Tidy Finance with R

www.tidy-finance.org

Stefan Voigt

December 2022

Tidy Finance with R- Welcome!

- The project: tidy-finance.org
- Is there a replication crisis in Finance?
- Sleeves up: data, the efficient frontier, machine learning for option pricing

What I am hoping for

- Feedback! Any form of ideas, questions, or critique (contact@tidy-finance.org)
- Collaboration! Tidy Finance is open-source, *please* contribute!

Who needs replicability?

- The academic community produces tons of research articles every year
- Ever tried to replicate a paper?
- Ever tempted to p-hack your results?
- Ever analyzed the effect that all the needy-greedy decisions you make along the way have on your results?

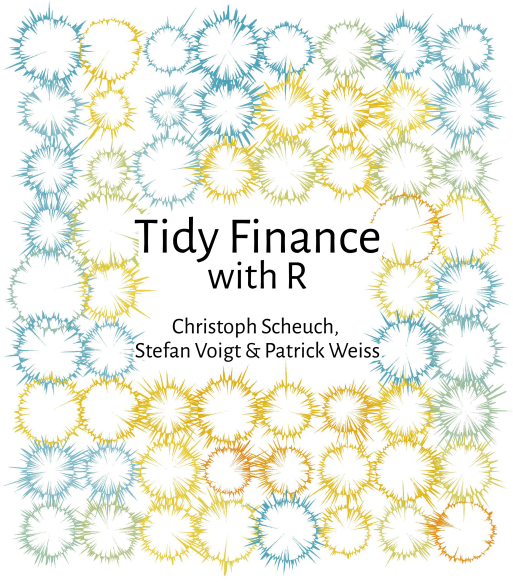
What can we do?

- At a **minimum**: Make sure your own work can be replicated (by future-you)
- **Little bigger**: Teach such principles to your students
- **Really big**: Eliminate (by sticking to replicating principles) the replication crisis in Finance and adjust for non-standard errors

Tidy Finance with R

Tidy Finance with R is ...

- ... an open-source book available at tidy-finance.org
- ... a (tiny) step towards **reproducible finance** by providing a fully transparent code base
- ... a resource for students, lecturers, and professionals using **R** for applications in finance
- ... a **tidy** approach to finance
- ... continuously maintained and expanded
- ... available as a print version in March 2023 (!)



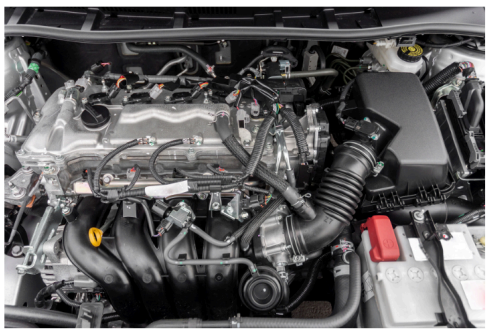
Who are we?



