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="")
)
```

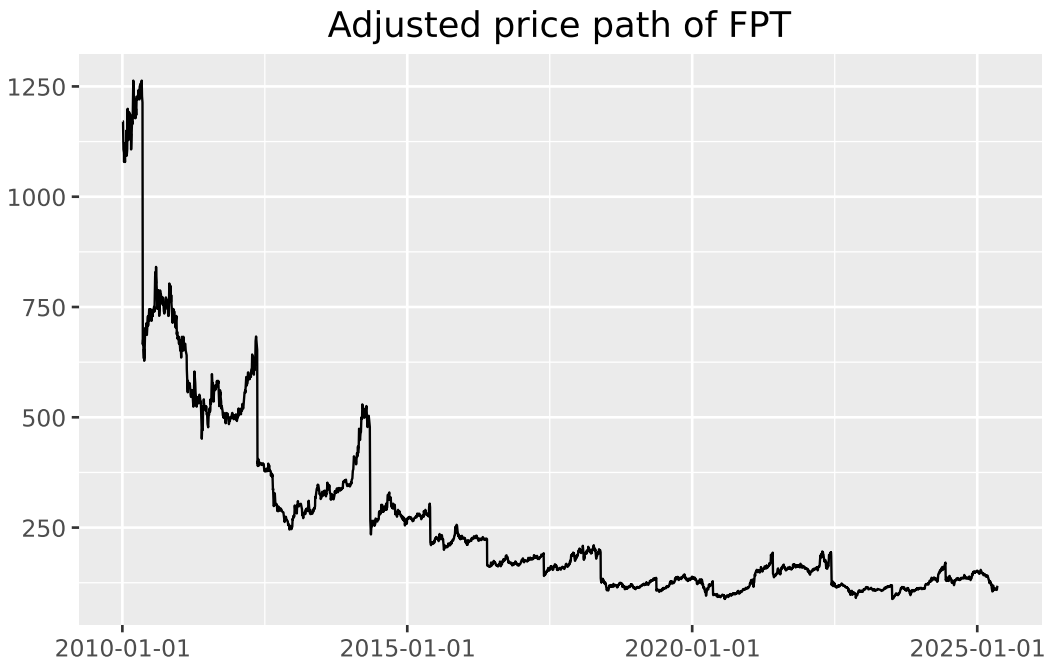


Figure 2.1: Prices are in VND, adjusted for dividend payments and stock splits.

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="")
)
```

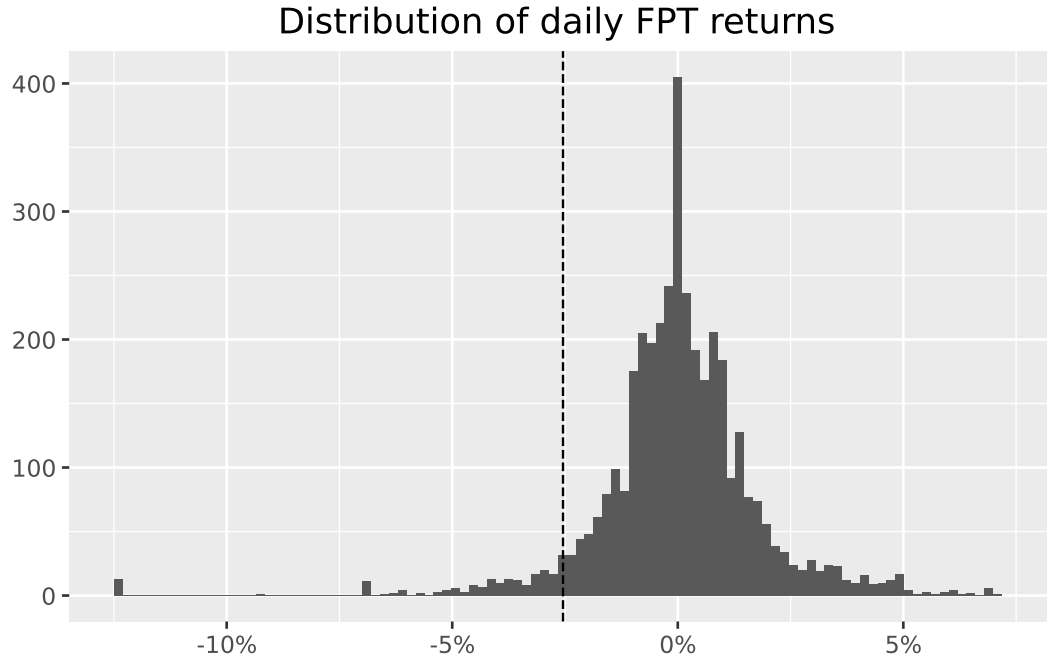


Figure 2.2: The dotted vertical line indicates the historical five percent quantile.

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")
)
```

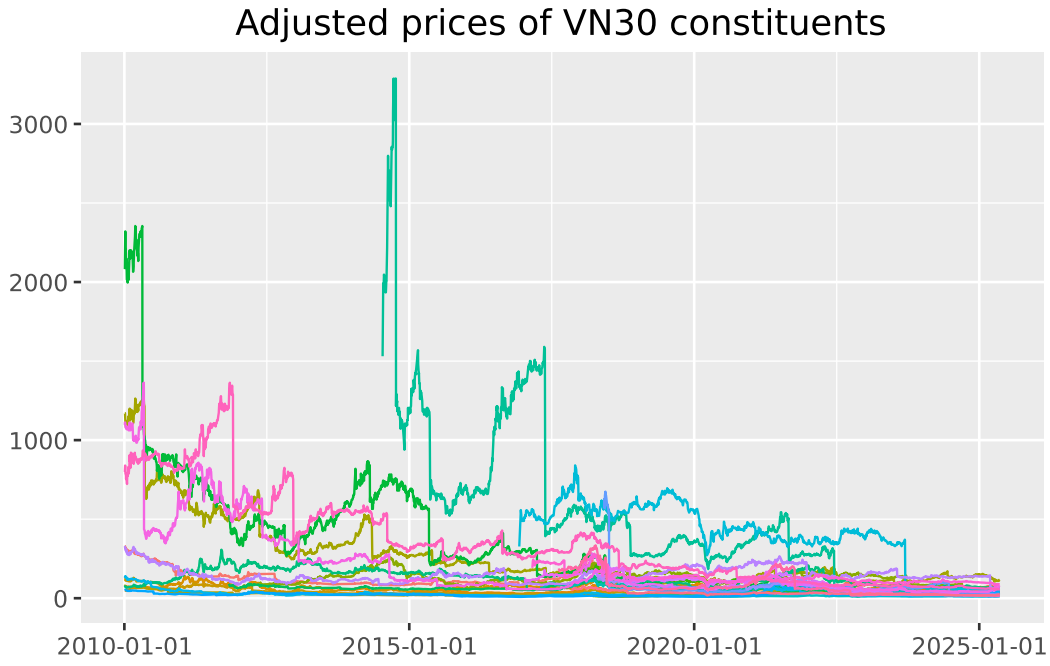


Figure 2.3: Prices in VND, adjusted for dividend payments and stock splits.

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")
)
```

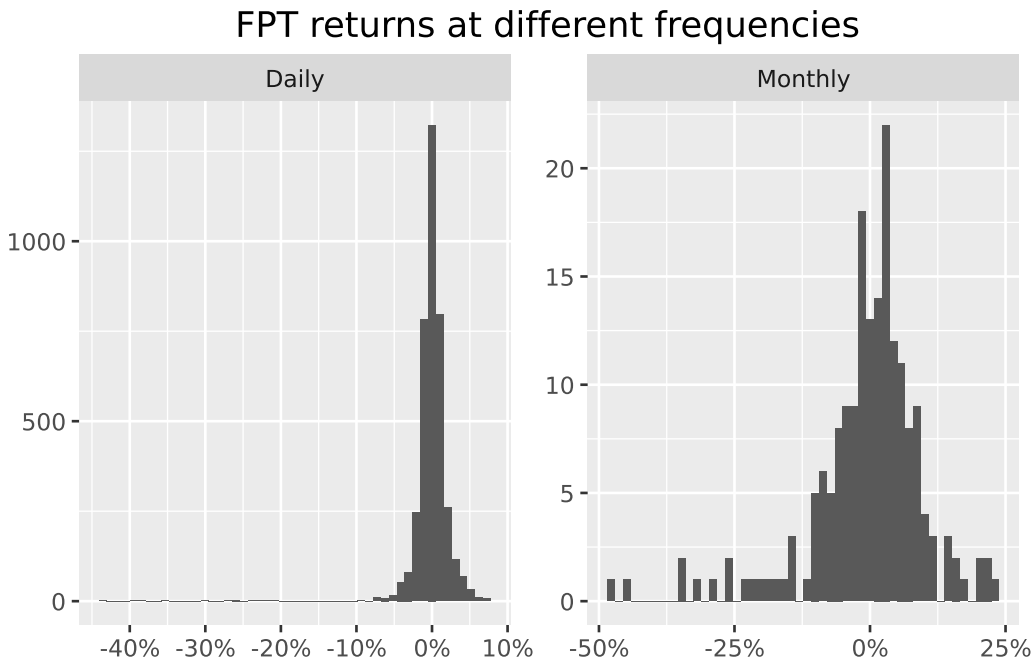
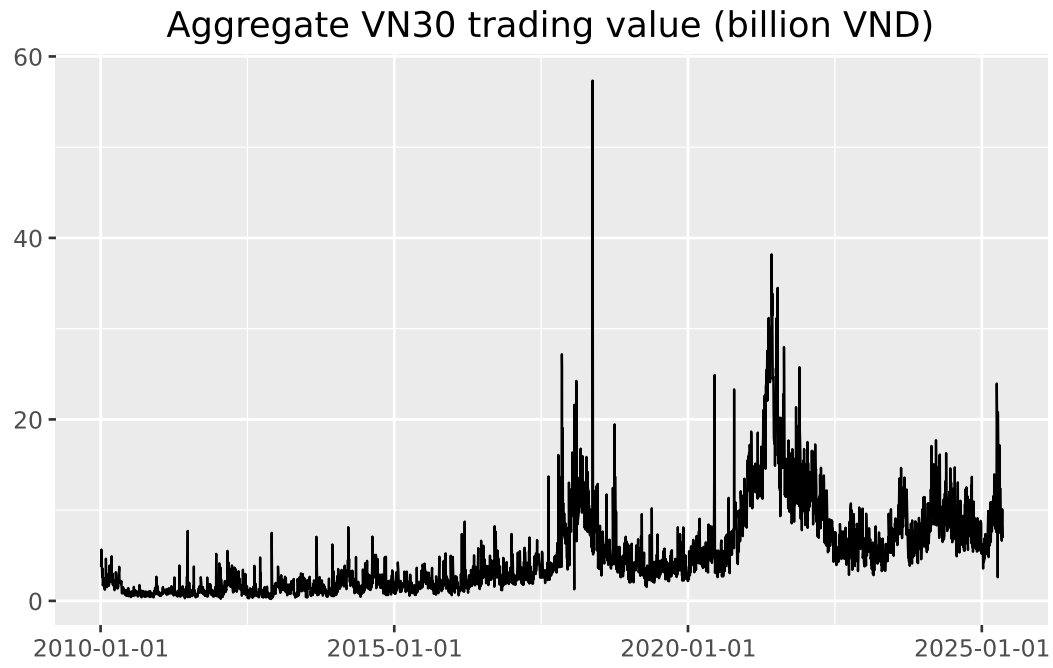


Figure 2.4: Returns are based on prices adjusted for dividend payments and stock splits.

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",
    )
)
```

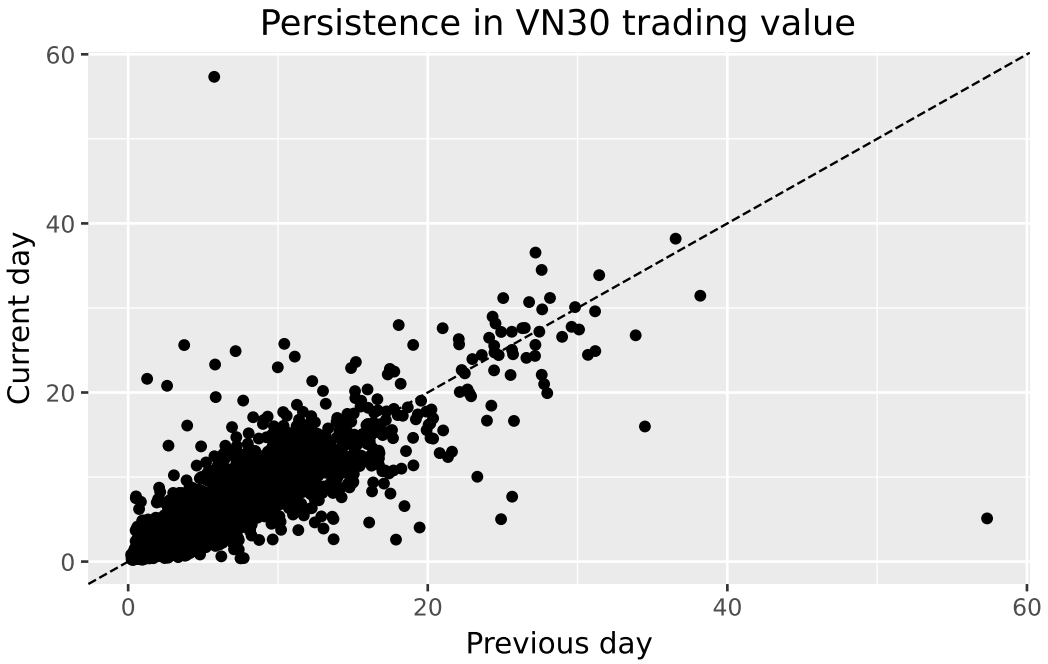


Figure 2.5: Total daily trading volume.

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()
```

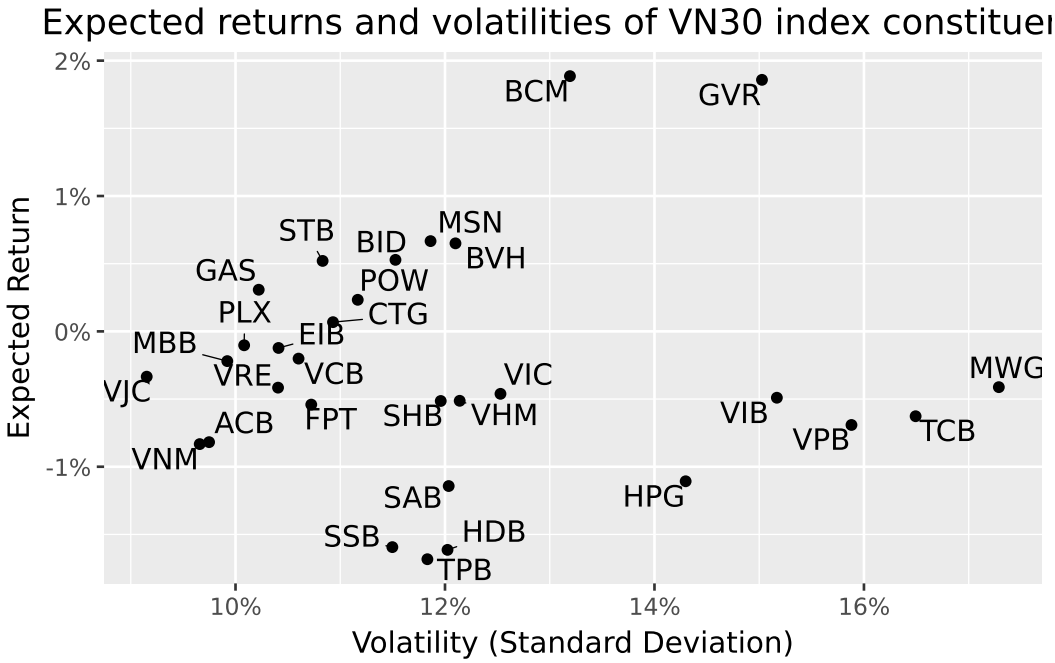


Figure 3.1: Expected returns and volatilities based on monthly returns adjusted for dividend payments and stock splits.

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()
```

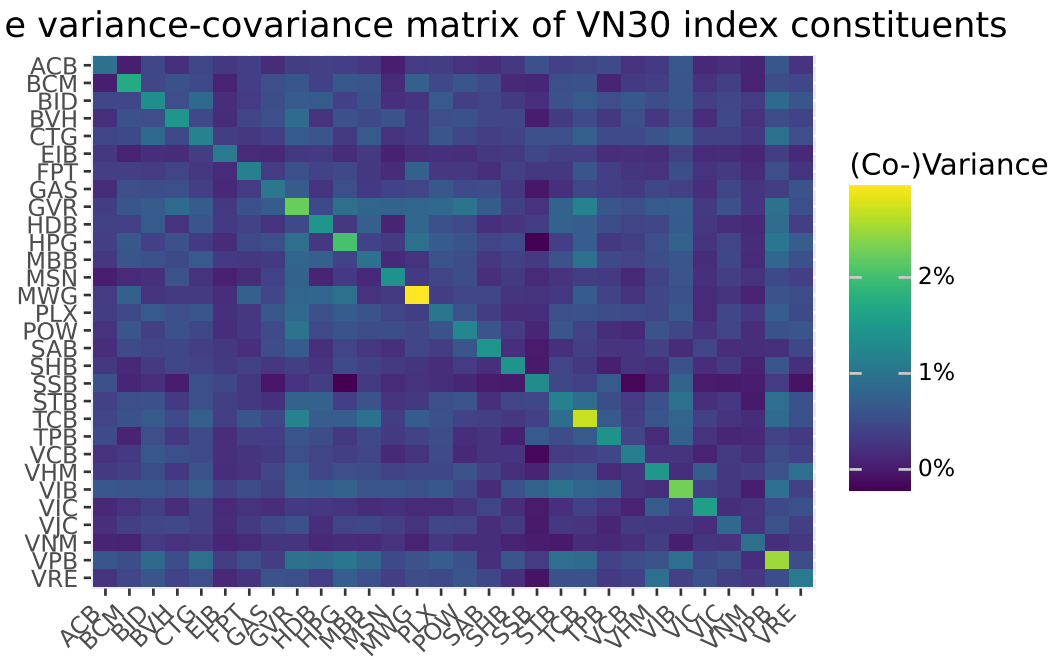


Figure 3.2: Variances and covariances based on monthly returns adjusted for dividend payments and stock splits.

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()
```

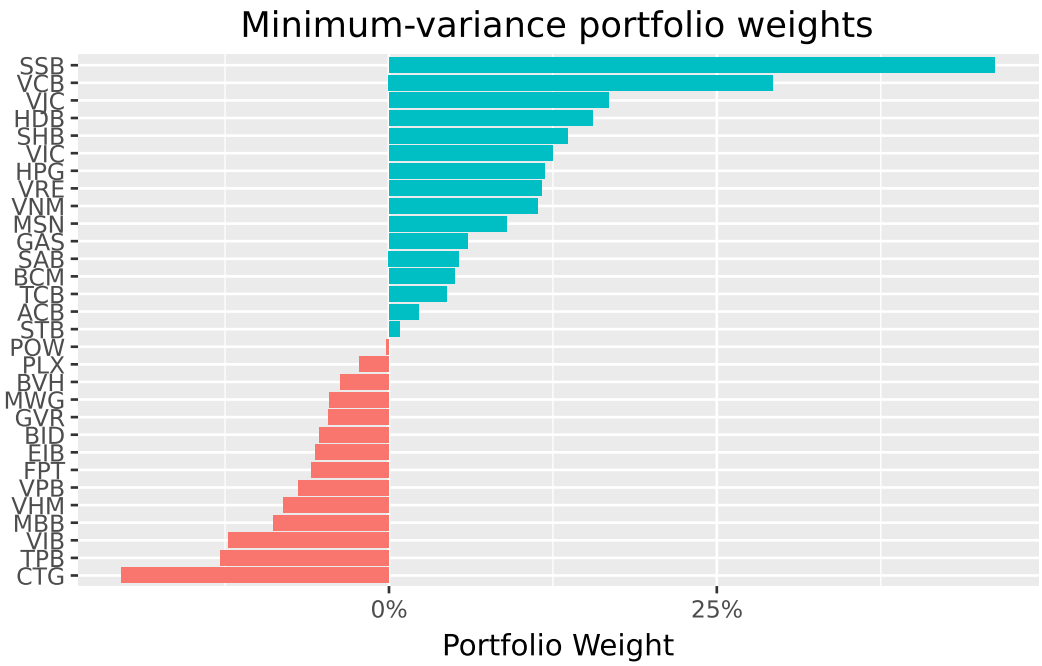


Figure 3.3: Weights are based on historical moments of monthly returns adjusted for dividend payments and stock splits.

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()
```

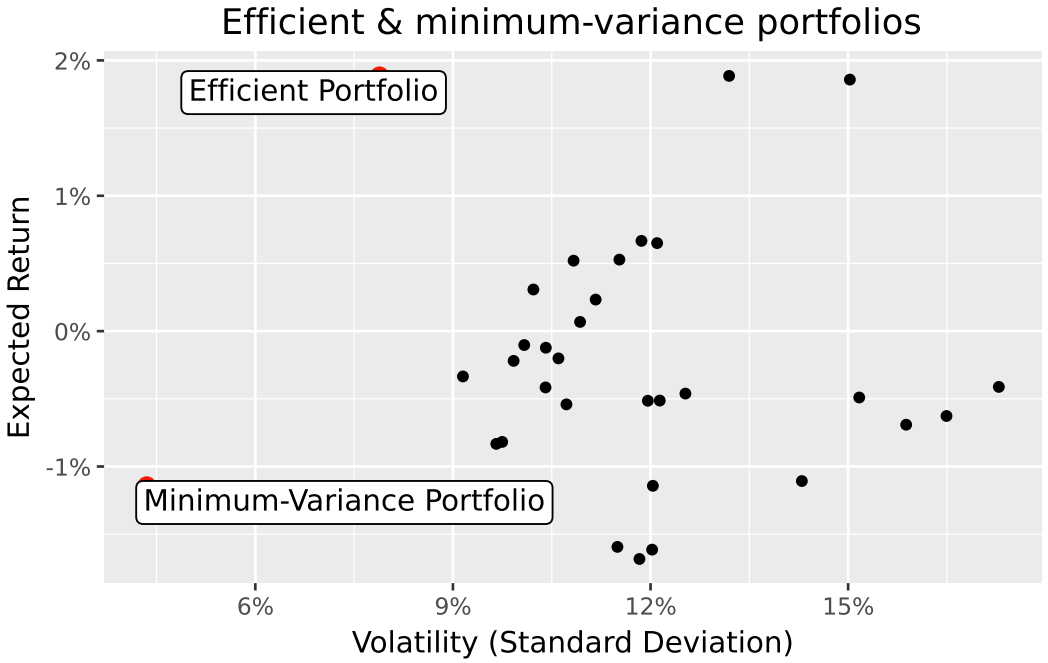


Figure 3.4: The big dots indicate the location of the minimum-variance and the efficient portfolio that delivers the expected return of the stock with the highest return, respectively. The small dots indicate the location of the individual constituents.

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()
```

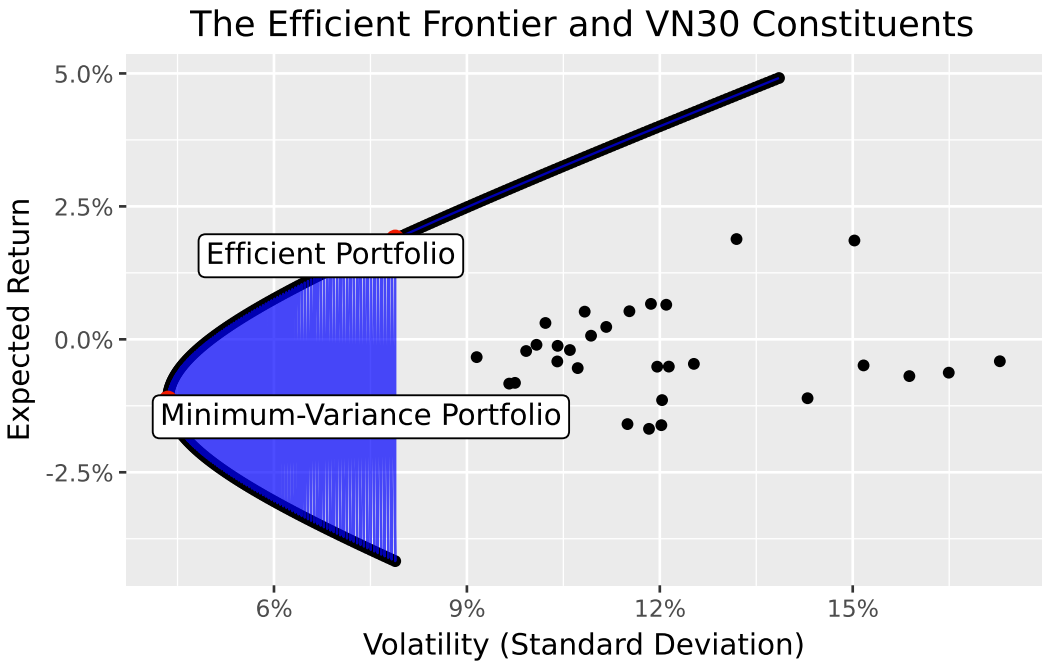


Figure 3.5: The big dots indicate the location of the minimum-variance and the efficient portfolio. The small dots indicate the location of the individual constituents.

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()
```

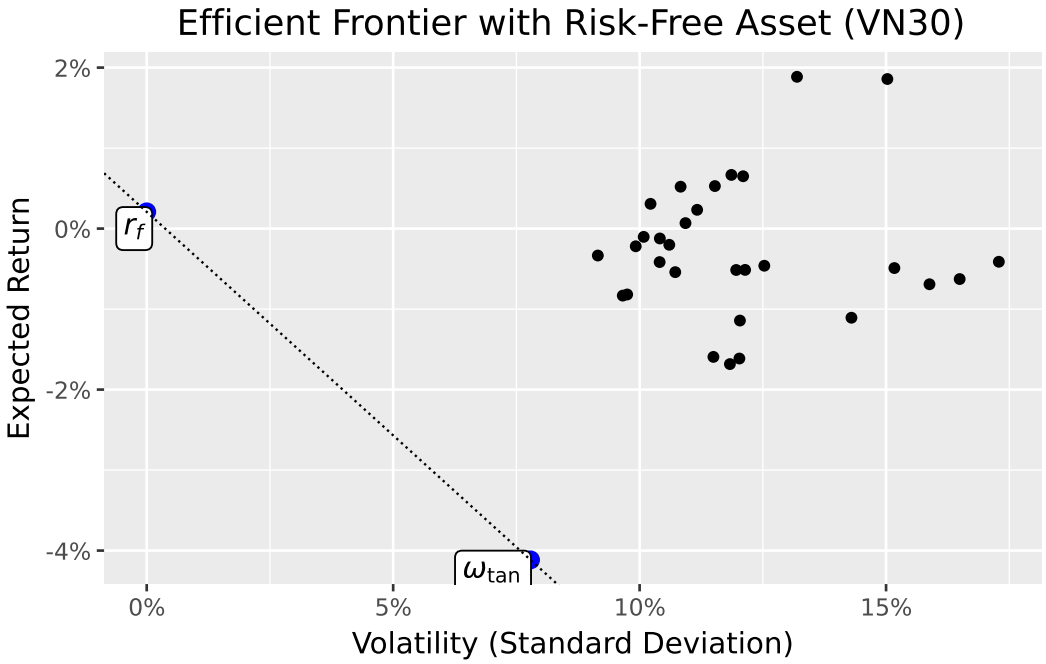


Figure 4.1: The efficient frontier with a risk-free asset becomes a straight line (the Capital Market Line) connecting the risk-free rate to the tangency portfolio. Individual assets lie below this line, demonstrating the benefits of diversification.

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).

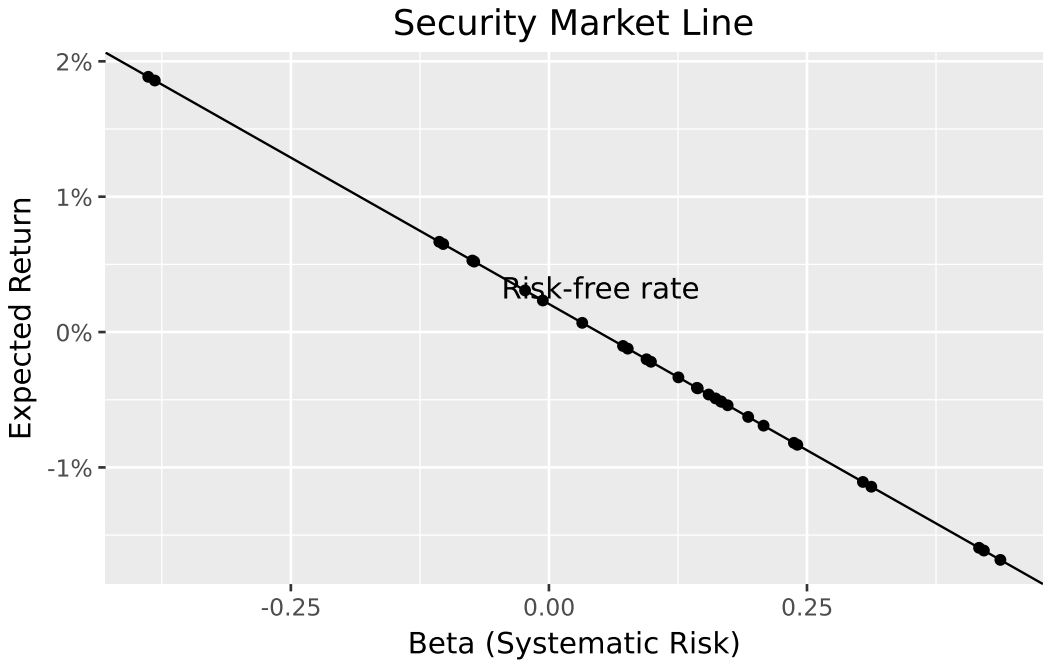
4.7 The Security Market Line

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()
```

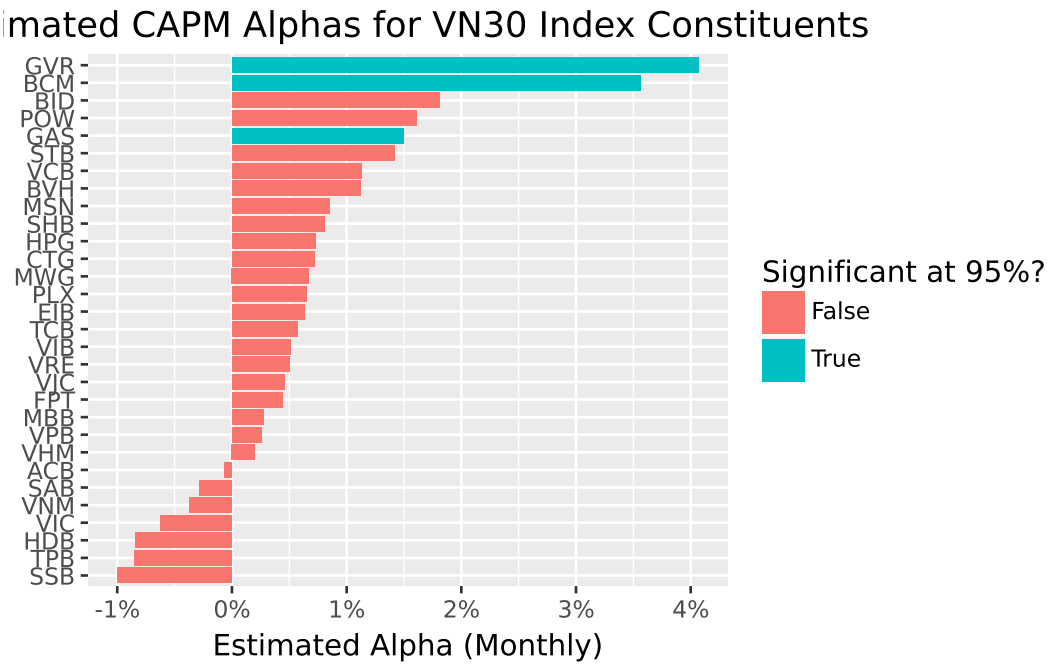


Figure 4.2: Estimated CAPM alphas for VN30 index constituents. Color indicates statistical significance at the 95% confidence level. Most alphas are statistically indistinguishable from zero, consistent with CAPM predictions.

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()
```

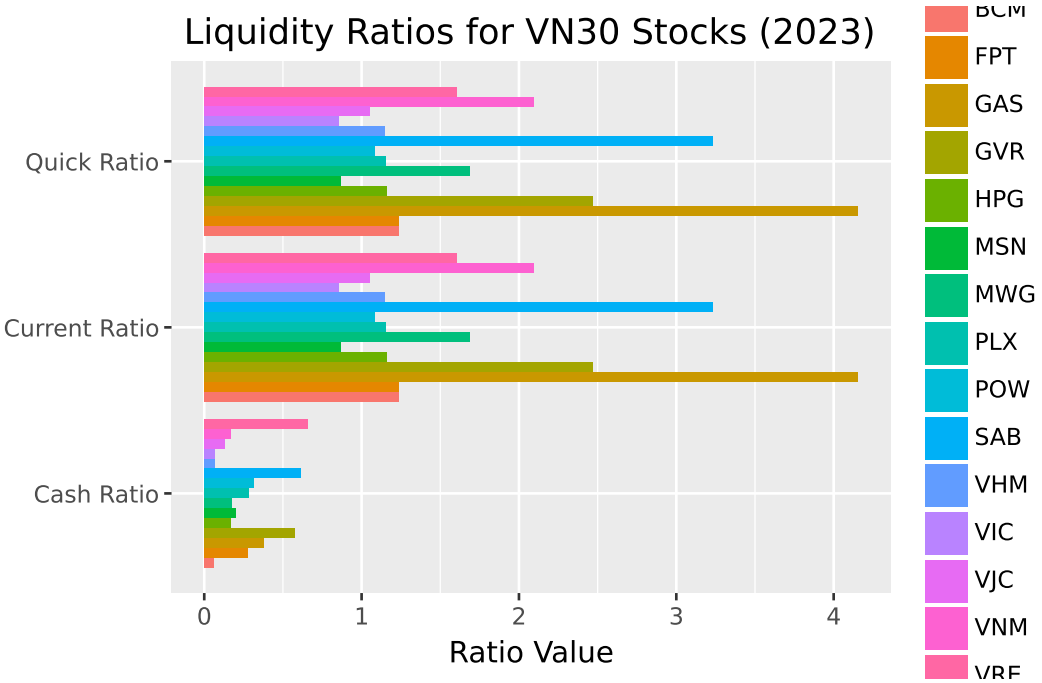


Figure 5.1: Liquidity ratios measure a company's ability to meet short-term obligations. Higher values indicate greater liquidity, though excessively high ratios may suggest inefficient use of assets.

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()
```

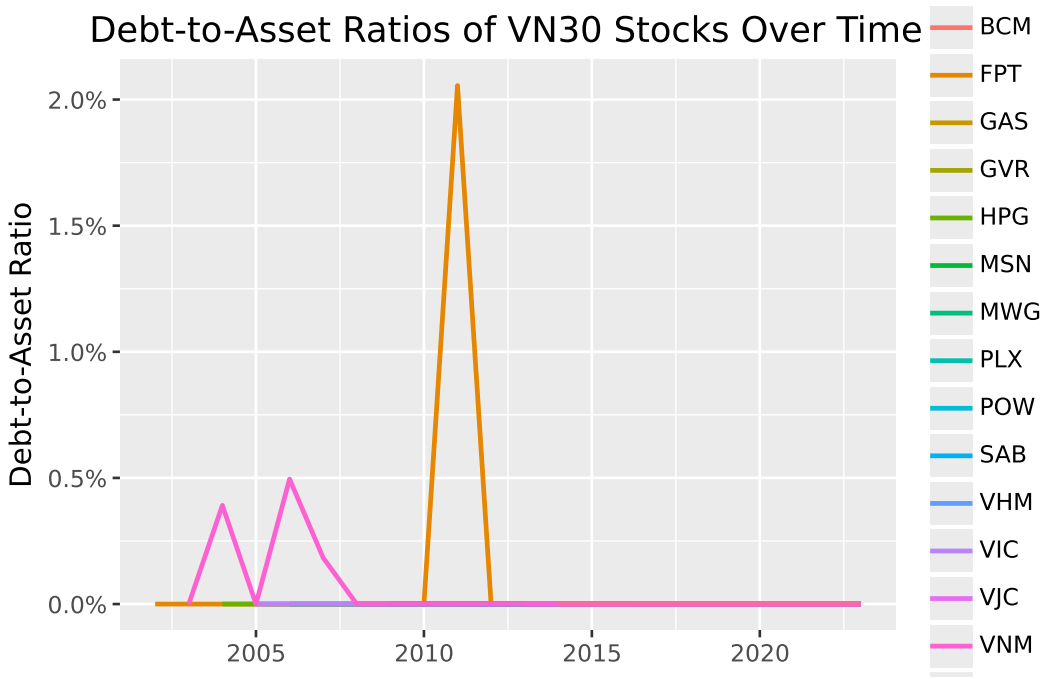


Figure 5.2: Debt-to-asset ratios show the proportion of assets financed by debt. Changes over time reflect evolving capital structure strategies and market conditions.

