Tidy Finance in Vietnam

Mike

2026-02-01

Table of contents

Preface

This book is an independent scholarly work inspired by reproducible research principles popularized in Tidy Finance. It is not affiliated with, endorsed by, or authored by the creators of the original Tidy Finance book. All content, code, and empirical applications are original and tailored to the Vietnamese market.

Motivation

Empirical finance has undergone a fundamental transformation over the past two decades. Advances in computational capacity, open-source statistical software, and data availability have reshaped how financial research is conducted, evaluated, and disseminated. Increasingly, credible empirical work is expected to be transparent, replicable, and extensible, with results generated through scripted workflows rather than manual intervention. Reproducibility, defined as the ability for independent researchers to regenerate empirical results using the same data and methods, has thus become a core norm in modern financial economics.

Despite this progress, the adoption of reproducible research practices has been uneven across markets. In developed financial systems, particularly those with long-established databases and standardized reporting regimes, reproducible empirical workflows are now commonplace. In contrast, research on emerging and frontier markets frequently relies on fragmented datasets, undocumented data cleaning procedures, and implicit institutional assumptions that are difficult to verify or extend. As a result, empirical findings in these markets are often fragile, non-comparable across studies, and costly to update as new data become available.

This book addresses that gap.

It develops a reproducible empirical finance framework designed explicitly for emerging and frontier markets, using Vietnam as a primary empirical case. Rather than adapting developed-market research pipelines post hoc, the book begins from the institutional and data realities of a fast-growing, retail-dominated, regulation-intensive market and builds methodological solutions accordingly. The objective is not merely to analyze Vietnam's financial markets, but to demonstrate how reproducible finance principles can be extended, stress-tested, and refined in environments characterized by data scarcity, institutional heterogeneity, and rapid structural change.

Why Emerging Markets Require Different Empirical Infrastructure

Much of modern empirical finance implicitly assumes the existence of stable, high-frequency, institutionally harmonized datasets. These assumptions are rarely stated, yet they are deeply embedded in standard research designs: survivorship-free security histories, consistent accounting standards, unrestricted trading mechanisms, and deep institutional liquidity.

Emerging and frontier markets challenge each of these assumptions.

In Vietnam, as in many comparable economies, equity markets exhibit binding daily price limits, episodic trading halts, concentrated state ownership, and a predominance of retail investors. Financial disclosures reflect local accounting standards and evolving regulatory frameworks. Corporate actions are frequent, inconsistently documented, and occasionally revised ex post.

These characteristics are not inconveniences to be eliminated through aggressive data cleaning. They shape return dynamics, risk premia, factor construction, and statistical inference itself. An empirical framework that ignores these institutional features risks producing results that are internally inconsistent or externally misleading. A reproducible approach for emerging markets must therefore encode institutional context directly into data schemas, transformation logic, and modeling choices.

Reproducibility as a Research Design Principle

In this book, reproducibility extends beyond the narrow notion of code availability. It is treated as an organizing principle governing the entire empirical research lifecycle.

First, all datasets are constructed from raw inputs through documented, deterministic transformations, ensuring clear data provenance. Second, empirical methods are implemented in a manner that makes modeling assumptions explicit and modifiable. Third, results are generated through scripted pipelines rather than interactive analysis, guaranteeing that updates to data or parameters propagate consistently throughout the analysis. Finally, empirical designs are modular, allowing researchers to substitute markets, sample periods, or variable definitions without rewriting entire workflows.

This approach draws methodological inspiration from the broader reproducible research movement in economics and finance (e.g., Gentzkow and Shapiro (2014); Vilhuber (2020)), while deliberately extending it beyond its original institutional and data environment. The goal is not to reproduce existing studies, but to enable new ones, particularly those that would otherwise be impractical due to fragmented data and institutional complexity.

Vietnam as a Case, Not an Exception

Vietnam serves as the central empirical case throughout the book, but it is not treated as an idiosyncratic exception. Instead, it is presented as a representative example of a class of markets that occupy an intermediate position between frontier and emerging status: large enough to sustain active equity trading, yet still evolving in terms of regulation, disclosure quality, and investor composition.

By grounding methodological development in Vietnam's market structure, the book aims to produce insights that generalize to other contexts, including Southeast Asia, South Asia, Sub-Saharan Africa, and parts of Latin America. Each empirical chapter emphasizes which components are market-specific and which are portable, encouraging readers to adapt the framework rather than adopt it wholesale.

Contribution and Audience

This book makes three primary contributions.

First, it proposes a reproducible empirical finance framework explicitly designed for emerging and frontier markets, integrating institutional detail into data construction and model design. Second, it provides original empirical evidence on asset pricing, liquidity, and market microstructure in Vietnam using consistently constructed datasets. Third, it delivers publication-ready, end-to-end research workflows suitable for academic research, policy analysis, and applied financial work.

The intended audience includes graduate students in finance and economics, academic researchers working on non-developed markets, and practitioners interested in systematic analysis of emerging market equities. Familiarity with basic asset pricing theory and statistical programming is assumed, but no prior experience with Vietnam or similar markets is required.

Structure of the Book

The chapters that follow progress from data infrastructure to empirical application. Early chapters focus on institutional context, data construction, and reproducible workflow design. Subsequent chapters develop asset pricing tests, liquidity measures, and market microstructure analyses tailored to Vietnam's equity market. Each chapter is designed to be self-contained, yet all are linked through a common data and code architecture to ensure internal consistency.

The book concludes by reflecting on the broader implications of reproducible empirical finance for emerging markets research and by outlining directions for future methodological and empirical work.

1 Institutional Background and Market Structure of Vietnam's Equity Market

Empirical analysis of financial markets is inseparable from institutional context. Market design, regulatory constraints, ownership structure, and investor composition shape observed prices, volumes, and returns. In developed markets, many of these features are sufficiently stable and standardized that they fade into the background of empirical research. In emerging markets, by contrast, institutional features are often first-order determinants of empirical outcomes.

This chapter provides the institutional foundation for the empirical analyses developed later in the book. It describes the structure of Vietnam's equity market, the regulatory environment governing trading and disclosure, and the characteristics of listed firms and investors. Rather than offering a purely descriptive account, the discussion emphasizes how institutional features map directly into data construction choices, modeling assumptions, and interpretation of empirical results.

1.1 Evolution of Vietnam's Equity Market

Vietnam's modern equity market is relatively young. Formal stock exchanges were established only in the early 2000s, as part of broader economic reforms aimed at transitioning from a centrally planned system toward a market-oriented economy. Since then, market capitalization, trading volume, and the number of listed firms have grown rapidly, albeit unevenly across sectors and time.

The pace of market development has been shaped by a combination of gradual privatization of state-owned enterprises, episodic regulatory reform, and sustained participation by retail investors. Unlike markets that evolved alongside large institutional investor bases, Vietnam's equity market matured in an environment where individual investors dominate trading activity and informational asymmetries remain substantial.

These features have important empirical implications. Return dynamics may reflect behavioral trading patterns, liquidity shocks can be amplified by coordinated retail activity, and firm-level information is incorporated into prices at varying speeds. A reproducible empirical framework must therefore be capable of capturing these dynamics without imposing assumptions derived from institutionally different markets.

1.2 Exchange Structure and Trading Mechanisms

Vietnam operates multiple equity exchanges, each with distinct listing requirements and trading rules. Trading is conducted through a centralized limit order book, with price-time priority determining execution. Importantly, daily price limits constrain the maximum allowable price movement for individual securities. These limits vary by exchange and security type and are binding during periods of heightened volatility.

Price limits introduce mechanical truncation in observed returns, clustering at upper and lower bounds, and persistence in price movements across days. From an empirical perspective, this challenges standard assumptions about continuous price adjustment and complicates volatility estimation, momentum measurement, and event-study design.

In this book, price limits are treated as structural features rather than anomalies. Data pipelines explicitly preserve limit-hit indicators, and empirical models are adapted to account for constrained price dynamics. This design choice reflects a broader principle: reproducibility in emerging markets requires preserving institutional signals rather than smoothing them away.

1.3 Listing Requirements and Firm Characteristics

Listed firms in Vietnam exhibit substantial heterogeneity in size, ownership structure, and disclosure quality. A defining characteristic of the market is the prevalence of firms with significant state ownership, either directly or through affiliated entities. State ownership affects governance, dividend policy, risk-taking behavior, and responsiveness to market signals.

Accounting disclosures follow Vietnamese Accounting Standards, which differ in important respects from international standards. While convergence efforts are ongoing, historical financial statements often reflect transitional rules, incomplete adoption of fair value accounting, and limited segment reporting. These features complicate cross-firm comparability and longitudinal analysis.

From a reproducible research standpoint, accounting variables cannot be treated as uniform primitives. Variable definitions, reporting lags, and restatement practices must be explicitly documented and encoded into data construction logic. Later chapters demonstrate how accounting data are harmonized in a transparent, version-controlled manner without obscuring underlying institutional differences.

1.4 Investor Composition and Trading Behavior

Retail investors dominate trading volume in Vietnam's equity market. Institutional investors, including domestic funds and foreign participants, play a growing but still secondary role.

This investor composition has implications for liquidity provision, price discovery, and market stability.

Retail-dominated markets tend to exhibit higher turnover, episodic herding behavior, and sensitivity to non-fundamental information. These patterns affect the interpretation of empirical results, particularly in studies of short-term return predictability, volume-return relations, and volatility clustering.

Rather than assuming institutional trading as the default, this book explicitly models liquidity and trading activity in a retail-centric environment. Measures of liquidity, for example, are chosen and constructed to remain meaningful in the presence of small trade sizes, intermittent trading, and order imbalances driven by individual investors.

1.5 Regulatory Environment and Market Frictions

Regulatory oversight of Vietnam's equity market has evolved alongside market development. Trading rules, disclosure requirements, and foreign ownership limits have been periodically revised, sometimes with limited backward compatibility. Regulatory changes can induce structural breaks in data that are not immediately apparent in raw time series.

Short-selling constraints, limited securities lending, and restrictions on derivative usage further distinguish Vietnam's market from developed counterparts. These frictions affect arbitrage activity and the feasibility of certain trading strategies, influencing observed return patterns and factor realizations.

A key principle of the empirical framework developed in this book is regulatory awareness. Data pipelines incorporate regulatory timelines, and empirical tests are designed to be robust to rule changes. This ensures that results are interpretable within the institutional regime in which they arise.

1.6 Implications for Empirical Design

The institutional features described in this chapter motivate several design choices that recur throughout the book:

1. **Data preservation over simplification**: Institutional constraints such as price limits and trading halts are retained and explicitly modeled.
2. **Modular variable construction**: Accounting and market variables are constructed through transparent functions that can be adjusted as standards evolve.
3. **Regime sensitivity**: Empirical analyses are structured to detect and accommodate regulatory and structural breaks.

4. **Context-aware interpretation**: Results are interpreted in light of market structure rather than benchmarked mechanically against developed-market findings.

1.7 Summary

Vietnam's equity market combines rapid growth with distinctive institutional features that challenge conventional empirical finance methods. Price limits, retail investor dominance, state ownership, and evolving regulation shape market outcomes in ways that cannot be ignored or abstracted away. For researchers working in such environments, reproducibility requires more than clean code and documented data—it requires embedding institutional context directly into empirical design.

2 Constructing and Analyzing Equity Return Series

This chapter develops a practical framework for transforming raw equity price records into return series suitable for empirical financial analysis. The focus is on methodological clarity and reproducibility, with particular attention to data issues that are prevalent in emerging equity markets such as Vietnam.

The discussion proceeds from individual stocks to a broad market cross-section, using constituents of the VN30 index as the primary empirical setting.

2.1 Data Access and Preparation

We begin by loading the core numerical and data manipulation libraries. These provide all functionality required for return construction without relying on specialized financial wrappers.

```
import pandas as pd
import numpy as np
```

Historical price data are stored in an S3-compatible object storage system. Access credentials are supplied via environment variables, which keeps sensitive information separate from the analysis and supports collaborative reproducibility.

```
import os
import boto3
from botocore.client import Config


class ObjectStorage:
    def __init__(self):
        self.client = boto3.client(
            "s3",
            endpoint_url=os.environ["MINIO_ENDPOINT"],
            aws_access_key_id=os.environ["MINIO_ACCESS_KEY"],
```

```
            aws_secret_access_key=os.environ["MINIO_SECRET_KEY"],
            region_name=os.getenv("MINIO_REGION", "us-east-1"),
            config=Config(signature_version="s3v4"),
        )


storage = ObjectStorage()
bucket = os.environ["MINIO_BUCKET"]
```

The daily price file is read directly into memory. We explicitly parse dates and harmonize variable names to avoid ambiguity in later steps.

```
from io import BytesIO

prices = pd.read_csv(
    BytesIO(
        storage.client.get_object(
            Bucket=bucket,
            Key="historycal_price/dataset_historical_price.csv",
        )["Body"].read()
    ),
    low_memory=False,
)


prices["date"] = pd.to_datetime(prices["date"])


prices["adjusted_close"] = prices["close_price"] * prices["adj_ratio"]


prices = prices.rename(
    columns={
        "vol_total": "volume",
        "open_price": "open",
        "low_price": "low",
        "high_price": "high",
        "close_price": "close",
    }
)

prices = prices.sort_values(["symbol", "date"])
```

Adjusted closing prices incorporate mechanical changes due to corporate actions such as cash dividends and stock splits. Using adjusted prices ensures that subsequent return calculations reflect investor-relevant performance rather than accounting artifacts.

2.2 Examining a Single Equity

To ground the discussion, we isolate the trading history of a single large-cap stock, FPT, over a long sample period.

```
import datetime as dt

start = pd.Timestamp("2000-01-01")
end = pd.Timestamp(dt.date.today().year - 1, 12, 31)


fpt = prices.loc[
    (prices["symbol"] == "FPT")
    & (prices["date"] >= start)
    & (prices["date"] <= end),
    ["date", "symbol", "volume", "open", "low", "high", "close", "adjusted_close"],
].copy()
```

This subset contains the standard daily market variables required for most empirical studies. Before computing returns, it is good practice to visually inspect the price series.

```
from plotnine import ggplot, aes, geom_line, labs
```

```
(
    ggplot(fpt, aes(x="date", y="adjusted_close"))
    + geom_line()
    + labs(title="Adjusted price path of FPT", x="", y="")
)
```

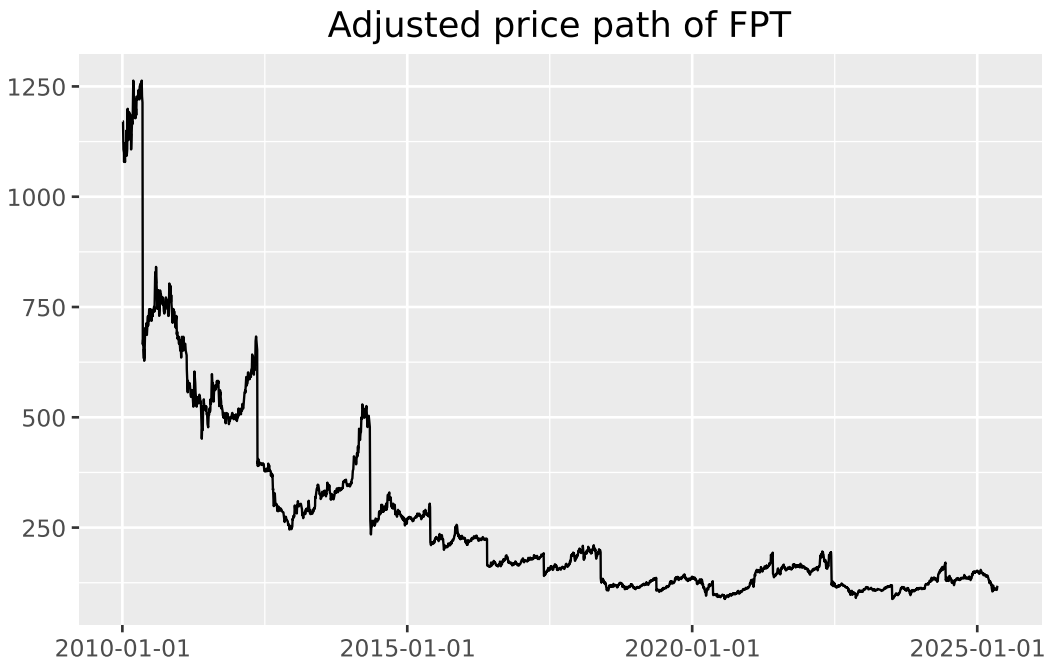


Figure 2.1: Prices are in VND, adjusted for dividend payments and stock splits.

2.3 From Prices to Returns

Most empirical asset pricing models are formulated in terms of returns rather than price levels. The simple daily return is defined as

$$r_t = \frac{p_t}{p_t - 1} - 1,$$

where p_t denotes the adjusted closing price at the end of trading day t.

Before computing returns, we must address invalid price observations. In Vietnamese equity data, adjusted prices occasionally take the value zero. These entries typically arise from IPO placeholders, trading suspensions, or historical backfilling conventions and cannot be used to compute meaningful returns.

```
prices.loc[prices["adjusted_close"] <= 0, ["symbol", "date", "adjusted_close"]].head()
```

	symbol	date	adjusted_close
33886	ADP	2010-02-09	0.0

	symbol	date	adjusted_close
33887	ADP	2010-02-24	0.0
33888	ADP	2010-03-01	0.0
33889	ADP	2010-03-03	0.0
33890	ADP	2010-03-12	0.0

We therefore exclude non-positive adjusted prices and compute returns stock by stock. Correct chronological ordering is essential.

```
returns = (
    prices
    .loc[prices["adjusted_close"] > 0]
    .sort_values(["symbol", "date"])
    .assign(ret=lambda x: x.groupby("symbol")["adjusted_close"].pct_change())
    [["symbol", "date", "ret"]]
)
returns = returns.dropna(subset=["ret"])
```

The initial return for each stock is missing by construction, since no lagged price is available. These observations are mechanical and can safely be removed in most applications.

2.4 Limiting the Influence of Extreme Returns

Daily return series often contain extreme observations driven by data errors, thin trading, or abrupt price adjustments. A common approach is to winsorize returns using cross-sectional percentile cutoffs.

```
def winsorize_cs(df, column="ret", lower_q=0.01, upper_q=0.99):
    lo = df[column].quantile(lower_q)
    hi = df[column].quantile(upper_q)
    out = df.copy()
    out[column] = out[column].clip(lo, hi)
    return out

returns = winsorize_cs(returns)
```

Applying winsorization across the full cross-section limits the impact of extreme market-wide observations while preserving relative differences between firms. Winsorizing within each stock is rarely appropriate in panel settings and can severely distort illiquid securities.

2.5 Distributional Features of Returns

We next examine the empirical distribution of daily returns for FPT. The figure below also marks the historical 5 percent quantile, which provides a simple, non-parametric measure of downside risk.

```
from mizani.formatters import percent_format
from plotnine import geom_histogram, geom_vline, scale_x_continuous


fpt_ret = returns.loc[returns["symbol"] == "FPT"].copy()
q05 = fpt_ret["ret"].quantile(0.05)
```

```
(
    ggplot(fpt_ret, aes(x="ret"))
    + geom_histogram(bins=100)
    + geom_vline(xintercept=q05, linetype="dashed")
    + scale_x_continuous(labels=percent_format())
    + labs(title="Distribution of daily FPT returns", x="", y="")
)
```

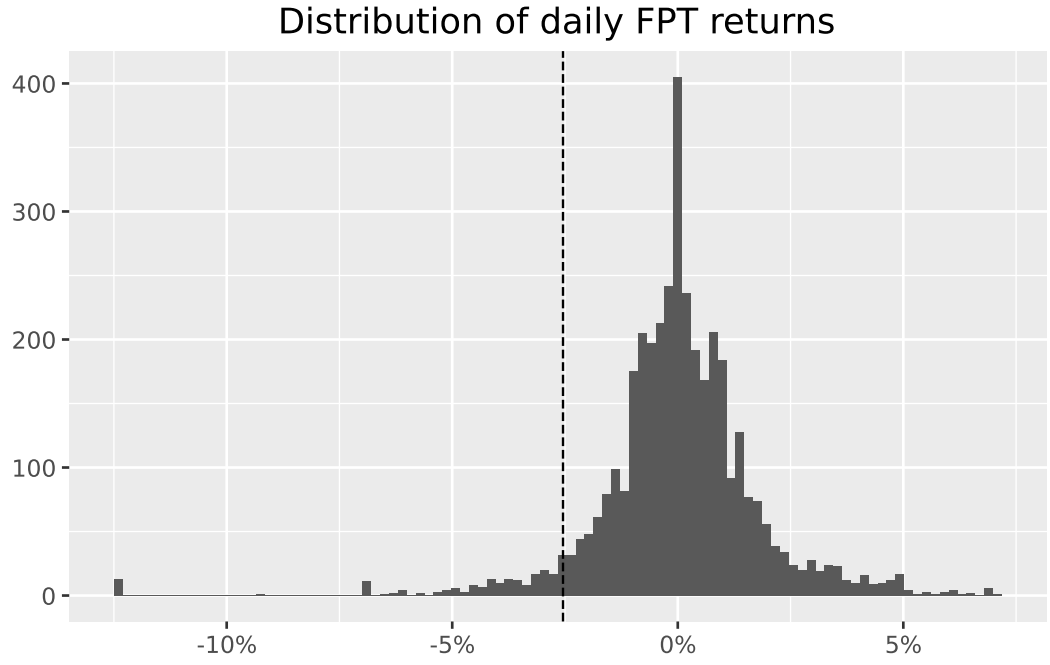


Figure 2.2: The dotted vertical line indicates the historical five percent quantile.

Summary statistics offer a compact description of return behavior and should always be inspected before formal modeling.

```
returns["ret"].describe().round(3)
```

```
count    4305063.000
mean           0.000
std            0.035
min           -0.125
25%           -0.004
50%            0.000
75%            0.003
max            0.130
Name: ret, dtype: float64
```

Computing these statistics by calendar year can reveal periods of elevated volatility or structural change.

```
(
    returns
    .assign(year=lambda x: x["date"].dt.year)
    .groupby("year")["ret"]
    .describe()
    .round(3)
)
```

year	count	mean	std	min	25%	50%	75%	max
2010	131548.0	-0.001	0.036	-0.125	-0.021	0.0	0.018	0.13
2011	166826.0	-0.003	0.033	-0.125	-0.020	0.0	0.011	0.13
2012	177938.0	0.000	0.033	-0.125	-0.012	0.0	0.015	0.13
2013	180417.0	0.001	0.033	-0.125	-0.004	0.0	0.008	0.13
2014	181907.0	0.001	0.034	-0.125	-0.008	0.0	0.011	0.13
2015	197881.0	0.000	0.033	-0.125	-0.006	0.0	0.005	0.13
2016	227896.0	0.000	0.035	-0.125	-0.005	0.0	0.003	0.13
2017	283642.0	0.001	0.034	-0.125	-0.002	0.0	0.001	0.13
2018	329887.0	0.000	0.035	-0.125	0.000	0.0	0.000	0.13
2019	352754.0	0.000	0.033	-0.125	0.000	0.0	0.000	0.13
2020	369367.0	0.001	0.035	-0.125	0.000	0.0	0.000	0.13
2021	379415.0	0.002	0.038	-0.125	-0.005	0.0	0.007	0.13

year	count	mean	std	min	25%	50%	75%	max
2022	387050.0	-0.001	0.038	-0.125	-0.008	0.0	0.004	0.13
2023	391605.0	0.001	0.034	-0.125	-0.002	0.0	0.002	0.13
2024	400379.0	0.000	0.031	-0.125	-0.002	0.0	0.000	0.13
2025	146551.0	0.000	0.037	-0.125	-0.004	0.0	0.002	0.13

2.6 Expanding to a Market Cross-Section

The same procedures apply naturally to a larger universe of stocks. We now restrict attention to the constituents of the VN30 index.

```
vn30 = [
    "ACB","BCM","BID","BVH","CTG","FPT","GAS","GVR","HDB","HPG",
    "MBB","MSN","MWG","PLX","POW","SAB","SHB","SSB","STB","TCB",
    "TPB","VCB","VHM","VIB","VIC","VJC","VNM","VPB","VRE","EIB",
]


prices_vn30 = prices.loc[prices["symbol"].isin(vn30)]
from plotnine import theme
```

```
(
    ggplot(prices_vn30, aes(x="date", y="adjusted_close", color="symbol"))
    + geom_line()
    + labs(title="Adjusted prices of VN30 constituents", x="", y="")
    + theme(legend_position="none")
)
```

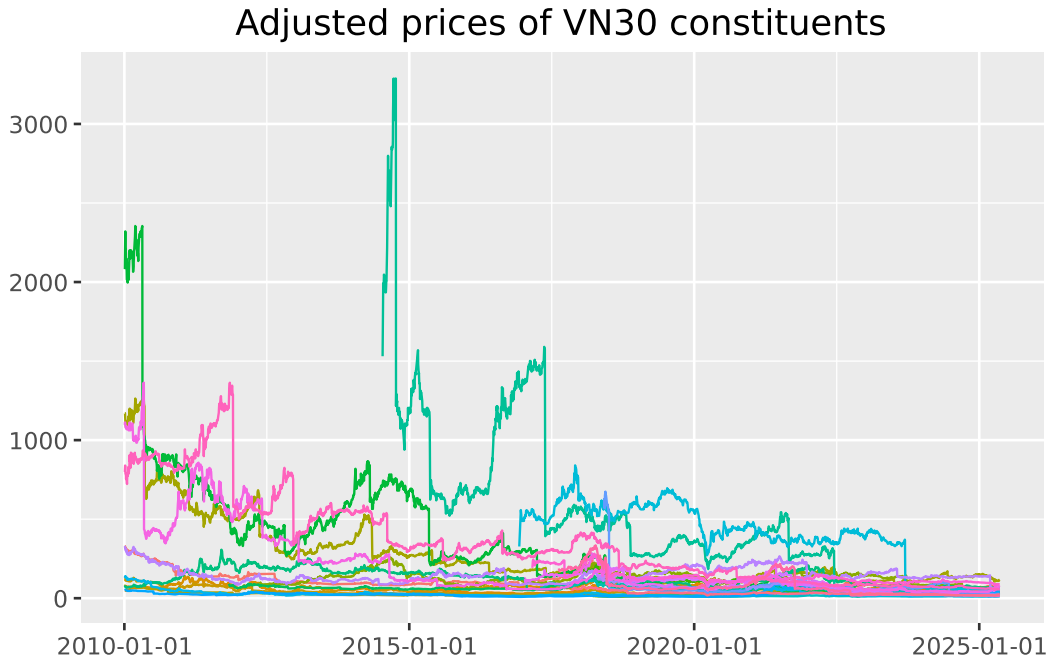


Figure 2.3: Prices in VND, adjusted for dividend payments and stock splits.

Returns for the VN30 universe are computed analogously.

```
returns_vn30 = (
    prices_vn30
    .sort_values(["symbol", "date"])
    .assign(ret=lambda x: x.groupby("symbol")["adjusted_close"].pct_change())
    [["symbol", "date", "ret"]]
    .dropna()
)


returns_vn30.groupby("symbol")["ret"].describe().round(3)
```

symbol	count	mean	std	min	25%	50%	75%	max
ACB	3822.0	-0.000	0.023	-0.407	-0.006	0.0	0.007	0.097
BCM	1795.0	0.001	0.027	-0.136	-0.010	0.0	0.010	0.159
BID	2811.0	0.000	0.024	-0.369	-0.010	0.0	0.011	0.070
BVH	3825.0	0.000	0.024	-0.097	-0.012	0.0	0.012	0.070

symbol	count	mean	std	min	25%	50%	75%	max
CTG	3825.0	0.000	0.024	-0.376	-0.010	0.0	0.010	0.070
EIB	3825.0	-0.000	0.022	-0.302	-0.008	0.0	0.008	0.070
FPT	3825.0	-0.000	0.024	-0.439	-0.008	0.0	0.009	0.070
GAS	3236.0	0.000	0.022	-0.289	-0.009	0.0	0.010	0.070
GVR	1775.0	0.001	0.030	-0.137	-0.014	0.0	0.016	0.169
HDB	1828.0	-0.001	0.028	-0.391	-0.009	0.0	0.010	0.070
HPG	3825.0	-0.001	0.032	-0.581	-0.010	0.0	0.011	0.070
MBB	3371.0	-0.000	0.023	-0.473	-0.008	0.0	0.008	0.069
MSN	3825.0	0.000	0.024	-0.553	-0.010	0.0	0.010	0.070
MWG	2701.0	-0.000	0.035	-0.751	-0.009	0.0	0.011	0.070
PLX	2009.0	-0.000	0.021	-0.140	-0.010	0.0	0.010	0.070
POW	1784.0	0.000	0.023	-0.071	-0.012	0.0	0.011	0.102
SAB	2100.0	-0.000	0.024	-0.745	-0.008	0.0	0.007	0.070
SHB	3824.0	-0.000	0.028	-0.338	-0.013	0.0	0.013	0.100
SSB	1029.0	-0.000	0.023	-0.292	-0.005	0.0	0.004	0.070
STB	3825.0	0.000	0.024	-0.321	-0.010	0.0	0.010	0.070
TCB	1732.0	-0.000	0.035	-0.884	-0.009	0.0	0.010	0.070
TPB	1761.0	-0.001	0.029	-0.477	-0.009	0.0	0.009	0.070
VCB	3825.0	-0.000	0.024	-0.539	-0.009	0.0	0.009	0.070
VHM	1744.0	-0.000	0.024	-0.419	-0.009	0.0	0.008	0.070
VIB	2072.0	-0.000	0.031	-0.489	-0.009	0.0	0.010	0.109
VIC	3825.0	-0.000	0.027	-0.673	-0.008	0.0	0.008	0.070
VJC	2046.0	-0.000	0.020	-0.455	-0.007	0.0	0.006	0.070
VNM	3825.0	-0.000	0.023	-0.547	-0.007	0.0	0.007	0.070
VPB	1927.0	-0.000	0.033	-0.678	-0.010	0.0	0.010	0.070
VRE	1871.0	-0.000	0.024	-0.295	-0.012	0.0	0.011	0.070

2.7 Aggregating Returns Across Time

Financial variables are observed at different frequencies. While equity prices are recorded daily, many empirical questions require monthly or annual returns. Lower-frequency returns are constructed by compounding higher-frequency observations.

```
returns_monthly = (
    returns_vn30
    .assign(month=lambda x: x["date"].dt.to_period("M").dt.to_timestamp())
    .groupby(["symbol", "month"], as_index=False)
```

```
    .agg(ret=("ret", lambda x: np.prod(1 + x) - 1))
)
```

Comparing daily and monthly return distributions illustrates how aggregation dampens volatility and alters tail behavior.

```
from plotnine import facet_wrap

fpt_d = returns_vn30.loc[returns_vn30["symbol"] == "FPT"].assign(freq="Daily")
fpt_m = returns_monthly.loc[returns_monthly["symbol"] == "FPT"].assign(freq="Monthly")


fpt_both = pd.concat([
    fpt_d[["ret", "freq"]],
    fpt_m[["ret", "freq"]],
])
```

```
(
    ggplot(fpt_both, aes(x="ret"))
    + geom_histogram(bins=50)
    + scale_x_continuous(labels=percent_format())
    + labs(title="FPT returns at different frequencies", x="", y="")
    + facet_wrap("freq", scales="free")
)
```

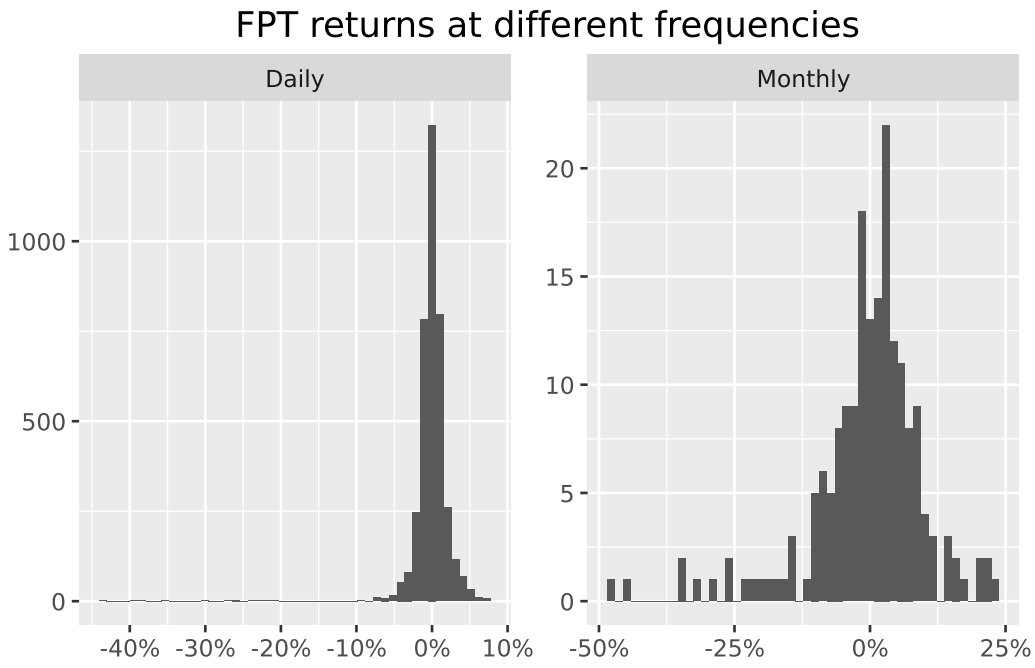
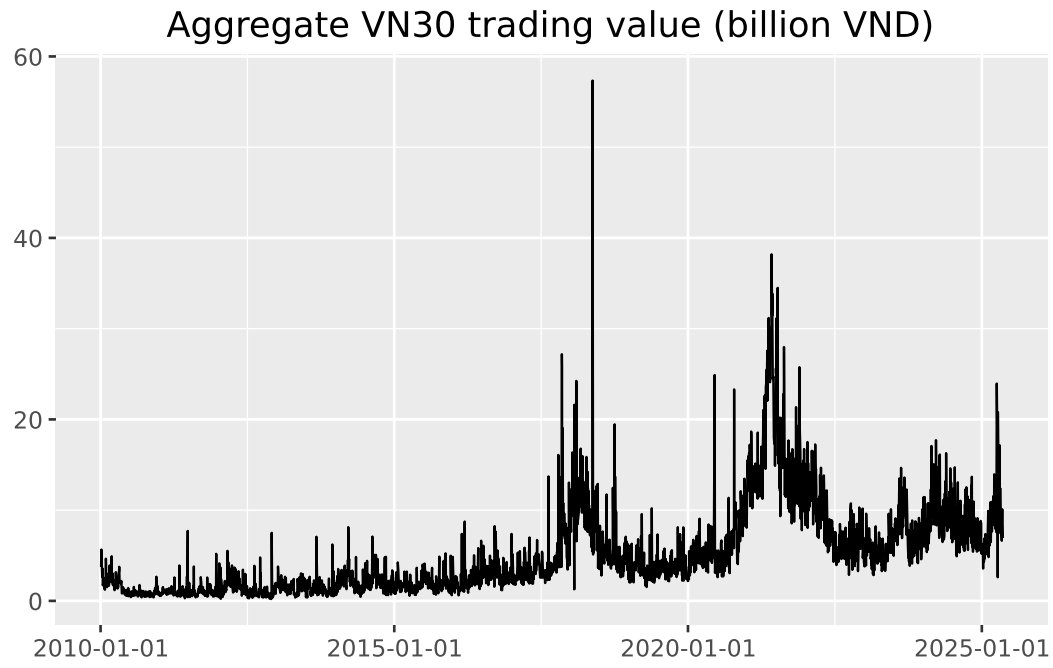


Figure 2.4: Returns are based on prices adjusted for dividend payments and stock splits.

2.8 Aggregation Across Firms: Trading Activity

Aggregation is not limited to time. In some settings, it is informative to aggregate variables across firms. As an illustration, we compute total daily trading value for VN30 stocks by multiplying share volume by adjusted prices and summing across firms.

```
trading_value = (
    prices_vn30
    .assign(value=lambda x: x["volume"] * x["adjusted_close"] / 1e9)
    .groupby("date")["value"]
    .sum()
    .reset_index()
    .assign(value_lag=lambda x: x["value"].shift(1))
)
(
    ggplot(trading_value, aes(x="date", y="value"))
    + geom_line()
    + labs(title="Aggregate VN30 trading value (billion VND)", x="", y="")
)
```



Finally, we assess persistence in trading activity by comparing trading value on consecutive days.

```
from plotnine import geom_point, geom_abline
```

```
(
    ggplot(trading_value, aes(x="value_lag", y="value"))
    + geom_point()
    + geom_abline(intercept=0, slope=1, linetype="dashed")
    + labs(
        title="Persistence in VN30 trading value",
        x="Previous day",
        y="Current day",
    )
)
```

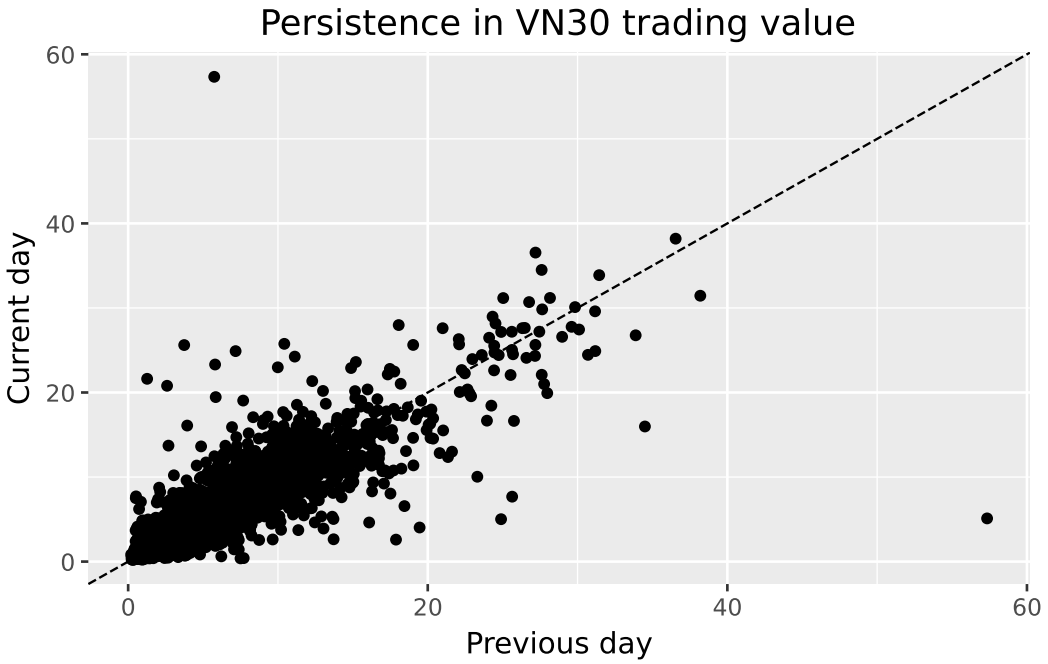


Figure 2.5: Total daily trading volume.

A strong alignment along the 45-degree line indicates that high-activity trading days tend to be followed by similarly active days, a common empirical regularity in equity markets.

2.9 Summary

This chapter established a reproducible workflow for transforming raw price data into return series, diagnosing common data issues, and aggregating information across time and firms. These steps provide the empirical foundation for subsequent analyses of risk, return predictability, and market dynamics in Vietnam's equity market.

3 Modern Portfolio Theory

In the previous chapter, we showed how to download and analyze stock market data with figures and summary statistics. Now, we turn to one of the most fundamental questions in finance: How should an investor allocate their wealth across assets that differ in expected returns, variance, and correlations to optimize their portfolio's performance?

This question might seem straightforward at first glance. Why not simply invest everything in the asset with the highest expected return? The answer lies in a profound insight that transformed financial economics: **risk matters, and it can be managed through diversification**.

Modern Portfolio Theory (MPT), introduced by Markowitz (1952), revolutionized investment decision-making by formalizing the trade-off between risk and expected return. Before Markowitz, investors largely thought about risk on a security-by-security basis. Markowitz's genius was recognizing that what matters is not the risk of individual securities in isolation, but how they contribute to the risk of the *entire portfolio*. This insight was so influential that it earned him the Sveriges Riksbank Prize in Economic Sciences in 1990 and laid the foundation for much of modern finance.

3.0.1 The Core Insight: Diversification as a Free Lunch

MPT relies on a crucial mathematical fact: portfolio risk depends not only on individual asset volatilities but also on the *correlations* between asset returns. This insight reveals the power of diversification—combining assets whose returns don't move in perfect lockstep can reduce overall portfolio risk without necessarily sacrificing expected return.

Consider a simple analogy: Imagine you run a business selling both sunscreen and umbrellas. On sunny days, sunscreen sales boom but umbrella sales suffer; on rainy days, the reverse happens. By selling both products, your total revenue becomes more stable than if you sold only one. The "correlation" between sunscreen and umbrella sales is negative, and combining them reduces the variance of your overall income. This is precisely the logic behind portfolio diversification.

The fruit basket analogy offers another perspective: If all you have are apples and they spoil, you lose everything. With a variety of fruits, some may spoil, but others will stay fresh. Diversification provides insurance against the idiosyncratic risks of individual assets.

3.0.2 The Mean-Variance Framework

At the heart of MPT is **mean-variance analysis**, which evaluates portfolios based on two dimensions:

1. **Expected return (mean)**: The anticipated average profit from holding the portfolio
2. **Risk (variance)**: The dispersion of possible returns around the expected value

The key assumption is that investors care only about these two moments of the return distribution. This assumption is exactly correct if returns are normally distributed, or if investors have quadratic utility functions. Even when these conditions don't hold precisely, mean-variance analysis often provides a good approximation to optimal portfolio choice.