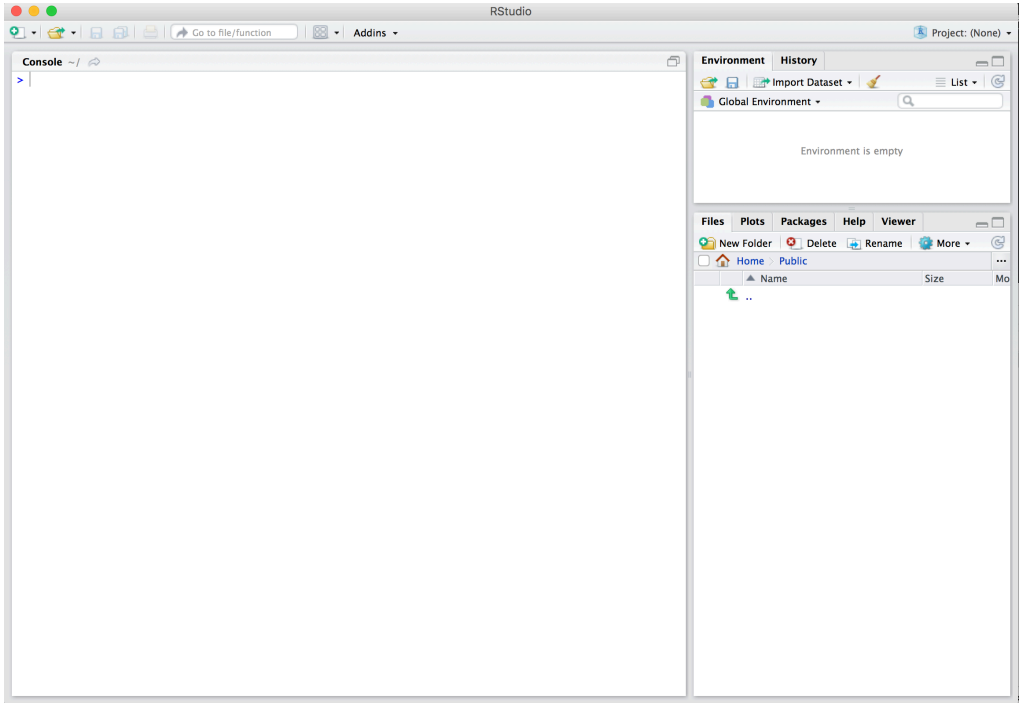
- Christoph Scheuch (Director of Product at the social trading platform wikifolio.com and external lecturer at the Vienna University of Economics and Business)
- Stefan Voigt (Assistant Professor of Finance at the Department of Economics at the University in Copenhagen and a research fellow at the Danish Finance Institute)
- Patrick Weiss (Vienna University of Economics and Business and Reykjavik University)

R? RStudio? What about Python?

- R is a language and environment for statistical computing and graphics
- R is free to download and use / R is open-source (Users can expand the functionality of R through add-ons called packages) / Data processing in R is very easy / Data visualization tools in R are pervasive / It is very easy to share your (reproducible) output from R
- While *R* runs computations, *RStudio* is an integrated development environment (IDE) that provides an interface by adding many convenient features and tools
- Python and R are nowadays the de-facto standard tool in finance (+ you can run Python scripts from within R and vice versa)

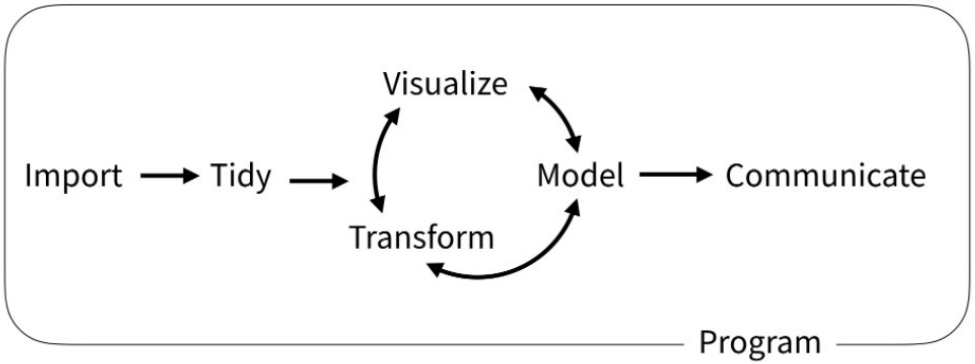
R: Engine	RStudio: Dashboard
	

Getting started with RStudio



- In case you did not set up R and RStudio yet: https://www.tidy-finance.org/#prerequisites
- If you have never used R or RStudio, Chapters 1-3 of this book are an excellent starting point: **Basic introduction to R**

Data science workflow with the tidyverse



- The tidyverse is “a framework for managing data that aims at doing the cleaning and preparing steps much easier”
- In my courses I work almost exclusively with tidyverse packages: ggplot, dplyr, readr, ...
- Hadley Wickhams and Garret Grolemunds famous book *R for Data Science* explains everything we need

Ready for a first case study? The main components of the tidyverse

- Start by getting the working directory right

```
setwd("path_to_your_course_folder")
```