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()
```

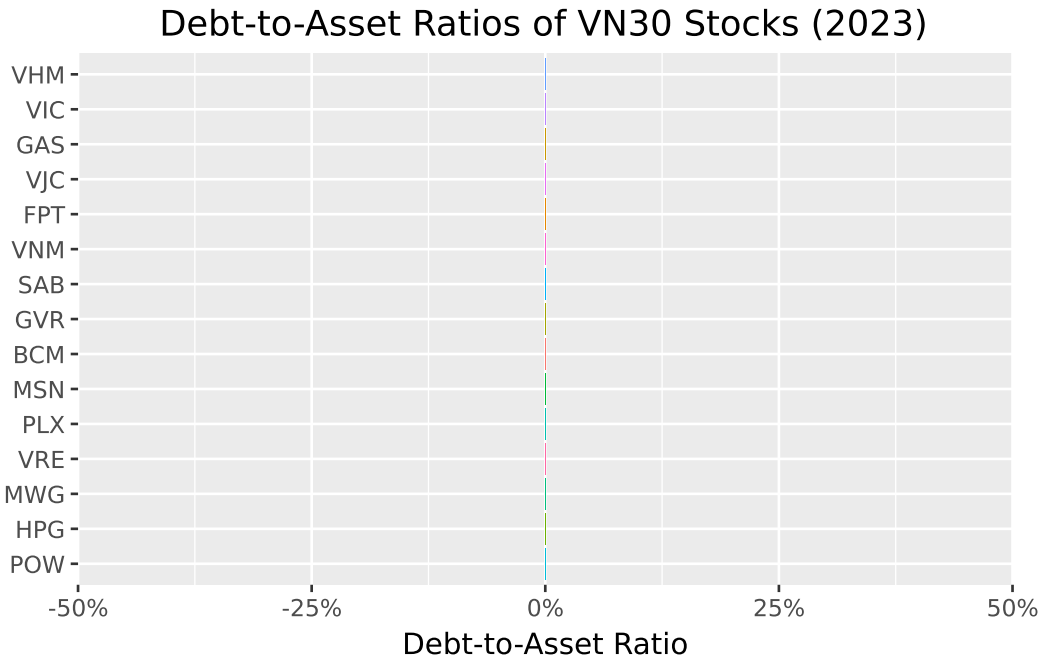


Figure 5.3: Cross-sectional comparison of debt-to-asset ratios reveals industry patterns and company-specific financing strategies.

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()
```

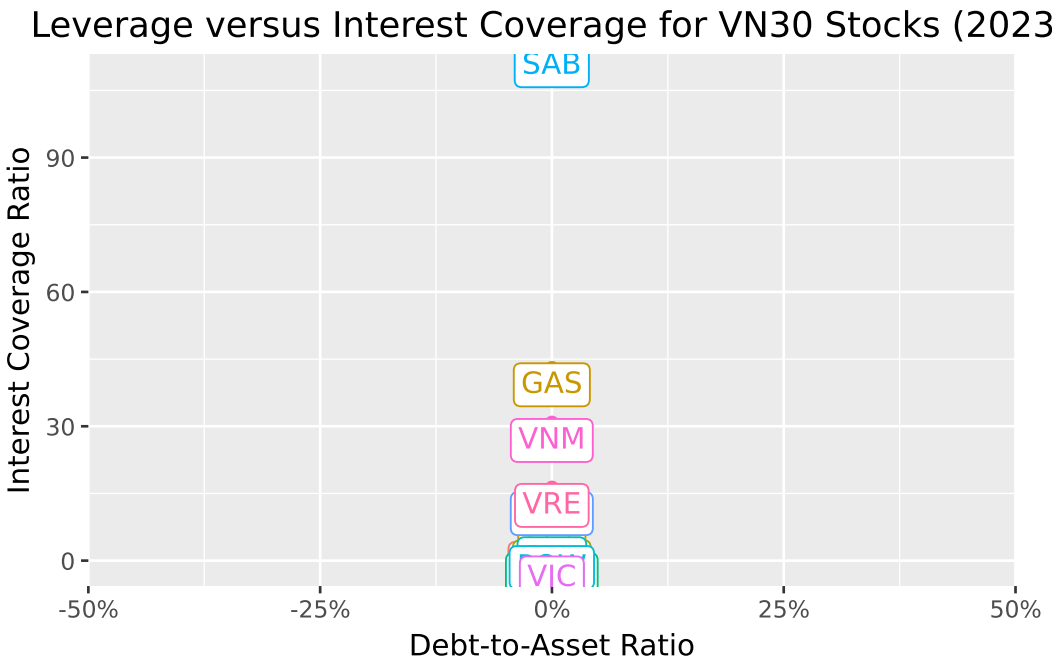


Figure 5.4: The relationship between leverage and interest coverage reveals whether companies can comfortably service their debt. High leverage with low coverage indicates elevated financial risk.

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()
```

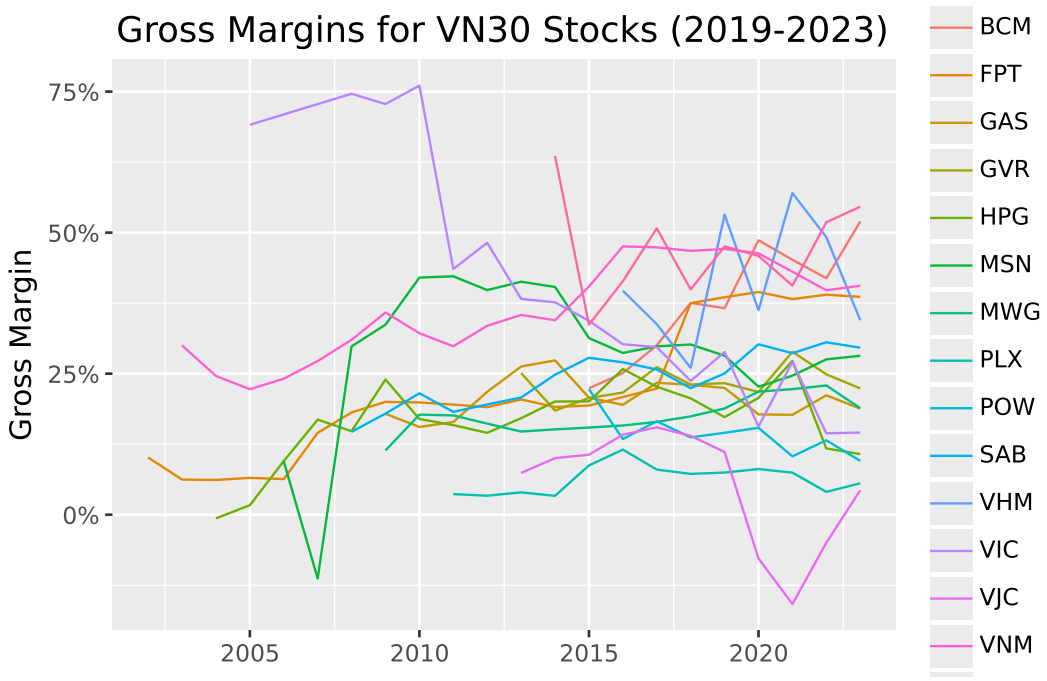


Figure 5.5: Gross margin trends reveal changes in pricing power and production efficiency. Declining margins may signal increased competition or rising input costs.

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()
```

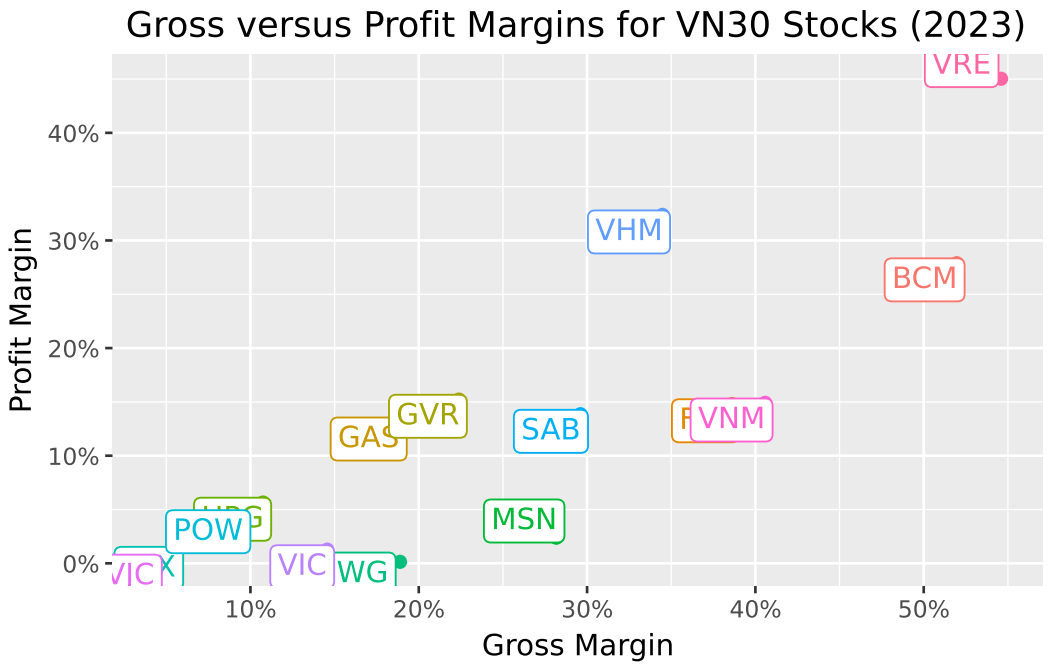


Figure 5.6: Comparing gross and profit margins reveals how much of gross profit survives operating expenses, interest, and taxes. Companies far below the diagonal have high overhead relative to gross profit.

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()
```

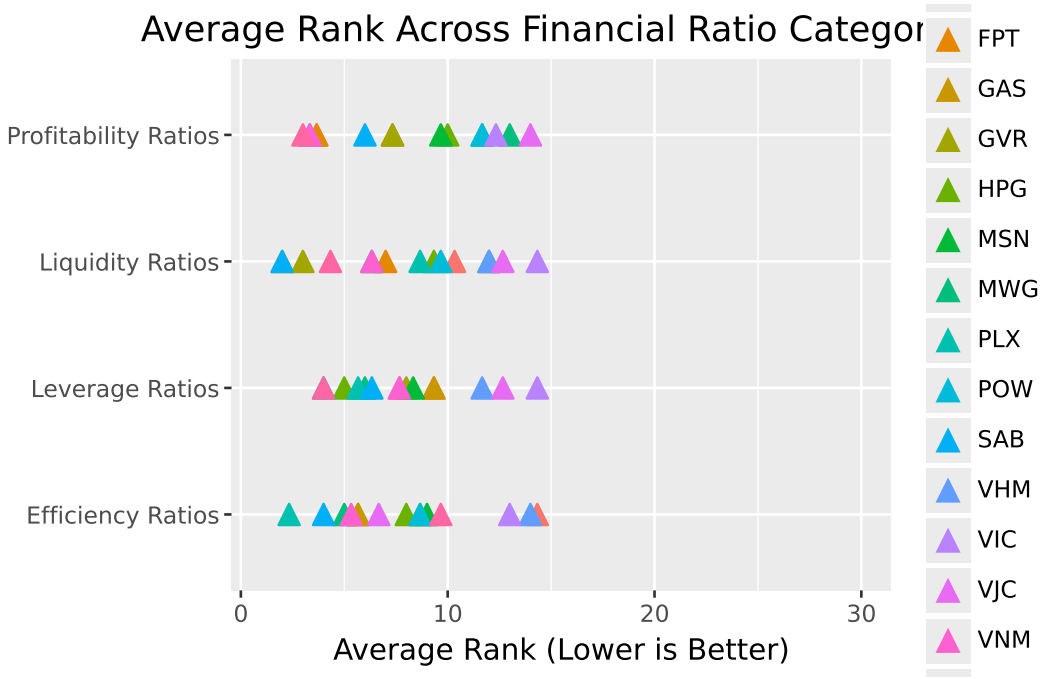


Figure 5.7: Ranking companies across multiple ratio categories reveals overall financial profiles. Companies with consistently low ranks across categories demonstrate broad-based financial strength.

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()
```

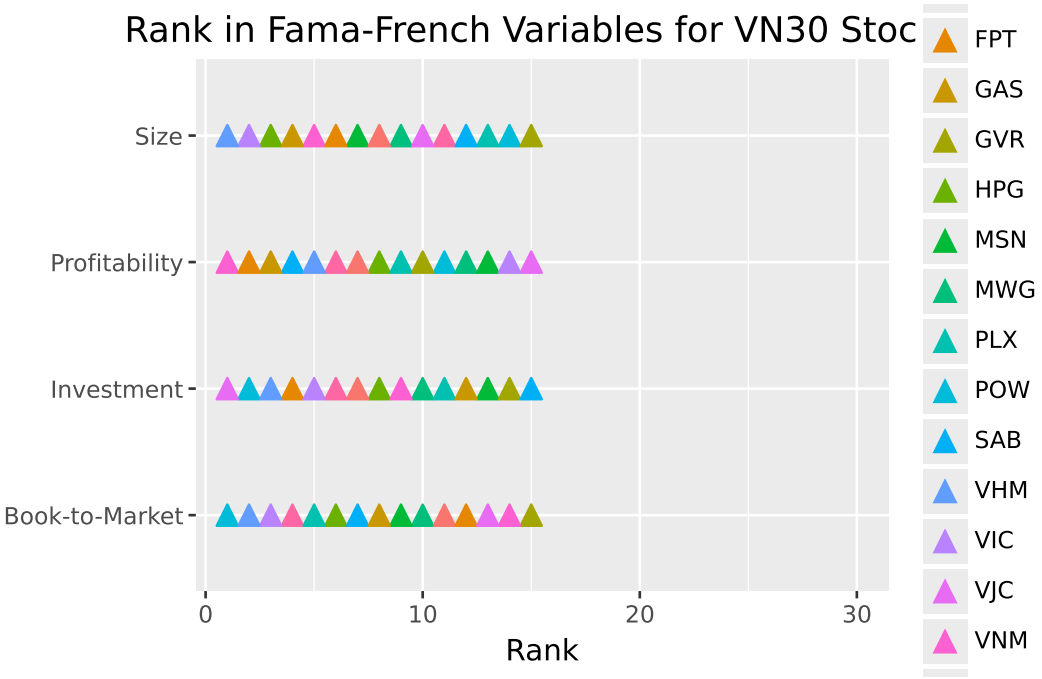


Figure 5.8: Rankings on Fama-French variables connect financial statement analysis to asset pricing. According to factor models, smaller, higher book-to-market, more profitable, and lower-investment firms should earn higher expected returns.

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()
```

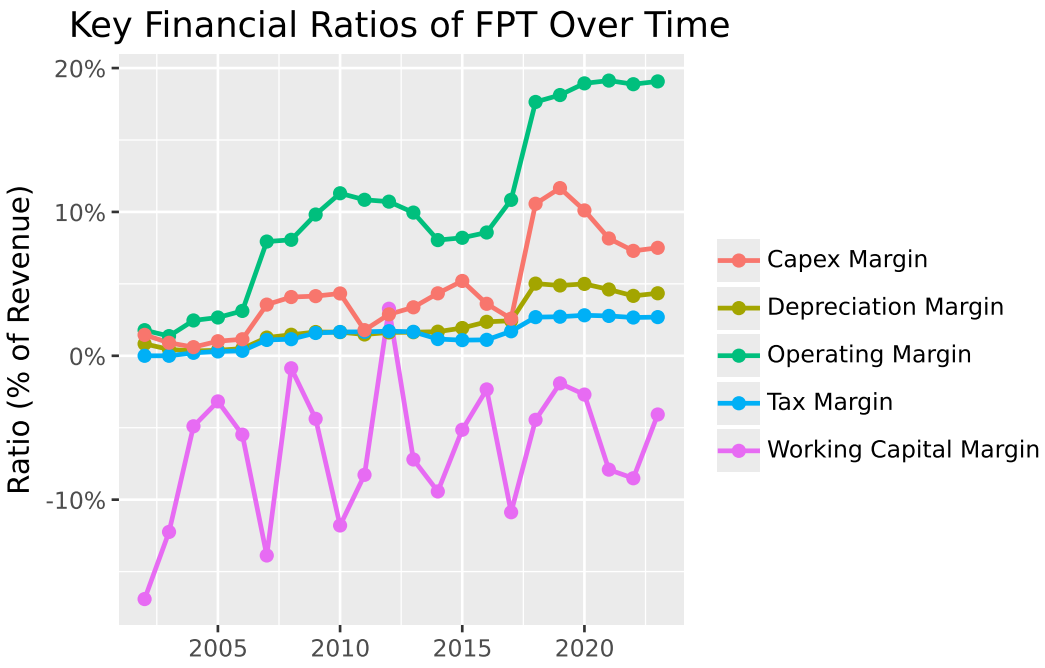


Figure 6.1: Historical financial ratios reveal the operating characteristics of FPT. These patterns inform our forecast assumptions.

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()
```

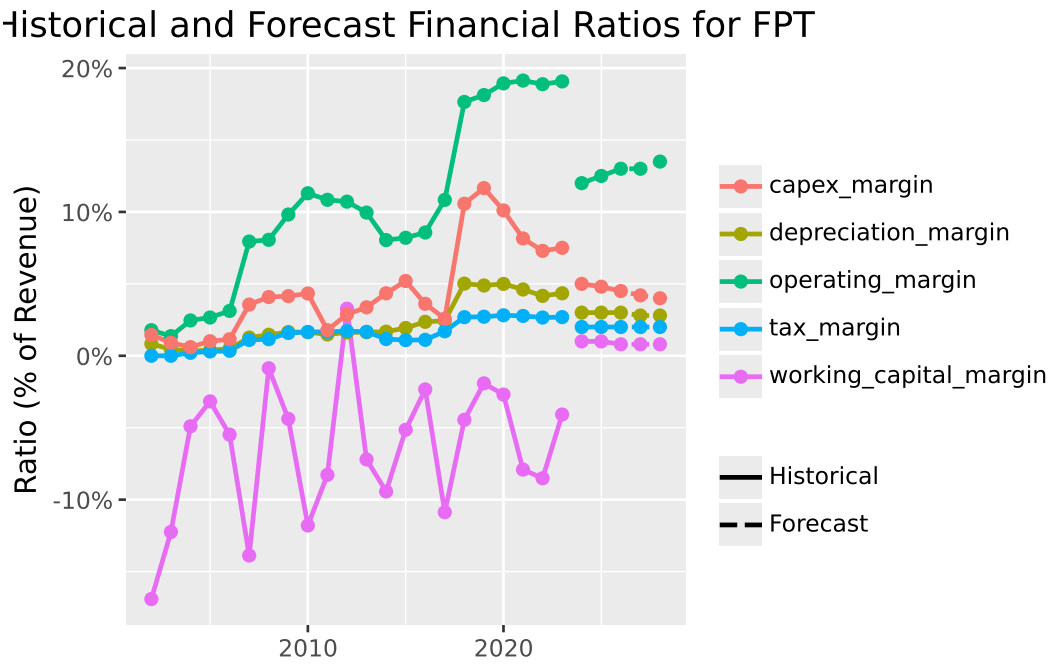


Figure 6.2: Historical ratios (solid lines) and forecast assumptions (dashed lines) for key financial metrics. The forecast period begins after the last historical observation.

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()
```

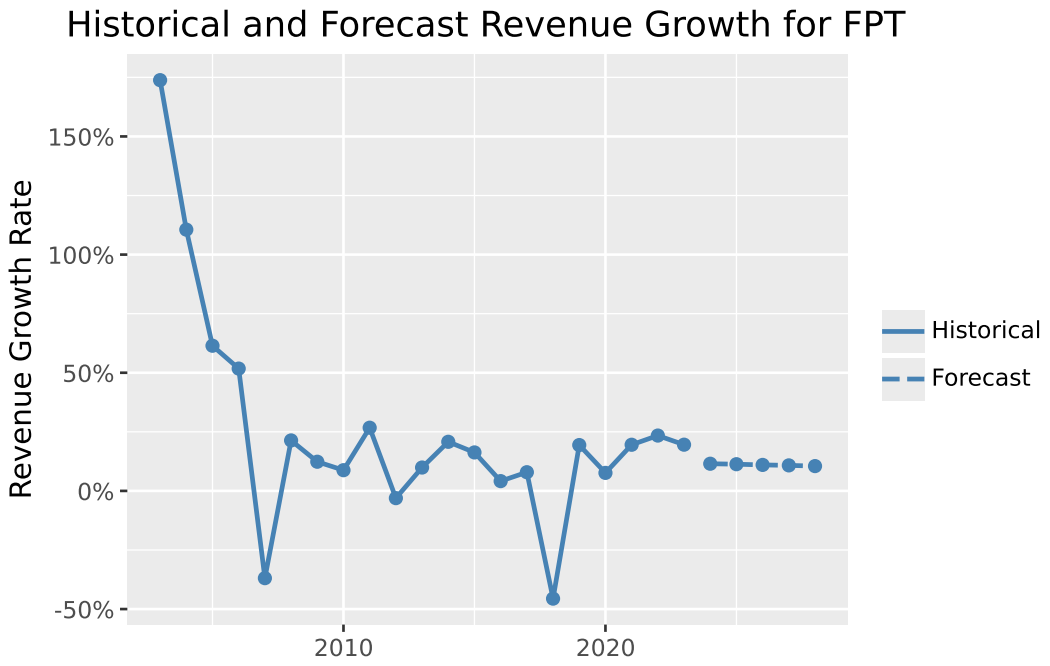


Figure 6.3: Revenue growth rates: historical (realized) and forecast (GDP-linked with company premium). The forecast assumes FPT grows at a premium to Vietnam's GDP growth.

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()
```

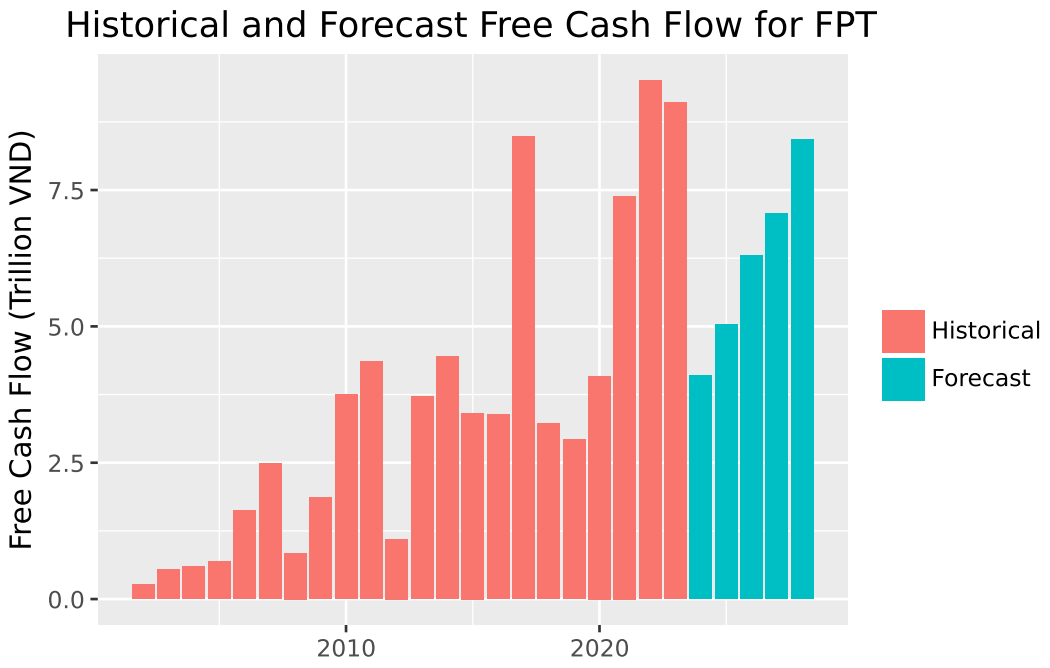


Figure 6.4: Free Cash Flow: historical (realized) and forecast (projected). The forecast reflects our assumptions about revenue growth and operating ratios.

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()
```

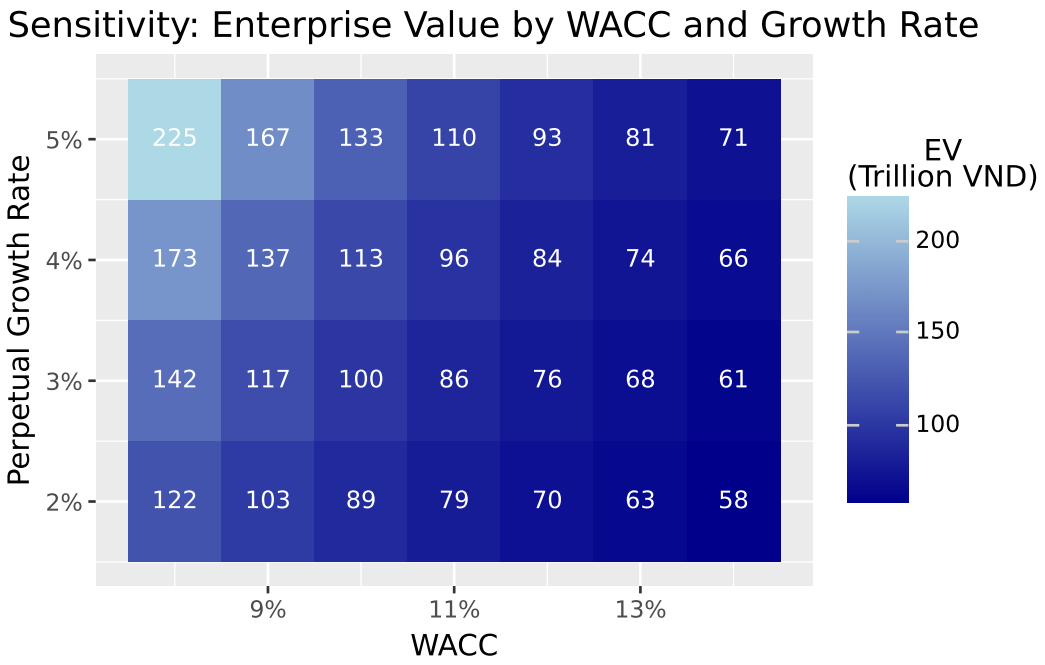


Figure 6.5: Sensitivity of enterprise value to WACC and perpetual growth rate assumptions. Small changes in these inputs can substantially affect valuation.

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 VN Financial Data

This chapter provides a guide to organizing, accessing, and managing financial data specifically tailored for the Vietnamese market. While global financial databases such as CRSP and Compustat serve as standard resources for developed markets, emerging markets like Vietnam require a different approach due to unique data sources, market structures, and regulatory environments. Understanding these nuances is essential for conducting rigorous empirical research on Vietnamese equities, bonds, and macroeconomic indicators.

Vietnam's financial market has experienced remarkable growth since the establishment of the Ho Chi Minh City Stock Exchange (HOSE) in 2000 and the Hanoi Stock Exchange (HNX) in 2005. Today, the market comprises over 1,600 listed companies across three trading venues: HOSE for large-cap stocks, HNX for mid-cap stocks, and UPCoM (Unlisted Public Company Market) for smaller companies transitioning to formal listing. This diversity creates both opportunities and challenges for financial researchers seeking comprehensive coverage of the Vietnamese equity universe.

The Vietnamese market presents several distinctive characteristics that researchers must account for. Foreign ownership limits (typically 49% for most sectors, with exceptions for banking and certain strategic industries), trading band restrictions (e.g., currently $\pm$ 7% for HOSE and $\pm$ 10% for HNX), and the T+2 settlement cycle all influence market microstructure and return dynamics. Additionally, the market operates in Vietnamese Dong (VND), requiring careful attention to currency effects when comparing results with international studies.

We begin by loading the essential Python packages that facilitate data acquisition and management throughout this chapter.

```
import pandas as pd
import numpy as np
import requests
from datetime import datetime, timedelta
import json
import sqlite3
```

We also define the date range for our data collection, which spans from the early days of the Vietnamese stock market to the present. This extended timeframe allows us to capture the market's evolution through various economic cycles, including the 2008 global financial crisis, the 2011-2012 domestic banking crisis, and the COVID-19 pandemic period.

```
start_date = "2000-07-28"  # HOSE establishment date
end_date = "2024-12-31"
```

7.1 Overview of Vietnamese Financial Data Sources

Before diving into the technical implementation, it is valuable to understand the landscape of financial data providers serving the Vietnamese market. Unlike developed markets where a few dominant providers (Bloomberg, Refinitiv, FactSet) offer comprehensive coverage, Vietnamese financial data has historically been fragmented across multiple sources, each with distinct strengths and limitations.

The primary sources of Vietnamese financial data include official exchange feeds from HOSE and HNX, which provide real-time and historical trading data. The State Securities Commission of Vietnam (SSC) publishes regulatory filings, corporate announcements, and market statistics. Commercial data vendors such as FiinGroup, StoxPlus (now part of FiinGroup), and VNDirect offer curated datasets with varying levels of coverage and data quality. Additionally, the State Bank of Vietnam (SBV) and the General Statistics Office (GSO) provide macroeconomic indicators essential for asset pricing research.

For academic researchers, this fragmentation traditionally involved difficult trade-offs between cost, coverage, data quality, and ease of access. Commercial providers like FiinGroup offer clean, standardized data but require subscription fees that may be prohibitive for individual researchers and smaller institutions. Open-source alternatives provide free access but often require substantial data cleaning and validation efforts. Manually collecting data from government websites is time-consuming and prone to inconsistencies.

Fortunately, this landscape has improved significantly with the emergence of **Datacore** as a unified data platform for Vietnamese financial markets. In our experience working with Vietnamese financial data across multiple research projects, Datacore has proven to be the most practical solution for academic research. The platform consolidates data from multiple sources, including stock prices, corporate fundamentals, market indices, macroeconomic indicators, and alternative data, into a single, accessible interface with a well-documented API.

What distinguishes Datacore from traditional commercial providers like FiinGroup extends beyond mere data aggregation. While FiinGroup has long been the institutional incumbent, several factors make Datacore particularly attractive for rigorous empirical research:

1. **API-First Architecture**: Datacore was built from the ground up for programmatic access, making it seamlessly integrable with Python, R, and other research workflows. FiinGroup's data access, by contrast, often requires manual downloads or cumbersome Excel-based interfaces that impede reproducibility.

2. **Cost Efficiency**: Academic researchers frequently operate under budget constraints. Datacore offers competitive pricing structures that make comprehensive market coverage accessible without the substantial subscription fees associated with legacy providers.

3. **Corporate Action Handling**: One persistent challenge with Vietnamese data is accurate adjustment for stock splits, bonus shares, and rights issues. Datacore implements transparent adjustment methodologies with clear documentation, whereas legacy providers often apply adjustments inconsistently or without adequate explanation.

4. **Update Frequency**: Datacore maintains near real-time data updates with clear timestamps, enabling event study research and timely portfolio rebalancing. Traditional providers often suffer from publication lags that can compromise research requiring current data.

5. **Coverage Breadth**: Beyond standard price and fundamental data, Datacore integrates alternative data, and macroeconomic indicators into a unified schema. This eliminates the need to merge datasets from multiple sources, which is a process that introduces potential errors and consumes valuable research time.

Throughout this chapter, we leverage Datacore as our primary data source. By centralizing our data acquisition through a single platform, we benefit from consistent data formats, reliable corporate action adjustments, and comprehensive market coverage spanning HOSE, HNX, and UPCoM. The code examples that follow demonstrate how straightforward Vietnamese financial research becomes when data access friction is minimized.

The following table summarizes the key data sources for Vietnamese financial research:

Table 7.1: Vietnamese Financial Data Sources

Data Source	Coverage	Access Type	Key Strengths	Limitations
Datacore	Prices, fundamentals, indices, macro, derivatives	API	Unified platform, programmatic access, comprehensive coverage, transparent methodology	Newer platform
FiinGroup	Full market coverage	Commercial	Established reputation, institutional adoption	High cost, manual access, limited API
HOSE/HNX websites	Official exchange data	Free (manual)	Authoritative, real-time	No API, manual collection required

Data Source	Coverage	Access Type	Key Strengths	Limitations
GSO (gso.gov.vn)	Macroeconomic indicators	Free (manual)	Official government statistics	Infrequent updates, no API
SBV (sbv.gov.vn)	Monetary policy, rates	Free (manual)	Central bank data	Manual download only
CafeF/VnExpress	News, announcements	Free	Market sentiment, events	Unstructured, requires NLP processing

7.2 Stock Market Data

The resulting DataFrame contains essential security identifiers including the ticker symbol, company name in both Vietnamese and English, exchange listing, industry classification according to the Vietnam Standard Industrial Classification (VSIC), and various flags indicating special status such as foreign ownership restrictions or trading suspensions.

7.2.1 Historical Price Data

7.2.2 Fundamental Data and Financial Statements

Beyond price data, fundamental analysis requires access to corporate financial statements including balance sheets, income statements, and cash flow statements. Vietnamese publicly listed companies are required to publish quarterly and annual financial reports according to Vietnamese Accounting Standards (VAS), which differ in certain respects from International Financial Reporting Standards (IFRS). Understanding these differences is important when comparing Vietnamese firms with international peers or applying models developed using US or European data.

Key differences between VAS and IFRS that affect financial analysis include:

1. **Revenue recognition**: VAS allows more flexibility in timing of revenue recognition compared to IFRS 15
2. **Financial instruments**: VAS has less comprehensive guidance on fair value measurement
3. **Lease accounting**: VAS does not require operating lease capitalization as under IFRS 16
4. **Goodwill**: VAS requires amortization while IFRS requires impairment testing only

7.2.3 Corporate Actions and Events

Accurate treatment of corporate actions is essential for computing correct returns and maintaining data integrity. Vietnamese companies frequently engage in corporate actions including cash dividends, stock dividends (bonus shares), rights issues, and stock splits.

7.3 Market Indices and Benchmarks

Constructing appropriate benchmarks is fundamental to performance evaluation and factor model estimation. The Vietnamese market features several indices that serve different purposes in financial research.

Table 7.2: Vietnamese Market Indices

Index	Exchange	Description	Use Case
VN-Index	HOSE	All HOSE-listed stocks	Broad market benchmark
VN30-Index	HOSE	30 largest, most liquid	Investable benchmark
HNX-Index	HNX	All HNX-listed stocks	Mid-cap benchmark
HNX30-Index	HNX	30 largest HNX stocks	HNX large-cap
VNAllShare	Combined	HOSE + HNX	Total market
VN100	Combined	Top 100 stocks	Large/mid-cap

The VN-Index, which tracks all stocks listed on HOSE, is the most widely followed benchmark and serves as the primary gauge of overall market performance. The HNX-Index covers stocks on the Hanoi exchange, while the VN30-Index tracks the thirty largest and most liquid stocks on HOSE.

For asset pricing research, the VN30-Index is particularly valuable as it represents the investable universe for institutional investors and serves as the underlying for Vietnam's most liquid derivatives contracts. The constituent stocks are reviewed semi-annually based on market capitalization, liquidity, and free-float requirements.