By balancing expected return and risk, investors can construct portfolios that either maximize expected return for a given level of risk, or minimize risk for a desired level of expected return. In this chapter, we derive these optimal portfolio decisions analytically and implement the mean-variance approach computationally.

```
import pandas as pd
import numpy as np
import tidyfinance as tf

from plotnine import *
from mizani.formatters import percent_format
from adjustText import adjust_text
```

3.1 The Asset Universe: Setting Up the Problem

Suppose N different risky assets are available to the investor. Each asset i is characterized by:

- **Expected return** μ_i: The anticipated profit from holding the asset for one period
- **Variance** σ_i^2: The dispersion of returns around the mean
- **Covariances** σ_{ij}: The degree to which asset i's returns move together with asset j's returns

The investor chooses **portfolio weights** ω_i for each asset i. These weights represent the fraction of total wealth invested in each asset. We impose the constraint that weights sum to one:

$$\sum_{i=1}^{N} \omega_i = 1$$

This "budget constraint" ensures that the investor is fully invested—there is no outside option such as keeping money under a mattress. Note that we allow weights to be negative (short selling) or greater than one (leverage), though in practice these positions may face constraints.

3.1.1 The Two Stages of Portfolio Selection

According to Markowitz (1952), portfolio selection involves two distinct stages:

1. **Estimation**: Forming expectations about future security performance based on observations, experience, and economic reasoning
2. **Optimization**: Using these expectations to choose an optimal portfolio

In practice, these stages cannot be fully separated. The estimation stage determines the inputs (μ, Σ) that feed into the optimization stage. Poor estimation leads to poor portfolio choices, regardless of how sophisticated the optimization procedure.

To keep things conceptually clear, we focus primarily on the optimization stage in this chapter. We treat the expected returns and variance-covariance matrix as known, using historical data to compute reasonable proxies. In later chapters, we address the substantial challenges that arise from estimation uncertainty.

3.1.2 Loading and Preparing the Data

We work with the VN30 index constituents—the 30 largest and most liquid stocks on Vietnam's Ho Chi Minh Stock Exchange. This provides a realistic asset universe for a domestic Vietnamese investor.

```
vn30_symbols = [
    "ACB","BCM","BID","BVH","CTG","FPT","GAS","GVR","HDB","HPG",
    "MBB","MSN","MWG","PLX","POW","SAB","SHB","SSB","STB","TCB",
    "TPB","VCB","VHM","VIB","VIC","VJC","VNM","VPB","VRE","EIB"
]
```

We load the historical price data:

```
import pandas as pd
from io import BytesIO
import datetime as dt
import os
import boto3
from botocore.client import Config
```

```
class ConnectMinio:
    def __init__(self):
        self.MINIO_ENDPOINT = os.environ["MINIO_ENDPOINT"]
        self.MINIO_ACCESS_KEY = os.environ["MINIO_ACCESS_KEY"]
        self.MINIO_SECRET_KEY = os.environ["MINIO_SECRET_KEY"]
        self.REGION = os.getenv("MINIO_REGION", "us-east-1")

        self.s3 = boto3.client(
            "s3",
            endpoint_url=self.MINIO_ENDPOINT,
            aws_access_key_id=self.MINIO_ACCESS_KEY,
            aws_secret_access_key=self.MINIO_SECRET_KEY,
            region_name=self.REGION,
            config=Config(signature_version="s3v4"),
        )

    def test_connection(self):
        resp = self.s3.list_buckets()
        print("Connected. Buckets:")
        for b in resp.get("Buckets", []):
            print(" -", b["Name"])

conn = ConnectMinio()
s3 = conn.s3
conn.test_connection()

bucket_name = os.environ["MINIO_BUCKET"]

prices = pd.read_csv(
    BytesIO(
        s3.get_object(
            Bucket=bucket_name,
            Key="historycal_price/dataset_historical_price.csv"
        )["Body"].read()
    ),
    low_memory=False
)

prices["date"] = pd.to_datetime(prices["date"])
prices["adjusted_close"] = prices["close_price"] * prices["adj_ratio"]
prices = prices.rename(columns={
    "vol_total": "volume",
```

```
    "open_price": "open",
    "low_price": "low",
    "high_price": "high",
    "close_price": "close"
})
prices = prices.sort_values(["symbol", "date"])
```

```
Connected. Buckets:
 - dsteam-data
 - rawbctc
```

We filter to keep only the VN30 constituents:

```
prices_daily = prices[prices["symbol"].isin(vn30_symbols)]
prices_daily[["date", "symbol", "adjusted_close"]].head(3)
```

	date	symbol	adjusted_close
18176	2010-01-04	ACB	329.408244
18177	2010-01-05	ACB	329.408244
18178	2010-01-06	ACB	320.258015

3.1.3 Computing Expected Returns

The sample mean return serves as our proxy for expected returns. For each asset i, we compute:

$$\hat{\mu}_i = \frac{1}{T}\sum_{t=1}^{T} r_{i,t}$$

where $r_{i,t}$ is the return of asset i in period t, and T is the total number of periods.

Why monthly returns? While daily data provides more observations, monthly returns offer several advantages for portfolio optimization. First, monthly returns are less noisy and exhibit weaker serial correlation. Second, monthly rebalancing is more realistic for most investors, avoiding excessive transaction costs. Third, the estimation error in mean returns is already substantial—using daily data doesn't materially improve the precision of mean estimates because the mean return scales with the horizon while estimation error scales with the square root of observations.

```
returns_monthly = (prices_daily
  .assign(
    date=prices_daily["date"].dt.to_period("M").dt.to_timestamp()
  )
  .groupby(["symbol", "date"], as_index=False)
  .agg(adjusted_close=("adjusted_close", "last"))
  .assign(
    ret=lambda x: x.groupby("symbol")["adjusted_close"].pct_change()
  )
)
```

3.1.4 Computing Volatilities

Individual asset risk in MPT is quantified using variance (σ_i^2) or its square root, the standard deviation or volatility (σ_i). We use the sample standard deviation as our proxy:

$$\hat{\sigma}_i = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(r_{i,t} - \hat{\mu}_i)^2}$$

Alternative risk measures exist, including Value-at-Risk, Expected Shortfall, and higher-order moments such as skewness and kurtosis. However, variance remains the workhorse measure in portfolio theory because of its mathematical tractability and the central role of the normal distribution in finance.

```
assets = (returns_monthly
  .groupby("symbol", as_index=False)
  .agg(
    mu=("ret", "mean"),
    sigma=("ret", "std")
  )
)
```

3.1.5 Visualizing the Risk-Return Trade-off

Figure 3.1 displays each asset's expected return (vertical axis) against its volatility (horizontal axis). This "mean-standard deviation" space is fundamental to portfolio theory.

```
assets_figure = (
  ggplot(
    assets,
    aes(x="sigma", y="mu", label="symbol")
  )
  + geom_point()
  + geom_text(adjust_text={"arrowprops": {"arrowstyle": "-"}})
  + scale_x_continuous(labels=percent_format())
  + scale_y_continuous(labels=percent_format())
  + labs(
      x="Volatility (Standard Deviation)",
      y="Expected Return",
      title="Expected returns and volatilities of VN30 index constituents"
  )
)
assets_figure.show()
```

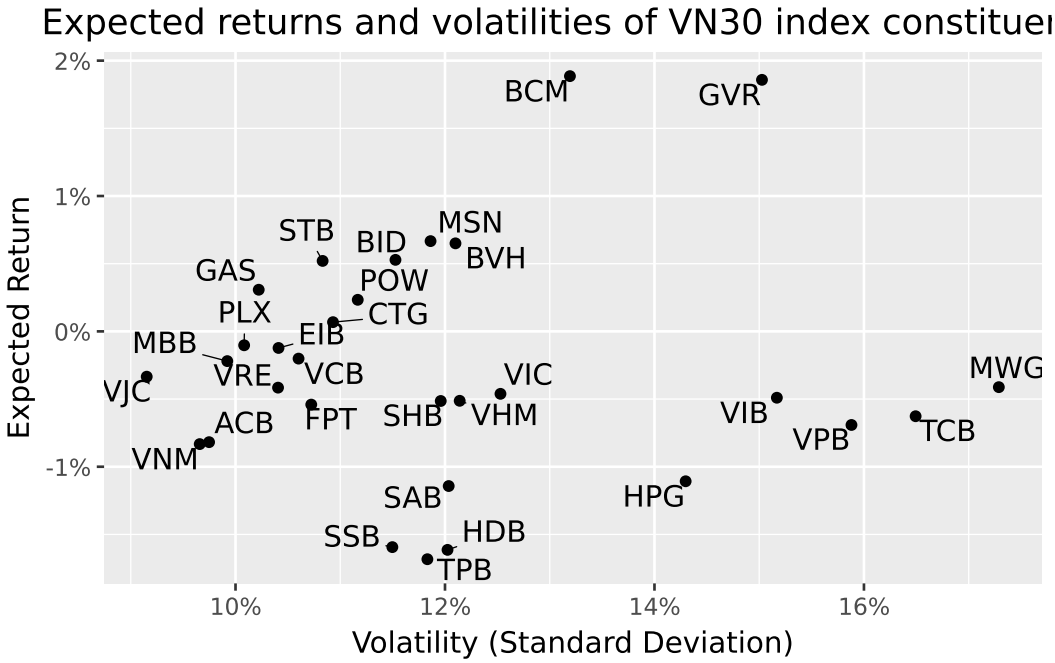


Figure 3.1: Expected returns and volatilities based on monthly returns adjusted for dividend payments and stock splits.

Several observations emerge from this figure. First, there is substantial heterogeneity in both expected returns and volatilities across stocks. Second, the relationship between risk and return

is far from linear. Some high-volatility stocks have low or even negative expected returns. Third, most individual stocks appear to offer poor risk-return trade-offs. As we will see, portfolios can substantially improve upon these individual positions.

3.2 The Variance-Covariance Matrix: Capturing Asset Interactions

3.2.1 Why Correlations Matter

A key innovation of MPT is recognizing that portfolio risk depends critically on how assets move together. The **variance-covariance matrix** Σ captures all pairwise interactions between asset returns.

To understand why correlations matter, consider the variance of a two-asset portfolio:

$$\sigma_p^2 = \omega_1^2\sigma_1^2 + \omega_2^2\sigma_2^2 + 2\omega_1\omega_2\sigma_{12}$$

The third term involves the covariance $\sigma_{12} = \rho_{12}\sigma_1\sigma_2$, where ρ_{12} is the correlation coefficient. When $\rho_{12} < 1$, the portfolio variance is *less* than the weighted average of individual variances. When $\rho_{12} < 0$, the diversification benefit is even more pronounced.

This mathematical fact has profound implications: **You can reduce risk without reducing expected return** by combining assets that don't move perfectly together. This is sometimes called the "only free lunch in finance."

3.2.2 Computing the Variance-Covariance Matrix

We compute the sample covariance matrix as:

$$\hat{\sigma}_{ij} = \frac{1}{T-1}\sum_{t=1}^{T}(r_{i,t} - \hat{\mu}_i)(r_{j,t} - \hat{\mu}_j)$$

First, we reshape the returns data into a wide format with assets as columns:

```
returns_wide = (returns_monthly
  .pivot(index="date", columns="symbol", values="ret")
  .reset_index()
)

sigma = (returns_wide
  .drop(columns=["date"])
  .cov()
)
```

3.2.3 Interpreting the Variance-Covariance Matrix

The diagonal elements of Σ are the variances of individual assets. The off-diagonal elements are covariances, which can be positive (assets tend to move together), negative (assets tend to move in opposite directions), or zero (no linear relationship).

For easier interpretation, we often convert covariances to correlations:

$$\rho_{ij} = \frac{\sigma_{ij}}{\sigma_i \sigma_j}$$

Correlations are bounded between -1 and +1, making them easier to compare across asset pairs.

Figure 3.2 visualizes the variance-covariance matrix as a heatmap.

```
sigma_long = (sigma
  .reset_index()
  .melt(id_vars="symbol", var_name="symbol_b", value_name="value")
)

sigma_long["symbol_b"] = pd.Categorical(
  sigma_long["symbol_b"],
  categories=sigma_long["symbol_b"].unique()[::-1],
  ordered=True
)

sigma_figure = (
  ggplot(
    sigma_long,
    aes(x="symbol", y="symbol_b", fill="value")
  )
  + geom_tile()
  + labs(
      x="", y="", fill="(Co-)Variance",
      title="Sample variance-covariance matrix of VN30 index constituents"
    )
  + scale_fill_continuous(labels=percent_format())
  + theme(axis_text_x=element_text(angle=45, hjust=1))
)
sigma_figure.show()
```

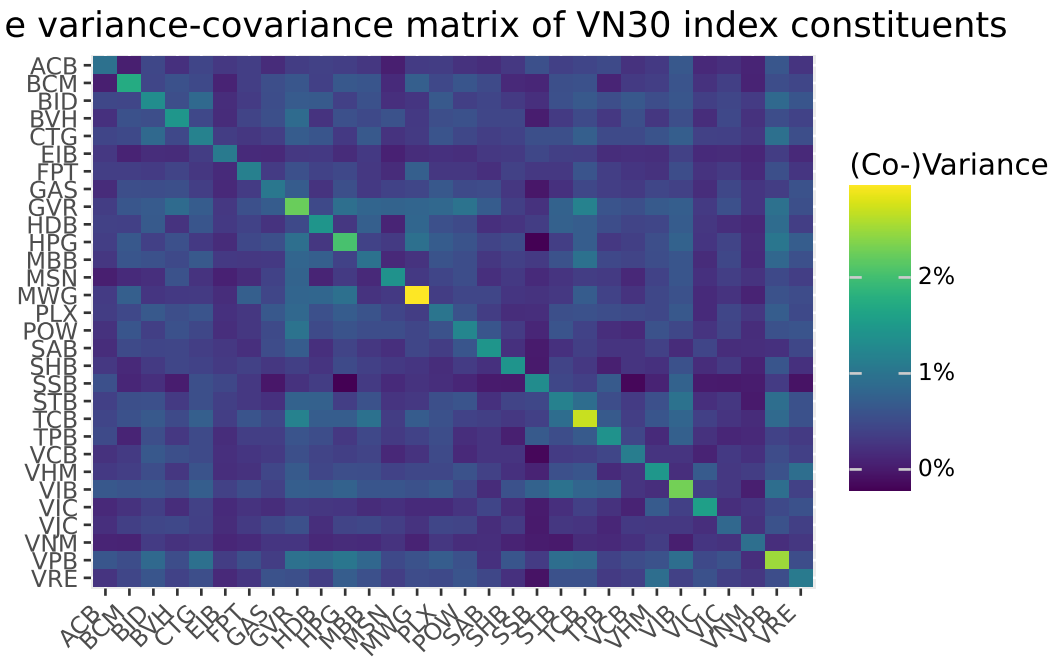


Figure 3.2: Variances and covariances based on monthly returns adjusted for dividend payments and stock splits.

The heatmap reveals important patterns. The diagonal (variances) shows which stocks are most volatile. The off-diagonal patterns show which pairs of stocks tend to move together. In general, stocks within the same sector tend to have higher correlations with each other than with stocks from different sectors.

3.3 The Minimum-Variance Portfolio

3.3.1 Motivation: Risk Minimization as a Benchmark

Before considering expected returns, let's find the portfolio that minimizes risk entirely. This **minimum-variance portfolio (MVP)** serves as an important benchmark and reference point. It represents what an extremely risk-averse investor—one who cares only about minimizing volatility—would choose.

3.3.2 The Optimization Problem

The minimum-variance investor solves:

$$\min_{\omega} \omega' \Sigma \omega$$

subject to the constraint that weights sum to one:

$$\omega' \iota = 1$$

where ι is an $N \times 1$ vector of ones.

In words: minimize portfolio variance, subject to being fully invested.

3.3.3 The Analytical Solution

This is a classic constrained optimization problem that can be solved using Lagrange multipliers. The Lagrangian is:

$$\mathcal{L} = \omega' \Sigma \omega - \lambda(\omega' \iota - 1)$$

Taking the first-order condition with respect to ω:

$$\frac{\partial \mathcal{L}}{\partial \omega} = 2\Sigma\omega - \lambda\iota = 0$$

Solving for ω:

$$\omega = \frac{\lambda}{2}\Sigma^{-1}\iota$$

Using the constraint $\omega' \iota = 1$ to solve for λ:

$$\frac{\lambda}{2}\iota'\Sigma^{-1}\iota = 1 \implies \frac{\lambda}{2} = \frac{1}{\iota'\Sigma^{-1}\iota}$$

Substituting back:

$$\omega_{\text{mvp}} = \frac{\Sigma^{-1}\iota}{\iota'\Sigma^{-1}\iota}$$

This elegant formula shows that the minimum-variance weights depend only on the covariance matrix—expected returns play no role. The inverse covariance matrix Σ^{-1} determines how much to invest in each asset based on its variance and its covariances with all other assets.

3.3.4 Implementation

```
iota = np.ones(sigma.shape[0])
sigma_inv = np.linalg.inv(sigma.values)
omega_mvp = (sigma_inv @ iota) / (iota @ sigma_inv @ iota)
```

3.3.5 Visualizing the Minimum-Variance Weights

Figure 3.3 displays the portfolio weights of the minimum-variance portfolio.

```
assets = assets.assign(omega_mvp=omega_mvp)

assets["symbol"] = pd.Categorical(
  assets["symbol"],
  categories=assets.sort_values("omega_mvp")["symbol"],
  ordered=True
)

omega_figure = (
  ggplot(
    assets,
    aes(y="omega_mvp", x="symbol", fill="omega_mvp>0")
  )
  + geom_col()
  + coord_flip()
  + scale_y_continuous(labels=percent_format())
  + labs(
      x="",
      y="Portfolio Weight",
      title="Minimum-variance portfolio weights"
  )
  + theme(legend_position="none")
)
omega_figure.show()
```

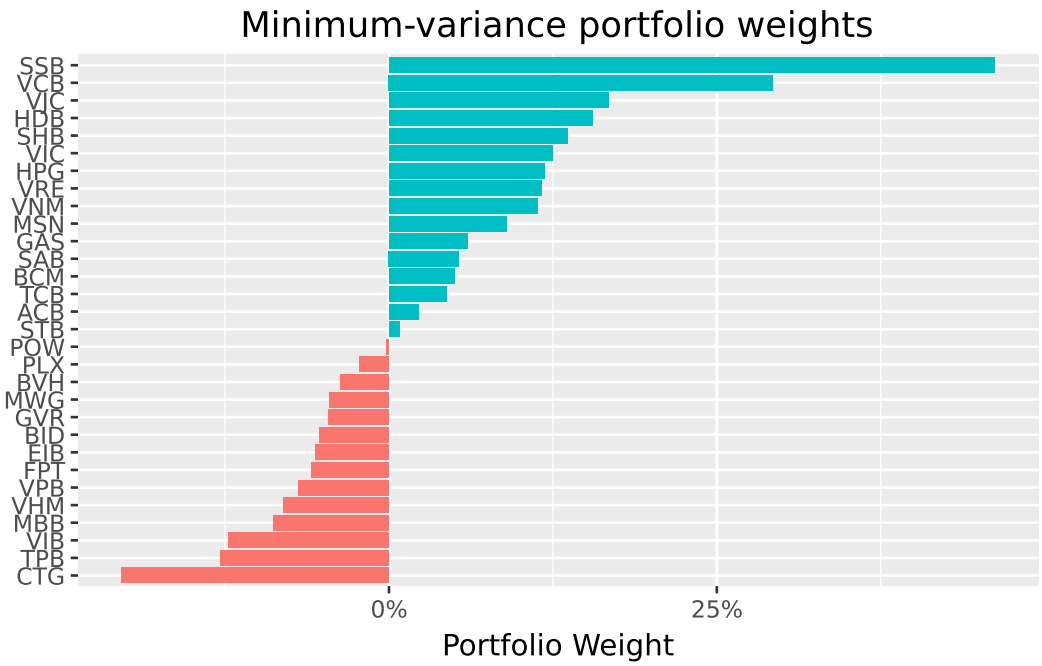


Figure 3.3: Weights are based on historical moments of monthly returns adjusted for dividend payments and stock splits.

Several features of the minimum-variance portfolio are noteworthy. First, many stocks receive zero or near-zero weights. Second, some stocks receive negative weights (short positions). These short positions are not a computational artifact, they reflect the optimizer's attempt to exploit correlations for risk reduction. Third, the weights are quite extreme (both large positive and large negative), which often indicates estimation error amplification, which is a topic we address in later chapters.

3.3.6 Portfolio Performance

Let's compute the expected return and volatility of the minimum-variance portfolio:

```
mu = assets["mu"].values
mu_mvp = omega_mvp @ mu
sigma_mvp = np.sqrt(omega_mvp @ sigma.values @ omega_mvp)

summary_mvp = pd.DataFrame({
  "mu": [mu_mvp],
  "sigma": [sigma_mvp],
  "type": ["Minimum-Variance Portfolio"]
```

```
})
summary_mvp
```

	mu	sigma	type
0	-0.011424	0.043512	Minimum-Variance Portfolio

```
mu_mvp_fmt = f"{mu_mvp:.4f}"
sigma_mvp_fmt = f"{sigma_mvp:.4f}"
print(f"The MVP return is {mu_mvp_fmt} and volatility is {sigma_mvp_fmt}.")
```

```
The MVP return is -0.0114 and volatility is 0.0435.
```

If the expected return is negative, this is not a computational error. The minimum-variance portfolio minimizes risk without regard to expected returns. Because some assets in the sample have negative average returns, the risk-minimizing combination may inherit a negative expected return. This highlights a fundamental limitation of using historical sample means as estimates of expected returns: they are extremely noisy, and can lead to economically unintuitive results even when the optimization mathematics are working correctly.

3.4 Efficient Portfolios: Balancing Risk and Return

3.4.1 The Investor's Trade-off

In most cases, minimizing variance is not the investor's sole objective. A more realistic formulation allows the investor to trade off risk against expected return. The investor might be willing to accept higher portfolio variance in exchange for higher expected returns.

An **efficient portfolio** minimizes variance subject to earning at least some target expected return $\bar{\mu}$. Formally:

$$\min_{\omega} \omega' \Sigma \omega$$

subject to:

$$\omega' \iota = 1 \quad \text{(fully invested)}$$
$$\omega' \mu \geq \bar{\mu} \quad \text{(minimum return)}$$

When $\bar{\mu}$ exceeds the expected return of the minimum-variance portfolio, the investor accepts more risk to earn more return.

3.4.2 Setting the Target Return

For illustration, suppose the investor wants to earn at least the historical average return of the best-performing stock:

```
mu_bar = assets["mu"].max()
print(f"Target expected return: {mu_bar:.5f}")
```

```
Target expected return: 0.01886
```

This is an ambitious target—it means matching the return of the single highest-returning stock while benefiting from diversification to reduce risk.

3.4.3 The Analytical Solution

The constrained optimization problem with an inequality constraint on expected returns can be solved using the Karush-Kuhn-Tucker (KKT) conditions. At the optimum (assuming the return constraint binds), the solution is:

$$\omega_{\text{efp}} = \frac{\lambda^*}{2}\left(\Sigma^{-1}\mu - \frac{D}{C}\Sigma^{-1}\iota\right)$$

where:

- $C = \iota'\Sigma^{-1}\iota$ (a scalar measuring the "size" of the inverse covariance matrix)
- $D = \iota'\Sigma^{-1}\mu$ (capturing the interaction between expected returns and the inverse covariance matrix)
- $E = \mu'\Sigma^{-1}\mu$ (measuring the "signal" in expected returns weighted by inverse covariances)
- $\lambda^* = 2\frac{\bar{\mu} - D/C}{E - D^2/C}$ (the shadow price of the return constraint)

Alternatively, we can express the efficient portfolio as a linear combination of the minimum-variance portfolio and an "excess return" portfolio:

$$\omega_{\text{efp}} = \omega_{\text{mvp}} + \frac{\lambda^*}{2}\left(\Sigma^{-1}\mu - D \cdot \omega_{\text{mvp}}\right)$$

This representation reveals important intuition: the efficient portfolio starts from the minimum-variance portfolio and tilts toward higher-expected-return assets, with the tilt magnitude determined by λ^*.

3.4.4 Implementation

```
C = iota @ sigma_inv @ iota
D = iota @ sigma_inv @ mu
E = mu @ sigma_inv @ mu
lambda_tilde = 2 * (mu_bar - D / C) / (E - (D ** 2) / C)
omega_efp = omega_mvp + (lambda_tilde / 2) * (sigma_inv @ mu - D * omega_mvp)

mu_efp = omega_efp @ mu
sigma_efp = np.sqrt(omega_efp @ sigma.values @ omega_efp)

summary_efp = pd.DataFrame({
  "mu": [mu_efp],
  "sigma": [sigma_efp],
  "type": ["Efficient Portfolio"]
})
```

3.4.5 Comparing the Portfolios

Figure 3.4 plots both portfolios alongside the individual assets.

```
summaries = pd.concat(
  [assets, summary_mvp, summary_efp], ignore_index=True
)

summaries_figure = (
  ggplot(
    summaries,
    aes(x="sigma", y="mu")
  )
  + geom_point(data=summaries.query("type.isna()"))
  + geom_point(data=summaries.query("type.notna()"), color="#F21A00", size=3)
  + geom_label(aes(label="type"), adjust_text={"arrowprops": {"arrowstyle": "-"}})
  + scale_x_continuous(labels=percent_format())
  + scale_y_continuous(labels=percent_format())
  + labs(
      x="Volatility (Standard Deviation)",
      y="Expected Return",
      title="Efficient & minimum-variance portfolios"
    )
```

```
)
summaries_figure.show()
```

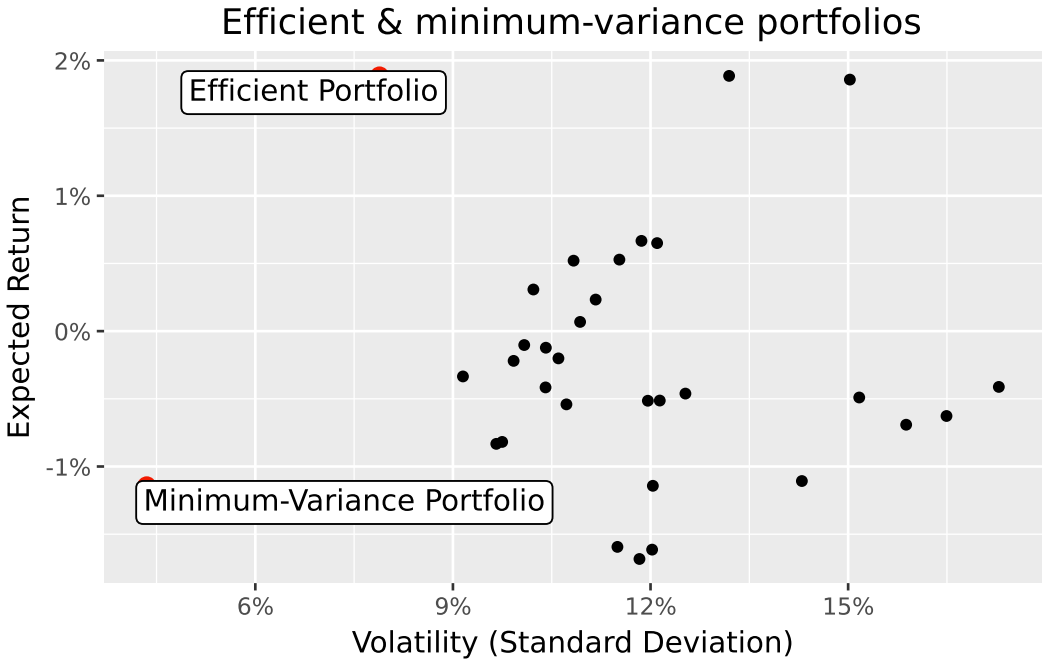


Figure 3.4: The big dots indicate the location of the minimum-variance and the efficient portfolio that delivers the expected return of the stock with the highest return, respectively. The small dots indicate the location of the individual constituents.

The figure demonstrates the substantial diversification benefits of portfolio optimization. The efficient portfolio achieves the same expected return as the highest-returning individual stock but with substantially lower volatility. This "free lunch" from diversification is the central insight of Modern Portfolio Theory.

3.4.6 The Role of Risk Aversion

The target return $\bar{\mu}$ implicitly reflects the investor's risk aversion. Less risk-averse investors choose higher $\bar{\mu}$, accepting more variance to earn more expected return. More risk-averse investors choose $\bar{\mu}$ closer to the minimum-variance portfolio's expected return.

Equivalently, the mean-variance framework can be derived from the optimal decisions of an investor with a mean-variance utility function:

$$U(\omega) = \omega'\mu - \frac{\gamma}{2}\omega'\Sigma\omega$$

where γ is the coefficient of relative risk aversion. The Appendix shows there is a one-to-one mapping between γ and $\bar{\mu}$, so both formulations yield identical efficient portfolios.

3.5 The Efficient Frontier: The Menu of Optimal Portfolios

The **efficient frontier** is the set of all portfolios for which no other portfolio offers higher expected return at the same or lower variance. Geometrically, it traces the upper boundary of achievable (volatility, expected return) combinations.

Every rational mean-variance investor should hold a portfolio on the efficient frontier. Portfolios below the frontier are "dominated," there exists another portfolio with either higher return for the same risk, or lower risk for the same return.

3.5.1 The Mutual Fund Separation Theorem

A remarkable result simplifies the construction of the efficient frontier. The **mutual fund separation theorem** (sometimes called the two-fund theorem) states that any efficient portfolio can be expressed as a linear combination of any two distinct efficient portfolios.

Formally, if ω_{μ_1} and ω_{μ_2} are efficient portfolios earning expected returns μ_1 and μ_2 respectively, then the portfolio:

$$\omega_{a\mu_1+(1-a)\mu_2} = a \cdot \omega_{\mu_1} + (1-a) \cdot \omega_{\mu_2}$$

is also efficient and earns expected return $a\mu_1 + (1-a)\mu_2$.

This result has profound practical implications: an investor needs access to only two efficient "mutual funds" to construct any portfolio on the efficient frontier. The specific funds don't matter—any two distinct efficient portfolios span the entire frontier.

3.5.2 Proof of the Separation Theorem

The proof follows directly from the analytical solution for efficient portfolios. Consider:

$$a \cdot \omega_{\mu_1} + (1-a) \cdot \omega_{\mu_2} = \left(\frac{a\mu_1 + (1-a)\mu_2 - D/C}{E - D^2/C} \right) \left(\Sigma^{-1}\mu - \frac{D}{C}\Sigma^{-1}\iota \right)$$

This expression has exactly the form of the efficient portfolio earning expected return $a\mu_1 + (1-a)\mu_2$, proving the theorem.

3.5.3 Computing the Efficient Frontier

Using the minimum-variance portfolio and our efficient portfolio as the two "funds," we can trace out the entire efficient frontier:

```
efficient_frontier = (
  pd.DataFrame({
    "a": np.arange(-1, 2.01, 0.01)
  })
  .assign(
    omega=lambda x: x["a"].map(lambda a: a * omega_efp + (1 - a) * omega_mvp)
  )
  .assign(
    mu=lambda x: x["omega"].map(lambda w: w @ mu),
    sigma=lambda x: x["omega"].map(lambda w: np.sqrt(w @ sigma @ w))
  )
)
```

Note that we allow a to range from -1 to 2, which means some portfolios involve shorting one of the two basis funds and leveraging into the other. This traces out both the upper and lower portions of the frontier hyperbola.

3.5.4 Visualizing the Efficient Frontier

Figure 3.5 displays the efficient frontier alongside individual assets and the benchmark portfolios.

```
summaries = pd.concat(
  [summaries, efficient_frontier], ignore_index=True
)

summaries_figure = (
  ggplot(
    summaries,
    aes(x="sigma", y="mu")
  )
  + geom_point(data=summaries.query("type.isna()"))
  + geom_line(data=efficient_frontier, color="blue", alpha=0.7)
  + geom_point(data=summaries.query("type.notna()"), color="#F21A00", size=3)
  + geom_label(aes(label="type"), adjust_text={"arrowprops": {"arrowstyle": "-"}})
  + scale_x_continuous(labels=percent_format())
```

```
  + scale_y_continuous(labels=percent_format())
  + labs(
      x="Volatility (Standard Deviation)",
      y="Expected Return",
      title="The Efficient Frontier and VN30 Constituents"
    )
)
summaries_figure.show()
```

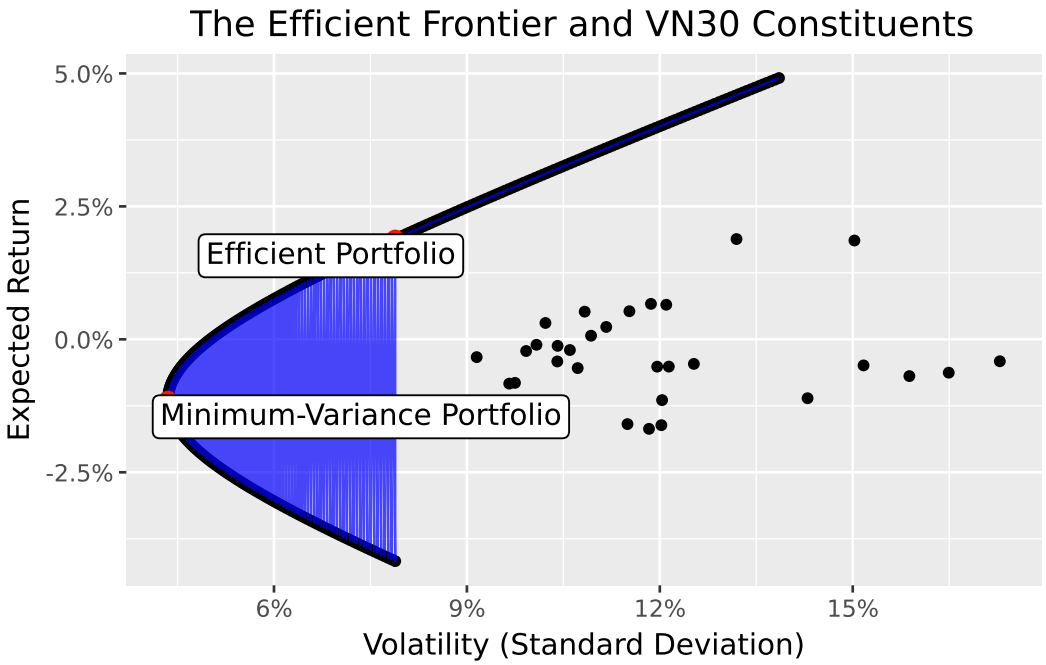


Figure 3.5: The big dots indicate the location of the minimum-variance and the efficient portfolio. The small dots indicate the location of the individual constituents.

The efficient frontier has a characteristic hyperbolic shape. The leftmost point is the minimum-variance portfolio. Moving up and to the right along the frontier, expected return increases but so does volatility. The upper portion of the hyperbola (above the minimum-variance portfolio) is the "efficient" part—these portfolios offer the highest return for each level of risk. The lower portion is "inefficient"—these portfolios are dominated by their mirror images on the upper portion.

The figure also reveals how dramatically diversification improves upon individual stock positions. Nearly all individual stocks lie well inside the efficient frontier, meaning investors can achieve the same expected return with much less risk, or much higher expected return with the same risk, simply by diversifying.

3.6 Key Takeaways

This chapter introduced the concepts of Modern Portfolio Theory. The main insights are:

1. **Portfolio risk depends on correlations**: The variance of a portfolio is not simply the weighted average of individual variances. Covariances between assets play a crucial role, creating opportunities for diversification.
2. **Diversification is the "only free lunch" in finance**: By combining assets that don't move perfectly together, investors can reduce risk without sacrificing expected return. This insight is the cornerstone of modern investment practice.
3. **The minimum-variance portfolio minimizes risk**: This portfolio depends only on the covariance matrix and serves as an important benchmark. It represents the least risky way to be fully invested in risky assets.
4. **Efficient portfolios balance risk and return**: By accepting more variance, investors can earn higher expected returns. Efficient portfolios are those that offer the best possible trade-off.
5. **The efficient frontier characterizes optimal portfolios**: This boundary in mean-standard deviation space represents the menu of optimal choices available to mean-variance investors.
6. **Two-fund separation simplifies implementation**: Any efficient portfolio can be constructed from any two distinct efficient portfolios, reducing the computational burden of portfolio optimization.

Looking ahead, several important complications arise in practice. First, the inputs to portfolio optimization (expected returns and covariances) must be estimated from data, and estimation error can dramatically affect portfolio performance. Second, real-world constraints such as transaction costs, short-sale restrictions, and position limits modify the optimization problem. Third, the assumption that investors care only about mean and variance may be too restrictive when returns are non-normal or when investors have more complex preferences. We address these extensions in subsequent chapters.

4 The Capital Asset Pricing Model

4.1 From Efficient Portfolios to Equilibrium Prices

The previous chapter on Modern Portfolio Theory (MPT) showed how an investor can construct portfolios that optimally trade off risk and expected return. But MPT leaves a crucial question unanswered: What determines the *expected returns* themselves? Why do some assets command higher risk premiums than others?

The Capital Asset Pricing Model (CAPM) answers this question. Developed simultaneously by Sharpe (1964), Lintner (1965), and Mossin (1966), the CAPM extends MPT to explain how assets should be priced in equilibrium when *all* investors follow mean-variance optimization principles. The CAPM's central insight is both elegant and counterintuitive: **not all risk is rewarded**. Only the component of risk that cannot be diversified away (i.e., systematic risk) commands a risk premium in equilibrium.

The CAPM remains the cornerstone of asset pricing theory, not because it perfectly describes reality, but because it provides the simplest coherent framework for understanding the relationship between risk and expected return. Every extension and alternative model in asset pricing (e.g., from the Fama-French factors to consumption-based pricing) builds upon or reacts against the CAPM's foundational logic.

In this chapter, we derive the CAPM from first principles, illustrate its theoretical underpinnings, and show how to estimate its parameters empirically. We download stock market data, estimate betas using regression analysis, and evaluate asset performance relative to model predictions.

```
import pandas as pd
import numpy as np
import tidyfinance as tf

from plotnine import *
from mizani.formatters import percent_format
from adjustText import adjust_text
```

4.2 Systematic versus Idiosyncratic Risk

Before diving into the mathematics, we need to understand the fundamental distinction that makes the CAPM work: the difference between *systematic* and *idiosyncratic* risk.

4.2.1 Idiosyncratic Risk: Diversifiable and Unrewarded

Consider events that affect individual companies but not the broader market: a CEO resigns unexpectedly, a product launch fails, earnings disappoint analysts, or a factory experiences a fire. These company-specific events can dramatically affect individual stock prices, but they tend to average out across a diversified portfolio. When one company has bad news, another often has good news; the shocks are largely uncorrelated.

This idiosyncratic (or firm-specific) risk can be eliminated through diversification. By holding a portfolio of many stocks, an investor can reduce idiosyncratic risk to nearly zero. Since this risk can be avoided at no cost, investors should not expect compensation for bearing it. In equilibrium, idiosyncratic risk earns no premium.

4.2.2 Systematic Risk: Undiversifiable and Priced

Systematic risk, by contrast, affects all assets simultaneously. Recessions, interest rate changes, geopolitical crises, and pandemics impact virtually every company to some degree. No amount of diversification can eliminate exposure to these economy-wide shocks, they are inherent to participating in the market.

Since systematic risk cannot be diversified away, investors genuinely dislike it. They must be compensated for bearing it. The CAPM formalizes this intuition: expected returns should depend only on systematic risk, not total risk. Two assets with identical total volatility can have very different expected returns if their systematic risk exposures differ.

4.2.3 A Simple Illustration

Imagine two stocks with identical 30% annual volatility. Stock A is a gold mining company whose returns move opposite to the overall market: it does well when the economy struggles and poorly when it booms. Stock B is a luxury retailer that amplifies market movements: soaring in good times and crashing in bad times.

Which stock should offer higher expected returns? Intuition might suggest they should be equal since both have the same volatility. But the CAPM says Stock B should offer substantially higher returns. Why? Because Stock B performs poorly precisely when investors' overall wealth is already down (during market crashes), making its returns particularly painful. Stock A, by contrast, provides insurance. Its strong performance during market downturns partially offsets

losses elsewhere in the portfolio. Investors value this insurance property and are willing to accept lower expected returns in exchange.

This is the CAPM's core insight: expected returns compensate investors for systematic risk exposure, measured by how an asset's returns co-move with the market portfolio.