- load the package `library(tidyverse)`
- Import data: `read_csv()`, `read_txt()`, .., or download with `tidyquant`

```
library(tidyverse)
library(tidyquant)
prices <- tq_get("AAPL", get = "stock.prices", from = "2000-01-01", to = "2021-12-31")
prices
```

```
## # A tibble: 5,535 x 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    0.852
## 2 AAPL   2000-01-04 0.967 0.988 0.903 0.915 512377600    0.780
## 3 AAPL   2000-01-05 0.926 0.987 0.920 0.929 778321600    0.792
## 4 AAPL   2000-01-06 0.948 0.955 0.848 0.848 767972800    0.723
## # ... with 5,531 more rows
```

Syntax with the pipe |>

- Verb(Subject, Complement) is replaced by `Subject |> Verb(Complement)`
- No need to name unimportant variables
- Clear readability

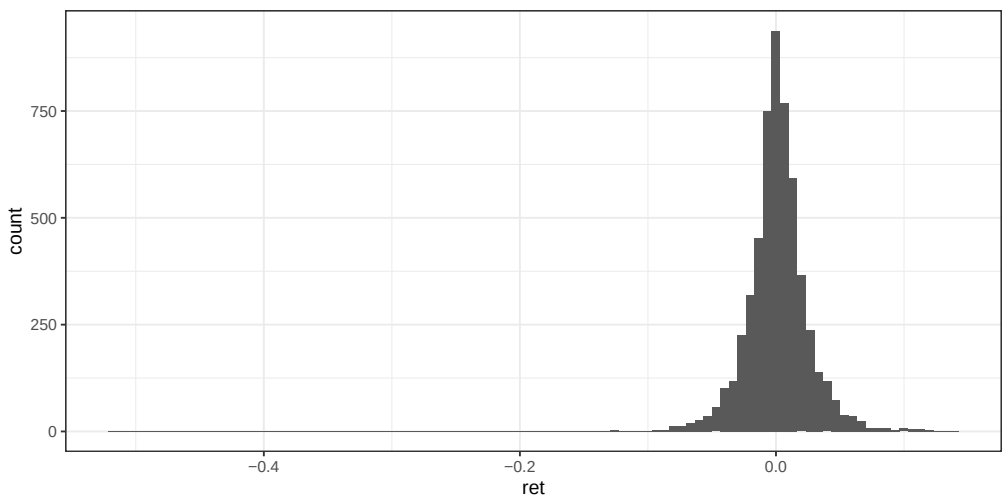
```
returns <- prices |>
  mutate(ret = adjusted / lag(adjusted) - 1) |>
  select(symbol, date, ret) |>
  drop_na()
```

- To understand the code chunk above: Net returns are $r_{t+1,i} := (P_{t+1,i} - P_{t,i})/P_{t,i} = R_{t+1,i} - 1$ where $P_{t+1,i}$ are daily prices (adjusted for dividends, splits, etc)

Inputs

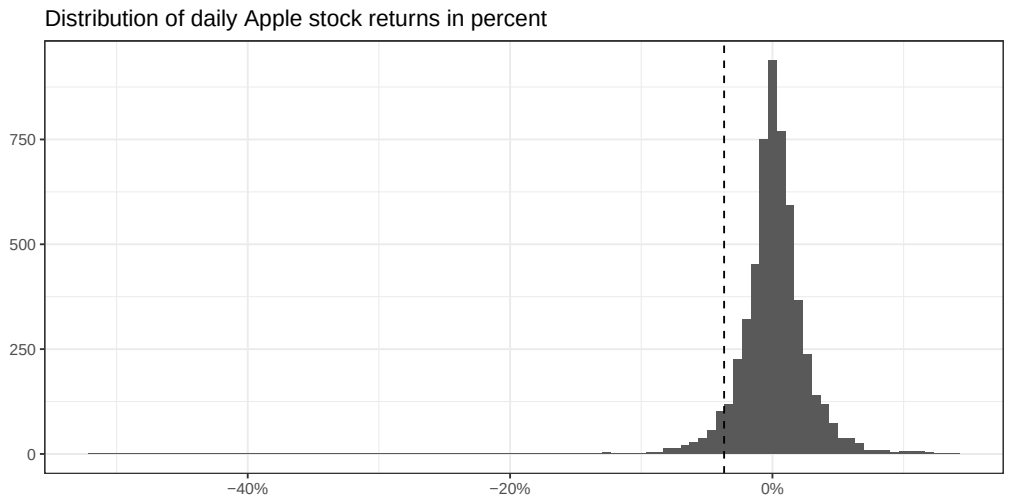
- Data (data frame being plotted)
- Geometrics (geometric shape that represents the data (point, boxplot, histogram)
- Aesthetics (color, size, shape)

```
p1 <- returns |>
  ggplot(aes(x = ret)) +
  geom_histogram(bins = 100)
p1
```



