Intro To Tidyquant

Matthew McDonald

Working with Stock Market Data

Loading Packages

```
library(tidyverse)
library(tidyquant)
library(scales)
```

There are two packages you haven't seen:

- **tidyquant**: a package that helps facilitate analysis of financial data in the tidyverse
- **scales**: provides useful scale functions for visualizations

Accessing Stock Data

```r
prices <- tq_get("AAPL",
  get = "stock.prices",
  from = "2000-01-01",
  to = "2022-12-31"
)
prices
```

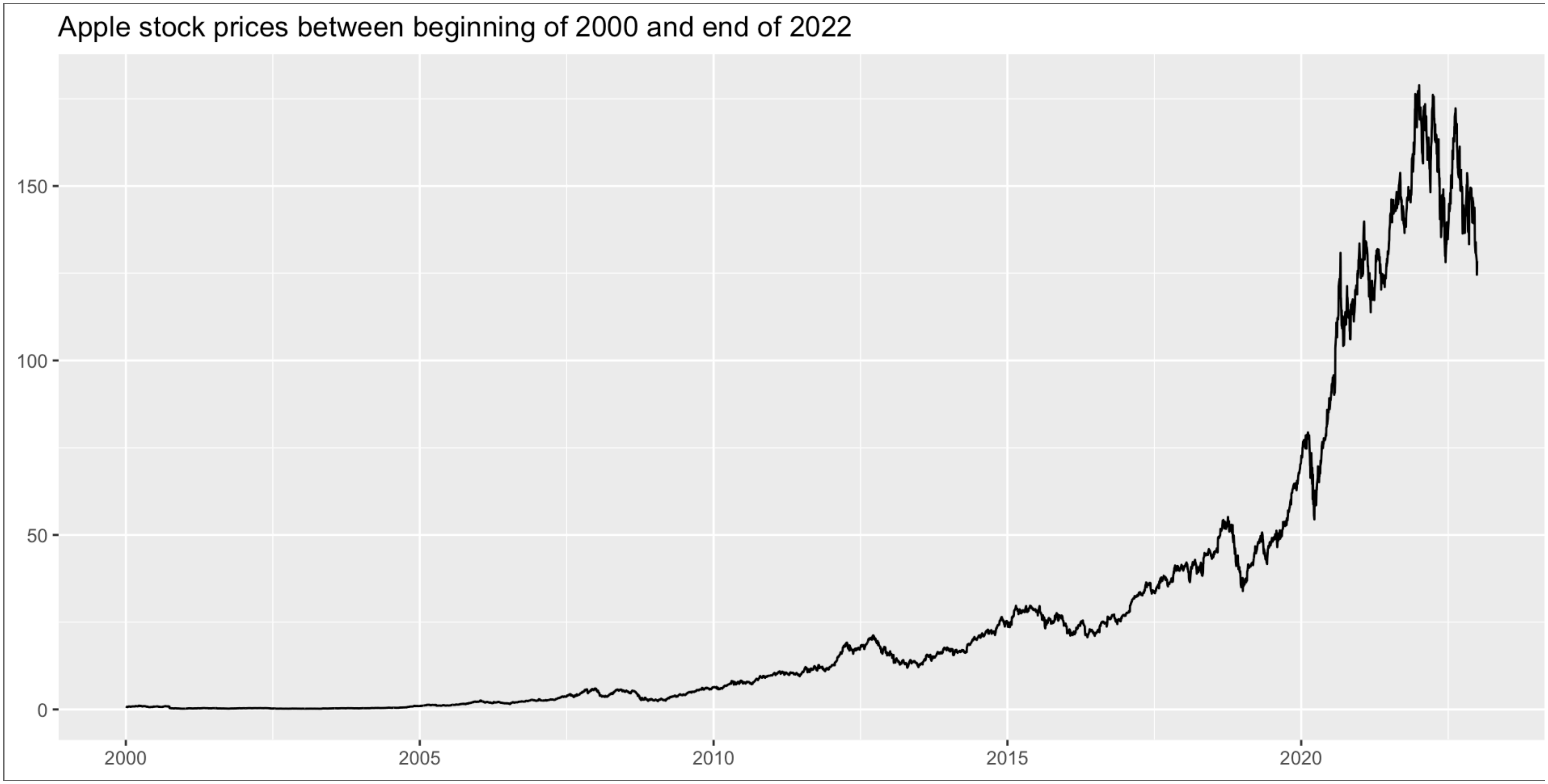
```
# A tibble: 5,787 × 8
   symbol date        open  high   low close     volume adjusted
   <chr>  <date>     <dbl> <dbl> <dbl> <dbl>      <dbl>    <dbl>
 1 AAPL   2000-01-03 0.936 1.00  0.908 0.999  535796800    0.842
 2 AAPL   2000-01-04 0.967 0.988 0.903 0.915  512377600    0.771
 3 AAPL   2000-01-05 0.926 0.987 0.920 0.929  778321600    0.782
 4 AAPL   2000-01-06 0.948 0.955 0.848 0.848  767972800    0.715
 5 AAPL   2000-01-07 0.862 0.902 0.853 0.888  460734400    0.749
 6 AAPL   2000-01-10 0.911 0.913 0.846 0.873  505064000    0.735
 7 AAPL   2000-01-11 0.857 0.887 0.808 0.828  441548800    0.698
 8 AAPL   2000-01-12 0.848 0.853 0.772 0.778  976068800    0.656
 9 AAPL   2000-01-13 0.844 0.882 0.826 0.864 1032684800    0.728
10 AAPL   2000-01-14 0.893 0.913 0.887 0.897  390376000    0.756
# ℹ 5,777 more rows
```

tq_get

- `tq_get` downloads stock market data from Yahoo!Finance if you do not specify another data source.
- The `adjusted` prices are corrected for anything that might affect the stock price after the market closes, e.g., stock splits and dividends.