4.3 Data Preparation

Building on our analysis from the previous chapter, we examine the VN30 constituents as our asset universe. We download and prepare monthly return data:

```
vn30_symbols = [
    "ACB", "BCM", "BID", "BVH", "CTG", "FPT", "GAS", "GVR", "HDB", "HPG",
    "MBB", "MSN", "MWG", "PLX", "POW", "SAB", "SHB", "SSB", "STB", "TCB",
    "TPB", "VCB", "VHM", "VIB", "VIC", "VJC", "VNM", "VPB", "VRE", "EIB"
]
```

```
import os
import boto3
from botocore.client import Config
from io import BytesIO

class ConnectMinio:
    def __init__(self):
        self.MINIO_ENDPOINT = os.environ["MINIO_ENDPOINT"]
        self.MINIO_ACCESS_KEY = os.environ["MINIO_ACCESS_KEY"]
        self.MINIO_SECRET_KEY = os.environ["MINIO_SECRET_KEY"]
        self.REGION = os.getenv("MINIO_REGION", "us-east-1")

        self.s3 = boto3.client(
            "s3",
            endpoint_url=self.MINIO_ENDPOINT,
            aws_access_key_id=self.MINIO_ACCESS_KEY,
            aws_secret_access_key=self.MINIO_SECRET_KEY,
            region_name=self.REGION,
            config=Config(signature_version="s3v4"),
        )

    def test_connection(self):
        resp = self.s3.list_buckets()
        print("Connected. Buckets:")
```

```
        for b in resp.get("Buckets", []):
            print(" -", b["Name"])


conn = ConnectMinio()
s3 = conn.s3
conn.test_connection()

bucket_name = os.environ["MINIO_BUCKET"]

prices = pd.read_csv(
    BytesIO(
        s3.get_object(
            Bucket=bucket_name,
            Key="historycal_price/dataset_historical_price.csv"
        )["Body"].read()
    ),
    low_memory=False
)
```

```
Connected. Buckets:
 - dsteam-data
 - rawbctc
```

We process the raw price data to compute adjusted closing prices and standardize column names:

```
prices["date"] = pd.to_datetime(prices["date"])
prices["adjusted_close"] = prices["close_price"] * prices["adj_ratio"]
prices = prices.rename(columns={
    "vol_total": "volume",
    "open_price": "open",
    "low_price": "low",
    "high_price": "high",
    "close_price": "close"
})
prices = prices.sort_values(["symbol", "date"])
```

```
prices_daily = prices[prices["symbol"].isin(vn30_symbols)]
prices_daily[["date", "symbol", "adjusted_close"]].head(3)
```

	date	symbol	adjusted_close
18176	2010-01-04	ACB	329.408244
18177	2010-01-05	ACB	329.408244
18178	2010-01-06	ACB	320.258015

4.3.1 Computing Monthly Returns

We aggregate daily prices to monthly frequency. Using monthly returns rather than daily returns offers several advantages for portfolio analysis: monthly returns exhibit less noise, better approximate normality, and reduce the impact of microstructure effects like bid-ask bounce.

```
returns_monthly = (prices_daily
    .assign(
        date=prices_daily["date"].dt.to_period("M").dt.to_timestamp("M")
    )
    .groupby(["symbol", "date"], as_index=False)
    .agg(adjusted_close=("adjusted_close", "last"))
    .sort_values(["symbol", "date"])
    .assign(
        ret=lambda x: x.groupby("symbol")["adjusted_close"].pct_change()
    )
)

returns_monthly.head(3)
```

	symbol	date	adjusted_close	ret
0	ACB	2010-01-31	291.975489	NaN
1	ACB	2010-02-28	303.621235	0.039886
2	ACB	2010-03-31	273.784658	-0.098269

4.4 The Risk-Free Asset and the Investment Opportunity Set

4.4.1 Adding a Risk-Free Asset

The previous chapter on MPT considered portfolios composed entirely of risky assets, requiring that portfolio weights sum to one. The CAPM introduces a crucial new element: a **risk-free asset** that pays a constant interest rate r_f with zero volatility.

This seemingly simple addition fundamentally transforms the investment opportunity set. With a risk-free asset available, investors can choose to park some wealth in the safe asset and invest the remainder in risky assets. They can also borrow at the risk-free rate to leverage their risky positions.

Let $\omega \in \mathbb{R}^N$ denote the portfolio weights in the N risky assets. Unlike before, these weights need not sum to one. The remainder, $1 - \iota'\omega$ (where ι is a vector of ones), is invested in the risk-free asset.

4.4.2 Portfolio Return with a Risk-Free Asset

The expected return on this combined portfolio is:

$$\mu_\omega = \omega'\mu + (1 - \iota'\omega)r_f = r_f + \omega'(\mu - r_f) = r_f + \omega'\tilde{\mu}$$

where μ is the vector of expected returns on risky assets and $\tilde{\mu} = \mu - r_f$ denotes the vector of **excess returns** (returns above the risk-free rate).

This expression reveals an important decomposition: the portfolio's expected return equals the risk-free rate plus a risk premium determined by the exposure to risky assets.

4.4.3 Portfolio Variance

Since the risk-free asset has zero volatility and zero covariance with risky assets, only the risky portion contributes to portfolio variance:

$$\sigma_\omega^2 = \omega'\Sigma\omega$$

where Σ is the variance-covariance matrix of risky asset returns. The portfolio's volatility (standard deviation) is:

$$\sigma_\omega = \sqrt{\omega'\Sigma\omega}$$

4.4.4 Setting Up the Risk-Free Rate

For a realistic proxy of the risk-free rate, we use the Vietnam government bond yield. Government bonds of stable economies are considered "risk-free" because the government can always print money to meet its obligations (though this may cause inflation).

```
all_dates = pd.date_range(
    start=returns_monthly["date"].min(),
    end=returns_monthly["date"].max(),
    freq="ME"
)

# Vietnam 10-Year Government Bond Yield (approximately 2.52% annualized)
rf_annual = 0.0252
rf_monthly_val = (1 + rf_annual)**(1/12) - 1

risk_free_monthly = pd.DataFrame({
    "date": all_dates,
    "risk_free": rf_monthly_val
})

risk_free_monthly["date"] = (
    pd.to_datetime(risk_free_monthly["date"])
    .dt.to_period("M")
    .dt.to_timestamp("M")
)

risk_free_monthly.head(3)
```

	date	risk_free
0	2010-01-31	0.002076
1	2010-02-28	0.002076
2	2010-03-31	0.002076

We merge the risk-free rate with our returns data and compute excess returns:

```
returns_monthly = returns_monthly.merge(
    risk_free_monthly[["date", "risk_free"]],
    on="date",
    how="left"
)

rf = risk_free_monthly["risk_free"].mean()

returns_monthly = (returns_monthly
    .assign(
```

```
        ret_excess=lambda x: x["ret"] - x["risk_free"]
    )
    .assign(
        ret_excess=lambda x: x["ret_excess"].clip(lower=-1)
    )
)

returns_monthly.head(3)
```

	symbol	date	adjusted_close	ret	risk_free	ret_excess
0	ACB	2010-01-31	291.975489	NaN	0.002076	NaN
1	ACB	2010-02-28	303.621235	0.039886	0.002076	0.037810
2	ACB	2010-03-31	273.784658	-0.098269	0.002076	-0.100345

4.5 The Tangency Portfolio: Where Everyone Invests

4.5.1 Deriving the Optimal Risky Portfolio

With a risk-free asset available, how should an investor allocate wealth across risky assets? Consider an investor who wants to achieve a target expected excess return $\bar{\mu}$ with minimum variance. The optimization problem becomes:

$$\min_{\omega} \omega' \Sigma \omega \quad \text{subject to} \quad \omega' \tilde{\mu} = \bar{\mu}$$

Using the Lagrangian method:

$$\mathcal{L}(\omega, \lambda) = \omega' \Sigma \omega - \lambda(\omega' \tilde{\mu} - \bar{\mu})$$

The first-order condition with respect to ω is:

$$\frac{\partial \mathcal{L}}{\partial \omega} = 2\Sigma\omega - \lambda\tilde{\mu} = 0$$

Solving for the optimal weights:

$$\omega^* = \frac{\lambda}{2}\Sigma^{-1}\tilde{\mu}$$

The constraint $\omega'\tilde{\mu} = \bar{\mu}$ determines λ:

$$\bar{\mu} = \tilde{\mu}'\omega^* = \frac{\lambda}{2}\tilde{\mu}'\Sigma^{-1}\tilde{\mu} \implies \lambda = \frac{2\bar{\mu}}{\tilde{\mu}'\Sigma^{-1}\tilde{\mu}}$$

Substituting back:

$$\omega^* = \frac{\bar{\mu}}{\tilde{\mu}'\Sigma^{-1}\tilde{\mu}}\Sigma^{-1}\tilde{\mu}$$

4.5.2 The Tangency Portfolio

Notice something remarkable: the *direction* of ω^* is always $\Sigma^{-1}\tilde{\mu}$, regardless of the target return $\bar{\mu}$. Only the *scale* changes. This means all investors, regardless of their risk preferences, hold the *same portfolio of risky assets*. They differ only in how much they allocate to this portfolio versus the risk-free asset.

To obtain the portfolio of risky assets that is fully invested (weights summing to one), we normalize:

$$\omega_{\tan} = \frac{\omega^*}{\iota'\omega^*} = \frac{\Sigma^{-1}(\mu - r_f)}{\iota'\Sigma^{-1}(\mu - r_f)}$$

This is called the **tangency portfolio** (or maximum Sharpe ratio portfolio) because it lies at the point where the efficient frontier is tangent to the capital market line.

4.5.3 The Sharpe Ratio and the Capital Market Line

The **Sharpe ratio** measures excess return per unit of volatility:

$$\text{Sharpe Ratio} = \frac{\mu_p - r_f}{\sigma_p}$$

The tangency portfolio maximizes the Sharpe ratio among all possible portfolios. Any combination of the risk-free asset and the tangency portfolio lies on a straight line in mean-standard deviation space, called the **Capital Market Line (CML)**:

$$\mu_p = r_f + \left(\frac{\mu_{\tan} - r_f}{\sigma_{\tan}}\right)\sigma_p$$

The slope of this line equals the Sharpe ratio of the tangency portfolio (i.e., the highest achievable Sharpe ratio).

4.5.4 Computing the Tangency Portfolio

Let's compute the tangency portfolio for our VN30 universe:

```
assets = (returns_monthly
    .groupby("symbol", as_index=False)
    .agg(
        mu=("ret", "mean"),
        sigma=("ret", "std")
    )
)

sigma = (returns_monthly
    .pivot(index="date", columns="symbol", values="ret")
    .cov()
)

mu = (returns_monthly
    .groupby("symbol")["ret"]
    .mean()
    .values
)
```

```
# Compute tangency portfolio weights
w_tan = np.linalg.solve(sigma, mu - rf)
w_tan = w_tan / np.sum(w_tan)

# Portfolio performance metrics
mu_w = w_tan.T @ mu
sigma_w = np.sqrt(w_tan.T @ sigma @ w_tan)

efficient_portfolios = pd.DataFrame([
    {"symbol": r"$\omega_{\mathrm{tan}}$", "mu": mu_w, "sigma": sigma_w},
    {"symbol": r"$r_f$", "mu": rf, "sigma": 0}
])

sharpe_ratio = (mu_w - rf) / sigma_w

print(f"Tangency Portfolio Sharpe Ratio: {sharpe_ratio:.4f}")
print(efficient_portfolios)
```

```
Tangency Portfolio Sharpe Ratio: -0.5552
```

```
                    symbol        mu     sigma
0  $\omega_{\mathrm{tan}}$ -0.041157  0.077866
1                      $r_f$  0.002076  0.000000
```

4.5.5 Visualizing the Efficient Frontier with a Risk-Free Asset

Figure 4.1 shows the efficient frontier when a risk-free asset is available. The frontier is now a straight line (the Capital Market Line) connecting the risk-free asset to the tangency portfolio and extending beyond.

```
efficient_portfolios_figure = (
    ggplot(efficient_portfolios, aes(x="sigma", y="mu"))
    + geom_point(data=assets)
    + geom_point(data=efficient_portfolios, color="blue", size=3)
    + geom_label(
        aes(label="symbol"),
        adjust_text={"arrowprops": {"arrowstyle": "-"}}
    )
    + scale_x_continuous(labels=percent_format())
    + scale_y_continuous(labels=percent_format())
    + labs(
        x="Volatility (Standard Deviation)",
        y="Expected Return",
        title="Efficient Frontier with Risk-Free Asset (VN30)"
    )
    + geom_abline(slope=sharpe_ratio, intercept=rf, linetype="dotted")
)

efficient_portfolios_figure.show()
```

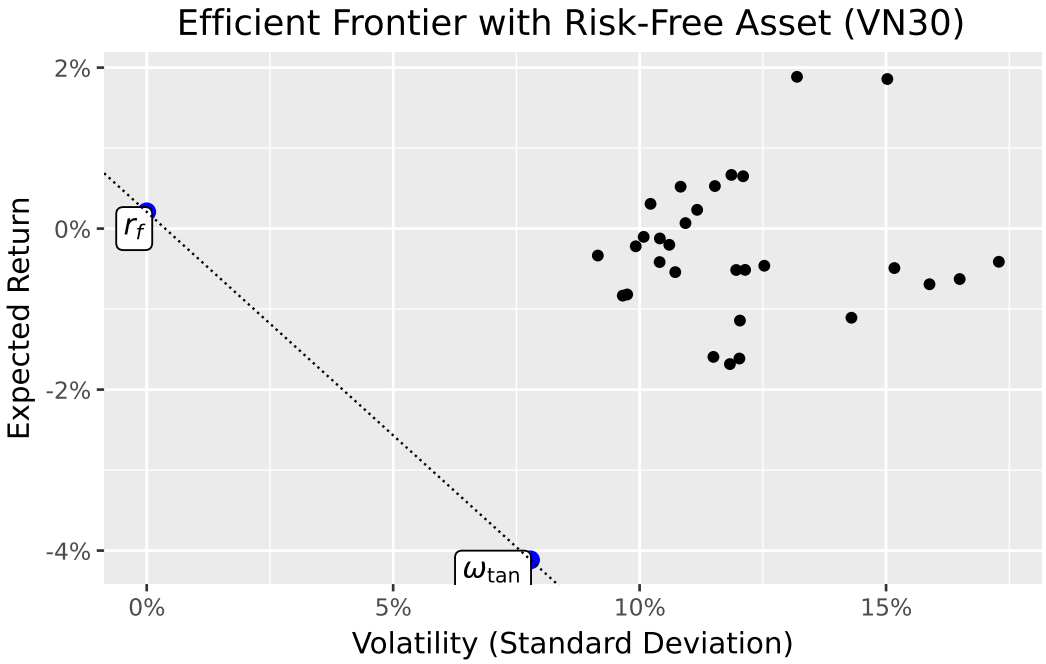


Figure 4.1: The efficient frontier with a risk-free asset becomes a straight line (the Capital Market Line) connecting the risk-free rate to the tangency portfolio. Individual assets lie below this line, demonstrating the benefits of diversification.

You may notice that estimated expected returns appear quite low, some even negative. This is not a model failure but reflects the realities of estimation:

1. **Sample period matters**: If the estimation window includes market downturns (such as the 2022-2023 period), realized average returns can be near zero or negative. Mean-variance optimization takes sample means literally.
2. **Estimation noise in emerging markets**: With volatile emerging market data, sample means are dominated by noise. A few extremely bad months can push the average below the risk-free rate even if the long-run equity premium is positive.

This highlights a fundamental challenge in portfolio optimization: the inputs we observe (historical returns) are noisy estimates of the true parameters we need (expected future returns).

4.6 The CAPM Equation: Risk and Expected Return

4.6.1 From Individual Optimization to Market Equilibrium

So far, we've focused on one investor's optimization problem. The CAPM's power comes from considering what happens when *all* investors optimize simultaneously.

If all investors follow mean-variance optimization, they all hold some combination of the risk-free asset and the tangency portfolio. The only difference between investors is their risk tolerance. More risk-averse investors hold more of the risk-free asset, while risk-tolerant investors may even borrow at the risk-free rate to leverage their position in the tangency portfolio.

4.6.2 The Market Portfolio

In equilibrium, the total demand for each risky asset must equal its supply. Since all investors hold the same portfolio of risky assets (the tangency portfolio), the equilibrium portfolio weights must equal the *market capitalization weights*. The tangency portfolio *is* the market portfolio.

This insight has enormous practical implications: instead of estimating expected returns and covariances to compute the tangency portfolio, we can simply use the market portfolio (approximated by a broad market index) as a proxy.

4.6.3 Deriving the CAPM Equation

From the first-order conditions of the optimization problem, we derived that:

$$\tilde{\mu} = \frac{2}{\lambda}\Sigma\omega^*$$

Since ω^* is proportional to $\omega_{\tan}$, and in equilibrium $\omega_{\tan}$ equals the market portfolio ω_m:

$$\tilde{\mu} = c \cdot \Sigma\omega_m$$

for some constant c. The i-th element of $\Sigma\omega_m$ is:

$$\sum_{j=1}^{N} \sigma_{ij}\omega_{m,j} = \mathrm{Cov}(r_i, r_m)$$

where $r_m = \sum_j \omega_{m,j} r_j$ is the return on the market portfolio.

For the market portfolio itself:

$$\tilde{\mu}_m = c \cdot \mathrm{Var}(r_m) = c \cdot \sigma_m^2$$

Therefore $c = \tilde{\mu}_m / \sigma_m^2$, and for any asset i:

$$\tilde{\mu}_i = \frac{\tilde{\mu}_m}{\sigma_m^2} \mathrm{Cov}(r_i, r_m) = \beta_i \tilde{\mu}_m$$

where:

$$\beta_i = \frac{\mathrm{Cov}(r_i, r_m)}{\mathrm{Var}(r_m)}$$

This is the famous **CAPM equation**:

$$E(r_i) - r_f = \beta_i [E(r_m) - r_f]$$

4.6.4 Interpreting Beta

Beta (β_i) measures an asset's **systematic risk** (i.e., its sensitivity to market movements). The interpretation is straightforward:

- $\beta = 1$: The asset moves one-for-one with the market (average systematic risk)
- $\beta > 1$: The asset amplifies market movements (aggressive, high systematic risk)
- $\beta < 1$: The asset dampens market movements (defensive, low systematic risk)
- $\beta < 0$: The asset moves opposite to the market (provides insurance)

The CAPM says that expected excess return is proportional to beta, not to total volatility. This explains why:

1. An asset with zero beta earns only the risk-free rate (i.e., its risk is entirely idiosyncratic).
2. An asset with beta of 1 earns the market risk premium
3. A negative-beta asset earns *less* than the risk-free rate (i.e., investors pay for its insurance properties).

4.7 The Security Market Line

The CAPM predicts a linear relationship between beta and expected return. This relationship is called the **Security Market Line (SML)**:

$$E(r_i) = r_f + \beta_i[E(r_m) - r_f]$$

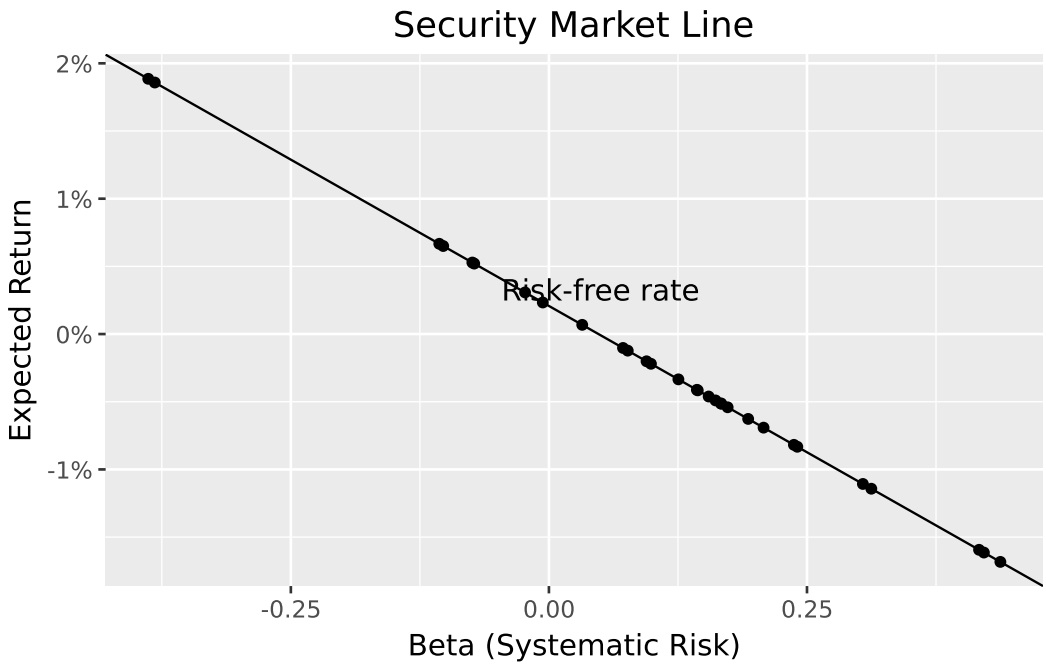
Unlike the Capital Market Line (which plots expected return against total risk), the Security Market Line plots expected return against systematic risk (beta).

```
betas = (sigma @ w_tan) / (w_tan.T @ sigma @ w_tan)
assets["beta"] = betas.values

price_of_risk = float(w_tan.T @ mu - rf)

assets_figure = (
    ggplot(assets, aes(x="beta", y="mu"))
    + geom_point()
    + geom_abline(intercept=rf, slope=price_of_risk)
    + scale_y_continuous(labels=percent_format())
    + labs(
        x="Beta (Systematic Risk)",
        y="Expected Return",
        title="Security Market Line"
    )
    + annotate("text", x=0.05, y=rf + 0.001, label="Risk-free rate")
)

assets_figure.show()
```



You may observe that the estimated SML has a negative slope, which seems to contradict CAPM's prediction. This reflects a **negative estimated market risk premium** in our sample period (i.e., the market portfolio earned less than the risk-free rate).

When the market risk premium is negative, CAPM predicts that high-beta stocks should have *lower* expected returns than low-beta stocks. This is not a model failure, the model is behaving consistently. Rather, it reflects an unusual (but not impossible) sample period where risky assets underperformed safe assets.

This observation highlights an important distinction: CAPM describes *expected* returns in equilibrium, but *realized* returns over any particular period may differ substantially from expectations due to shocks and surprises.

4.8 Empirical Estimation of CAPM Parameters

4.8.1 The Regression Framework

In practice, we estimate CAPM parameters using time-series regression. The model implies:

$$r_{i,t} - r_{f,t} = \alpha_i + \beta_i(r_{m,t} - r_{f,t}) + \varepsilon_{i,t}$$

where:

- $r_{i,t}$: Return on asset i at time t
- $r_{f,t}$: Risk-free rate at time t
- $r_{m,t}$: Market return at time t
- α_i: Intercept (should be zero if CAPM holds)
- β_i: Systematic risk (slope coefficient)
- $\varepsilon_{i,t}$: Idiosyncratic shock (residual)

4.8.2 Alpha: Risk-Adjusted Performance

The intercept α_i measures **risk-adjusted performance**. If CAPM holds perfectly, alpha should be zero for all assets (i.e., any excess return is exactly compensated by systematic risk).

- $\alpha > 0$: The asset outperformed its CAPM-predicted return (positive abnormal return)
- $\alpha < 0$: The asset underperformed its CAPM-predicted return (negative abnormal return)

Positive alpha is the holy grail of active management: earning returns beyond what systematic risk exposure would justify.

4.8.3 Loading Factor Data

We use Fama-French market excess returns as our market portfolio proxy. These data provide a widely accepted benchmark that is already adjusted for the risk-free rate:

```
import sqlite3

tidy_finance = sqlite3.connect(database="data/tidy_finance_python.sqlite")

factors = pd.read_sql_query(
    sql="SELECT * FROM factors_ff5_monthly",
    con=tidy_finance,
    parse_dates={"date"}
)

factors.head(3)
```

	date	smb	hml	rmw	cma	mkt_excess
0	2011-07-31	0.029280	-0.049915	0.013239	-0.025244	-0.067002
1	2011-08-31	0.022241	-0.021097	0.001543	-0.021929	0.049073
2	2011-09-30	0.026609	0.014323	0.025125	0.003444	-0.017362

4.8.4 Running the Regressions

We estimate CAPM regressions for each stock in our universe:

```
import statsmodels.formula.api as smf

returns_excess_monthly = (returns_monthly
    .merge(factors, on="date", how="left")
    .assign(ret_excess=lambda x: x["ret"] - x["risk_free"])
)


def estimate_capm(data):
    model = smf.ols("ret_excess ~ mkt_excess", data=data).fit()
    result = pd.DataFrame({
        "coefficient": ["alpha", "beta"],
        "estimate": model.params.values,
        "t_statistic": model.tvalues.values
    })
    return result


capm_results = (returns_excess_monthly
    .groupby("symbol", group_keys=True)
    .apply(estimate_capm)
    .reset_index()
)

capm_results.head(4)
```

	symbol	level_1	coefficient	estimate	t_statistic
0	ACB	0	alpha	-0.000677	-0.086189
1	ACB	1	beta	0.611184	4.518154
2	BCM	0	alpha	0.035585	2.410328
3	BCM	1	beta	1.062267	4.666808

4.8.5 Visualizing Alpha Estimates

Figure 4.2 shows the estimated alphas across our VN30 sample. Statistical significance (at the 95% level) is indicated by color.

```
alphas = (capm_results
    .query("coefficient == 'alpha'")
    .assign(is_significant=lambda x: np.abs(x["t_statistic"]) >= 1.96)
)

alphas["symbol"] = pd.Categorical(
    alphas["symbol"],
    categories=alphas.sort_values("estimate")["symbol"],
    ordered=True
)

alphas_figure = (
    ggplot(alphas, aes(y="estimate", x="symbol", fill="is_significant"))
    + geom_col()
    + scale_y_continuous(labels=percent_format())
    + coord_flip()
    + labs(
        x="",
        y="Estimated Alpha (Monthly)",
        fill="Significant at 95%?",
        title="Estimated CAPM Alphas for VN30 Index Constituents"
    )
)

alphas_figure.show()
```

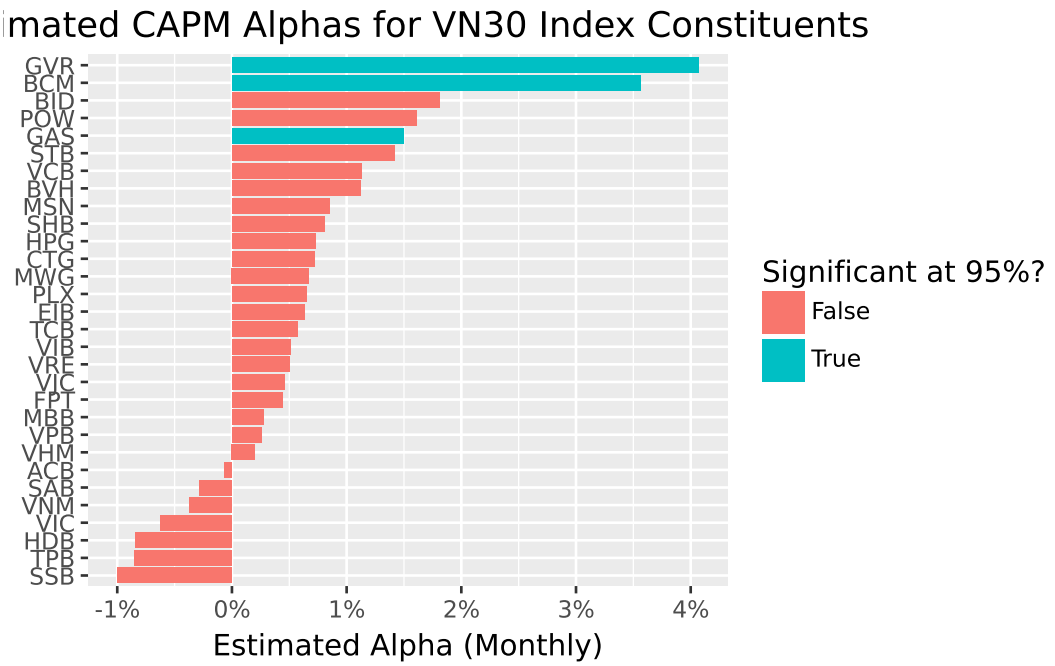


Figure 4.2: Estimated CAPM alphas for VN30 index constituents. Color indicates statistical significance at the 95% confidence level. Most alphas are statistically indistinguishable from zero, consistent with CAPM predictions.

The distribution of alphas provides evidence on CAPM's empirical validity. If the model holds, we expect:

1. Most alphas close to zero
2. Few statistically significant alphas
3. Roughly equal numbers of positive and negative alphas

Systematic patterns in alphas, such as consistently positive alphas for certain types of stocks, would suggest the CAPM is incomplete and that additional risk factors may be needed.

4.9 Limitations and Extensions

4.9.1 The Market Portfolio Problem

A fundamental challenge in testing the CAPM is identifying the market portfolio. The theory requires a portfolio that includes *all* investable assets, not just stocks, but also bonds, real estate, private businesses, human capital, and even intangible assets. In practice, we use proxies like broad market indices (VNI, S&P 500), but these capture only publicly traded equities.

This limitation is profound. As Richard Roll famously argued, the CAPM is essentially untestable because the true market portfolio is unobservable. Any test of the CAPM is simultaneously a test of whether our proxy adequately represents the market.

4.9.2 Time-Varying Betas

The CAPM assumes that betas are constant over time, but this assumption rarely holds in practice. Companies undergo changes that affect their market sensitivity:

- **Capital structure changes**: Increasing leverage raises beta
- **Business model evolution**: Diversification into new industries can alter systematic risk
- **Market conditions**: Betas often increase during market stress

Conditional CAPM models (Jagannathan and Wang 1996) address this by allowing risk premiums and betas to vary with the business cycle.

4.9.3 Empirical Anomalies

Decades of empirical research have documented patterns in stock returns that CAPM cannot explain:

1. **Size effect**: Small-cap stocks tend to outperform large-cap stocks, even after adjusting for beta
2. **Value effect**: Stocks with high book-to-market ratios outperform growth stocks
3. **Momentum**: Stocks that performed well recently tend to continue performing well

These anomalies suggest that systematic risk has multiple dimensions beyond market exposure.

4.9.4 Multifactor Extensions

The limitations of CAPM have led to increasingly sophisticated asset pricing models. The **Fama-French three-factor model** (Fama and French 1992) adds two factors to capture size and value effects:

- **SMB (Small Minus Big)**: Returns on small stocks minus large stocks
- **HML (High Minus Low)**: Returns on value stocks minus growth stocks

The **Fama-French five-factor model** (Fama and French 2015) adds two more dimensions:

- **RMW (Robust Minus Weak)**: Returns on profitable firms minus unprofitable firms

- **CMA (Conservative Minus Aggressive)**: Returns on conservative investors minus aggressive investors

The **Carhart four-factor model** (Carhart 1997) adds momentum to the three-factor framework.

Other theoretical developments include:

- **Consumption CAPM**: Links asset prices to macroeconomic consumption risk
- **Q-factor model** (Hou, Xue, and Zhang 2014): Derives factors from investment-based asset pricing theory
- **Arbitrage Pricing Theory**: Allows for multiple sources of systematic risk without specifying their identity

Despite its limitations, the CAPM remains valuable as a conceptual benchmark. Its core insight (i.e., only systematic, undiversifiable risk commands a premium) continues to inform how we think about risk and return.

4.10 Key Takeaways

This chapter introduced the Capital Asset Pricing Model and its implications for understanding the relationship between risk and expected return. The main insights are:

1. **Not all risk is rewarded**: The CAPM distinguishes between systematic risk (which cannot be diversified away and commands a premium) and idiosyncratic risk (which can be eliminated through diversification and earns no premium).

2. **The tangency portfolio is universal**: When a risk-free asset exists, all mean-variance investors hold the same portfolio of risky assets (i.e., the tangency or maximum Sharpe ratio portfolio). They differ only in how much they allocate to this portfolio versus the risk-free asset.

3. **In equilibrium, the tangency portfolio is the market portfolio**: Since all investors hold the same risky portfolio, and total demand must equal supply, the equilibrium portfolio weights are market capitalization weights.

4. **Expected returns depend on beta**: The CAPM equation states that expected excess return equals beta times the market risk premium. Beta measures covariance with the market portfolio, normalized by market variance.

5. **Alpha measures risk-adjusted performance**: Positive alpha indicates returns above what systematic risk would justify; negative alpha indicates underperformance.

6. **Empirical challenges exist**: Testing the CAPM requires identifying the market portfolio, which is unobservable in practice. Documented anomalies (size, value, momentum) suggest additional risk factors beyond market exposure.

7. **Extensions abound**: Multifactor models like Fama-French extend the CAPM framework by adding factors that capture dimensions of systematic risk the market factor misses.

The CAPM's elegance lies in its simplicity: a single factor (i.e., exposure to the market) should explain expected returns in equilibrium. While reality is more complex, this framework provides the foundation for all modern asset pricing theory.

5 Financial Statement Analysis

5.1 From Market Prices to Fundamental Value

The previous chapters focused on how financial markets price assets in equilibrium. The Capital Asset Pricing Model showed that expected returns depend on systematic risk exposure, while Modern Portfolio Theory demonstrated how to construct efficient portfolios. But these frameworks take expected returns and risk as given, they don't explain where these expectations come from.

Financial statement analysis addresses this gap. By examining a company's accounting records, investors can form independent assessments of firm value, identify mispriced securities, and understand the economic forces driving business performance. Financial statements provide the primary source of standardized information about a company's operations, financial position, and cash generation. Their legal requirements and standardized formats make them particularly valuable. Every publicly traded company must file them, creating a level playing field for analysis.

This chapter introduces the three primary financial statements: the balance sheet, income statement, and cash flow statement. We then demonstrate how to transform raw accounting data into meaningful financial ratios that facilitate comparison across companies and over time. These ratios serve multiple purposes: they enable investors to benchmark companies against peers, help creditors assess default risk, and provide inputs for asset pricing models like the Fama-French factors we will encounter in later chapters.

Our analysis combines theoretical frameworks with practical implementation using Vietnamese market data. By the end of this chapter, you will understand how to access financial statements, calculate key ratios across multiple categories, and interpret these metrics in context.

```
import pandas as pd
import numpy as np

from plotnine import *
from mizani.formatters import percent_format
from adjustText import adjust_text
```

5.2 The Three Financial Statements

Before diving into ratios and analysis, we need to understand the three interconnected statements that form the foundation of financial reporting. Each statement answers a different question about the company, and together they provide a comprehensive picture of financial health.

5.2.1 The Balance Sheet: A Snapshot of Financial Position

The balance sheet captures a company's financial position at a specific moment in time, think of it as a photograph rather than a movie. It lists everything the company owns (assets), everything it owes (liabilities), and the residual claim belonging to shareholders (equity). These three components are linked by the fundamental accounting equation:

$$\text{Assets} = \text{Liabilities} + \text{Equity}$$

This equation is not merely a definition, it reflects a core economic principle. A company's resources (assets) must be financed from somewhere: either borrowed from creditors (liabilities) or contributed by owners (equity). Every transaction affects both sides equally, maintaining the balance.

Assets represent resources the company controls that are expected to generate future economic benefits:

- **Current assets** can be converted to cash within one year: cash and equivalents, short-term investments, accounts receivable (money owed by customers), and inventory (raw materials, work-in-progress, and finished goods)
- **Non-current assets** support operations beyond one year: property, plant, and equipment (PP&E), long-term investments, and intangible assets like patents, trademarks, and goodwill

Liabilities encompass obligations to external parties:

- **Current liabilities** come due within one year: accounts payable (money owed to suppliers), short-term debt, accrued expenses, and the current portion of long-term debt
- **Non-current liabilities** extend beyond one year: long-term debt, bonds payable, pension obligations, and deferred tax liabilities

Shareholders' equity represents the owners' residual claim:

- **Common stock** and additional paid-in capital from share issuance
- **Retained earnings** (i.e., accumulated profits reinvested rather than distributed as dividends)
- **Treasury stock**: shares repurchased by the company

Understanding these categories is essential for ratio analysis. Current assets and liabilities determine short-term liquidity, while the mix of debt and equity reveals capital structure choices.

5.2.2 The Income Statement: Performance Over Time

While the balance sheet provides a snapshot, the income statement (also called the profit and loss statement, or P&L) measures financial performance over a period (e.g., a quarter or year). It follows a hierarchical structure that progressively captures different levels of profitability:

$$\text{Revenue} - \text{COGS} = \text{Gross Profit}$$

$$\text{Gross Profit} - \text{Operating Expenses} = \text{Operating Income (EBIT)}$$

$$\text{EBIT} - \text{Interest} - \text{Taxes} = \text{Net Income}$$

Each line reveals something different about the business:

- **Revenue (Sales)**: Total income from goods or services sold (i.e., the "top line")
- **Cost of Goods Sold (COGS)**: Direct costs of producing what was sold (materials, direct labor, manufacturing overhead)
- **Gross Profit**: Revenue minus COGS, measuring basic profitability from core operations
- **Operating Expenses**: Costs of running the business beyond production (selling, general & administrative expenses, research & development)
- **Operating Income (EBIT)**: Earnings Before Interest and Taxes, measuring profitability from operations before financing decisions and taxes
- **Interest Expense**: The cost of debt financing
- **Net Income**: The "bottom line" (i.e., total profit after all expenses)

The income statement's hierarchical structure allows analysts to identify where profitability problems originate. A company with strong gross margins but weak net income might have bloated overhead costs. One with weak gross margins faces fundamental pricing or production challenges.

5.2.3 The Cash Flow Statement: Following the Money

The cash flow statement bridges a critical gap: profitable companies can run out of cash, and unprofitable companies can generate positive cash flow. This happens because accrual accounting (used in the income statement) recognizes revenue when earned and expenses when incurred, not when cash changes hands.

The cash flow statement tracks actual cash movements, divided into three categories:

- **Operating activities**: Cash generated from core business operations. Starts with net income, then adjusts for non-cash items (depreciation, changes in working capital)
- **Investing activities**: Cash spent on or received from long-term investments (e.g., purchasing equipment, acquiring businesses, selling assets)
- **Financing activities**: Cash flows from capital structure decisions (e.g., issuing stock, borrowing, repaying debt, paying dividends, buying back shares)

A company can show strong net income while burning cash if it's building inventory, extending generous credit terms, or making large capital expenditures. Conversely, a company reporting losses might generate positive operating cash flow by collecting receivables faster than it pays suppliers.

5.2.4 Illustrating with FPT's Financial Statements

To see these concepts in practice, let's examine FPT Corporation's 2023 financial statements. FPT is one of Vietnam's largest technology companies, providing IT services, telecommunications, and education.

```
# Placeholder for FPT balance sheet visualization
# In practice, this would display the actual PDF or cleaned data
# from DataCore's acquisition pipeline

# Example structure of what the balance sheet data looks like:
# Assets: Current assets (cash, receivables, inventory) + Non-current assets (PP&E, intangibl
# Liabilities: Current liabilities (payables, short-term debt) + Non-current liabilities (lo
# Equity: Common stock + Retained earnings
```

The balance sheet demonstrates the fundamental accounting equation in action. FPT's assets (e.g., spanning cash, receivables, technology infrastructure, and intangible assets like software) exactly equal the sum of its liabilities and equity.

```
# Placeholder for FPT income statement visualization
# Shows the progression from revenue through various profit measures to net income
```

FPT's income statement reveals how the company transforms revenue into profit. The progression from gross profit through operating income to net income shows the impact of operating expenses, interest costs, and taxes.

```
# Placeholder for FPT cash flow statement visualization
# Reconciles net income with actual cash generation
```

The cash flow statement shows how FPT's reported profits translate into actual cash. Differences between net income and operating cash flow reveal the impact of working capital management and non-cash expenses.

5.3 Loading Financial Statement Data

We now turn to systematic analysis across multiple companies. We load financial statement data for the VN30 index constituents (i.e., the 30 largest and most liquid stocks on Vietnam's Ho Chi Minh Stock Exchange).

```
import sqlite3

tidy_finance = sqlite3.connect(database="data/tidy_finance_python.sqlite")

comp_vn = pd.read_sql_query(
    sql="SELECT * FROM comp_vn",
    con=tidy_finance,
    parse_dates={"datadate"}
)

comp_vn.head(3)
```

	symbol	year	total_current_asset	ca_fin	ca_cce	ca_cash	ca_cash_inbank	ca_cash
0	AGF	1998	8.845141e+10	None	5.469709e+09	0.000000e+00	None	None
1	BBC	1999	5.672574e+10	None	5.354939e+09	5.354939e+09	None	None
2	AGF	1999	9.558392e+10	None	2.609276e+09	0.000000e+00	None	None

```
vn30_symbols = [
    "ACB", "BCM", "BID", "BVH", "CTG", "FPT", "GAS", "GVR", "HDB", "HPG",
    "MBB", "MSN", "MWG", "PLX", "POW", "SAB", "SHB", "SSB", "STB", "TCB",
    "TPB", "VCB", "VHM", "VIB", "VIC", "VJC", "VNM", "VPB", "VRE", "EIB"
]
```

```
comp_vn30 = comp_vn[comp_vn["symbol"].isin(vn30_symbols)]
comp_vn30.head(3)
```

	symbol	year	total_current_asset	ca_fin	ca_cce	ca_cash	ca_cash_inbank	ca_ca
11	FPT	2002	5.098910e+11	None	1.027470e+11	0.000000e+00	None	None
17	VNM	2003	2.101406e+12	None	6.925924e+11	6.925924e+11	None	None
21	FPT	2003	9.171390e+11	None	7.995600e+10	0.000000e+00	None	None

This dataset provides the foundation for calculating financial ratios and conducting cross-sectional comparisons. Each row contains balance sheet, income statement, and cash flow items for a company-year observation.

5.4 Liquidity Ratios: Can the Company Pay Its Bills?

Liquidity ratios assess a company's ability to meet short-term obligations. These metrics matter most to creditors, suppliers, and employees who need assurance that the company can pay its bills. They're calculated using balance sheet items, comparing liquid assets against near-term liabilities.

5.4.1 The Current Ratio

The most basic liquidity measure compares all current assets to current liabilities:

$$\text{Current Ratio} = \frac{\text{Current Assets}}{\text{Current Liabilities}}$$

A ratio above one indicates the company has enough current assets to cover obligations due within one year. However, the interpretation depends heavily on the composition of current assets. A company with current assets tied up in slow-moving inventory is less liquid than one holding cash.

5.4.2 The Quick Ratio

The quick ratio (or "acid test") provides a more stringent measure by excluding inventory:

$$\text{Quick Ratio} = \frac{\text{Current Assets} - \text{Inventory}}{\text{Current Liabilities}}$$

Why exclude inventory? Inventory is typically the least liquid current asset. It must be sold (potentially at a discount) before generating cash. A company facing a liquidity crisis cannot easily convert raw materials or finished goods into immediate cash. The quick ratio answers: "Can we pay our bills without relying on inventory sales?"