ggplot2 - (Many) other options

```
quantile_05 <- quantile(returns |> pull(ret), probs = 0.05)

p1 +
  geom_vline(aes(xintercept = quantile_05), linetype = "dashed") +
  scale_x_continuous(labels = scales::percent) +
  labs(x = NULL, y = NULL, title = "Distribution of daily Apple stock returns in percent")
```



- Many more themes here: https://ggplot2.tidyverse.org/reference/ggtheme.html
- Virtually unlimited possibilities, see Cedric Scherers blog
- Another source of inspiration: Generative art with `ggplot2`

Sleeves up!

Your background? Tell me something about you!

Figure 1: https://padlet.com/stefanvoigt2/chjr2szc6y1ujwrg

Arguably, a serious issue in finance: Data is hard to get

- Chapters 3 & 4: How to get, clean, and work with CRSP, Compustat, and TRACE data
- Chapter 5: Other (and free!) data providers

Some free alternatives

- Kenneth French homepage (`frenchdata`)
- Global Factor Data
- Open Source Asset Pricing
- IEX (`Riex`)
- SimFin (`simfinapi`)

Walk-through 1: The efficient frontier

Problem

- Our asset universe consists of N assets with expected return μ and variance-covariance matrix Σ
- We want to choose efficient portfolios
- Efficient? Achieve minimum variance *given a desired expected return* $\bar{\mu}$

$$\omega_{\text{eff}}(\bar{\mu}) = \arg\min w'\Sigma w \text{ s.t. } \iota' w = 1 \text{ and } \omega'\mu \geq \bar{\mu}$$

- Good news: There is an analytical solution (I hope you trust me)
- The efficient portfolio weight takes the form (for $\bar{\mu} \geq D/C = \mu'\omega_{\text{mvp}}$)

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

where $\tilde{\lambda} = 2\frac{\bar{\mu} - D/C}{E - D^2/C}$, $C = \iota'\Sigma^{-1}\iota$, $D = \iota'\Sigma^{-1}\mu$, $E = \mu'\Sigma^{-1}\mu$ and $\omega_{\text{mvp}} = \frac{\Sigma^{-1}\iota}{\iota'\Sigma^{-1}\iota}$

More problems

- What is μ? What is Σ?
- Transaction costs? Constraints? See Chapter 17 for an full-blown analysis...

Pragmatic roadmap

1. We need data!
2. We need to estimate $\hat{\mu}$ and $\hat{\Sigma}$!
3. We have to compute $\omega_{\text{eff}}\left(\bar{\mu}\right)$ for different levels of desired returns $\bar{\mu}$

We need data!

- Say our asset universe is the Dow Jones index

```
ticker <- tq_index("DOW")
```

- We use 22 years for estimation (all DJ30 indices that have been traded continuously)

```
index_prices <- tq_get(ticker, get = "stock.prices", from = "2000-01-01", to = "2022-12-31")
index_prices <- index_prices |> group_by(symbol) |> mutate(n = n()) |>
  ungroup() |> filter(n == max(n)) |> select(-n)
```

- We compute (monthly) returns

```
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)
```

We need to estimate $\hat{\mu}$ and $\hat{\Sigma}$!

