},
  year = {2011},
  bdsk-url-1 = {http://www.jstatsoft.org/v40/i03/},
}
```

Divergent Philosophies, 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 much like a numeric matrix, the only major difference being row names consisting of the date or date-time information. The advantage of the "xts" system is the ability to manage numeric data with date or date-time references. The objects can be subset or transformed to different periodicity very easily. The disadvantage is the application is very strict to numeric data, but because of its focus it manages the numeric, time-based data extremely well.

Much of the recent innovation in data analysis has occurred within the "tidy" ecosystem. With the entrance of the "tidyverse", scaling analysis using the *"split-apply-combine"* framework has become easy, efficient, and core functionality. Further, advances in date and date-time functionality has enabled management of time

series data within the "tibble" ("tidy" data frame) data structure. As the data science field grows, more innovative functionality will continue to be developed within the "tidy" ecosystem, much of which will be (and is already) useful to the field of financial analysis.

The two systems, "xts" and the "tidyverse", are very different on a fundamental and philosophical level. Both have advantages that are needed within the realm of financial analysis, but unfortunately the two systems do not work well together. The "xts" system is strictly numeric-based, while the "tidy" system is strictly data frame-based. Passing data between systems is difficult if not painful, and without communication between each the full potential of financial analysis within R is limited.

To solve this problem, the `tidyquant` package was developed as a way to integrate many of the "xts" based financial analysis packages within the "tidyverse". The central idea is to integrate the two systems enabling the user to gain the benefits of both systems. The `tidyquant` package works with "tidy" data frame input and output, and as a result fits seamlessly within the "tidy" ecosystem allowing for analysis to follow the data science workflow discussed in detail in *"R for Data Science"* [REF]. 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. The primary benefit of this integration is the ability to implement the *"split-apply-combine"* framework to scale complex analysis.

In the next section, the *"split-apply-combine"* framework is discussed conceptually using non-financial data and then a demonstration of the `tidyquant` package is presented to illustrate some of the key benefits of the integration of "xts" based financial packages into the "tidy" ecosystem within the realm of financial analysis.

Split-Apply-Combine, The Concept

The *"split-apply-combine"* framework is described in *"The Split-Apply-Combine Strategy for Data Analysis"* [REF]. To summarize, the core concept is to split a data set into groups, apply functions to independent groups, and then recombine the results. The value in this approach is the framework enables scaling analysis from one group to many groups and comparing each group to each other.

A simple example using the `mtcars` data set illustrates the *"split-apply-combine"* framework. The `mtcars` data set includes the various attributes for 32 vehicles along with the fuel efficiency (MPG) of each vehicle. Start with a question:

> How does engine size affect fuel consumption?

Load the `tidyverse` and the `mtcars` data set in R.

```
library(tidyverse)
data("mtcars")
```

Next, view the data set. The `as_tibble` function is used to convert to the "tidy" data frame structure. The data set consists of 12 columns (features) and 32 rows (observations) of related to various automobiles. Fortunately, the frame of the problem statement narrows the analysis to two variables: "mpg", continuous numeric data describing the fuel efficiency of each vehicle, and "cyl", discrete numeric data indicating number of engine cylinders for each vehicle. The "cyl" data can be grouped on to keep like observations together.

```
mtcars <- mtcars %>%
    rownames_to_column(var = "model") %>%
    as_tibble()
mtcars
```

```
## # A tibble: 32 × 12
##                model   mpg   cyl  disp    hp  drat    wt  qsec    vs    am
##                <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1          Mazda RX4  21.0     6 160.0   110  3.90 2.620 16.46     0     1
```