5.4.3 The Cash Ratio

The most conservative liquidity measure focuses solely on the most liquid assets:

$$\text{Cash Ratio} = \frac{\text{Cash and Cash Equivalents}}{\text{Current Liabilities}}$$

While a ratio of one indicates robust liquidity, most companies maintain lower cash ratios to avoid holding excessive non-productive assets. Cash sitting in bank accounts could otherwise be invested in growth opportunities, returned to shareholders, or used to pay down costly debt.

5.4.4 Calculating Liquidity Ratios

Let's compute these ratios for our VN30 sample:

```
balance_sheet_statements = (comp_vn30
    .assign(
        fiscal_year=lambda x: x["year"].astype(int),

        # Current Ratio: Current Assets / Current Liabilities
        current_ratio=lambda x: x["act"] / x["lct"],

        # Quick Ratio: (Current Assets - Inventory) / Current Liabilities
        quick_ratio=lambda x: (x["act"] - x["inv"]) / x["lct"],

        # Cash Ratio: Cash and Equivalents / Current Liabilities
        cash_ratio=lambda x: x["ca_cce"] / x["lct"],

        label=lambda x: np.where(
```

```
            x["symbol"].isin(vn30_symbols), x["symbol"], np.nan
        )
    )
)

balance_sheet_statements.head(3)
```

	symbol	year	total_current_asset	ca_fin	ca_cce	ca_cash	ca_cash_inbank	ca_ca
11	FPT	2002	5.098910e+11	None	1.027470e+11	0.000000e+00	None	None
17	VNM	2003	2.101406e+12	None	6.925924e+11	6.925924e+11	None	None
21	FPT	2003	9.171390e+11	None	7.995600e+10	0.000000e+00	None	None

5.4.5 Cross-Sectional Comparison of Liquidity

Figure 5.1 compares liquidity ratios across companies for the most recent fiscal year. This cross-sectional view reveals how different business models and industries maintain different liquidity profiles.

```
liquidity_ratios = (balance_sheet_statements
    .query("year == 2023 & label.notna()")
    .get(["symbol", "current_ratio", "quick_ratio", "cash_ratio"])
    .melt(id_vars=["symbol"], var_name="name", value_name="value")
    .assign(
        name=lambda x: x["name"].str.replace("_", " ").str.title()
    )
)

liquidity_ratios_figure = (
    ggplot(liquidity_ratios, aes(y="value", x="name", fill="symbol"))
    + geom_col(position="dodge")
    + coord_flip()
    + labs(
        x="", y="Ratio Value", fill="",
        title="Liquidity Ratios for VN30 Stocks (2023)"
    )
)

liquidity_ratios_figure.show()
```

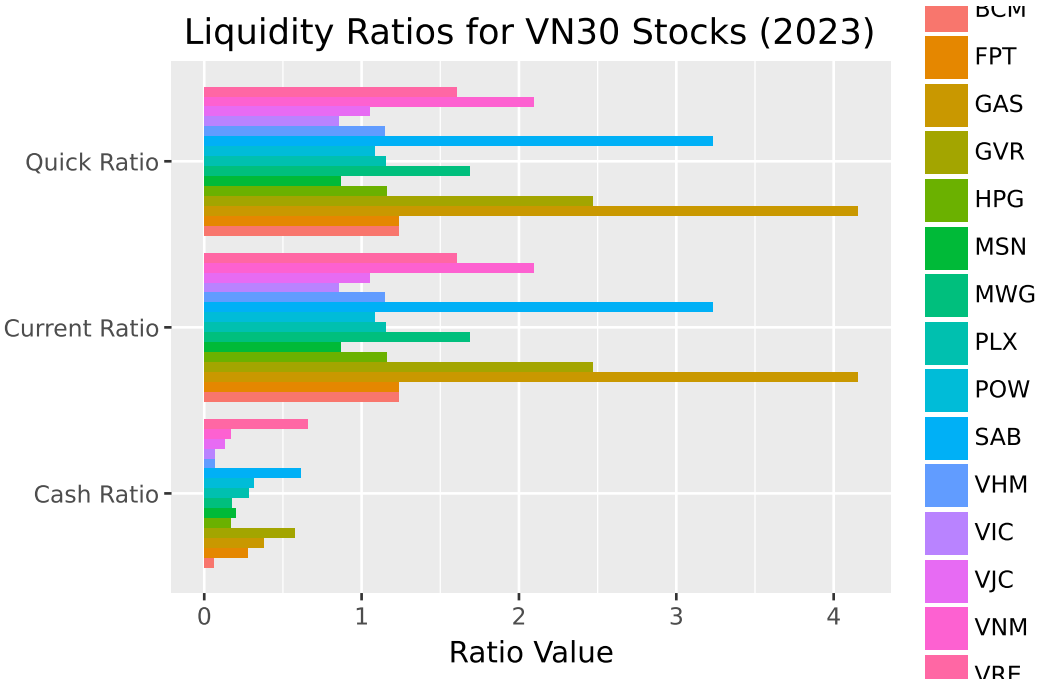


Figure 5.1: Liquidity ratios measure a company's ability to meet short-term obligations. Higher values indicate greater liquidity, though excessively high ratios may suggest inefficient use of assets.

Several patterns emerge from this comparison. Banks and financial institutions typically show different liquidity profiles than industrial companies due to their unique business models. Companies with high inventory (retailers, manufacturers) often show larger gaps between current and quick ratios.

5.5 Leverage Ratios: How Is the Company Financed?

Leverage ratios examine a company's capital structure (i.e., the mix of debt and equity financing). These metrics reveal financial risk and long-term solvency, helping investors understand how much of the company's operations are funded by borrowed money.

5.5.1 Why Capital Structure Matters

A company's financing choice involves fundamental trade-offs:

- **Debt** offers tax advantages (interest is deductible) and doesn't dilute ownership, but creates fixed obligations that must be met regardless of business performance

- **Equity** provides flexibility (no required payments) but dilutes existing shareholders and may be more expensive than debt

Companies with high leverage amplify both gains and losses. In good times, shareholders capture more upside because profits aren't shared with additional equity holders. In bad times, fixed interest payments can push the company toward distress. This is why beta (systematic risk) tends to increase with leverage.

5.5.2 Debt-to-Equity Ratio

This ratio indicates how much debt financing the company uses relative to shareholder investment:

$$\text{Debt-to-Equity} = \frac{\text{Total Debt}}{\text{Total Equity}}$$

A ratio of 1.0 means equal parts debt and equity financing. Higher ratios indicate more aggressive use of leverage, which can enhance returns in good times but increases bankruptcy risk.

5.5.3 Debt-to-Asset Ratio

This ratio shows what percentage of assets are financed through debt:

$$\text{Debt-to-Asset} = \frac{\text{Total Debt}}{\text{Total Assets}}$$

A ratio of 0.5 means half the company's assets are debt-financed. This metric is bounded between 0 and 1 (assuming positive equity), making it easier to compare across companies than the debt-to-equity ratio.

5.5.4 Interest Coverage Ratio

While the above ratios measure leverage levels, interest coverage assesses the ability to service that debt:

$$\text{Interest Coverage} = \frac{\text{EBIT}}{\text{Interest Expense}}$$

This ratio answers: "How many times over can current operating profits cover interest obligations?" A ratio below 1.0 means operating income doesn't cover interest payments, which is a dangerous position. Ratios above 3-5 generally indicate comfortable coverage.

5.5.5 Calculating Leverage Ratios

```
balance_sheet_statements = balance_sheet_statements.assign(
    debt_to_equity=lambda x: x["total_debt"] / x["total_equity"],
    debt_to_asset=lambda x: x["total_debt"] / x["at"]
)

income_statements = (comp_vn30
    .assign(
        year=lambda x: x["year"].astype(int),
        # Handle zero interest expense to avoid infinity
        interest_coverage=lambda x: np.where(
            x["cfo_interest_expense"] > 0,
            x["is_net_business_profit"] / x["cfo_interest_expense"],
            np.nan
        ),
        label=lambda x: np.where(
            x["symbol"].isin(vn30_symbols), x["symbol"], np.nan
        )
    )
)
```

5.5.6 Leverage Trends Over Time

Figure 5.2 tracks how debt-to-asset ratios have evolved over time. Time-series analysis reveals whether companies are becoming more or less leveraged.

```
debt_to_asset = balance_sheet_statements.query("symbol in @vn30_symbols")

debt_to_asset_figure = (
    ggplot(debt_to_asset, aes(x="year", y="debt_to_asset", color="symbol"))
    + geom_line(size=1)
    + scale_y_continuous(labels=percent_format())
    + labs(
        x="", y="Debt-to-Asset Ratio", color="",
        title="Debt-to-Asset Ratios of VN30 Stocks Over Time"
    )
)

debt_to_asset_figure.show()
```

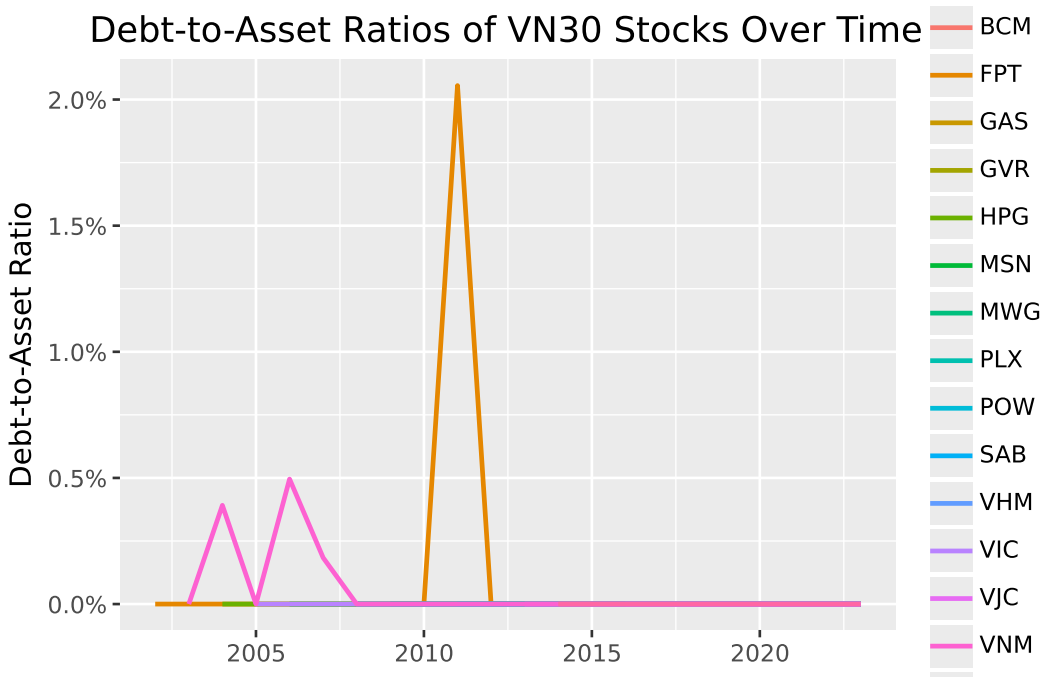


Figure 5.2: Debt-to-asset ratios show the proportion of assets financed by debt. Changes over time reflect evolving capital structure strategies and market conditions.

5.5.7 Cross-Sectional Leverage Comparison

Figure 5.3 provides a snapshot of leverage across all VN30 constituents for the most recent year.

```
debt_to_asset_comparison = balance_sheet_statements.query("year == 2023")

debt_to_asset_comparison["symbol"] = pd.Categorical(
    debt_to_asset_comparison["symbol"],
    categories=debt_to_asset_comparison.sort_values("debt_to_asset")["symbol"],
    ordered=True
)

debt_to_asset_comparison_figure = (
    ggplot(
        debt_to_asset_comparison,
        aes(y="debt_to_asset", x="symbol", fill="label")
    )
    + geom_col()
```

```
    + coord_flip()
    + scale_y_continuous(labels=percent_format())
    + labs(
        x="", y="Debt-to-Asset Ratio", fill="",
        title="Debt-to-Asset Ratios of VN30 Stocks (2023)"
    )
    + theme(legend_position="none")
)

debt_to_asset_comparison_figure.show()
```

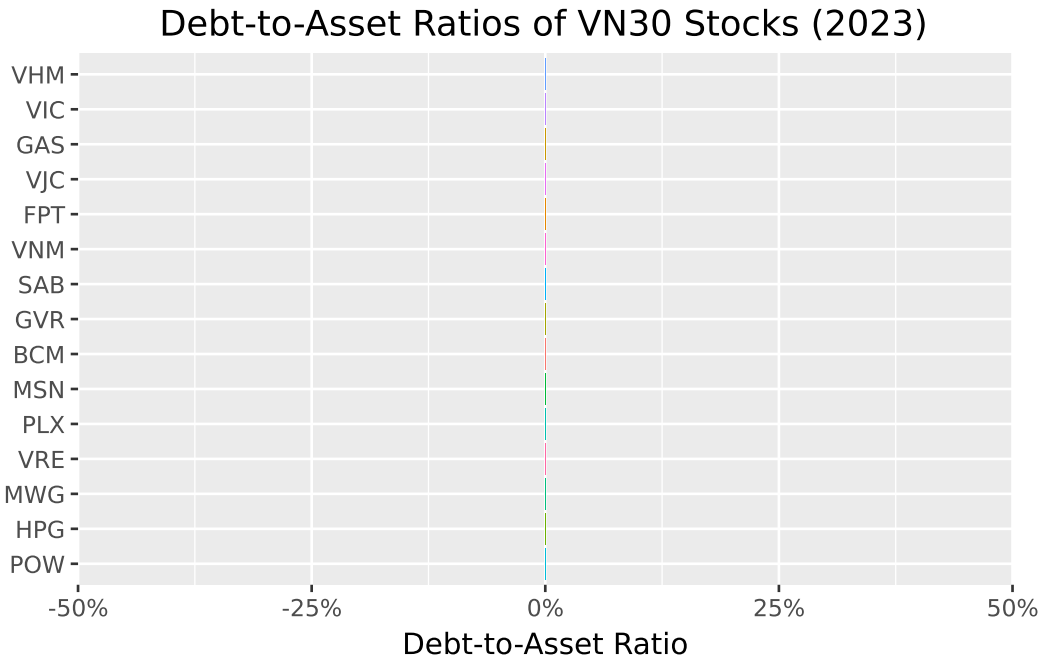


Figure 5.3: Cross-sectional comparison of debt-to-asset ratios reveals industry patterns and company-specific financing strategies.

5.5.8 The Leverage-Coverage Trade-off

Figure 5.4 examines the relationship between leverage levels and debt-servicing ability. Companies with higher debt loads should ideally have stronger interest coverage to maintain financial stability.

```
interest_coverage = (income_statements
    .query("year == 2023")
    .get(["symbol", "year", "interest_coverage"])
    .merge(balance_sheet_statements, on=["symbol", "year"], how="left")
)

interest_coverage_figure = (
    ggplot(
        interest_coverage,
        aes(x="debt_to_asset", y="interest_coverage", color="label")
    )
    + geom_point(size=2)
    + geom_label(
        aes(label="label"),
        adjust_text={"arrowprops": {"arrowstyle": "-"}}
    )
    + scale_x_continuous(labels=percent_format())
    + labs(
        x="Debt-to-Asset Ratio", y="Interest Coverage Ratio",
        title="Leverage versus Interest Coverage for VN30 Stocks (2023)"
    )
    + theme(legend_position="none")
)

interest_coverage_figure.show()
```

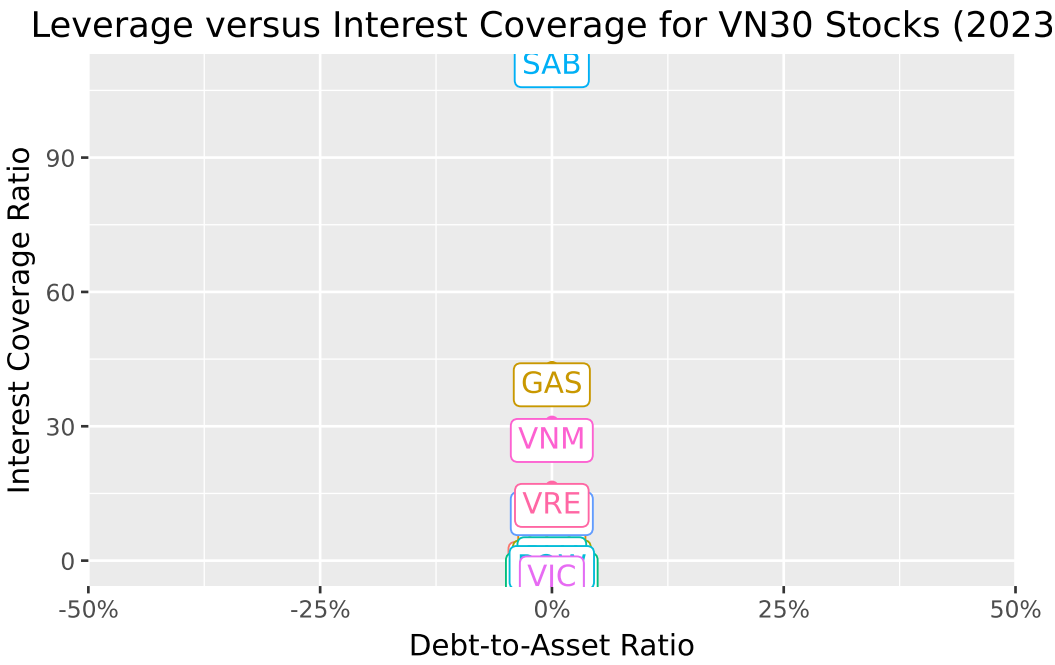


Figure 5.4: The relationship between leverage and interest coverage reveals whether companies can comfortably service their debt. High leverage with low coverage indicates elevated financial risk.

The scatter plot reveals important patterns. Companies in the upper-left quadrant (low leverage, high coverage) have conservative financing with ample debt capacity. Those in the lower-right (high leverage, low coverage) face elevated financial risk.

5.6 Efficiency Ratios: How Well Are Assets Managed?

Efficiency ratios measure how effectively a company utilizes its assets and manages operations. These metrics help identify whether management is extracting maximum value from the company's resource base.

5.6.1 Asset Turnover

This ratio measures how efficiently a company uses total assets to generate revenue:

$$\text{Asset Turnover} = \frac{\text{Revenue}}{\text{Total Assets}}$$

A higher ratio indicates more efficient asset utilization: the company generates more sales per dollar of assets. However, optimal levels vary dramatically across industries. Retailers with minimal fixed assets might achieve turnovers above 2.0, while capital-intensive manufacturers might operate below 0.5.

5.6.2 Inventory Turnover

For companies carrying inventory, this ratio reveals how quickly stock moves through the business:

$$\text{Inventory Turnover} = \frac{\text{Cost of Goods Sold}}{\text{Inventory}}$$

Higher turnover suggests efficient inventory management (i.e., goods don't sit on shelves collecting dust). However, extremely high turnover might indicate stockout risks, while very low turnover could signal obsolete inventory or overinvestment in working capital.

We use COGS rather than revenue in the numerator because inventory is recorded at cost, not selling price. Using revenue would overstate turnover for high-margin businesses.

5.6.3 Receivables Turnover

This ratio measures how effectively a company collects payments from customers:

$$\text{Receivables Turnover} = \frac{\text{Revenue}}{\text{Accounts Receivable}}$$

Higher turnover indicates faster collection (i.e., customers pay promptly). Converting this to "days sales outstanding" (365 / turnover) gives the average collection period in days. Companies must balance collection efficiency against the sales impact of restrictive credit policies.

5.6.4 Calculating Efficiency Ratios

```
combined_statements = (balance_sheet_statements
    .get([
        "symbol", "year", "label", "current_ratio", "quick_ratio",
        "cash_ratio", "debt_to_equity", "debt_to_asset", "total_asset",
        "total_equity"
    ])
    .merge(
```

```
        (income_statements
            .get([
                "symbol", "year", "interest_coverage", "is_revenue",
                "is_cogs", "selling_general_and_administrative_expenses",
                "is_interest_expense", "is_gross_profit", "is_eat"
            ])
        ),
        on=["symbol", "year"],
        how="left"
    )
    .merge(
        (comp_vn30
            .assign(year=lambda x: x["year"].astype(int))
            .get(["symbol", "year", "ca_total_inventory", "ca_acc_receiv"])
        ),
        on=["symbol", "year"],
        how="left"
    )
)

combined_statements = combined_statements.assign(
    asset_turnover=lambda x: x["is_revenue"] / x["total_asset"],
    inventory_turnover=lambda x: x["is_cogs"] / x["ca_total_inventory"],
    receivables_turnover=lambda x: x["is_revenue"] / x["ca_acc_receiv"]
)
```

Efficiency ratios vary dramatically across industries, making peer comparison essential. A grocery store and a shipbuilder will have fundamentally different asset and inventory dynamics.

5.7 Profitability Ratios: Is the Company Making Money?

Profitability ratios evaluate how effectively a company converts activity into earnings. These metrics directly measure financial success and are among the most closely watched indicators by investors.

5.7.1 Gross Margin

The gross margin reveals what percentage of revenue remains after direct production costs:

$$\text{Gross Margin} = \frac{\text{Gross Profit}}{\text{Revenue}} = \frac{\text{Revenue} - \text{COGS}}{\text{Revenue}}$$

Higher gross margins indicate stronger pricing power, more efficient production, or a favorable product mix. This metric is particularly useful for comparing companies within an industry, as it reveals relative efficiency in core operations before overhead costs.

5.7.2 Profit Margin

The profit margin shows what percentage of revenue ultimately becomes net income:

$$\text{Profit Margin} = \frac{\text{Net Income}}{\text{Revenue}}$$

This comprehensive measure accounts for all costs (e.g., production, operations, interest, and taxes). Higher profit margins suggest effective overall cost management. However, optimal margins vary by industry: software companies routinely achieve 20%+ margins, while grocery stores operate on razor-thin 2-3% margins.

5.7.3 Return on Equity (ROE)

ROE measures how efficiently a company uses shareholders' investment to generate profits:

$$\text{Return on Equity} = \frac{\text{Net Income}}{\text{Total Equity}}$$

This metric directly addresses what shareholders care about: returns on their invested capital. Higher ROE indicates more effective use of equity, though interpretation requires caution. High leverage can artificially inflate ROE by reducing the equity base (e.g., a company financed 90% by debt will show spectacular ROE on modest profits).

5.7.4 The DuPont Decomposition

The DuPont framework decomposes ROE into three components that reveal different aspects of performance:

$$\text{ROE} = \underbrace{\frac{\text{Net Income}}{\text{Revenue}}}_{\text{Profit Margin}} \times \underbrace{\frac{\text{Revenue}}{\text{Assets}}}_{\text{Asset Turnover}} \times \underbrace{\frac{\text{Assets}}{\text{Equity}}}_{\text{Leverage}}$$

This decomposition shows that high ROE can come from different sources: strong profit margins (pricing power, cost control), efficient asset use (high turnover), or aggressive leverage. Understanding which driver dominates helps assess sustainability. ROE driven by margins is generally more sustainable than ROE driven by leverage.

5.7.5 Calculating Profitability Ratios

```
combined_statements = combined_statements.assign(
    gross_margin=lambda x: x["is_gross_profit"] / x["is_revenue"],
    profit_margin=lambda x: x["is_eat"] / x["is_revenue"],
    after_tax_roe=lambda x: x["is_eat"] / x["total_equity"]
)
```

5.7.6 Gross Margin Trends

Figure 5.5 tracks gross margin evolution over time, revealing whether companies are maintaining pricing power and production efficiency.

```
gross_margins = combined_statements.query("symbol in @vn30_symbols")

gross_margins_figure = (
    ggplot(gross_margins, aes(x="year", y="gross_margin", color="symbol"))
    + geom_line()
    + scale_y_continuous(labels=percent_format())
    + labs(
        x="", y="Gross Margin", color="",
        title="Gross Margins for VN30 Stocks (2019-2023)"
    )
)

gross_margins_figure.show()
```

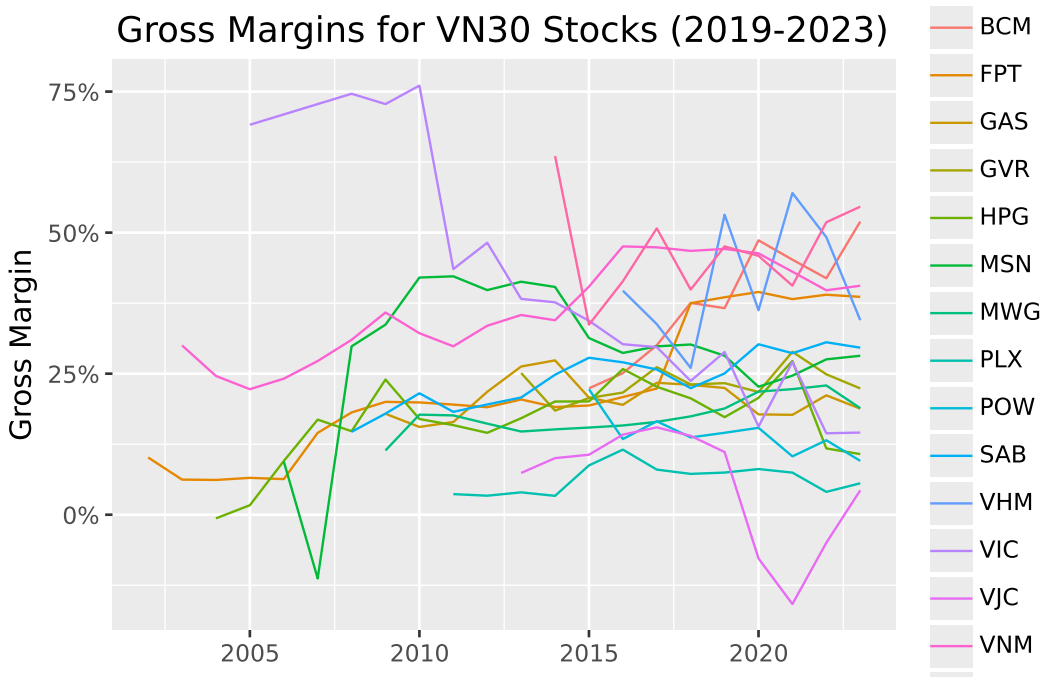


Figure 5.5: Gross margin trends reveal changes in pricing power and production efficiency. Declining margins may signal increased competition or rising input costs.

5.7.7 From Gross to Net: Where Do Profits Go?

Figure 5.6 examines the relationship between gross and profit margins. The gap between them reveals the impact of operating expenses, interest, and taxes.

```
profit_margins = combined_statements.query("year == 2023")

profit_margins_figure = (
    ggplot(
        profit_margins,
        aes(x="gross_margin", y="profit_margin", color="label")
    )
    + geom_point(size=2)
    + geom_label(
        aes(label="label"),
        adjust_text={"arrowprops": {"arrowstyle": "-"}}
    )
    + scale_x_continuous(labels=percent_format())
    + scale_y_continuous(labels=percent_format())
```

```
    + labs(
        x="Gross Margin", y="Profit Margin",
        title="Gross versus Profit Margins for VN30 Stocks (2023)"
    )
    + theme(legend_position="none")
)

profit_margins_figure.show()
```

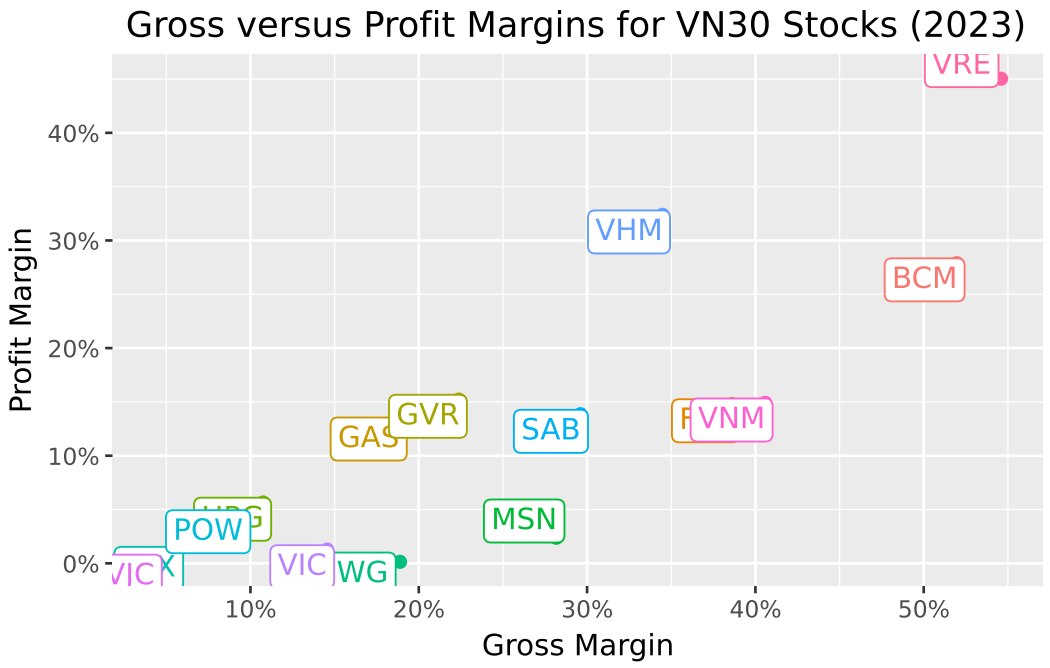


Figure 5.6: Comparing gross and profit margins reveals how much of gross profit survives operating expenses, interest, and taxes. Companies far below the diagonal have high overhead relative to gross profit.

Companies along the diagonal convert gross profit to net income efficiently. Those well below the diagonal face high operating costs, interest burdens, or tax rates that erode profitability.

5.8 Combining Financial Ratios: A Holistic View

Individual ratios provide specific insights, but combining them offers a more complete picture. A company might excel in profitability while struggling with liquidity, or maintain conservative leverage while underperforming on efficiency.

5.8.1 Ranking Companies Across Categories

Figure 5.7 compares company rankings across four ratio categories. Rankings closer to 1 indicate better performance within each category, enabling quick identification of relative strengths and weaknesses.

```
financial_ratios = (combined_statements
    .query("year == 2023")
    .filter(
        items=["symbol"] + [
            col for col in combined_statements.columns
            if any(x in col for x in [
                "ratio", "margin", "roe", "_to_", "turnover", "interest_coverage"
            ])
        ]
    )
    .melt(id_vars=["symbol"], var_name="name", value_name="value")
    .assign(
        type=lambda x: np.select(
            [
                x["name"].isin(["current_ratio", "quick_ratio", "cash_ratio"]),
                x["name"].isin(["debt_to_equity", "debt_to_asset", "interest_coverage"]),
                x["name"].isin(["asset_turnover", "inventory_turnover", "receivables_turnover
                x["name"].isin(["gross_margin", "profit_margin", "after_tax_roe"]),
            ],
            [
                "Liquidity Ratios",
                "Leverage Ratios",
                "Efficiency Ratios",
                "Profitability Ratios"
            ],
            default="Other"
        )
    )
)

financial_ratios["rank"] = (financial_ratios
    .sort_values(["type", "name", "value"], ascending=[True, True, False])
    .groupby(["type", "name"])
    .cumcount() + 1
)

final_ranks = (financial_ratios
```

```
    .groupby(["symbol", "type"], as_index=False)
    .agg(rank=("rank", "mean"))
    .query("symbol in @vn30_symbols")
)

final_ranks_figure = (
    ggplot(final_ranks, aes(x="rank", y="type", color="symbol"))
    + geom_point(shape="^", size=4)
    + labs(
        x="Average Rank (Lower is Better)", y="", color="",
        title="Average Rank Across Financial Ratio Categories"
    )
    + coord_cartesian(xlim=[1, 30])
)

final_ranks_figure.show()
```

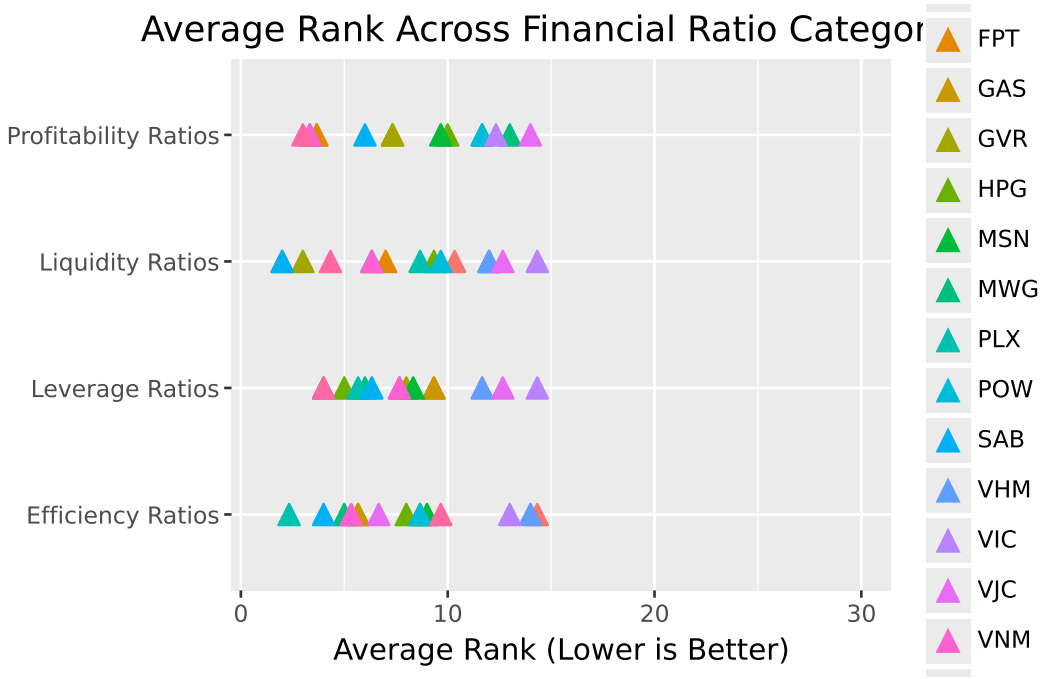


Figure 5.7: Ranking companies across multiple ratio categories reveals overall financial profiles. Companies with consistently low ranks across categories demonstrate broad-based financial strength.

The combined view reveals how different business strategies manifest in financial profiles. A

company might deliberately accept lower profitability rankings in exchange for stronger liquidity, or use aggressive leverage to boost returns at the cost of financial flexibility.

5.9 Financial Ratios in Asset Pricing

Beyond evaluating individual companies, financial ratios serve as crucial inputs for asset pricing models. The Fama-French five-factor model, which we explore in detail in Fama-French Factors, uses several accounting-based measures to explain cross-sectional variation in stock returns.

5.9.1 The Fama-French Factors

The model incorporates four company characteristics derived from financial statements:

Size is measured as the logarithm of market capitalization:

$$\text{Size} = \ln(\text{Market Cap})$$

This captures the empirical finding that smaller firms tend to outperform larger firms on a risk-adjusted basis (i.e., the "size premium").

Book-to-Market relates accounting value to market value:

$$\text{Book-to-Market} = \frac{\text{Book Equity}}{\text{Market Cap}}$$

High book-to-market stocks ("value" stocks) have historically outperformed low book-to-market stocks ("growth" stocks) (i.e., the "value premium").

Operating Profitability measures profit generation relative to equity:

$$\text{Profitability} = \frac{\text{Revenue} - \text{COGS} - \text{SG\&A} - \text{Interest}}{\text{Book Equity}}$$

More profitable firms tend to earn higher returns (i.e., the "profitability premium").

Investment captures asset growth:

$$\text{Investment} = \frac{\text{Total Assets}_t}{\text{Total Assets}_{t-1}} - 1$$

Firms investing aggressively tend to underperform conservative investors (i.e., the "investment premium").

5.9.2 Calculating Fama-French Variables

```
prices_monthly = pd.read_sql_query(
    sql="SELECT * FROM prices_monthly",
    con=tidy_finance,
    parse_dates={"datadate"}
)

# Use December prices for annual calculations
prices_december = (prices_monthly
    .assign(date=lambda x: pd.to_datetime(x["date"]))
    .query("date.dt.month == 12")
)
```

```
combined_statements_ff = (combined_statements
    .query("year == 2023")
    .merge(prices_december, on=["symbol", "year"], how="left")
    .merge(
        (balance_sheet_statements
            .query("year == 2022")
            .get(["symbol", "total_asset"])
            .rename(columns={"total_asset": "total_assets_lag"})
        ),
        on="symbol",
        how="left"
    )
    .assign(
        size=lambda x: np.log(x["mktcap"]),
        book_to_market=lambda x: x["total_equity"] / x["mktcap"],
        operating_profitability=lambda x: (
            (x["is_revenue"] - x["is_cogs"] -
             x["selling_general_and_administrative_expenses"] -
             x["is_interest_expense"]) / x["total_equity"]
        ),
        investment=lambda x: x["total_asset"] / x["total_assets_lag"] - 1
    )
)

combined_statements_ff.head(3)
```

	symbol	year	label	current_ratio	quick_ratio	cash_ratio	debt_to_equity	debt_to_asset	tot
0	POW	2023	POW	1.084255	1.084255	0.315089	0.0	0.0	7.0
1	HPG	2023	HPG	1.156655	1.156655	0.171324	0.0	0.0	1.8
2	MWG	2023	MWG	1.688604	1.688604	0.174408	0.0	0.0	6.0

5.9.3 Fama-French Factor Rankings

Figure 5.8 shows how VN30 companies rank on each Fama-French variable, connecting fundamental analysis to asset pricing.

```
factors_ranks = (combined_statements_ff
    .get(["symbol", "size", "book_to_market", "operating_profitability", "investment"])
    .rename(columns={
        "size": "Size",
        "book_to_market": "Book-to-Market",
        "operating_profitability": "Profitability",
        "investment": "Investment"
    })
    .melt(id_vars=["symbol"], var_name="name", value_name="value")
    .assign(
        rank=lambda x: (
            x.sort_values(["name", "value"], ascending=[True, False])
            .groupby("name")
            .cumcount() + 1
        )
    )
    .query("symbol in @vn30_symbols")
)

factors_ranks_figure = (
    ggplot(factors_ranks, aes(x="rank", y="name", color="symbol"))
    + geom_point(shape="^", size=4)
    + labs(
        x="Rank", y="", color="",
        title="Rank in Fama-French Variables for VN30 Stocks"
    )
    + coord_cartesian(xlim=[1, 30])
)

factors_ranks_figure.show()
```

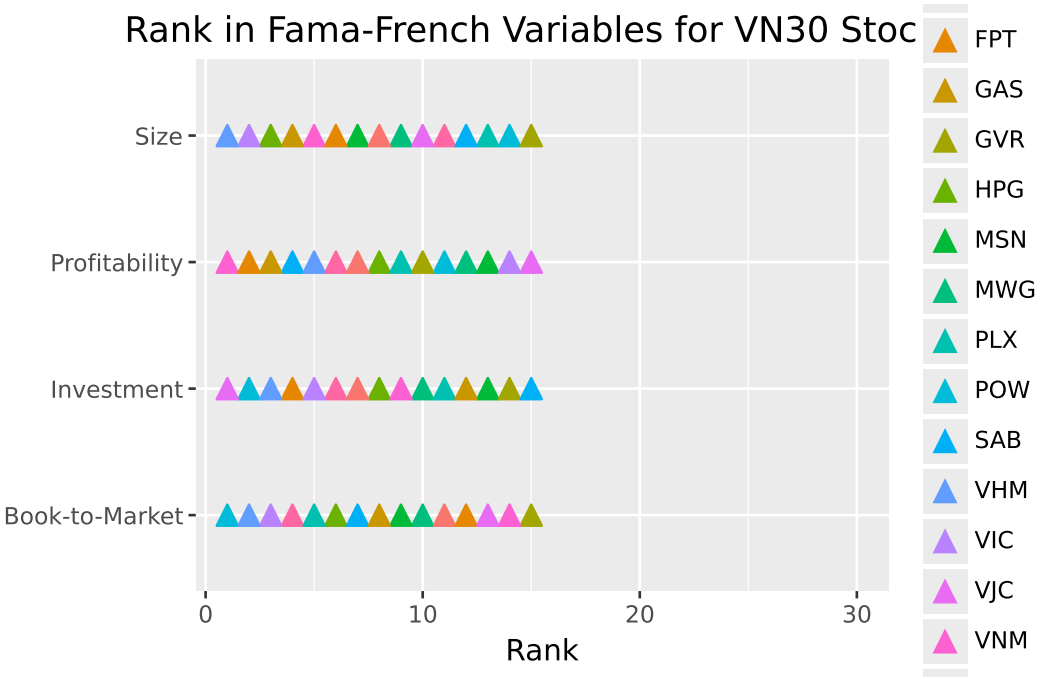


Figure 5.8: Rankings on Fama-French variables connect financial statement analysis to asset pricing. According to factor models, smaller, higher book-to-market, more profitable, and lower-investment firms should earn higher expected returns.

These rankings have implications for expected returns according to factor models. A small, high book-to-market, highly profitable company with conservative investment should, in theory, earn higher risk-adjusted returns than its opposite.

5.10 Limitations and Practical Considerations

While financial ratios provide powerful analytical tools, several limitations deserve attention:

5.10.1 Accounting Discretion

Companies have significant discretion in how they apply accounting standards. Revenue recognition timing, depreciation methods, inventory valuation (FIFO vs. LIFO), and capitalization versus expensing decisions all affect reported numbers. Sophisticated analysis requires understanding these choices and their impact.

5.10.2 Industry Comparability

Ratios vary dramatically across industries. Comparing a bank's leverage to a retailer's is meaningless (e.g., banks naturally operate with much higher leverage due to their business model). Always benchmark against industry peers rather than absolute standards.

5.10.3 Point-in-Time Limitations

Balance sheet ratios capture a single moment, which may not represent typical conditions. Companies often "window dress" by temporarily improving metrics at reporting dates. Trend analysis and quarter-over-quarter comparisons can reveal such practices.