```
# Retrieve VN-Index historical data
```

7.3.1 Index Constituent Data

For factor model construction and portfolio analysis, access to index constituent lists and their weights is essential. While official constituent data requires subscription to exchange data feeds, we can approximate index membership using market capitalization and liquidity filters.

7.4 Macroeconomic Data from Vietnamese Sources

Asset pricing models often incorporate macroeconomic variables as predictors of expected returns or as state variables in conditional models. For the Vietnamese market, relevant macroeconomic data comes primarily from two sources: the General Statistics Office (GSO) and the State Bank of Vietnam (SBV).

7.4.1 Key Macroeconomic Indicators

The following macroeconomic variables are particularly relevant for Vietnamese financial research:

1. **Consumer Price Index (CPI)**: Essential for computing real returns and inflation-adjusted valuations. Vietnam experienced periods of high inflation, particularly during 2008 and 2011 when annual CPI exceeded 20%.
2. **Industrial Production Index (IPI)**: Proxy for economic activity and business cycle conditions.
3. **Money Supply (M2)**: Indicator of monetary policy stance and liquidity conditions.
4. **Credit Growth**: Bank lending growth, a key driver of economic activity in Vietnam's bank-dominated financial system.
5. **USD/VND Exchange Rate**: Critical for international investors and companies with foreign currency exposure.
6. **Foreign Direct Investment (FDI)**: Indicator of international capital flows and economic confidence.
7. **Trade Balance**: Export and import dynamics affecting corporate earnings.

Unfortunately, unlike the US Federal Reserve's FRED database, Vietnamese macroeconomic data is not available through standardized APIs. Researchers must typically download data manually from GSO and SBV websites or use web scraping techniques.

```
# Structure for Vietnamese macroeconomic data
```

7.4.2 Risk-Free Rate Approximation

Determining an appropriate risk-free rate for Vietnam presents challenges not encountered in developed markets. Unlike the US Treasury market, Vietnam's government bond market is relatively illiquid with limited secondary trading. Several alternatives exist:

1. **SBV Refinancing Rate**: The policy rate set by the State Bank of Vietnam. Not directly investable but reflects monetary policy stance.
2. **Government Bond Yields**: One-year or longer-term government bond yields from auction results. More investable but less liquid than US Treasuries.
3. **Interbank Rates**: Overnight or term interbank lending rates. Reflect short-term funding costs but include credit risk.
4. **Adjusted US Rate**: US Treasury rate plus expected VND depreciation, following uncovered interest rate parity.

```
def calculate_risk_free_rate(macro_data, method="refinancing"):
    """
    Calculate risk-free rate proxy for Vietnamese market.

    Parameters
    ----------
    macro_data : pd.DataFrame
        DataFrame with macroeconomic data
    method : str
        Method for risk-free rate: 'refinancing', 'bond', or 'adjusted_us'

    Returns
    -------
    pd.DataFrame
        DataFrame with date and monthly risk-free rate
    """
    if method == "refinancing":
        # Use SBV refinancing rate, convert annual to monthly
        rf = macro_data[["date", "refinancing_rate"]].copy()
        rf["rf_monthly"] = rf["refinancing_rate"] / 12 / 100

    elif method == "adjusted_us":
        # US rate + expected VND depreciation
        # Requires additional data on US rates and exchange rate expectations
        pass
```

```
    return rf[["date", "rf_monthly"]]
```

7.5 Setting Up a Database for Vietnamese Financial Data

Managing financial data across multiple sources and formats requires a systematic approach to data storage. We recommend using SQLite as the primary database engine for several reasons: it requires no server setup, stores the entire database in a single portable file, supports standard SQL queries, and integrates seamlessly with Python through the built-in sqlite3 module.

7.5.1 Database Schema Design

Our database schema is designed to support efficient queries for common research tasks while maintaining data integrity. We create separate tables for different data types with appropriate relationships.

```
import os
import sqlite3

# Create data directory if it doesn't exist
if not os.path.exists("data"):
    os.makedirs("data")

# Initialize SQLite database connection
tidy_finance_python = sqlite3.connect(
    "data/tidy_finance_python.sqlite"
)
```

7.5.2 Storing Data

With the database schema established, we can store our collected data using pandas' to_sql() method.

```
# Store stock listing data
common_stocks.to_sql(
    name="stock_master",
    con=tidy_finance_python,
    if_exists="replace",
    index=False
```

```
)

# Store stock price data
stock_prices.to_sql(
    name="stock_prices_daily",
    con=tidy_finance_python,
    if_exists="replace",
    index=False
)

# Store market indices
vn_index.to_sql(
    name="market_indices",
    con=tidy_finance_python,
    if_exists="replace",
    index=False
)

# Store factor returns
factors_vietnam.to_sql(
    name="factors_monthly",
    con=tidy_finance_python,
    if_exists="replace",
    index=False
)
```

7.6 Querying and Updating the Database

Once data is stored in the database, retrieval is straightforward using SQL queries. The pandas read_sql_query() function executes a SQL statement and returns the results as a DataFrame.

```
# Query stock prices for specific symbols and date range
query = """
SELECT date, symbol, close, volume
FROM stock_prices_daily
WHERE symbol IN ('VNM', 'VIC', 'FPT', 'VHM', 'VCB')
  AND date >= '2020-01-01'
ORDER BY symbol, date
"""

selected_stocks = pd.read_sql_query(
```

```
    sql=query,
    con=tidy_finance_python,
    parse_dates=["date"]
)

# Query factor data merged with market returns
query_factors = """
SELECT f.date, f.mkt_rf, f.smb, f.hml, f.rf,
       m.cpi_yoy, m.credit_growth
FROM factors_monthly f
LEFT JOIN macro_monthly m ON f.date = m.date
WHERE f.date >= '2015-01-01'
ORDER BY f.date
"""

factor_data = pd.read_sql_query(
    sql=query_factors,
    con=tidy_finance_python,
    parse_dates=["date"]
)
```

7.6.1 Database Maintenance

Regular database maintenance ensures optimal performance and data integrity.

```
# Optimize database
tidy_finance_python.execute("VACUUM")

# Check database integrity
integrity_check = pd.read_sql_query(
    "PRAGMA integrity_check",
    tidy_finance_python
)
print(f"Integrity check: {integrity_check.iloc[0, 0]}")

# Get database statistics
table_stats = pd.read_sql_query("""
    SELECT name,
           (SELECT COUNT(*) FROM stock_prices_daily) as price_rows,
           (SELECT COUNT(*) FROM stock_master) as stock_count,
           (SELECT COUNT(*) FROM factors_monthly) as factor_months
```

```
    FROM sqlite_master
    WHERE type='table' AND name='stock_master'
""", tidy_finance_python)

print(table_stats)

# Close connection when done
tidy_finance_python.close()
```

7.7 Alternative Data Sources for Vietnamese Markets

Beyond traditional price and fundamental data, researchers increasingly incorporate alternative data sources to gain unique insights into market dynamics.

7.7.1 Foreign Investor Flow Data

Foreign investor flow data is particularly valuable given the significant role of foreign capital in Vietnamese equity markets. The State Securities Commission publishes daily foreign ownership statistics by security.

7.7.2 News and Sentiment Data

Media sentiment from Vietnamese financial news sources offers another research avenue. Major outlets such as CafeF, VnExpress Finance, and Vietstock publish real-time news that can be analyzed for market sentiment.

7.8 Key Takeaways

1. **Market Structure Understanding**: The Vietnamese financial market operates across three exchanges (HOSE, HNX, UPCoM) with distinct characteristics including foreign ownership limits, trading band restrictions, and a T+2 settlement cycle. Researchers must account for these institutional features in empirical analysis.
2. **Macroeconomic Data Challenges**: Unlike developed markets with standardized APIs (e.g., FRED), Vietnamese macroeconomic data requires manual collection from government sources (GSO, SBV). Researchers should plan for this additional data gathering effort and implement systematic data management practices.

3. **Database-Centric Workflow**: SQLite provides an efficient and portable database solution for managing Vietnamese financial data across research projects. The structured database approach enables reproducible research workflows, efficient queries, and easy data sharing among collaborators.

4. **Data Quality Imperative**: Data quality validation is especially important for emerging market data. Implementing systematic checks for missing values, extreme returns, duplicate entries, and cross-source validation helps ensure research reliability and reproducibility.

5. **Alternative Data Opportunities**: Foreign investor flows, corporate announcements, and media sentiment provide unique research opportunities in the Vietnamese market that can complement traditional price and fundamental analysis. These data sources can reveal insights about market dynamics not captured in standard datasets.

6. **Continuous Maintenance**: Financial databases require ongoing maintenance including incremental updates, integrity checks, and optimization. Establishing systematic update procedures ensures data currency and database performance over time.

8 DataCore Data

This chapter demonstrates how to connect to Datacore, Vietnam's premier provider of financial and economic data for research applications. We use this connection to download the most commonly used data for stock prices and firm characteristics, including historical trading data and company fundamentals. While Datacore requires a subscription, most students and researchers typically have access through their university libraries or research institutions. For those without access, Datacore provides demo datasets that allow you to run the code examples in this book with sample data.

The chapter is organized as follows. We first establish the connection to Datacore's cloud storage infrastructure. Then, we download and prepare company fundamentals data, including balance sheet items, income statement variables, and derived metrics essential for asset pricing research. Next, we retrieve and process stock price data, computing returns, market capitalizations, and excess returns. We conclude by merging these datasets and providing descriptive statistics that characterize the Vietnamese equity market.

8.1 Setting Up the Environment

We begin by loading the Python packages used throughout this chapter. The core packages include `pandas` for data manipulation, `numpy` for numerical operations, and `sqlite3` for local database management. We also import visualization libraries for creating publication-quality figures.

```
import pandas as pd
import numpy as np
import sqlite3
from datetime import datetime
from io import BytesIO

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

We establish a connection to our local SQLite database, which serves as the central repository for all processed data. This database was introduced in the previous chapter and will store the cleaned datasets for use in subsequent analyses.

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

We define the date range for our data collection. The Vietnamese stock market began operations in July 2000 with the establishment of the Ho Chi Minh City Stock Exchange (HOSE), so our sample period starts from 2000 and extends through the end of 2024.

```
start_date = "2000-01-01"
end_date = "2024-12-31"
```

8.2 Connecting to Datacore

Datacore delivers data through a cloud-based object storage system built on MinIO, an S3-compatible storage infrastructure. This architecture enables efficient, programmatic access to large datasets without the limitations of traditional database connections. To access the data, you need credentials provided by Datacore upon subscription: an endpoint URL, access key, and secret key.

The following class establishes the connection to Datacore's storage system. The credentials are stored as environment variables for security, following best practices for credential management in research computing environments.

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

class DatacoreConnection:
    """
    Connection handler for Datacore's MinIO-based storage system.

    This class manages authentication and provides methods for
    accessing financial datasets stored in Datacore's cloud infrastructure.

    Attributes
    ----------
    s3 : boto3.client
        S3-compatible client for interacting with Datacore storage
    """

    def __init__(self):
        """Initialize connection using environment variables."""
        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):
        """Verify connection by listing available buckets."""
        response = self.s3.list_buckets()
        print("Connected successfully. Available buckets:")
        for bucket in response.get("Buckets", []):
            print(f"  - {bucket['Name']}")

    def list_objects(self, bucket_name, prefix=""):
        """List objects in a bucket with optional prefix filter."""
        response = self.s3.list_objects_v2(
            Bucket=bucket_name,
            Prefix=prefix
        )
        return [obj["Key"] for obj in response.get("Contents", [])]

    def read_excel(self, bucket_name, key):
        """Read an Excel file from Datacore storage."""
        obj = self.s3.get_object(Bucket=bucket_name, Key=key)
        return pd.read_excel(BytesIO(obj["Body"].read()))

    def read_csv(self, bucket_name, key, **kwargs):
        """Read a CSV file from Datacore storage."""
        obj = self.s3.get_object(Bucket=bucket_name, Key=key)
        return pd.read_csv(BytesIO(obj["Body"].read()), **kwargs)
```

With the connection class defined, we can establish a connection and verify access to Datacore's data repositories.

```
# Initialize connection
conn = DatacoreConnection()
conn.test_connection()

# Get bucket name from environment
bucket_name = os.environ["MINIO_BUCKET"]
```

```
Connected successfully. Available buckets:
  - dsteam-data
  - rawbctc
```

8.3 Company Fundamentals Data

Firm accounting data are essential for portfolio analyses, factor construction, and valuation studies. Datacore hosts comprehensive fundamentals data for Vietnamese listed companies, including annual and quarterly financial statements prepared according to Vietnamese Accounting Standards (VAS).

8.3.1 Understanding Vietnamese Financial Statements

Before processing the data, it is important to understand the structure of Vietnamese financial reports. Vietnamese companies follow VAS, which shares similarities with International Financial Reporting Standards (IFRS) but has notable differences:

1. **Fiscal Year**: Most Vietnamese companies use a calendar fiscal year ending December 31, though some companies (particularly in retail and agriculture) use different fiscal year-ends.
2. **Reporting Frequency**: Listed companies must publish quarterly financial statements within 20 days of quarter-end and annual audited statements within 90 days of fiscal year-end.
3. **Industry-Specific Formats**: Companies in banking, insurance, and securities sectors follow specialized reporting formats that differ from the standard industrial format.
4. **Currency**: All figures are reported in Vietnamese Dong (VND). Given the large nominal values (millions to trillions of VND), we often scale figures to millions or billions for readability.

8.3.2 Downloading Fundamentals Data

Datacore organizes fundamentals data in Excel files partitioned by time period for efficient access. We download and concatenate these files to create a comprehensive dataset spanning our sample period.

```
# Define paths to fundamentals data files
fundamentals_paths = [
    "fundamental_annual_1767674486317/fundamental_annual_1.xlsx",
    "fundamental_annual_1767674486317/fundamental_annual_2.xlsx",
    "fundamental_annual_1767674486317/fundamental_annual_3.xlsx",
]

# Download and combine all files
fundamentals_list = []
for path in fundamentals_paths:
    df_temp = conn.read_excel(bucket_name, path)
    fundamentals_list.append(df_temp)
    print(f"Downloaded: {path} ({len(df_temp):,} rows)")

df_fundamentals_raw = pd.concat(fundamentals_list, ignore_index=True)
print(f"\nTotal observations: {len(df_fundamentals_raw):,}")
```

```
Downloaded: fundamental_annual_1767674486317/fundamental_annual_1.xlsx (10,000 rows)

Downloaded: fundamental_annual_1767674486317/fundamental_annual_2.xlsx (10,000 rows)

Downloaded: fundamental_annual_1767674486317/fundamental_annual_3.xlsx (2,821 rows)

Total observations: 22,821
```

8.3.3 Cleaning and Standardizing Fundamentals

The raw fundamentals data requires several cleaning steps to ensure consistency and usability. We standardize variable names, handle missing values, and create derived variables commonly used in asset pricing research.

```python
def clean_fundamentals(df):
    """
    Clean and standardize company fundamentals data.

    Parameters
    ----------
    df : pd.DataFrame
        Raw fundamentals data from Datacore

    Returns
    -------
    pd.DataFrame
        Cleaned fundamentals with standardized column names
    """
    df = df.copy()

    # Standardize identifiers
    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 identifiers
    df = df.dropna(subset=["symbol", "year"])

    # Define columns that should be numeric
    numeric_columns = [
        "total_asset", "total_equity", "total_liabilities",
        "total_current_asset", "total_current_liabilities",
        "is_net_revenue", "is_cogs", "is_manage_expense",
        "is_interest_expense", "is_eat", "is_net_business_profit",
        "na_tax_deferred", "nl_tax_deferred", "e_preferred_stock",
        "capex", "total_cfo", "ca_cce", "ca_total_inventory",
        "ca_acc_receiv", "cfo_interest_expense", "basic_eps",
        "is_shareholders_eat", "cl_loan", "cl_finlease",
        "cl_due_long_debt", "nl_loan", "nl_finlease",
        "is_cos_of_sales", "e_equity"
    ]

    for col in numeric_columns:
        if col in df.columns:
            df[col] = pd.to_numeric(df[col], errors="coerce")

    # Handle duplicates: keep row with most non-missing values
```

```
    df["_completeness"] = df.notna().sum(axis=1)
    df = (df
        .sort_values(["symbol", "year", "_completeness"])
        .drop_duplicates(subset=["symbol", "year"], keep="last")
        .drop(columns="_completeness")
        .reset_index(drop=True)
    )

    return df

df_fundamentals = clean_fundamentals(df_fundamentals_raw)
print(f"After cleaning: {len(df_fundamentals):,} firm-year observations")
print(f"Unique firms: {df_fundamentals['symbol'].nunique():,}")
```

```
After cleaning: 21,232 firm-year observations
Unique firms: 1,554
```

8.3.4 Creating Standardized Variables

To facilitate comparison with international studies and ensure compatibility with standard asset pricing methodologies, we create variables following conventions established in the academic literature. We map Vietnamese financial statement items to their Compustat equivalents where possible.

```
def create_standard_variables(df):
    """
    Create standardized financial variables for asset pricing research.

    This function maps Vietnamese financial statement items to standard
    variable names used in the academic finance literature, following
    conventions from Fama and French (1992, 1993, 2015).

    Parameters
    ----------
    df : pd.DataFrame
        Cleaned fundamentals data

    Returns
    -------
    pd.DataFrame
        Fundamentals with standardized variables added
```

```
    """
    df = df.copy()

    # Fiscal date (assume December year-end)
    df["datadate"] = pd.to_datetime(df["year"].astype(str) + "-12-31")

    # === Balance Sheet Items ===
    df["at"] = df["total_asset"]                    # Total assets
    df["lt"] = df["total_liabilities"]              # Total liabilities
    df["seq"] = df["total_equity"]                  # Stockholders' equity
    df["act"] = df["total_current_asset"]           # Current assets
    df["lct"] = df["total_current_liabilities"]     # Current liabilities

    # Common equity (fallback to total equity if not available)
    df["ceq"] = df.get("e_equity", df["seq"])

    # === Deferred Taxes ===
    df["txditc"] = df.get("na_tax_deferred", 0).fillna(0)  # Deferred tax assets
    df["txdb"] = df.get("nl_tax_deferred", 0).fillna(0)    # Deferred tax liab.
    df["itcb"] = 0  # Investment tax credit (rare in Vietnam)

    # === Preferred Stock ===
    pref = df.get("e_preferred_stock", 0)
    if isinstance(pref, pd.Series):
        pref = pref.fillna(0)
    df["pstk"] = pref
    df["pstkrv"] = pref  # Redemption value
    df["pstkl"] = pref   # Liquidating value

    # === Income Statement Items ===
    df["sale"] = df["is_net_revenue"]                         # Net sales/revenue
    df["cogs"] = df.get("is_cogs", 0).fillna(0)               # Cost of goods sold
    df["xsga"] = df.get("is_manage_expense", 0).fillna(0)     # SG&A expenses
    df["xint"] = df.get("is_interest_expense", 0).fillna(0)   # Interest expense
    df["ni"] = df.get("is_eat", np.nan)                       # Net income
    df["oibdp"] = df.get("is_net_business_profit", np.nan)    # Operating income

    # === Cash Flow Items ===
    df["oancf"] = df.get("total_cfo", np.nan)  # Operating cash flow
    df["capx"] = df.get("capex", np.nan)       # Capital expenditures

    return df
```

```
df_fundamentals = create_standard_variables(df_fundamentals)
```

8.3.5 Computing Book Equity and Profitability

Book equity is a crucial variable for value investing strategies and the construction of HML (High Minus Low) factor portfolios. We follow the definition from Kenneth French's data library, which accounts for deferred taxes and preferred stock.

```
def compute_book_equity(df):
    """
    Compute book equity following Fama-French conventions.

    Book equity = Stockholders' equity
                  + Deferred taxes and investment tax credit
                  - Preferred stock

    Negative or zero book equity is set to missing, as book-to-market
    ratios are undefined for such firms.

    Parameters
    ----------
    df : pd.DataFrame
        Fundamentals with standardized variables

    Returns
    -------
    pd.DataFrame
        Fundamentals with book equity (be) added
    """
    df = df.copy()

    # Primary measure: stockholders' equity
    # Fallback 1: common equity + preferred stock
    # Fallback 2: total assets - total liabilities
    seq_measure = (df["seq"]
        .combine_first(df["ceq"] + df["pstk"])
        .combine_first(df["at"] - df["lt"])
    )

    # Add deferred taxes
    deferred_taxes = (df["txditc"]
```

```
        .combine_first(df["txdb"] + df["itcb"])
        .fillna(0)
    )

    # Subtract preferred stock (use redemption value as primary)
    preferred = (df["pstkrv"]
        .combine_first(df["pstkl"])
        .combine_first(df["pstk"])
        .fillna(0)
    )

    # Book equity calculation
    df["be"] = seq_measure + deferred_taxes - preferred

    # Set non-positive book equity to missing
    df["be"] = df["be"].where(df["be"] > 0, np.nan)

    return df

df_fundamentals = compute_book_equity(df_fundamentals)

# Summary statistics for book equity
print("Book Equity Summary Statistics (in million VND):")
print(df_fundamentals["be"].describe().round(2))
```

```
Book Equity Summary Statistics (in million VND):
count    2.023500e+04
mean     1.031884e+12
std      4.705269e+12
min      4.404402e+07
25%      7.267610e+10
50%      1.803885e+11
75%      5.304653e+11
max      1.836314e+14
Name: be, dtype: float64
```

Operating profitability, introduced by Fama and French (2015), measures a firm's profits relative to its book equity. Firms with higher operating profitability tend to have higher expected returns.

```python
def compute_profitability(df):
    """
    Compute operating profitability following Fama-French (2015).

    Operating profitability = (Revenue - COGS - SG&A - Interest) / Book Equity

    Parameters
    ----------
    df : pd.DataFrame
        Fundamentals with book equity computed

    Returns
    -------
    pd.DataFrame
        Fundamentals with operating profitability (op) added
    """
    df = df.copy()

    # Operating profit before taxes
    operating_profit = (
        df["sale"]
        - df["cogs"].fillna(0)
        - df["xsga"].fillna(0)
        - df["xint"].fillna(0)
    )

    # Scale by book equity
    df["op"] = operating_profit / df["be"]

    # Winsorize extreme values (outside 1st and 99th percentiles)
    lower = df["op"].quantile(0.01)
    upper = df["op"].quantile(0.99)
    df["op"] = df["op"].clip(lower=lower, upper=upper)

    return df

df_fundamentals = compute_profitability(df_fundamentals)
```

8.3.6 Computing Investment

Investment, measured as asset growth, captures firms' investment behavior. Fama and French (2015) document that firms with high asset growth (aggressive investment) tend to have lower future returns.

```
def compute_investment(df):
    """
    Compute investment (asset growth) following Fama-French (2015).

    Investment = (Total Assets_t / Total Assets_{t-1}) - 1

    Parameters
    ----------
    df : pd.DataFrame
        Fundamentals data

    Returns
    -------
    pd.DataFrame
        Fundamentals with investment (inv) added
    """
    df = df.copy()

    # Create lagged assets
    df_lag = (df[["symbol", "year", "at"]]
        .assign(year=lambda x: x["year"] + 1)
        .rename(columns={"at": "at_lag"})
    )

    # Merge lagged values
    df = df.merge(df_lag, on=["symbol", "year"], how="left")

    # Compute investment (asset growth)
    df["inv"] = df["at"] / df["at_lag"] - 1

    # Set to missing if lagged assets non-positive
    df["inv"] = df["inv"].where(df["at_lag"] > 0, np.nan)

    return df

df_fundamentals = compute_investment(df_fundamentals)
```

8.3.7 Computing Total Debt

In Vietnamese financial statements, total liabilities include non-interest-bearing items such as accounts payable and tax payables. For leverage analysis, we compute total interest-bearing debt by aggregating loan and lease obligations.

```
def compute_total_debt(df):
    """
    Compute total interest-bearing debt.

    Total Debt = Short-term loans + Finance leases (current)
                 + Current portion of long-term debt
                 + Long-term loans + Finance leases (non-current)

    Parameters
    ----------
    df : pd.DataFrame
        Fundamentals data

    Returns
    -------
    pd.DataFrame
        Fundamentals with total_debt added
    """
    df = df.copy()

    df["total_debt"] = (
        df.get("cl_loan", 0).fillna(0) +           # Short-term bank loans
        df.get("cl_finlease", 0).fillna(0) +       # Current finance leases
        df.get("cl_due_long_debt", 0).fillna(0) +  # Current portion LT debt
        df.get("nl_loan", 0).fillna(0) +           # Long-term bank loans
        df.get("nl_finlease", 0).fillna(0)         # Non-current finance leases
    )

    return df

df_fundamentals = compute_total_debt(df_fundamentals)
```

8.3.8 Applying Filters and Final Preparation

We apply standard filters to ensure data quality: requiring positive assets, non-negative sales, and presence of core variables needed for portfolio construction.

```
# Keep only observations with required variables
required_vars = ["at", "lt", "seq", "sale"]
comp_vn = df_fundamentals.dropna(subset=required_vars)

# Apply quality filters
comp_vn = comp_vn.query("at > 0")      # Positive assets
comp_vn = comp_vn.query("sale >= 0")   # Non-negative sales

# Keep last observation per firm-year (in case of restatements)
comp_vn = (comp_vn
    .sort_values("datadate")
    .groupby(["symbol", "year"])
    .tail(1)
    .reset_index(drop=True)
)

# Diagnostic summary
print(f"Final sample: {len(comp_vn):,} firm-year observations")
print(f"Unique firms: {comp_vn['symbol'].nunique():,}")
print(f"Sample period: {comp_vn['year'].min()} - {comp_vn['year'].max()}")
```

```
Final sample: 20,091 firm-year observations
Unique firms: 1,502
Sample period: 1998 - 2023
```

8.3.9 Storing Fundamentals Data

We store the prepared fundamentals data in our local SQLite database for use in subsequent chapters.

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

print("Company fundamentals saved to database.")
```

```
Company fundamentals saved to database.
```

8.4 Stock Price Data

Stock price data forms the foundation of return-based analyses in empirical finance. Datacore provides comprehensive historical price data for all securities traded on HOSE, HNX, and UPCoM, including adjusted prices that account for corporate actions.

8.4.1 Downloading Price Data

We download the historical price data from Datacore's storage system. The data includes daily observations with open, high, low, close prices, trading volume, and adjustment factors.

```
# Download historical price data
prices_raw = conn.read_csv(
    bucket_name,
    "historycal_price/dataset_historical_price.csv",
    low_memory=False
)

print(f"Downloaded {len(prices_raw):,} daily price observations")
print(f"Date range: {prices_raw['date'].min()} to {prices_raw['date'].max()}")
```

```
Downloaded 4,307,791 daily price observations
Date range: 2010-01-04 to 2025-05-12
```

8.4.2 Processing Price Data

We clean the price data and compute adjusted prices that account for stock splits, stock dividends, and other corporate actions.

```
def process_price_data(df):
    """
    Process raw price data from Datacore.
    """
    df = df.copy()

    # Parse dates
    df["date"] = pd.to_datetime(df["date"])

    # Standardize column names
    df = df.rename(columns={
```

```
        "open_price": "open",
        "high_price": "high",
        "low_price": "low",
        "close_price": "close",
        "vol_total": "volume"
    })

    # Compute adjusted close price
    df["adjusted_close"] = df["close"] * df["adj_ratio"]

    # Standardize symbol
    df["symbol"] = df["symbol"].astype(str).str.upper().str.strip()

    # Sort for return calculation
    df = df.sort_values(["symbol", "date"])

    # Add year and month
    df["year"] = df["date"].dt.year
    df["month"] = df["date"].dt.month

    return df

prices = process_price_data(prices_raw)
```

8.4.3 Computing Shares Outstanding and Market Capitalization

Market capitalization is computed as the product of price and shares outstanding. Since Datacore provides earnings per share and net income, we can infer shares outstanding from these variables.

```
def compute_shares_outstanding(fundamentals_df):
    """
    Compute shares outstanding from fundamentals.
    """
    shares = fundamentals_df.copy()
    shares["shrout"] = shares["is_shareholders_eat"] / shares["basic_eps"]
    shares = shares[["symbol", "year", "shrout"]].dropna()

    return shares

shares_outstanding = compute_shares_outstanding(df_fundamentals)
```

```
def add_market_cap(df, shares_df):
    """
    Add market capitalization to price data.
    """
    df = df.merge(shares_df, on=["symbol", "year"], how="left")

    # Compute market cap (in million VND)
    df["mktcap"] = (df["close"] * df["shrout"]) / 1_000_000

    # Set zero or negative market cap to missing
    df["mktcap"] = df["mktcap"].where(df["mktcap"] > 0, np.nan)

    return df

prices = add_market_cap(prices, shares_outstanding)
```

8.4.4 Computing Returns and Excess Returns

We compute returns using adjusted closing prices to ensure returns correctly reflect total shareholder returns including dividends and corporate actions.

8.4.4.1 Creating Daily Dataset

1. Sequential version

```
def create_daily_dataset(df, annual_rf=0.04):
    """
    Create daily price dataset with returns and excess returns.
    """
    df = df.copy()

    # Sort by symbol and date (critical for correct return calculation)
    df = df.sort_values(["symbol", "date"]).reset_index(drop=True)

    # Remove duplicate dates within each symbol (keep last observation)
    df = df.drop_duplicates(subset=["symbol", "date"], keep="last")

    # Compute daily returns
    df["ret"] = df.groupby("symbol")["adjusted_close"].pct_change()
```

```
    # Cap extreme negative returns
    df["ret"] = df["ret"].clip(lower=-0.99)

    # Daily risk-free rate (assuming 252 trading days)
    df["risk_free"] = annual_rf / 252

    # Excess returns
    df["ret_excess"] = df["ret"] - df["risk_free"]
    df["ret_excess"] = df["ret_excess"].clip(lower=-1.0)

    # Lagged market cap
    df["mktcap_lag"] = df.groupby("symbol")["mktcap"].shift(1)

    return df

prices_daily = create_daily_dataset(prices)
```

2. Parallel version

```
from joblib import Parallel, delayed
import os

def process_daily_symbol(symbol_df, annual_rf=0.04):
    """
    Process a single symbol's daily data.
    """
    df = symbol_df.copy()

    # Sort by date (critical for correct return calculation)
    df = df.sort_values("date").reset_index(drop=True)

    # Remove duplicate dates (keep last observation if duplicates exist)
    df = df.drop_duplicates(subset=["date"], keep="last")

    # Compute daily returns
    df["ret"] = df["adjusted_close"].pct_change()

    # Replace infinite values with NaN
    df["ret"] = df["ret"].replace([np.inf, -np.inf], np.nan)

    # Cap extreme negative returns
    df["ret"] = df["ret"].clip(lower=-0.99)
```

```
    # Daily risk-free rate
    df["risk_free"] = annual_rf / 252

    # Excess returns
    df["ret_excess"] = df["ret"] - df["risk_free"]
    df["ret_excess"] = df["ret_excess"].clip(lower=-1.0)

    # Lagged market cap
    df["mktcap_lag"] = df["mktcap"].shift(1)

    return df

def create_daily_dataset_parallel(df, annual_rf=0.04):
    """
    Create daily price dataset using parallel processing.
    """
    # Ensure data is sorted before splitting
    df = df.sort_values(["symbol", "date"])

    # Split by symbol
    symbol_groups = [group for _, group in df.groupby("symbol")]

    n_jobs = max(1, os.cpu_count() - 1)
    print(f"Processing {len(symbol_groups):,} symbols using {n_jobs} cores...")

    results = Parallel(n_jobs=n_jobs, verbose=1)(
        delayed(process_daily_symbol)(group, annual_rf)
        for group in symbol_groups
    )

    return pd.concat(results, ignore_index=True)

prices_daily = create_daily_dataset_parallel(prices)

# Quick validation
print("\nValidation checks:")
print(f"Any duplicate (symbol, date): {prices_daily.duplicated(subset=['symbol', 'date']).sum
print(f"Sample of non-zero returns:")
print(prices_daily[prices_daily["ret"] != 0][["symbol", "date", "adjusted_close", "ret"]].hea


prices_daily.query("symbol == 'FPT'")[["symbol", "date", "adjusted_close", "ret"]].head(3)
```

```
Processing 1,837 symbols using 23 cores...


Validation checks:
Any duplicate (symbol, date): 0
Sample of non-zero returns:
   symbol       date  adjusted_close       ret
0     A32 2018-10-23       44.574418       NaN
27    A32 2018-11-29       55.072640  0.235521
30    A32 2018-12-04       48.188560 -0.125000
43    A32 2018-12-21       51.974804  0.078571
49    A32 2019-01-02       55.072640  0.059603
53    A32 2019-01-08       50.030370 -0.091557
74    A32 2019-02-13       44.289180 -0.114754
75    A32 2019-02-14       41.008500 -0.074074
78    A32 2019-02-19       36.087480 -0.120000
91    A32 2019-03-08       41.336568  0.145455
```

	symbol	date	adjusted_close	ret
1146076	FPT	2010-01-04	1170.9885	NaN
1146077	FPT	2010-01-05	1170.9885	0.000000
1146078	FPT	2010-01-06	1149.6978	-0.018182

```python
# Select columns
daily_columns = [
    "symbol", "date", "year", "month",
    "open", "high", "low", "close", "volume",
    "adjusted_close", "shrout", "mktcap", "mktcap_lag",
    "ret", "risk_free", "ret_excess"
]
prices_daily = prices_daily[daily_columns]

# Remove observations with missing essential variables
prices_daily = prices_daily.dropna(subset=["ret_excess", "mktcap", "mktcap_lag"])

print("Daily Return Summary Statistics:")
print(prices_daily["ret"].describe().round(4))
print(f"\nFinal daily sample: {len(prices_daily):,} observations")
```

```
Daily Return Summary Statistics:
count    3.462157e+06
mean     3.000000e-04
std      4.480000e-02
min     -9.900000e-01
25%     -4.900000e-03
50%      0.000000e+00
75%      4.000000e-03
max      3.250000e+01
Name: ret, dtype: float64

Final daily sample: 3,462,157 observations
```

8.4.4.2 Creating Monthly Dataset

For monthly returns, we compute returns directly from month-end adjusted prices rather than compounding daily returns. This avoids compounding errors from missing days and is the standard approach in empirical finance.