Plotting with ggplot2

```r
prices |>
  ggplot(aes(x = date, y = adjusted)) +
  geom_line() +
  labs(
    x = NULL,
    y = NULL,
    title = "Apple stock prices between beginning of 2000 and end of 2022"
  )
```



Prices are in USD, adjusted for dividend payments and stock splits.

Calculating Returns

Instead of analyzing prices, we compute daily net returns defined as $r_t = p_t/p_{t-1} - 1$, where p_t is the adjusted day t price. In that context, the function `lag()` is helpful, which returns the previous value in a vector.

```
returns <- prices |>
  arrange(date) |>
  mutate(ret = adjusted / lag(adjusted) - 1) |>
  select(symbol, date, ret)
returns
```

```
# A tibble: 5,787 × 3
   symbol date           ret
   <chr>  <date>       <dbl>
 1 AAPL   2000-01-03 NA
 2 AAPL   2000-01-04 -0.0843
 3 AAPL   2000-01-05  0.0146
 4 AAPL   2000-01-06 -0.0865
 5 AAPL   2000-01-07  0.0474
 6 AAPL   2000-01-10 -0.0176
 7 AAPL   2000-01-11 -0.0512
 8 AAPL   2000-01-12 -0.0600
 9 AAPL   2000-01-13  0.110
10 AAPL   2000-01-14  0.0381
# ℹ 5,777 more rows
```

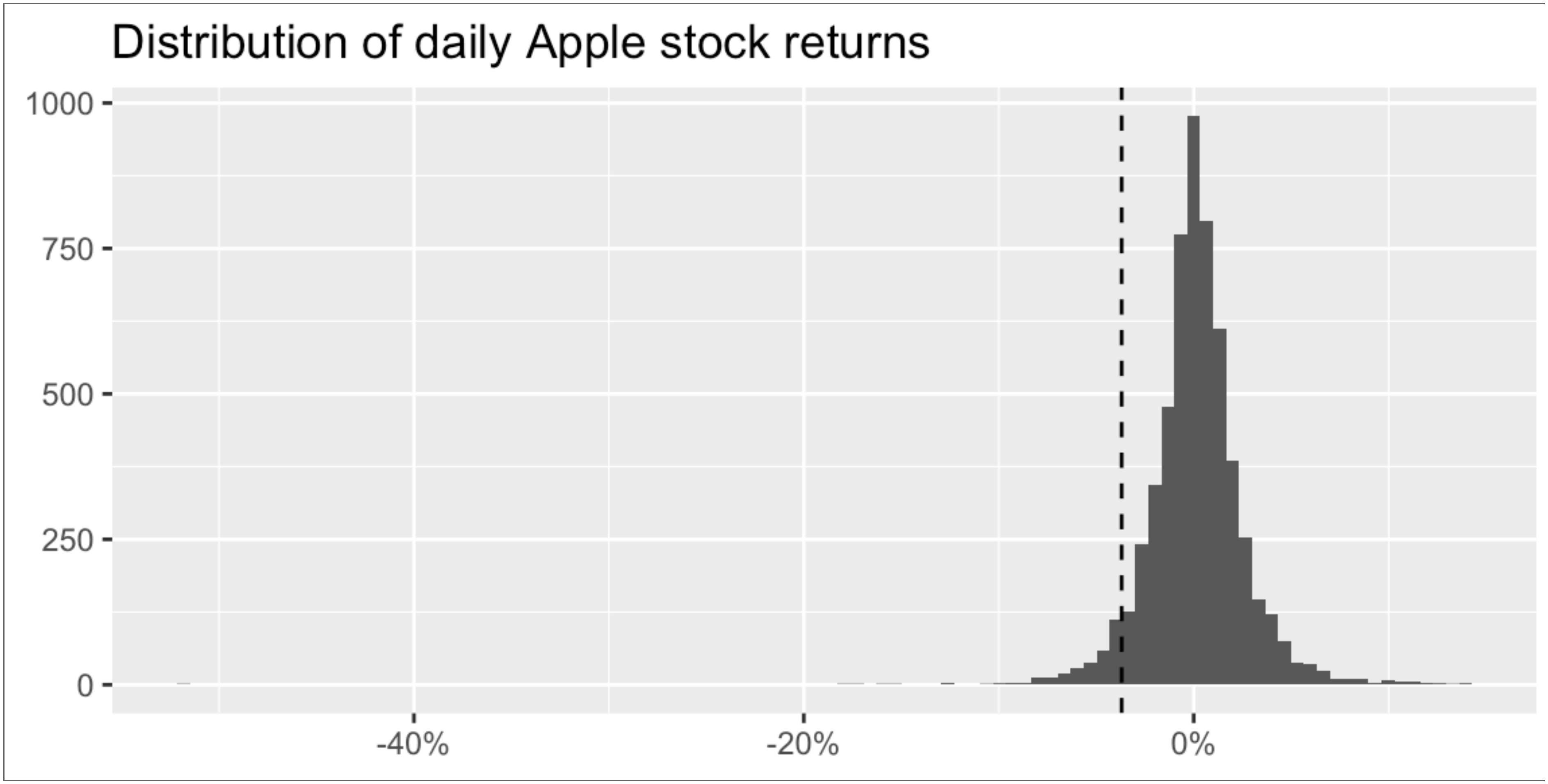
Removing NA Records

```r
returns <- returns |>
  drop_na(ret)
returns
```

```
# A tibble: 5,786 × 3
   symbol date            ret
   <chr>  <date>        <dbl>
 1 AAPL   2000-01-04 -0.0843
 2 AAPL   2000-01-05  0.0146
 3 AAPL   2000-01-06 -0.0865
 4 AAPL   2000-01-07  0.0474
 5 AAPL   2000-01-10 -0.0176
 6 AAPL   2000-01-11 -0.0512
 7 AAPL   2000-01-12 -0.0600
 8 AAPL   2000-01-13  0.110
 9 AAPL   2000-01-14  0.0381
10 AAPL   2000-01-18  0.0348
# ℹ 5,776 more rows
```