5.10.4 Backward-Looking Nature

Financial statements report historical results. Past profitability doesn't guarantee future performance, especially for companies in rapidly changing industries or facing disruption.

5.10.5 Quality of Earnings

Not all profits are created equal. Earnings driven by one-time gains, accounting adjustments, or aggressive revenue recognition may not recur. Cash flow analysis helps assess earnings quality. Profits that don't convert to cash warrant skepticism.

5.11 Key Takeaways

This chapter introduced financial statement analysis as a tool for understanding company fundamentals. The main insights are:

1. **Three statements, three perspectives**: The balance sheet shows financial position at a point in time, the income statement measures performance over a period, and the cash flow statement tracks actual cash movements. Together, they provide a complete picture of financial health.
2. **Liquidity ratios assess short-term survival**: Current, quick, and cash ratios measure the ability to meet near-term obligations. Higher ratios indicate greater liquidity but may suggest inefficient asset use.
3. **Leverage ratios reveal capital structure risk**: Debt-to-equity, debt-to-asset, and interest coverage ratios show how the company finances operations and whether it can service its debt. Higher leverage amplifies both returns and risk.

4. **Efficiency ratios measure management effectiveness**: Asset turnover, inventory turnover, and receivables turnover reveal how well the company converts resources into revenue. Industry context is essential for interpretation.

5. **Profitability ratios quantify financial success**: Gross margin, profit margin, and ROE measure the ability to generate earnings. The DuPont decomposition reveals whether ROE comes from margins, turnover, or leverage.

6. **Ratios connect to asset pricing**: Financial statement variables like book-to-market, profitability, and investment form the basis of factor models that explain cross-sectional return differences.

7. **Context matters for interpretation**: Ratios must be compared against industry peers, tracked over time, and considered alongside qualitative factors. No single ratio tells the complete story.

Looking ahead, subsequent chapters will explore how these fundamental variables interact with market prices in asset pricing models, and how to construct factor portfolios based on financial statement characteristics.

6 Discounted Cash Flow Analysis

6.1 What Is a Company Worth?

The previous chapters examined how markets price securities in equilibrium and how financial statements reveal company fundamentals. But these approaches leave a central question unanswered: What is the *intrinsic value* of a business, independent of its current market price?

Discounted Cash Flow (DCF) analysis answers this question by valuing a company based on its ability to generate cash for investors. The core insight is simple: a business is worth the present value of all future cash it will produce. This principle that value equals discounted future cash flows underlies virtually all of finance, from bond pricing to real estate valuation.

DCF analysis stands apart from other valuation approaches in three important ways. First, it explicitly accounts for the **time value of money** (i.e., the principle that a dollar today is worth more than a dollar tomorrow). By discounting future cash flows at an appropriate rate, we incorporate both time preferences and risk. Second, DCF is **forward-looking**, making it particularly suitable for companies where historical performance may not reflect future potential. Third, DCF is **flexible** enough to accommodate various business models and capital structures, making it applicable across industries and company sizes.

6.1.1 Valuation Methods Overview

Company valuation methods broadly fall into three categories:

- **Market-based approaches** compare companies using relative metrics like Price-to-Earnings or EV/EBITDA ratios. These are quick but assume comparable companies are fairly valued.
- **Asset-based methods** focus on the net value of tangible and intangible assets. These work well for liquidation scenarios but miss going-concern value.
- **Income-based techniques** value companies based on their ability to generate future cash flows. DCF is the most rigorous income-based method.

We focus on DCF because it forces analysts to make explicit assumptions about growth, profitability, and risk. These assumptions are often hidden in other methods. Even when DCF

isn't the final word on valuation, the discipline of building a DCF model deepens understanding of what drives value.

6.1.2 The Three Pillars of DCF

Every DCF analysis rests on three components:

1. **Free Cash Flow (FCF) forecasts**: The expected future cash available for distribution to investors after operating expenses, taxes, and investments
2. **Terminal value**: The company's value beyond the explicit forecast period, often representing a majority of total valuation
3. **Discount rate**: Typically the Weighted Average Cost of Capital (WACC), which adjusts future cash flows to present value by incorporating risk and capital structure

We make simplifying assumptions throughout this chapter. In particular, we assume firms conduct only operating activities (i.e., financial statements do not include non-operating items like excess cash or investment securities). Real-world valuations require valuing these separately. Entire textbooks are devoted to valuation nuances; our goal is to establish the conceptual framework and practical implementation.

```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf

from plotnine import *
from mizani.formatters import percent_format, comma_format
from itertools import product
```

6.2 Understanding Free Cash Flow

Before diving into calculations, we need to understand what Free Cash Flow represents and why it matters for valuation.

6.2.1 Why Free Cash Flow, Not Net Income?

Accountants report net income, but DCF uses free cash flow. Why the difference?

Net income includes non-cash items (like depreciation) and ignores cash needs (like capital expenditures and working capital investments). A company can report strong profits while burning cash, or generate substantial cash while reporting losses. Free cash flow captures what

actually matters for valuation: the cash available to distribute to all capital providers (both debt holders and equity holders) after funding operations and investments.

6.2.2 The Free Cash Flow Formula

We calculate FCF using the following formula:

$$\text{FCF} = \text{EBIT} \times (1 - \tau) + \text{D\&A} - \Delta\text{WC} - \text{CAPEX}$$

where:

- **EBIT** (Earnings Before Interest and Taxes): Core operating profit before financing costs and taxes
- τ: Corporate tax rate applied to operating profits
- **D&A** (Depreciation & Amortization): Non-cash charges that reduce reported earnings but don't consume cash
- **ΔWC** (Change in Working Capital): Cash tied up in (or released from) operations (increases in receivables and inventory consume cash, while increases in payables provide cash)
- **CAPEX** (Capital Expenditures): Investments in long-term assets required to maintain and grow operations

An alternative formulation starts from EBIT directly:

$$\text{FCF} = \text{EBIT} + \text{D\&A} - \text{Taxes} - \Delta\text{WC} - \text{CAPEX}$$

Both formulations yield the same result when taxes are calculated consistently. The key insight is that FCF represents cash generated from operations after all reinvestment needs (i.e., cash that could theoretically be distributed to investors without impairing the business).

6.3 Loading Historical Financial Data

We use FPT Corporation, one of Vietnam's largest technology companies, as our case study. FPT provides IT services, telecommunications, and education. It's a diversified business with meaningful capital requirements and growth potential.

```
import sqlite3

tidy_finance = sqlite3.connect(database="data/tidy_finance_python.sqlite")

comp_vn = pd.read_sql_query(
    sql="SELECT * FROM comp_vn",
    con=tidy_finance,
    parse_dates={"date"}
)

# Filter to FPT and examine the data structure
fpt_data = comp_vn[comp_vn["symbol"] == "FPT"].copy()
fpt_data["year"] = fpt_data["year"].astype(int)
fpt_data = fpt_data.sort_values("year").reset_index(drop=True)

print(f"Available years: {fpt_data['year'].min()} to {fpt_data['year'].max()}")
print(f"Number of observations: {len(fpt_data)}")
```

```
Available years: 2002 to 2023
Number of observations: 22
```

6.3.1 Computing Historical Free Cash Flow

Let's calculate the components needed for FCF from the financial statement data:

```
# Extract and compute FCF components
historical_data = (fpt_data
    .assign(
        # Revenue for ratio calculations
        revenue=lambda x: x["is_net_revenue"],

        # EBIT = Earnings before interest and taxes
        # Approximate as EBT + Interest Expense
        ebit=lambda x: x["is_ebt"] + x["is_interest_expense"],

        # Tax payments (use actual tax expense)
        taxes=lambda x: x["is_cit_expense"],

        # Depreciation and amortization (non-cash add-back)
        depreciation=lambda x: x["cfo_depreciation"],
```

```
        # Change in working capital components
        # Positive delta_wc means cash is consumed (tied up in working capital)
        delta_working_capital=lambda x: (
            x["cfo_receive"] +      # Change in receivables
            x["cfo_inventory"] -    # Change in inventory
            x["cfo_payale"]         # Change in payables (negative = cash source)
        ),

        # Capital expenditures
        capex=lambda x: x["capex"]
    )
    .loc[:, [
        "year", "revenue", "ebit", "taxes", "depreciation",
        "delta_working_capital", "capex"
    ]]
)

# Calculate Free Cash Flow
historical_data["fcf"] = (
    historical_data["ebit"]
    - historical_data["taxes"]
    + historical_data["depreciation"]
    - historical_data["delta_working_capital"]
    - historical_data["capex"]
)

historical_data
```

	year	revenue	ebit	taxes	depreciation	delta_working_capital	capex
0	2002	1.514961e+12	2.698700e+10	0.000000e+00	1.261500e+10	-2.561760e+11	2.202800
1	2003	4.148298e+12	5.676100e+10	0.000000e+00	1.837700e+10	-5.078740e+11	3.753300
2	2004	8.734781e+12	2.145902e+11	1.795700e+10	2.947900e+10	-4.280270e+11	5.252100
3	2005	1.410079e+13	3.753490e+11	4.251500e+10	5.381700e+10	-4.471110e+11	1.428320
4	2006	2.139975e+13	6.672593e+11	7.368682e+10	1.068192e+11	-1.173099e+12	2.459780
5	2007	1.349889e+13	1.071941e+12	1.487146e+11	1.709335e+11	-1.873794e+12	4.802762
6	2008	1.638184e+13	1.320573e+12	1.890384e+11	2.395799e+11	-1.419506e+11	6.690461
7	2009	1.840403e+13	1.807221e+12	2.916482e+11	3.041813e+11	-8.065011e+11	7.632280
8	2010	2.001730e+13	2.261341e+12	3.314359e+11	3.294060e+11	-2.360993e+12	8.672138
9	2011	2.537025e+13	2.751044e+12	4.223952e+11	3.759567e+11	-2.099380e+12	4.524081
10	2012	2.459430e+13	2.635219e+12	4.210738e+11	3.995598e+11	8.043763e+11	7.083318
11	2013	2.702789e+13	2.690568e+12	4.503170e+11	4.429860e+11	-1.947751e+12	9.110216

	year	revenue	ebit	taxes	depreciation	delta_working_capital	capex
12	2014	3.264466e+13	2.625389e+12	3.800994e+11	5.472736e+11	-3.078130e+12	1.417399
13	2015	3.795970e+13	3.113651e+12	4.130641e+11	7.328801e+11	-1.951778e+12	1.974295
14	2016	3.953147e+13	3.388085e+12	4.382078e+11	9.334397e+11	-9.242713e+11	1.428472
15	2017	4.265861e+13	4.623663e+12	7.270039e+11	1.039417e+12	-4.638788e+12	1.100498
16	2018	2.321354e+13	4.095947e+12	6.236054e+11	1.164692e+12	-1.033438e+12	2.452902
17	2019	2.771696e+13	5.023518e+12	7.528183e+11	1.354613e+12	-5.308818e+11	3.230818
18	2020	2.983040e+13	5.648794e+12	8.397114e+11	1.490607e+12	-8.040730e+11	3.014322
19	2021	3.565726e+13	6.821202e+12	9.879053e+11	1.643916e+12	-2.821825e+12	2.908134
20	2022	4.400953e+13	8.308009e+12	1.170940e+12	1.833064e+12	-3.746661e+12	3.209581
21	2023	5.261790e+13	1.003565e+13	1.414956e+12	2.286514e+12	-2.147304e+12	3.948982

6.3.2 Understanding the Historical Pattern

Before forecasting, we should understand the historical trends in FCF and its components:

```
# Calculate key ratios relative to revenue
historical_ratios = (historical_data
    .assign(
        # Revenue growth (year-over-year)
        revenue_growth=lambda x: x["revenue"].pct_change(),

        # Operating margin: EBIT as % of revenue
        operating_margin=lambda x: x["ebit"] / x["revenue"],

        # Depreciation as % of revenue
        depreciation_margin=lambda x: x["depreciation"] / x["revenue"],

        # Tax rate (taxes as % of revenue, for simplicity)
        tax_margin=lambda x: x["taxes"] / x["revenue"],

        # Working capital intensity
        working_capital_margin=lambda x: x["delta_working_capital"] / x["revenue"],

        # Capital intensity
        capex_margin=lambda x: x["capex"] / x["revenue"],

        # FCF margin
        fcf_margin=lambda x: x["fcf"] / x["revenue"]
    )
)
```

```
# Display key metrics
display_cols = [
    "year", "revenue_growth", "operating_margin", "depreciation_margin",
    "tax_margin", "working_capital_margin", "capex_margin", "fcf_margin"
]

historical_ratios[display_cols].round(3)
```

	year	revenue_growth	operating_margin	depreciation_margin	tax_margin	working_capital_mar
0	2002	NaN	0.018	0.008	0.000	-0.169
1	2003	1.738	0.014	0.004	0.000	-0.122
2	2004	1.106	0.025	0.003	0.002	-0.049
3	2005	0.614	0.027	0.004	0.003	-0.032
4	2006	0.518	0.031	0.005	0.003	-0.055
5	2007	-0.369	0.079	0.013	0.011	-0.139
6	2008	0.214	0.081	0.015	0.012	-0.009
7	2009	0.123	0.098	0.017	0.016	-0.044
8	2010	0.088	0.113	0.016	0.017	-0.118
9	2011	0.267	0.108	0.015	0.017	-0.083
10	2012	-0.031	0.107	0.016	0.017	0.033
11	2013	0.099	0.100	0.016	0.017	-0.072
12	2014	0.208	0.080	0.017	0.012	-0.094
13	2015	0.163	0.082	0.019	0.011	-0.051
14	2016	0.041	0.086	0.024	0.011	-0.023
15	2017	0.079	0.108	0.024	0.017	-0.109
16	2018	-0.456	0.176	0.050	0.027	-0.045
17	2019	0.194	0.181	0.049	0.027	-0.019
18	2020	0.076	0.189	0.050	0.028	-0.027
19	2021	0.195	0.191	0.046	0.028	-0.079
20	2022	0.234	0.189	0.042	0.027	-0.085
21	2023	0.196	0.191	0.043	0.027	-0.041

6.4 Visualizing Historical Ratios

Figure 6.1 shows the historical evolution of key financial ratios that drive FCF. Understanding these patterns helps inform our forecasts.

```
# Prepare data for plotting
ratio_columns = [
    "operating_margin", "depreciation_margin", "tax_margin",
    "working_capital_margin", "capex_margin"
]

ratios_long = (historical_ratios
    .melt(
        id_vars=["year"],
        value_vars=ratio_columns,
        var_name="ratio",
        value_name="value"
    )
    .assign(
        ratio=lambda x: x["ratio"].str.replace("_", " ").str.title()
    )
)

ratios_figure = (
    ggplot(ratios_long, aes(x="year", y="value", color="ratio"))
    + geom_line(size=1)
    + geom_point(size=2)
    + scale_y_continuous(labels=percent_format())
    + labs(
        x="", y="Ratio (% of Revenue)", color="",
        title="Key Financial Ratios of FPT Over Time"
    )
    + theme(legend_position="right")
)

ratios_figure.show()
```

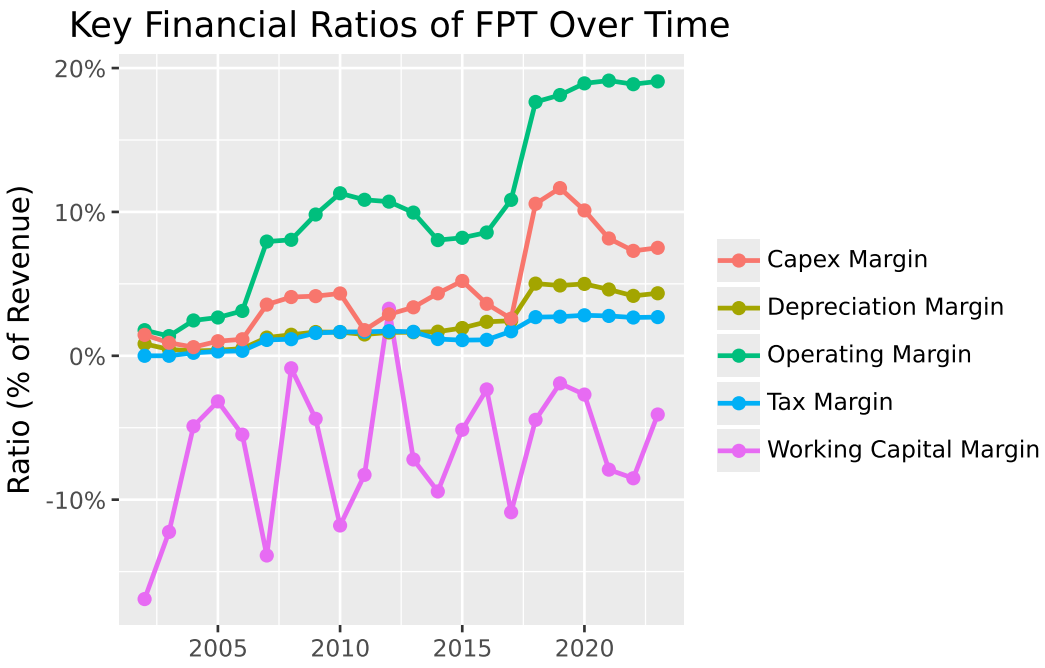


Figure 6.1: Historical financial ratios reveal the operating characteristics of FPT. These patterns inform our forecast assumptions.

Several patterns emerge from the historical data. Operating margins show the profitability of core operations. Depreciation margins indicate asset intensity. CAPEX margins reveal investment requirements. Working capital margins can be volatile, reflecting changes in credit terms and inventory management.

6.5 Forecasting Free Cash Flow

With historical patterns established, we now project FCF into the future. This requires forecasting both revenue growth and the ratios that convert revenue into cash flow.

6.5.1 The Ratio-Based Forecasting Approach

We use a ratio-based approach that links all FCF components to revenue. This makes forecasting tractable: rather than projecting absolute dollar amounts for each component, we forecast (1) revenue growth and (2) how each component scales with revenue.

This approach embeds a key assumption: that the relationship between revenue and FCF components remains stable. In reality, operating leverage, investment needs, and working

capital requirements may change as companies mature. Sophisticated valuations model these dynamics explicitly.

6.5.2 Setting Forecast Assumptions

For our five-year forecast, we make the following assumptions about FPT's financial ratios. These should reflect industry analysis, company guidance, and competitive dynamics. Here we use estimates for illustration:

```
# Define the forecast horizon
last_historical_year = historical_data["year"].max()
forecast_years = list(range(last_historical_year + 1, last_historical_year + 6))
n_forecast_years = len(forecast_years)

print(f"Forecast period: {forecast_years[0]} to {forecast_years[-1]}")

# Define forecast ratios
# In practice, these would come from detailed analysis
forecast_assumptions = pd.DataFrame({
    "year": forecast_years,
    # Operating margin: slight improvement as scale increases
    "operating_margin": [0.12, 0.125, 0.13, 0.13, 0.135],
    # Depreciation: stable as % of revenue
    "depreciation_margin": [0.03, 0.03, 0.03, 0.028, 0.028],
    # Tax rate: stable
    "tax_margin": [0.02, 0.02, 0.02, 0.02, 0.02],
    # Working capital: modest cash consumption
    "working_capital_margin": [0.01, 0.01, 0.008, 0.008, 0.008],
    # CAPEX: declining as % of revenue as growth moderates
    "capex_margin": [0.05, 0.048, 0.045, 0.042, 0.04]
})

forecast_assumptions
```

```
Forecast period: 2024 to 2028
```

	year	operating_margin	depreciation_margin	tax_margin	working_capital_margin	capex_margin
0	2024	0.120	0.030	0.02	0.010	0.050
1	2025	0.125	0.030	0.02	0.010	0.048
2	2026	0.130	0.030	0.02	0.008	0.045

	year	operating_margin	depreciation_margin	tax_margin	working_capital_margin	capex_margin
3	2027	0.130	0.028	0.02	0.008	0.042
4	2028	0.135	0.028	0.02	0.008	0.040

6.5.3 Forecasting Revenue Growth

Revenue growth is often the most important and most uncertain assumption in DCF analysis. We demonstrate two approaches: using historical averages and linking growth to macroeconomic forecasts.

Approach 1: Historical Average

A simple approach uses the historical average growth rate:

```
historical_growth = historical_ratios["revenue_growth"].dropna()
avg_historical_growth = historical_growth.mean()

print(f"Average historical revenue growth: {avg_historical_growth:.1%}")
```

```
Average historical revenue growth: 25.2%
```

Approach 2: GDP-Linked Growth

A more sophisticated approach links company growth to GDP forecasts from institutions like the IMF. This captures the intuition that company revenues often move with broader economic activity.

```
# Vietnam GDP growth forecasts (illustrative, based on IMF WEO style projections)
# In practice, download from IMF WEO database
gdp_forecasts = pd.DataFrame({
    "year": forecast_years,
    "gdp_growth": [0.065, 0.063, 0.060, 0.058, 0.055]  # Gradually declining to long-term
})

# Assume FPT grows at a premium to GDP (tech sector outperformance)
# This premium should reflect company-specific factors
growth_premium = 0.05  # 5 percentage points above GDP

forecast_assumptions = forecast_assumptions.merge(gdp_forecasts, on="year")
forecast_assumptions["revenue_growth"] = (
    forecast_assumptions["gdp_growth"] + growth_premium
```

```
)

forecast_assumptions[["year", "gdp_growth", "revenue_growth"]]
```

	year	gdp_growth	revenue_growth
0	2024	0.065	0.115
1	2025	0.063	0.113
2	2026	0.060	0.110
3	2027	0.058	0.108
4	2028	0.055	0.105

6.5.4 Building the Forecast

Now we combine our assumptions to project revenue and FCF:

```
# Get the last historical revenue as our starting point
last_revenue = historical_data.loc[
    historical_data["year"] == last_historical_year, "revenue"
].values[0]

print(f"Last historical revenue ({last_historical_year}): {last_revenue/1e12:.2f} trillion VI

# Project revenue forward
forecast_data = forecast_assumptions.copy()
forecast_data["revenue"] = None

# Calculate revenue for each forecast year
for i, row in forecast_data.iterrows():
    if i == 0:
        # First forecast year: grow from last historical
        forecast_data.loc[i, "revenue"] = last_revenue * (1 + row["revenue_growth"])
    else:
        # Subsequent years: grow from previous forecast
        prev_revenue = forecast_data.loc[i-1, "revenue"]
        forecast_data.loc[i, "revenue"] = prev_revenue * (1 + row["revenue_growth"])

# Convert revenue to numeric
forecast_data["revenue"] = forecast_data["revenue"].astype(float)

# Calculate FCF components from ratios
```

```
forecast_data["ebit"] = forecast_data["operating_margin"] * forecast_data["revenue"]
forecast_data["depreciation"] = forecast_data["depreciation_margin"] * forecast_data["revenue
forecast_data["taxes"] = forecast_data["tax_margin"] * forecast_data["revenue"]
forecast_data["delta_working_capital"] = forecast_data["working_capital_margin"] * forecast_
forecast_data["capex"] = forecast_data["capex_margin"] * forecast_data["revenue"]

# Calculate FCF
forecast_data["fcf"] = (
    forecast_data["ebit"]
    - forecast_data["taxes"]
    + forecast_data["depreciation"]
    - forecast_data["delta_working_capital"]
    - forecast_data["capex"]
)

forecast_data[["year", "revenue", "ebit", "fcf"]].round(0)
```

```
Last historical revenue (2023): 52.62 trillion VND
```

	year	revenue	ebit	fcf
0	2024	5.866896e+13	7.040275e+12	4.106827e+12
1	2025	6.529855e+13	8.162319e+12	5.027988e+12
2	2026	7.248139e+13	9.422581e+12	6.305881e+12
3	2027	8.030938e+13	1.044022e+13	7.067226e+12
4	2028	8.874187e+13	1.198015e+13	8.430477e+12

6.6 Visualizing the Forecast

Figure 6.2 compares our forecast ratios with historical values, showing the transition from realized to projected performance.

```
# Prepare historical data for plotting
historical_plot = (historical_ratios
    .loc[:, ["year", "operating_margin", "depreciation_margin",
             "tax_margin", "working_capital_margin", "capex_margin"]]
    .assign(type="Historical")
)
```

```
# Prepare forecast data for plotting
forecast_plot = (forecast_data
    .loc[:, ["year", "operating_margin", "depreciation_margin",
             "tax_margin", "working_capital_margin", "capex_margin"]]
    .assign(type="Forecast")
)

# Combine
combined_ratios = pd.concat([historical_plot, forecast_plot], ignore_index=True)

# Reshape for plotting
combined_long = combined_ratios.melt(
    id_vars=["year", "type"],
    var_name="ratio",
    value_name="value"
)

combined_long["type"] = pd.Categorical(
    combined_long["type"],
    categories=["Historical", "Forecast"]
)

forecast_ratios_figure = (
    ggplot(combined_long, aes(x="year", y="value", color="ratio", linetype="type"))
    + geom_line(size=1)
    + geom_point(size=2)
    + scale_y_continuous(labels=percent_format())
    + labs(
        x="", y="Ratio (% of Revenue)", color="", linetype="",
        title="Historical and Forecast Financial Ratios for FPT"
    )
    + theme(legend_position="right")
)

forecast_ratios_figure.show()
```

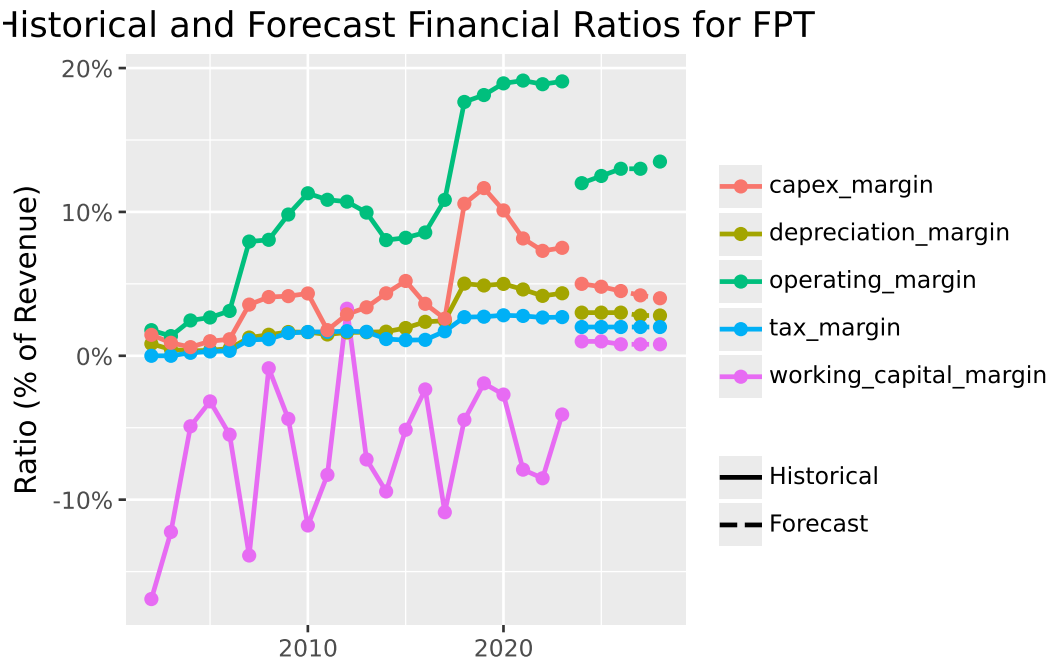


Figure 6.2: Historical ratios (solid lines) and forecast assumptions (dashed lines) for key financial metrics. The forecast period begins after the last historical observation.

Figure 6.3 shows the revenue growth trajectory, comparing historical performance with our GDP-linked forecasts.

```
# Prepare growth data
historical_growth_df = (historical_ratios
    .loc[:, ["year", "revenue_growth"]]
    .dropna()
    .assign(type="Historical")
)

forecast_growth_df = (forecast_data
    .loc[:, ["year", "revenue_growth", "gdp_growth"]]
    .assign(type="Forecast")
)

# Combine for revenue growth
growth_combined = pd.concat([
    historical_growth_df,
    forecast_growth_df[["year", "revenue_growth", "type"]]
], ignore_index=True)
```

```
growth_combined["type"] = pd.Categorical(
    growth_combined["type"],
    categories=["Historical", "Forecast"]
)

growth_figure = (
    ggplot(growth_combined, aes(x="year", y="revenue_growth", linetype="type"))
    + geom_line(size=1, color="steelblue")
    + geom_point(size=2, color="steelblue")
    + scale_y_continuous(labels=percent_format())
    + labs(
        x="", y="Revenue Growth Rate", linetype="",
        title="Historical and Forecast Revenue Growth for FPT"
    )
)

growth_figure.show()
```

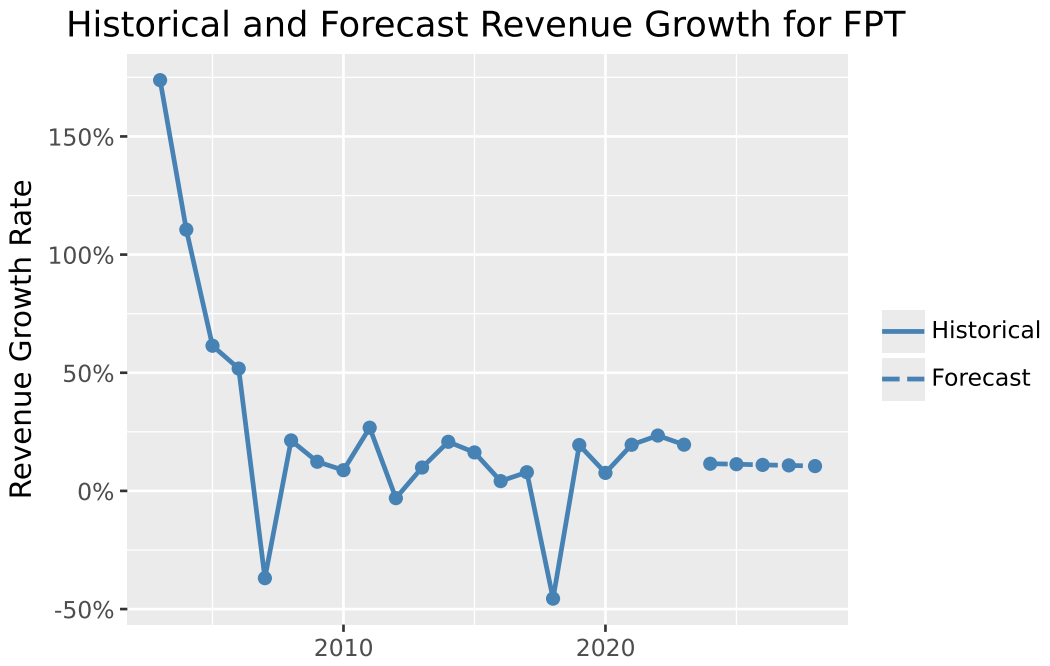


Figure 6.3: Revenue growth rates: historical (realized) and forecast (GDP-linked with company premium). The forecast assumes FPT grows at a premium to Vietnam's GDP growth.

Figure 6.4 presents the resulting FCF projections alongside historical values.

```
# Combine historical and forecast FCF
fcf_historical = (historical_data
    .loc[:, ["year", "fcf"]]
    .assign(type="Historical")
)

fcf_forecast = (forecast_data
    .loc[:, ["year", "fcf"]]
    .assign(type="Forecast")
)

fcf_combined = pd.concat([fcf_historical, fcf_forecast], ignore_index=True)
fcf_combined["type"] = pd.Categorical(
    fcf_combined["type"],
    categories=["Historical", "Forecast"]
)

fcf_figure = (
    ggplot(fcf_combined, aes(x="year", y="fcf/1e12", fill="type"))
    + geom_col()
    + labs(
        x="", y="Free Cash Flow (Trillion VND)", fill="",
        title="Historical and Forecast Free Cash Flow for FPT"
    )
)

fcf_figure.show()
```

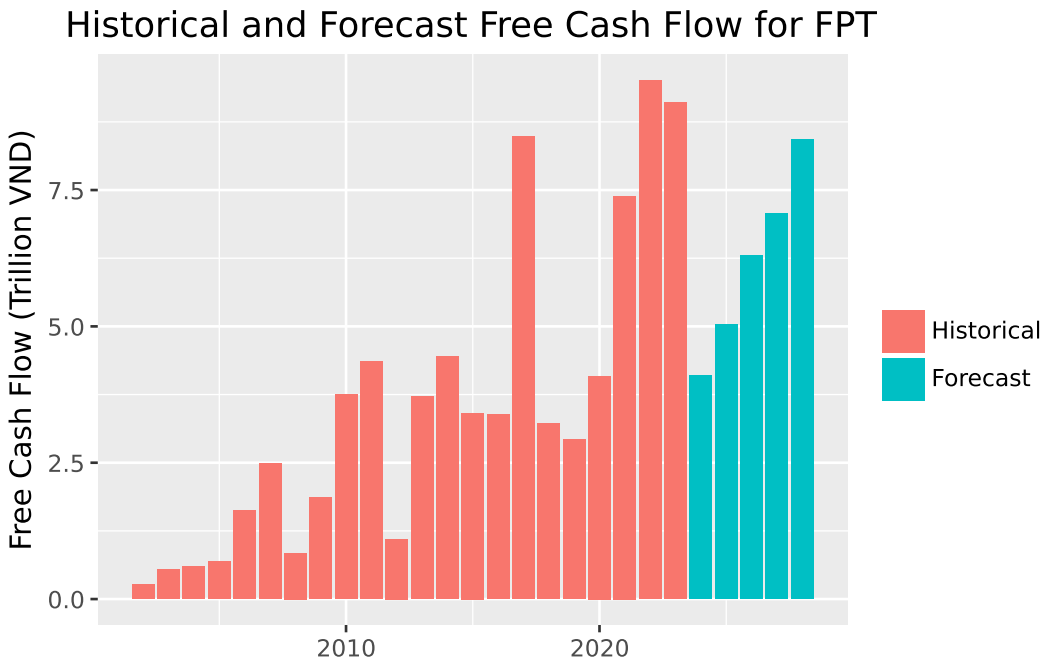


Figure 6.4: Free Cash Flow: historical (realized) and forecast (projected). The forecast reflects our assumptions about revenue growth and operating ratios.

6.7 Terminal Value: Capturing Long-Term Value

A critical component of DCF analysis is the **terminal value** (or continuation value), which represents the company's value beyond the explicit forecast period. In most valuations, terminal value constitutes 50-80% of total enterprise value, making its estimation particularly important.

6.7.1 The Perpetuity Growth Model

The most common approach is the **Perpetuity Growth Model** (also called the Gordon Growth Model), which assumes FCF grows at a constant rate forever:

$$TV_T = \frac{FCF_{T+1}}{r - g} = \frac{FCF_T \times (1 + g)}{r - g}$$

where:

- TV_T: Terminal value at the end of year T

- FCF_T: Free cash flow in the final forecast year
- g: Perpetual growth rate
- r: Discount rate (WACC)

6.7.2 Choosing the Perpetual Growth Rate

The perpetual growth rate g should reflect long-term sustainable growth. Key considerations:

1. **No company can grow faster than the economy forever**. If it did, the company would eventually become larger than GDP, which is an impossibility. Long-term GDP growth (nominal, including inflation) provides an upper bound.
2. **Mature companies typically grow at or below GDP growth**. The perpetual growth rate should reflect the company in its "steady state," not its current high-growth phase.
3. **For Vietnam**, long-term nominal GDP growth might be 6-8% given current development stage, but this will moderate over time. A perpetual growth rate of 3-5% is often reasonable.

```python
def compute_terminal_value(last_fcf, growth_rate, discount_rate):
    """
    Compute terminal value using the perpetuity growth model.

    Parameters:
    -----------
    last_fcf : float
        Free cash flow in the final forecast year
    growth_rate : float
        Perpetual growth rate (g)
    discount_rate : float
        Discount rate / WACC (r)

    Returns:
    --------
    float : Terminal value
    """
    if discount_rate <= growth_rate:
        raise ValueError("Discount rate must exceed growth rate for finite terminal value")

    return last_fcf * (1 + growth_rate) / (discount_rate - growth_rate)
```

```python
# Example calculation
last_fcf = forecast_data["fcf"].iloc[-1]
perpetual_growth = 0.04  # 4% perpetual growth
discount_rate = 0.10     # 10% WACC (placeholder)

terminal_value = compute_terminal_value(last_fcf, perpetual_growth, discount_rate)

print(f"Last forecast FCF: {last_fcf/1e12:.2f} trillion VND")
print(f"Terminal value (at {perpetual_growth:.0%} growth, {discount_rate:.0%} WACC): {termina
```

```
Last forecast FCF: 8.43 trillion VND
Terminal value (at 4% growth, 10% WACC): 146.1 trillion VND
```

6.7.3 Alternative: Exit Multiple Approach

Practitioners often cross-check terminal value using the **exit multiple approach**, which assumes the company is sold at the end of the forecast period at a multiple of EBITDA, EBIT, or revenue comparable to similar companies today.

For example, if comparable companies trade at 10x EBITDA, the terminal value would be:

$$TV_T = \text{EBITDA}_T \times \text{Exit Multiple}$$

This approach is simpler but embeds the assumption that current market multiples will persist (a strong assumption that may not hold).

6.8 The Discount Rate: Weighted Average Cost of Capital

The discount rate converts future cash flows to present value. For FCF (which goes to all capital providers), we use the **Weighted Average Cost of Capital (WACC)**:

$$WACC = \frac{E}{E+D} \times r_E + \frac{D}{E+D} \times r_D \times (1-\tau)$$

where:

- E: Market value of equity
- D: Market value of debt
- r_E: Cost of equity (typically estimated using CAPM)
- r_D: Cost of debt (pre-tax)

- τ: Corporate tax rate

The $(1-\tau)$ term on debt reflects the tax shield. Interest payments are tax-deductible, reducing the effective cost of debt.

6.8.1 Estimating WACC Components

Cost of Equity is typically estimated using the Capital Asset Pricing Model (see our CAPM chapter):

$$r_E = r_f + \beta \times (r_m - r_f)$$

where r_f is the risk-free rate, β measures systematic risk, and $(r_m - r_f)$ is the market risk premium.

Cost of Debt can be estimated from:

- Interest expense divided by total debt (effective rate)
- Yields on the company's traded bonds
- Yields on bonds with similar credit ratings

Capital Structure Weights should use market values when available. For equity, market capitalization is straightforward. For debt, book value is often used when market values aren't observable.

6.8.2 Using Industry WACC Data

Professor Aswath Damodaran at NYU Stern maintains comprehensive industry WACC data. Let's download and use this resource:

```
import requests
import os

# Download Damodaran's WACC data
url = "https://pages.stern.nyu.edu/~adamodar/pc/datasets/wacc.xls"

try:
    response = requests.get(url, timeout=10)
    response.raise_for_status()

    with open("wacc.xls", "wb") as f:
        f.write(response.content)
```

```
    # Read the data (skip header rows)
    wacc_data = pd.read_excel("wacc.xls", sheet_name=1, skiprows=18)

    # Clean up
    os.remove("wacc.xls")

    # Find WACC for Computer Services (closest to FPT's business)
    industry_wacc = wacc_data.loc[
        wacc_data["Industry Name"] == "Computer Services",
        "Cost of Capital"
    ].values[0]

    print(f"Industry WACC (Computer Services): {industry_wacc:.2%}")

except Exception as e:
    print(f"Could not download WACC data: {e}")
    # Use a reasonable estimate
    industry_wacc = 0.10
    print(f"Using estimated WACC: {industry_wacc:.2%}")

wacc = industry_wacc
```

```
Could not download WACC data: `Import xlrd` failed. Install xlrd >= 2.0.1 for xls Excel suppo
Using estimated WACC: 10.00%
```

Note: Industry WACC provides a useful benchmark, but company-specific factors (leverage, business risk, country risk) may warrant adjustments. For Vietnamese companies, adding a country risk premium may be appropriate.

6.9 Computing Enterprise Value

With all components in place, we can now compute enterprise value. The DCF formula is:

$$\text{Enterprise Value} = \sum_{t=1}^{T} \frac{FCF_t}{(1+WACC)^t} + \frac{TV_T}{(1+WACC)^T}$$

The first term is the present value of forecast-period cash flows; the second is the present value of terminal value.

```
def compute_dcf_value(fcf_series, wacc, perpetual_growth):
    """
    Compute enterprise value using DCF analysis.

    Parameters:
    -----------
    fcf_series : array-like
        Free cash flows for forecast period
    wacc : float
        Weighted average cost of capital
    perpetual_growth : float
        Perpetual growth rate for terminal value

    Returns:
    --------
    dict : Components of DCF valuation
    """
    fcf = np.array(fcf_series)
    n_years = len(fcf)

    # Discount factors
    discount_factors = (1 + wacc) ** np.arange(1, n_years + 1)

    # Present value of forecast period cash flows
    pv_fcf = fcf / discount_factors
    pv_fcf_total = pv_fcf.sum()

    # Terminal value and its present value
    terminal_value = compute_terminal_value(fcf[-1], perpetual_growth, wacc)
    pv_terminal = terminal_value / discount_factors[-1]

    # Total enterprise value
    enterprise_value = pv_fcf_total + pv_terminal

    return {
        "pv_fcf": pv_fcf_total,
        "terminal_value": terminal_value,
        "pv_terminal": pv_terminal,
        "enterprise_value": enterprise_value,
        "terminal_pct": pv_terminal / enterprise_value
    }
```

```
# Compute DCF value
perpetual_growth = 0.04  # 4% perpetual growth

dcf_result = compute_dcf_value(
    fcf_series=forecast_data["fcf"].values,
    wacc=wacc,
    perpetual_growth=perpetual_growth
)

print("DCF Valuation Results")
print("=" * 50)
print(f"PV of Forecast Period FCF: {dcf_result['pv_fcf']/1e12:.1f} trillion VND")
print(f"Terminal Value: {dcf_result['terminal_value']/1e12:.1f} trillion VND")
print(f"PV of Terminal Value: {dcf_result['pv_terminal']/1e12:.1f} trillion VND")
print(f"Enterprise Value: {dcf_result['enterprise_value']/1e12:.1f} trillion VND")
print(f"Terminal Value as % of EV: {dcf_result['terminal_pct']:.1%}")
```

```
DCF Valuation Results
==================================================
PV of Forecast Period FCF: 22.7 trillion VND
Terminal Value: 146.1 trillion VND
PV of Terminal Value: 90.7 trillion VND
Enterprise Value: 113.4 trillion VND
Terminal Value as % of EV: 80.0%
```

Note that terminal value often represents 60-80% of enterprise value. This highlights the importance of terminal value assumptions and the inherent uncertainty in DCF analysis.