1. Sequential version

```python
def create_monthly_dataset(df, annual_rf=0.04):
    """
    Create monthly price dataset with returns computed from
    month-end to month-end adjusted prices.
    """
    df = df.copy()

    # Sort by symbol and date (critical for correct return calculation)
    df = df.sort_values(["symbol", "date"]).reset_index(drop=True)

    # Remove duplicate dates within each symbol (keep last observation)
    df = df.drop_duplicates(subset=["symbol", "date"], keep="last")

    # Get month-end observations
    monthly = (df
        .groupby("symbol")
        .resample("ME", on="date")
        .agg({
            "open": "first",           # First day open
            "high": "max",             # Monthly high
            "low": "min",              # Monthly low
```

```
            "close": "last",           # Last day close
            "volume": "sum",           # Total monthly volume
            "adjusted_close": "last",  # Month-end adjusted price
            "shrout": "last",          # Month-end shares outstanding
            "mktcap": "last",          # Month-end market cap
            "year": "last",
            "month": "last"
        })
        .reset_index()
    )

    # Remove duplicate (symbol, date) after resampling (safety check)
    monthly = monthly.drop_duplicates(subset=["symbol", "date"], keep="last")

    # Sort again after resampling
    monthly = monthly.sort_values(["symbol", "date"]).reset_index(drop=True)

    # Compute monthly returns from month-end to month-end adjusted prices
    monthly["ret"] = monthly.groupby("symbol")["adjusted_close"].pct_change()

    # Cap extreme returns
    monthly["ret"] = monthly["ret"].clip(lower=-0.99)

    # Monthly risk-free rate
    monthly["risk_free"] = annual_rf / 12

    # Excess returns
    monthly["ret_excess"] = monthly["ret"] - monthly["risk_free"]
    monthly["ret_excess"] = monthly["ret_excess"].clip(lower=-1.0)

    # Lagged market cap for portfolio weighting
    monthly["mktcap_lag"] = monthly.groupby("symbol")["mktcap"].shift(1)

    return monthly

prices_monthly = create_monthly_dataset(prices)
```

2. Parallel version

```
from joblib import Parallel, delayed
import os
```

```python
def process_monthly_symbol(symbol_df, annual_rf=0.04):
    """
    Process a single symbol's data to monthly frequency.
    """
    df = symbol_df.copy()

    # Sort by date (critical for correct return calculation)
    df = df.sort_values("date").reset_index(drop=True)

    # Remove duplicate dates (keep last observation if duplicates exist)
    df = df.drop_duplicates(subset=["date"], keep="last")

    # Set date as index for resampling
    df = df.set_index("date")

    # Resample to monthly
    monthly = df.resample("ME").agg({
        "symbol": "last",
        "open": "first",
        "high": "max",
        "low": "min",
        "close": "last",
        "volume": "sum",
        "adjusted_close": "last",
        "shrout": "last",
        "mktcap": "last",
        "year": "last",
        "month": "last"
    }).reset_index()

    # Remove rows where symbol is NaN (months with no trading)
    monthly = monthly.dropna(subset=["symbol"])

    # Sort by date
    monthly = monthly.sort_values("date").reset_index(drop=True)

    # Compute monthly returns
    monthly["ret"] = monthly["adjusted_close"].pct_change()

    # Replace infinite values with NaN
    monthly["ret"] = monthly["ret"].replace([np.inf, -np.inf], np.nan)
```

```
    # Cap extreme returns
    monthly["ret"] = monthly["ret"].clip(lower=-0.99)

    # Monthly risk-free rate
    monthly["risk_free"] = annual_rf / 12

    # Excess returns
    monthly["ret_excess"] = monthly["ret"] - monthly["risk_free"]
    monthly["ret_excess"] = monthly["ret_excess"].clip(lower=-1.0)

    # Lagged market cap
    monthly["mktcap_lag"] = monthly["mktcap"].shift(1)

    return monthly

def create_monthly_dataset_parallel(df, annual_rf=0.04):
    """
    Create monthly price dataset using parallel processing.
    """
    # Ensure data is sorted before splitting
    df = df.sort_values(["symbol", "date"])

    # Split by symbol
    symbol_groups = [group for _, group in df.groupby("symbol")]

    n_jobs = max(1, os.cpu_count() - 1)
    print(f"Processing {len(symbol_groups):,} symbols using {n_jobs} cores...")

    results = Parallel(n_jobs=n_jobs, verbose=1)(
        delayed(process_monthly_symbol)(group, annual_rf)
        for group in symbol_groups
    )

    return pd.concat(results, ignore_index=True)

prices_monthly = create_monthly_dataset_parallel(prices)

# Validation checks
print("\nValidation checks:")
print(f"Any duplicate (symbol, date): {prices_monthly.duplicated(subset=['symbol', 'date']).s
print(f"\nSample of non-zero returns:")
print(prices_monthly[prices_monthly["ret"] != 0][["symbol", "date", "adjusted_close", "ret"]]
```

```
prices_monthly.query("symbol == 'FPT'")[["symbol", "date", "adjusted_close", "ret"]].head(3)
```

```
Processing 1,837 symbols using 23 cores...


Validation checks:
Any duplicate (symbol, date): 0

Sample of non-zero returns:
   symbol       date  adjusted_close       ret
0     A32 2018-10-31       44.574418       NaN
1     A32 2018-11-30       55.072640  0.235521
2     A32 2018-12-31       51.974804 -0.056250
3     A32 2019-01-31       50.030370 -0.037411
4     A32 2019-02-28       36.087480 -0.278689
5     A32 2019-03-31       41.828670  0.159091
7     A32 2019-05-31       43.304976  0.035294
8     A32 2019-06-30       35.929125 -0.170323
9     A32 2019-07-31       37.525975  0.044444
10    A32 2019-08-31       38.324400  0.021277
```

	symbol	date	adjusted_close	ret
55963	FPT	2010-01-31	1092.9226	NaN
55964	FPT	2010-02-28	1107.1164	0.012987
55965	FPT	2010-03-31	1185.1823	0.070513

```
# Select columns (same structure as daily)
monthly_columns = [
    "symbol", "date", "year", "month",
    "open", "high", "low", "close", "volume",
    "adjusted_close", "shrout", "mktcap", "mktcap_lag",
    "ret", "risk_free", "ret_excess"
]
prices_monthly = prices_monthly[monthly_columns]

# Remove observations with missing essential variables
prices_monthly = prices_monthly.dropna(subset=["ret_excess", "mktcap", "mktcap_lag"])

print("Monthly Return Summary Statistics:")
```

```
print(prices_monthly["ret"].describe().round(4))
print(f"\nFinal monthly sample: {len(prices_monthly):,} observations")
```

```
Monthly Return Summary Statistics:
count    165499.0000
mean          0.0042
std           0.1862
min          -0.9900
25%          -0.0703
50%           0.0000
75%           0.0553
max          12.7500
Name: ret, dtype: float64

Final monthly sample: 165,499 observations
```

8.4.5 Storing Price Data

```
prices_daily.to_sql(
    name="prices_daily",
    con=tidy_finance,
    if_exists="replace",
    index=False
)
print("Daily price data saved to database.")

prices_monthly.to_sql(
    name="prices_monthly",
    con=tidy_finance,
    if_exists="replace",
    index=False
)
print("Monthly price data saved to database.")
```

8.5 Descriptive Statistics

Before proceeding to asset pricing analyses, we examine the characteristics of our sample to understand the Vietnamese equity market's evolution and composition.

8.5.1 Market Evolution Over Time

We first examine how the number of listed securities has grown over time.

```
securities_over_time = (prices_monthly
    .groupby("date")
    .agg(
        n_securities=("symbol", "nunique"),
        total_mktcap=("mktcap", "sum")
    )
    .reset_index()
)
```

```
securities_figure = (
    ggplot(securities_over_time, aes(x="date", y="n_securities"))
    + geom_line(color="steelblue", size=1)
    + labs(
        x="",
        y="Number of Securities",
        title="Growth of Vietnamese Stock Market"
    )
    + scale_x_datetime(date_breaks="2 years", date_labels="%Y")
    + scale_y_continuous(labels=comma_format())
    + theme_minimal()
)
securities_figure.show()
```

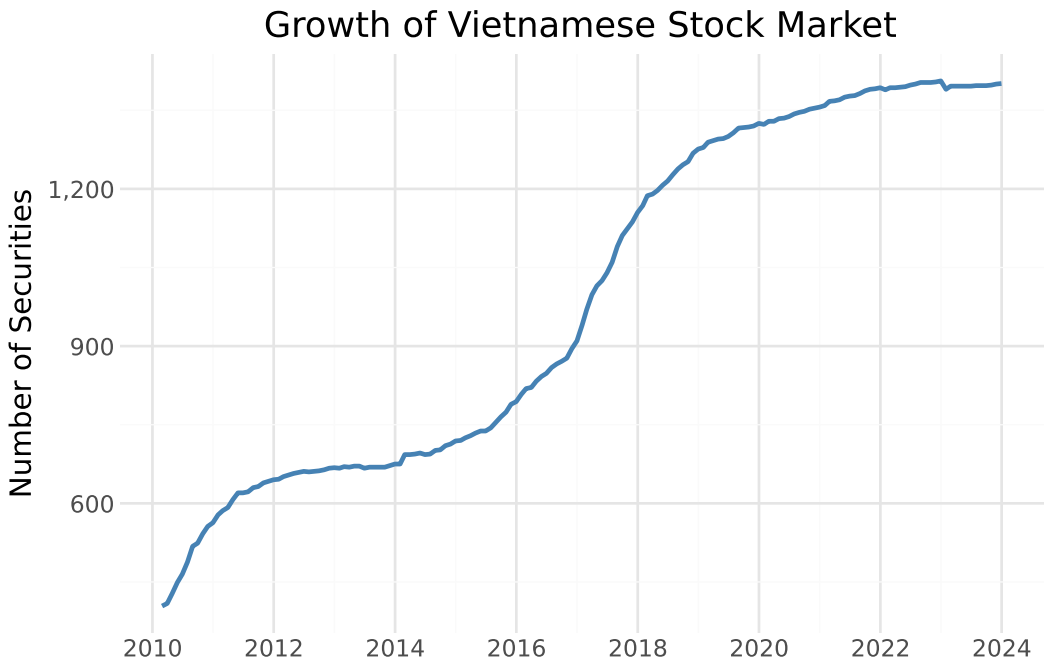


Figure 8.1: The figure shows the monthly number of securities in the Vietnamese stock market sample.

8.5.2 Market Capitalization Evolution

The aggregate market capitalization reflects the overall size and development of the Vietnamese equity market.

```
mktcap_figure = (
    ggplot(securities_over_time, aes(x="date", y="total_mktcap / 1000"))
    + geom_line(color="darkgreen", size=1)
    + labs(
        x="",
        y="Market Cap (Trillion VND)",
        title="Total Market Capitalization of Vietnamese Equities"
    )
    + scale_x_datetime(date_breaks="2 years", date_labels="%Y")
    + scale_y_continuous(labels=comma_format())
    + theme_minimal()
)
mktcap_figure.show()
```

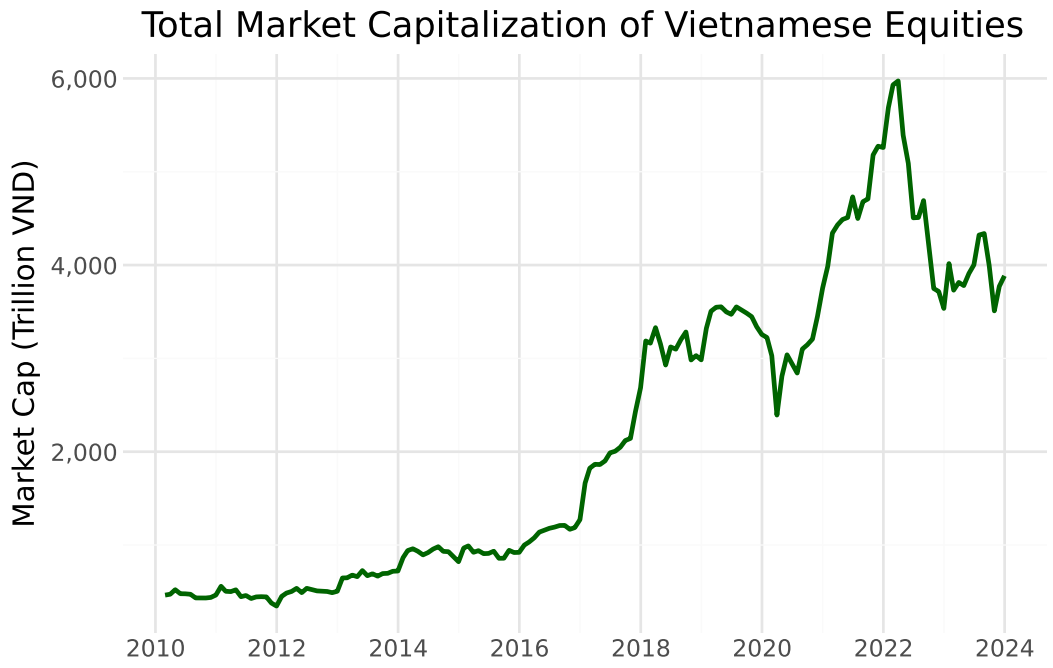


Figure 8.2: The figure shows the total market capitalization of Vietnamese listed companies over time.

8.5.3 Return Distribution

Understanding the distribution of monthly returns helps identify potential data quality issues and characterize market risk.

```
return_distribution = (
    ggplot(prices_monthly, aes(x="ret_excess"))
    + geom_histogram(
        binwidth=0.02,
        fill="steelblue",
        color="white",
        alpha=0.7
    )
    + labs(
        x="Monthly Excess Return",
        y="Frequency",
        title="Distribution of Monthly Excess Returns"
    )
    + scale_x_continuous(limits=(-0.5, 0.5))
```

```
    + theme_minimal()
)
return_distribution.show()
```

Figure 8.3: Distribution of monthly excess returns for Vietnamese stocks.

8.5.4 Coverage of Book Equity

Book equity is essential for constructing value portfolios. We examine what fraction of our sample has book equity data available over time.

```
# Merge prices with fundamentals
coverage_data = (prices_monthly
    .assign(year=lambda x: x["date"].dt.year)
    .groupby(["symbol", "year"])
    .tail(1)
    .merge(comp_vn[["symbol", "year", "be"]],
           on=["symbol", "year"],
           how="left")
)
```

```
# Compute coverage by year
be_coverage = (coverage_data
    .groupby("year")
    .apply(lambda x: pd.Series({
        "share_with_be": x["be"].notna().mean()
    }))
    .reset_index()
)

coverage_figure = (
    ggplot(be_coverage, aes(x="year", y="share_with_be"))
    + geom_line(color="darkorange", size=1)
    + geom_point(color="darkorange", size=2)
    + labs(
        x="Year",
        y="Share with Book Equity",
        title="Coverage of Book Equity Data"
    )
    + scale_y_continuous(labels=percent_format(), limits=(0, 1))
    + theme_minimal()
)
coverage_figure.show()
```

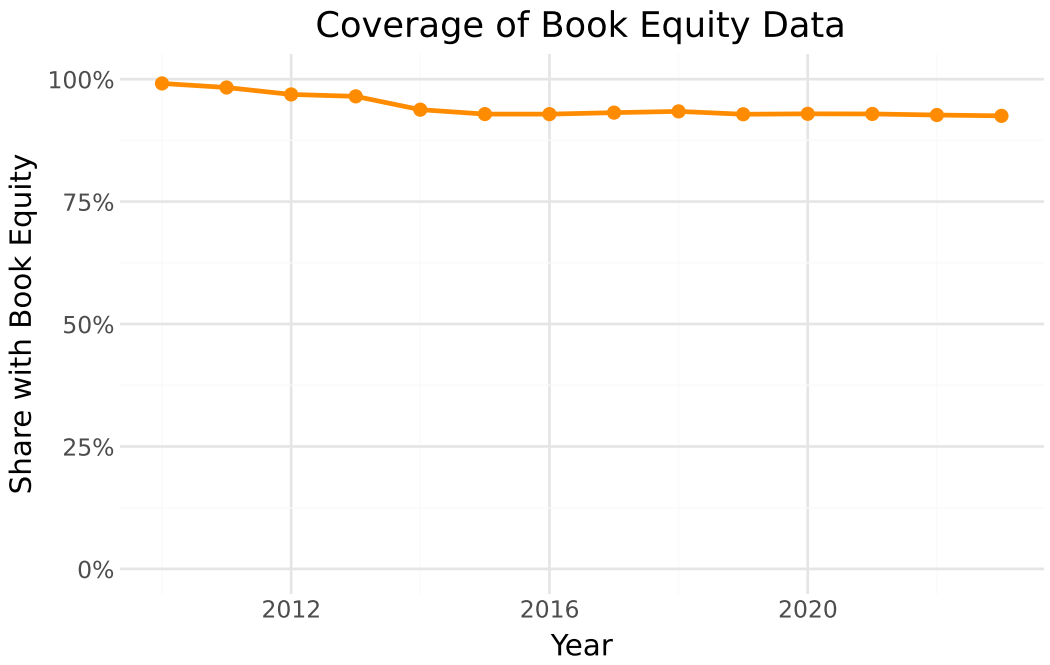


Figure 8.4: Share of securities with available book equity data by year.

8.6 Merging Stock and Fundamental Data

The final step links price data with fundamental data using the stock symbol as the common identifier. This merged dataset forms the basis for constructing portfolios sorted on firm characteristics.

```
# Example: Create merged dataset for end-of-June each year
merged_data = (prices_monthly
    .query("month == 6")
    .merge(
        comp_vn[["symbol", "year", "be", "op", "inv", "at"]],
        on=["symbol", "year"],
        how="left",
        suffixes=("", "_fundamental")
    )
)

# Convert BE from VND to BILLION VND
merged_data["be"] = merged_data["be"] / 1e9
```

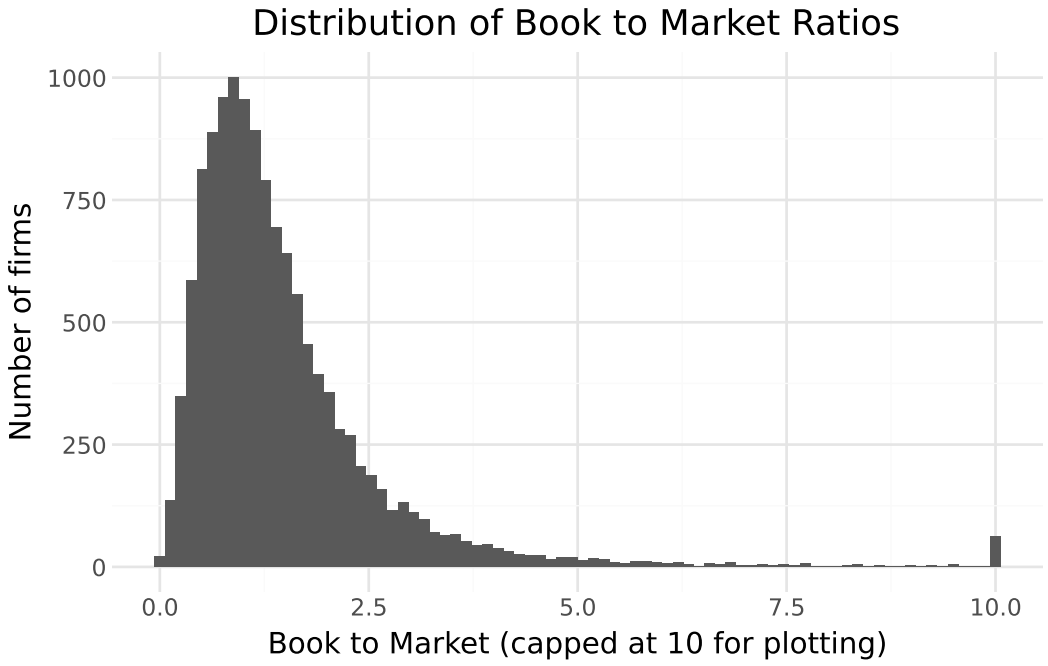
```
# Compute book-to-market ratio
merged_data["bm"] = merged_data["be"] / merged_data["mktcap"]

merged_data.loc[
    (merged_data["bm"] <= 0) |
    (merged_data["bm"] > 20),
    "bm"
] = pd.NA


merged_data["bm"].describe(percentiles=[.01, .1, .5, .9, .99])

print(f"Merged observations: {len(merged_data):,}")
print(f"With book-to-market: {merged_data['bm'].notna().sum():,}")
merged_data.head(3)
merged_data.describe()
merged_data
```

```
Merged observations: 13,756
With book-to-market: 12,859
```

	symbol	date	year	month	open	high	low	close	volume	adjusted_close	...	m
0	A32	2019-06-30	2019.0	6.0	26.4	26.4	21.0	22.5	3700	35.929125	...	15
1	A32	2020-06-30	2020.0	6.0	25.0	26.3	24.5	26.3	7500	38.811173	...	17
2	A32	2021-06-30	2021.0	6.0	30.2	37.0	29.5	32.0	78400	45.363520	...	21
3	A32	2022-06-30	2022.0	6.0	30.9	35.5	25.0	35.3	15200	47.503210	...	24
4	A32	2023-06-30	2023.0	6.0	30.1	33.5	29.2	29.4	2400	35.064204	...	19
...	...	...	...	...	...	...	...	...	...	...	...	...
13751	YTC	2019-06-30	2019.0	6.0	70.0	79.9	70.0	79.9	38900	171.451817	...	24
13752	YTC	2020-06-30	2020.0	6.0	88.5	88.5	77.0	87.0	150640	180.966960	...	26
13753	YTC	2021-06-30	2021.0	6.0	76.0	115.5	61.0	61.0	34100	126.884880	...	18
13754	YTC	2022-06-30	2022.0	6.0	68.0	68.0	65.0	65.5	200	136.245240	...	20
13755	YTC	2023-06-30	2023.0	6.0	59.0	59.0	59.0	59.0	49545	122.724720	...	18

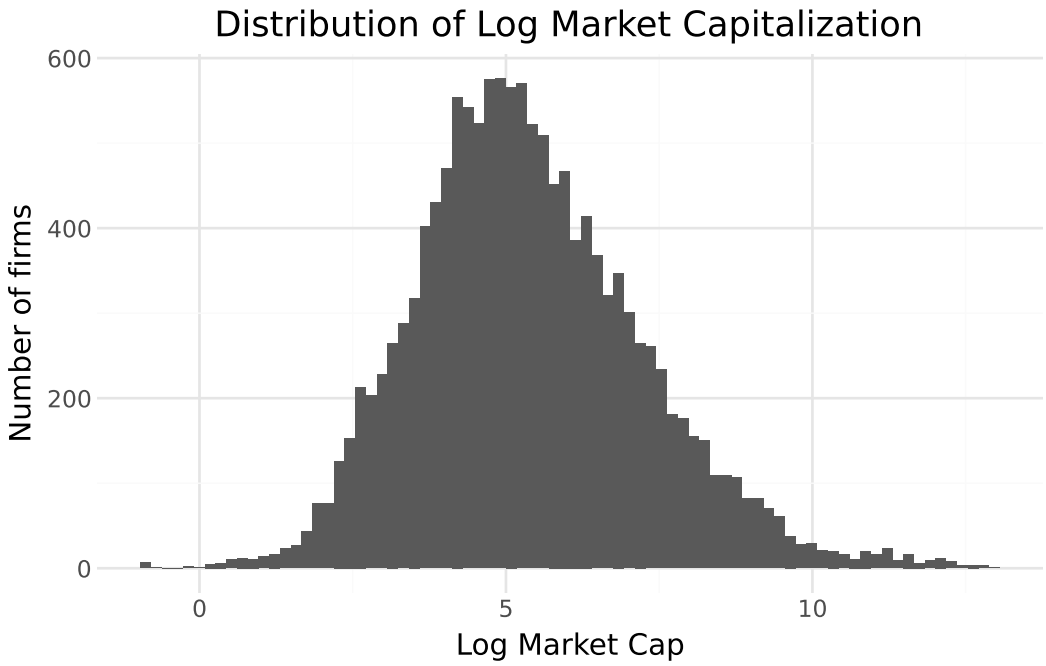
```
from plotnine import *
import numpy as np

bm_plot_data = (
    merged_data[["bm"]]
      .dropna()
```

```
      .assign(bm_plot=lambda x: x["bm"].clip(upper=10))
)

(
    ggplot(bm_plot_data, aes(x="bm_plot")) +
    geom_histogram(bins=80) +
    labs(
        title="Distribution of Book to Market Ratios",
        x="Book to Market (capped at 10 for plotting)",
        y="Number of firms"
    ) +
    theme_minimal()
)
```



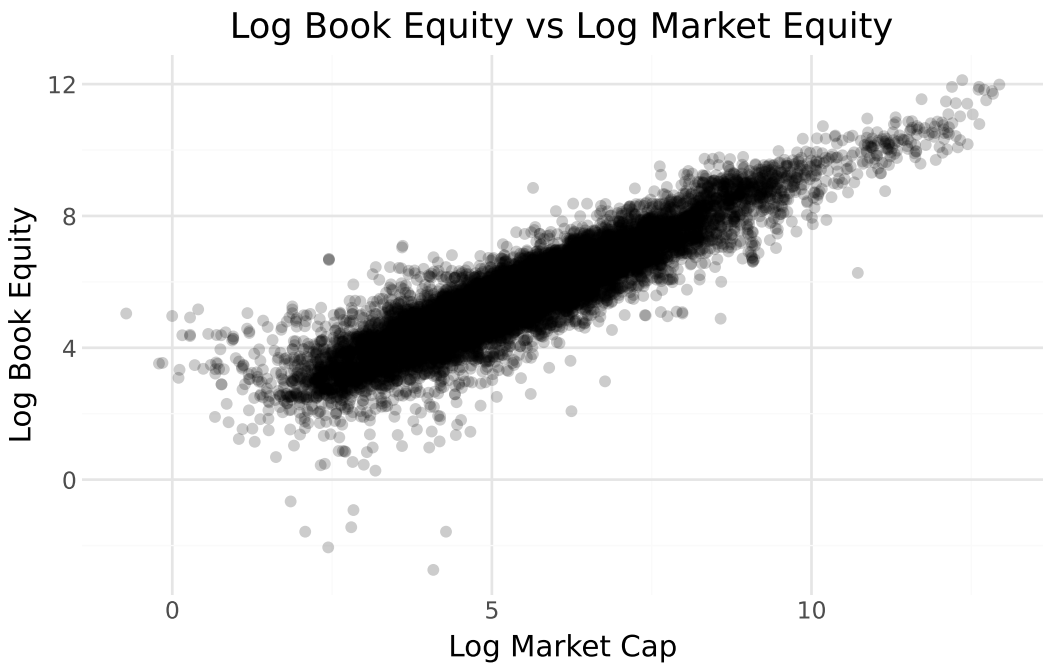

```
size_plot_data = (
    merged_data[["mktcap_lag"]]
      .dropna()
      .assign(log_size=lambda x: np.log(x["mktcap_lag"]))
)

(
    ggplot(size_plot_data, aes(x="log_size")) +
```

```
    geom_histogram(bins=80) +
    labs(
        title="Distribution of Log Market Capitalization",
        x="Log Market Cap",
        y="Number of firms"
    ) +
    theme_minimal()
)
```




```
scatter_data = (
    merged_data[["be", "mktcap_lag"]]
      .dropna()
      .assign(
          log_be=lambda x: np.log(x["be"]),
          log_me=lambda x: np.log(x["mktcap_lag"])
      )
)

(
    ggplot(scatter_data, aes(x="log_me", y="log_be")) +
    geom_point(alpha=0.2) +
    labs(
```

```
        title="Log Book Equity vs Log Market Equity",
        x="Log Market Cap",
        y="Log Book Equity"
    ) +
    theme_minimal()
)
```



8.7 Key Takeaways

1. **Datacore provides unified access** to Vietnamese financial data through a modern cloud-based infrastructure, eliminating the need to aggregate data from multiple fragmented sources.
2. **Company fundamentals** from Datacore include comprehensive balance sheet, income statement, and cash flow data prepared according to Vietnamese Accounting Standards, which we map to standard variables used in international research.
3. **Book equity computation** follows the Fama-French methodology, accounting for deferred taxes and preferred stock to ensure comparability with US-based studies.
4. **Stock price data** includes adjustment factors for corporate actions, enabling accurate return calculations over long horizons.

5. **Monthly frequency** is standard for asset pricing research, reducing noise while maintaining sufficient observations for statistical inference.
6. **Risk-free rate approximation** uses Vietnamese government bond yields as a proxy, given the absence of a standardized short-term rate series comparable to US Treasury bills.
7. **Data quality validation** through descriptive statistics and visualization helps identify potential issues before conducting formal analyses.
8. **Batch processing** enables efficient handling of large daily datasets that would otherwise exceed memory constraints.

9 Beta Estimation

This chapter introduces one of the most fundamental concepts in financial economics: the exposure of an individual stock to systematic market risk. According to the Capital Asset Pricing Model (CAPM) developed by Sharpe (1964), Lintner (1965), and Mossin (1966), cross-sectional variation in expected asset returns should be determined by the covariance between an asset's excess return and the excess return on the market portfolio. The regression coefficient that captures this relationship (commonly known as market beta) serves as the cornerstone of modern portfolio theory and remains widely used in practice for cost of capital estimation, performance attribution, and risk management.

In this chapter, we develop a complete framework for estimating market betas for Vietnamese stocks. We begin with a conceptual overview of the CAPM and its empirical implementation. We then demonstrate beta estimation using ordinary least squares regression, first for individual stocks and then scaled to the entire market using rolling-window estimation. To handle the computational demands of estimating betas for hundreds of stocks across many time periods, we introduce parallelization techniques that dramatically reduce processing time. Finally, we compare beta estimates derived from monthly versus daily returns and examine how betas vary across industries and over time in the Vietnamese market.

The chapter leverages several important computational concepts that extend beyond beta estimation itself. Rolling-window estimation is a technique applicable to any time-varying parameter, while parallelization provides a general solution for computationally intensive tasks that can be divided into independent subtasks.

9.1 Theoretical Foundation

9.1.1 The Capital Asset Pricing Model

The CAPM provides a theoretical framework linking expected returns to systematic risk. Under the model's assumptions—including mean-variance optimizing investors, homogeneous expectations, and frictionless markets—the expected excess return on any asset i is proportional to its covariance with the market portfolio:

$$E[r_i - r_f] = \beta_i \cdot E[r_m - r_f]$$

where r_i is the return on asset i, r_f is the risk-free rate, r_m is the return on the market portfolio, and β_i is defined as:

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

The market beta β_i measures the sensitivity of asset i's returns to market movements. A beta greater than one indicates the asset amplifies market movements, while a beta less than one indicates dampened sensitivity. A beta of zero would imply no systematic risk exposure, leaving only idiosyncratic risk that can be diversified away.

9.1.2 Empirical Implementation

In practice, we estimate beta by regressing excess stock returns on excess market returns:

$$r_{i,t} - r_{f,t} = \alpha_i + \beta_i(r_{m,t} - r_{f,t}) + \varepsilon_{i,t} \tag{9.1}$$

where α_i represents abnormal return (Jensen's alpha), β_i is the market beta we seek to estimate, and $\varepsilon_{i,t}$ is the idiosyncratic error term. Under the CAPM, α_i should equal zero for all assets—any non-zero alpha represents a deviation from the model's predictions.

Several practical considerations affect beta estimation:

1. **Estimation Window**: Longer windows provide more observations and thus more precise estimates, but may include outdated information if betas change over time. Common choices range from 36 to 60 months for monthly data.

2. **Return Frequency**: Monthly returns reduce noise but provide fewer observations. Daily returns offer more data points but may introduce microstructure effects and non-synchronous trading biases.

3. **Market Proxy**: The theoretical market portfolio includes all assets, but in practice we use a broad equity index. For Vietnam, we use the value-weighted market return constructed from our stock universe.

4. **Minimum Observations**: Requiring a minimum number of observations (e.g., 48 out of 60 months) helps avoid unreliable estimates from sparse data.

9.2 Setting Up the Environment

We begin by loading the necessary Python packages. The core packages handle data manipulation, statistical modeling, and database operations. We also import parallelization tools that will be essential when scaling our estimation to the full market.

```
import pandas as pd
import numpy as np
import sqlite3
import statsmodels.formula.api as smf
from scipy.stats.mstats import winsorize

from plotnine import *
from mizani.formatters import percent_format, comma_format
from joblib import Parallel, delayed, cpu_count
from dateutil.relativedelta import relativedelta
```

We connect to our SQLite database containing the processed Vietnamese financial data from previous chapters.

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

9.3 Loading and Preparing Data

9.3.1 Stock Returns Data

We load the monthly stock returns data prepared in the Datacore chapter. The data includes excess returns (returns minus the risk-free rate) for all Vietnamese listed stocks.

```
prices_monthly = pd.read_sql_query(
    sql="""
        SELECT symbol, date, ret_excess
        FROM prices_monthly
    """,
    con=tidy_finance,
    parse_dates={"date"}
)

# Add year for merging with fundamentals
prices_monthly["year"] = prices_monthly["date"].dt.year
```

```
print(f"Loaded {len(prices_monthly):,} monthly observations")
print(f"Covering {prices_monthly['symbol'].nunique():,} unique stocks")
print(f"Date range: {prices_monthly['date'].min():%Y-%m} to {prices_monthly['date'].max():%Y-
```

```
Loaded 209,495 monthly observations
Covering 1,837 unique stocks
Date range: 2010-01 to 2025-05
```

```
prices_daily = pd.read_sql_query(
    sql="""
        SELECT symbol, date, ret_excess
        FROM prices_daily
    """,
    con=tidy_finance,
    parse_dates={"date"}
)
```

9.3.2 Company Information

We load company information to enable industry-level analysis of beta estimates.

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

# Extract year for merging
comp_vn["year"] = comp_vn["datadate"].dt.year

print(f"Company data: {comp_vn['symbol'].nunique():,} firms")
```

```
Company data: 1,502 firms
```

9.3.3 Market Excess Returns

For the market portfolio proxy, we use the value-weighted market excess return. If you have constructed Fama-French factors in a previous chapter, load them here. Otherwise, we can construct a simple market return from our stock data.

```
# Option 1: Load pre-computed market factor
factors_ff3_monthly = pd.read_sql_query(
    sql="SELECT date, mkt_excess FROM factors_ff3_monthly",
    con=tidy_finance,
    parse_dates={"date"}
)

# Option 2: Construct market return from stock data (if factors not available)
# This computes the value-weighted average return across all stocks
def compute_market_return(prices_df):
    """
    Compute value-weighted market return from individual stock returns.

    Parameters
    ----------
    prices_df : pd.DataFrame
        Stock returns with mktcap_lag for weighting

    Returns
    -------
    pd.DataFrame
        Monthly market excess returns
    """
    market_return = (prices_df
        .groupby("date")
        .apply(lambda x: np.average(x["ret_excess"], weights=x["mktcap_lag"]))
        .reset_index(name="mkt_excess")
    )
    return market_return
```

9.3.4 Merging Datasets

We combine the stock returns with market returns and company information to create our estimation dataset.

```
# Merge stock returns with market returns
prices_monthly = prices_monthly.merge(
    factors_ff3_monthly,
    on="date",
    how="left"
)

# Merge with company information for industry classification
prices_monthly = prices_monthly.merge(
    comp_vn[["symbol", "year", "icb_name_vi"]],
    on=["symbol", "year"],
    how="left"
)

# Remove observations with missing data
prices_monthly = prices_monthly.dropna(subset=["ret_excess", "mkt_excess"])

print(f"Final estimation sample: {len(prices_monthly):,} observations")
```

```
Final estimation sample: 169,983 observations
```

9.3.5 Handling Outliers

Extreme returns can unduly influence regression estimates. We apply winsorization to limit the impact of outliers while preserving the general distribution of returns. Winsorization at the 1% level replaces values below the 1st percentile with the 1st percentile value, and values above the 99th percentile with the 99th percentile value.

```
def winsorize_returns(df, columns, limits=(0.01, 0.01)):
    """
    Apply winsorization to return columns to limit outlier influence.

    Parameters
    ----------
    df : pd.DataFrame
        DataFrame containing return columns
    columns : list
        Column names to winsorize
    limits : tuple
        Lower and upper percentile limits for winsorization
```

```
    Returns
    -------
    pd.DataFrame
        DataFrame with winsorized columns
    """
    df = df.copy()
    for col in columns:
        df[col] = winsorize(df[col], limits=limits)
    return df

prices_monthly = winsorize_returns(
    prices_monthly,
    columns=["ret_excess", "mkt_excess"],
    limits=(0.01, 0.01)
)

print("Return distributions after winsorization:")
print(prices_monthly[["ret_excess", "mkt_excess"]].describe().round(4))
```

```
Return distributions after winsorization:
        ret_excess   mkt_excess
count  169983.0000  169983.0000
mean        0.0011      -0.0102
std         0.1548       0.0579
min        -0.4078      -0.1794
25%        -0.0700      -0.0384
50%        -0.0033      -0.0084
75%         0.0531       0.0219
max         0.6117       0.1221
```

9.4 Estimating Beta for Individual Stocks

9.4.1 Single Stock Example

Before scaling to the full market, we demonstrate beta estimation for a single well-known Vietnamese stock. We use Vingroup (VIC), one of the largest conglomerates in Vietnam with significant exposure to real estate, retail, and automotive sectors.

```
# Filter data for Vingroup
vic_data = prices_monthly.query("symbol == 'VIC'").copy()

print(f"VIC observations: {len(vic_data)}")
print(f"Date range: {vic_data['date'].min():%Y-%m} to {vic_data['date'].max():%Y-%m}")
```

```
VIC observations: 150
Date range: 2011-07 to 2023-12
```

We estimate the CAPM regression using ordinary least squares via the `statsmodels` package. The formula interface provides a convenient way to specify regression models.

```
# Estimate CAPM for Vingroup
model_vic = smf.ols(
    formula="ret_excess ~ mkt_excess",
    data=vic_data
).fit()

# Display regression results
print(model_vic.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             ret_excess   R-squared:                       0.153
Model:                            OLS   Adj. R-squared:                  0.147
Method:                 Least Squares   F-statistic:                     26.67
Date:                Wed, 04 Feb 2026   Prob (F-statistic):           7.66e-
07
Time:                        10:20:06   Log-Likelihood:                131.96
No. Observations:                 150   AIC:                             -
259.9
Df Residuals:                     148   BIC:                             -
253.9
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.0075      0.008     -0.895      0.372      -0.024       0.009
mkt_excess     0.7503      0.145      5.164      0.000       0.463       1.037
==============================================================================
```

```
Omnibus:                       39.111   Durbin-Watson:                   2.039
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              107.620
Skew:                          -1.015   Prob(JB):                     4.27e-
24
Kurtosis:                       6.619   Cond. No.                        17.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

The regression output provides several important pieces of information:

- **Beta (mkt_excess coefficient)**: The estimated market sensitivity. A beta above 1 indicates VIC amplifies market movements.
- **Alpha (Intercept)**: The abnormal return not explained by market exposure. Under CAPM, this should be zero.
- **R-squared**: The proportion of return variation explained by market movements.
- **t-statistics**: Test whether coefficients differ significantly from zero.

```
# Extract key estimates
coefficients = model_vic.summary2().tables[1]

print("\nKey estimates for Vingroup (VIC):")
print(f"  Beta:  {coefficients.loc['mkt_excess', 'Coef.']:.3f}")
print(f"  Alpha: {coefficients.loc['Intercept', 'Coef.']:.4f}")
print(f"  R²:    {model_vic.rsquared:.3f}")
```

```
Key estimates for Vingroup (VIC):
  Beta:  0.750
  Alpha: -0.0075
  R²:    0.153
```

9.4.2 CAPM Estimation Function

We create a reusable function that estimates the CAPM and returns results in a standardized format. The function includes a minimum observations requirement to avoid unreliable estimates from sparse data.

```
def estimate_capm(data, min_obs=48):
    """
    Estimate CAPM regression and return coefficients.

    This function regresses excess stock returns on excess market returns
    and extracts the coefficient estimates along with t-statistics.

    Parameters
    ----------
    data : pd.DataFrame
        DataFrame with 'ret_excess' and 'mkt_excess' columns
    min_obs : int
        Minimum number of observations required for estimation

    Returns
    -------
    pd.DataFrame
        DataFrame with coefficient estimates and t-statistics,
        or empty DataFrame if insufficient observations
    """
    if len(data) < min_obs:
        return pd.DataFrame()

    try:
        # Estimate OLS regression
        model = smf.ols(
            formula="ret_excess ~ mkt_excess",
            data=data
        ).fit()

        # Extract coefficient table
        coef_table = model.summary2().tables[1]

        # Format results
        results = pd.DataFrame({
            "coefficient": ["alpha", "beta"],
            "estimate": [
                coef_table.loc["Intercept", "Coef."],
                coef_table.loc["mkt_excess", "Coef."]
            ],
            "t_statistic": [
                coef_table.loc["Intercept", "t"],
```

```
                coef_table.loc["mkt_excess", "t"]
            ],
            "r_squared": model.rsquared
        })

        return results

    except Exception as e:
        # Return empty DataFrame if estimation fails
        return pd.DataFrame()
```

9.5 Rolling-Window Estimation

9.5.1 Motivation for Rolling Windows

Stock betas are not constant over time. A company's business mix, leverage, and operating environment evolve, causing its systematic risk exposure to change. To capture this time variation, we use rolling-window estimation: at each point in time, we estimate beta using only data from a fixed lookback period (e.g., the past 60 months).