Visualizing Returns

```
quantile_05 <- quantile(returns |> pull(ret), probs = 0.05)
returns |>
  ggplot(aes(x = ret)) +
  geom_histogram(bins = 100) +
  geom_vline(aes(xintercept = quantile_05),
    linetype = "dashed"
  ) +
  labs(x = NULL, y = NULL,
    title = "Distribution of daily Apple stock returns"
  ) +
  scale_x_continuous(labels = percent)
```



Distribution of daily Apple stock returns
1000
750
500
250
0
-40%
-20%
0%

Summarizing Returns

```
returns |>
  summarize(across(
    ret,
    list(
      daily_mean = mean,
      daily_sd = sd,
      daily_min = min,
      daily_max = max
    )
  ))
```

```
# A tibble: 1 × 4
  ret_daily_mean ret_daily_sd ret_daily_min ret_daily_max
           <dbl>        <dbl>         <dbl>         <dbl>
1        0.00120       0.0251        -0.519         0.139
```

Summarizing Using group_by

```r
returns |>
  group_by(year = year(date)) |>
  summarize(across(
    ret,
    list(
      daily_mean = mean,
      daily_sd = sd,
      daily_min = min,
      daily_max = max
    ),
    .names = "{.fn}"
  )) |>
  print(n = Inf)
```

```
# A tibble: 23 × 5
    year daily_mean daily_sd daily_min daily_max
   <dbl>      <dbl>    <dbl>     <dbl>     <dbl>
 1  2000 -0.00346     0.0549   -0.519     0.137
 2  2001  0.00233     0.0393   -0.172     0.129
 3  2002 -0.00121     0.0305   -0.150     0.0846
 4  2003  0.00186     0.0234   -0.0814    0.113
 5  2004  0.00470     0.0255   -0.0558    0.132
 6  2005  0.00349     0.0245   -0.0921    0.0912
 7  2006  0.000949    0.0243   -0.0633    0.118
 8  2007  0.00366     0.0238   -0.0702    0.105
 9  2008 -0.00265     0.0367   -0.179     0.139
10  2009  0.00382     0.0214   -0.0502    0.0676
11  2010  0.00183     0.0169   -0.0496    0.0769
12  2011  0.00104     0.0165   -0.0559    0.0589
13  2012  0.00130     0.0186   -0.0644    0.0887
14  2013  0.000472    0.0180   -0.124     0.0514
15  2014  0.00145     0.0136   -0.0799    0.0820
16  2015  0.0000199   0.0168   -0.0612    0.0574
17  2016  0.000575    0.0147   -0.0657    0.0650
18  2017  0.00164     0.0111   -0.0388    0.0610
19  2018 -0.0000573   0.0181   -0.0663    0.0704
20  2019  0.00266     0.0165   -0.0996    0.0683
```

The across function

The across function allows you to apply a function (or functions) across multiple columns. It can also be used in the function `mutate`.
In case you wonder: the additional argument `.names = "{.fn}"` in `across()` determines how to name the output columns. The specification is rather flexible and allows almost arbitrary column names, which can be useful for reporting. The `print()` function simply controls the output options for the R console.

Scaling Up the Analysis

Incorporating more tickers

```r
symbols <- tq_index("DOW") |>
  filter(company != "US DOLLAR")
symbols
```

```
# A tibble: 30 × 8
   symbol company        identifier sedol weight sector shares_held
local_currency
   <chr>  <chr>          <chr>      <chr>  <dbl> <chr>        <dbl>
<chr>
 1 GS     GOLDMAN SAC… 38141G104  2407… 0.0899 -          5415976 USD
 2 UNH    UNITEDHEALT… 91324P102  2917… 0.0737 -          5415976 USD
 3 HD     HOME DEPOT … 437076102  2434… 0.0572 -          5415976 USD
 4 MSFT   MICROSOFT C… 594918104  2588… 0.0569 -          5415976 USD
 5 CAT    CATERPILLAR… 149123101  2180… 0.0502 -          5415976 USD
 6 SHW    SHERWIN WIL… 824348106  2804… 0.0496 -          5415976 USD
 7 V      VISA INC CL… 92826C839  B2PZ… 0.0484 -          5415976 USD
 8 CRM    SALESFORCE … 79466L302  2310… 0.0452 -          5415976 USD
 9 AXP    AMERICAN EX… 025816109  2026… 0.0429 -          5415976 USD
10 MCD    MCDONALD S … 580135101  2550… 0.0426 -          5415976 USD
# ℹ 20 more rows
```