```
## 2      Mazda RX4 Wag  21.0     6 160.0   110  3.90 2.875 17.02     0     1
## 3         Datsun 710  22.8     4 108.0    93  3.85 2.320 18.61     1     1
## 4     Hornet 4 Drive  21.4     6 258.0   110  3.08 3.215 19.44     1     0
## 5  Hornet Sportabout  18.7     8 360.0   175  3.15 3.440 17.02     0     0
## 6            Valiant  18.1     6 225.0   105  2.76 3.460 20.22     1     0
## 7          Duster 360  14.3     8 360.0   245  3.21 3.570 15.84     0     0
## 8           Merc 240D  24.4     4 146.7    62  3.69 3.190 20.00     1     0
## 9            Merc 230  22.8     4 140.8    95  3.92 3.150 22.90     1     0
## 10           Merc 280  19.2     6 167.6   123  3.92 3.440 18.30     1     0
## # ... with 22 more rows, and 2 more variables: gear <dbl>, carb <dbl>
```

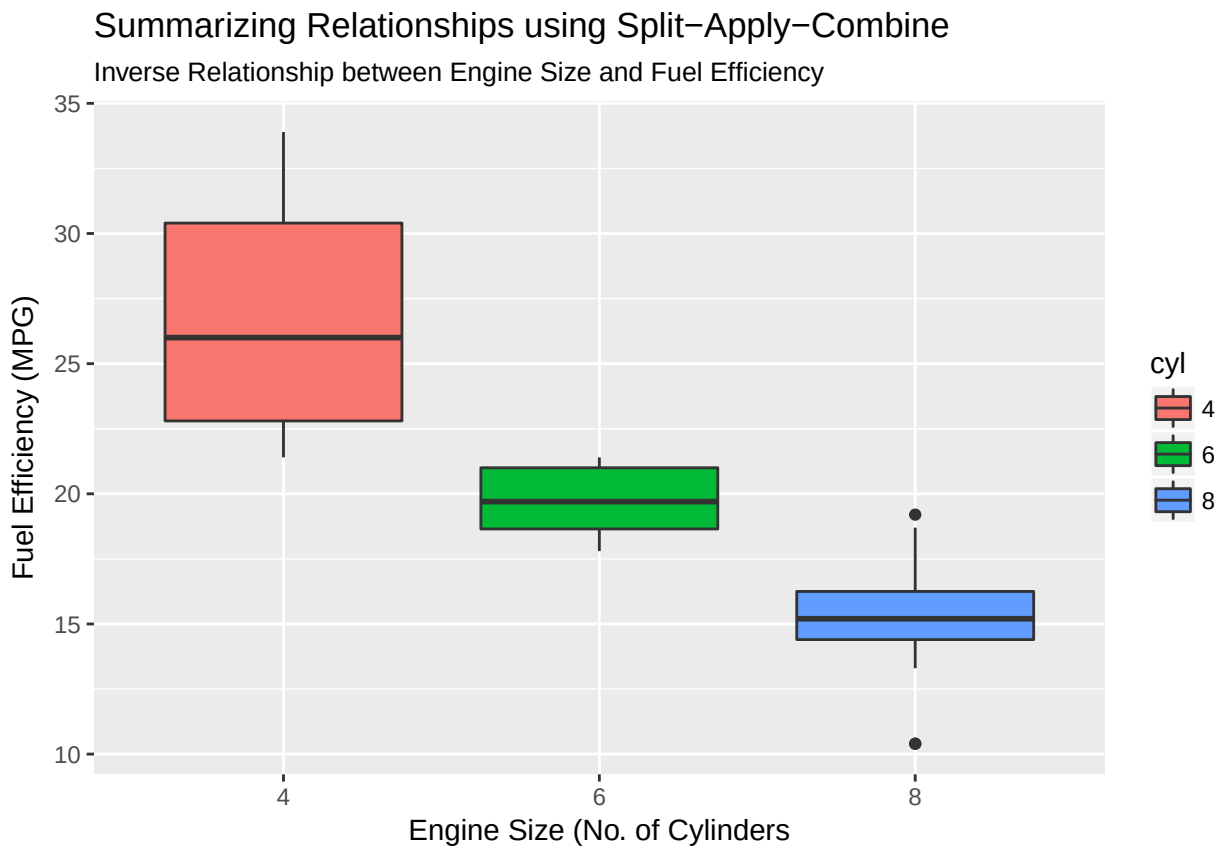
Many solutions exist to compare fuel consumption and engine size. For simplicity, the mean and standard deviation are used to characterize the relationship between number of cylinders, "cyl", and miles per gallon, "mpg". Implementing *"split-apply-combine"* is as easy as grouping by a categorical variable and summarizing by the target measure. In this case, the categorical variable is the "cyl" variable.