- Suppose you have T observations of the $(N \times 1)$ return vector r_{t+1}

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

```
## [1] 274  27
```

- The sample counterparts $\hat{\mu}$ and $\hat{\Sigma}$ are

$$\hat{\mu} = \frac{1}{T}\sum_{t=1}^{T} r_t \text{ and } \hat{\Sigma} = \frac{1}{T-1}\sum_{t=1}^{T}\left((r_t - \hat{\mu})(r_t - \hat{\mu})'\right)$$

- We estimate the (sample) moments with R

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

- R provides tons of packages for GARCH, stochastic volatility, RV models

We have to compute $\omega_{\text{eff}}\left(\bar{\mu}\right)$

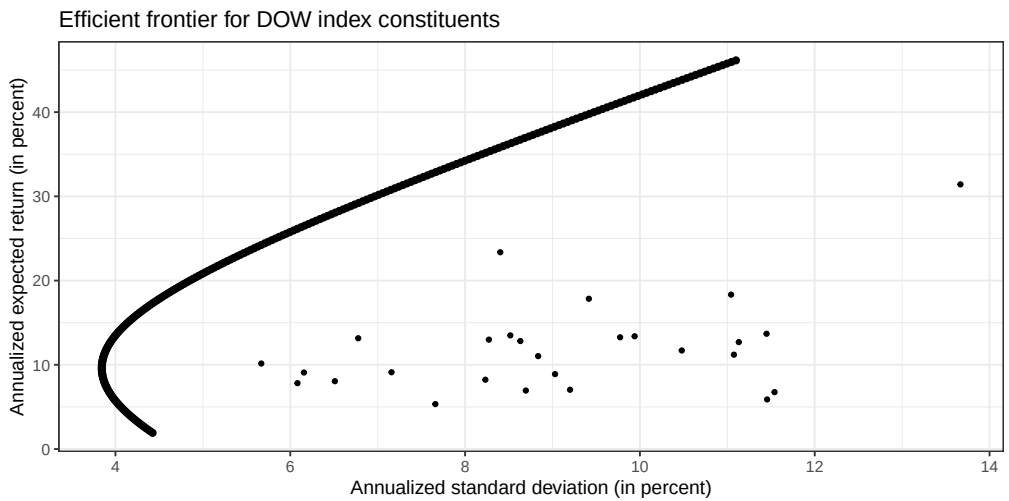
```r
N <- ncol(returns_matrix)
iota <- rep(1, N)
mvp_weights <- solve(Sigma) %*% iota
mvp_weights <- mvp_weights / sum(mvp_weights)
```

```r
mu_bar <- 3 * t(mvp_weights) %*% mu

C <- as.numeric(t(iota) %*% solve(Sigma) %*% iota)
D <- as.numeric(t(iota) %*% solve(Sigma) %*% mu)
E <- as.numeric(t(mu) %*% solve(Sigma) %*% mu)

lambda_tilde <- as.numeric(2 * (mu_bar - D / C) / (E - D^2 / C))
efp_weights <- mvp_weights +
  lambda_tilde / 2 * (solve(Sigma) %*% mu - D * mvp_weights)
```

```r
c <- seq(from = -0.4, to = 1.9, by = 0.01)
res <- tibble(
  c = c,
  mu = NA,
  sd = NA
)

for (i in seq_along(c)) {
  w <- (1 - c[i]) * mvp_weights + (c[i]) * efp_weights
  res$mu[i] <- 12 * 100 * t(w) %*% mu
  res$sd[i] <- 12 * sqrt(100) * sqrt(t(w) %*% Sigma %*% w)
}
```

