Software, Rice Database, Technical Indicators

BUSI 722: Data-Driven Finance II

Kerry Back

Review: CAPM & Fama-French Factors

The CAPM

The Capital Asset Pricing Model says that the expected excess return of any asset is proportional to the market's expected excess return:

$$\mathbb{E}[r_i] - r_f = \beta_i\big(\mathbb{E}[r_m] - r_f\big)$$

- r_i = return on asset i, r_f = risk-free rate, r_m = market return
- $\beta_i = \dfrac{\text{Cov}(r_i, r_m)}{\text{Var}(r_m)}$
- Higher beta $\Rightarrow$ higher expected return

CAPM ⇔ Zero Alphas

Run a time-series regression for each asset i:

$$r_{it} - r_{ft} = \alpha_i + \beta_i(r_{mt} - r_{ft}) + \varepsilon_{it}$$

- The CAPM is equivalent to saying $\alpha_i = 0$ for every asset.
- If $\alpha_i > 0$: the asset earns more than the CAPM predicts — it is "underpriced."
- If $\alpha_i < 0$: the asset earns less than the CAPM predicts — it is "overpriced."

Testing the CAPM = testing whether alphas are jointly zero.

The Size Effect

Small stocks have historically earned higher average returns than large stocks, even after adjusting for beta.

- Sort stocks into portfolios by market capitalization
- Small-stock portfolios earn positive CAPM alphas; large-stock portfolios earn negative alphas
- This is evidence against the CAPM

Size is measured by market capitalization = price $\times$ shares outstanding.

The Value Effect

Value stocks (high book-to-market) have historically outperformed growth stocks (low book-to-market), even after adjusting for beta.

- Book-to-market = book equity / market equity
- High B/M ("value") stocks earn positive CAPM alphas; low B/M ("growth") stocks earn negative alphas
- This is further evidence against the CAPM

Fama-French Three-Factor Model (1993)

Fama and French proposed replacing the CAPM with a three-factor model:

$$\mathbb{E}[r_i] - r_f = \beta_i^{\mathsf{MKT}} \cdot \mathsf{MKT} + \beta_i^{\mathsf{SMB}} \cdot \mathsf{SMB} + \beta_i^{\mathsf{HML}} \cdot \mathsf{HML}$$

- **MKT** $= r_m - r_f$: market excess return
- **SMB** (Small Minus Big): return on small stocks minus return on large stocks
- **HML** (High Minus Low): return on high B/M stocks minus return on low B/M stocks

Interpreting the Three-Factor Model

The three-factor model says that expected returns are explained by exposures to three sources of risk.

- The model "works" if alphas are zero when we regress excess returns on the three factors:

$$r_{it} - r_{ft} = \alpha_i + \beta_i^{\mathsf{MKT}}\mathsf{MKT}_t + \beta_i^{\mathsf{SMB}}\mathsf{SMB}_t + \beta_i^{\mathsf{HML}}\mathsf{HML}_t + \varepsilon_{it}$$

- The size and value effects that produced nonzero CAPM alphas are captured by SMB and HML.

- But ... other anomalies (momentum, profitability, investment) still produce nonzero alphas.

Setup: Claude Pro & Claude Code

Initial Steps

1. Sign up for Claude Pro:

1. Visit claude.ai and create an account (email or Google/Apple sign-in)
2. After signing in, click "Upgrade to Pro" in the sidebar
3. Choose the monthly plan and complete payment

2. YouTube Video

Mac – Install Claude Code

Step 1: Install Node.js

- Using Homebrew: `brew install node`
- Or download installer from nodejs.org

Step 2: Install Claude Code

- Run: `npm install -g @anthropic-ai/claude-code`

Step 3: Verify Installation

- Run: `claude doctor`

Windows – Install Claude Code

Step 1: Install Node.js

- Download and install from nodejs.org
- Choose the LTS (Long Term Support) version

Step 2: Install Git for Windows

- Download from git-scm.com/download/win
- During installation, select "Git Bash" (default option)

Step 3: Install Claude Code

- In PowerShell, run: npm install -g @anthropic-ai/claude-code

Install Python & Create Virtual Environment

Install Python:

- Tell Claude Code: "Install Python 3.13 and add it to the path."
- Tell Claude Code: "Upgrade pip"

Create Virtual Environment:

- Tell Claude Code: "Create a virtual environment using Python 3.13 in my current directory."