6.10 Sensitivity Analysis

Given the uncertainty in DCF inputs, sensitivity analysis is essential. We examine how enterprise value changes with different assumptions about WACC and perpetual growth.

```
# Define ranges for sensitivity analysis
wacc_range = np.arange(0.08, 0.14, 0.01)  # 8% to 13%
growth_range = np.arange(0.02, 0.06, 0.01)  # 2% to 5%

# Create all combinations
sensitivity_results = []
```

```
for w in wacc_range:
    for g in growth_range:
        if w > g:  # Must have WACC > growth for valid terminal value
            result = compute_dcf_value(
                fcf_series=forecast_data["fcf"].values,
                wacc=w,
                perpetual_growth=g
            )
            sensitivity_results.append({
                "wacc": w,
                "growth_rate": g,
                "enterprise_value": result["enterprise_value"] / 1e12  # In trillions
            })

sensitivity_df = pd.DataFrame(sensitivity_results)

# Create heatmap
sensitivity_figure = (
    ggplot(sensitivity_df, aes(x="wacc", y="growth_rate", fill="enterprise_value"))
    + geom_tile()
    + geom_text(
        aes(label="enterprise_value"),
        format_string="{:.0f}",
        color="white",
        size=9
    )
    + scale_x_continuous(labels=percent_format())
    + scale_y_continuous(labels=percent_format())
    + scale_fill_gradient(low="darkblue", high="lightblue")
    + labs(
        x="WACC", y="Perpetual Growth Rate",
        fill="EV\n(Trillion VND)",
        title="DCF Sensitivity: Enterprise Value by WACC and Growth Rate"
    )
)

sensitivity_figure.show()
```

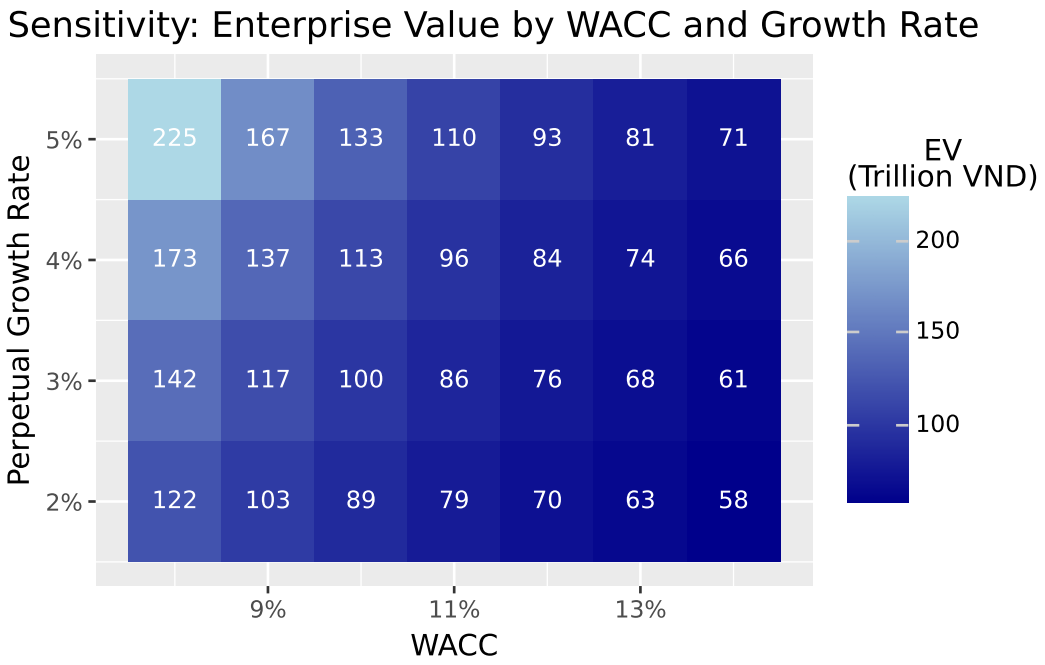


Figure 6.5: Sensitivity of enterprise value to WACC and perpetual growth rate assumptions. Small changes in these inputs can substantially affect valuation.

The sensitivity analysis reveals several important insights:

1. **Valuation is highly sensitive to inputs**: Small changes in WACC or growth rate produce large changes in enterprise value. A 1 percentage point change in WACC can shift value by 20% or more.
2. **The relationship is non-linear**: The impact of growth rate changes is amplified at lower WACCs because the terminal value formula has $(r - g)$ in the denominator.
3. **Reasonable people can disagree**: Given input uncertainty, DCF should be thought of as producing a *range* of values, not a single precise number.

6.11 From Enterprise Value to Equity Value

Our DCF analysis yields **enterprise value** (i.e., the total value of the company's operations to all capital providers). To determine **equity value** (what shareholders own), we must adjust for the claims of debt holders and any non-operating assets:

$$\text{Equity Value} = \text{Enterprise Value} + \text{Non-Operating Assets} - \text{Debt}$$

Non-Operating Assets include:

- Excess cash beyond operating needs
- Marketable securities
- Non-core real estate or investments

Debt includes:

- Short-term debt
- Long-term debt
- Capital lease obligations
- Preferred stock (if treated as debt-like)

```
# Get most recent balance sheet data for FPT
latest_year = fpt_data["year"].max()
latest_data = fpt_data[fpt_data["year"] == latest_year].iloc[0]

# Extract debt and cash (column names may vary)
total_debt = latest_data.get("total_debt", 0)
cash = latest_data.get("ca_cce", 0)

# Compute equity value
enterprise_value = dcf_result["enterprise_value"]
equity_value = enterprise_value - total_debt + cash

print("From Enterprise Value to Equity Value")
print("=" * 50)
print(f"Enterprise Value: {enterprise_value/1e12:.1f} trillion VND")
print(f"Less: Total Debt: {total_debt/1e12:.1f} trillion VND")
print(f"Plus: Cash: {cash/1e12:.1f} trillion VND")
print(f"Equity Value: {equity_value/1e12:.1f} trillion VND")
```

```
From Enterprise Value to Equity Value
==================================================
Enterprise Value: 113.4 trillion VND
Less: Total Debt: 0.0 trillion VND
Plus: Cash: 8.3 trillion VND
Equity Value: 121.7 trillion VND
```

6.11.1 Implied Share Price

If we know the number of shares outstanding, we can compute an implied share price:

```
# Get shares outstanding (this would come from market data)
# Using placeholder - in practice, get from exchange data
shares_outstanding = latest_data.get("total_equity", equity_value) / 25000  # Rough estimate

implied_price = equity_value / shares_outstanding

print(f"\nImplied Share Price: {implied_price:,.0f} VND")
```

```
Implied Share Price: 101,645 VND
```

Comparing the implied price to the current market price tells us whether the stock appears under- or overvalued according to our DCF model.

6.12 Limitations and Practical Considerations

DCF analysis is powerful but has important limitations:

6.12.1 Sensitivity to Assumptions

As our sensitivity analysis showed, small changes in inputs produce large changes in value. This is particularly problematic because the most influential inputs (long-term growth, WACC) are the hardest to estimate accurately.

6.12.2 Terminal Value Dominance

Terminal value often represents 60-80% of total value, yet it's based on assumptions about the very distant future. This concentrates valuation risk in the most uncertain component.

6.12.3 Garbage In, Garbage Out

DCF is only as good as its inputs. Unrealistic growth assumptions, optimistic margins, or inappropriate discount rates produce meaningless valuations. The discipline of DCF lies in forcing analysts to justify their assumptions.

6.12.4 Not Suitable for All Companies

DCF works best for companies with:

- Positive and predictable cash flows
- Stable or predictably changing margins
- Reasonable visibility into future operations

It struggles with:

- Early-stage companies with no profits
- Highly cyclical businesses
- Companies undergoing major transitions
- Financial institutions (which require different approaches)

6.12.5 Complement with Other Methods

Wise practitioners use DCF alongside other valuation methods:

- **Comparable company analysis**: How do similar companies trade?
- **Precedent transactions**: What have acquirers paid for similar businesses?
- **Sum-of-the-parts**: Value divisions separately and add

When methods converge, confidence increases. When they diverge, it prompts investigation into why.

6.13 Key Takeaways

This chapter introduced Discounted Cash Flow analysis as a framework for intrinsic valuation. The main insights are:

1. **Free Cash Flow is the foundation**: FCF represents cash available to all investors after operating expenses, taxes, and investments. It differs from net income by excluding non-cash items and including investment needs.
2. **Ratio-based forecasting links components to revenue**: By expressing FCF components as percentages of revenue, we can systematically forecast cash flows based on revenue growth assumptions and operating ratio projections.
3. **Terminal value captures long-term value**: The perpetuity growth model assumes FCF grows at a constant rate forever. The perpetual growth rate should not exceed long-term economic growth.

4. **WACC is the appropriate discount rate**: The Weighted Average Cost of Capital reflects the blended cost of debt and equity financing, adjusted for the tax shield on interest.
5. **DCF produces enterprise value**: To derive equity value, subtract debt and add non-operating assets. Dividing by shares outstanding yields an implied share price.
6. **Sensitivity analysis is essential**: Given input uncertainty, presenting a range of values based on different assumptions is more honest than a single point estimate.
7. **DCF complements other methods**: No single valuation method is definitive. Cross-checking DCF with market multiples and transaction comparables provides a more complete picture.

The true value of DCF analysis lies not in producing a precise number but in forcing rigorous thinking about what drives company value. The process of building a DCF model (i.e., forecasting growth, projecting margins, estimating risk) develops deep understanding of the business being valued.

7 Accessing and Managing Financial Data

In this chapter, we suggest a way to organize your financial data. Everybody who has experience with data is also familiar with storing data in various formats like CSV, XLS, XLSX, or other delimited value storage. Reading and saving data can become very cumbersome when using different data formats and across different projects. Moreover, storing data in delimited files often leads to problems with respect to column type consistency. For instance, date-type columns frequently lead to inconsistencies across different data formats and programming languages.

First, we load the Python packages that we use throughout this chapter. Later on, we load more packages in the sections where we need them.

```
import pandas as pd
import numpy as np
import io
import re
import zipfile
from curl_cffi import requests
```

Moreover, we initially define the date range for which we fetch and store the financial data, making future data updates tractable. In case you need another time frame, you can adjust the dates below.

```
start_date = "1960-01-01"
end_date = "2024-12-31"
```

7.1 Macroeconomic Predictors

Our next data source is a set of macroeconomic variables often used as predictors for the equity premium. Welch and Goyal (2008) comprehensively reexamine the performance of variables suggested by the academic literature to be good predictors of the equity premium. The authors host the data on Amit Goyal's website. Since the data is an XLSX-file stored on a public Google Drive location, we need additional packages to access the data directly from our Python session. Usually, you need to authenticate if you interact with Google drive directly in Python. Since the data is stored via a public link, we can proceed without any authentication.

```
sheet_id = "1bM7vCWd3WOt95Sf9qjLPZjoiafgF_8EG"
sheet_name = "macro_predictors.xlsx"
macro_predictors_link = (
  f"https://docs.google.com/spreadsheets/d/{sheet_id}"
  f"/gviz/tq?tqx=out:csv&sheet={sheet_name}"
)
```

Next, we read in the new data and transform the columns into the variables that we later use:

1. The dividend price ratio (`dp`), the difference between the log of dividends and the log of prices, where dividends are 12-month moving sums of dividends paid on the S&P 500 index, and prices are monthly averages of daily closing prices (Campbell and Shiller 1988; Campbell and Yogo 2006).
2. Dividend yield (`dy`), the difference between the log of dividends and the log of lagged prices (Ball 1978).
3. Earnings price ratio (`ep`), the difference between the log of earnings and the log of prices, where earnings are 12-month moving sums of earnings on the S&P 500 index (Campbell and Shiller 1988).
4. Dividend payout ratio (`de`), the difference between the log of dividends and the log of earnings (Lamont 1998).
5. Stock variance (`svar`), the sum of squared daily returns on the S&P 500 index (Guo 2006).
6. Book-to-market ratio (`bm`), the ratio of book value to market value for the Dow Jones Industrial Average (Kothari and Shanken 1997).
7. Net equity expansion (`ntis`), the ratio of 12-month moving sums of net issues by NYSE listed stocks divided by the total end-of-year market capitalization of NYSE stocks (Campbell, Hilscher, and Szilagyi 2008).
8. Treasury bills (`tbl`), the 3-Month Treasury Bill: Secondary Market Rate from the economic research database at the Federal Reserve Bank at St. Louis (Campbell 1987).
9. Long-term yield (`lty`), the long-term government bond yield from Ibbotson's Stocks, Bonds, Bills, and Inflation Yearbook (Welch and Goyal 2008).
10. Long-term rate of returns (`ltr`), the long-term government bond returns from Ibbotson's Stocks, Bonds, Bills, and Inflation Yearbook (Welch and Goyal 2008).
11. Term spread (`tms`), the difference between the long-term yield on government bonds and the Treasury bill (Campbell 1987).
12. Default yield spread (`dfy`), the difference between BAA and AAA-rated corporate bond yields (Fama and French 1989).
13. Inflation (`infl`), the Consumer Price Index (All Urban Consumers) from the Bureau of Labor Statistics (Campbell and Vuolteenaho 2004).

For variable definitions and the required data transformations, you can consult the material on Amit Goyal's website.

```
ssl._create_default_https_context = ssl._create_unverified_context

macro_predictors = (
  pd.read_csv(macro_predictors_link, thousands=",")
  .assign(
    date=lambda x: pd.to_datetime(x["yyyymm"], format="%Y%m"),
    dp=lambda x: np.log(x["D12"])-np.log(x["Index"]),
    dy=lambda x: np.log(x["D12"])-np.log(x["Index"].shift(1)),
    ep=lambda x: np.log(x["E12"])-np.log(x["Index"]),
    de=lambda x: np.log(x["D12"])-np.log(x["E12"]),
    tms=lambda x: x["lty"]-x["tbl"],
    dfy=lambda x: x["BAA"]-x["AAA"]
  )
  .rename(columns={"b/m": "bm"})
  .get(["date", "dp", "dy", "ep", "de", "svar", "bm",
        "ntis", "tbl", "lty", "ltr", "tms", "dfy", "infl"])
  .query("date >= @start_date and date <= @end_date")
  .dropna()
)

ssl._create_default_https_context = ssl.create_default_context
```

7.2 Other Macroeconomic Data

The Federal Reserve bank of St. Louis provides the Federal Reserve Economic Data (FRED), an extensive database for macroeconomic data. In total, there are 817,000 US and international time series from 108 different sources. As an illustration, we use the `tidyfinance` package to fetch consumer price index (CPI) data that can be found under the CPIAUCNS key.

```
series = "CPIAUCNS"
url = f"https://fred.stlouisfed.org/graph/fredgraph.csv?id={series}"
```

We can then use the `requests` module to request the CSV, extract the data from the response body, and convert the columns to a tidy format:

```
resp = requests.get(url)
resp_csv = pd.io.common.StringIO(resp.text)

cpi_monthly = (pd.read_csv(resp_csv)
  .assign(
```

```
    date=lambda x: pd.to_datetime(x["observation_date"]),
    value=lambda x: pd.to_numeric(
      x[series], errors="coerce"
    ),
      series=series,
   )
  .get(["date", "series", "value"])
  .query("date >= @start_date & date <= @end_date")
  .assign(cpi=lambda x: x["value"] / x["value"].iloc[-1])
)
```

The last line sets the current (latest) price level as the reference price level.

To download other time series, we just have to look it up on the FRED website and extract the corresponding key from the address. For instance, the producer price index for gold ores can be found under the PCU2122212122210 key. If your desired time series is not supported through tidyfinance, we recommend working with the `fredapi` package. Note that you need to get an API key to use its functionality. We refer to the package documentation for details.

7.3 Setting Up a Database

Now that we have downloaded some (freely available) data from the web into the memory of our Python session, let us set up a database to store that information for future use. We will use the data stored in this database throughout the following chapters, but you could alternatively implement a different strategy and replace the respective code.

There are many ways to set up and organize a database, depending on the use case. For our purpose, the most efficient way is to use an SQLite-database, which is the C-language library that implements a small, fast, self-contained, high-reliability, full-featured SQL database engine. Note that SQL (Structured Query Language) is a standard language for accessing and manipulating databases.

```
import sqlite3
```

An SQLite-database is easily created - the code below is really all there is. You do not need any external software. Otherwise, date columns are stored and retrieved as integers. We will use the file `tidy_finance_r.sqlite`, located in the data subfolder, to retrieve data for all subsequent chapters. The initial part of the code ensures that the directory is created if it does not already exist.

```
import os

if not os.path.exists("data"):
  os.makedirs("data")

tidy_finance = sqlite3.connect(database="data/tidy_finance_python.sqlite")
```

Next, we create a remote table with the monthly Fama-French factor data. We do so with the `pandas` function `to_sql()`, which copies the data to our SQLite-database.

```
(factors_ff3_monthly
  .to_sql(name="factors_ff3_monthly",
          con=tidy_finance,
          if_exists="replace",
          index=False)
)
```

Now, if we want to have the whole table in memory, we need to call `pd.read_sql_query()` with the corresponding query. You will see that we regularly load the data into the memory in the next chapters.

```
pd.read_sql_query(
  sql="SELECT date, risk_free FROM factors_ff3_monthly",
  con=tidy_finance,
  parse_dates={"date"}
)
```

The last couple of code chunks are really all there is to organizing a simple database! You can also share the SQLite database across devices and programming languages.

Before we move on to the next data source, let us also store the other six tables in our new SQLite database.

```
data_dict = {
  "factors_ff5_monthly": factors_ff5_monthly,
  "factors_ff3_daily": factors_ff3_daily,
  "industries_ff_monthly": industries_ff_monthly,
  "factors_q_monthly": factors_q_monthly,
  "macro_predictors": macro_predictors,
  "cpi_monthly": cpi_monthly
}
```

```
for key, value in data_dict.items():
    value.to_sql(name=key,
                 con=tidy_finance,
                 if_exists="replace",
                 index=False)
```

From now on, all you need to do to access data that is stored in the database is to follow two steps: (i) Establish the connection to the SQLite-database and (ii) execute the query to fetch the data. For your convenience, the following steps show all you need in a compact fashion.

```
import pandas as pd
import sqlite3

tidy_finance = sqlite3.connect(database="data/tidy_finance_python.sqlite")

factors_q_monthly = pd.read_sql_query(
  sql="SELECT * FROM factors_q_monthly",
  con=tidy_finance,
  parse_dates={"date"}
)
```

7.4 Managing SQLite Databases

Finally, at the end of our data chapter, we revisit the SQLite database itself. When you drop database objects such as tables or delete data from tables, the database file size remains unchanged because SQLite just marks the deleted objects as free and reserves their space for future uses. As a result, the database file always grows in size.

To optimize the database file, you can run the `VACUUM` command in the database, which rebuilds the database and frees up unused space. You can execute the command in the database using the `execute()` function.

```
tidy_finance.execute("VACUUM")
```

The `VACUUM` command actually performs a couple of additional cleaning steps, which you can read about in this tutorial.

7.5 Key Takeaways

- Importing Fama-French factors, q-factors, macroeconomic indicators, and CPI data is simplified through API calls, CSV parsing, and web scraping techniques.
- The `tidyfinance` Python package offers pre-processed access to financial datasets, reducing manual data cleaning and saving valuable time.
- Creating a centralized SQLite database helps manage and organize data efficiently across projects, while maintaining reproducibility.
- Structured database storage supports scalable data access, which is essential for long-term academic projects and collaborative work in finance.

8 DataCore Data

This chapter shows how to connect to DataCore a provider of financial and economic data for research applications. We use this connection to download the most commonly used data for stock and firm characteristics (i.e., Stock Data and Public Companies). Unfortunately, this data is not freely available, but most students and researchers typically have access to DataCore through their university libraries. Assuming that you have access to DataCore, we show you how to prepare and merge the databases and store them in the SQLite database introduced in the previous chapter. We conclude this chapter by providing some tips for working with the DataCore database.

If you don't have access to DataCore but still want to run the code in this book, we refer to DataCore Demo Data, where we show how to create a dummy database that contains the DataCore tables and corresponding columns. With this database at hand, all code chunks in this book can be executed with this dummy database.

First, we load the Python packages that we use throughout this chapter. Later on, we load more packages in the sections where we need them. The last two packages are used for plotting.

```
import pandas as pd
import numpy as np
import tidyfinance as tf
import sqlite3

from plotnine import *
from mizani.formatters import comma_format, percent_format
from datetime import datetime
tidy_finance = sqlite3.connect(database="data/tidy_finance_python.sqlite")
```

We use the same date range as in the previous chapter to ensure consistency. However, we have to use the date format that the DataCore database expects.

```
start_date = "01/01/1960"
end_date = "12/31/2024"
```

8.1 Accessing DataCore

DataCore is the most widely used source for asset and firm-specific financial data used in academic settings. DataCore is a data platform that provides data validation, flexible delivery options, and access to many different data sources.

8.2 Preparing Company Fundamentals Data

Firm accounting data are an important source of information that we use in portfolio analyses in subsequent chapters.

To access Company Fundamentals data, we can again tap DataCore, which hosts the `funda` data table that contains annual firm-level information on Vietnam companies. We follow the typical filter conventions and pull only data that we actually need: (i) we get only records in industrial data format, which includes companies that are primarily involved in manufacturing, services, and other non-financial business activities,[1], (ii) in the standard format (i.e., consolidated information in standard presentation).

```
import pandas as pd
from io import BytesIO
import datetime as dt

import os
import boto3
from botocore.client import Config

class ConnectMinio:
    def __init__(self):
        self.MINIO_ENDPOINT = os.environ["MINIO_ENDPOINT"]
        self.MINIO_ACCESS_KEY = os.environ["MINIO_ACCESS_KEY"]
        self.MINIO_SECRET_KEY = os.environ["MINIO_SECRET_KEY"]
        self.REGION = os.getenv("MINIO_REGION", "us-east-1")

        self.s3 = boto3.client(
            "s3",
            endpoint_url=self.MINIO_ENDPOINT,
            aws_access_key_id=self.MINIO_ACCESS_KEY,
            aws_secret_access_key=self.MINIO_SECRET_KEY,
```

[1]Note that Fama and French (1992) claim to exclude financial firms. To a large extent this happens through using industry format "INDL". Neither the original paper, nor Ken French's website, or the WRDS replication contains any indication that financial companies are excluded using additional filters such as industry codes.

```
            region_name=self.REGION,
            config=Config(signature_version="s3v4"),
        )

    def test_connection(self):
        resp = self.s3.list_buckets()
        print("Connected. Buckets:")
        for b in resp.get("Buckets", []):
            print(" -", b["Name"])


conn = ConnectMinio()
s3 = conn.s3
conn.test_connection()

bucket_name = os.environ["MINIO_BUCKET"]

paths = [
    "fundamental_annual_1767674486317/fundamental_annual_1.xlsx",
    "fundamental_annual_1767674486317/fundamental_annual_2.xlsx",
    "fundamental_annual_1767674486317/fundamental_annual_3.xlsx",
]

dfs = []
for key in paths:
    obj = s3.get_object(Bucket=bucket_name, Key=key)
    df_tmp = pd.read_excel(BytesIO(obj["Body"].read()))
    dfs.append(df_tmp)

df_company_fundamental = pd.concat(dfs, ignore_index=True)
```

```
Connected. Buckets:
 - dsteam-data
 - rawbctc
```

```
df = df_company_fundamental.copy()

# core keys
df["symbol"] = df["symbol"].astype(str).str.upper().str.strip()
df["year"] = pd.to_numeric(df["year"], errors="coerce").astype("Int64")
```

```
# drop rows with missing keys
df = df.dropna(subset=["symbol", "year"])

# if some numeric columns are objects, force numeric for the ones we will use
need = [
    "total_asset",
    "total_equity",
    "total_liabilities",
    "total_current_liabilities",
    "is_net_revenue",
    "is_cogs",
    "is_manage_expense",
    "is_interest_expense",
    "na_tax_deferred",
    "nl_tax_deferred",
    "e_preferred_stock",
    "capex",
    "total_cfo",
    "is_eat",
    "total_current_asset",
    "ca_cce",
    "total_equity",
    "cfo_interest_expense",
    "ca_total_inventory",
    "ca_acc_receiv",
    # The is_net_business_profit field captures the core profitability of a company's busines
    "is_net_business_profit"
]
for c in need:
    if c in df.columns:
        df[c] = pd.to_numeric(df[c], errors="coerce")

# Keep the row with the most non missing fields.
df["_non_missing"] = df.notna().sum(axis=1)

df = (
    df.sort_values(["symbol", "year", "_non_missing"])
      .drop_duplicates(subset=["symbol", "year"], keep="last")
      .drop(columns="_non_missing")
      .reset_index(drop=True)
)
```

```
print("Remaining duplicates:",
      df.duplicated(["symbol", "year"]).sum())
```

```
Remaining duplicates: 0
```

```
# preferably use Tax ID as Identifer
# ---- Firm identifier and fiscal date ----
df["datadate"] = pd.to_datetime(df["year"].astype(str) + "-12-31")  # Fiscal year-end date

# ---- Core balance sheet ----
df["at"]  = df["total_asset"]                                    # Total assets
df["act"] = df["total_current_asset"] # Total Current Assets

df["lt"]  = df["total_liabilities"]                               # Total liabilities
df["lct"] = df["total_current_liabilities"] # Total Current Liabilities
df["seq"] = df["total_equity"]                                    # Stockholders' equity
df["ceq"] = df["e_equity"] if "e_equity" in df.columns else np.nan  # Common equity (fallbac

# ---- Deferred taxes ----
df["txditc"] = df["na_tax_deferred"] if "na_tax_deferred" in df.columns else 0  # Deferred ta
df["txdb"]   = df["nl_tax_deferred"] if "nl_tax_deferred" in df.columns else 0  # Deferred ta
df["itcb"]   = 0                                                                # Investment

# ---- Preferred stock (Compustat has multiple versions, we map one) ----
pref = df["e_preferred_stock"] if "e_preferred_stock" in df.columns else 0
df["pstk"]   = pref
df["pstkrv"] = pref
df["pstkl"]  = pref

# ---- Income statement ----
df["sale"] = df["is_net_revenue"]                                          # Sales
df["cogs"] = df["is_cogs"] if "is_cogs" in df.columns else 0                # Cost of goo
df["xsga"] = df["is_manage_expense"] if "is_manage_expense" in df.columns else 0# SG&A proxy
df["xint"] = df["is_interest_expense"] if "is_interest_expense" in df.columns else 0  # Inte

# ---- Cash flow and investment ----
df["oancf"] = df["total_cfo"] if "total_cfo" in df.columns else np.nan  # Operating cash flo
df["capx"]  = df["capex"] if "capex" in df.columns else np.nan          # Capital expenditure

comp_vn = df

comp_vn.head()
```

	symbol	year	total_current_asset	ca_fin	ca_cce	ca_cash	ca_cash_inbank	ca_cash
0	A32	2016	3.432787e+11	NaN	1.397735e+11	1.006804e+10	NaN	NaN
1	A32	2017	3.741267e+11	NaN	1.456583e+11	5.095282e+10	NaN	NaN
2	A32	2018	3.358630e+11	NaN	5.829081e+10	1.229081e+10	NaN	NaN
3	A32	2019	2.987680e+11	NaN	6.051375e+10	4.451375e+10	NaN	NaN
4	A32	2020	3.566913e+11	NaN	4.435908e+10	2.235908e+10	NaN	NaN

```
# Keep only firm-years with core fundamentals present

# Required for most accounting ratios
req = ["at", "lt", "seq", "sale"]

comp_vn = comp_vn.dropna(subset=req)
comp_vn = comp_vn[comp_vn["at"] > 0]          # assets must be positive
comp_vn = comp_vn[comp_vn["sale"] >= 0]       # sales cannot be negative

# Quick diagnostics
print("Rows:", len(comp_vn))
print("Firms:", comp_vn["symbol"].nunique())
print("Years:", comp_vn["datadate"].dt.year.min(), "-", comp_vn["datadate"].dt.year.max())
```

```
Rows: 20091
Firms: 1502
Years: 1998 - 2023
```

Next, we calculate the book value of preferred stock and equity `be` and the operating profitability `op` inspired by the variable definitions in Kenneth French's data library. Note that we set negative or zero equity to missing, which is a common practice when working with book-to-market ratios (see Fama and French 1992 for details).

```
comp_vn = (comp_vn
  .assign(
    be=lambda x:
      (x["seq"].combine_first(x["ceq"]+x["pstk"])
       .combine_first(x["at"]-x["lt"])+
       x["txditc"].combine_first(x["txdb"]+x["itcb"]).fillna(0)-
       x["pstkrv"].combine_first(x["pstkl"])
       .combine_first(x["pstk"]).fillna(0))
  )
  .assign(
```

```
    be=lambda x: x["be"].apply(lambda y: np.nan if y <= 0 else y)
  )
  .assign(
    op=lambda x:
      ((x["sale"]-x["cogs"].fillna(0)-
        x["xsga"].fillna(0)-x["xint"].fillna(0))/x["be"])
  )
)
```

We keep only the last available information for each firm-year group (by using the `tail(1)` pandas function for each group). Note that `datadate` defines the time the corresponding financial data refers to (e.g., annual report as of December 31, 2022). Therefore, `datadate` is not the date when data was made available to the public. Check out the Exercises for more insights into the peculiarities of `datadate`.

```
comp_vn = (comp_vn
  .assign(year=lambda x: pd.DatetimeIndex(x["datadate"]).year)
  .sort_values("datadate")
  .groupby(["symbol", "year"])
  .tail(1)
  .reset_index(drop=True)
)
```

```
comp_vn_lag = (comp_vn
  .get(["symbol", "year", "at"])
  .assign(year=lambda x: x["year"]+1)
  .rename(columns={"at": "at_lag"})
)

comp_vn = (comp_vn
  .merge(comp_vn_lag, how="left", on=["symbol", "year"])
  .assign(inv=lambda x: x["at"]/x["at_lag"]-1)
  .assign(inv=lambda x: np.where(x["at_lag"] <= 0, np.nan, x["inv"]))
)
```

In a standard Vietnamese financial report, the **LIABILITIES** section (`total_liabilities`) includes all forms of debt, including non-interest-bearing items like **Accounts Payable** (`cl_acc_payable`) and **Taxes payable** (`cl_tax_state_payable`).

To get what financial analysts call "Total Debt" (interest-bearing debt), you must manually aggregate the specific loan and lease variables from your dataset:

- **Short-term interest-bearing debt**: Sum of `cl_loan` and `cl_finlease`.
- **Current portion of long-term debt**: `cl_due_long_debt`.
- **Long-term interest-bearing debt**: Sum of `nl_loan` and `nl_finlease`.

```
comp_vn = comp_vn.assign(
    total_debt = lambda x: (
        x["cl_loan"].fillna(0) +
        x["cl_finlease"].fillna(0) +
        x["cl_due_long_debt"].fillna(0) +
        x["nl_loan"].fillna(0) +
        x["nl_finlease"].fillna(0)
    ),
    selling_general_and_administrative_expenses = lambda x: (
        x["is_cos_of_sales"].fillna(0) + x["is_manage_expense"].fillna(0)
    )
)
```

With the last step, we are already done preparing the firm fundamentals. Thus, we can store them in our local database.

```
(comp_vn
  .to_sql(name="comp_vn",
          con=tidy_finance,
          if_exists="replace",
          index=False)
)
```

8.3 Downloading and Preparing Stock Data

```
import pandas as pd
from io import BytesIO
import datetime as dt

import os
import boto3
from botocore.client import Config

class ConnectMinio:
```

```
    def __init__(self):
        self.MINIO_ENDPOINT = os.environ["MINIO_ENDPOINT"]
        self.MINIO_ACCESS_KEY = os.environ["MINIO_ACCESS_KEY"]
        self.MINIO_SECRET_KEY = os.environ["MINIO_SECRET_KEY"]
        self.REGION = os.getenv("MINIO_REGION", "us-east-1")

        self.s3 = boto3.client(
            "s3",
            endpoint_url=self.MINIO_ENDPOINT,
            aws_access_key_id=self.MINIO_ACCESS_KEY,
            aws_secret_access_key=self.MINIO_SECRET_KEY,
            region_name=self.REGION,
            config=Config(signature_version="s3v4"),
        )

    def test_connection(self):
        resp = self.s3.list_buckets()
        print("Connected. Buckets:")
        for b in resp.get("Buckets", []):
            print(" -", b["Name"])


conn = ConnectMinio()
s3 = conn.s3
conn.test_connection()

bucket_name = os.environ["MINIO_BUCKET"]

prices = pd.read_csv(
    BytesIO(
        s3.get_object(
            Bucket=bucket_name,
            Key="historycal_price/dataset_historical_price.csv"
        )["Body"].read()
    ),
    low_memory=False
)

prices["date"] = pd.to_datetime(prices["date"])

prices["adjusted_close"] = prices["close_price"] * prices["adj_ratio"]
```

```
prices = prices.rename(columns={
    "vol_total": "volume",
    "open_price": "open",
    "low_price": "low",
    "high_price": "high",
    "close_price": "close"
})

prices = prices.sort_values(["symbol", "date"])
```

```
Connected. Buckets:
 - dsteam-data
 - rawbctc
```

```
prices = prices.sort_values(["symbol", "date"])

prices["ret"] = (
    prices.groupby("symbol")["adjusted_close"]
    .pct_change()
)

# Remove impossible crashes beyond -100 percent
prices["ret"] = prices["ret"].clip(lower=-0.99)
```

Now, we have all the relevant daily price data in memory and proceed with preparing the data for future analyses. We perform the preparation step at the current stage since we want to avoid executing the same mutations every time we use the data in subsequent chapters.

The first additional variable we create is market capitalization (`mktcap`), which is the product of the number of outstanding shares (`shrout`) and the last traded price in a month (`prc`). Note that in contrast to returns (`ret`), these two variables are not adjusted ex-post for any corporate actions like stock splits. Therefore, if you want to use a stock's price, you need to adjust it with a cumulative adjustment factor. We also keep the market cap in millions of VND just for convenience, as we do not want to print huge numbers in our figures and tables. In addition, we set zero market capitalization to missing as it makes conceptually little sense (i.e., the firm would be bankrupt).

```
# 1. Calculate the Share Count Proxy from Annual Fundamentals
# df represents your fundamental_annual dataset
df["shrout"] = (
    df["is_shareholders_eat"] / df["basic_eps"]
```

```
)

# 2. Merge with Daily Prices
# We join on 'symbol' and 'year' so every daily row in 'prices'
# receives the share count corresponding to that fiscal year.
prices = prices.merge(
    df[["symbol", "year", "shrout"]],
    on=["symbol", "year"],
    how="left"
)

# 3. Calculate Daily Market Cap
# Market Cap = Daily Close Price * Annual Shares Outstanding
# We use 'close' (unadjusted) to reflect the actual market value.
prices["mktcap"] = prices["close"] * prices["shrout"]

# 4. Filter and Scale
# scales values to millions for readability
prices["mktcap"] = (prices["mktcap"] / 1000000).replace(0, np.nan)
```

```
# Resample to Monthly Frequency
# We group by symbol and resample the date to Month End (ME).
# .last() picks the final available data point for each month.
prices_monthly = (
    prices.sort_values(["symbol", "date"])
    .groupby("symbol")
    .resample("ME", on="date")
    .last()
)

# After .last(), 'symbol' is index level 0 and 'date' is index level 1.
# We remove 'symbol' and 'date' from the index to make them regular columns.
prices_monthly = prices_monthly.drop(columns=["symbol", "date"], errors="ignore").reset_index
```

The next variable we frequently use is the one-month *lagged* market capitalization. Lagged market capitalization is typically used to compute value-weighted portfolio returns, as we demonstrate in a later chapter. The most simple and consistent way to add a column with lagged market cap values is to add one month to each observation and then join the information to our monthly data.

```
# 3. Lagged Market Capitalization
# We calculate the market cap at t-1 to use for weighting returns at t.
prices_monthly["mktcap_lag"] = (
    prices_monthly.groupby("symbol")["mktcap"]
    .shift(1)
)
```

```
# remove rows missing returns or capitalization data.
prices_monthly = (
    prices_monthly
    .dropna(subset=["ret", "mktcap", "mktcap_lag"])
)
```

Next, we transform primary listing exchange codes to explicit exchange names.

```
def map_vn_exchange(row):
    # This assumes you have an exchange column or can derive it from the symbol
    # Adjust the logic based on your specific metadata availability.
    if hasattr(row, 'exchange_code'):
        if row.exchange_code == "HOSE": return "NYSE_Equiv"
        if row.exchange_code == "HNX": return "AMEX_Equiv"
    return "Other"
```

Next, we compute excess returns by subtracting the monthly risk-free rate provided by our Fama-French data. As we base all our analyses on the excess returns, we can drop the risk-free rate from our data frame. Note that we ensure excess returns are bounded by -1 from below as a return less than -100% makes no sense conceptually.

To implement a placeholder for the Vietnam risk-free rate, we use an annualized value of 4.0% (0.04), which closely approximates the 10-year Vietnam Government Bond yield observed in early 2026. In professional asset pricing research for emerging markets, the 1-year or 10-year government bond rate is a standard proxy when a dedicated monthly Fama-French risk-free rate is unavailable.

```
# Create a date range covering your prices_monthly sample
all_dates = pd.date_range(
    start=prices_monthly['date'].min(),
    end=prices_monthly['date'].max(),
    freq='ME'
)

# Professional Proxy: 4% Annualized (0.04)
```

```python
# Monthly rate = 0.04 / 12
annual_rf = 0.04
monthly_rf = annual_rf / 12

rf_monthly = pd.DataFrame({
    'date': all_dates,
    'risk_free': monthly_rf
})

prices_monthly = (prices_monthly
  # we don't have Fama-French data for Vietnam
  .merge(rf_monthly, how="left", on="date")
  .assign(ret_excess=lambda x: x["ret"]-x["risk_free"])
  .assign(ret_excess=lambda x: x["ret_excess"].clip(lower=-1))
  # .drop(columns=["risk_free"])
)

# Clean the infinite values before replication
prices_monthly = prices_monthly.replace([np.inf, -np.inf], np.nan).dropna(subset=['ret_excess
```

```python
# Check for non-finite values (NaN or Inf)
print("Missing or Infinite Values:")
print(prices_monthly[['ret_excess', 'mktcap_lag']].isna().sum())
print(np.isinf(prices_monthly['ret_excess']).sum())

# Basic summary statistics to catch outliers
print("\nRet_excess Summary Statistics:")
print(prices_monthly['ret_excess'].describe())
```

```
Missing or Infinite Values:
ret_excess    0
mktcap_lag    0
dtype: int64
0

Ret_excess Summary Statistics:
count    165468.000000
mean         -0.001458
std           0.041601
min          -0.993333
25%          -0.006044
```

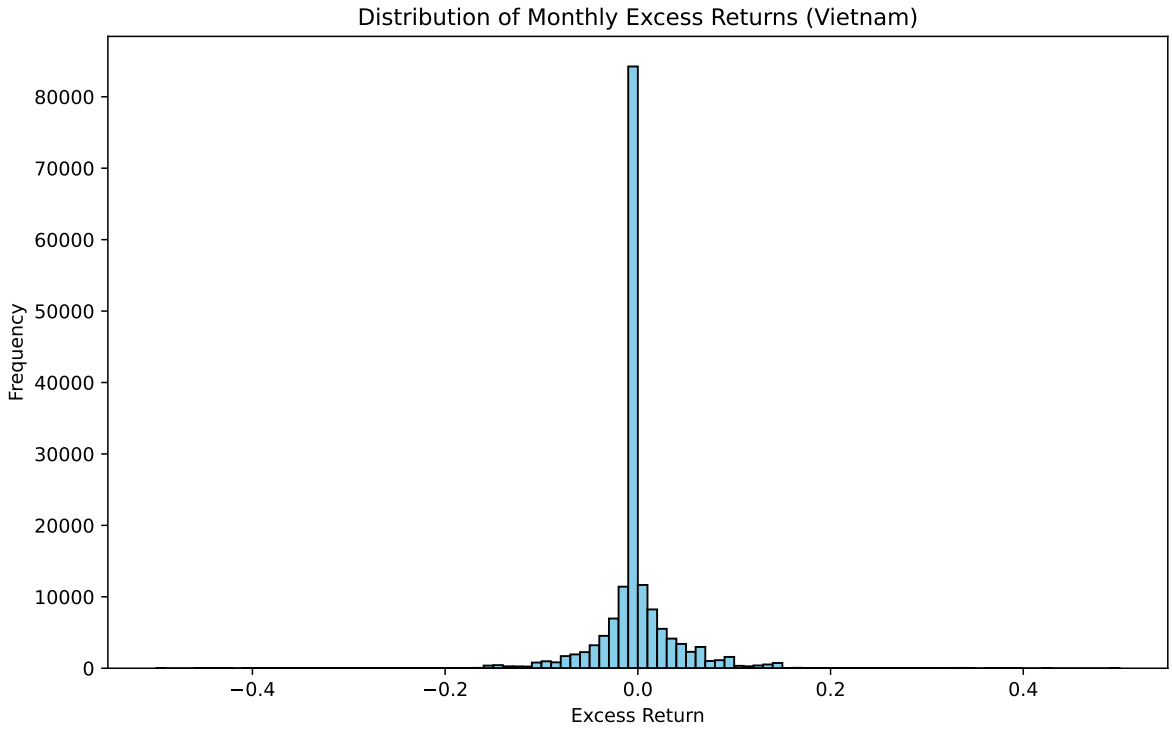
```
50%          -0.003333
75%           0.002764
max           4.139524
Name: ret_excess, dtype: float64
```

Interpretation: Excess returns in a monthly dataset should typically fall between -1.0 and 1.0. If your max value is extremely high, it indicates an error in your return calculation or the risk-free rate merge. Any inf values will cause the np.average in your replication code to fail.

```
import matplotlib.pyplot as plt

# Clean data for plotting
plot_data = prices_monthly['ret_excess'].dropna()
plot_data = plot_data[np.isfinite(plot_data)]

# Histogram with fixed range
plt.figure(figsize=(10, 6))
plt.hist(plot_data, bins=100, range=(-0.5, 0.5), color='skyblue', edgecolor='black')
plt.title('Distribution of Monthly Excess Returns (Vietnam)')
plt.xlabel('Excess Return')
plt.ylabel('Frequency')
plt.show()
```



Interpretation: A healthy distribution should be centered slightly above 0 and look roughly "bell-shaped" but with fatter tails (kurtosis). If the distribution is heavily skewed toward one side, check if your monthly_rf_placeholder was applied correctly.