```
mtcars %>%
    mutate(cyl = factor(cyl)) %>%
    group_by(cyl) %>%
    summarize(mpg.mean = mean(mpg),
              mpg.sd = sd(mpg))
```

```
## # A tibble: 3 × 3
##      cyl mpg.mean   mpg.sd
##   <fctr>    <dbl>    <dbl>
## 1      4 26.66364 4.509828
## 2      6 19.74286 1.453567
## 3      8 15.10000 2.560048
```

From the results, the vehicles, when grouped by "cyl", appear to have an inverse relationship between number of cylinders and miles per gallon. We can also visualize this relationship using `ggplot2`.

```
mtcars %>%
    mutate(cyl = factor(cyl)) %>%
    ggplot(aes(x = cyl, y = mpg, fill = cyl)) +
    geom_boxplot() +
    labs(title = "Summarizing Relationships using Split-Apply-Combine",
         subtitle = "Inverse Relationship between Engine Size and Fuel Efficiency",
         x = "Engine Size (No. of Cylinders",
         y = "Fuel Efficiency (MPG)")
```



The *"split-apply-combine"* framework is very useful to solve a wide range of complex problems. In the next section, examples are presented to illustrate the power of unlocking the framework along with the data science workflow tools within financial analysis applications.

Split-Apply-Combine, Applications in Finance

The R 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 advance: 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 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 withing the "market". [FOOTNOTE]

```
Using the mean and standard deviation of stock returns has roots in Brownian motion and is known as the
```

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 some financial 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 SP500 index is a good place to 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")
sp500
```

```
## # A tibble: 502 × 2
##    symbol                    company
##     <chr>                      <chr>
## 1     MMM                         3M
## 2     ABT        ABBOTT LABORATORIES
## 3    ABBV                  ABBVIE INC
## 4     ACN                   ACCENTURE
## 5    ATVI        ACTIVISION BLIZZARD
## 6     AYI              ACUITY BRANDS
## 7    ADBE              ADOBE SYSTEMS
## 8     AAP         ADVANCE AUTO PARTS
## 9     AET                       AETNA
## 10    AMG AFFILIATED MANAGERS GROUP
## # ... with 492 more rows
```

Next, we need the historical stock prices for each stock within the SP500 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 data frame 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 SP500 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.

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

```
## # A tibble: 1,205,845 × 9
##    symbol company       date  open  high   low close  volume adjusted
##     <chr>   <chr>     <date> <dbl> <dbl> <dbl> <dbl>   <dbl>    <dbl>
## 1     MMM      3M 2007-01-03 77.53 78.85 77.38 78.26 3781500 60.31064
```

```
## 2     MMM      3M 2007-01-04 78.40 78.41 77.45 77.95 2968400 60.07174
## 3     MMM      3M 2007-01-05 77.89 77.90 77.01 77.42 2765200 59.66330
## 4     MMM      3M 2007-01-08 77.42 78.04 76.97 77.59 2434500 59.79431
## 5     MMM      3M 2007-01-09 78.00 78.23 77.44 77.68 1896800 59.86367
## 6     MMM      3M 2007-01-10 77.31 77.96 77.04 77.85 1787500 59.99468
## 7     MMM      3M 2007-01-11 78.05 79.03 77.88 78.65 2372500 60.61120
## 8     MMM      3M 2007-01-12 78.41 79.50 78.22 79.36 2582200 61.15835
## 9     MMM      3M 2007-01-16 79.48 79.62 78.92 79.56 2526600 61.31248
## 10    MMM      3M 2007-01-17 79.33 79.51 78.75 78.91 2711300 60.81156
## # ... with 1,205,835 more rows
```

Next, the *"split-apply-combine"* framework is used to group the prices by stock symbol and to calculate the logarithmic daily returns. The transform is applied using `tq_transform`, which is used in situations where periodicity changes (or can change). The `quantmod` OHLC (open, high, low, close) notation is used to collect the adjusted prices (`ohlc_fun = Ad`) and send these prices to the `periodReturn` function (`transform_fun = periodReturn`). The additional arguments `period = "daily"` and `type = "log"` are passed to the transformation 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_transform(ohlc_fun = Ad, transform_fun = periodReturn,
                 period = "daily", type = "log", col_rename = "dlr")
sp500_returns
```