We have to compute $\omega_{\text{eff}}\left(\bar{\mu}\right)$

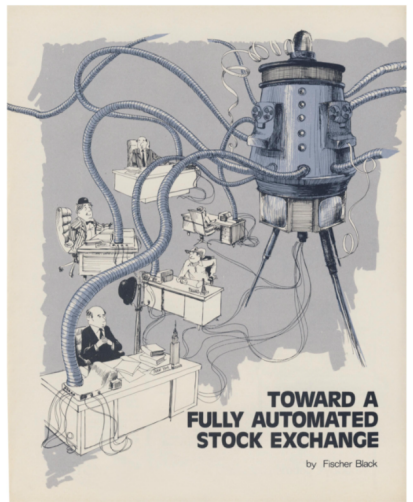
```
res |>
  ggplot(aes(x = sd, y = mu)) +
  geom_point() +
  geom_point(data = tibble(mu = 12 * 100 * mu, sd = 12 * 10 * sqrt(diag(Sigma))),
    aes(y = mu, x = sd), size = 1) +
  labs(x = "Annualized standard deviation (in percent)", y = "Annualized expected return (in percent)",
    title = "Efficient frontier for DOW index constituents")
```



Buzzword bingo: Machine learning

Machine Learning

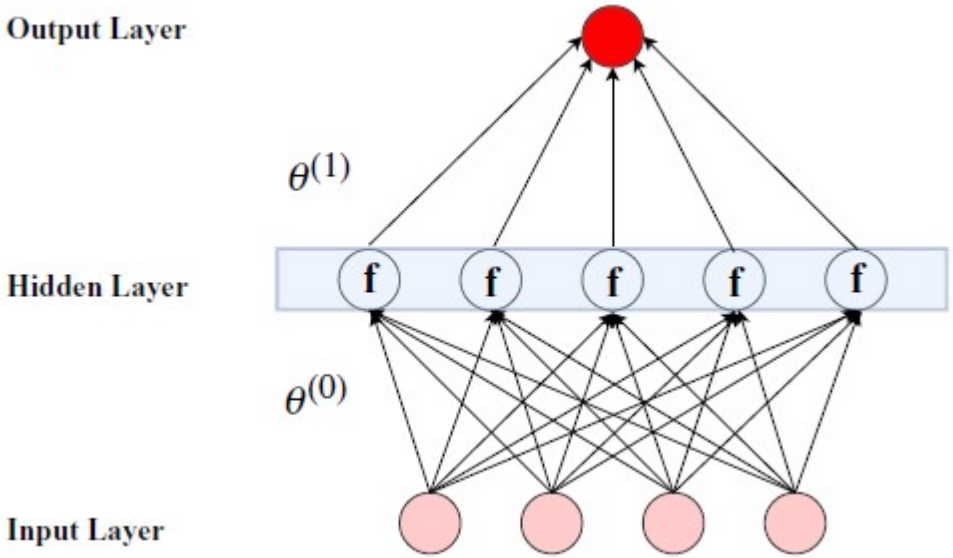
- Diverse collection of high-dimensional models for statistical prediction
- “regularization” methods for model selection and mitigation of overfit
- efficient algorithms
- Machine learning has great potential for improving risk premium measurement, which is fundamentally a problem of prediction



But...

- improved predictions are only measurements
- The measurements do not tell us about economic mechanisms or equilibria
- Machine learning methods on their own do not identify deep fundamental associations among asset prices and conditioning variables

Neural networks: Intuition



- 4 input units
- In this example, there is 1 hidden layer with 5 neurons
- Each neurons receives information from each input layer
- The results from each neuron, $x_k^1 = f\left(\theta_k^0 + \sum_{j=1}^{4} z_j \theta_{k,j}^0\right)$ are finally aggregated into one output forecast

$$\theta_0^1 + \sum_{j=1}^{5} x_j^1 \theta_j^1$$

Plenty of decisions

- Depth (number of hidden layers), Activation function, number of neurons, connections of units (dense or sparse), regularization to avoid overfitting, learning rate
- There is no clear theoretical guidance on these choices but rather a large number of rules of thumbs
- Despite the computational challenges, implementation in R is not tedious at all: We can use the API to tensorflow
- Take a look at this amazing visualization
- Due to the data transformation process that DNNs perform, they are highly sensitive to the individual scale of the feature values: Standardize the feature sets!