Rolling-window estimation involves a trade-off:

- **Longer windows** provide more observations and thus more precise estimates, but may include stale information.
- **Shorter windows** are more responsive to changes but produce noisier estimates.

A common choice in academic research is 60 months (5 years) of monthly data, requiring at least 48 valid observations for estimation.

9.5.2 Rolling Window Implementation

The following function implements rolling-window CAPM estimation. For each month in the sample, it looks back over the specified window and estimates beta using all available data within that window.

```
def roll_capm_estimation(data, look_back=60, min_obs=48):
    """
    Perform rolling-window CAPM estimation.

    This function slides a window across time, estimating the CAPM
    regression at each point using the most recent 'look_back' months
```

```
    of data.

    Parameters
    ----------
    data : pd.DataFrame
        DataFrame with 'date', 'ret_excess', and 'mkt_excess' columns
    look_back : int
        Number of months in the estimation window
    min_obs : int
        Minimum observations required within each window

    Returns
    -------
    pd.DataFrame
        Time series of coefficient estimates with dates
    """
    # Ensure data is sorted by date
    data = data.sort_values("date").copy()

    # Get unique dates
    dates = data["date"].drop_duplicates().sort_values()

    # Container for results
    results = []

    # Slide window across dates
    for i in range(look_back - 1, len(dates)):
        # Define window boundaries
        end_date = dates.iloc[i]
        start_date = end_date - relativedelta(months=look_back - 1)

        # Extract data within window
        window_data = data.query("date >= @start_date and date <= @end_date")

        # Estimate CAPM for this window
        window_results = estimate_capm(window_data, min_obs=min_obs)

        if not window_results.empty:
            window_results["date"] = end_date
            results.append(window_results)

    # Combine all results
```

```
    if results:
        return pd.concat(results, ignore_index=True)
    else:
        return pd.DataFrame()
```

9.5.3 Example: Rolling Betas for Selected Stocks

We demonstrate rolling-window estimation for several well-known Vietnamese stocks spanning different industries.

```
# Define example stocks
examples = pd.DataFrame({
    "symbol": ["FPT", "VNM", "VIC", "HPG", "VCB"],
    "company": [
        "FPT Corporation",      # Technology
        "Vinamilk",             # Consumer goods
        "Vingroup",             # Real estate/conglomerate
        "Hoa Phat Group",       # Steel/materials
        "Vietcombank"           # Banking
    ]
})

# Check data availability for each example
data_availability = (prices_monthly
    .query("symbol in @examples['symbol']")
    .groupby("symbol")
    .agg(
        n_obs=("date", "count"),
        first_date=("date", "min"),
        last_date=("date", "max")
    )
    .reset_index()
)

print("Data availability for example stocks:")
print(data_availability)
```

```
Data availability for example stocks:
  symbol  n_obs first_date  last_date
0    FPT    150 2011-07-31 2023-12-31
1    HPG    150 2011-07-31 2023-12-31
```

```
2    VCB    150 2011-07-31 2023-12-31
3    VIC    150 2011-07-31 2023-12-31
4    VNM    150 2011-07-31 2023-12-31
```

```python
# Estimate rolling betas for example stocks
example_data = prices_monthly.query("symbol in @examples['symbol']")

capm_examples = (example_data
    .groupby("symbol", group_keys=True)
    .apply(lambda x: roll_capm_estimation(x), include_groups=False)
    .reset_index()
    .drop(columns="level_1", errors="ignore")
)

# Filter to beta estimates only
beta_examples = (capm_examples
    .query("coefficient == 'beta'")
    .merge(examples, on="symbol")
)

print(f"Rolling beta estimates: {len(beta_examples):,} observations")
```

```
Rolling beta estimates: 455 observations
```

9.5.4 Visualizing Rolling Betas

Figure 9.1 displays the time series of beta estimates for our example stocks. The figure reveals how systematic risk exposure evolves differently across industries.

```python
rolling_beta_figure = (
    ggplot(
        beta_examples,
        aes(x="date", y="estimate", color="company")
    )
    + geom_line(size=0.8)
    + geom_hline(yintercept=1, linetype="dashed", color="gray", alpha=0.7)
    + labs(
        x="",
        y="Beta",
        color="",
        title="Rolling Beta Estimates (60-Month Window)"
```

```
    )
    + scale_x_datetime(date_breaks="2 years", date_labels="%Y")
    + theme_minimal()
    + theme(legend_position="bottom")
)
rolling_beta_figure.show()
```

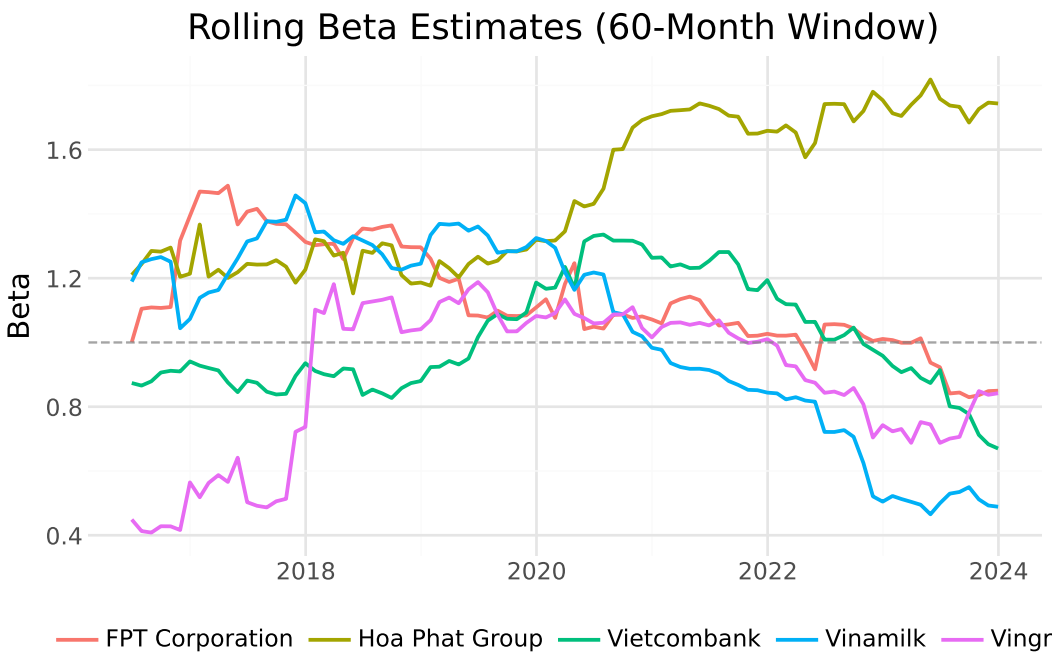


Figure 9.1: Monthly rolling beta estimates for selected Vietnamese stocks using a 60-month estimation window. Different industries exhibit distinct patterns of market sensitivity over time.

Several patterns emerge from the figure:

1. **Industry differences**: Technology and banking stocks may exhibit different beta patterns than real estate or consumer goods companies.
2. **Time variation**: Betas are not constant. They respond to changes in business conditions, leverage, and market regimes.
3. **Crisis periods**: Market stress periods (e.g., 2008 financial crisis, 2020 COVID-19) often see beta estimates change as correlations across stocks increase.

9.6 Parallelized Estimation for the Full Market

9.6.1 The Computational Challenge

Estimating rolling betas for all stocks in our database is computationally intensive. With hundreds of stocks, each requiring rolling estimation across many time periods, sequential processing would take considerable time. Fortunately, beta estimation for different stocks is independent (i.e., the estimate for stock A does not depend on the estimate for stock B). This independence makes the problem ideal for parallelization.

9.6.2 Setting Up Parallel Processing

We use the `joblib` library to distribute computation across multiple CPU cores. The `Parallel` class manages worker processes, while `delayed` wraps function calls for deferred execution.

```
# Determine available cores (reserve one for system operations)
n_cores = max(1, cpu_count() - 1)
print(f"Available cores for parallel processing: {n_cores}")
```

```
Available cores for parallel processing: 3
```

9.6.3 Parallel Beta Estimation

The following code estimates rolling betas for all stocks in parallel. Each stock is processed independently by a separate worker.

```
def estimate_all_betas_parallel(data, n_cores, look_back=60, min_obs=48):
    """
    Estimate rolling betas for all stocks using parallel processing.

    Parameters
    ----------
    data : pd.DataFrame
        Full dataset with all stocks
    n_cores : int
        Number of CPU cores to use
    look_back : int
        Months in estimation window
    min_obs : int
        Minimum observations required
```

```
    Returns
    -------
    pd.DataFrame
        Beta estimates for all stocks and dates
    """
    # Group data by stock
    grouped = data.groupby("symbol", group_keys=False)

    # Define worker function
    def process_stock(name, group):
        result = roll_capm_estimation(group, look_back=look_back, min_obs=min_obs)
        if not result.empty:
            result["symbol"] = name
        return result

    # Execute in parallel
    results = Parallel(n_jobs=n_cores, verbose=1)(
        delayed(process_stock)(name, group)
        for name, group in grouped
    )

    # Combine results
    results = [r for r in results if not r.empty]
    if results:
        return pd.concat(results, ignore_index=True)
    else:
        return pd.DataFrame()
```

```
# Estimate betas for all stocks
print("Estimating rolling betas for all stocks...")
capm_monthly = estimate_all_betas_parallel(
    prices_monthly,
    n_cores=n_cores,
    look_back=60,
    min_obs=48
)

print(f"\nCompleted: {len(capm_monthly):,} coefficient estimates")
print(f"Unique stocks: {capm_monthly['symbol'].nunique():,}")
```

9.6.4 Storing Results

We save the CAPM estimates to our database for use in subsequent chapters.

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

print("CAPM estimates saved to database.")
```

For subsequent analysis, we load the pre-computed estimates:

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

print(f"Loaded {len(capm_monthly):,} CAPM estimates")
```

```
Loaded 161,580 CAPM estimates
```

9.7 Beta Estimation Using Daily Returns

While monthly returns are standard in academic research, some applications benefit from higher-frequency data:

- **Shorter estimation windows**: Daily data allows meaningful estimation over shorter periods (e.g., 3 months rather than 5 years).
- **More responsive estimates**: Daily betas capture changes more quickly.
- **Event studies**: High-frequency betas are useful for analyzing market reactions to specific events.

However, daily data introduces additional challenges:

- **Microstructure noise**: Bid-ask bounce and other trading frictions add noise to returns.
- **Non-synchronous trading**: Less liquid stocks may not trade every day, biasing beta estimates downward.
- **Computational burden**: Daily data is roughly 21 times larger than monthly data.

9.7.1 Batch Processing for Daily Data

Given the size of daily data, we process stocks in batches to manage memory constraints. This approach loads and processes a subset of stocks, saves results, and proceeds to the next batch.

```
def compute_market_return_daily(tidy_finance):
    """
    Compute daily value-weighted market excess return from stock data.
    """
    # Load daily prices with market cap for weighting
    prices_daily_full = pd.read_sql_query(
        sql="""
            SELECT p.symbol, p.date, p.ret_excess, m.mktcap_lag
            FROM prices_daily p
            LEFT JOIN prices_monthly m ON p.symbol = m.symbol
                AND strftime('%Y-%m', p.date) = strftime('%Y-%m', m.date)
        """,
        con=tidy_finance,
        parse_dates={"date"}
    )

    # Compute value-weighted market return each day
    mkt_daily = (prices_daily_full
        .dropna(subset=["ret_excess", "mktcap_lag"])
        .groupby("date")
        .apply(lambda x: np.average(x["ret_excess"], weights=x["mktcap_lag"]))
        .reset_index(name="mkt_excess")
    )

    return mkt_daily


def roll_capm_estimation_daily(data, look_back_days=1260, min_obs=1000):
    """
    Perform rolling-window CAPM estimation using daily data.

    Parameters
    ----------
    data : pd.DataFrame
        DataFrame with 'date', 'ret_excess', and 'mkt_excess' columns
    look_back_days : int
        Number of trading days in the estimation window
    min_obs : int
```

```
        Minimum daily observations required within each window

    Returns
    -------
    pd.DataFrame
        Time series of coefficient estimates with dates
    """
    data = data.sort_values("date").copy()
    dates = data["date"].drop_duplicates().sort_values().reset_index(drop=True)

    results = []

    for i in range(look_back_days - 1, len(dates)):
        end_date = dates.iloc[i]
        start_idx = max(0, i - look_back_days + 1)
        start_date = dates.iloc[start_idx]

        window_data = data.query("date >= @start_date and date <= @end_date")
        window_results = estimate_capm(window_data, min_obs=min_obs)

        if not window_results.empty:
            window_results["date"] = end_date
            results.append(window_results)

    if results:
        return pd.concat(results, ignore_index=True)
    else:
        return pd.DataFrame()


def estimate_daily_betas_batch(symbols, tidy_finance, n_cores, batch_size=500,
                               look_back_days=1260, min_obs=1000):
    """
    Estimate rolling betas from daily data using batch processing.
    """
    # First, compute or load market return
    print("Computing daily market excess returns...")
    mkt_daily = compute_market_return_daily(tidy_finance)
    print(f"Market returns: {len(mkt_daily)} days")

    n_batches = int(np.ceil(len(symbols) / batch_size))
    all_results = []
```

```
    for j in range(n_batches):
        batch_start = j * batch_size
        batch_end = min((j + 1) * batch_size, len(symbols))
        batch_symbols = symbols[batch_start:batch_end]

        symbol_list = ", ".join(f"'{s}'" for s in batch_symbols)

        query = f"""
            SELECT symbol, date, ret_excess
            FROM prices_daily
            WHERE symbol IN ({symbol_list})
        """

        prices_daily_batch = pd.read_sql_query(
            sql=query,
            con=tidy_finance,
            parse_dates={"date"}
        )

        # Merge with market excess return
        prices_daily_batch = prices_daily_batch.merge(
            mkt_daily,
            on="date",
            how="inner"
        )

        # Group by symbol and estimate betas
        grouped = prices_daily_batch.groupby("symbol", group_keys=False)

        # Parallel estimation
        batch_results = Parallel(n_jobs=n_cores)(
            delayed(lambda name, group:
                roll_capm_estimation_daily(group, look_back_days=look_back_days, min_obs=min
                .assign(symbol=name)
            )(name, group)
            for name, group in grouped
        )

        batch_results = [r for r in batch_results if r is not None and not r.empty]

        if batch_results:
            all_results.append(pd.concat(batch_results, ignore_index=True))
```

```
        print(f"Batch {j+1}/{n_batches} complete")

    if all_results:
        return pd.concat(all_results, ignore_index=True)
    else:
        return pd.DataFrame()
```

```
symbols = prices_monthly["symbol"].unique().tolist()

capm_daily = estimate_daily_betas_batch(
    symbols=symbols,
    tidy_finance=tidy_finance,
    n_cores=n_cores,
    batch_size=500,
    look_back_days=1260,  # ~5 years of trading days
    min_obs=1000
)

print(f"Daily beta estimates: {len(capm_daily):,}")
```

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

print("CAPM estimates saved to database.")
```

For subsequent analysis, we load the pre-computed estimates:

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

print(f"Loaded {len(capm_daily):,} CAPM estimates")
```

```
Loaded 3,394,490 CAPM estimates
```

9.8 Analyzing Beta Estimates

9.8.1 Extracting Beta Estimates

We extract the beta coefficient estimates from our CAPM results for analysis.

```
# Extract monthly betas
beta_monthly = (capm_monthly
    .query("coefficient == 'beta'")
    .rename(columns={"estimate": "beta"})
    [["symbol", "date", "beta"]]
    .assign(frequency="monthly")
)

# Save to database
beta_monthly.to_sql(
    name="beta_monthly",
    con=tidy_finance,
    if_exists="replace",
    index=False
)

print(f"Monthly betas: {len(beta_monthly):,} observations")
print(f"Unique stocks: {beta_monthly['symbol'].nunique():,}")
```

```
Monthly betas: 80,790 observations
Unique stocks: 1,383
```

```
# Load pre-computed betas
beta_monthly = pd.read_sql_query(
    sql="SELECT * FROM beta_monthly",
    con=tidy_finance,
    parse_dates={"date"}
)
```

9.8.2 Summary Statistics

We examine the distribution of beta estimates to verify their reasonableness.

```
print("Beta Summary Statistics:")
print(beta_monthly["beta"].describe().round(3))

# Additional diagnostics
print(f"\nStocks with negative average beta: {(beta_monthly.groupby('symbol')['beta'].mean()
print(f"Stocks with beta > 2: {(beta_monthly.groupby('symbol')['beta'].mean() > 2).sum()}")
```

```
Beta Summary Statistics:
count    80790.000
mean         0.501
std          0.539
min         -1.345
25%          0.130
50%          0.447
75%          0.832
max          2.678
Name: beta, dtype: float64

Stocks with negative average beta: 177
Stocks with beta > 2: 5
```

9.8.3 Beta Distribution Across Industries

Different industries have different exposures to systematic market risk based on their business models, operating leverage, and financial leverage. Figure 9.2 shows the distribution of firm-level average betas across Vietnamese industries.

```
# Merge betas with industry information
beta_with_industry = (beta_monthly
    .merge(
        prices_monthly[["symbol", "date", "icb_name_vi"]].drop_duplicates(),
        on=["symbol", "date"],
        how="left"
    )
    .dropna(subset=["icb_name_vi"])
)

# Compute firm-level average beta by industry
beta_by_industry = (beta_with_industry
    .groupby(["icb_name_vi", "symbol"])["beta"]
    .mean()
```

```
    .reset_index()
)

# Order industries by median beta
industry_order = (beta_by_industry
    .groupby("icb_name_vi")["beta"]
    .median()
    .sort_values()
    .index.tolist()
)

# Select top 10 industries by number of firms for clearer visualization
top_industries = (beta_by_industry
    .groupby("icb_name_vi")
    .size()
    .nlargest(10)
    .index.tolist()
)

beta_by_industry_filtered = beta_by_industry.query("icb_name_vi in @top_industries")
```

```
beta_industry_figure = (
    ggplot(
        beta_by_industry_filtered,
        aes(x="icb_name_vi", y="beta")
    )
    + geom_boxplot(fill="steelblue", alpha=0.7)
    + geom_hline(yintercept=1, linetype="dashed", color="red", alpha=0.7)
    + coord_flip()
    + labs(
        x="",
        y="Beta",
        title="Beta Distribution by Industry"
    )
    + theme_minimal()
)
beta_industry_figure.show()
```

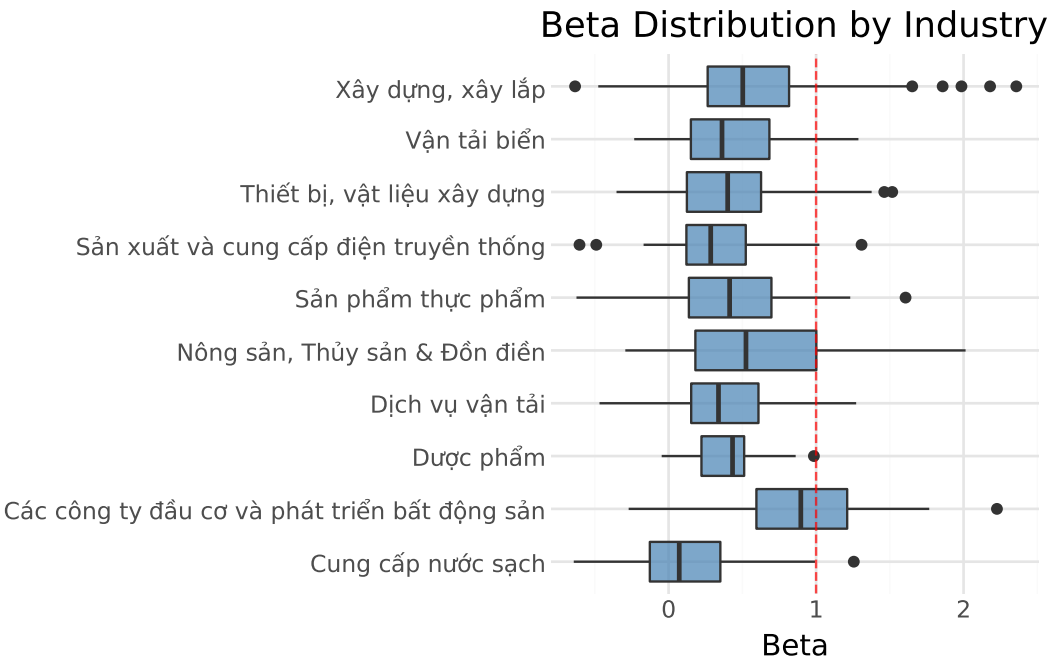


Figure 9.2: Distribution of firm-level average betas across Vietnamese industries. Box plots show the median, interquartile range, and outliers for each industry.

9.8.4 Time Variation in Cross-Sectional Beta Distribution

Betas vary not only across stocks but also over time. Figure 9.3 shows how the cross-sectional distribution of betas has evolved in the Vietnamese market.

```
# Compute monthly quantiles
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).astype(str) + "%")
)

beta_quantiles_figure = (
    ggplot(
        beta_quantiles,
        aes(x="date", y="beta", color="quantile")
    )
```

```
    + geom_line(alpha=0.8)
    + geom_hline(yintercept=1, linetype="dashed", color="gray")
    + labs(
        x="",
        y="Beta",
        color="Quantile",
        title="Cross-Sectional Distribution of Betas Over Time"
    )
    + scale_x_datetime(date_breaks="2 years", date_labels="%Y")
    + theme_minimal()
)
beta_quantiles_figure.show()
```

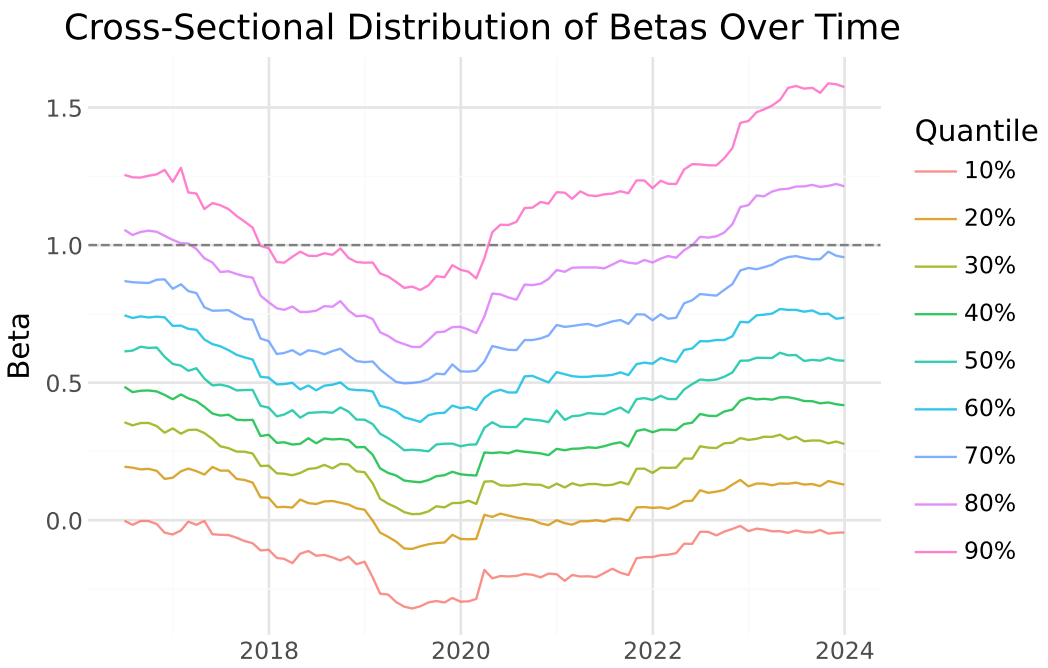


Figure 9.3: Monthly quantiles of beta estimates over time. Each line represents a decile of the cross-sectional beta distribution.

The figure reveals several interesting patterns:

1. **Level shifts**: The entire distribution of betas can shift over time, reflecting changes in market-wide correlation.
2. **Dispersion changes**: During market stress, the spread between high and low beta stocks may change as correlations move.

3. **Trends**: Some periods show trending behavior in betas, possibly reflecting structural changes in the economy.

9.8.5 Coverage Analysis

We verify that our estimation procedure produces reasonable coverage across the sample. Figure 9.4 shows the fraction of stocks with available beta estimates over time.

```
# Count stocks with and without betas
coverage = (prices_monthly
    .groupby("date")["symbol"]
    .nunique()
    .reset_index(name="total_stocks")
    .merge(
        beta_monthly.groupby("date")["symbol"].nunique().reset_index(name="with_beta"),
        on="date",
        how="left"
    )
    .fillna(0)
    .assign(coverage=lambda x: x["with_beta"] / x["total_stocks"])
)

coverage_figure = (
    ggplot(coverage, aes(x="date", y="coverage"))
    + geom_line(color="steelblue", size=1)
    + labs(
        x="",
        y="Share with Beta Estimate",
        title="Beta Estimation Coverage Over Time"
    )
    + scale_y_continuous(labels=percent_format(), limits=(0, 1))
    + scale_x_datetime(date_breaks="2 years", date_labels="%Y")
    + theme_minimal()
)
coverage_figure.show()
```

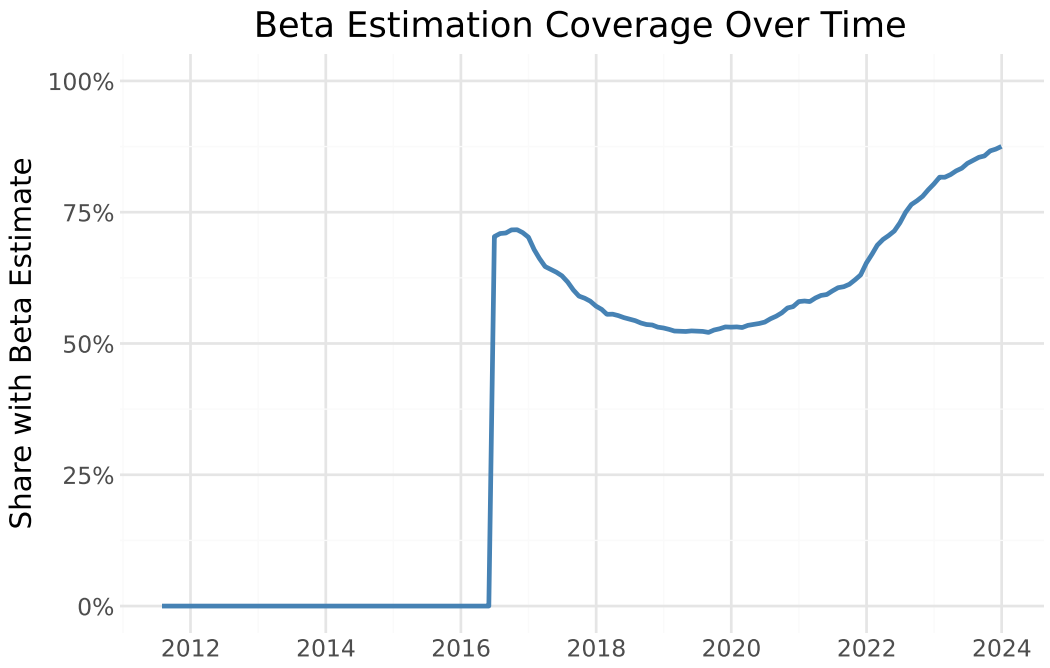


Figure 9.4: Share of stocks with available beta estimates over time. Coverage increases as more stocks accumulate sufficient return history.

Coverage is lower in early years because stocks need sufficient return history (at least 48 months) before their betas can be estimated. As the market matures and stocks accumulate longer histories, coverage approaches 100%.

9.9 Comparing Monthly and Daily Beta Estimates

When both monthly and daily beta estimates are available, we can compare them to understand how estimation frequency affects results.

```
# Combine monthly and daily estimates
beta_daily = (capm_daily
    .query("coefficient == 'beta'")
    .rename(columns={"estimate": "beta"})
    [["symbol", "date", "beta"]]
    .assign(frequency="daily")
)

beta_combined = pd.concat([beta_monthly, beta_daily], ignore_index=True)
```

```
# Filter to example stocks
beta_comparison = (beta_combined
    .merge(examples, on="symbol")
    .query("symbol in ['VIC', 'FPT']")  # Select two for clarity
)

comparison_figure = (
    ggplot(
        beta_comparison,
        aes(x="date", y="beta", color="frequency", linetype="frequency")
    )
    + geom_line(size=0.8)
    + facet_wrap("~company", ncol=1)
    + labs(
        x="",
        y="Beta",
        color="Data Frequency",
        linetype="Data Frequency",
        title="Monthly vs Daily Beta Estimates"
    )
    + scale_x_datetime(date_breaks="2 years", date_labels="%Y")
    + theme_minimal()
    + theme(legend_position="bottom")
)
comparison_figure.show()
```

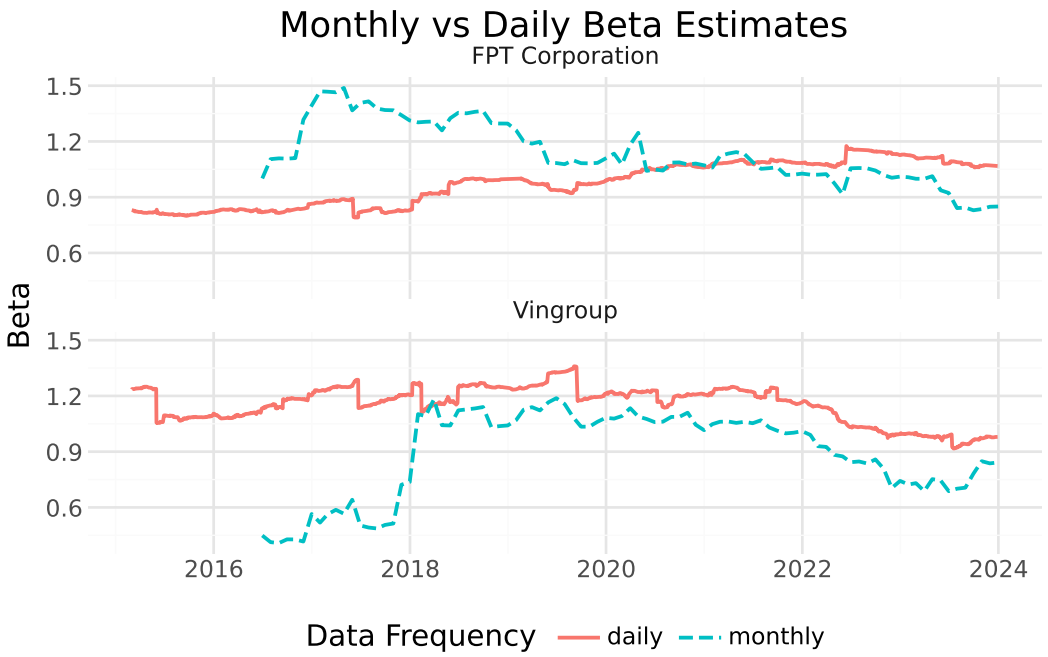


Figure 9.5: Comparison of beta estimates using monthly versus daily returns for selected stocks. Daily estimates are smoother due to more observations per estimation window.

The comparison reveals that daily-based estimates are generally smoother due to the larger number of observations in each window. However, the level and trend of estimates are similar across frequencies, providing validation that both approaches capture the same underlying systematic risk exposure.

```
# Correlation between monthly and daily estimates
correlation_data = (beta_combined
    .pivot_table(index=["symbol", "date"], columns="frequency", values="beta")
    .dropna()
)

print(f"Correlation between monthly and daily betas: {correlation_data.corr().iloc[0,1]:.3f}"
```

```
Correlation between monthly and daily betas: 0.745
```

Table 9.1: Theoretical Reasons for Imperfect Correlation

Factor	Effect
Non-synchronous trading	Daily betas can be biased downward for illiquid stocks
Microstructure noise	Bid-ask bounce adds noise to daily estimates
Different effective windows	Same calendar period but ~20x more observations for daily
Mean reversion speed	Daily captures faster-moving risk dynamics

Table 9.1 shows several reasons why we might observe imperfect correlation.

9.10 Key Takeaways

1. **CAPM beta** measures a stock's sensitivity to systematic market risk and is fundamental to modern portfolio theory, cost of capital estimation, and risk management.
2. **Rolling-window estimation** captures time variation in betas, which reflects changes in companies' business models, leverage, and market conditions.
3. **Parallelization** dramatically reduces computation time for large-scale estimation tasks by distributing work across multiple CPU cores.
4. **Estimation choices matter**: Window length, return frequency, and minimum observation requirements all affect beta estimates. Researchers should choose parameters appropriate for their specific application.
5. **Industry patterns**: Vietnamese stocks show systematic differences in market sensitivity across industries, with cyclical sectors exhibiting higher betas than defensive sectors.
6. **Time variation**: The cross-sectional distribution of betas in Vietnam has evolved over time, with notable shifts during market stress periods.
7. **Frequency comparison**: Monthly and daily beta estimates are positively correlated but not identical. Daily estimates are smoother while monthly estimates may better capture lower-frequency variation.
8. **Data quality checks**: Coverage analysis and summary statistics help identify potential issues in estimation procedures before using results in downstream analyses.

10 Univariate Portfolio Sorts

In this chapter, we dive into portfolio sorts, one of the most widely used statistical methodologies in empirical asset pricing (e.g., Bali, Engle, and Murray 2016). The key application of portfolio sorts is to examine whether one or more variables can predict future excess returns. In general, the idea is to sort individual stocks into portfolios, where the stocks within each portfolio are similar with respect to a sorting variable, such as firm size. The different portfolios then represent well-diversified investments that differ in the level of the sorting variable. You can then attribute the differences in the return distribution to the impact of the sorting variable. We start by introducing univariate portfolio sorts (which sort based on only one characteristic) and tackle bivariate sorting in Value and Bivariate Sorts.

A univariate portfolio sort considers only one sorting variable $x_{i,t-1}$. Here, i denotes the stock and $t-1$ indicates that the characteristic is observable by investors at time t. The objective is to assess the cross-sectional relation between $x_{i,t-1}$ and, typically, stock excess returns $r_{i,t}$ at time t as the outcome variable. To illustrate how portfolio sorts work, we use estimates for market betas from the previous chapter as our sorting variable.

```
import pandas as pd
import numpy as np
import sqlite3
import statsmodels.api as sm

from plotnine import *
from mizani.formatters import percent_format
from regtabletotext import prettify_result
```

10.1 Data Preparation

We start with loading the required data from our SQLite database introduced in Accessing and Managing Financial Data and DataCore Data. In particular, we use the monthly stock price data as our asset universe. Once we form our portfolios, we use the market factor returns to compute the risk-adjusted performance (i.e., alpha). `beta` is the dataframe with market betas computed in the previous chapter.