```
## Source: local data frame [1,205,845 x 3]
## Groups: symbol [502]
##
##    symbol       date          dlr
##     <chr>     <date>        <dbl>
## 1     MMM 2007-01-03  0.000000000
## 2     MMM 2007-01-04 -0.003969091
## 3     MMM 2007-01-05 -0.006822440
## 4     MMM 2007-01-08  0.002193382
## 5     MMM 2007-01-09  0.001159338
## 6     MMM 2007-01-10  0.002186048
## 7     MMM 2007-01-11  0.010223770
## 8     MMM 2007-01-12  0.008986823
## 9     MMM 2007-01-16  0.002516944
## 10    MMM 2007-01-17 -0.008203410
## # ... with 1,205,835 more rows
```

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 enables applying 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.

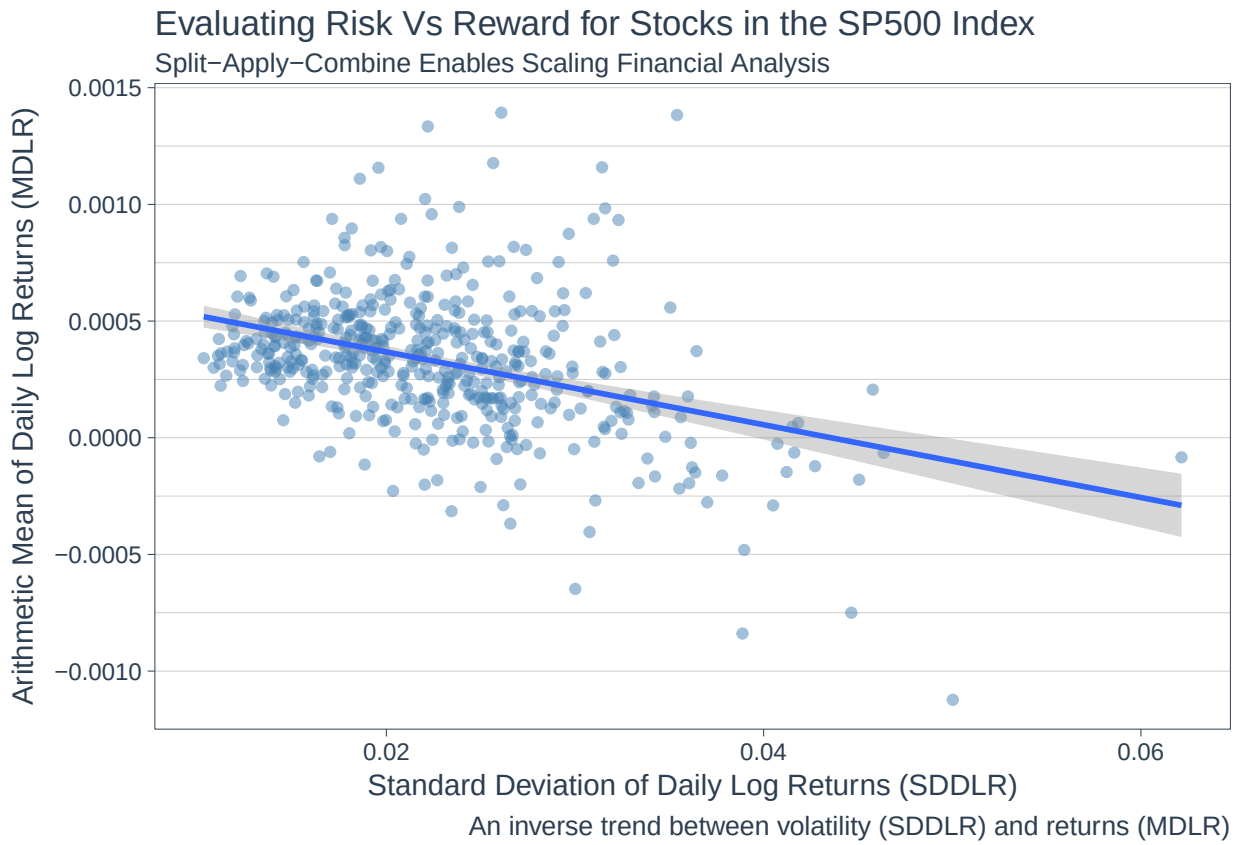
```
sp500_stats <- sp500_returns %>%
    tq_performance(Ra = dlr, performance_fun = table.Stats, ci = 0.95, digits = 6)
sp500_stats
```