Using tq_get for the Dow

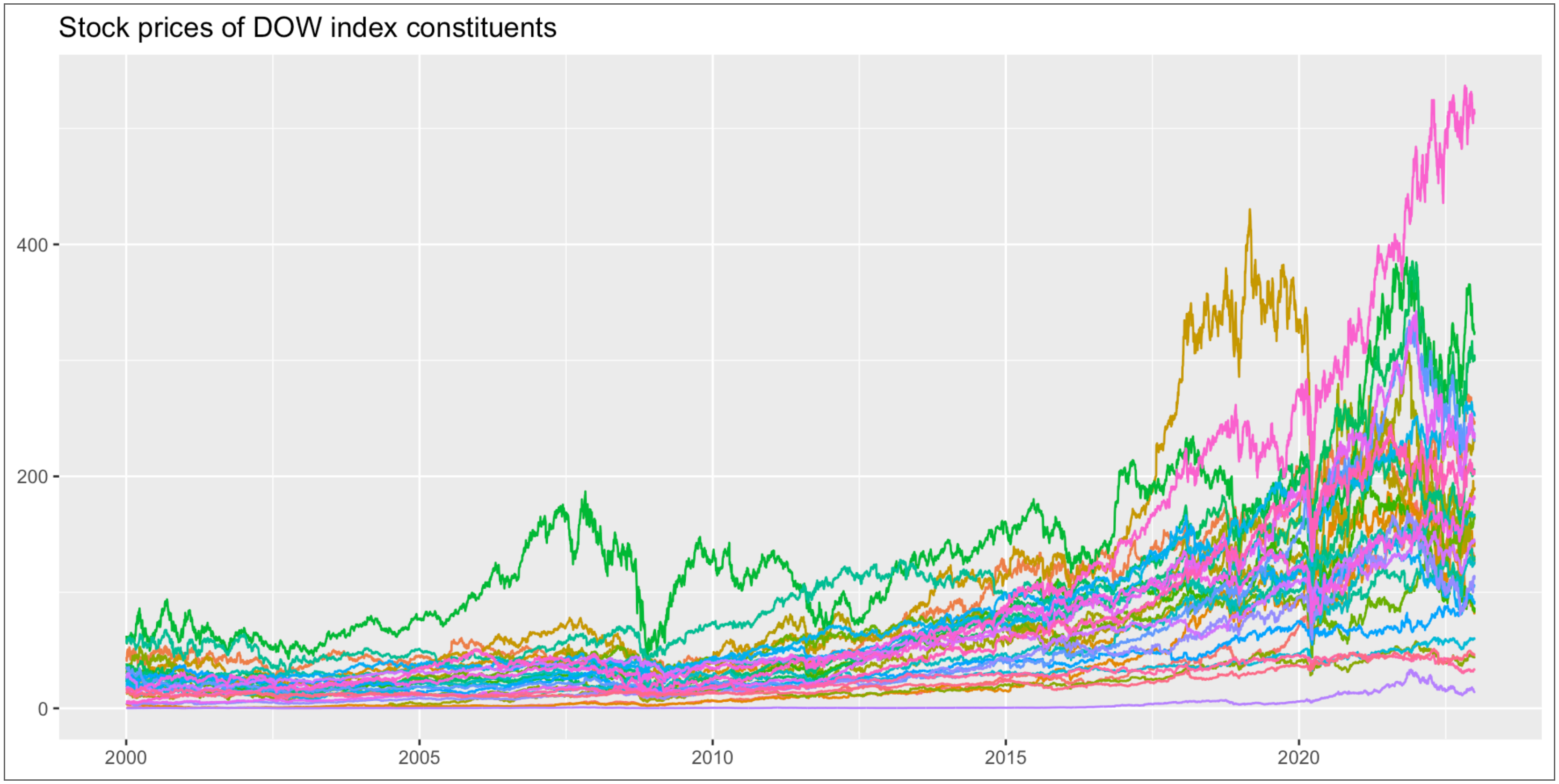
```r
index_prices <- tq_get(symbols,
  get = "stock.prices",
  from = "2000-01-01",
  to = "2022-12-31"
)
```

The resulting tibble contains 170425 daily observations for 30 different corporations.

Plotting The Constituent Prices

```r
index_prices |>
  ggplot(aes(
    x = date,
    y = adjusted,
    color = symbol
  )) +
  geom_line() +
  labs(
    x = NULL,
    y = NULL,
    color = NULL,
    title = "Stock prices of DOW index constituents"
  ) +
  theme(legend.position = "none")
```

Plotting The Constituent Prices



Calculating Summaries Stats For the Constituents

```
all_returns <- index_prices |>
  group_by(symbol) |>
  mutate(ret = adjusted / lag(adjusted) - 1) |>
  select(symbol, date, ret) |>
  drop_na(ret)

all_returns |>
  group_by(symbol) |>
  summarize(across(
    ret,
    list(
      daily_mean = mean,
      daily_sd = sd,
      daily_min = min,
      daily_max = max
    ),
    .names = "{.fn}"
  )) |>
  print(n = Inf)
```

Calculating Summaries Stats For the Constituents

```
# A tibble: 30 × 5
   symbol daily_mean daily_sd daily_min daily_max
   <chr>       <dbl>    <dbl>     <dbl>     <dbl>
 1 AAPL     0.00120    0.0251    -0.519     0.139
 2 AMGN     0.000489   0.0197    -0.134     0.151
 3 AMZN     0.00101    0.0319    -0.248     0.345
 4 AXP      0.000518   0.0229    -0.176     0.219
 5 BA       0.000595   0.0224    -0.238     0.243
 6 CAT      0.000709   0.0204    -0.145     0.147
 7 CRM      0.00110    0.0270    -0.271     0.260
 8 CSCO     0.000317   0.0237    -0.162     0.244
 9 CVX      0.000553   0.0176    -0.221     0.227
10 DIS      0.000418   0.0195    -0.184     0.160
11 GS       0.000550   0.0231    -0.190     0.265
12 HD       0.000543   0.0194    -0.287     0.141
13 HON      0.000515   0.0194    -0.174     0.282
14 IBM      0.000273   0.0165    -0.155     0.120
15 JNJ      0.000408   0.0122    -0.158     0.122
16 JPM      0.000582   0.0242    -0.207     0.251
17 KO       0.000338   0.0132    -0.101     0.139
18 MCD      0.000533   0.0147    -0.159     0.181
19 MMM      0.000378   0.0150    -0.129     0.126
20 MRK      0.000383   0.0168    -0.268     0.130
21 MSFT     0.000513   0.0194    -0.156     0.196
```