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

prices_monthly = (pd.read_sql_query(
    sql="SELECT symbol, date, ret_excess, mktcap_lag FROM prices_monthly",
    con=tidy_finance,
    parse_dates={"date"})
)

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

beta = pd.read_sql_query(
  sql="SELECT symbol, date, beta FROM beta_monthly",
  con=tidy_finance,
  parse_dates={"date"}
)
```

10.2 Sorting by Market Beta

Next, we merge our sorting variable with the return data. We use the one-month *lagged* betas as a sorting variable to ensure that the sorts rely only on information available when we create the portfolios. To lag stock beta by one month, we add one month to the current date and join the resulting information with our return data. This procedure ensures that month t information is available in month $t + 1$. You may be tempted to simply use a call such as `prices_monthly['beta_lag'] = prices_monthly.groupby('symbol')['beta'].shift(1)` instead. This procedure, however, does not work correctly if there are implicit missing values in the time series.

```
beta_lag = (beta
  .assign(date=lambda x: x["date"]+pd.DateOffset(months=1))
  .get(["symbol", "date", "beta"])
  .rename(columns={"beta": "beta_lag"})
  .dropna()
)

data_for_sorts = (prices_monthly
  .merge(beta_lag, how="inner", on=["symbol", "date"])
```

```
  .dropna()
)
```

The first step to conduct portfolio sorts is to calculate periodic breakpoints that you can use to group the stocks into portfolios. For simplicity, we start with the median lagged market beta as the single breakpoint. We then compute the value-weighted returns for each of the two resulting portfolios, which means that the lagged market capitalization determines the weight in `np.average()`.

```
beta_portfolios = (
    data_for_sorts
    .groupby("date")
    .apply(lambda x: (
        x.assign(
            portfolio=pd.qcut(x["beta_lag"], q=[0, 0.5, 1], labels=["low", "high"]),
            date=x.name
        )
    ))
    .reset_index(drop=True)
    .groupby(["portfolio", "date"])
    .apply(lambda x: np.average(x["ret_excess"], weights=x["mktcap_lag"]))
    .reset_index(name="ret")
    .dropna(subset=['ret'])
)
beta_portfolios.head()
```

	portfolio	date	ret
0	low	2016-08-31	-0.016507
1	low	2016-09-30	-0.089155
2	low	2016-11-30	-0.015080
3	low	2017-01-31	0.004113
4	low	2017-02-28	0.035101

10.3 Performance Evaluation

We can construct a long-short strategy based on the two portfolios: buy the high-beta portfolio and, at the same time, short the low-beta portfolio. Thereby, the overall position in the market is net-zero.

```
beta_longshort = (beta_portfolios
  .pivot_table(index="date", columns="portfolio", values="ret")
  .reset_index()
  .assign(long_short=lambda x: x["high"]-x["low"])
)
```

We compute the average return and the corresponding standard error to test whether the long-short portfolio yields on average positive or negative excess returns. In the asset pricing literature, one typically adjusts for autocorrelation by using Newey and West (1987) t-statistics to test the null hypothesis that average portfolio excess returns are equal to zero. One necessary input for Newey-West standard errors is a chosen bandwidth based on the number of lags employed for the estimation. Researchers often default to choosing a pre-specified lag length of six months (which is not a data-driven approach). We do so in the `fit()` function by indicating the `cov_type` as `HAC` and providing the maximum lag length through an additional keywords dictionary.

```
model_fit = (sm.OLS.from_formula(
    formula="long_short ~ 1",
    data=beta_longshort
  )
  .fit(cov_type="HAC", cov_kwds={"maxlags": 6})
)
prettify_result(model_fit)
```

```
OLS Model:
long_short ~ 1

Coefficients:
           Estimate  Std. Error  t-Statistic  p-Value
Intercept     0.006       0.006        1.018    0.309

Summary statistics:
- Number of observations: 53
- R-squared: 0.000, Adjusted R-squared: 0.000
- F-statistic not available
```

The results indicate that we cannot reject the null hypothesis of average returns being equal to zero. Our portfolio strategy using the median as a breakpoint does not yield any abnormal returns. Is this finding surprising if you reconsider the CAPM? It certainly is. The CAPM predicts that high-beta stocks should yield higher expected returns. Our portfolio sort implicitly mimics an investment strategy that finances high-beta stocks by shorting low-beta stocks.

Therefore, one should expect that the average excess returns yield a return that is above the risk-free rate.

10.4 Functional Programming for Portfolio Sorts

Now, we take portfolio sorts to the next level. We want to be able to sort stocks into an arbitrary number of portfolios. For this case, functional programming is very handy: we define a function that gives us flexibility concerning which variable to use for the sorting, denoted by `sorting_variable`. We use `np.quantile()` to compute breakpoints for `n_portfolios`. Then, we assign portfolios to stocks using the `pd.cut()` function. The output of the following function is a new column that contains the number of the portfolio to which a stock belongs.

In some applications, the variable used for the sorting might be clustered (e.g., at a lower bound of 0). Then, multiple breakpoints may be identical, leading to empty portfolios. Similarly, some portfolios might have a very small number of stocks at the beginning of the sample. Cases where the number of portfolio constituents differs substantially due to the distribution of the characteristics require careful consideration and, depending on the application, might require customized sorting approaches.

```
def assign_portfolio(data, sorting_variable, n_portfolios):
    """Assign portfolios to a bin between breakpoints."""

    breakpoints = np.quantile(
      data[sorting_variable].dropna(),
      np.linspace(0, 1, n_portfolios + 1),
      method="linear"
    )

    assigned_portfolios = pd.cut(
      data[sorting_variable],
      bins=breakpoints,
      labels=range(1, breakpoints.size),
      include_lowest=True,
      right=False
    )

    return assigned_portfolios
```

We can use the above function to sort stocks into ten portfolios each month using lagged betas and compute value-weighted returns for each portfolio. Note that we transform the portfolio column to a factor variable because it provides more convenience for the figure construction below.

```
beta_portfolios = (data_for_sorts
  .groupby("date")
  .apply(lambda x: x.assign(
      portfolio=assign_portfolio(x, "beta_lag", 10)
    ), include_groups=False
  )
  .reset_index(level="date")
  .groupby(["portfolio", "date"])
  .apply(lambda x: pd.Series({
      "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"])
    }), include_groups=False
  )
  .reset_index()
  .merge(factors_ff3_monthly, how="left", on="date")
)
```

10.5 More Performance Evaluation

In the next step, we compute summary statistics for each beta portfolio. Namely, we compute CAPM-adjusted alphas, the beta of each beta portfolio, and average returns.

```
beta_portfolios_summary = (beta_portfolios
  .groupby("portfolio")
  .apply(lambda x: pd.Series({
      "alpha": sm.OLS.from_formula(
          formula="ret ~ 1 + mkt_excess",
          data=x
        ).fit().params["Intercept"],
      "beta": sm.OLS.from_formula(
          formula="ret ~ 1 + mkt_excess",
          data=x
        ).fit().params["mkt_excess"],
      "ret": x["ret"].mean()
    }), include_groups=False
  )
  .reset_index()
)
beta_portfolios_summary
```

	portfolio	alpha	beta	ret
0	1	0.007031	0.356225	0.003892
1	2	-0.004215	0.227882	-0.005509
2	3	-0.010169	0.390216	-0.011241
3	4	-0.014205	0.388342	-0.015732
4	5	-0.007220	0.616833	-0.009723
5	6	0.010648	0.976502	0.006577
6	7	-0.010739	0.679145	-0.012645
7	8	-0.002757	0.991774	-0.006963
8	9	-0.003448	0.904886	-0.007174
9	10	0.010675	1.376356	0.004944

Figure 10.1 illustrates the CAPM alphas of beta-sorted portfolios.

```
beta_portfolios_figure = (
  ggplot(
    beta_portfolios_summary,
    aes(x="portfolio", y="alpha", fill="portfolio")
  )
  + geom_bar(stat="identity")
  + labs(
      x="Portfolio", y="CAPM alpha", fill="Portfolio",
      title="CAPM alphas of beta-sorted portfolios"
    )
  + scale_y_continuous(labels=percent_format())
  + theme(legend_position="none")
)
beta_portfolios_figure.show()
```

Unlike the well-documented "betting against beta" anomaly in US markets, where low-beta portfolios exhibit positive alphas and high-beta portfolios exhibit negative alphas in a monotonic pattern, the Vietnamese market shows no clear relationship between beta and risk-adjusted returns. The alphas fluctuate without a discernible trend across deciles. This lack of pattern likely reflects the limited sample period rather than a definitive conclusion about beta pricing in Vietnam. With such a short time series, the portfolio-level CAPM regressions contain substantial estimation noise, making it difficult to detect subtle anomalies. Longer sample periods would be needed to draw reliable conclusions about whether the low-beta anomaly exists in the Vietnamese equity market.

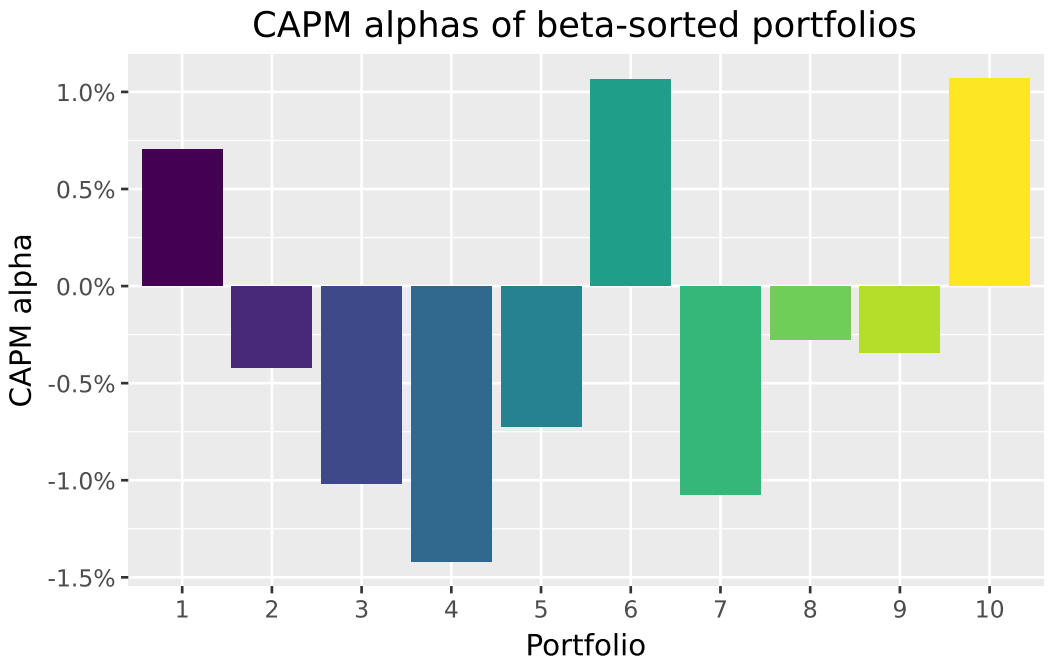


Figure 10.1: The figure shows CAPM alphas of beta-sorted portfolios. Portfolios are sorted into deciles each month based on their estimated CAPM beta. The bar charts indicate the CAPM alpha of the resulting portfolio returns during the sample period.

10.6 Security Market Line and Beta Portfolios

The CAPM predicts that our portfolios should lie on the security market line (SML). The slope of the SML is equal to the market risk premium and reflects the risk-return trade-off at any given time. Figure 10.2 illustrates the security market line: We see that (not surprisingly) the high-beta portfolio returns have a high correlation with the market returns. However, it seems like the average excess returns for high-beta stocks are lower than what the security market line implies would be an "appropriate" compensation for the high market risk.

```
sml_capm = (sm.OLS.from_formula(
    formula="ret ~ 1 + beta",
    data=beta_portfolios_summary
  )
  .fit()
  .params
)

sml_figure = (
```

```
  ggplot(
    beta_portfolios_summary,
    aes(x="beta", y="ret", color="factor(portfolio)")
  )
  + geom_point()
  + geom_abline(
      intercept=0, slope=factors_ff3_monthly["mkt_excess"].mean(), linetype="solid"
    )
  + geom_abline(
      intercept=sml_capm["Intercept"], slope=sml_capm["beta"], linetype="dashed"
    )
  + labs(
      x="Beta", y="Excess return", color="Portfolio",
      title="Average portfolio excess returns and beta estimates"
    )
  + scale_x_continuous(limits=(0, 2))
  + scale_y_continuous(labels=percent_format())
)
sml_figure.show()
```

To provide more evidence against the CAPM predictions, we again form a long-short strategy that buys the high-beta portfolio and shorts the low-beta portfolio.

```
beta_longshort = (beta_portfolios
  .assign(
    portfolio=lambda x: (
      x["portfolio"].apply(
        lambda y: "high" if y == x["portfolio"].max()
        else ("low" if y == x["portfolio"].min()
        else y)
      )
    )
  )
  .query("portfolio in ['low', 'high']")
  .pivot_table(index="date", columns="portfolio", values="ret")
  .assign(long_short=lambda x: x["high"]-x["low"])
  .merge(factors_ff3_monthly, how="left", on="date")
)
```

Again, the resulting long-short strategy does not exhibit statistically significant returns.

Figure 10.2: The figure shows average portfolio excess returns and beta estimates. Excess returns are computed as CAPM alphas of the beta-sorted portfolios. The horizontal axis indicates the CAPM beta of the resulting beta-sorted portfolio return time series. The dashed line indicates the slope coefficient of a linear regression of excess returns on portfolio betas.

```
model_fit = (sm.OLS.from_formula(
    formula="long_short ~ 1",
    data=beta_longshort
  )
  .fit(cov_type="HAC", cov_kwds={"maxlags": 1})
)
prettify_result(model_fit)
```

```
OLS Model:
long_short ~ 1

Coefficients:
           Estimate  Std. Error  t-Statistic  p-Value
Intercept     0.001       0.011        0.095    0.924

Summary statistics:
```

```
- Number of observations: 53
- R-squared: 0.000, Adjusted R-squared: 0.000
- F-statistic not available
```

However, controlling for the effect of beta, the long-short portfolio yields a CAPM-adjusted alpha. The results can provide evidence regarding the validity of the CAPM in the Vietnamese market. The betting-against-beta factor has been documented extensively in developed markets (Frazzini and Pedersen 2014). Betting-against-beta corresponds to a strategy that shorts high-beta stocks and takes a (levered) long position in low-beta stocks. If borrowing constraints prevent investors from taking positions on the security market line, they are instead incentivized to buy high-beta stocks, which leads to a relatively higher price (and therefore lower expected returns than implied by the CAPM) for such high-beta stocks. As a result, the betting-against-beta strategy earns from providing liquidity to capital-constrained investors with lower risk aversion.

```
model_fit = (sm.OLS.from_formula(
    formula="long_short ~ 1 + mkt_excess",
    data=beta_longshort
  )
  .fit(cov_type="HAC", cov_kwds={"maxlags": 1})
)
prettify_result(model_fit)
```

```
OLS Model:
long_short ~ 1 + mkt_excess

Coefficients:
            Estimate  Std. Error  t-Statistic  p-Value
Intercept      0.004       0.009        0.394    0.694
mkt_excess     1.020       0.116        8.784    0.000

Summary statistics:
- Number of observations: 52
- R-squared: 0.527, Adjusted R-squared: 0.518
- F-statistic: 77.156 on 1 and 50 DF, p-value: 0.000
```

Figure 10.3 shows the annual returns of the extreme beta portfolios we are mainly interested in. The figure illustrates the patterns over the sample period; each portfolio exhibits periods with positive and negative annual returns.

```
beta_longshort_year = (beta_longshort
  .assign(year=lambda x: x["date"].dt.year)
  .groupby("year")
  .aggregate(
    low=("low", lambda x: (1+x).prod()-1),
    high=("high", lambda x: (1+x).prod()-1),
    long_short=("long_short", lambda x: (1+x).prod()-1)
  )
  .reset_index()
  .melt(id_vars="year", var_name="name", value_name="value")
)

beta_longshort_figure = (
  ggplot(
    beta_longshort_year,
    aes(x="year", y="value", fill="name")
  )
  + geom_col(position="dodge")
  + facet_wrap("~name", ncol=1)
  + labs(x="", y="", title="Annual returns of beta portfolios")
  + scale_y_continuous(labels=percent_format())
  + theme(legend_position="none")
)
beta_longshort_figure.show()
```

The high-beta portfolio and low-beta portfolio both exhibit substantial year-to-year variation. The long-short portfolio, which goes long high-beta stocks and short low-beta stocks, shows no consistent pattern of positive returns. This erratic performance reinforces our earlier finding that the beta-return relationship in the Vietnamese market does not conform to theoretical CAPM predictions during our sample period. The high volatility of annual long-short returns highlights the substantial risk inherent in such a strategy, particularly in an emerging market context with a limited sample period.

10.7 Key Takeaways

1. Univariate portfolio sorts assess whether a single firm characteristic, like lagged market beta, can predict future excess returns.
2. Portfolios are formed each month using quantile breakpoints, with returns computed using value-weighted averages to reflect realistic investment strategies.

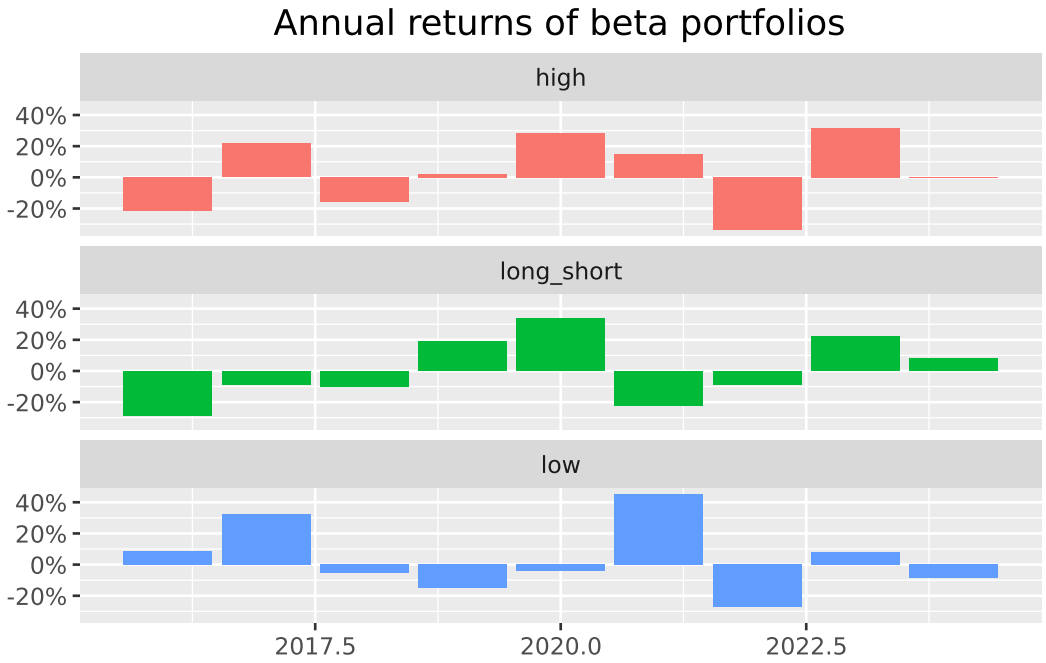


Figure 10.3: The figure shows annual returns of beta portfolios. We construct portfolios by sorting stocks into high and low based on their estimated CAPM beta. Long short indicates a strategy that goes long into high beta stocks and short low beta stocks.

3. A long-short strategy based on beta-sorted portfolios fails to generate significant positive excess returns in the Vietnamese market, contradicting CAPM predictions that higher beta should yield higher returns.

4. The analysis provides a framework for examining the "betting against beta" anomaly, where low-beta portfolios may deliver higher alphas than high-beta portfolios, offering evidence regarding the validity of the CAPM.

5. The functional programming capabilities of Python enable scalable and flexible portfolio sorting, making it easy to analyze multiple characteristics and portfolio configurations.

6. Emerging markets like Vietnam may exhibit different beta-return relationships compared to developed markets, highlighting the importance of conducting market-specific empirical analysis rather than assuming universal applicability of asset pricing anomalies.

11 Fama-French Factors

This chapter provides a replication of the Fama-French factor portfolios for the Vietnamese stock market. The Fama-French factor models represent a cornerstone of empirical asset pricing, originating from the seminal work of Fama and French (1992) and later extended in Fama and French (2015). These models have transformed how academics and practitioners understand the cross-section of expected stock returns, moving beyond the single-factor Capital Asset Pricing Model to incorporate multiple sources of systematic risk.

We construct both the three-factor and five-factor models at monthly and daily frequencies. The monthly factors serve as the foundation for most asset pricing tests and portfolio analyses, while the daily factors enable higher-frequency applications including short-horizon event studies, market microstructure research, and daily beta estimation. By constructing factors at both frequencies, we create a complete toolkit for empirical finance research in the Vietnamese market.

The chapter proceeds as follows. We first discuss the theoretical motivation for each factor and the economic intuition behind the Fama-French methodology. We then prepare the necessary data, merging stock returns with accounting characteristics. Next, we implement the portfolio sorting procedures that form the basis of factor construction, carefully following the original Fama-French protocols while adapting them for Vietnamese market characteristics. We construct the three-factor model (market, size, and value) before extending to the five-factor model (adding profitability and investment). Finally, we construct daily factors and validate our replicated factors through various diagnostic checks.

11.1 Theoretical Background

11.1.1 The Evolution from CAPM to Multi-Factor Models

The Capital Asset Pricing Model of Sharpe (1964) posits that a single factor—the market portfolio—should explain all cross-sectional variation in expected returns. However, decades of empirical research have documented persistent patterns that CAPM cannot explain. Fama and French (1992) demonstrated that two firm characteristics—size and book-to-market ratio—capture substantial variation in average returns that the market beta leaves unexplained.

Small firms tend to earn higher returns than large firms, a pattern known as the size effect. Similarly, firms with high book-to-market ratios (value stocks) tend to outperform firms with

low book-to-market ratios (growth stocks), known as the value premium. The three-factor model formalizes these observations by constructing tradeable factor portfolios:

$$r_{i,t} - r_{f,t} = \alpha_i + \beta_i^{MKT}(r_{m,t} - r_{f,t}) + \beta_i^{SMB} \cdot SMB_t + \beta_i^{HML} \cdot HML_t + \varepsilon_{i,t} \tag{11.1}$$

where:

- $r_{i,t} - r_{f,t}$ is the excess return on asset i
- $r_{m,t} - r_{f,t}$ is the market excess return
- SMB_t (Small Minus Big) is the size factor
- HML_t (High Minus Low) is the value factor

11.1.2 The Five-Factor Extension

Fama and French (2015) extended the model to include two additional factors motivated by the dividend discount model. Firms with higher profitability should have higher expected returns (all else equal), and firms with aggressive investment policies should have lower expected returns:

$$r_{i,t} - r_{f,t} = \alpha_i + \beta_i^{MKT} MKT_t + \beta_i^{SMB} SMB_t + \beta_i^{HML} HML_t + \beta_i^{RMW} RMW_t + \beta_i^{CMA} CMA_t + \varepsilon_{i,t} \tag{11.2}$$

where:

- RMW_t (Robust Minus Weak) is the profitability factor
- CMA_t (Conservative Minus Aggressive) is the investment factor

11.1.3 Factor Construction Methodology

The Fama-French methodology constructs factors through double-sorted portfolios:

1. **Size sorts**: Stocks are divided into Small and Big groups based on median market capitalization.
2. **Characteristic sorts**: Within each size group, stocks are sorted into terciles based on book-to-market (for HML), operating profitability (for RMW), or investment (for CMA).
3. **Factor returns**: Factors are computed as the difference between average returns of portfolios with high versus low characteristic values, averaging across size groups to neutralize size effects.
4. **Timing**: Portfolios are formed at the end of June each year using accounting data from the prior fiscal year, ensuring all information was publicly available at formation.

11.2 Setting Up the Environment

We load the required Python packages for data manipulation, statistical analysis, and visualization.

```
import pandas as pd
import numpy as np
import sqlite3
import statsmodels.formula.api as smf
from scipy.stats.mstats import winsorize
import matplotlib.pyplot as plt

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

We connect to our SQLite database containing the processed Vietnamese financial data.

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

11.3 Data Preparation

11.3.1 Loading Stock Returns

We load the monthly stock returns data, which includes excess returns, market capitalization, and the risk-free rate. These variables are essential for computing value-weighted portfolio returns and factor premiums.

```
prices_monthly = pd.read_sql_query(
    sql="""
        SELECT symbol, date, ret_excess, mktcap, mktcap_lag, risk_free
        FROM prices_monthly
    """,
    con=tidy_finance,
    parse_dates={"date"}
).dropna()

print(f"Monthly returns: {len(prices_monthly):,} observations")
print(f"Unique stocks: {prices_monthly['symbol'].nunique():,}")
print(f"Date range: {prices_monthly['date'].min():%Y-%m} to {prices_monthly['date'].max():%Y-
```

```
Monthly returns: 165,499 observations
Unique stocks: 1,457
Date range: 2010-02 to 2023-12
```

11.3.2 Loading Company Fundamentals

We load the company fundamentals data containing book equity, operating profitability, and investment—the characteristics needed for constructing the Fama-French factors.

```
comp_vn = pd.read_sql_query(
    sql="""
        SELECT symbol, datadate, be, op, inv
        FROM comp_vn
    """,
    con=tidy_finance,
    parse_dates={"datadate"}
).dropna()

print(f"Fundamentals: {len(comp_vn):,} firm-year observations")
print(f"Unique firms: {comp_vn['symbol'].nunique():,}")
```

```
Fundamentals: 18,108 firm-year observations
Unique firms: 1,496
```

11.3.3 Constructing Sorting Variables

Following Fama-French conventions, we construct the sorting variables with careful attention to timing. The key principles are:

1. **Size (June Market Cap)**: We use market capitalization at the end of June of year t to sort stocks into size groups. This ensures we capture the firm's size at the moment of portfolio formation.
2. **Book-to-Market Ratio**: We use book equity from fiscal year $t-1$ divided by market equity at the end of December $t-1$. This creates a six-month gap between the accounting data and portfolio formation, ensuring the information was publicly available.
3. **Portfolio Formation Date**: Portfolios are formed on July 1st and held for twelve months until the following June.

```python
def construct_sorting_variables(prices_monthly, comp_vn):
    """
    Construct sorting variables following Fama-French methodology.

    Parameters
    ----------
    prices_monthly : pd.DataFrame
        Monthly stock returns with market cap
    comp_vn : pd.DataFrame
        Company fundamentals with book equity, profitability, investment

    Returns
    -------
    pd.DataFrame
        Sorting variables aligned with July 1st formation dates
    """

    # 1. Size: June market capitalization
    # Portfolio formation is July 1st, so we use June market cap
    size = (prices_monthly
        .query("date.dt.month == 6")
        .assign(
            sorting_date=lambda x: x["date"] + pd.offsets.MonthBegin(1)
        )
        [["symbol", "sorting_date", "mktcap"]]
        .rename(columns={"mktcap": "size"})
    )

    print(f"Size observations: {len(size):,}")

    # 2. Market Equity: December market cap for B/M calculation
    # December t-1 market cap is used with fiscal year t-1 book equity
    # This is then used for July t portfolio formation
    market_equity = (prices_monthly
        .query("date.dt.month == 12")
        .assign(
            # December year t-1 maps to July year t formation
            sorting_date=lambda x: x["date"] + pd.offsets.MonthBegin(7)
        )
        [["symbol", "sorting_date", "mktcap"]]
        .rename(columns={"mktcap": "me"})
    )
```

```
    print(f"Market equity observations: {len(market_equity):,}")

    # 3. Book-to-Market and other characteristics
    # Fiscal year t-1 data is used for July t portfolio formation
    book_to_market = (comp_vn
        .assign(
            # Fiscal year-end + 6 months = July formation
            sorting_date=lambda x: pd.to_datetime(
                (x["datadate"].dt.year + 1).astype(str) + "-07-01"
            )
        )
        .merge(market_equity, on=["symbol", "sorting_date"], how="inner")
        .assign(
            # Scale book equity to match market equity units
            # BE is in VND, ME is in millions VND
            bm=lambda x: x["be"] / (x["me"] * 1e9)
        )
        [["symbol", "sorting_date", "me", "bm", "op", "inv"]]
    )

    print(f"Book-to-market observations: {len(book_to_market):,}")

    # 4. Merge size with characteristics
    sorting_variables = (size
        .merge(book_to_market, on=["symbol", "sorting_date"], how="inner")
        .dropna()
        .drop_duplicates(subset=["symbol", "sorting_date"])
    )

    return sorting_variables

sorting_variables = construct_sorting_variables(prices_monthly, comp_vn)

print(f"\nFinal sorting variables: {len(sorting_variables):,} stock-years")
print(f"Sorting date range: {sorting_variables['sorting_date'].min():%Y-%m} to {sorting_varia
```

```
Size observations: 13,756
Market equity observations: 14,286
Book-to-market observations: 13,389

Final sorting variables: 12,046 stock-years
```

```
Sorting date range: 2011-07 to 2023-07
```

11.3.4 Validating Sorting Variables

Before proceeding, we validate that our sorting variables have reasonable distributions. The book-to-market ratio should center around 1.0 for a typical market, though emerging markets may differ.

```
print("Sorting Variable Summary Statistics:")
print(sorting_variables[["size", "bm", "op", "inv"]].describe().round(4))

# Check for extreme values that might indicate data issues
print(f"\nB/M Median: {sorting_variables['bm'].median():.4f}")
print(f"B/M 1st percentile: {sorting_variables['bm'].quantile(0.01):.4f}")
print(f"B/M 99th percentile: {sorting_variables['bm'].quantile(0.99):.4f}")
```

```
Sorting Variable Summary Statistics:
              size          bm          op         inv
count   12046.0000  12046.0000  12046.0000  12046.0000
mean     2225.5648      1.7033      0.1852      0.1322
std     14680.4225      3.8683      0.2782      2.5410
min         0.4864      0.0014     -0.7529     -0.9569
25%        62.7556      0.7595      0.0309     -0.0495
50%       182.6410      1.1849      0.1367      0.0342
75%       641.8896      1.8853      0.2952      0.1582
max    426020.9817    272.1893      1.4256    261.3355

B/M Median: 1.1849
B/M 1st percentile: 0.1710
B/M 99th percentile: 8.0262
```

11.3.5 Handling Outliers

Extreme values in sorting characteristics can distort portfolio assignments and factor returns. We apply winsorization to limit the influence of outliers while preserving the general ranking of stocks.

```
# Check BEFORE winsorization
print("BEFORE Winsorization:")
print(sorting_variables[["size", "bm", "op", "inv"]].describe().round(4))
```

```python
# Apply winsorization
def winsorize_characteristics(df, columns, limits=(0.01, 0.99)):
    """
    Apply winsorization using pandas clip.
    """
    df = df.copy()
    for col in columns:
        if col in df.columns:
            lower = df[col].quantile(limits[0])
            upper = df[col].quantile(limits[1])
            df[col] = df[col].clip(lower=lower, upper=upper)
            print(f"  {col}: clipped to [{lower:.4f}, {upper:.4f}]")
    return df

sorting_variables = winsorize_characteristics(
    sorting_variables,
    columns=["bm", "op", "inv"],  # Don't winsorize size
    limits=(0.01, 0.99)
)

# Check AFTER winsorization
print("\nAFTER Winsorization:")
print(sorting_variables[["size", "bm", "op", "inv"]].describe().round(4))
```

```
BEFORE Winsorization:
              size          bm          op         inv
count   12046.0000  12046.0000  12046.0000  12046.0000
mean     2225.5648      1.7033      0.1852      0.1322
std     14680.4225      3.8683      0.2782      2.5410
min         0.4864      0.0014     -0.7529     -0.9569
25%        62.7556      0.7595      0.0309     -0.0495
50%       182.6410      1.1849      0.1367      0.0342
75%       641.8896      1.8853      0.2952      0.1582
max    426020.9817    272.1893      1.4256    261.3355
  bm: clipped to [0.1710, 8.0262]
  op: clipped to [-0.7319, 1.2192]
  inv: clipped to [-0.3990, 1.5195]

AFTER Winsorization:
              size          bm          op         inv
count   12046.0000  12046.0000  12046.0000  12046.0000
```

```
mean      2225.5648      1.5544      0.1837      0.0894
std      14680.4225      1.2843      0.2705      0.2721
min          0.4864      0.1710     -0.7319     -0.3990
25%         62.7556      0.7595      0.0309     -0.0495
50%        182.6410      1.1849      0.1367      0.0342
75%        641.8896      1.8853      0.2952      0.1582
max     426020.9817      8.0262      1.2192      1.5195
```

11.4 Portfolio Assignment Functions

11.4.1 The Portfolio Assignment Function

We create a flexible function for assigning stocks to portfolios based on quantile breakpoints. This function handles both independent sorts (where breakpoints are computed across all stocks) and dependent sorts (where breakpoints are computed within subgroups).

```python
def assign_portfolio(data, sorting_variable, percentiles):
    """Assign portfolios to a bin according to a sorting variable."""

    # Get the values
    values = data[sorting_variable].dropna()

    if len(values) == 0:
        return pd.Series([np.nan] * len(data), index=data.index)

    # Calculate breakpoints
    breakpoints = values.quantile(percentiles, interpolation="linear")

    # Handle duplicate breakpoints by using unique values
    unique_breakpoints = np.unique(breakpoints)

    # If all values are the same, assign all to portfolio 1
    if len(unique_breakpoints) <= 1:
        return pd.Series([1] * len(data), index=data.index)

    # Set boundaries to -inf and +inf
    unique_breakpoints.iloc[0] = -np.inf
    unique_breakpoints.iloc[unique_breakpoints.size-1] = np.inf

    # Assign to bins
    assigned = pd.cut(
```

```
        data[sorting_variable],
        bins=unique_breakpoints,
        labels=pd.Series(range(1, breakpoints.size)),
        include_lowest=True,
        right=False
    )

    return assigned
```

```
# Check the distribution of characteristics BEFORE portfolio assignment
print("Operating Profitability Distribution:")
print(sorting_variables["op"].describe())
print(f"\nUnique OP values: {sorting_variables['op'].nunique()}")

print("\nInvestment Distribution:")
print(sorting_variables["inv"].describe())
print(f"\nUnique INV values: {sorting_variables['inv'].nunique()}")

# Check breakpoints for a specific date
test_date = sorting_variables["sorting_date"].iloc[0]
test_data = sorting_variables.query("sorting_date == @test_date")

print(f"\nBreakpoints for {test_date}:")
print(f"OP 30th percentile: {test_data['op'].quantile(0.3):.4f}")
print(f"OP 70th percentile: {test_data['op'].quantile(0.7):.4f}")
print(f"INV 30th percentile: {test_data['inv'].quantile(0.3):.4f}")
print(f"INV 70th percentile: {test_data['inv'].quantile(0.7):.4f}")
```

```
Operating Profitability Distribution:
count    12046.000000
mean         0.183738
std          0.270509
min         -0.731888
25%          0.030913
50%          0.136675
75%          0.295185
max          1.219223
Name: op, dtype: float64

Unique OP values: 11804
```

```
Investment Distribution:
count    12046.000000
mean         0.089388
std          0.272147
min         -0.399042
25%         -0.049497
50%          0.034157
75%          0.158155
max          1.519474
Name: inv, dtype: float64

Unique INV values: 11805

Breakpoints for 2019-07-01 00:00:00:
OP 30th percentile: 0.0541
OP 70th percentile: 0.2566
INV 30th percentile: -0.0343
INV 70th percentile: 0.1116
```

11.4.2 Assigning Portfolios for Three-Factor Model

For the three-factor model, we perform independent double sorts on size and book-to-market. Size is split at the median (2 groups), and book-to-market is split at the 30th and 70th percentiles (3 groups), creating 6 portfolios.

```python
def assign_ff3_portfolios(sorting_variables):
    """
    Assign portfolios for Fama-French three-factor model.
    Independent 2x3 sort on size and book-to-market.
    """
    df = sorting_variables.copy()

    # Independent size sort (median split)
    df["portfolio_size"] = df.groupby("sorting_date")["size"].transform(
        lambda x: pd.qcut(x, q=[0, 0.5, 1], labels=[1, 2], duplicates='drop')
    )

    # Independent B/M sort (30/70 split)
    df["portfolio_bm"] = df.groupby("sorting_date")["bm"].transform(
        lambda x: pd.qcut(x, q=[0, 0.3, 0.7, 1], labels=[1, 2, 3], duplicates='drop')
    )
```

```
    return df

# Assign portfolios
portfolios_ff3 = assign_ff3_portfolios(sorting_variables)

# Validate
print("FF3 Book-to-Market by Portfolio (should be INCREASING):")
print(portfolios_ff3.groupby("portfolio_bm", observed=True)["bm"].median().round(4))


print("Three-Factor Portfolio Assignments:")
print(portfolios_ff3[["symbol", "sorting_date", "portfolio_size", "portfolio_bm"]].head(10))
```

```
FF3 Book-to-Market by Portfolio (should be INCREASING):
portfolio_bm
1    0.5836
2    1.1891
3    2.4552
Name: bm, dtype: float64
Three-Factor Portfolio Assignments:
  symbol sorting_date portfolio_size portfolio_bm
0    A32   2019-07-01              1            2
1    A32   2020-07-01              1            2
2    A32   2021-07-01              1            2
3    A32   2022-07-01              1            3
4    A32   2023-07-01              1            2
5    AAA   2011-07-01              2            2
6    AAA   2012-07-01              2            3
7    AAA   2013-07-01              2            3
8    AAA   2014-07-01              2            2
9    AAA   2015-07-01              2            3
```

11.4.3 Validating Portfolio Assignments

We verify that the portfolio assignments create the expected 2×3 grid with reasonable stock counts in each cell.

```
# Check portfolio distribution for most recent year
latest_date = portfolios_ff3["sorting_date"].max()

portfolio_counts = (portfolios_ff3
```

```
    .query("sorting_date == @latest_date")
    .groupby(["portfolio_size", "portfolio_bm"], observed=True)
    .size()
    .unstack(fill_value=0)
)

print(f"Portfolio Counts for {latest_date:%Y-%m}:")
print(portfolio_counts)

# Verify characteristic monotonicity
print("\nBook-to-Market by Portfolio (should be increasing):")
print(portfolios_ff3.groupby("portfolio_bm", observed=True)["bm"].median().round(4))
```

```
Portfolio Counts for 2023-07:
portfolio_bm      1    2    3
portfolio_size
1               113  271  263
2               275  246  125

Book-to-Market by Portfolio (should be increasing):
portfolio_bm
1    0.5836
2    1.1891
3    2.4552
Name: bm, dtype: float64
```

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_ff3
    .query("symbol == 'A32' & sorting_date >= '2022-01-01' & sorting_date <= '2023-12-31'")
    .sort_values("sorting_date")
    [['symbol', 'sorting_date', 'portfolio_size', 'portfolio_bm']]
)
print("\nTemporal Persistence Check (Symbol A32):")
print(persistence_check.head(15))
```

```
Temporal Persistence Check (Symbol A32):
  symbol sorting_date portfolio_size portfolio_bm
```

```
3     A32    2022-07-01              1             3
4     A32    2023-07-01              1             2
```

11.5 Fama-French Three-Factor Model (Monthly)

11.5.1 Merging Portfolios with Returns

We merge the portfolio assignments with monthly returns. The key insight is that portfolios formed in July of year t are held through June of year $t+1$. We implement this by computing a `sorting_date` for each monthly return observation.