Want to use neural networks in R?

- Follow the `tensorflow` installation steps and study some of their examples

Most of Machine learning in one expression

- The structure is always the same: regularization and empirical choice of tuning parameters
- Method defines function class $\mathcal{F}$ (e.g., linear model) and a regularizer $R(f)$ (shrinkage intensity, depth of tree) that expresses the complexity of a function
- Picking the prediction function then involves two steps: First, conditional on a level of complexity, pick the best in-sample loss-minimizing function

$$\min \sum_{i=1}^{n} L\left(f(x_i), y_i\right) \text{ over } f \in \mathcal{F} \text{ subject to } R(f) \leq c$$

- Second, estimate the optimal level of complexity c using empirical tuning

Final Case Study: Option Pricing

- Recall: The value of a call option for a non-dividend-paying underlying stock in terms of the Black–Scholes parameters is:

$$C(S_t, t) = N(d_1)S_t - N(d_2)Ke^{-r(T-t)}$$

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

- V is the price of the option as a function of stock price S and time t, r is the risk-free interest rate, and σ is the volatility of the stock. $N(\cdot)$ is the standard normal cumulative distribution function.

```
black_scholes_price <- function(S, K = 70, r = 0, T = 1, sigma = 0.2) {
  d1 <- (log(S / K) + (r + sigma^2 / 2) * T) / (sigma * sqrt(T))
  value <- S * pnorm(d1) - K * exp(-r * T) * pnorm(d1 - sigma * sqrt(T))

  return(value)
}
```

- Can machine learning methods **learn** the Black-Scholes equation after observing different specifications and corresponding prices?

Start with simulated data

- We compute option prices (`option_prices`) for Call options for a grid of different combinations of maturity (`T`), risk-free rate (`r`), volatility (`sigma`), strike price (`K`) and current stock price (`S`)
- To make it harder: Add an idiosyncratic error term to each observation

```
set.seed(420)

option_prices <- expand_grid(
  S = 40:60,
  K = 20:90,
  r = seq(from = 0, to = 0.05, by = 0.01),
  T = seq(from = 3 / 12, to = 2, by = 1 / 12),
  sigma = seq(from = 0.1, to = 0.8, by = 0.1)
) |>
  mutate(
    black_scholes = black_scholes_price(S, K, r, T, sigma),
    observed_price = map(
      black_scholes,
      function(x) x + rnorm(2, sd = 0.15)
    )
  ) |>
  unnest(observed_price)
```

Learning Black-Scholes

- R provides unparalleled workflows for ML: `tidymodels`
- Excellent documentation: https://www.tmwr.org/

```
library(tidymodels) # For ML applications
split <- initial_split(option_prices, prop = 1 / 100)
```

- We start with a pre-processing plan (`recipe`)

```
rec <- recipe(observed_price ~ .,
  data = option_prices
) |>
  step_rm(black_scholes) |>
  step_normalize(all_predictors())
```

Build a model with tidymodels

- Makes use of a range of packages combined in `tidymodels`

```
nnet_model <- mlp(
  epochs = 500,
  hidden_units = 10,
) |>
  set_mode("regression") |>
  set_engine("keras")
```

- `mlp` contains the definition of our model with all required information (single layer, feed-forward neural network)
- `set_engine("keras")` indicates the API character of the `tidymodels` workflow: Under the hood, the package `keras` is doing the heavy lifting, while `mlp` provides a unified framework to collect the inputs
- Why is this amazing? You can change the model (e.g. change mixture to get ridge or call neural net instead of linear regression)!

```
rf_model <- rand_forest(trees = 50, min_n = 2000) |>
  set_engine("ranger") |>
  set_mode("regression")
```

Results?

- `workflow` ends with combining everything necessary for the (serious) data science workflow: a recipe and a model. Now we are ready to use `fit`.

```
nn_fit <- workflow() |>
  add_recipe(rec) |>
  add_model(nnet_model) |>
  fit(data = training(split))
```

How do the different models perform?

```
nn_fit |> predict(testing(split))
```