Other Indices

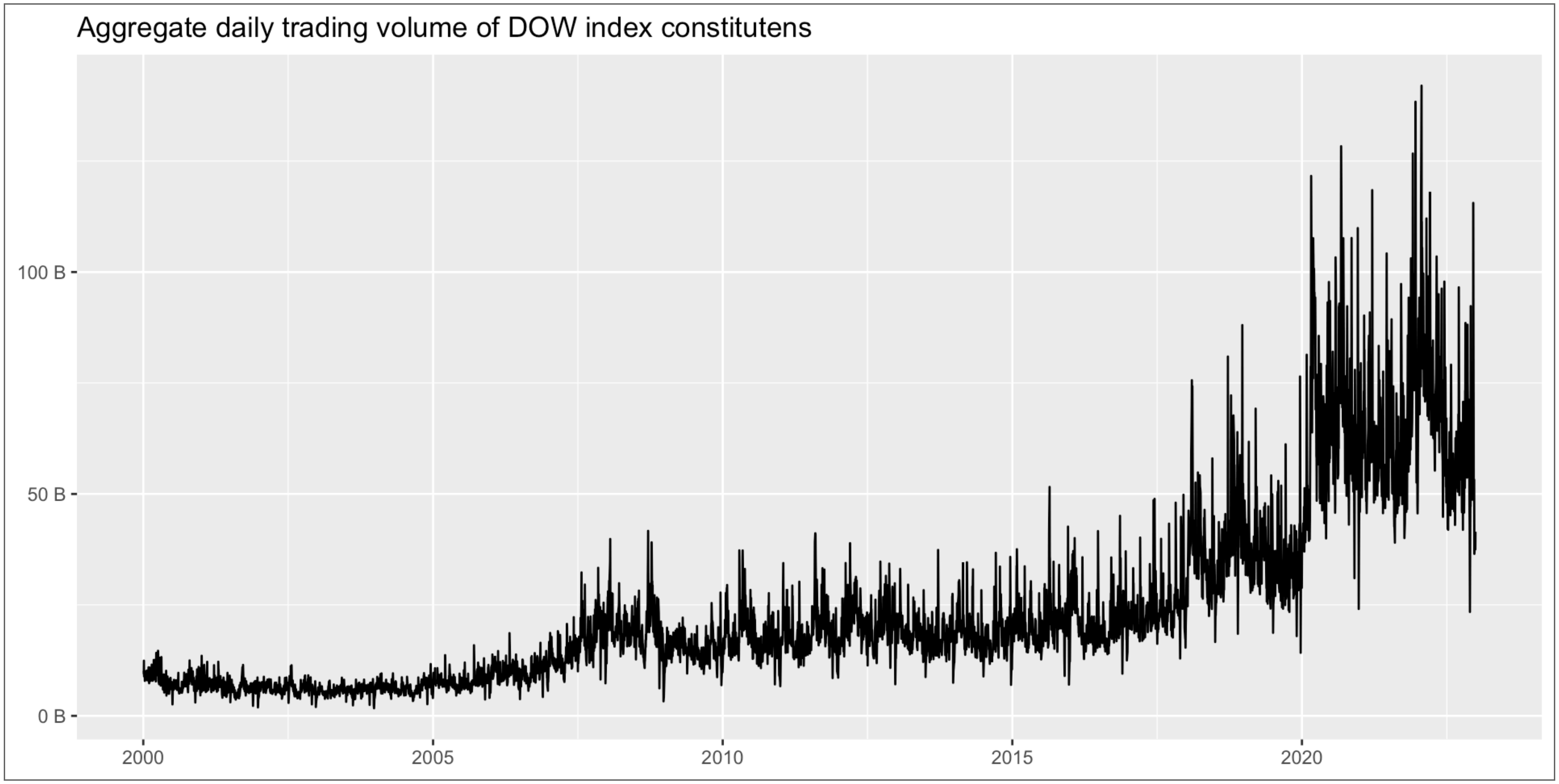
Note that you are now also equipped with all tools to download price data for *each* symbol listed in the S&P 500 index with the same number of lines of code. Just use `symbol <- tq_index("SP500")`, which provides you with a tibble that contains each symbol that is (currently) part of the S&P 500. However, don't try this if you are not prepared to wait for a couple of minutes because this is quite some data to download!

Other Forms of Data Aggregation

```
trading_volume <- index_prices |>
  group_by(date) |>
  summarize(trading_volume = sum(volume * adjusted))

trading_volume |>
  ggplot(aes(x = date, y = trading_volume)) +
  geom_line() +
  labs(
    x = NULL, y = NULL,
    title = "Aggregate daily trading volume of DOW index constituten
  ) +
    scale_y_continuous(labels = unit_format(unit = "B", scale = 1e-9
```