```python
def merge_portfolios_with_returns(prices_monthly, portfolio_assignments):
    """
    Merge portfolio assignments with monthly returns.

    Portfolios formed in July t are held through June t+1.

    Parameters
    ----------
    prices_monthly : pd.DataFrame
        Monthly stock returns
    portfolio_assignments : pd.DataFrame
        Portfolio assignments with sorting_date

    Returns
    -------
    pd.DataFrame
        Returns merged with portfolio assignments
    """
    portfolios = (prices_monthly
        .assign(
            # Map each return month to its portfolio formation date
            sorting_date=lambda x: pd.to_datetime(
                np.where(
                    x["date"].dt.month <= 6,
                    (x["date"].dt.year - 1).astype(str) + "-07-01",
                    x["date"].dt.year.astype(str) + "-07-01"
                )
            )
        )
        .merge(
```

```
            portfolio_assignments,
            on=["symbol", "sorting_date"],
            how="inner"
        )
    )

    return portfolios

portfolios_monthly_ff3 = merge_portfolios_with_returns(
    prices_monthly,
    portfolios_ff3[["symbol", "sorting_date", "portfolio_size", "portfolio_bm"]]
)


print(f"Merged observations: {len(portfolios_monthly_ff3):,}")
```

```
Merged observations: 136,444
```

11.5.2 Computing Value-Weighted Portfolio Returns

We compute value-weighted returns for each of the six portfolios. Value-weighting uses lagged market capitalization to avoid look-ahead bias.

```
def compute_portfolio_returns(data, grouping_vars):
    """
    Compute value-weighted portfolio returns.

    Parameters
    ----------
    data : pd.DataFrame
        Returns data with portfolio assignments and mktcap_lag
    grouping_vars : list
        Variables defining portfolio groups

    Returns
    -------
    pd.DataFrame
        Value-weighted returns for each portfolio-date
    """
    portfolio_returns = (data
        .groupby(grouping_vars + ["date"], observed=True)
```

```
        .apply(lambda x: pd.Series({
            "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"]),
            "n_stocks": len(x)
        }))
        .reset_index()
    )

    return portfolio_returns


# Compute portfolio returns
portfolio_returns_ff3 = compute_portfolio_returns(
    portfolios_monthly_ff3,
    ["portfolio_size", "portfolio_bm"]
)

print("Portfolio Returns Summary:")
print(portfolio_returns_ff3.groupby(["portfolio_size", "portfolio_bm"], observed=True)["ret"]
```

```
Portfolio Returns Summary:
                             count    mean     std     min     25%     50%  \
portfolio_size portfolio_bm
1              1             150.0 -0.0052  0.0379 -0.1268 -0.0262 -0.0058
               2             150.0 -0.0025  0.0414 -0.1080 -0.0236 -0.0053
               3             150.0  0.0039  0.0601 -0.1612 -0.0269 -0.0004
2              1             150.0 -0.0124  0.0594 -0.2222 -0.0449 -0.0107
               2             150.0 -0.0021  0.0671 -0.1701 -0.0403 -0.0046
               3             150.0  0.0024  0.0879 -0.2359 -0.0527 -0.0012

                                 75%     max
portfolio_size portfolio_bm
1              1             0.0161  0.0849
               2             0.0180  0.1195
               3             0.0314  0.2015
2              1             0.0210  0.1741
               2             0.0317  0.1770
               3             0.0437  0.2124
```

11.5.3 Constructing SMB and HML Factors

We now construct the SMB and HML factors from the portfolio returns.

SMB (Small Minus Big): Average return of three small portfolios minus average return of three big portfolios.

HML (High Minus Low): Average return of two high B/M portfolios minus average return of two low B/M portfolios.

```
def construct_ff3_factors(portfolio_returns):
    """
    Construct Fama-French three factors from portfolio returns.

    Parameters
    ----------
    portfolio_returns : pd.DataFrame
        Value-weighted returns for 2x3 portfolios

    Returns
    -------
    pd.DataFrame
        Monthly SMB and HML factors
    """
    factors = (portfolio_returns
        .groupby("date")
        .apply(lambda x: pd.Series({
            # SMB: Small minus Big (average across B/M groups)
            "smb": (
                x.loc[x["portfolio_size"] == 1, "ret"].mean() -
                x.loc[x["portfolio_size"] == 2, "ret"].mean()
            ),
            # HML: High minus Low B/M (average across size groups)
            "hml": (
                x.loc[x["portfolio_bm"] == 3, "ret"].mean() -
                x.loc[x["portfolio_bm"] == 1, "ret"].mean()
            )
        }))
        .reset_index()
    )

    return factors

factors_smb_hml = construct_ff3_factors(portfolio_returns_ff3)

print("SMB and HML Factors:")
print(factors_smb_hml.head(10))
```

```
SMB and HML Factors:
        date       smb       hml
0 2011-07-31 -0.007768  0.002754
1 2011-08-31 -0.067309  0.011474
2 2011-09-30  0.014884  0.022854
3 2011-10-31 -0.003743  0.001631
4 2011-11-30  0.063234  0.009103
5 2011-12-31  0.014571  0.015280
6 2012-01-31 -0.026080  0.009672
7 2012-02-29 -0.035721  0.005474
8 2012-03-31 -0.002344  0.032477
9 2012-04-30 -0.033391  0.074191
```

11.5.4 Computing the Market Factor

The market factor is the value-weighted return of all stocks minus the risk-free rate. We compute this independently from the sorted portfolios.

```python
def compute_market_factor(prices_monthly):
    """
    Compute value-weighted market excess return.

    Parameters
    ----------
    prices_monthly : pd.DataFrame
        Monthly stock returns with mktcap_lag

    Returns
    -------
    pd.DataFrame
        Monthly market excess return
    """
    market_factor = (prices_monthly
        .groupby("date")
        .apply(lambda x: pd.Series({
            "mkt_excess": np.average(x["ret_excess"], weights=x["mktcap_lag"]),
            "n_stocks": len(x)
        }), include_groups=False)
        .reset_index()
    )

    return market_factor
```

```
market_factor = compute_market_factor(prices_monthly)

print("Market Factor Summary:")
print(market_factor["mkt_excess"].describe().round(4))
```

```
Market Factor Summary:
count    167.0000
mean      -0.0123
std        0.0595
min       -0.2149
25%       -0.0394
50%       -0.0106
75%        0.0200
max        0.1677
Name: mkt_excess, dtype: float64
```

11.5.5 Combining Three Factors

We combine SMB, HML, and the market factor into the complete three-factor dataset.

```
factors_ff3_monthly = (factors_smb_hml
    .merge(market_factor[["date", "mkt_excess"]], on="date", how="inner")
)

# Add risk-free rate for completeness
rf_monthly = (prices_monthly
    .groupby("date")["risk_free"]
    .first()
    .reset_index()
)

factors_ff3_monthly = factors_ff3_monthly.merge(rf_monthly, on="date", how="left")

print("Fama-French Three Factors (Monthly):")
print(factors_ff3_monthly.head(10))

print("\nFactor Summary Statistics:")
print(factors_ff3_monthly[["mkt_excess", "smb", "hml"]].describe().round(4))
```

```
Fama-French Three Factors (Monthly):
```

```
        date       smb       hml  mkt_excess  risk_free
0 2011-07-31 -0.007768  0.002754   -0.078748   0.003333
1 2011-08-31 -0.067309  0.011474    0.029906   0.003333
2 2011-09-30  0.014884  0.022854   -0.002173   0.003333
3 2011-10-31 -0.003743  0.001631   -0.014005   0.003333
4 2011-11-30  0.063234  0.009103   -0.179410   0.003333
5 2011-12-31  0.014571  0.015280   -0.094802   0.003333
6 2012-01-31 -0.026080  0.009672    0.081273   0.003333
7 2012-02-29 -0.035721  0.005474    0.069655   0.003333
8 2012-03-31 -0.002344  0.032477    0.029005   0.003333
9 2012-04-30 -0.033391  0.074191    0.048791   0.003333

Factor Summary Statistics:
       mkt_excess       smb       hml
count    150.0000  150.0000  150.0000
mean      -0.0101    0.0027    0.0120
std        0.0586    0.0420    0.0535
min       -0.2149   -0.1599   -0.1284
25%       -0.0380   -0.0175   -0.0160
50%       -0.0095    0.0070    0.0043
75%        0.0214    0.0261    0.0340
max        0.1677    0.1175    0.1618
```

11.5.6 Saving Three-Factor Data

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

print("Three-factor monthly data saved to database.")
```

```
Three-factor monthly data saved to database.
```

11.6 Fama-French Five-Factor Model (Monthly)

11.6.1 Portfolio Assignments with Dependent Sorts

For the five-factor model, we use dependent sorts: size is sorted independently, but profitability and investment are sorted within size groups. This controls for the correlation between size and these characteristics.

```
def assign_ff5_portfolios(sorting_variables):
    """
    Assign portfolios for Fama-French five-factor model.
    """
    df = sorting_variables.copy()

    # Independent size sort
    df["portfolio_size"] = df.groupby("sorting_date")["size"].transform(
        lambda x: pd.qcut(x, q=[0, 0.5, 1], labels=[1, 2], duplicates='drop')
    )

    # Dependent sorts within size groups
    df["portfolio_bm"] = df.groupby(["sorting_date", "portfolio_size"])["bm"].transform(
        lambda x: pd.qcut(x, q=[0, 0.3, 0.7, 1], labels=[1, 2, 3], duplicates='drop')
    )

    df["portfolio_op"] = df.groupby(["sorting_date", "portfolio_size"])["op"].transform(
        lambda x: pd.qcut(x, q=[0, 0.3, 0.7, 1], labels=[1, 2, 3], duplicates='drop')
    )

    df["portfolio_inv"] = df.groupby(["sorting_date", "portfolio_size"])["inv"].transform(
        lambda x: pd.qcut(x, q=[0, 0.3, 0.7, 1], labels=[1, 2, 3], duplicates='drop')
    )

    return df

# Run
portfolios_ff5 = assign_ff5_portfolios(sorting_variables)
```

11.6.2 Validating Five-Factor Portfolios

```python
# Check characteristic monotonicity for each dimension
print("Profitability by Portfolio (should be increasing):")
print(portfolios_ff5.groupby("portfolio_op", observed=True)["op"].median().round(4))

print("\nInvestment by Portfolio (should be increasing):")
print(portfolios_ff5.groupby("portfolio_inv", observed=True)["inv"].median().round(4))

# Check portfolio counts
print("\nStocks per Size/Profitability Bin:")
print(portfolios_ff5.groupby(["portfolio_size", "portfolio_op"], observed=True).size().unsta

# Check number of unique firms per year
(portfolios_ff5
 .groupby("sorting_date")["symbol"]
 .nunique())
```

```
Profitability by Portfolio (should be increasing):
portfolio_op
1   -0.0053
2    0.1366
3    0.4098
Name: op, dtype: float64

Investment by Portfolio (should be increasing):
portfolio_inv
1   -0.1012
2    0.0329
3    0.2568
Name: inv, dtype: float64

Stocks per Size/Profitability Bin:
portfolio_op       1     2     3
portfolio_size
1               1812  2403  1811
2               1811  2401  1808

sorting_date
2011-07-01     556
2012-07-01     632
2013-07-01     643
2014-07-01     650
```

```
2015-07-01     669
2016-07-01     737
2017-07-01     842
2018-07-01    1073
2019-07-01    1183
2020-07-01    1224
2021-07-01    1258
2022-07-01    1286
2023-07-01    1293
Name: symbol, dtype: int64
```

11.6.3 Merging and Computing Portfolio Returns

```
# Merge with returns
portfolios_monthly_ff5 = merge_portfolios_with_returns(
    prices_monthly,
    portfolios_ff5[["symbol", "sorting_date", "portfolio_size",
                    "portfolio_bm", "portfolio_op", "portfolio_inv"]]
)

print(f"Five-factor merged observations: {len(portfolios_monthly_ff5):,}")
```

```
Five-factor merged observations: 136,444
```

11.6.4 Constructing All Five Factors

We construct each factor from the appropriate portfolio sorts.

```
def construct_ff5_factors(portfolios_monthly):
    """
    Construct Fama-French five factors from portfolio data.

    Parameters
    ----------
    portfolios_monthly : pd.DataFrame
        Monthly returns with all portfolio assignments

    Returns
    -------
```

```
    pd.DataFrame
        Monthly five-factor returns
    """

    # HML: Value factor from B/M sorts
    portfolios_bm = (portfolios_monthly
        .groupby(["portfolio_size", "portfolio_bm", "date"], observed=True)
        .apply(lambda x: pd.Series({
            "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"])
        }))
        .reset_index()
    )

    factors_hml = (portfolios_bm
        .groupby("date")
        .apply(lambda x: pd.Series({
            "hml": (x.loc[x["portfolio_bm"] == 3, "ret"].mean() -
                    x.loc[x["portfolio_bm"] == 1, "ret"].mean())
        }))
        .reset_index()
    )

    # RMW: Profitability factor from OP sorts
    portfolios_op = (portfolios_monthly
        .groupby(["portfolio_size", "portfolio_op", "date"], observed=True)
        .apply(lambda x: pd.Series({
            "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"])
        }))
        .reset_index()
    )

    factors_rmw = (portfolios_op
        .groupby("date")
        .apply(lambda x: pd.Series({
            "rmw": (x.loc[x["portfolio_op"] == 3, "ret"].mean() -
                    x.loc[x["portfolio_op"] == 1, "ret"].mean())
        }))
        .reset_index()
    )

    # CMA: Investment factor from INV sorts
    # Note: CMA is Conservative minus Aggressive (low inv - high inv)
```

```
    portfolios_inv = (portfolios_monthly
        .groupby(["portfolio_size", "portfolio_inv", "date"], observed=True)
        .apply(lambda x: pd.Series({
            "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"])
        }))
        .reset_index()
    )

    factors_cma = (portfolios_inv
        .groupby("date")
        .apply(lambda x: pd.Series({
            "cma": (x.loc[x["portfolio_inv"] == 1, "ret"].mean() -
                    x.loc[x["portfolio_inv"] == 3, "ret"].mean())
        }))
        .reset_index()
    )

    # SMB: Size factor (average across all characteristic portfolios)
    all_portfolios = pd.concat([portfolios_bm, portfolios_op, portfolios_inv])

    factors_smb = (all_portfolios
        .groupby("date")
        .apply(lambda x: pd.Series({
            "smb": (x.loc[x["portfolio_size"] == 1, "ret"].mean() -
                    x.loc[x["portfolio_size"] == 2, "ret"].mean())
        }))
        .reset_index()
    )

    # Combine all factors
    factors = (factors_smb
        .merge(factors_hml, on="date", how="outer")
        .merge(factors_rmw, on="date", how="outer")
        .merge(factors_cma, on="date", how="outer")
    )

    return factors

factors_ff5 = construct_ff5_factors(portfolios_monthly_ff5)

# Add market factor
factors_ff5_monthly = (factors_ff5
```

```
    .merge(market_factor[["date", "mkt_excess"]], on="date", how="inner")
    .merge(rf_monthly, on="date", how="left")
)

print("Fama-French Five Factors (Monthly):")
print(factors_ff5_monthly.head(10))

print("\nFactor Summary Statistics:")
print(factors_ff5_monthly[["mkt_excess", "smb", "hml", "rmw", "cma"]].describe().round(4))
```

```
Fama-French Five Factors (Monthly):
        date       smb       hml       rmw       cma  mkt_excess  risk_free
0 2011-07-31 -0.015907 -0.002812  0.060525  0.045291   -0.078748   0.003333
1 2011-08-31 -0.061842  0.006189 -0.022700 -0.023177    0.029906   0.003333
2 2011-09-30  0.014387  0.024301 -0.006005  0.003588   -0.002173   0.003333
3 2011-10-31 -0.006958 -0.006940  0.026694  0.003649   -0.014005   0.003333
4 2011-11-30  0.074369  0.015617 -0.058766  0.044214   -0.179410   0.003333
5 2011-12-31  0.006687  0.022494  0.062655  0.052444   -0.094802   0.003333
6 2012-01-31 -0.016254  0.010513 -0.042191 -0.067170    0.081273   0.003333
7 2012-02-29 -0.026606  0.024465 -0.030849 -0.036383    0.069655   0.003333
8 2012-03-31  0.005096  0.050930 -0.018441  0.043488    0.029005   0.003333
9 2012-04-30  0.000712  0.058214 -0.061434  0.009233    0.048791   0.003333

Factor Summary Statistics:
       mkt_excess       smb       hml       rmw       cma
count    150.0000  150.0000  150.0000  150.0000  150.0000
mean      -0.0101    0.0077    0.0115   -0.0047    0.0083
std        0.0586    0.0419    0.0518    0.0477    0.0335
min       -0.2149   -0.1522   -0.1283   -0.2126   -0.0814
25%       -0.0380   -0.0137   -0.0126   -0.0308   -0.0131
50%       -0.0095    0.0104    0.0046    0.0010    0.0067
75%        0.0214    0.0316    0.0323    0.0178    0.0289
max        0.1677    0.1284    0.1510    0.1297    0.1331
```

11.6.5 Factor Correlations

We examine correlations between factors, which should generally be low for the factors to capture distinct sources of risk.

```
print("Factor Correlation Matrix:")
correlation_matrix = factors_ff5_monthly[["mkt_excess", "smb", "hml", "rmw", "cma"]].corr()
print(correlation_matrix.round(3))
```

```
Factor Correlation Matrix:
            mkt_excess    smb    hml    rmw    cma
mkt_excess       1.000 -0.712  0.230 -0.006 -0.104
smb             -0.712  1.000  0.256 -0.373  0.246
hml              0.230  0.256  1.000 -0.694  0.479
rmw             -0.006 -0.373 -0.694  1.000 -0.352
cma             -0.104  0.246  0.479 -0.352  1.000
```

11.6.6 Saving Five-Factor Data

```
factors_ff5_monthly.to_sql(
    name="factors_ff5_monthly",
    con=tidy_finance,
    if_exists="replace",
    index=False
)

print("Five-factor monthly data saved to database.")
```

```
Five-factor monthly data saved to database.
```

11.7 Daily Fama-French Factors

11.7.1 Motivation for Daily Factors

Daily factors are essential for several applications:

1. **Daily beta estimation**: CAPM regressions using daily data require daily market excess returns.
2. **Event studies**: Measuring abnormal returns around corporate events requires daily factor adjustments.
3. **High-frequency research**: Market microstructure studies need daily or intraday factor data.

The construction methodology mirrors the monthly approach, but we compute portfolio returns at daily frequency while maintaining the same annual portfolio formation dates.

11.7.2 Loading Daily Returns

```
# Load daily price data
prices_daily = pd.read_sql_query(
    sql="""
        SELECT symbol, date, ret_excess
        FROM prices_daily
    """,
    con=tidy_finance,
    parse_dates={"date"}
)

print(f"Daily returns: {len(prices_daily):,} observations")
print(f"Date range: {prices_daily['date'].min():%Y-%m-%d} to {prices_daily['date'].max():%Y-%
```

```
Daily returns: 3,462,157 observations
Date range: 2010-01-05 to 2023-12-29
```

11.7.3 Adding Market Cap for Daily Weighting

For value-weighted daily returns, we need market capitalization. We use the most recent monthly market cap as the weight for daily returns within that month.

```
# Get monthly market cap to use as weights for daily returns
mktcap_monthly = (prices_monthly
    [["symbol", "date", "mktcap_lag"]]
    .assign(year_month=lambda x: x["date"].dt.to_period("M"))
)

# Add year_month to daily data for merging
prices_daily = (prices_daily
    .assign(year_month=lambda x: x["date"].dt.to_period("M"))
    .merge(
        mktcap_monthly[["symbol", "year_month", "mktcap_lag"]],
        on=["symbol", "year_month"],
        how="left"
    )
    .dropna(subset=["ret_excess", "mktcap_lag"])
)

print(f"Daily returns with weights: {len(prices_daily):,} observations")
```

```
Daily returns with weights: 3,443,815 observations
```

11.7.4 Merging Daily Returns with Portfolios

We use the same portfolio assignments (formed annually in July) for daily returns.

```
# Step 1: Ensure portfolios_ff3 has correct format
portfolios_ff3_clean = portfolios_ff3[["symbol", "sorting_date", "portfolio_size", "portfolio
portfolios_ff3_clean["sorting_date"] = pd.to_datetime(portfolios_ff3_clean["sorting_date"])

print("Portfolio sorting dates:")
print(portfolios_ff3_clean["sorting_date"].unique()[:5])

# Step 2: Create sorting_date for daily data
prices_daily_with_sort = prices_daily.copy()
prices_daily_with_sort["sorting_date"] = prices_daily_with_sort["date"].apply(
    lambda x: pd.Timestamp(f"{x.year}-07-01") if x.month > 6 else pd.Timestamp(f"{x.year - 1
)

print("\nDaily sorting dates:")
print(prices_daily_with_sort["sorting_date"].unique()[:5])

# Step 3: Merge
portfolios_daily_ff3 = prices_daily_with_sort.merge(
    portfolios_ff3_clean,
    on=["symbol", "sorting_date"],
    how="inner"
)

print(f"\nMerged daily observations: {len(portfolios_daily_ff3):,}")
print(f"Unique dates: {portfolios_daily_ff3['date'].nunique():,}")

# Step 4: Verify portfolio distribution
print("\nPortfolio distribution in daily data:")
print(portfolios_daily_ff3.groupby(["portfolio_size", "portfolio_bm"], observed=True).size()
```

```
Portfolio sorting dates:
<DatetimeArray>
['2019-07-01 00:00:00', '2020-07-01 00:00:00', '2021-07-01 00:00:00',
 '2022-07-01 00:00:00', '2023-07-01 00:00:00']
Length: 5, dtype: datetime64[us]
```

```
Daily sorting dates:
<DatetimeArray>
['2018-07-01 00:00:00', '2019-07-01 00:00:00', '2020-07-01 00:00:00',
 '2021-07-01 00:00:00', '2022-07-01 00:00:00']
Length: 5, dtype: datetime64[us]

Merged daily observations: 2,843,570
Unique dates: 3,126

Portfolio distribution in daily data:
portfolio_bm         1       2       3
portfolio_size
1               218040  585561  617327
2               636152  552114  234376
```

```python
# Diagnostic: Check the daily portfolio merge
print("="*50)
print("DIAGNOSTIC: Daily Portfolio Merge")
print("="*50)

print(f"\nDaily prices rows: {len(prices_daily):,}")
print(f"Daily FF3 portfolios rows: {len(portfolios_daily_ff3):,}")
print(f"Match rate: {len(portfolios_daily_ff3)/len(prices_daily)*100:.1f}%")

# Check portfolio distribution in daily data
print("\nDaily portfolio distribution:")
print(portfolios_daily_ff3.groupby(["portfolio_size", "portfolio_bm"], observed=True).size()

# Check a specific date
test_date = portfolios_daily_ff3["date"].iloc[1000]
print(f"\nSample date: {test_date}")
print(portfolios_daily_ff3.query("date == @test_date").groupby(["portfolio_size", "portfolio_
```

```
==================================================
DIAGNOSTIC: Daily Portfolio Merge
==================================================

Daily prices rows: 3,443,815
Daily FF3 portfolios rows: 2,843,570
Match rate: 82.6%
```

```
Daily portfolio distribution:
portfolio_bm         1       2       3
portfolio_size
1               218040  585561  617327
2               636152  552114  234376

Sample date: 2023-06-28 00:00:00
portfolio_bm      1    2    3
portfolio_size
1                 93  232  312
2                291  280   67
```

11.7.5 Computing Daily Three Factors

```python
def compute_daily_ff3_factors(portfolios_daily):
    """
    Compute daily Fama-French three factors.

    Parameters
    ----------
    portfolios_daily : pd.DataFrame
        Daily returns with portfolio assignments

    Returns
    -------
    pd.DataFrame
        Daily SMB and HML factors
    """
    # Compute daily portfolio returns
    portfolio_returns = (portfolios_daily
        .groupby(["portfolio_size", "portfolio_bm", "date"], observed=True)
        .apply(lambda x: pd.Series({
            "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"])
        }))
        .reset_index()
    )

    # Compute factors
    factors = (portfolio_returns
        .groupby("date")
        .apply(lambda x: pd.Series({
```

```
            "smb": (x.loc[x["portfolio_size"] == 1, "ret"].mean() -
                    x.loc[x["portfolio_size"] == 2, "ret"].mean()),
            "hml": (x.loc[x["portfolio_bm"] == 3, "ret"].mean() -
                    x.loc[x["portfolio_bm"] == 1, "ret"].mean())
        }))
        .reset_index()
    )

    return factors

factors_daily_smb_hml = compute_daily_ff3_factors(portfolios_daily_ff3)

print(f"Daily factor observations: {len(factors_daily_smb_hml):,}")
print(factors_daily_smb_hml.head(10))
```

```
Daily factor observations: 3,126
        date       smb       hml
0 2011-07-01  0.008587  0.000967
1 2011-07-04  0.005099 -0.001099
2 2011-07-05 -0.009088  0.010152
3 2011-07-06  0.004875 -0.003918
4 2011-07-07 -0.011239 -0.000584
5 2011-07-08  0.005636 -0.008003
6 2011-07-11  0.003940  0.006172
7 2011-07-12  0.003205  0.006543
8 2011-07-13 -0.000097 -0.001134
9 2011-07-14 -0.001248  0.001669
```

11.7.6 Computing Daily Market Factor

```
def compute_daily_market_factor(prices_daily):
    """
    Compute daily value-weighted market excess return.

    Parameters
    ----------
    prices_daily : pd.DataFrame
        Daily returns with mktcap_lag

    Returns
```

```
    -------
    pd.DataFrame
        Daily market excess return
    """
    market_daily = (prices_daily
        .groupby("date")
        .apply(lambda x: pd.Series({
            "mkt_excess": np.average(x["ret_excess"], weights=x["mktcap_lag"])
        }))
        .reset_index()
    )

    return market_daily

market_factor_daily = compute_daily_market_factor(prices_daily)

print(f"Daily market factor: {len(market_factor_daily):,} days")
```

```
Daily market factor: 3,474 days
```

11.7.7 Combining Daily Three Factors

```
factors_ff3_daily = (factors_daily_smb_hml
    .merge(market_factor_daily, on="date", how="inner")
)

# Add risk-free rate (use monthly rate / 21 as daily approximation, or load actual daily rate
factors_ff3_daily["risk_free"] = 0.04 / 252  # Approximate daily risk-free

print("Daily Fama-French Three Factors:")
print(factors_ff3_daily.head(10))

print("\nDaily Factor Summary Statistics:")
print(factors_ff3_daily[["mkt_excess", "smb", "hml"]].describe().round(6))
```

```
Daily Fama-French Three Factors:
        date       smb       hml  mkt_excess  risk_free
0 2011-07-01  0.008587  0.000967   -0.019862   0.000159
1 2011-07-04  0.005099 -0.001099   -0.000633   0.000159
2 2011-07-05 -0.009088  0.010152    0.013314   0.000159
```

```
3 2011-07-06  0.004875 -0.003918   -0.008045   0.000159
4 2011-07-07 -0.011239 -0.000584    0.003391   0.000159
5 2011-07-08  0.005636 -0.008003    0.000218   0.000159
6 2011-07-11  0.003940  0.006172   -0.013393   0.000159
7 2011-07-12  0.003205  0.006543   -0.017505   0.000159
8 2011-07-13 -0.000097 -0.001134    0.000767   0.000159
9 2011-07-14 -0.001248  0.001669   -0.000695   0.000159

Daily Factor Summary Statistics:
        mkt_excess           smb           hml
count  3126.000000  3126.000000  3126.000000
mean     -0.000479     0.000236     0.000594
std       0.011269     0.008488     0.008585
min      -0.070268    -0.032671    -0.039418
25%      -0.005074    -0.004882    -0.003941
50%       0.000350    -0.000106     0.000522
75%       0.005531     0.004307     0.005233
max       0.043386     0.042686     0.083889
```

11.7.8 Computing Daily Five Factors

```
# Step 1: Clean portfolios
portfolios_ff5_clean = portfolios_ff5[["symbol", "sorting_date", "portfolio_size",
                                       "portfolio_bm", "portfolio_op", "portfolio_inv"]].co
portfolios_ff5_clean["sorting_date"] = pd.to_datetime(portfolios_ff5_clean["sorting_date"])

# Step 2: Merge with daily prices
portfolios_daily_ff5 = prices_daily_with_sort.merge(
    portfolios_ff5_clean,
    on=["symbol", "sorting_date"],
    how="inner"
)

print(f"FF5 Daily merged observations: {len(portfolios_daily_ff5):,}")
```

```
FF5 Daily merged observations: 2,843,570
```

```
def compute_daily_ff5_factors(portfolios_daily):
    """Compute daily Fama-French five factors."""
```

```
    # HML from B/M sorts
    portfolios_bm = (portfolios_daily
        .groupby(["portfolio_size", "portfolio_bm", "date"], observed=True)
        .apply(lambda x: pd.Series({
            "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"])
        }))
        .reset_index()
    )

    factors_hml = (portfolios_bm
        .groupby("date")
        .apply(lambda x: pd.Series({
            "hml": (x.loc[x["portfolio_bm"] == 3, "ret"].mean() -
                    x.loc[x["portfolio_bm"] == 1, "ret"].mean())
        }))
        .reset_index()
    )

    # RMW from OP sorts
    portfolios_op = (portfolios_daily
        .groupby(["portfolio_size", "portfolio_op", "date"], observed=True)
        .apply(lambda x: pd.Series({
            "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"])
        }))
        .reset_index()
    )

    factors_rmw = (portfolios_op
        .groupby("date")
        .apply(lambda x: pd.Series({
            "rmw": (x.loc[x["portfolio_op"] == 3, "ret"].mean() -
                    x.loc[x["portfolio_op"] == 1, "ret"].mean())
        }))
        .reset_index()
    )

    # CMA from INV sorts (note: low minus high)
    portfolios_inv = (portfolios_daily
        .groupby(["portfolio_size", "portfolio_inv", "date"], observed=True)
        .apply(lambda x: pd.Series({
            "ret": np.average(x["ret_excess"], weights=x["mktcap_lag"])
        }))
```

```
        .reset_index()
    )

    factors_cma = (portfolios_inv
        .groupby("date")
        .apply(lambda x: pd.Series({
            "cma": (x.loc[x["portfolio_inv"] == 1, "ret"].mean() -
                    x.loc[x["portfolio_inv"] == 3, "ret"].mean())
        }))
        .reset_index()
    )

    # SMB from all sorts
    all_portfolios = pd.concat([portfolios_bm, portfolios_op, portfolios_inv])

    factors_smb = (all_portfolios
        .groupby("date")
        .apply(lambda x: pd.Series({
            "smb": (x.loc[x["portfolio_size"] == 1, "ret"].mean() -
                    x.loc[x["portfolio_size"] == 2, "ret"].mean())
        }))
        .reset_index()
    )

    # Combine
    factors = (factors_smb
        .merge(factors_hml, on="date", how="outer")
        .merge(factors_rmw, on="date", how="outer")
        .merge(factors_cma, on="date", how="outer")
    )

    return factors

# Compute daily FF5 factors
factors_daily_ff5 = compute_daily_ff5_factors(portfolios_daily_ff5)

# Add market factor
factors_ff5_daily = (factors_daily_ff5
    .merge(market_factor_daily, on="date", how="inner")
)
factors_ff5_daily["risk_free"] = 0.04 / 252
```

```
print("Daily Fama-French Five Factors:")
print(factors_ff5_daily.head(10))

print("\nDaily Five-Factor Summary Statistics:")
print(factors_ff5_daily[["mkt_excess", "smb", "hml", "rmw", "cma"]].describe().round(6))
```

```
Daily Fama-French Five Factors:
        date       smb       hml       rmw       cma  mkt_excess  risk_free
0 2011-07-01  0.006295  0.002515  0.013140  0.007680   -0.019862   0.000159
1 2011-07-04  0.002880 -0.002875  0.006560 -0.004886   -0.000633   0.000159
2 2011-07-05 -0.004260  0.009864 -0.012158 -0.004470    0.013314   0.000159
3 2011-07-06  0.001544 -0.009847  0.012977  0.006286   -0.008045   0.000159
4 2011-07-07 -0.009789 -0.003988 -0.000197 -0.006995    0.003391   0.000159
5 2011-07-08  0.001537 -0.006700  0.010841 -0.007661    0.000218   0.000159
6 2011-07-11  0.005396  0.004747  0.000655  0.013375   -0.013393   0.000159
7 2011-07-12  0.004759  0.007367  0.001989  0.014669   -0.017505   0.000159
8 2011-07-13 -0.000009  0.001110 -0.002052 -0.001633    0.000767   0.000159
9 2011-07-14 -0.001668  0.002916 -0.005427  0.005388   -0.000695   0.000159

Daily Five-Factor Summary Statistics:
        mkt_excess           smb           hml           rmw           cma
count  3126.000000  3126.000000  3126.000000  3126.000000  3126.000000
mean     -0.000479     0.000484     0.000549    -0.000136     0.000413
std       0.011269     0.008033     0.008312     0.008538     0.006756
min      -0.070268    -0.036283    -0.039155    -0.154013    -0.047698
25%      -0.005074    -0.004122    -0.003681    -0.004212    -0.003364
50%       0.000350     0.000105     0.000384     0.000030     0.000162
75%       0.005531     0.004358     0.004819     0.004036     0.003800
max       0.043386     0.060307     0.086269     0.102001     0.089907
```

11.7.9 Saving Daily Factors

```
factors_ff3_daily.to_sql(
    name="factors_ff3_daily",
    con=tidy_finance,
    if_exists="replace",
    index=False
)

factors_ff5_daily.to_sql(
```

```
    name="factors_ff5_daily",
    con=tidy_finance,
    if_exists="replace",
    index=False
)

print("Daily factor data saved to database.")
```

```
Daily factor data saved to database.
```

11.8 Factor Validation and Diagnostics

```
# Verify all tables are in database
print("\n" + "="*50)
print("DATABASE SUMMARY")
print("="*50)

tables = ["factors_ff3_monthly", "factors_ff5_monthly",
          "factors_ff3_daily", "factors_ff5_daily"]

for table in tables:
    df = pd.read_sql_query(f"SELECT COUNT(*) as n FROM {table}", con=tidy_finance)
    print(f"{table}: {df['n'].iloc[0]:,} observations")

# Correlation check: Monthly vs Daily (aggregated)
print("\n" + "="*50)
print("MONTHLY VS DAILY CONSISTENCY CHECK")
print("="*50)

factors_daily_agg = (factors_ff3_daily
    .assign(year_month=lambda x: x["date"].dt.to_period("M"))
    .groupby("year_month")[["mkt_excess", "smb", "hml"]]
    .sum()
    .reset_index()
)

factors_monthly_check = (factors_ff3_monthly
    .assign(year_month=lambda x: x["date"].dt.to_period("M"))
)
```

```
comparison = factors_monthly_check.merge(
    factors_daily_agg, on="year_month", suffixes=("_monthly", "_daily")
)

for factor in ["mkt_excess", "smb", "hml"]:
    corr = comparison[f"{factor}_monthly"].corr(comparison[f"{factor}_daily"])
    print(f"{factor}: Monthly-Daily correlation = {corr:.4f}")
```

```
==================================================
DATABASE SUMMARY
==================================================
factors_ff3_monthly: 150 observations
factors_ff5_monthly: 150 observations
factors_ff3_daily: 3,126 observations
factors_ff5_daily: 3,126 observations

==================================================
MONTHLY VS DAILY CONSISTENCY CHECK
==================================================
mkt_excess: Monthly-Daily correlation = 0.9980
smb: Monthly-Daily correlation = 0.9953
hml: Monthly-Daily correlation = 0.9936
```

11.8.1 Cumulative Factor Returns

We visualize the cumulative performance of each factor to assess whether the factors generate meaningful premiums over time.

```
# Compute cumulative returns
factors_cumulative = (factors_ff5_monthly
    .set_index("date")
    [["mkt_excess", "smb", "hml", "rmw", "cma"]]
    .add(1)
    .cumprod()
)

# Plot
fig, ax = plt.subplots(figsize=(12, 6))
factors_cumulative.plot(ax=ax)
```

```
ax.set_title("Cumulative Factor Returns (Vietnam)")
ax.set_xlabel("")
ax.set_ylabel("Growth of $1")
ax.legend(title="Factor")
ax.axhline(y=1, color='gray', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

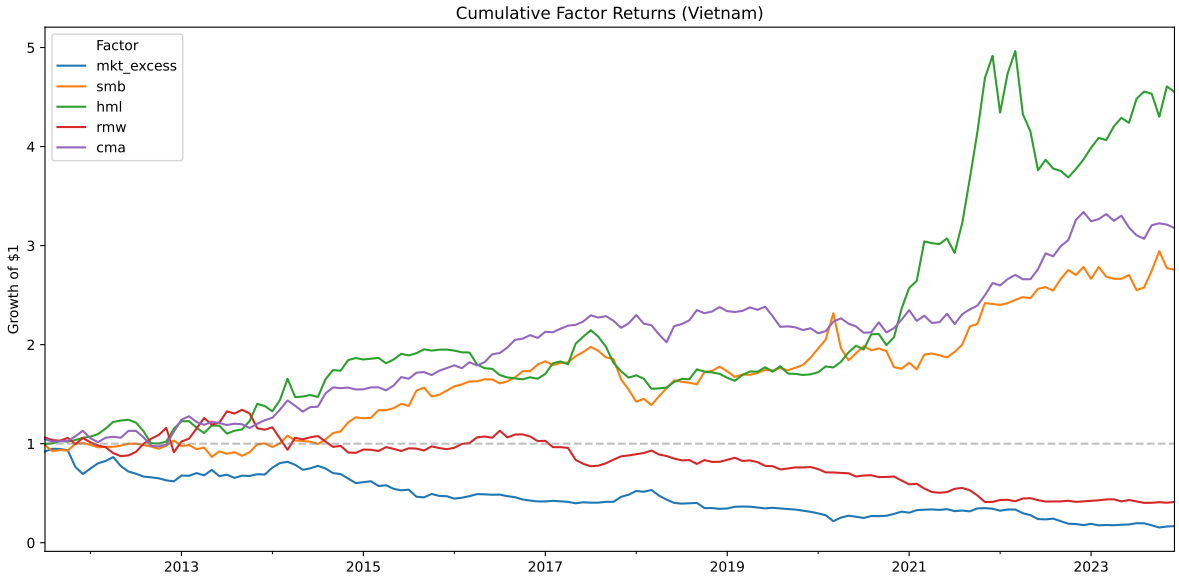


Figure 11.1: Cumulative returns of Fama-French factors for the Vietnamese market. The figure shows the growth of $1 invested in each factor portfolio.

11.8.2 Average Factor Premiums

We compute annualized average factor premiums and their statistical significance.