```
## Source: local data frame [502 x 17]
## Groups: symbol [502]
##
##    symbol ArithmeticMean GeometricMean  Kurtosis `LCLMean(0.95)`  Maximum
##     <chr>          <dbl>         <dbl>     <dbl>           <dbl>    <dbl>
```

```
## 1     MMM       0.000431      0.000331  5.806040       -0.000120 0.094204
## 2     ABT       0.000302      0.000215  6.294066       -0.000211 0.091903
## 3    ABBV       0.000714      0.000565  3.868658       -0.000350 0.095982
## 4     ACN       0.000543      0.000404  8.532566       -0.000108 0.151577
## 5    ATVI       0.000604      0.000359 10.735599       -0.000263 0.209206
## 6     AYI       0.000701      0.000420  6.011008       -0.000225 0.161304
## 7    ADBE       0.000376      0.000147  8.992646       -0.000457 0.134165
## 8     AAP       0.000634      0.000430 14.825940       -0.000156 0.153215
## 9     AET       0.000453      0.000208  7.370411       -0.000410 0.163419
## 10    AMG       0.000126     -0.000289  8.848371       -0.000996 0.211618
## # ... with 492 more rows, and 11 more variables: Median <dbl>,
## #   Minimum <dbl>, NAs <dbl>, Observations <dbl>, Quartile1 <dbl>,
## #   Quartile3 <dbl>, SEMean <dbl>, Skewness <dbl>, Stdev <dbl>,
## #   `UCLMean(0.95)` <dbl>, Variance <dbl>
```

Finally, we have the data needed to visualize risk versus reward. 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 or 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 Index",
         subtitle = "Split-Apply-Combine Enables Scaling Financial Analysis",
         caption = "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_stats %>%
    filter(Observations >= 252 * 5,
           Stdev <= 0.02,
           ArithmeticMean >= 0.0005) %>%
    arrange(desc(ArithmeticMean))
```

```
## Source: local data frame [52 x 17]
## Groups: symbol [52]
##
##    symbol ArithmeticMean GeometricMean  Kurtosis `LCLMean(0.95)`  Maximum
##     <chr>          <dbl>         <dbl>     <dbl>           <dbl>    <dbl>
## 1     TDG       0.001157      0.000966  7.969939        0.000392 0.139356
## 2    FBHS       0.001110      0.000937  2.586426        0.000110 0.087617
## 3    DLPH       0.000938      0.000791  4.216286        0.000002 0.065790
## 4    ROST       0.000897      0.000732  5.158961        0.000187 0.124237
## 5    ORLY       0.000857      0.000699 10.209425        0.000162 0.168772
## 6     AGN       0.000825      0.000667  6.061143        0.000130 0.115273
## 7      EW       0.000817      0.000618 28.019706        0.000046 0.155802
## 8    DLTR       0.000803      0.000619  7.308803        0.000053 0.120219
## 9     AZO       0.000753      0.000632 12.719800        0.000143 0.177334
## 10    TJX       0.000708      0.000563  4.572350        0.000043 0.102042
## # ... with 42 more rows, and 11 more variables: Median <dbl>,
```

```
## #   Minimum <dbl>, NAs <dbl>, Observations <dbl>, Quartile1 <dbl>,
## #   Quartile3 <dbl>, SEMean <dbl>, Skewness <dbl>, Stdev <dbl>,
## #   `UCLMean(0.95)` <dbl>, Variance <dbl>
```

Example 2: Evaluating Performance of Multiple Portfolio Blends

Portfolio aggregation is a useful technique to reduce risk while maintaining acceptable returns. In this example, the goal is to evaluate a few blended portfolios of "FANG" stocks ("FB", "AMZN", "NFLX", and "GOOG") [FOOTNOTE]. 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 could be applied to an entire index of assets and hundreds of weighted blends.

```
# FOOTNOTE
The acronymn, "FANG", was popularized by Jim Cramer of the popular CNBC show, _Mad Money_. http://www.i
```

Throughout the past half decade, the FANG stocks have experienced a unique combination of high returns and high volatility, making the FANG stocks 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 over a since 2013 [FOOTNOTE]:

- 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

```
# FOOTNOTE
FOOTNOTE: 2013 was the first full year of trading data for FB. Portfolio aggregation prior to 2013 canno
```

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

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

```
## # A tibble: 8,717 × 8
##    symbol       date  open  high   low close    volume adjusted
##     <chr>     <date> <dbl> <dbl> <dbl> <dbl>     <dbl>    <dbl>
## 1      FB 2012-05-18 42.05 45.00 38.00 38.23 573576400    38.23
## 2      FB 2012-05-21 36.53 36.66 33.00 34.03 168192700    34.03
## 3      FB 2012-05-22 32.61 33.59 30.94 31.00 101786600    31.00
## 4      FB 2012-05-23 31.37 32.50 31.36 32.00  73600000    32.00
## 5      FB 2012-05-24 32.95 33.21 31.77 33.03  50237200    33.03
## 6      FB 2012-05-25 32.90 32.95 31.11 31.91  37149800    31.91
## 7      FB 2012-05-29 31.48 31.69 28.65 28.84  78063400    28.84
## 8      FB 2012-05-30 28.70 29.55 27.86 28.19  57267900    28.19
## 9      FB 2012-05-31 28.55 29.67 26.83 29.60 111639200    29.60
## 10     FB 2012-06-01 28.89 29.15 27.39 27.72  41855500    27.72
## # ... with 8,707 more rows
```

Next, transform to monthly returns using the *"split-apply-combine"* framework. Use `group_by` to group on the "symbol" column, and `tq_transform` to transform the adjusted prices to monthly arithmetic returns. Note that "FB" was actively traded for a full year beginning in 2013, so it makes sense to compare the investments since then.

```
FANG_returns <- FANG %>%
    filter(date >= ymd("2013-01-01"),
           date <  ymd("2017-01-01")) %>%
    group_by(symbol) %>%
    tq_transform(ohlc_fun = Ad,
                 transform_fun = periodReturn,
                 period = "monthly",
                 col_rename = "returns")
FANG_returns
```

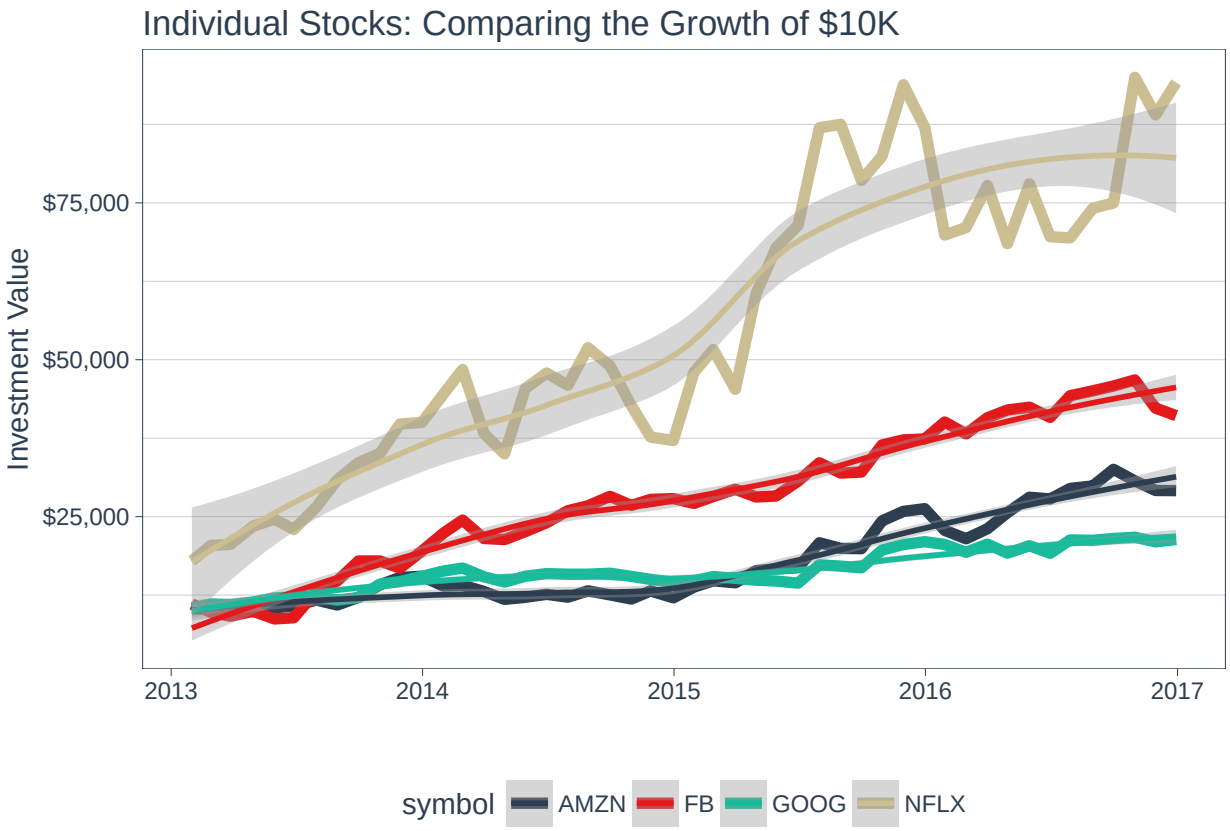
```
## Source: local data frame [192 x 3]
## Groups: symbol [4]
##
##    symbol       date       returns
##     <chr>     <date>         <dbl>
## 1      FB 2013-01-31  0.1064285714
## 2      FB 2013-02-28 -0.1204002582
## 3      FB 2013-03-28 -0.0612844037
## 4      FB 2013-04-30  0.0856137608
## 5      FB 2013-05-31 -0.1231544833
## 6      FB 2013-06-28  0.0217658727
## 7      FB 2013-07-31  0.4790996977
## 8      FB 2013-08-30  0.1220109272
## 9      FB 2013-09-30  0.2165172871
## 10     FB 2013-10-31 -0.0003981883
## # ... with 182 more rows
```