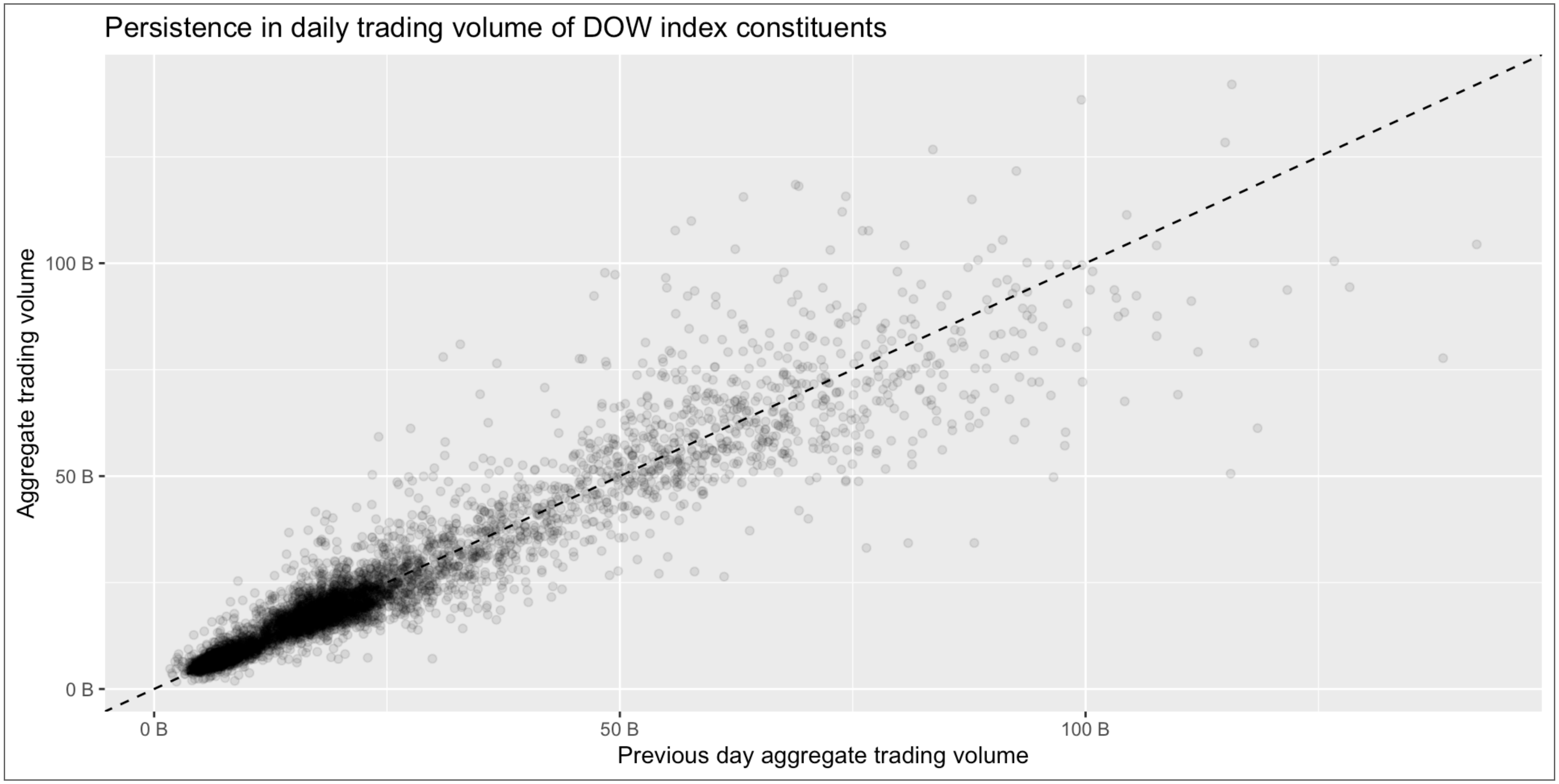
Other Forms of Data Aggregation



Persistence of high-volume trading days

```
trading_volume |>
  ggplot(aes(x = lag(trading_volume), y = trading_volume)) +
  geom_point(alpha=0.1) +
  geom_abline(aes(intercept = 0, slope = 1),
    linetype = "dashed"
  ) +
  labs(
    x = "Previous day aggregate trading volume",
    y = "Aggregate trading volume",
    title = "Persistence in daily trading volume of DOW index consti
  ) +
  scale_x_continuous(labels = unit_format(unit = "B", scale = 1e-9))
  scale_y_continuous(labels = unit_format(unit = "B", scale = 1e-9))
```

Persistence of high-volume trading days



Portfolio Choice Problems

Optimal Portfolio

The standard framework for optimal portfolio selection considers investors that prefer higher future returns but dislike future return volatility (defined as the square root of the return variance)

Effificient Frontier

the set of portfolios which satisfies the condition that no other portfolio exists with a higher expected return but with the same volatility (the square root of the variance, i.e., the risk)

Calculating Monthly Returns

```r
index_prices <- index_prices |>
  group_by(symbol) |>
  mutate(n = n()) |>
  ungroup() |>
  filter(n == max(n)) |>
  select(-n)

returns <- index_prices |>
  mutate(month = floor_date(date, "month")) |>
  group_by(symbol, month) |>
  summarize(price = last(adjusted), .groups = "drop_last") |>
  mutate(ret = price / lag(price) - 1) |>
  drop_na(ret) |>
  select(-price)

returns
```

Calculating Monthly Returns

```
# A tibble: 7,700 × 3
# Groups:   symbol [28]
   symbol month          ret
   <chr>  <date>       <dbl>
 1 AAPL   2000-02-01  0.105
 2 AAPL   2000-03-01  0.185
 3 AAPL   2000-04-01 -0.0865
 4 AAPL   2000-05-01 -0.323
 5 AAPL   2000-06-01  0.247
 6 AAPL   2000-07-01 -0.0298
 7 AAPL   2000-08-01  0.199
 8 AAPL   2000-09-01 -0.577
 9 AAPL   2000-10-01 -0.240
10 AAPL   2000-11-01 -0.157
# ℹ 7,690 more rows
```

Transform Data For Analysis

Next, we transform the returns from a tidy tibble into a $(T \times N)$ matrix with one column for each of the N symbols and one row for each of the T months

```
returns_matrix <- returns |>
  pivot_wider(
    names_from = symbol,
    values_from = ret
  ) |>
  select(-month)
```

Sample Average Return Vector

to compute the sample average return vector

$$\mu \hat{=} \frac{1}{T} \sum_{t=1}^{T} r_t$$

where r_t is the N vector of returns on date t

```
mu <- colMeans(returns_matrix)
```

Sample Covariance Matrix

$$\hat{\Sigma} = \frac{1}{T-1}\sum_{t=1}^{T}(r_t - \hat{\mu})(r_t - \hat{\mu})^{\top}.$$

```r
sigma <- cov(returns_matrix)
```

Minimum Variance Portfolio Weights

The minimum variance portfolio is the vector of portfolio weights that are the solution to

$$\omega_{\text{mvp}} = \arg\min \omega' \Sigma \omega \text{ s.t. } \sum_{i=1}^{N} \omega_i = 1.$$

The constraint that weights sum up to one simply implies that all funds are distributed across the available asset universe, i.e., there is no possibility to retain cash.