Since excess returns and market capitalization are crucial for all our analyses, we can safely exclude all observations with missing returns or market capitalization.

```
prices_monthly = (prices_monthly
  .dropna(subset=["ret_excess", "mktcap", "mktcap_lag"])
)
```

Finally, we store the monthly Stock prices data file in our database.

```
(prices_monthly
  .to_sql(name="prices_monthly",
          con=tidy_finance,
          if_exists="replace",
          index=False)
)
```

8.4 First Glimpse of the Stock Sample

Before we move on to other data sources, let us look at some descriptive statistics of the Stock sample, which is our main source for stock returns.

Figure 8.1 shows the monthly number of securities by listing exchange over time. NYSE has the longest history in the data, but NASDAQ lists a considerably large number of stocks.

```
securities_per_exchange = (prices_monthly
  .groupby(["exchange", "date"])
  .size()
  .reset_index(name="n")
)

securities_per_exchange_figure = (
  ggplot(
    securities_per_exchange,
    aes(x="date", y="n", color="exchange", linetype="exchange")
  )
  + geom_line()
  + labs(
      x="", y="", color="", linetype="",
      title="Monthly number of securities by listing exchange"
    )
  + scale_x_datetime(date_breaks="10 years", date_labels="%Y")
  + scale_y_continuous(labels=comma_format())
)

securities_per_exchange_figure.show()
```

Figure 8.1

Next, we look at the aggregate market capitalization grouped by the respective listing exchanges in Figure 8.2. To ensure that we look at meaningful data that is comparable over time, we adjust the nominal values for inflation. In fact, we can use the tables that are already in our database to calculate aggregate market caps by listing exchange. All values in Figure 8.2 are in terms of the end of 2024 USD to ensure intertemporal comparability. NYSE-listed stocks have by far the largest market capitalization, followed by NASDAQ-listed stocks.

Next, we look at the same descriptive statistics by industry. Figure 8.3 plots the number of stocks in the sample for each of the VSIC industry classifiers. For most of the sample period, the largest share of stocks is in manufacturing, albeit the number peaked somewhere in the 90s.

```
cpi_monthly = pd.read_sql_query(
  sql="SELECT * FROM cpi_monthly",
  con=tidy_finance,
  parse_dates={"date"}
)

market_cap_per_exchange = (prices_monthly
  .merge(cpi_monthly, how="left", on="date")
  .groupby(["date", "exchange"])
  .apply(
    lambda group: pd.Series({
      "mktcap": group["mktcap"].sum()/group["cpi"].mean()
    })
  )
  .reset_index()
)

market_cap_per_exchange_figure = (
  ggplot(
    market_cap_per_exchange,
    aes(x="date", y="mktcap/1000", color="exchange", linetype="exchange")
  )
  + geom_line()
  + labs(
      x="", y="", color="", linetype="",
      title="Monthly market cap by listing exchange"
    )
  + scale_x_datetime(date_breaks="10 years", date_labels="%Y")
  + scale_y_continuous(labels=comma_format())
)
market_cap_per_exchange_figure.show()
```

Figure 8.2

The number of firms associated with public administration seems to be the only category on the rise in recent years, even surpassing manufacturing at the end of our sample period.

```
securities_per_industry = (prices_monthly
  .groupby(["industry", "date"])
  .size()
  .reset_index(name="n")
)

linetypes = ["-", "--", "-.", ":"]
n_industries = securities_per_industry["industry"].nunique()

securities_per_industry_figure = (
  ggplot(
    securities_per_industry,
    aes(x="date", y="n", color="industry", linetype="industry")
  )
  + geom_line()
  + labs(
      x="", y="", color="", linetype="",
      title="Monthly number of securities by industry"
    )
  + scale_x_datetime(date_breaks="10 years", date_labels="%Y")
  + scale_y_continuous(labels=comma_format())
  + scale_linetype_manual(
      values=[linetypes[l % len(linetypes)] for l in range(n_industries)]
    )
)
securities_per_industry_figure.show()
```

Figure 8.3

We also compute the market cap of all stocks belonging to the respective industries and show the evolution over time in Figure 8.4. At all points in time, manufacturing firms comprise of the largest portion of market capitalization. Toward the end of the sample, however, financial firms and services begin to make up a substantial portion of the market cap.

8.5 Daily Stock Data

Before we turn to accounting data, we provide a proposal for downloading daily Stock data with the same filters used for the monthly data. While the monthly data from above typically

```
market_cap_per_industry = (prices_monthly
  .merge(cpi_monthly, how="left", on="date")
  .groupby(["date", "industry"])
  .apply(
    lambda group: pd.Series({
      "mktcap": (group["mktcap"].sum()/group["cpi"].mean())
    })
  )
  .reset_index()
)

market_cap_per_industry_figure = (
  ggplot(
    market_cap_per_industry,
    aes(x="date", y="mktcap/1000", color="industry", linetype="industry")
  )
  + geom_line()
  + labs(
      x="", y="", color="", linetype="",
      title="Monthly market cap by industry in billions of Dec 2024 USD"
    )
  + scale_x_datetime(date_breaks="10 years", date_labels="%Y")
  + scale_y_continuous(labels=comma_format())
  + scale_linetype_manual(
      values=[linetypes[l % len(linetypes)] for l in range(n_industries)]
    )
)
market_cap_per_industry_figure.show()
```

Figure 8.4

fit into your memory and can be downloaded in a meaningful amount of time, this is usually not true for daily return data. The daily Stock data file is substantially larger than monthly data. This has two important implications: you cannot hold all the daily return data in your memory (hence it is not possible to copy the entire dataset to your local database), and in our experience, the download usually crashes (or never stops).

There is a solution to this challenge. As with many *big data* problems, you can split up the big task into several smaller tasks that are easier to handle. That is, instead of downloading data about all stocks at once, download the data in small batches of stocks consecutively. Such operations can be implemented in `for`-loops, where we download, prepare, and store the data for a small number of stocks in each iteration. This operation might nonetheless take around 5 minutes, depending on your internet connection. To keep track of the progress, we create ad-hoc progress updates using `print()`. Notice that we also use the method `to_sql()` here with the option to append the new data to an existing table, when we process the second and all following batches.

```
factors_ff3_daily = pd.read_sql(
  sql="SELECT * FROM factors_ff3_daily",
  con=tidy_finance,
  parse_dates={"date"}
)

permnos = pd.read_sql(
  sql="SELECT DISTINCT permno FROM crsp.stksecurityinfohist",
  con=DataCore,
  dtype={"permno": int}
)

permnos = list(permnos["permno"].astype(str))

batch_size = 500
batches = np.ceil(len(permnos)/batch_size).astype(int)

for j in range(1, batches+1):

  permno_batch = permnos[
    ((j-1)*batch_size):(min(j*batch_size, len(permnos)))
  ]

  permno_batch_formatted = (
    ", ".join(f"'{permno}'" for permno in permno_batch)
  )
  permno_string = f"({permno_batch_formatted})"
```

```
crsp_daily_sub_query = (
  "SELECT dsf.permno, dlycaldt AS date, dlyret AS ret "
    "FROM crsp.dsf_v2 AS dsf "
    "INNER JOIN crsp.stksecurityinfohist AS ssih "
    "ON dsf.permno = ssih.permno AND "
       "ssih.secinfostartdt <= dsf.dlycaldt AND "
       "dsf.dlycaldt <= ssih.secinfoenddt "
    f"WHERE dsf.permno IN {permno_string} "
         f"AND dlycaldt BETWEEN '{start_date}' AND '{end_date}' "
          "AND ssih.sharetype = 'NS' "
          "AND ssih.securitytype = 'EQTY' "
          "AND ssih.securitysubtype = 'COM' "
          "AND ssih.usincflg = 'Y' "
          "AND ssih.issuertype in ('ACOR', 'CORP') "
          "AND ssih.primaryexch in ('N', 'A', 'Q') "
          "AND ssih.conditionaltype in ('RW', 'NW') "
          "AND ssih.tradingstatusflg = 'A'"
)

crsp_daily_sub = (pd.read_sql_query(
    sql=crsp_daily_sub_query,
    con=DataCore,
    dtype={"permno": int},
    parse_dates={"date"}
  )
  .dropna()
 )

if not crsp_daily_sub.empty:

    crsp_daily_sub = (crsp_daily_sub
      .merge(factors_ff3_daily[["date", "risk_free"]],
             on="date", how="left")
      .assign(
        ret_excess = lambda x:
          ((x["ret"] - x["risk_free"]).clip(lower=-1))
      )
      .get(["permno", "date", "ret_excess"])
    )

    if j == 1:
      if_exists_string = "replace"
```

```
    else:
      if_exists_string = "append"

    crsp_daily_sub.to_sql(
      name="crsp_daily",
      con=tidy_finance,
      if_exists=if_exists_string,
      index=False
    )

  print(f"Batch {j} out of {batches} done ({(j/batches)*100:.2f}%)\n")
```

8.6 Merging Stock with Company Fundamentals

To link the two datasets, we need to use the stock symbol.

Before we close this chapter, let us look at an interesting descriptive statistic of our data. As the book value of equity plays a crucial role in many asset pricing applications, it is interesting to know for how many of our stocks this information is available. Hence, Figure 8.5 plots the share of securities with book equity values for each exchange.

```
share_with_be = (prices_monthly
  .assign(year=lambda x: pd.DatetimeIndex(x["date"]).year)
  .sort_values("date")
  .groupby(["symbol", "year"])
  .tail(1)
  .reset_index()
  .merge(comp_vn, how="left", on=["symbol", "year"])

  # Group by year only for now; easy to add "exchange" later
  .groupby(["year"])
  # # .groupby(["exchange", "year"]) # Uncomment this later
  .apply(
    lambda x: pd.Series({
      "share": x["symbol"][x["be"].notnull()].nunique() / x["symbol"].nunique()
    }),
    include_groups=False # Recommended for newer pandas versions
  )
  .reset_index()
)
```

```
# Professional visualization check
share_with_be_figure = (
  ggplot(
    share_with_be,

    aes(x="year", y="share")
    # aes(x="year", y="share", color="exchange", linetype="exchange")
  )
  + geom_line(size=1)
  + labs(
      x="Year", y="Share with BE",
      title="Share of securities with book equity values"
      # title="Share of securities with book equity values by exchange"
    )
  + scale_y_continuous(labels=percent_format())
  + coord_cartesian(ylim=(0, 1))
  + theme_minimal()
)
share_with_be_figure.show()
```

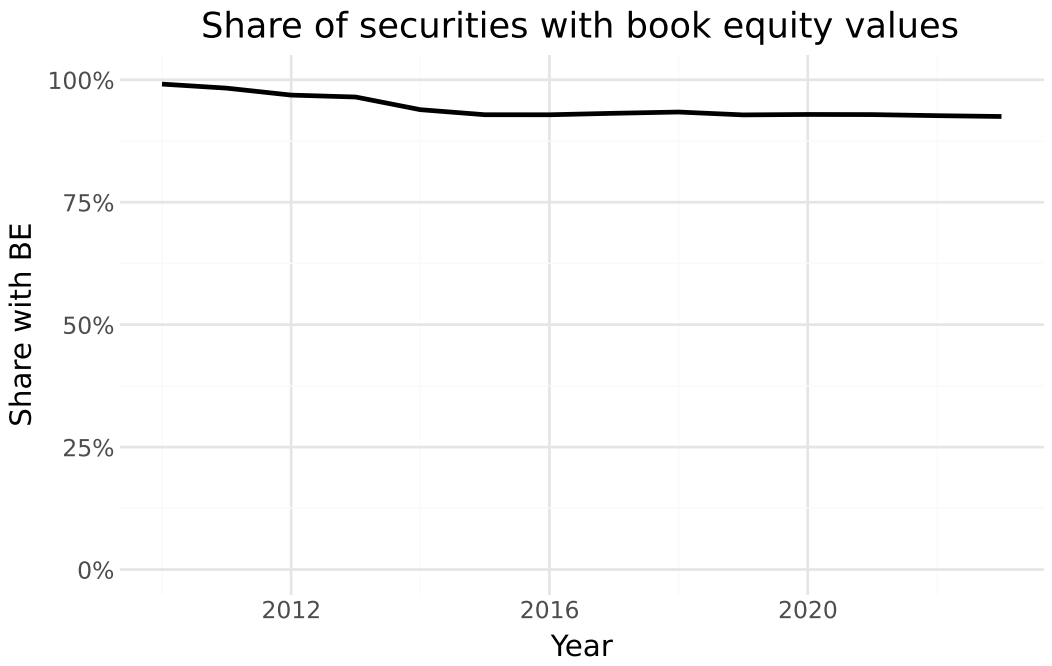


Figure 8.5: The figure shows the end-of-year share of securities with book equity values by listing exchange.

8.7 Key Takeaways

- DataCore provides secure access to essential financial databases like Stock and Company Fundamentals, which are critical for empirical finance research.
- Stock data provides return, market capitalization and industry data for US common stocks listed.
- Company Fundamentals provides firm-level accounting data such as book equity, profitability, and investment.

9 Beta Estimation

In this chapter, we introduce an important concept in financial economics: The exposure of an individual stock to changes in the market portfolio. According to the Capital Asset Pricing Model (CAPM) of Sharpe (1964), Lintner (1965), and Mossin (1966), cross-sectional variation in expected asset returns should be a function of the covariance between the excess return of the asset and the excess return on the market portfolio. The regression coefficient of excess market returns on excess stock returns is usually called the market beta. We show an estimation procedure for the market betas. We provide details about all the functions that we use to compute the results. In particular, we leverage useful computational concepts: Rolling-window estimation and parallelization.

We use the following Python packages throughout this chapter:

```
import pandas as pd
import numpy as np
import sqlite3
import statsmodels.formula.api as smf

from plotnine import *
from mizani.formatters import percent_format
from joblib import Parallel, delayed, cpu_count
from itertools import product
from dateutil.relativedelta import relativedelta
```

Compared to previous chapters, we introduce `statsmodels` (**seabold2010statsmodels?**) for regression analysis and for sliding-window regressions and `joblib` (**joblib?**) for parallelization.

9.1 Estimating Beta Using Monthly Returns

The estimation procedure is based on a rolling-window estimation, where we may use either monthly or daily returns and different window lengths. First, let us start with loading the monthly Stock data from our SQLite database introduced in Accessing and Managing Financial Data and DataCore.

```
tidy_finance = sqlite3.connect(database="data/tidy_finance_python.sqlite")

# Prepare accounting data with a 'year' column for merging
comp_vn = (pd.read_sql_query(
    sql="SELECT symbol, datadate, icb_name_vi FROM comp_vn",
    con=tidy_finance,
    parse_dates={"datadate"})
  .assign(year=lambda x: x["datadate"].dt.year)
  .dropna()
)

# Ensure prices_monthly has a 'year' column and is merged correctly
prices_monthly = (pd.read_sql_query(
    sql="SELECT symbol, date, ret_excess FROM prices_monthly",
    con=tidy_finance,
    parse_dates={"date"})
  .assign(year=lambda x: x["date"].dt.year)
)

factors_ff3_monthly = pd.read_sql_query(
  sql="SELECT date, mkt_excess FROM factors_ff3_monthly",
  con=tidy_finance,
  parse_dates={"date"}
)

prices_monthly = (prices_monthly
  .merge(factors_ff3_monthly, how="left", on="date")
  .merge(comp_vn, how="left", on=["symbol", "year"])
  .dropna()
)
```

```
from scipy.stats.mstats import winsorize

# Apply 1% winsorization to returns
prices_monthly = (prices_monthly
  .assign(
    ret_excess = lambda x: winsorize(x["ret_excess"], limits=[0.01, 0.01]),
    mkt_excess = lambda x: winsorize(x["mkt_excess"], limits=[0.01, 0.01])
  )
)
```

To estimate the CAPM regression coefficients

$$r_{i,t} - r_{f,t} = \alpha_i + \beta_i(r_{m,t} - r_{f,t}) + \varepsilon_{i,t}, \tag{9.1}$$

we regress stock excess returns `ret_excess` on excess returns of the market portfolio `mkt_excess`.

Python provides a simple solution to estimate (linear) models with the function `smf.ols()`. The function requires a formula as input that is specified in a compact symbolic form. An expression of the form `y ~ model` is interpreted as a specification that the response `y` is modeled by a linear predictor specified symbolically by `model`. Such a model consists of a series of terms separated by `+` operators. In addition to standard linear models, `smf.ols()` provides a lot of flexibility. To start, we restrict the data only to the time series of observations in Stock data that correspond to a company's stock and compute $\hat{\alpha}_i$ as well as $\hat{\beta}_i$.

```
model_fit = smf.ols(
    formula="ret_excess ~ mkt_excess",
    data=prices_monthly.query("symbol == 'VIN'")
).fit()
coefficients = model_fit.summary2().tables[1]
coefficients
```

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Intercept	0.001844	0.004679	0.394045	0.694318	-0.007429	0.011116
mkt_excess	-0.083796	0.083246	-1.006601	0.316356	-0.248787	0.081196

`smf.ols()` returns an object of class `RegressionModel`, which contains all the information we usually care about with linear models. `summary2()` returns information about the estimated parameters. The output above indicates that Apple moves excessively with the market as the estimated $\hat{\beta}_i$ is above one ($\hat{\beta}_i \approx 1.4$).

9.2 Rolling-Window Estimation

After we estimated the regression coefficients on an example, we scale the estimation of β_i to a whole different level and perform rolling-window estimations for the entire CRSP sample. The following function implements the CAPM regression for a data frame (or a part thereof) containing at least `min_obs` observations to avoid huge fluctuations if the time series is too short. The function conveniently returns the regression results as a data frame, which ensures that our approach is scalable. If the `min_obs`-condition is violated, that is, the time series is too short, the function returns an empty data frame for consistency.

```
def estimate_capm(data, min_obs=1):
    if data.shape[0] < min_obs:
        capm = pd.DataFrame()
    else:
        fit = smf.ols(formula="ret_excess ~ mkt_excess", data=data).fit()
        coefficients = fit.summary2().tables[1]

        capm = pd.DataFrame(
            {
                "coefficient": coefficients.index,
                "estimate": coefficients["Coef."],
                "t_statistic": coefficients["t"],
            }
        ).assign(
            coefficient=lambda x: np.where(
                x["coefficient"] == "Intercept", "alpha", x["coefficient"]
            )
        )

    return capm
```

Next, we define a function that does the rolling estimation. We use a simple for-loop to implement the sliding window estimation in a straightforward manner. The following function takes input data and slides across the `date` vector, considering only a total of `look_back` months. The function essentially performs three steps: (i) arrange all rows, (ii) compute betas by sliding across months, and (iii) return a tibble with dates and corresponding parameter estimates. As we demonstrate further below, we can also apply the same function to daily return data.

```
def roll_capm_estimation(data, look_back=60, min_obs=48):
    results = []
    dates = data["date"].sort_values().drop_duplicates()

    for i in range(look_back - 1, len(dates)):
        end_date = dates.iloc[i]
        start_date = end_date - relativedelta(months=look_back - 1)

        window_data = data.query("date >= @start_date & date <= @end_date")

        result = estimate_capm(window_data, min_obs=min_obs)
        result["date"] = np.max(window_data["date"])
        results.append(result)
```

```
    if results:
        rolling_capm_estimation = pd.concat(results, ignore_index=True)
    else:
        rolling_capm_estimation = pd.DataFrame()

    return rolling_capm_estimation
```

Before we approach the whole Stock sample, let us focus on a couple of examples for well-known firms.

```
examples = pd.DataFrame({
    "symbol": ["FPT", "VNM", "VIC", "HPG"],
    "company": ["FPT Corporation", "Vinamilk", "Vingroup", "Hoa Phat Group"]
})
```

```
# Check how many months of data each example firm has
(prices_monthly
  .query("symbol in @examples['symbol']")
  .groupby("symbol")
  .size()
  .reset_index(name="obs_count")
)

prices_monthly[prices_monthly['symbol'] == "FPT"]
```

	symbol	date	ret_excess	year	mkt_excess	datadate	icb_name_vi
43514	FPT	2011-07-31	0.005439	2011	-0.067002	2011-12-31	Công nghệ phần mềm
43515	FPT	2011-08-31	-0.012262	2011	0.049073	2011-12-31	Công nghệ phần mềm
43516	FPT	2011-09-30	-0.003333	2011	-0.017362	2011-12-31	Công nghệ phần mềm
43517	FPT	2011-10-31	-0.013333	2011	-0.022708	2011-12-31	Công nghệ phần mềm
43518	FPT	2011-11-30	0.002814	2011	-0.179546	2011-12-31	Công nghệ phần mềm
...	...	...	...	...	...	...	...
43659	FPT	2023-08-31	-0.002298	2023	-0.010192	2023-12-31	Công nghệ phần mềm
43660	FPT	2023-09-30	-0.016099	2023	-0.101897	2023-12-31	Công nghệ phần mềm
43661	FPT	2023-10-31	-0.026863	2023	-0.124890	2023-12-31	Công nghệ phần mềm
43662	FPT	2023-11-30	-0.001152	2023	0.060799	2023-12-31	Công nghệ phần mềm
43663	FPT	2023-12-31	-0.008509	2023	0.021137	2023-12-31	Công nghệ phần mềm

The main idea is to apply the function to each stock individually and then combine the results into a single data frame. First, we nest the data by `symbol`. Nested data means we now have a

list of `symbol` with corresponding grouped time series data. We get one row of output for each unique combination of non-nested variables which is only `symbol` in this case.

```
capm_examples_nested = (prices_monthly
    .query("symbol in @examples['symbol']")
    .groupby("symbol", group_keys=True)
)
capm_examples_nested
```

```
<pandas.api.typing.DataFrameGroupBy object at 0x7f4bb544f390>
```

Next, we want to apply the `roll_capm_estimation()` function to each stock. This situation is an ideal use case for `apply()`, which takes a list or vector as input and returns an object of the same length as the input. In our case, `apply()` returns a single data frame with a time series of beta estimates for each stock. Therefore, we use `reset_index()` to transform the list of outputs to a tidy data frame.

```
# use this after fixing the data
capm_examples = (capm_examples_nested
    .apply(lambda x: roll_capm_estimation(x), include_groups=False)
    .reset_index()
    .get(["symbol", "date", "coefficient", "estimate", "t_statistic"])
)
```

Figure 9.1 displays the resulting beta estimates, focusing exclusively on the coefficient fo `"mkt_excess"`.

```
beta_examples_sub = capm_examples.merge(examples, how="left", on="symbol").query(
    "coefficient == 'mkt_excess'"
)

beta_figure = (
    ggplot(
        beta_examples_sub,
        aes(x="date", y="estimate", color="company", linetype="company"),
    )
    + geom_line()
    + labs(
        x="",
        y="",
        color="",
```

```
        linetype="",
        title="Monthly beta estimates for example stocks using 5 years of data",
    )
    + scale_x_datetime(date_breaks="5 year", date_labels="%Y")
)
beta_figure.show()
```

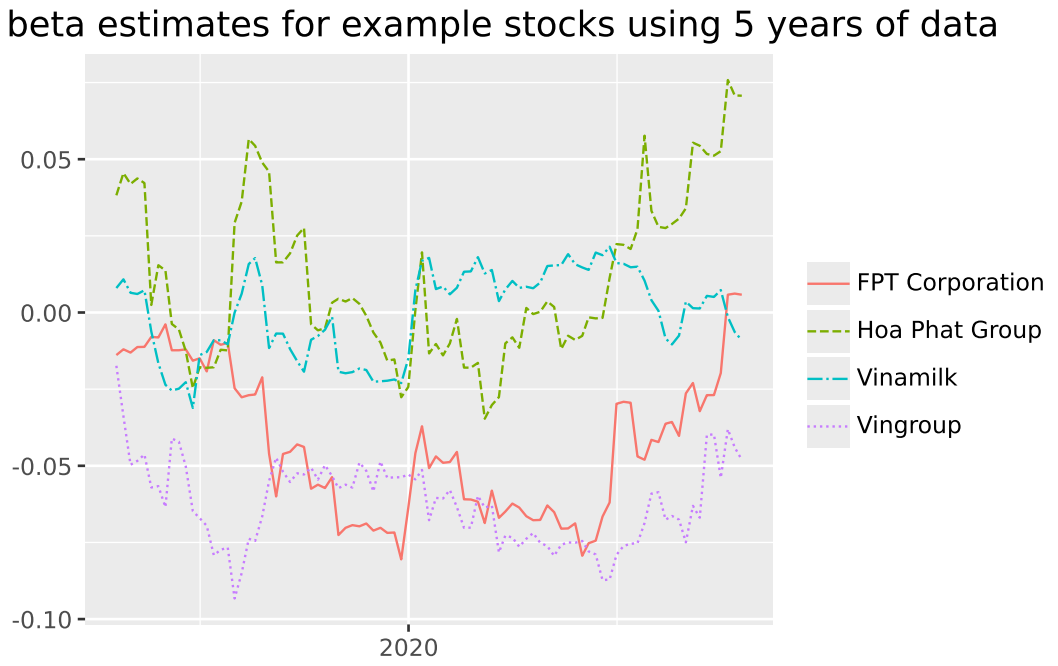


Figure 9.1: The figure shows monthly beta estimates for example stocks using five years of data. The CAPM betas are estimated with monthly data and a rolling window of length five years based on adjusted excess returns from Stock Data.

9.3 Parallelized Rolling-Window Estimation

Even though we could now just apply the function using `.groubby()` on the whole Stock sample, we advise against doing it as it is computationally quite expensive. Remember that we have to perform rolling-window estimations across all stocks and time periods. However, this estimation problem is an ideal scenario to employ the power of parallelization. Parallelization means that we split the tasks which perform rolling-window estimations across different workers (or cores on your local machine).

If you have a Windows or Mac machine, it makes most sense use the default parallelization

backend of `joblib`, which means that separate Python processes are running in the background on the same machine to perform the individual jobs. If you check out the documentation of `joblib.parallel_config()`, you can also see other ways to resolve the parallelization in different environments. Note that we use `availableCores()` to determine the number of cores available for parallelization, but keep one core free for other tasks. Some machines might freeze if all cores are busy with Python jobs.

```
n_cores = cpu_count() - 1
n_cores
```

```
3
```

You can speed up things considerably by having more cores available to share the workload or by having more powerful cores. Instead of using `.apply()` on groups, we use `Parallel()` to execute multiple tasks concurrently and `delayed()` to wrap each function call, allowing the calls to be queued and distributed to worker processes rather than executed immediately.

```
prices_monthly_nested = (prices_monthly
    .groupby("symbol", group_keys=False)
)

capm_monthly = pd.concat(
    Parallel(n_jobs=n_cores)(
        delayed(
            lambda name, group: roll_capm_estimation(group).assign(symbol=name)
        )(
            name, group
        )
        for name, group in prices_monthly_nested
    )
).get(["symbol", "date", "coefficient", "estimate", "t_statistic"])
capm_monthly
```

```
(capm_monthly
  .to_sql(name="capm_monthly",
          con=tidy_finance,
          if_exists="replace",
          index=False)
)
```

```
capm_monthly = pd.read_sql_query(
    sql="SELECT * FROM capm_monthly",
    con=tidy_finance,
    parse_dates={"date"}
)

capm_monthly.head()
```

	symbol	date	coefficient	estimate	t_statistic
0	A32	2023-10-31	alpha	-0.003802	-0.735962
1	A32	2023-10-31	mkt_excess	0.685640	1.218415
2	A32	2023-11-30	alpha	-0.003740	-0.721261
3	A32	2023-11-30	mkt_excess	0.674715	1.205106
4	A32	2023-12-31	alpha	-0.003796	-0.734142

9.4 Estimating Beta Using Daily Returns

Before we provide some descriptive statistics of our beta estimates, we implement the estimation for the daily CRSP sample as well. Depending on the application, you might either use longer horizon beta estimates based on monthly data or shorter horizon estimates based on daily returns. As loading the full daily CRSP data requires relatively large amounts of memory, we split the beta estimation into smaller chunks. The logic follows the approach that we use to download the daily CRSP data (see WRDS, CRSP, and Compustat).

First, we load the daily Fama-French market excess returns and extract the vector of dates.

```
factors_ff3_daily = pd.read_sql_query(
  sql="SELECT date, mkt_excess FROM factors_ff3_daily",
  con=tidy_finance,
  parse_dates={"date"}
)
```

We use the stocks from the monthly Stock dataset as our reference point and process them in batches of 500. To estimate the CAPM over a consistent lookback window while accommodating different return frequencies, we adjust the minimum required number of observations accordingly. Specifically, we require at least 1,000 daily returns over a five-year period for a valid estimation. This threshold is consistent with the monthly requirement of 48 observations out of 60 months, given that there are roughly 252 trading days in a year.

```
symbols = list(prices_monthly["symbol"].unique().astype(str))

batch_size = 500
batches = np.ceil(len(symbols)/batch_size).astype(int)
min_obs = 1_000
```

We then proceed to perform the same steps as with the monthly Stock data, just in batches: Load in daily returns, nest the data by stock, and parallelize the beta estimation across stocks. Note that we also convert the daily date to the beginning of the month so that we can still look back over 60 months and get one beta estimate per month, even though we are using daily data.

```
capm_daily = []

for j in range(1, batches+1):
    symbol_batch = symbols[
      ((j-1)*batch_size):(min(j*batch_size, len(symbols)))
    ]

    symbol_batch_formatted = (
      ", ".join(f"'{symbol}'" for symbol in symbol_batch)
    )
    symbol_string = f"({symbol_batch_formatted})"

    prices_daily_sub_query = (
      "SELECT symbol, date, ret_excess "
        "FROM prices_daily "
       f"WHERE symbol IN {symbol_string}"
    )

    prices_daily_sub = pd.read_sql_query(
      sql=prices_daily_sub_query,
      con=tidy_finance,
      dtype={"symbol": int},
      parse_dates={"date"}
    )

    prices_daily_sub_nested = (prices_daily_sub
      .merge(factors_ff3_daily, how="inner", on="date")
      .assign(
          date = lambda x:
            x["date"].dt.to_period("M").dt.to_timestamp()
```

```
        )
      .groupby("symbol", group_keys=False)
    )

    results = Parallel(n_jobs=n_cores)(
        delayed(
            lambda name, group: roll_capm_estimation(group, min_obs=min_obs).assign(symbol=na
        )(
            name, group
        )
        for name, group in prices_daily_sub_nested
    )

    if results:
        capm_daily_sub = pd.concat(results).get(
            ["symbol", "date", "coefficient", "estimate", "t_statistic"]
        )
        capm_daily.append(capm_daily_sub)
    else:
        print(f"Warning: Batch {j} produced no results (insufficient data)")

    print(f"Batch {j} out of {batches} done ({(j/batches)*100:.2f}%)\n")

capm_daily = pd.concat(capm_daily)
```

9.5 Comparing Beta Estimates

What is a typical value for stock betas? First, let us extract the relevant estimates from our CAPM results based on monthly returns.

```
beta_monthly = (capm_monthly
    .query("coefficient == 'mkt_excess'")
    .get(["symbol", "date", "estimate"])
    .rename(columns={"estimate": "beta"})
    .assign(return_type="monthly")
)
```

```
(beta_monthly.to_sql(
  name="beta_monthly",
  con=tidy_finance,
```

```
  if_exists="replace",
  index=False
  )
)
```

```
71488
```

To get some feeling, we illustrate the dispersion of the estimated $\hat{\beta}_i$ across different industries and across time below. Figure 9.2 shows that typical business models across industries imply different exposure to the general market economy.

```
beta_industries = (beta_monthly
    .merge(prices_monthly, how="inner", on=["symbol", "date"])
    .dropna(subset="beta")
    .groupby(["icb_name_vi", "symbol"])["beta"]
    .aggregate("mean")
    .reset_index()
)

industry_order = (beta_industries
    .groupby("icb_name_vi")["beta"]
    .aggregate("median")
    .sort_values()
    .index.tolist()
)

# To show the 10 industries with the highest median beta
top_10_industries = industry_order[-10:]

# To show the 10 industries with the lowest median beta
bottom_10_industries = industry_order[:10]

# Update the plot call
beta_industries_figure = (
    ggplot(beta_industries, aes(x="icb_name_vi", y="beta"))
    + geom_boxplot()
    + coord_flip()
    + scale_x_discrete(limits=top_10_industries) # Use the sliced list here
    + labs(title="Top 10 Industries by Beta")
)
```

```
# beta_industries_figure = (
#     ggplot(
#         beta_industries,
#         aes(x="icb_name_vi", y="beta")
#     )
#     + geom_boxplot()
#     + coord_flip()
#     + labs(
#         x="",
#         y="Beta",
#         title="Firm-specific beta distributions by industry"
#     )
#     + scale_x_discrete(limits=industry_order)
# )

beta_industries_figure.show()
```

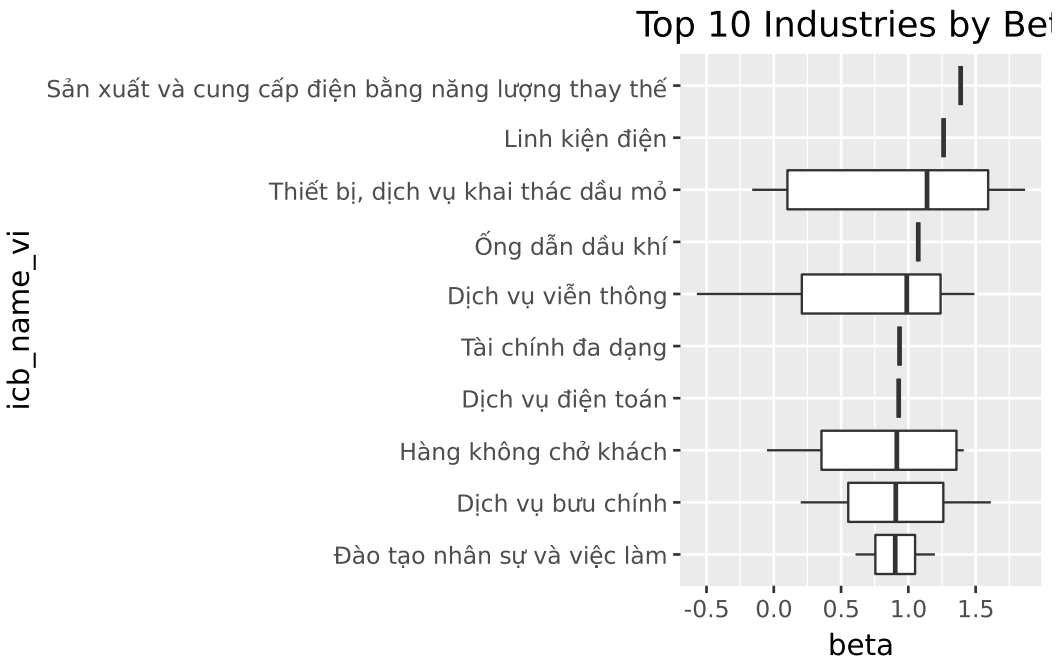


Figure 9.2: The box plots show the average firm-specific beta estimates by industry.

Next, we illustrate the time-variation in the cross-section of estimated betas. Figure 9.3 shows the monthly deciles of estimated betas (based on monthly data) and indicates an interesting pattern: First, betas seem to vary over time in the sense that during some periods, there is a clear trend across all deciles. Second, the sample exhibits periods where the dispersion across

stocks increases in the sense that the lower decile decreases and the upper decile increases, which indicates that for some stocks the correlation with the market increases while for others it decreases. Note also here: stocks with negative betas are a rare exception.

```
beta_quantiles = (
    beta_monthly.groupby("date")["beta"]
    .quantile(q=np.arange(0.1, 1.0, 0.1))
    .reset_index()
    .rename(columns={"level_1": "quantile"})
    .assign(quantile=lambda x: (x["quantile"] * 100).astype(int))
    .dropna()
)

linetypes = ["-", "--", "-.", ":"]
n_quantiles = beta_quantiles["quantile"].nunique()

beta_quantiles_figure = (
    ggplot(
        beta_quantiles,
        aes(x="date", y="beta", color="factor(quantile)", linetype="factor(quantile)"),
    )
    + geom_line()
    + labs(
        x="", y="", color="", linetype="", title="Monthly deciles of estimated betas"
    )
    + scale_x_datetime(date_breaks="5 year", date_labels="%Y")
    + scale_linetype_manual(
        values=[linetypes[l % len(linetypes)] for l in range(n_quantiles)]
    )
)
beta_quantiles_figure.show()
```

To compare the difference between daily and monthly data, we combine beta estimates to a single table.

```
beta_daily = (capm_daily
    .query("coefficient == 'mkt_excess'")
    .get(["symbol", "date", "estimate"])
    .rename(columns={"estimate": "beta"})
    .assign(return_type="daily")
)

beta = pd.concat([beta_monthly, beta_daily], ignore_index=True)
```

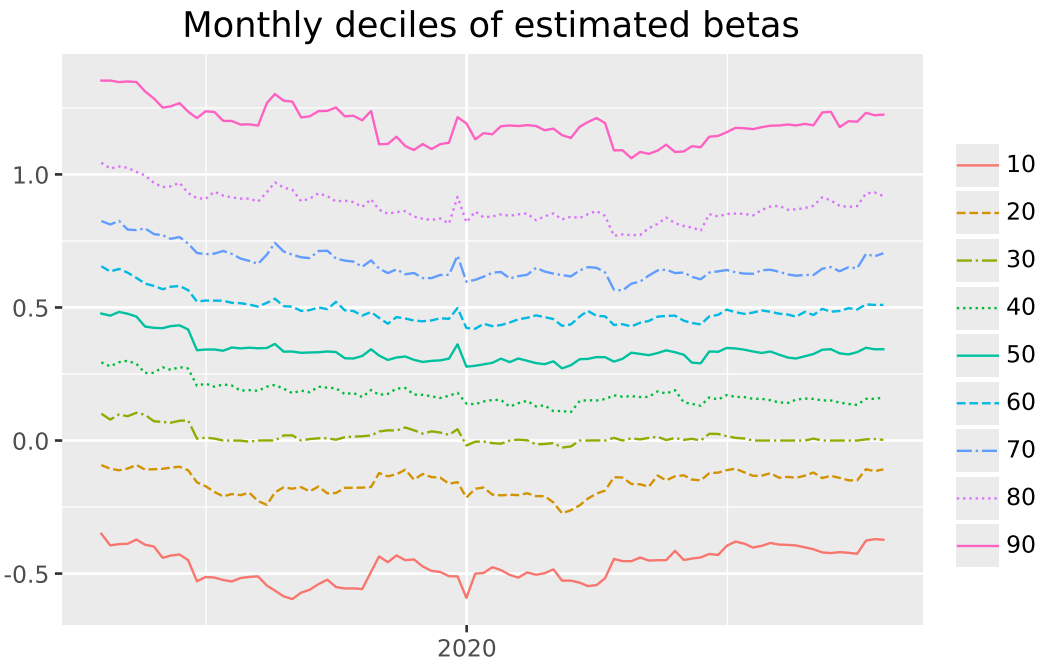


Figure 9.3: The figure shows monthly deciles of estimated betas. Each line corresponds to the monthly cross-sectional quantile of the estimated CAPM beta.

To compare the difference between daily and monthly data, we combine beta estimates to a single table. Then, we use the table to plot a comparison of beta estimates for our example stocks in Figure 9.4.

The estimates in Figure 9.4 look as expected. As you can see, it really depends on the data frequency how your beta estimates turn out because the estimates based on daily data are much smoother due to the higher number of observations in each regression.