Individual Asset Performance

Before the portfolios are generated and evaluated, it makes sense 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 [INSERT REF BACON]. 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 risk 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
```

```
## Source: local data frame [4 x 2]
## Groups: symbol [4]
##
##   symbol VaR.monthly
##    <chr>       <dbl>
## 1     FB -0.05643582
## 2   AMZN -0.09687902
## 3   GOOG -0.05662687
## 4   NFLX -0.09775804
```

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
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: Make a portfolio by repeating the stock returns

Use `tq_repeat_df` to repeat `n = 4` times. This function grows the data frame row-wise, adding an index column named "portfolio" and grouping by "portfolio". We now have four portfolio groups that will be evaluated.

```
FANG_returns_mult <- FANG_returns %>%
    tq_repeat_df(n = 4)
FANG_returns_mult
```

```
## Source: local data frame [768 x 4]
## Groups: portfolio [4]
##
##    portfolio symbol       date       returns
##        <int>  <chr>     <date>         <dbl>
## 1          1     FB 2013-01-31  0.1064285714
## 2          1     FB 2013-02-28 -0.1204002582
## 3          1     FB 2013-03-28 -0.0612844037
## 4          1     FB 2013-04-30  0.0856137608
## 5          1     FB 2013-05-31 -0.1231544833
## 6          1     FB 2013-06-28  0.0217658727
## 7          1     FB 2013-07-31  0.4790996977
## 8          1     FB 2013-08-30  0.1220109272
## 9          1     FB 2013-09-30  0.2165172871
## 10         1     FB 2013-10-31 -0.0003981883
## # ... with 758 more rows
```

Step 2: Create a Weights Table

Construct a weights table using the portfolio blending parameters, which will be used to map to 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
```