The solution to the above equation is $\omega_{\text{mvp}} = \frac{\Sigma^{-1}\iota}{\iota'\Sigma^{-1}\iota}$, where ι is a vector of ones and Σ^{-1} is the inverse of Σ.

Calculating MVP Weights

```
N <- ncol(returns_matrix)
iota <- rep(1, N)
sigma_inv <- solve(sigma)
mvp_weights <- sigma_inv %*% iota
mvp_weights <- mvp_weights / sum(mvp_weights)
```

The command `solve(A, b)` returns the solution of a system of equations $Ax = b$. If `b` is not provided, as in the example above, it defaults to the identity matrix such that `solve(sigma)` delivers Σ^{-1} (if a unique solution exists).

Expected Portfolio Return and Volatility

- expected portfolio return: $\omega'_{\text{mvp}}\mu$
- expected portfolio volatility: $\sqrt{\omega'_{\text{mvp}}\Sigma\omega_{\text{mvp}}}$

```r
tibble(
  average_ret = as.numeric(t(mvp_weights) %*% mu),
  volatility = as.numeric(sqrt(t(mvp_weights) %*% sigma %*% mvp_weig
)
```

```
# A tibble: 1 × 2
  average_ret volatility
        <dbl>      <dbl>
1     0.00848     0.0320
```

Finding MVP for any return

choose ω_{eff} as the solution to
$\omega_{\text{eff}}(\bar{\mu}) = \arg\min \omega' \Sigma \omega$ s.t. $\omega' \iota = 1$ and $\omega' \mu \geq \bar{\mu}$.

Solving for 3x return

The code below implements the analytic solution to this optimization problem for a benchmark return $\bar{\mu}$, which is set to 3 times the expected return of the minimum variance portfolio.

```
benchmark_multiple <- 3
mu_bar <- benchmark_multiple * t(mvp_weights) %*% mu
C <- as.numeric(t(iota) %*% sigma_inv %*% iota)
D <- as.numeric(t(iota) %*% sigma_inv %*% mu)
E <- as.numeric(t(mu) %*% sigma_inv %*% mu)
lambda_tilde <- as.numeric(2 * (mu_bar - D / C) / (E - D^2 / C))
efp_weights <- mvp_weights +
  lambda_tilde / 2 * (sigma_inv %*% mu - D * mvp_weights)
```

What’s going on there?

Define Target Return Multiple

```
benchmark_multiple <- 3
```

- We set the **benchmark multiple** to 3.
- This means the new portfolio should have an **expected return three times** that of the GMV portfolio.

Compute Target Expected Return

```
mu_bar <- benchmark_multiple * t(mvp_weights) %*% mu
```

- $\mu_{\text{GMV}} = w_{\text{GMV}} \cdot \mu$
- $\mu_{\text{bar}} = 3 \times \mu_{\text{GMV}}$
- This sets the **target expected return**.

Compute Constants for Efficient Frontier

```
C <- as.numeric(t(iota) %*% sigma_inv %*% iota)
D <- as.numeric(t(iota) %*% sigma_inv %*% mu)
E <- as.numeric(t(mu) %*% sigma_inv %*% mu)
```

- $C = 1^T \Sigma^{-1} 1$ (Normalization constant)
- $D = 1^T \Sigma^{-1} \mu$ (Return-weighted sum)
- $E = \mu^T \Sigma^{-1} \mu$ (Risk-adjusted return)

Understanding (C), (D), and (E)

- These constants are derived from **mean-variance portfolio optimization** and define the **efficient frontier**.
- $C = 1^T \Sigma^{-1} 1$
  - Measures total **risk-adjusted exposure** of an equal-weighted portfolio.
  - Ensures that portfolio weights sum to one.
- $D = 1^T \Sigma^{-1} \mu$
  - Represents the **risk-adjusted expected returns** of the portfolio.
  - Helps determine the **trade-off between risk and return**.

- $E = \mu^T \Sigma^{-1} \mu$
  - Measures the **risk-adjusted total expected return**.
  - Helps find the optimal portfolio maximizing return per unit of risk.
- These constants are key to computing **efficient frontier portfolios** and adjusting weights for **minimum variance solutions**.

Compute Lambda Scaling Factor

```r
lambda_tilde <- as.numeric(2 * (mu_bar - D / C) / (E - D^2 / C))
```

- Computes the **scaling factor** to adjust the GMV portfolio along the efficient frontier.
- λ_{tilde} adjusts the portfolio to reach the desired return while maintaining minimum variance.

Compute Final Portfolio Weights

```
efp_weights <- mvp_weights +
  lambda_tilde / 2 * (sigma_inv %*% mu - D * mvp_weights)
```

- **Adjusts the GMV portfolio weights** using the efficient frontier equation.
- Moves along the efficient frontier to achieve the **3× return target**.

using calculated efp_weights

```
tibble(
  average_ret = as.numeric(t(efp_weights) %*% mu),
  volatility = as.numeric(sqrt(t(efp_weights) %*% sigma %*% efp_weig
)
```

```
# A tibble: 1 × 2
  average_ret volatility
        <dbl>      <dbl>
1      0.0254     0.0546
```

The Efficient Frontier

Mutual Fund Seperation Theroem

The mutual fund separation theorem states that as soon as we have two efficient portfolios (such as the minimum variance portfolio ω_{mvp} and the efficient portfolio for a higher required level of expected returns $\omega_{\text{eff}}(\bar{\mu})$, we can characterize the entire efficient frontier by combining these two portfolios. That is, any linear combination of the two portfolio weights will again represent an efficient portfolio.