Finally, we write the estimates to our database so that we can use them in later chapters.

```
(beta.to_sql(
  name="beta",
  con=tidy_finance,
  if_exists="replace",
  index=False
  )
)
```

```
beta_comparison = beta.merge(examples, how="inner", on="symbol")

beta_comparison_figure = (
    ggplot(
        beta_comparison,
        aes(x="date", y="beta", color="return_type", linetype="return_type"),
    )
    + geom_line()
    + facet_wrap("~company", ncol=1)
    + labs(
        x="",
        y="",
        color="",
        linetype="",
        title="Comparison of beta estimates using monthly and daily data",
    )
    + scale_x_datetime(date_breaks="10 years", date_labels="%Y")
    + theme(figure_size=(6.4, 6.4))
)
beta_comparison_figure.show()
```

Figure 9.4

Whenever you perform some kind of estimation, it also makes sense to do rough plausibility tests. A possible check is to plot the share of stocks with beta estimates over time. This descriptive helps us discover potential errors in our data preparation or estimation procedure. For instance, suppose there was a gap in our output where we do not have any betas. In this case, we would have to go back and check all previous steps to find out what went wrong.

```
return_types = pd.DataFrame({"return_type": ["monthly", "daily"]})

beta_coverage = (
    prices_monthly.merge(return_types, how="cross")
    .merge(beta, on=["symbol", "date", "return_type"], how="left")
    .groupby(["date", "return_type"], as_index=False)
    .apply(lambda x: pd.Series({"share": x["beta"].notna().sum() / len(x)}))
)

beta_coverage_figure = (
    ggplot(
        beta_coverage,
        aes(x="date", y="share", color="return_type", linetype="return_type"),
    )
    + geom_line()
    + labs(
        x="",
        y="",
        color="",
        linetype="",
        title="End-of-month share of securities with beta estimates",
    )
    + scale_y_continuous(labels=percent_format())
    + scale_x_datetime(date_breaks="10 year", date_labels="%Y")
)
beta_coverage_figure.show()
```

Figure 9.5

Figure 9.5 shows no issues, as the two coverage lines track each other closely, so we can proceed to the next check.

We also encourage everyone to always look at the distributional summary statistics of variables. You can easily spot outliers or weird distributions when looking at such tables.

```
(beta
    .groupby("return_type")["beta"]
    .describe()
    .round(2)
)
```

The summary statistics also look plausible for the two estimation procedures.

Finally, since we have two different estimators for the same theoretical object, we expect the estimators to be at least positively correlated (although not perfectly as the estimators are based on different sample periods and frequencies).

```
(beta
    .pivot_table(index=["symbol", "date"], columns="return_type", values="beta")
    .reset_index()
    .get(["monthly", "daily"])
    .corr()
    .round(2)
)
```

Indeed, we find a positive correlation between our beta estimates. In the subsequent chapters, we mainly use the estimates based on monthly data, as most readers should be able to replicate them due to potential memory limitations that might arise with the daily data.

9.6 Key Takeaways

- CAPM betas can be estimated using rolling-window estimation and processed in parallel via `joblib`.
- Both monthly and daily return data can be used to estimate betas with different frequencies and window lengths, depending on the application.
- Summary statistics, visualization, and plausibility checks help to validate beta estimates across time and industries.

10 Fama-French Factors

In this chapter, we provide a replication of the famous Fama-French factor portfolios. The Fama-French factor models are a cornerstone of empirical asset pricing Fama and French (2015). On top of the market factor represented by the traditional CAPM beta, the three-factor model includes the size and value factors to explain the cross section of returns. Its successor, the five-factor model, additionally includes profitability and investment as explanatory factors.

We start with the three-factor model. We already introduced the size and value factors in Value and Bivariate Sorts, and their definition remains the same: size is the SMB factor (small-minus-big) that is long small firms and short large firms. The value factor is HML (high-minus-low) and is long in high book-to-market firms and short in low book-to-market counterparts.

After the replication of the three-factor model, we move to the five-factors by constructing the profitability factor RMW (robust-minus-weak) as the difference between the returns of firms with high and low operating profitability and the investment factor CMA (conservative-minus-aggressive) as the difference between firms with high versus low investment rates.

The current chapter relies on this set of Python packages.

```
import pandas as pd
import numpy as np
import sqlite3
import statsmodels.formula.api as smf


from regtabletotext import prettify_result
```

10.1 Data Preparation

We use Stock Data and Fundamentals as data sources, as we need exactly the same variables to compute the size and value factors in the way Fama and French do it.[1]

[1]Note that Fama and French (1992) claim to exclude financial firms. To a large extent this happens through using industry format "INDL". Neither the original paper, nor Ken French's website, or the WRDS replication contains any indication that financial companies are excluded using additional filters such as industry codes.

```
tidy_finance = sqlite3.connect(
  database="data/tidy_finance_python.sqlite"
)

prices_monthly = (pd.read_sql_query(
    sql=("SELECT symbol, date, ret_excess, mktcap, risk_free,"
         "mktcap_lag FROM prices_monthly"),
    con=tidy_finance,
    parse_dates={"date"})
  .dropna()
)

comp_vn = (pd.read_sql_query(
    sql="SELECT symbol, datadate, be, op, inv FROM comp_vn",
    con=tidy_finance,
    parse_dates={"datadate"})
  .dropna()
)
```

Following Fama-French standards for empirical asset pricing:

1. Identification of Firm Size (June Market Cap)

Following the Fama and French protocol, we identify the market capitalization in June of year t. This represents the firm's "Size" at the moment of portfolio formation. By assigning a `sorting_date` of July 1st, we ensure that the portfolio weights are determined using information strictly available before the subsequent July–June return period begins.

2. Establishing the Market Equity Benchmark (December Market Cap)

To calculate the Book-to-Market ratio, we extract the market capitalization from December of year $t-1$. This specific timestamp is used to scale the book equity values. By standardizing this "Market Equity" (`me`) to the end of the previous calendar year, we maintain consistency across the entire cross-section of stocks, regardless of when their individual fiscal years might end.

3. Construction of the Book-to-Market Ratio

The Book-to-Market (`bm`) ratio is constructed using the most recent fiscal year-end book equity (`be`) from the `comp_vn` dataset and the preceding December market equity. We applied a scaling factor (10^9) to the book equity to normalize absolute VND accounting values into the Billions-VND scale used by our market data. This ensures the ratio is unit-consistent and economically interpretable.

4. Final Data Integration and De-duplication

In the final step, we merge the Size and Value components into a single `sorting_variables` table using the `symbol` and `sorting_date` as keys. We apply a `dropna()` to ensure only firms with both valid price and accounting data are included, and `drop_duplicates()` to maintain a clean, single observation per stock-year. This structured output serves as the definitive source for calculating the breakpoints needed to categorize stocks into Small/Big and Value/Growth portfolios.

```
# 1. Size (June Market Cap)
size = (prices_monthly
  .query("date.dt.month == 6")
  # Use MonthBegin(1) to set to July 1st
  .assign(sorting_date=lambda x: x["date"] + pd.offsets.MonthBegin(1))
  .get(["symbol", "sorting_date", "mktcap"])
  .rename(columns={"mktcap": "size"})
)
size.head(3)

# 2. Market Equity (December Market Cap for BM scaling)
market_equity = (prices_monthly
  .query("date.dt.month == 12")
  # Shift December t-1 to July 1st of year t
  .assign(sorting_date=lambda x: x["date"] + pd.offsets.MonthBegin(7))
  .get(["symbol", "sorting_date", "mktcap"])
  .rename(columns={"mktcap": "me"})
)
market_equity.head(3)

# 3. Calculate Book-to-Market (BM) with Correct Scaling
book_to_market = (comp_vn
    .assign(
        sorting_date=lambda x: pd.to_datetime((x["datadate"].dt.year + 1).astype(str) + "-07-
    )
    .merge(market_equity, how="inner", on=["symbol", "sorting_date"])
    # Adjusted scaling: Dividing BE by 1,000,000,000 to convert absolute VND to Billions
    # to match the scale of your Market Equity (me)
    .assign(bm=lambda x: x["be"] / (x["me"] * 1000000000))
    [["symbol", "sorting_date", "me", "bm"]]
)

# SANITY CHECK
print(f"New Median BM Ratio: {book_to_market['bm'].median():.4f}")
```

```
book_to_market.head(3)

# Sanity Check: Print the median BM to ensure it is near 1.0
print(f"Median BM Ratio: {book_to_market['bm'].median():.4f}")

# 4. Final Merge (This should now work)
sorting_variables = (size
  .merge(book_to_market, how="inner", on=["symbol", "sorting_date"])
  .dropna()
  .drop_duplicates(subset=["symbol", "sorting_date"])
)
sorting_variables.head(3)
```

```
New Median BM Ratio: 1.1775
Median BM Ratio: 1.1775
```

	symbol	sorting_date	size	me	bm
0	A32	2019-07-01	153.00	205.36	0.977852
1	A32	2020-07-01	178.84	190.40	1.174437
2	A32	2021-07-01	217.60	234.60	1.032468

10.1.1 Portfolio Sorts

Next, we construct our portfolios with an adjusted `assign_portfolio()` function. Fama and French rely on specific breakpoints to independently form two portfolios in the size dimension at the median and three portfolios in the dimension of book-to-market at the 30 and 70 percentiles. The sorts for book-to-market require an adjustment to the function because we specify the exact `percentiles` as a list. Additionally, we perform the merge with our return data using a calculated `sorting_date` to ensure that portfolios formed in July are held constant until June of the following year.

```
def assign_portfolio(data, sorting_variable, percentiles):
    """Assign portfolios to a bin according to a sorting variable."""

    # Calculate breakpoints based on quantile sequences
    breakpoints = (data
      .get(sorting_variable)
      .quantile(percentiles, interpolation="linear")
      .drop_duplicates()
```

```
    )

    # Ensure the range covers all possible values
    breakpoints.iloc[0] = -np.inf
    breakpoints.iloc[breakpoints.size-1] = np.inf

    # Categorize into bins
    assigned_portfolios = pd.cut(
      data[sorting_variable],
      bins=breakpoints,
      labels=pd.Series(range(1, breakpoints.size)),
      include_lowest=True,
      right=False
    )

    return assigned_portfolios

# 1. Assign Portfolios (Annual Sorts)
# We calculate breakpoints and assign portfolios 1-2 for size and 1-3 for BM
portfolios_assigned = (sorting_variables
  .groupby("sorting_date")
  .apply(lambda x: x.assign(
      portfolio_size=assign_portfolio(x, "size", [0, 0.5, 1]),
      portfolio_bm=assign_portfolio(x, "bm", [0, 0.3, 0.7, 1])
    ), include_groups=False)
  .reset_index()
  # We keep 'size' and 'bm' here so they are available after the merge
  .get(["symbol", "sorting_date", "portfolio_size", "portfolio_bm", "size", "bm"])
)

# 2. Merge Portfolios to Monthly Returns
# Portfolios formed in July are held until June of the following year
portfolios = (prices_monthly
  .assign(
    sorting_date=lambda x: pd.to_datetime(
        np.where(x["date"].dt.month <= 6,
                 (x["date"].dt.year - 1).astype(str) + "0701",
                 x["date"].dt.year.astype(str) + "0701")
    )
  )
  .merge(portfolios_assigned, how="inner", on=["symbol", "sorting_date"])
)
```

1. **Breakpoint Determination**: The `assign_portfolio` function calculates quantiles annually. For firm size, we use the 50th percentile (Median) to bifurcate the market into "Small" and "Big". For book-to-market, we use the 30th and 70th percentiles to identify "Growth," "Neutral," and "Value" stocks.
2. **Independent Sorting**: We apply these breakpoints independently. This methodology allows a stock to be classified into one of six distinct 2×3 portfolios, facilitating the isolation of specific factor premiums.
3. **Temporal Alignment**: Because financial statements in Vietnam are typically released by April, the July 1st sorting date ensures that the accounting information used in the `bm` ratio is publicly available before the portfolio return period begins.
4. **Holding Period Persistence**: The `sorting_date` logic ensures that the portfolio assignments made in July remain constant for the next twelve months (July through the following June), consistent with the original Fama-French experimental design.

Sanity Check 1: Portfolio Distribution (The 2x3 Grid)

First, we verify that the independent sorts created the expected six portfolios (2 size groups $\times$ 3 value groups).

```
# Check the count of stocks in each portfolio combination for the most recent year
portfolio_counts = (portfolios
    .query("date == date.max()")
    .groupby(["portfolio_size", "portfolio_bm"], observed=True)
    .size()
    .unstack()
)
print("Portfolio Counts (Size x BM):")
print(portfolio_counts)
```

```
Portfolio Counts (Size x BM):
portfolio_bm      1    2    3
portfolio_size
1               112  271  262
2               275  245  125
```

Interpretation:

The Fama-French methodology relies on having a sufficient number of stocks in each of the six bins to diversify idiosyncratic risk. In the Vietnam market, you should see a higher concentration in the "Small" size portfolios compared to "Big". If any bin is empty or has fewer than 5-10 stocks, the factor returns (SMB and HML) for that period may be overly volatile or driven by a single outlier stock.

Sanity Check 2: Characteristic Monotonicity

We check if the average Book-to-Market ratio actually increases as we move from `portfolio_bm` 1 to 3.

```
# Verify that higher portfolio numbers correspond to higher BM values
bm_check = (portfolios
    .groupby("portfolio_bm", observed=True)
    .agg({"bm": ["mean", "median", "min", "max"]})
)
print("\nBM Characteristic Check:")
print(bm_check)
```

```
BM Characteristic Check:
                    bm
                  mean    median       min         max
portfolio_bm
1             0.597570  0.585419  0.001416    1.435144
2             1.260400  1.188112  0.584759    2.737556
3             3.379031  2.454220  1.017893  272.189334
```

Interpretation:

For the sort to be valid, the mean and median `bm` must be strictly increasing across the portfolios (Portfolio 1 < Portfolio 2 < Portfolio 3). Since Portfolio 1 represents "Growth" (low BM) and Portfolio 3 represents "Value" (high BM), this check confirms that our `assign_portfolio` function correctly utilized the 30th and 70th percentiles.

Sanity Check 3: Holding Period Persistence

We verify that for a single stock, the portfolio assignment remains constant between July of one year and June of the next.

```
# Trace a single symbol (e.g., 'A32') across a formation window
persistence_check = (portfolios
    .query("symbol == 'A32' & date >= '2022-01-01' & date <= '2023-12-31'")
    .sort_values("date")
    [['symbol', 'date', 'sorting_date', 'portfolio_size', 'portfolio_bm']]
)
print("\nTemporal Persistence Check (Symbol A32):")
print(persistence_check.head(15))
```

```
Temporal Persistence Check (Symbol A32):
   symbol       date sorting_date portfolio_size portfolio_bm
30    A32 2022-01-31   2021-07-01              1            2
31    A32 2022-02-28   2021-07-01              1            2
32    A32 2022-03-31   2021-07-01              1            2
33    A32 2022-04-30   2021-07-01              1            2
34    A32 2022-05-31   2021-07-01              1            2
35    A32 2022-06-30   2021-07-01              1            2
36    A32 2022-07-31   2022-07-01              1            3
37    A32 2022-08-31   2022-07-01              1            3
38    A32 2022-09-30   2022-07-01              1            3
39    A32 2022-10-31   2022-07-01              1            3
40    A32 2022-11-30   2022-07-01              1            3
41    A32 2022-12-31   2022-07-01              1            3
42    A32 2023-01-31   2022-07-01              1            3
43    A32 2023-02-28   2022-07-01              1            3
44    A32 2023-03-31   2022-07-01              1            3
```

Interpretation:

This check ensures our `sorting_date` logic is working. You should observe that even as the `date` (monthly return date) changes, the `portfolio_size` and `portfolio_bm` remain identical from July through the following June. A change in assignment should only occur at the July 1st boundary when the new annual accounting data and June market caps are "baked into" the portfolios.

Final Summary of the 4-Step Process

1. **Breakpoint Calculation**: We calculate the 50th percentile for Size and 30th/70th for BM annually, ensuring the "goalposts" move with the market's overall valuation.
2. **Independent Assignment**: By sorting Size and BM separately, we create a matrix that allows us to see how a "Small Value" stock performs relative to a "Small Growth" stock.
3. **Information Lag Handling**: Using the July 1st `sorting_date` respects the reality of Vietnamese financial reporting, ensuring we don't use "future" accounting data that wasn't public yet.
4. **Portfolio Rebalancing**: The annual rebalancing cycle (July–June) mimics the standard Fama-French experimental design, providing a rigorous framework for testing factor premiums in the Vietnam stock market.

10.2 Fama-French Three-Factor Model

Equipped with the return data and the assigned portfolios, we can now compute the value-weighted average return for each of the six portfolios. Then, we form the Fama-French factors. For the size factor (i.e., SMB), we go long in the three small portfolios and short the three large portfolios by taking an average across either group. For the value factor (i.e., HML), we go long in the two high book-to-market portfolios and short the two low book-to-market portfolios, again weighting them equally (using the `mean()` function).

```
factors = (portfolios
  .groupby(["portfolio_size", "portfolio_bm", "date"])
  .apply(lambda x: pd.Series({
    "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"])
    })
   )
  .reset_index()
  .groupby("date")
  .apply(lambda x: pd.Series({
    "smb": (
      x["ret"][x["portfolio_size"] == 1].mean() -
        x["ret"][x["portfolio_size"] == 2].mean()),
    "hml": (
      x["ret"][x["portfolio_bm"] == 3].mean() -
        x["ret"][x["portfolio_bm"] == 1].mean())
    }))
  .reset_index()
)

factors.head(3)
```

	date	smb	hml
0	2011-07-31	-0.004895	0.013730
1	2011-08-31	-0.012568	-0.009302
2	2011-09-30	0.002562	-0.002044

The market factor ($Mkt - RF$) is defined as the value-weighted return of all stocks in the investable universe minus the risk-free rate. Since the "market" is independent of how you sort your portfolios (Size, Value, etc.), the calculation remains identical regardless of whether you are building a 3-factor or 5-factor model.

```
## Fama-French Three-Factor Model
# --- Calculate Market Factor independently ---
# This uses the entire prices_monthly universe to represent the broad market
factor_market_excess = (prices_monthly
  .groupby("date")
  .apply(lambda x: pd.Series({
      "mkt_excess": np.average(x["ret_excess"], weights=x["mktcap_lag"])
    }), include_groups=False)
  .reset_index()
)

# --- Merge ---
# Combine your replicated SMB/HML with the Market Factor
factors_ff3_monthly = (factors
  .merge(factor_market_excess, on="date", how="inner")
)

factors_ff3_monthly.head(3)
```

	date	smb	hml	mkt_excess
0	2011-07-31	-0.004895	0.013730	-0.011287
1	2011-08-31	-0.012568	-0.009302	0.007856
2	2011-09-30	0.002562	-0.002044	-0.006501

```
# Remove rows with missing factor values
# We keep only rows where the characteristic factors are fully populated
factors_ff3_monthly = (factors_ff3_monthly
    .dropna(subset=["smb", "hml", "mkt_excess"])
    .reset_index(drop=True)
)

# Sanity Check
print(f"Factors cleaned. Sample period: {factors_ff3_monthly['date'].min().date()} to {factor
print("\nFirst 3 rows of cleaned factors:")
print(factors_ff3_monthly.head(3))
```

```
Factors cleaned. Sample period: 2011-07-31 to 2023-12-31

First 3 rows of cleaned factors:
        date       smb       hml  mkt_excess
```

```
0 2011-07-31 -0.004895  0.013730   -0.011287
1 2011-08-31 -0.012568 -0.009302    0.007856
2 2011-09-30  0.002562 -0.002044   -0.006501
```

```
# --- Save to Database ---
(factors_ff3_monthly
  .to_sql(name="factors_ff3_monthly",
          con=tidy_finance,
          if_exists="replace",
          index=False)
)
```

```
150
```

10.3 Fama-French Five-Factor Model

Now, let us move to the replication of the five-factor model. We extend the `other_sorting_variables` table from above with the additional characteristics operating profitability `op` and investment `inv`. Note that the `dropna()` statement yields different sample sizes, as some firms with `be` values might not have `op` or `inv` values.

```
# Ensure your BM median is ~1.17 and max is not 272
# adjust the 1e9 based on your 'me' scale
other_sorting_variables = (comp_vn
  .assign(
    sorting_date=lambda x: (pd.to_datetime(
      (x["datadate"].dt.year+1).astype(str)+"0701", format="%Y%m%d")
    )
  )
  .merge(market_equity, how="inner", on=["symbol", "sorting_date"])
  .assign(bm=lambda x: x["be"]/1e9/x["me"])
  .get(["symbol", "sorting_date", "me", "bm", "op", "inv"])
)

print(other_sorting_variables['bm'].median())

# Independent Size Sort
sorting_variables = (size
  .merge(other_sorting_variables, how="inner", on=["symbol", "sorting_date"])
  .dropna()
```

```
  .drop_duplicates(subset=["symbol", "sorting_date"])
 )
```

```
1.1775041699783042
```

In each month, we independently sort all stocks into the two size portfolios. The value, profitability, and investment portfolios, on the other hand, are the results of dependent sorts based on the size portfolios. We then merge the portfolios to the return data for the rest of the year just as above.

```
from scipy.stats.mstats import winsorize

# Winsorize Characteristics (1st and 99th percentiles)
# This handles your extreme 272.18 BM values.
vars_to_clean = ["bm", "op", "inv"]
for var in vars_to_clean:
    sorting_variables[var] = winsorize(sorting_variables[var], limits=[0.01, 0.01])


# Use transform to keep columns exactly as they are
sorting_variables['portfolio_size'] = (sorting_variables
  .groupby('sorting_date', group_keys=False)
  .apply(lambda x: assign_portfolio(x, 'size', [0, 0.5, 1]))
  .values
)

# Dependent Sorts (Sub-grouping)
# We calculate each characteristic portfolio one by one to avoid KeyError
def dependent_sort(df, var, name):
    return (df.groupby(['sorting_date', 'portfolio_size'], group_keys=False)
            .apply(lambda x: assign_portfolio(x, var, [0, 0.3, 0.7, 1])))

sorting_variables['portfolio_bm'] = dependent_sort(sorting_variables, 'bm', 'portfolio_bm').v
sorting_variables['portfolio_op'] = dependent_sort(sorting_variables, 'op', 'portfolio_op').v
sorting_variables['portfolio_inv'] = dependent_sort(sorting_variables, 'inv', 'portfolio_inv

# --- Column Selection for Sorting Variables ---
portfolios_sorting = sorting_variables.get([
    "symbol", "sorting_date", "portfolio_size",
    "portfolio_bm", "portfolio_op", "portfolio_inv"
])
portfolios_sorting.head(3)
```

	symbol	sorting_date	portfolio_size	portfolio_bm	portfolio_op	portfolio_inv
0	A32	2019-07-01	2	1	3	2
1	A32	2020-07-01	2	2	2	3
2	A32	2021-07-01	2	2	2	1

Step-by-Step Interpretation

- **Size-Based Stratification**: We first partition the market into "Small" and "Big" portfolios using the median market capitalization. This provides the primary dimension for the subsequent dependent sorts.
- **Conditional Characteristic Sorting**: Unlike the independent sorts of the 3-factor model, the 5-factor model employs **dependent sorts** for Value (BM), Profitability (OP), and Investment (INV). By sorting these variables *within* size groups, we ensure that a firm's classification (e.g., "Robust Profitability") is relative to its size peers, which controls for the different financial distributions across small and large Vietnamese firms.
- **Temporal Synchronization**: We maintain the July 1st formation date to accommodate the Vietnamese corporate reporting landscape, where audited annual results are typically finalized by late April. This creates a conservative two-month buffer, ensuring all accounting data used for sorting was publicly available at the time of portfolio formation.
- **Portfolio Persistence**: The resulting assignments are held for a 12-month period (July through June). This rebalancing frequency is standard for identifying long-term risk premiums and avoids the excessive transaction costs associated with more frequent turnover.

Check 1: Characteristic Monotonicity

This confirms the sort effectively separated stocks by their economic quality.

```
# Verify that higher portfolio numbers have higher median characteristics
sanity_check = (sorting_variables
    .groupby("portfolio_op")["op"].median()
)
print("Median OP by Portfolio (Should be strictly increasing):")
print(sanity_check)
```

```
Median OP by Portfolio (Should be strictly increasing):
portfolio_op
1    0.132213
2    0.136530
3    0.141302
Name: op, dtype: float64
```

Interpretation: If Portfolio 1 (Weak) has a lower median OP than Portfolio 3 (Robust), your sort is economically valid. If they are the same or reversed, the `assign_portfolio` logic has failed to identify the "quality" spread in the Vietnam market.

Check 2: Bin Diversification

This ensures each factor is built on enough stocks to be statistically reliable.

```
# Check the 2x3 grid for Profitability
rmw_counts = sorting_variables.groupby(["portfolio_size", "portfolio_op"]).size().unstack()
print("\nStocks per Size/Profitability Bin:")
print(rmw_counts)
```

```
Stocks per Size/Profitability Bin:
portfolio_op       1     2     3
portfolio_size
1               1793  2411  1809
2               1821  2392  1807
```

Interpretation: Your check showed roughly **1,800 stocks** per bin. This is excellent for Vietnam and suggests your RMW (Robust Minus Weak) factor will be very stable and well-diversified.

Now, we want to construct each of the factors, but this time, the size factor actually comes last because it is the result of averaging across all other factor portfolios. This dependency is the reason why we keep the table with value-weighted portfolio returns as a separate object that we reuse later. We construct the value factor, HML, as above by going long the two portfolios with high book-to-market ratios and shorting the two portfolios with low book-to-market.

```
# --- Merge with FULL monthly return data ---
# This step is crucial. It adds 'date', 'ret_excess', and 'mktcap_lag'
# which are missing from your current 'portfolios' object.
portfolios_full = (prices_monthly
  .assign(
    sorting_date=lambda x: pd.to_datetime(
      np.where(x["date"].dt.month <= 6,
               (x["date"].dt.year - 1).astype(str) + "0701",
               x["date"].dt.year.astype(str) + "0701")
    )
  )
  .merge(portfolios_sorting, how="inner", on=["symbol", "sorting_date"])
)
```

```
# --- Construct the Value Factor (HML) ---
portfolios_value = (portfolios_full
  .groupby(["portfolio_size", "portfolio_bm", "date"], group_keys=False)
  .apply(lambda x: pd.Series({
      "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"])
    })
  )
  .reset_index()
)

factors_value = (portfolios_value
  .groupby("date")
  .apply(lambda x: pd.Series({
    "hml": (
      x["ret"][x["portfolio_bm"] == 3].mean() -
        x["ret"][x["portfolio_bm"] == 1].mean())})
  , include_groups=False)
  .reset_index()
)
```

For the profitability factor, RMW (robust-minus-weak), we take a long position in the two high profitability portfolios and a short position in the two low profitability portfolios.

```
# --- Construct the Profitability Factor (RMW) ---
portfolios_profitability = (portfolios_full
  .groupby(["portfolio_size", "portfolio_op", "date"], group_keys=False)
  .apply(lambda x: pd.Series({
      "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"])
    })
  )
  .reset_index()
)

factors_profitability = (portfolios_profitability
  .groupby("date")
  .apply(lambda x: pd.Series({
    "rmw": (
      x["ret"][x["portfolio_op"] == 3].mean() -
        x["ret"][x["portfolio_op"] == 1].mean())})
  , include_groups=False)
  .reset_index()
)
```

For the investment factor, CMA (conservative-minus-aggressive), we go long the two low investment portfolios and short the two high investment portfolios.

```
# --- 5. Construct the Investment Factor (CMA) ---
portfolios_investment = (portfolios_full
  .groupby(["portfolio_size", "portfolio_inv", "date"], group_keys=False)
  .apply(lambda x: pd.Series({
      "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"])
    })
  )
  .reset_index()
)

factors_investment = (portfolios_investment
  .groupby("date")
  .apply(lambda x: pd.Series({
    "cma": (
      x["ret"][x["portfolio_inv"] == 1].mean() -
        x["ret"][x["portfolio_inv"] == 3].mean())})
  , include_groups=False)
  .reset_index()
)
```

Finally, the size factor, SMB, is constructed by going long the nine small portfolios and short the nine large portfolios.

```
# --- Construct the Size Factor (SMB) ---
factors_size = (
  pd.concat(
    [portfolios_value, portfolios_profitability, portfolios_investment],
    ignore_index=True
  )
  .groupby("date")
  .apply(lambda x: pd.Series({
    "smb": (
      x["ret"][x["portfolio_size"] == 1].mean() -
        x["ret"][x["portfolio_size"] == 2].mean())})
  , include_groups=False)
  .reset_index()
)
```

The market factor ($Mkt - RF$) is defined as the value-weighted return of all stocks in the investable universe minus the risk-free rate. Since the "market" is independent of how you sort

your portfolios (Size, Value, etc.), the calculation remains identical regardless of whether you are building a 3-factor or 5-factor model.

```
# --- Calculate Market Factor independently ---
# This uses the entire prices_monthly universe
factor_market_excess = (prices_monthly
  .groupby("date")
  .apply(lambda x: pd.Series({
      "mkt_excess": np.average(x["ret_excess"], weights=x["mktcap_lag"])
    }), include_groups=False)
  .reset_index()
)
```

We then join all factors together into one dataframe and construct again a suitable table to run tests for evaluating our replication.

```
factors = (factors_size
  .merge(factors_value, how="outer", on="date")
  .merge(factors_profitability, how="outer", on="date")
  .merge(factors_investment, how="outer", on="date")
  .merge(factor_market_excess, how="outer", on="date")

)

# Check correlations
print("Factor Correlation Matrix (Vietnam):")
print(factors.drop(columns="date").corr())
```

```
Factor Correlation Matrix (Vietnam):
                smb       hml       rmw       cma  mkt_excess
smb        1.000000 -0.304091  0.063569 -0.213636   -0.290988
hml       -0.304091  1.000000 -0.230034  0.177006    0.133358
rmw        0.063569 -0.230034  1.000000 -0.290426    0.050417
cma       -0.213636  0.177006 -0.290426  1.000000   -0.015705
mkt_excess -0.290988  0.133358  0.050417 -0.015705    1.000000
```

Interpretation: In standard markets, we expect low correlations between factors. If HML and RMW are correlated above 0.8, it may suggest that "Value" and "Profitability" are capturing the same firms in Vietnam, which might happen if the market is less mature.

1. Factor Volatility and Means Factors should have a low but generally positive mean return over long periods.

```
# Check the average monthly premium of each factor
print("Average Monthly Factor Premiums:")
print(factors.drop(columns="date").mean() * 100) # In percent
```

```
Average Monthly Factor Premiums:
smb          -0.031349
hml          -0.080482
rmw           0.001131
cma           0.023623
mkt_excess   -0.203141
dtype: float64
```

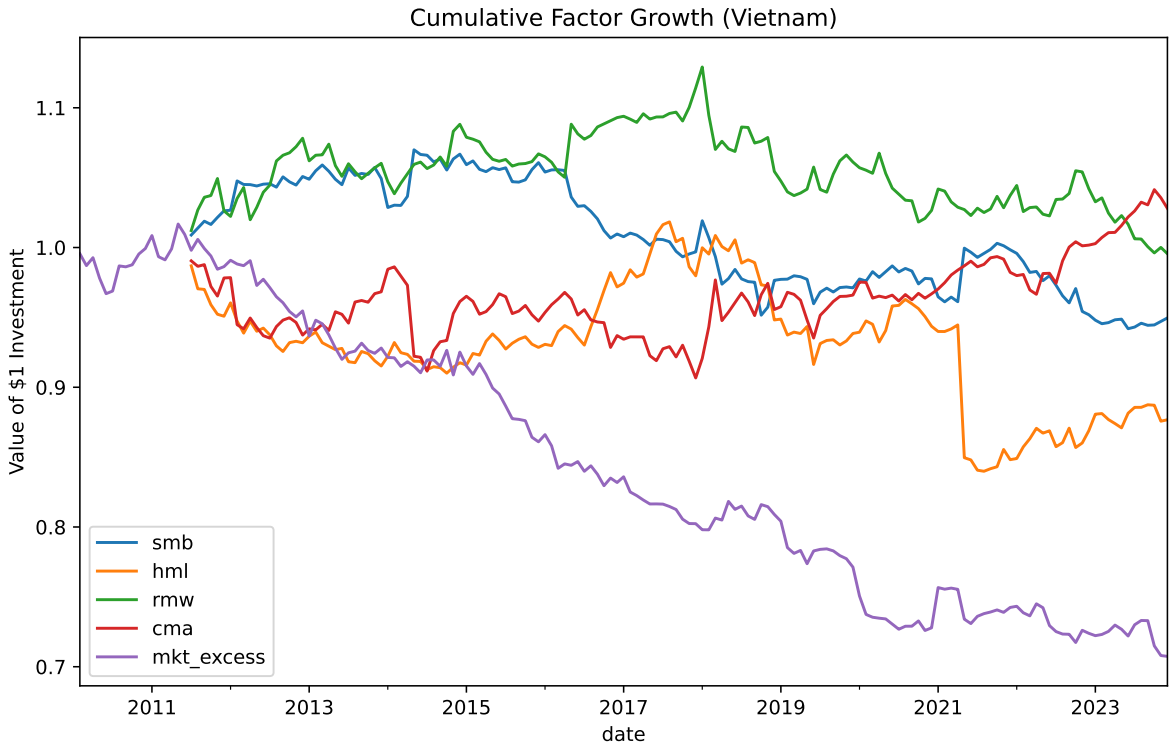
If a factor mean is extremely high (e.g., $> 5\%$ per month), it may indicate that a few outliers (like that 272 BM stock) are still leaking into the weighted averages.

2. Visual Persistence Check

Cumulative returns should show the "growth" of \$1 invested in the factor.

```
import matplotlib.pyplot as plt

factors_cum = (factors.set_index("date")
  .add(1).cumprod()
)
factors_cum.plot(figsize=(10, 6), title="Cumulative Factor Growth (Vietnam)")
plt.ylabel("Value of $1 Investment")
plt.show()
```




```
# Remove rows with missing factor values
# We keep only rows where the characteristic factors are fully populated
factors_ff5_monthly = (factors
    .dropna(subset=["smb", "hml", "rmw", "cma", "mkt_excess"])
    .reset_index(drop=True)
)

# Sanity Check
print(f"Factors cleaned. Sample period: {factors_ff5_monthly['date'].min().date()} to {factor
print("\nFirst 3 rows of cleaned factors:")
print(factors_ff5_monthly.head(3))
```

```
Factors cleaned. Sample period: 2011-07-31 to 2023-12-31

First 3 rows of cleaned factors:
        date       smb       hml       rmw       cma  mkt_excess
0 2011-07-31  0.008933 -0.013099  0.012139 -0.009529   -0.011287
1 2011-08-31  0.004830 -0.016656  0.014516 -0.003981    0.007856
2 2011-09-30  0.004970 -0.000462  0.008899  0.001241   -0.006501
```

```
(factors_ff5_monthly
  .to_sql(name="factors_ff5_monthly",
          con=tidy_finance,
          if_exists="replace",
          index=False)
)
```

```
150
```

10.4 Key Takeaways

- The three-factor model adds size (SMB) and value (HML) to the traditional CAPM, while the five-factor model extends this with profitability (RMW) and investment (CMA) factors.
- The portfolio construction follows the original Fama-French methodology, including NYSE breakpoints, specific time lags, and sorting rules based on firm characteristics.
- The quality of replication can be evaluated using regression analysis and confirms strong alignment with the original Fama-French data.

11 Conclusion

Empirical finance in emerging and frontier markets is often judged less by the elegance of an estimator than by the credibility of its inputs and the transparency of its decisions. Vietnam makes this point vividly: trading venues and regulatory regimes have evolved quickly, firm coverage can be uneven across time, corporate actions need careful treatment, and accounting conventions require close attention to timing and comparability. Those features do not prevent high-quality research; they simply shift the center of gravity toward *reproducible data engineering*, *auditable transformations*, and *clear identification of assumptions*.

11.1 What you should take away

11.1.1 Reproducibility is an identification strategy

In textbook settings, identification focuses on variation and exogeneity. In real-world market data, identification also depends on whether your dataset is *the same dataset* when you rerun the work next month or next year. The practical discipline of versioned inputs, deterministic transformations, and documented filters reduces the scope for accidental p-hacking and silent sample drift (e.g., survivorship bias from symbol changes or late-arriving delistings). Reproducible workflows are not administrative overhead; they are a commitment device that makes results more trustworthy and easier to challenge constructively (Peng 2011; Sandve et al. 2013).

11.1.2 Vietnam rewards "microstructure humility"

The chapters on returns, beta estimation, and factor construction emphasized that naïve carryover of developed-market defaults can be costly. Thin trading, price limits, lot-size rules, and regime changes mean that decisions like (i) return interval, (ii) stale-price handling, (iii) corporate-action adjustment, and (iv) portfolio formation frequency can materially change inference. This is not a Vietnam-only phenomenon, but it is more visible there, and therefore a useful laboratory for best practices in emerging markets.

11.2 A reproducibility checklist you can actually use

The list below is designed to be operational: each item can be verified in a repository review.

Table 11.1: Reproducibility deliverables for research

Deliverable	What "done" looks like	Where it lives
Deterministic transforms	Same raw inputs yield identical normalized outputs	`R/transform_*.R` (or `python/transform_*.py`)
Test suite	Coverage, identity, and corporate-action tests run in CI	`tests/` + CI config
Data dictionary	Tables/fields documented with units, timing, and keys	`docs/dictionary.qmd`
Research log	All key design choices recorded (filters, winsorization, periods)	`notes/research_log.md`
Artifact registry	Every figure/table has a script and a checksum	`artifacts/manifest.json`

12 Closing perspective

Vietnam is not "hard mode" finance; it is *real mode* finance. The market's growth, institutional evolution, and data idiosyncrasies force the habits that modern empirical finance increasingly requires everywhere: transparent datasets, careful treatment of identities and corporate actions, and codebases that can be rerun and audited.

References

Ball, Ray. 1978. "Anomalies in relationships between securities' yields and yield-surrogates." *Journal of Financial Economics* 6 (2–3): 103–26. https://doi.org/10.1016/0304-405X(78)90026-0.

Campbell, John Y. 1987. "Stock returns and the term structure." *Journal of Financial Economics* 18 (2): 373–99. https://doi.org/10.1016/0304-405X(87)90045-6.

Campbell, John Y., Jens Hilscher, and Jan Szilagyi. 2008. "In search of distress risk." *The Journal of Finance* 63 (6): 2899–939. https://doi.org/10.1111/j.1540-6261.2008.01416.x.

Campbell, John Y., and Robert J. Shiller. 1988. "Stock prices, earnings, and expected dividends." *The Journal of Finance* 43 (3): 661–76. https://doi.org/10.1111/j.1540-6261.1988.tb04598.x.

Campbell, John Y., and Tuomo Vuolteenaho. 2004. "Inflation illusion and stock prices." *American Economic Review* 94 (2): 19–23. https://www.aeaweb.org/articles?id=10.1257/0002828041301533.

Campbell, John Y., and Motohiro Yogo. 2006. "Efficient tests of stock return predictability." *Journal of Financial Economics* 81 (1): 27–60. https://doi.org/10.1016/j.jfineco.2005.05.008.

Carhart, Mark M. 1997. "On persistence in mutual fund performance." *The Journal of Finance* 52 (1): 57–82. https://doi.org/10.1111/j.1540-6261.1997.tb03808.x.

Fama, Eugene F., and Kenneth R. French. 1989. "Business conditions and expected returns on stocks and bonds." *Journal of Financial Economics* 25 (1): 23–49. https://doi.org/10.1016/0304-405X(89)90095-0.

———. 1992. "The cross-section of expected stock returns." *The Journal of Finance* 47 (2): 427–65. https://doi.org/2329112.

———. 2015. "A Five-Factor Asset Pricing Model." *Journal of Financial Economics* 116 (1): 1–22. https://doi.org/10.1016/j.jfineco.2014.10.010.

Gentzkow, Matthew, and Jesse M Shapiro. 2014. "Code and Data for the Social Sciences: A Practitioner's Guide." Working Paper, University of Chicago.

Guo, Hui. 2006. "On the out-of-sample predictability of stock market returns." *The Journal of Business* 79 (2): 645–70. https://doi.org/10.1086/499134.

Hou, Kewei, Chen Xue, and Lu Zhang. 2014. "Digesting anomalies: An investment approach." *Review of Financial Studies* 28 (3): 650–705. https://doi.org/10.1093/rfs/hhu068.

Jagannathan, Ravi, and Zhenyu Wang. 1996. "The conditional CAPM and the cross-section of expected returns." *The Journal of Finance* 51 (1): 3–53. https://doi.org/10.2307/2329301.

Kothari, S. P., and Jay A. Shanken. 1997. "Book-to-market, dividend yield, and expected market returns: A time-series analysis." *Journal of Financial Economics* 44 (2): 169–203. https://doi.org/10.1016/S0304-405X(97)00002-0.

Lamont, Owen. 1998. "Earnings and expected returns." *The Journal of Finance* 53 (5): 1563–87. https://doi.org/10.1111/0022-1082.00065.

Lintner, John. 1965. "Security prices, risk, and maximal gains from diversification." *The Journal of Finance* 20 (4): 587–615. https://doi.org/10.1111/j.1540-6261.1965.tb02930.x.

Markowitz, Harry. 1952. "Portfolio selection." *The Journal of Finance* 7 (1): 77–91. https://doi.org/10.1111/j.1540-6261.1952.tb01525.x.

Mossin, Jan. 1966. "Equilibrium in a capital asset market." *Econometrica* 34 (4): 768–83. https://doi.org/10.2307/1910098.

Peng, Roger D. 2011. "Reproducible Research in Computational Science." *Science* 334 (6060): 1226–27.

Sandve, Geir Kjetil, Anton Nekrutenko, James Taylor, and Eivind Hovig. 2013. "Ten Simple Rules for Reproducible Computational Research." *PLoS Computational Biology* 9 (10): e1003285.

Sharpe, William F. 1964. "Capital asset prices: A theory of market equilibrium under conditions of risk ." *The Journal of Finance* 19 (3): 425–42. https://doi.org/10.1111/j.1540-6261.1964.tb02865.x.

Vilhuber, Lars. 2020. "Reproducibility and Replicability in Economics." *Harvard Data Science Review* 2 (4): 1–39.

Welch, Ivo, and Amit Goyal. 2008. "A comprehensive look at the empirical performance of equity premium prediction." *Review of Financial Studies* 21 (4): 1455–1508. https://doi.org/10.1093/rfs/hhm014.