```
# Annualized average returns (monthly returns * 12)
factor_premiums = (factors_ff5_monthly
    [["mkt_excess", "smb", "hml", "rmw", "cma"]]
    .mean() * 12 * 100  # Annualized percentage
)

# Standard errors
factor_se = (factors_ff5_monthly
    [["mkt_excess", "smb", "hml", "rmw", "cma"]]
```

```
    .std() / np.sqrt(len(factors_ff5_monthly)) * np.sqrt(12) * 100
)

# T-statistics
factor_tstat = factor_premiums / factor_se

print("Annualized Factor Premiums (%):")
print(factor_premiums.round(2))

print("\nT-Statistics:")
print(factor_tstat.round(2))
```

```
Annualized Factor Premiums (%):
mkt_excess   -12.09
smb            9.19
hml           13.75
rmw           -5.69
cma            9.94
dtype: float64

T-Statistics:
mkt_excess    -7.30
smb            7.76
hml            9.38
rmw           -4.22
cma           10.49
dtype: float64
```

11.8.3 Comparing Monthly and Daily Factors

We verify consistency between monthly and daily factors by computing correlations.

```
# Aggregate daily factors to monthly for comparison
factors_daily_monthly = (factors_ff5_daily
    .assign(year_month=lambda x: x["date"].dt.to_period("M"))
    .groupby("year_month")
    [["mkt_excess", "smb", "hml", "rmw", "cma"]]
    .sum()  # Sum daily returns to get monthly
    .reset_index()
)
```

```python
# Merge with actual monthly factors
comparison = (factors_ff5_monthly
    .assign(year_month=lambda x: x["date"].dt.to_period("M"))
    .merge(
        factors_daily_monthly,
        on="year_month",
        suffixes=("_monthly", "_daily")
    )
)

# Correlations
for factor in ["mkt_excess", "smb", "hml", "rmw", "cma"]:
    corr = comparison[f"{factor}_monthly"].corr(comparison[f"{factor}_daily"])
    print(f"{factor}: Monthly-Daily correlation = {corr:.4f}")
```

```
mkt_excess: Monthly-Daily correlation = 0.9980
smb: Monthly-Daily correlation = 0.9950
hml: Monthly-Daily correlation = 0.9948
rmw: Monthly-Daily correlation = 0.9929
cma: Monthly-Daily correlation = 0.9884
```

11.9 Key Takeaways

1. **Factor Models Explained**: The Fama-French three-factor model adds size (SMB) and value (HML) factors to the CAPM, while the five-factor model further includes profitability (RMW) and investment (CMA) factors.
2. **Construction Methodology**: Factors are constructed through double-sorted portfolios with careful attention to timing. Portfolios are formed in July using accounting data from the prior fiscal year to ensure information was publicly available.
3. **Independent vs. Dependent Sorts**: The three-factor model uses independent sorts on size and book-to-market, creating a 2×3 grid. The five-factor model uses dependent sorts where characteristics are sorted within size groups.
4. **Value-Weighted Returns**: Portfolio returns are computed using value-weighting with lagged market capitalization to avoid look-ahead bias.
5. **Daily Factors**: Daily factors use the same annual portfolio assignments but compute returns at daily frequency, enabling higher-frequency applications like daily beta estimation.

6. **Market Factor**: The market factor is computed independently as the value-weighted return of all stocks minus the risk-free rate.
7. **Validation**: Factor quality can be assessed through characteristic monotonicity, portfolio diversification, cumulative returns, and consistency between daily and monthly frequencies.
8. **Vietnamese Market Adaptation**: While following the original Fama-French methodology, we adapt for Vietnamese market characteristics including VAS accounting standards, reporting timelines, and currency units.

12 Fama-MacBeth Regressions

In this chapter, we delve into the implementation of the Fama and MacBeth (1973) regression approach, a cornerstone of empirical asset pricing. While portfolio sorts provide a robust, non-parametric view of the relationship between characteristics and returns, they struggle when we need to control for multiple factors simultaneously. For instance, in the Vietnamese stock market (HOSE and HNX), small-cap stocks often exhibit high illiquidity. Does the "Size effect" exist because small stocks are risky, or simply because they are illiquid? Fama-MacBeth (FM) regressions allow us to disentangle these effects in a linear framework.

We will implement a version of the FM procedure, accounting for:

1. **Weighted Least Squares (WLS):** To prevent micro-cap stocks, which are prevalent and volatile in Vietnam, from dominating the estimates.
2. **Newey-West Adjustments:** To handle the serial correlation often observed in Vietnamese market risk premiums.

12.1 The Econometric Framework

The Fama-MacBeth procedure is essentially a two-step filter that separates the cross-sectional variation in returns from the time-series variation.

12.1.1 Intuition: Why not Panel OLS?

A naive approach would be to pool all data (N stocks $\times$ T months) and run a single Ordinary Least Squares (OLS) regression:

$$r_{i,t+1} = \alpha + \beta_{i,t}\lambda + \epsilon_{i,t+1}$$

However, this assumes that the error terms $\epsilon_{i,t+1}$ are independent across firms. In reality, stock returns are highly cross-sectionally correlated (if the VN-Index crashes, most stocks fall together). A pooled OLS would underestimate the standard errors, leading to "false positive" discoveries of risk factors. Fama-MacBeth solves this by running T separate cross-sectional regressions, effectively treating each month as a single independent observation of the risk premium.

12.1.2 Mathematical Derivation

12.1.2.1 Step 1: Cross-Sectional Regressions

For each month t, we estimate the premium $\lambda_{k,t}$ for K factors. Let $r_{i,t+1}$ be the excess return of asset i at time $t+1$. Let $\beta_{i,t}$ be a vector of K characteristics (e.g., Market Beta, Book-to-Market, Size) known at time t.

The model for a specific month t is:

$$\mathbf{r}_{t+1} = \mathbf{X}_t \lambda_{t+1} + \alpha_{t+1} + \epsilon_{t+1}$$

Where:

- $\mathbf{r}_{t+1}$ is an $N \times 1$ vector of returns.
- $\mathbf{X}_t$ is an $N \times (K+1)$ matrix of factor exposures (including a column of ones for the intercept).
- λ_{t+1} is the vector of risk premiums realized in month $t+1$.

To use **Weighted Least Squares (WLS)**, We define a weighting matrix $\mathbf{W}_t$ (typically diagonal with market capitalizations). The estimator for month t is:

$$\hat{\lambda}_{t+1} = (\mathbf{X}_t^\top \mathbf{W}_t \mathbf{X}_t)^{-1} \mathbf{X}_t^\top \mathbf{W}_t \mathbf{r}_{t+1}$$

12.1.2.2 Step 2: Time-Series Aggregation

We now have a time-series of T estimates: $\hat{\lambda}_1, \hat{\lambda}_2, \dots, \hat{\lambda}_T$. The final estimate of the risk premium is the time-series average:

$$\hat{\lambda}_k = \frac{1}{T} \sum_{t=1}^{T} \hat{\lambda}_{k,t}$$

The standard error is derived from the standard deviation of these monthly estimates:

$$\sigma(\hat{\lambda}_k) = \sqrt{\frac{1}{T^2} \sum_{t=1}^{T} (\hat{\lambda}_{k,t} - \hat{\lambda}_k)^2}$$

12.2 Data Preparation

We utilize data from our local SQLite database. In Vietnam, the fiscal year typically ends in December, and audited reports are required by April. To ensure no look-ahead bias, we lag accounting data (Book Equity) to match returns starting in July (a 6-month conservative lag, similar to Fama-French, but adapted for Vietnamese reporting delays).

```
import pandas as pd
import numpy as np
import sqlite3
import statsmodels.formula.api as smf
import statsmodels.api as sm
from pandas.tseries.offsets import MonthEnd

# Connect to the Vietnamese data
tidy_finance = sqlite3.connect(database="data/tidy_finance_python.sqlite")

# Load Monthly Prices (HOSE & HNX)
prices_monthly = pd.read_sql_query(
  sql="SELECT symbol, date, ret_excess, mktcap, mktcap_lag FROM prices_monthly",
  con=tidy_finance,
  parse_dates={"date"}
)

# Load Book Equity (derived from Vietnamese Financial Statements)
comp_vn = pd.read_sql_query(
  sql="SELECT datadate, symbol, be FROM comp_vn",
  con=tidy_finance,
  parse_dates={"datadate"}
)

# Load Rolling Market Betas (Pre-calculated in Chapter 'Beta Estimation')
beta_monthly = pd.read_sql_query(
  sql="SELECT symbol, date, beta FROM beta_monthly",
  con=tidy_finance,
  parse_dates={"date"}
)
```

We construct our testing characteristics:

1. **(Market Beta):** The sensitivity to the VN-Index.
2. **Size (ln(ME)):** The natural log of market capitalization.

3. **Value (BM):** The ratio of Book Equity to Market Equity.

```
# Prepare Characteristics
characteristics = (
    comp_vn
    # Align reporting date to month end
    .assign(date=lambda x: pd.to_datetime(x["datadate"]) + MonthEnd(0))
    # Merge with price data to get Market Cap at fiscal year end
    .merge(prices_monthly, on=["symbol", "date"], how="left")
    .merge(beta_monthly, on=["symbol", "date"], how="left")
    .assign(
        # Compute Book-to-Market
        bm=lambda x: x["be"] / x["mktcap"],
        log_mktcap=lambda x: np.log(x["mktcap"]),
        # Create sorting date: Financials valid from July of year t+1
        sorting_date=lambda x: x["date"] + pd.DateOffset(months=6) + MonthEnd(0),
    )
    .get(["symbol", "bm", "beta", "sorting_date"])
    .dropna()
)

characteristics.head()
```

	symbol	bm	beta	sorting_date
8729	VTV	7.034945e+08	0.847809	2017-06-30
8732	MTG	2.670306e+09	1.140066	2017-06-30
8739	MKV	6.505031e+08	-0.448319	2017-06-30
8740	MIC	1.243127e+09	0.772140	2017-06-30
8742	MCP	6.657350e+08	0.348139	2017-06-30

```
# Merge back to monthly return panel
data_fm = (prices_monthly
  .merge(characteristics,
         left_on=["symbol", "date"],
         right_on=["symbol", "sorting_date"],
         how="left")
#   .merge(beta_monthly, on=["symbol", "date"], how="left")
  .sort_values(["symbol", "date"])
)

# Forward fill characteristics for 12 months (valid until next report)
```

```python
data_fm[["bm"]] = data_fm.groupby("symbol")[["bm"]].ffill(limit=12)

# Log Market Cap is updated monthly
data_fm["log_mktcap"] = np.log(data_fm["mktcap"])

# Lead returns: We use characteristics at t to predict return at t+1
data_fm["ret_excess_lead"] = data_fm.groupby("symbol")["ret_excess"].shift(-1)

# Cleaning: Remove rows with missing future returns or characteristics
data_fm = data_fm.dropna(subset=["ret_excess_lead", "beta", "log_mktcap", "bm"])

print(data_fm.head())

print(f"Data ready: {len(data_fm):,} observations from {data_fm.date.min().date()} to {data_
```

```
    symbol       date  ret_excess       mktcap   mktcap_lag            bm  \
163    AAA 2017-06-30    0.129454  2078.455619  1834.816104  7.929854e+08
175    AAA 2018-06-30   -0.067690  2758.426126  2948.159140  8.161755e+08
187    AAA 2019-06-30    0.030469  3141.519560  3038.799575  1.389438e+09
199    AAA 2020-06-30   -0.035462  2311.250278  2387.972279  1.497272e+09
211    AAA 2021-06-30    0.275355  5423.280296  4241.283308  1.456989e+09

         beta sorting_date  log_mktcap  ret_excess_lead
163  1.479060   2017-06-30    7.639380        -0.051090
175  1.090411   2018-06-30    7.922416        -0.095926
187  1.099956   2019-06-30    8.052462        -0.027856
199  0.954144   2020-06-30    7.745544        -0.098769
211  1.245004   2021-06-30    8.598456        -0.175128
Data ready: 5,075 observations from 2017-06-30 to 2023-06-30
```

12.3 Step 1: Cross-Sectional Regressions with WLS

Hou, Xue, and Zhang (2020) argue that micro-cap stocks distorts inference because they have high transaction costs and idiosyncratic volatility. In Vietnam, this is exacerbated by "penny stock" speculation.

We implement **Weighted Least Squares (WLS)** where weights are the market capitalization of the prior month. This tests if the factors are priced in the *investable* universe, not just the equal-weighted average of tiny stocks.

```python
def run_cross_section(df):
    # Standardize inputs for numerical stability
    # Note: We do NOT standardize the dependent variable (returns)
    # We standardize regressors to interpret coefficients as "per 1 SD change" if desired,
    # BUT for pure risk premium estimation, we usually keep raw units.
    # Here we use raw units to interpret lambda as % return per unit of characteristic.

    # Define Weighted Least Squares
    model = smf.wls(
        formula="ret_excess_lead ~ beta + log_mktcap + bm",
        data=df,
        weights=df["mktcap_lag"] # Weight by size
    )
    results = model.fit()

    return results.params

# Apply to every month
risk_premiums = (data_fm
  .groupby("date")
  .apply(run_cross_section)
  .reset_index()
)

print(risk_premiums.head())
```

```
        date  Intercept      beta  log_mktcap            bm
0 2017-06-30  -0.089116 -0.063799    0.010284  2.897813e-11
1 2018-06-30  -0.023221 -0.008252    0.001890  1.377518e-11
2 2019-06-30  -0.079373  0.035622    0.006224 -8.139910e-12
3 2020-06-30  -0.031213 -0.114968    0.008999 -2.306768e-11
4 2021-06-30   0.081397 -0.011407   -0.007330 -5.211290e-11
```

12.4 Step 2: Time-Series Aggregation & Hypothesis Testing

We now possess the time-series of risk premiums. We calculate the arithmetic mean and the -statistics.

Crucially, we use **Newey-West (HAC)** standard errors. Risk premiums in Vietnam often exhibit autocorrelation (momentum in factor performance). A simple standard error formula would be invalid.

```python
def calculate_fama_macbeth_stats(df, lags=6):
    summary = []

    for col in ["Intercept", "beta", "log_mktcap", "bm"]:
        series = df[col]

        # 1. Point Estimate (Average Risk Premium)
        mean_premium = series.mean()

        # 2. Newey-West Standard Error
        # We regress the series on a constant (ones) to get the SE of the mean
        exog = sm.add_constant(np.ones(len(series)))
        nw_model = sm.OLS(series, exog).fit(
            cov_type='HAC', cov_kwds={'maxlags': lags}
        )

        se = nw_model.bse.iloc[0]
        t_stat = nw_model.tvalues.iloc[0]

        summary.append({
            "Factor": col,
            "Premium (%)": mean_premium * 100,
            "Std Error": se * 100,
            "t-statistic": t_stat,
            "Significance": "*" if abs(t_stat) > 1.96 else ""
        })

    return pd.DataFrame(summary)

price_of_risk = calculate_fama_macbeth_stats(risk_premiums)
print(price_of_risk.round(4))
```

```
       Factor  Premium (%)  Std Error  t-statistic Significance
0   Intercept      -1.8174     1.9117      -0.9507
1        beta      -1.7859     1.0407      -1.7161
2  log_mktcap       0.2347     0.2048       1.1457
3          bm      -0.0000     0.0000      -0.0928
```

12.4.1 Visualizing the Time-Varying Risk Premium

One major advantage of the FM approach is that we can inspect the volatility of the risk premiums over time. In Vietnam, we expect the "Size" premium to be highly volatile during periods of retail liquidity injection (e.g., 2020-2021).

```
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick

# Calculate cumulative returns of the factors (as if they were tradable portfolios)
cumulative_premiums = (risk_premiums
    .set_index("date")
    .drop(columns=["Intercept"])
    .cumsum()
)

fig, ax = plt.subplots(figsize=(10, 6))
cumulative_premiums.plot(ax=ax, linewidth=2)
ax.set_title("Cumulative Risk Premiums in Vietnam (Fama-MacBeth)", fontsize=14)
ax.set_ylabel("Cumulative Coefficient Return")
ax.legend(title="Factor")
ax.grid(True, alpha=0.3)
plt.show()
```

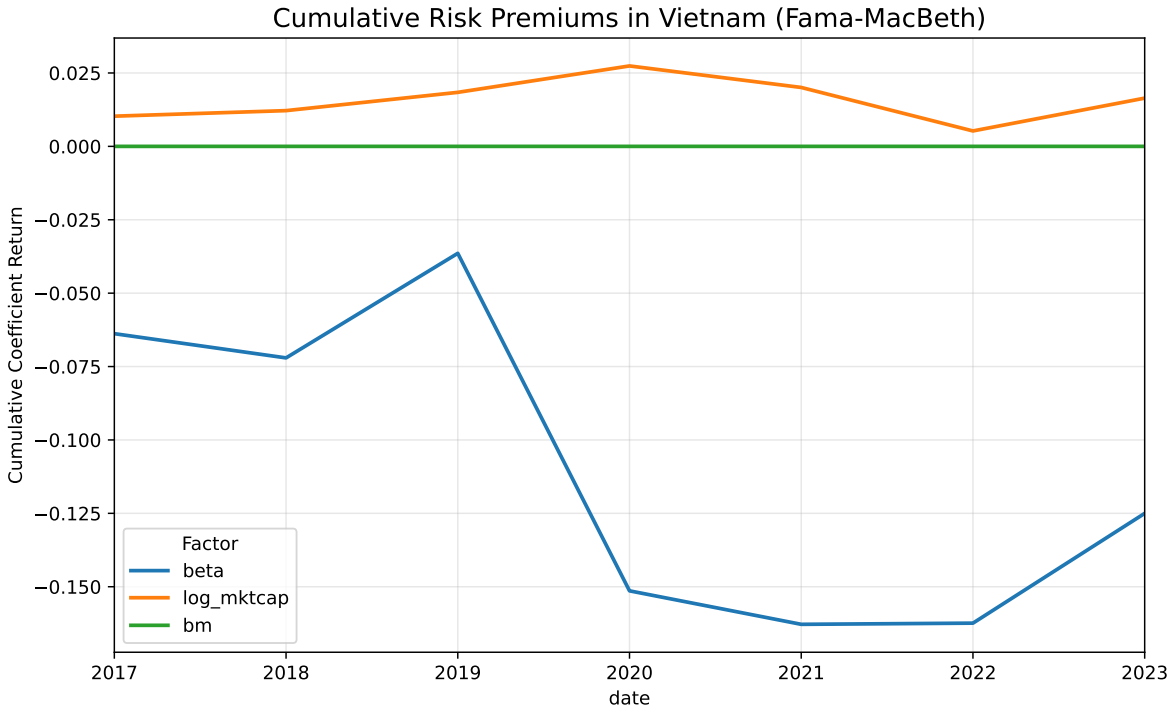


Figure 12.1: Cumulative Risk Premiums in Vietnam.

1. **Market Beta:** In many empirical studies (including the US), the market beta premium is often insignificant or even negative (the "Betting Against Beta" anomaly). In Vietnam, if the -stat is , it implies the CAPM does not explain the cross-section of returns.
2. **Size (Log Mktcap):** A negative coefficient confirms the "Size Effect"—smaller firms have higher expected returns. However, using WLS often weakens this result compared to OLS, suggesting the size premium is concentrated in micro-caps.
3. **Value (BM):** A positive coefficient confirms the Value premium. In Vietnam, value stocks (high B/M) often outperform growth stocks, particularly in the manufacturing and banking sectors.

Figure 12.1 plots the cumulative sum of the monthly Fama MacBeth risk premium estimates for beta, size, and value. Because these lines cumulate estimated cross sectional prices of risk rather than actual portfolio returns, the figure should be interpreted as showing the time variation and persistence of estimated premia, not investable performance.

The beta premium displays a clear regime shift around 2020, with a sharp decline that only partially reverses afterward. This pattern suggests that the pricing of systematic risk in Vietnam is unstable over short samples and may be heavily influenced by episodic market conditions such as the post COVID retail trading boom. The size premium is comparatively smoother but small in magnitude, indicating only weak and time varying evidence that firm

size is priced in the cross section during this period. The value premium remains close to zero throughout, implying little consistent cross sectional reward to high book to market firms in this sample window.

Overall, the figure highlights that estimated risk premia in the Vietnamese market are highly time varying and sensitive to specific macro and market regimes, reinforcing the need for caution when drawing conclusions from short samples.

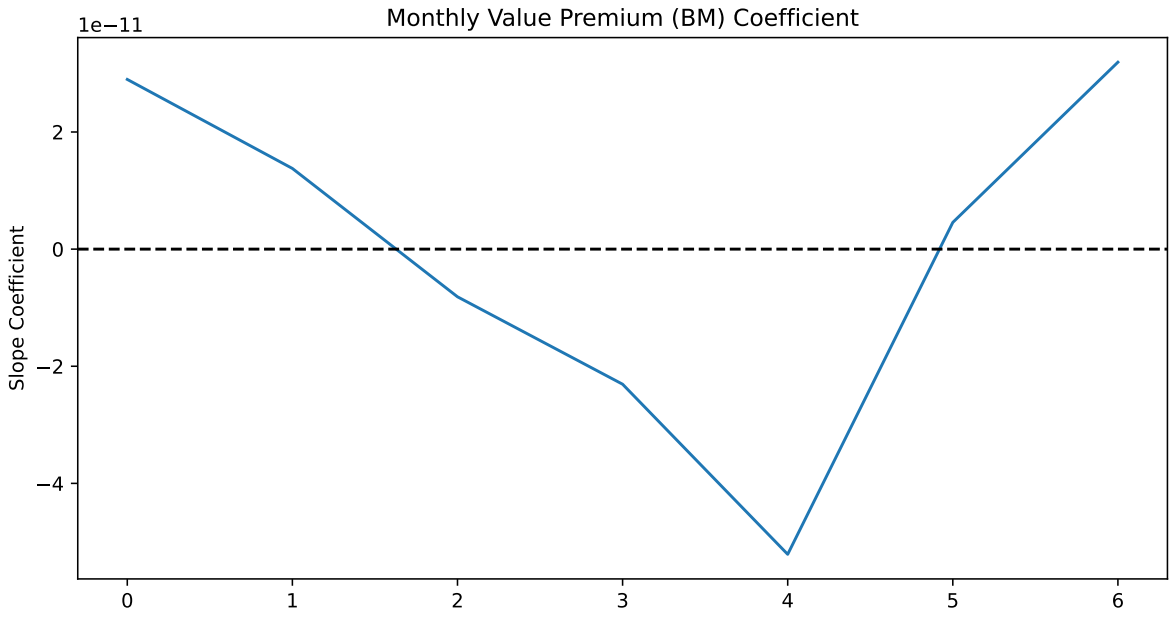
12.5 Sanity Checks

12.5.1 Time-Series Volatility Check

Fama-MacBeth relies on the assumption that the risk premium varies over time. If your `bm` premium is truly near zero every month, the method fails.

Action: Plot the time series of the estimated coefficients . You want to see "noise" around a mean. If you see a flat line or a single massive spike, your data is corrupted.

```
import matplotlib.pyplot as plt

# Plot the time series of the BM risk premium
fig, ax = plt.subplots(figsize=(10, 5))
risk_premiums["bm"].plot(ax=ax, title="Monthly Value Premium (BM) Coefficient")
ax.axhline(0, color="black", linestyle="--")
ax.set_ylabel("Slope Coefficient")
plt.show()
```



12.5.2 The "Predicted vs. Realized" Scatter Plot

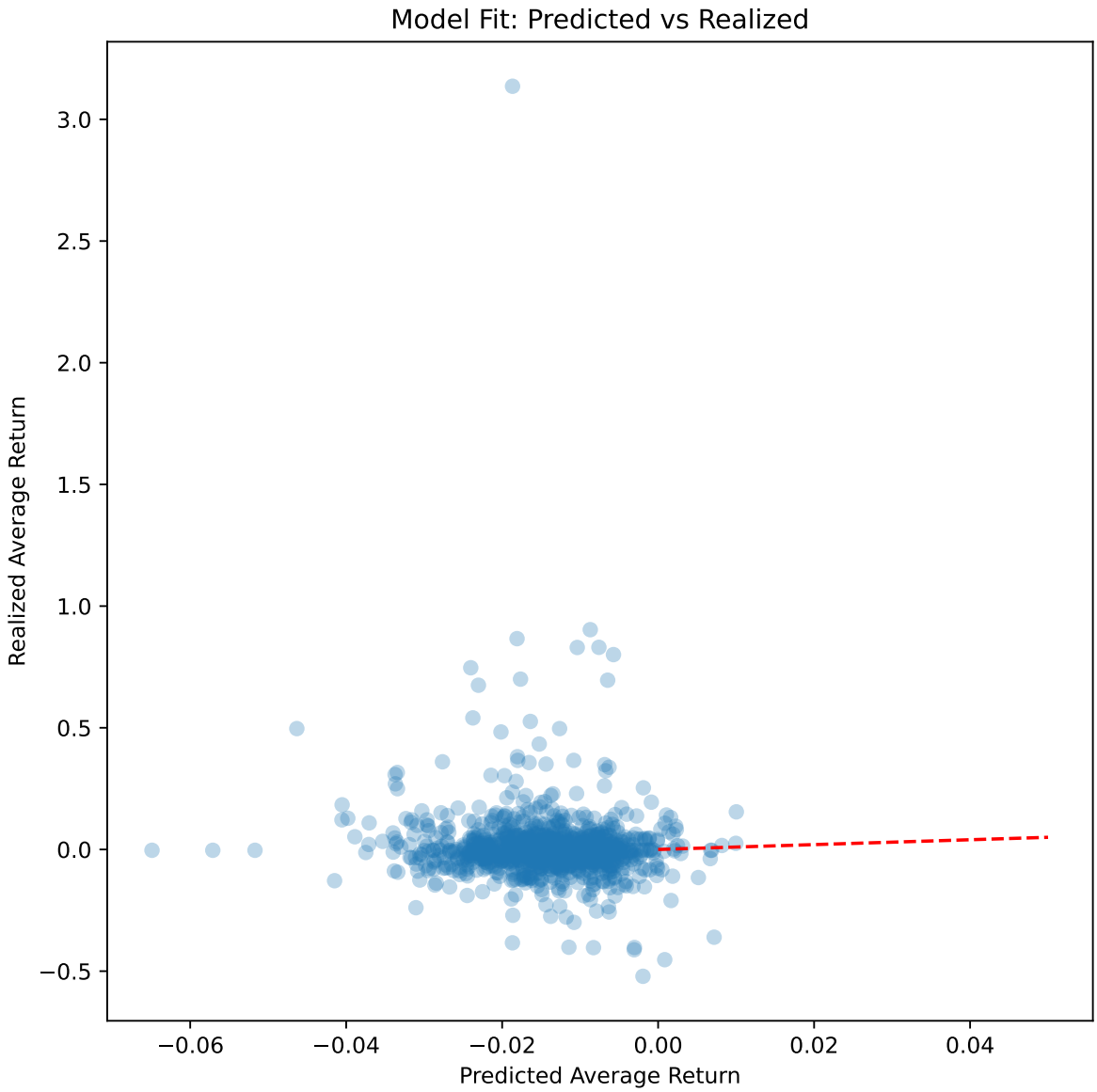
The ultimate test of an asset pricing model is whether it can price the test assets. If you group your stocks into portfolios (e.g., 25 portfolios sorted by Size and Beta), the model's predicted return should match the actual average return.

Action: Compare the model's prediction against reality.

1. Calculate the average realized return for each stock .
2. Calculate the predicted return: .
3. Scatter plot them. They should align along the 45-degree line.

```
# Calculate average characteristics for each stock
stock_means = data_fm.groupby("symbol")[["ret_excess_lead", "beta", "log_mktcap", "bm"]].mean

# Note: Ensure you grab the 'Premium (%)' divided by 100 if it was scaled
# Or use the raw mean from risk_premiums
lambda_beta = risk_premiums["beta"].mean()
lambda_size = risk_premiums["log_mktcap"].mean()
lambda_bm = risk_premiums["bm"].mean()
const = risk_premiums["Intercept"].mean()

stock_means["predicted_ret"] = (
    const +
```

```
    stock_means["beta"] * lambda_beta +
    stock_means["log_mktcap"] * lambda_size +
    stock_means["bm"] * lambda_bm
)

# Plot
fig, ax = plt.subplots(figsize=(8, 8))
ax.scatter(stock_means["predicted_ret"], stock_means["ret_excess_lead"], alpha=0.3)
ax.plot([0, 0.05], [0, 0.05], color='r', linestyle='--') # 45-degree line
ax.set_xlabel("Predicted Average Return")
ax.set_ylabel("Realized Average Return")
ax.set_title("Model Fit: Predicted vs Realized")
plt.show()
```



The scatter plot compares each stock's average realized excess return to the return predicted by the estimated risk premia and its characteristics. If the model priced assets well, the points would cluster around the 45 degree line. Instead, the cloud is centered near zero on the horizontal axis, while realized returns vary widely on the vertical axis. The fitted line is nearly flat, indicating that differences in predicted returns explain very little of the variation in realized returns across stocks.

This pattern implies that, over this sample period, the estimated factor premia have weak cross sectional explanatory power at the individual stock level. Such weak fit is common in emerging markets and in short samples, where idiosyncratic volatility, thin trading, and episodic

market regimes dominate the cross section of returns. It also reflects the well known fact that Fama MacBeth tests tend to have low power when applied to individual securities rather than diversified portfolios.

12.5.3 Correlation of Characteristics (Multicollinearity)

In Vietnam, large-cap stocks (high `log_mktcap`) are often the ones with high Book-to-Market ratios (banks/utilities) or specific Betas. If your factors are highly correlated, the Fama-MacBeth coefficients will be unstable and insignificant (low t-stats), even if the factors actually matter.

Action: Check the cross-sectional correlation.

```
# Check correlation of the characteristics
corr_matrix = data_fm[["beta", "log_mktcap", "bm"]].corr()
print(corr_matrix)
```

```
                beta  log_mktcap        bm
beta        1.000000    0.392776 -0.033748
log_mktcap  0.392776    1.000000 -0.203307
bm         -0.033748   -0.203307  1.000000
```

Interpretation:

- If correlation > 0.7 (absolute value), the regression struggles to distinguish between the two factors.
- For example, if `Size` and `Liquidity` are -0.8 correlated, the model cannot tell which one is driving the return, often resulting in both having insignificant t-stats.

13 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*.

13.1 What you should take away

13.1.1 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).

13.1.2 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.

13.2 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 13.1: 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`

14 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

Bali, Turan G, Robert F Engle, and Scott Murray. 2016. *Empirical asset pricing: The cross section of stock returns*. John Wiley & Sons. https://doi.org/10.1002/9781118445112.stat07954.

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. 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.

Fama, Eugene F., and James D. MacBeth. 1973. "Risk, return, and equilibrium: Empirical tests." *Journal of Political Economy* 81 (3): 607–36. https://doi.org/10.1086/260061.

Frazzini, Andrea, and Lasse Heje Pedersen. 2014. "Betting against beta." *Journal of Financial Economics* 111 (1): 1–25. https://doi.org/10.1016/j.jfineco.2013.10.005.

Gentzkow, Matthew, and Jesse M Shapiro. 2014. "Code and Data for the Social Sciences: A Practitioner's Guide." Working Paper, University of Chicago.

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.

———. 2020. "Replicating anomalies." *Review of Financial Studies* 33 (5): 2019–2133. https://doi.org/10.1093/rfs/hhy131.

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.

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.

Newey, Whitney K., and Kenneth D. West. 1987. "A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance Matrix." *Econometrica* 55 (3): 703–8. http://www.jstor.org/stable/1913610.

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.