Calculating the Efficient Frontier

```r
length_year <- 12
a <- seq(from = -0.4, to = 1.9, by = 0.01)
res <- tibble(
  a = a,
  mu = NA,
  sd = NA
)
for (i in seq_along(a)) {
  w <- (1 - a[i]) * mvp_weights + (a[i]) * efp_weights
  res$mu[i] <- length_year * t(w) %*% mu
  res$sd[i] <- sqrt(length_year) * sqrt(t(w) %*% sigma %*% w)
}
```

Explaining the Code

The code above proceeds in two steps: First, we compute a vector of combination weights a and then we evaluate the resulting linear combination with $a \in \mathbb{R}$:

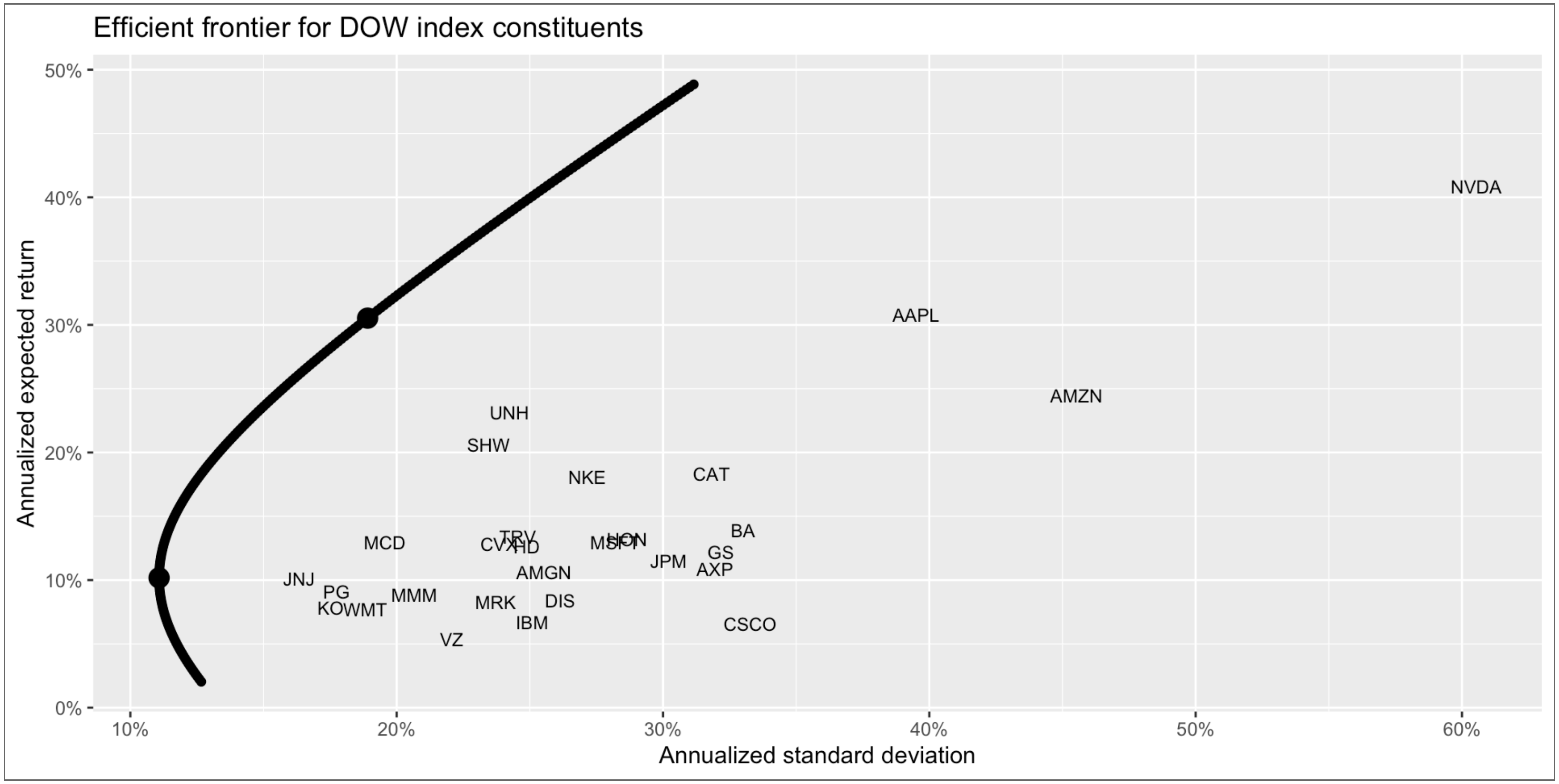
$$\omega^* = a\omega_{\text{eff}}(\bar{\mu}) + (1-a)\omega_{\text{mvp}} = \omega_{\text{mvp}} + \frac{\lambda^*}{2}\left(\Sigma^{-1}\mu - \frac{D}{C}\Sigma^{-1}\iota\right)$$

with $\lambda^* = 2\frac{a\bar{\mu} + (1-a)\tilde{\mu} - D/C}{E - D^2/C}$ where $C = \iota'\Sigma^{-1}\iota$, $D = \iota'\Sigma^{-1}\mu$, and $E = \mu'\Sigma^{-1}\mu$.

Visualizing the Efficient Frontier

```
res |>
  ggplot(aes(x = sd, y = mu)) +
  geom_point() +
  geom_point(
    data = res |> filter(a %in% c(0, 1)),
    size = 4
  ) +
  geom_text(
    data = tibble(
      ticker = colnames(returns_matrix),
      mu = length_year * mu,
      sd = sqrt(length_year) * sqrt(diag(sigma))
    ),
    aes(y = mu, x = sd, label=ticker), size = 3
  ) +
  labs(
    x = "Annualized standard deviation",
    y = "Annualized expected return",
    title = "Efficient frontier for DOW index constituents"
  ) +
  scale_x_continuous(labels = percent) +
  scale_y_continuous(labels = percent)
```

Visualizing the Efficient Frontier



Explaining the Efficient Frontier

The line in the prior lot indicates the efficient frontier: the set of portfolios a mean-variance efficient investor would choose from. Compare the performance relative to the individual assets (the dots) - it should become clear that diversifying yields massive performance gains (at least as long as we take the parameters Σ and μ as given).