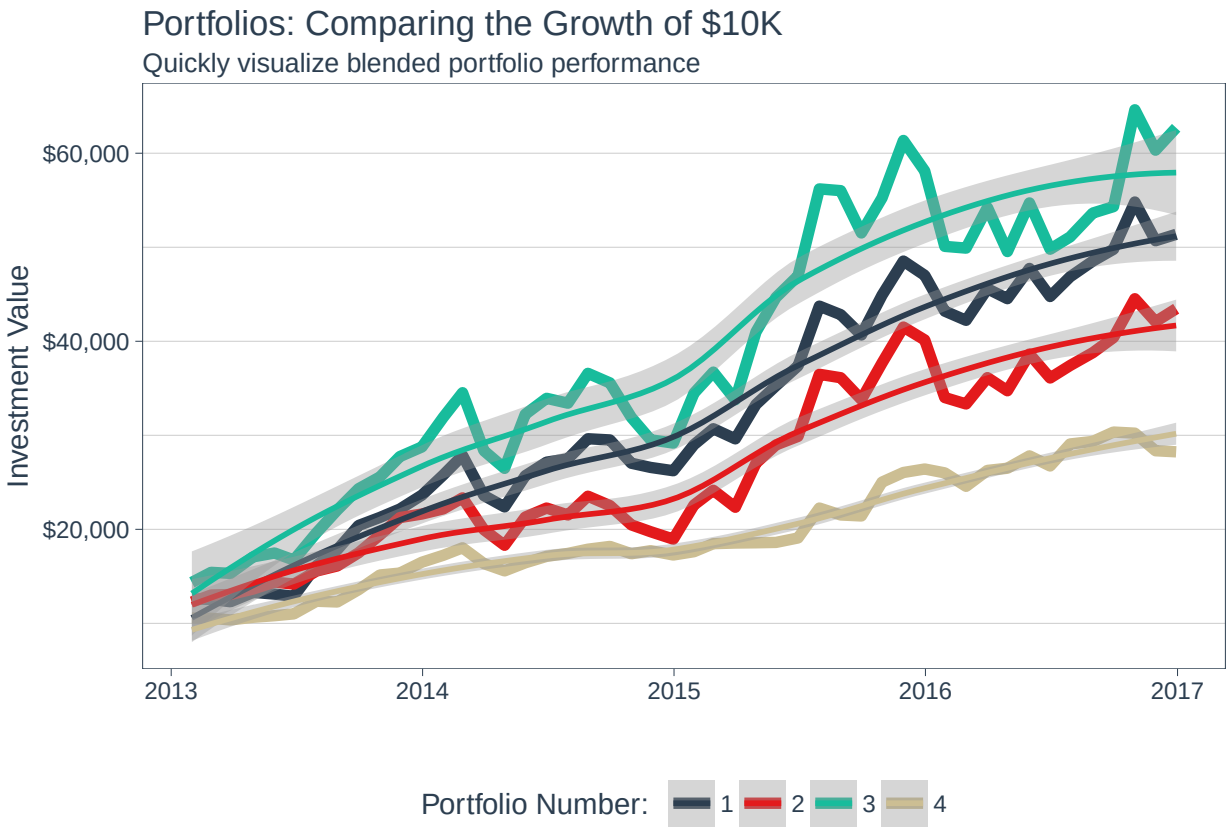
```
## Source: local data frame [16 x 3]
## Groups: portfolio [4]
##
##    portfolio stocks weights
##        <int>  <chr>   <dbl>
## 1          1     FB    0.50
## 2          1   AMZN    0.25
## 3          1   NFLX    0.25
## 4          1   GOOG    0.00
## 5          2     FB    0.00
## 6          2   AMZN    0.50
## 7          2   NFLX    0.25
## 8          2   GOOG    0.25
## 9          3     FB    0.25
## 10         3   AMZN    0.00
## 11         3   NFLX    0.50
## 12         3   GOOG    0.25
## 13         4     FB    0.25
## 14         4   AMZN    0.25
## 15         4   NFLX    0.00
## 16         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` has additional arguments to create a wealth index, which is the compounded returns. 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)`.

```
# C: 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
```

```
## Source: local data frame [4 x 2]
## Groups: portfolio [4]
##
##   portfolio VaR.monthly
##       <int>       <dbl>
## 1         1 -0.08533856
## 2         2 -0.10551077
## 3         3 -0.10774177
## 4         4 -0.05269662
```

Tables 1 and 2 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.

Table 1: FANG Results

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

Table 2: Portfolio Results

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

Future Possibilities

The *"split-apply-combine"* framework presents an opportunity for progress in the realm of financial analysis due to the scale at which analysis can now be performed. Portfolio analysis is especially interesting because as the number of components and the number of portfolio blend variations increase, the number of portfolio combinations become infinite. Yet, with the scaling capabilities within the "tidy" ecosystem, 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 technology, and optimization algorithms, the tools are available to scale analyses to terabytes worth of financial data in real time. The work is not complete, but, as innovation continues in the realm of data science, the realm of financial analysis will benefit in scale and speed.