Install packages in the virtual environment:

- numpy pandas scipy statsmodels scikit-learn

VS Code & Database Setup

Install and Open VS Code

- Install VS Code from code.visualstudio.com, then File → Open Folder → your course folder
- Install extensions: Python, Jupyter, Claude Code, Data Wrangler, Rainbow CSV
- View → Command Palette → "Python: Select Interpreter" → choose venv

Launch Claude Code in VS Code

- If you see the orange Claude icon in the top toolbar, click it.
- If not, create a new file (File $\rightarrow$ New Text File) and you should see it.
- Or View $\rightarrow$ Command Palette and enter "Claude Code: Open in New Tab."

Test: Ask Claude Code: What is the sum of the first 1,000 integers?

Stock Market Database

- Database Guide: https://portal-guide.rice-business.org
- Visit data-portal.rice-business.org to get an access token
- The data portal is an AI agent that uses ChatGPT to generate SQL and query the database

Create .env File:

- Tell Claude Code to create a .env file
- Add: RICE_ACCESS_TOKEN=your_token_here

Database Tables

Three tables are relevant for price-based analysis:

- **SEP** (prices): `ticker`, `date`, `close` (split-adjusted), `closeadj` (adjusted for splits, dividends, and spinoffs), `volume`
- **DAILY** (valuation metrics): `ticker`, `date`, `marketcap` (USD millions), `pb`, `pe`, `ps`
- **TICKERS** (company info): `ticker`, `sector`, `industry`, `scalemarketcap`

Full documentation at https://portal-guide.rice-business.org

Querying the Data Portal

Ask Claude Code to query the database directly.

Example prompts:

- "Query the SEP table for end-of-month close and closeadj for all stocks from 2018 through 2024."
- "Query the DAILY table for end-of-month marketcap for all stocks from 2018 through 2024."
- "Query the TICKERS table for sector and industry for all tickers."

Claude writes Python code that sends HTTP requests to the API, using your access token from the .env file.

Computing Returns

Monthly returns are computed from closeadj (the fully adjusted price):

$$r_t = \frac{\text{closeadj}_t}{\text{closeadj}_{t-1}} - 1$$

- Use closeadj, not close: it accounts for dividends and spinoffs, so the return includes total return.
- close (split-adjusted only) is useful for computing moving averages and chart prices.
- Group by ticker when computing returns — never mix prices across stocks.

Computing Momentum

Momentum is the cumulative return from month $t-13$ to month $t-2$:

$$\text{momentum}_t = \frac{\texttt{closeadj}_{t-2}}{\texttt{closeadj}_{t-13}} - 1$$

- Skips the most recent month ($t-1$): short-term reversal contaminates the signal.
- Requires 13 months of price history, so the first 12 months are `NaN`.
- **Lagged return** (the prior month's return) is a separate signal that captures short-term reversal.

Moving Averages

Moving averages smooth out price noise and reveal trends.

- **Short MA** (3-month): $MA_3 = \frac{1}{3} \sum_{j=0}^{2} close_{t-j}$. Tracks recent price action.
- **Long MA** (12-month): $MA_{12} = \frac{1}{12} \sum_{j=0}^{11} close_{t-j}$. Captures the longer-term trend.
- **Signals**: MA Ratio (close/MA_{12}, uptrend if > 1) and Golden Cross ($MA_3 > MA_{12}$).

Compute moving averages from close (split-adjusted), grouped by ticker.

Assigning Quantiles Each Month

Each month, sort stocks into groups (deciles, quintiles, etc.) by a signal:

```python
# Assign momentum deciles (1 = lowest, 10 = highest)
df["mom_decile"] = df.groupby("month")["momentum"].transform(
    lambda x: pd.qcut(x, 10, labels=False, duplicates="drop") + 1
)

# Assign size quintiles (1 = smallest, 5 = largest)
df["size_q"] = df.groupby("month")["marketcap"].transform(
    lambda x: pd.qcut(x, 5, labels=False, duplicates="drop") + 1
)
```

- groupby("month") ensures sorting is cross-sectional (within each month).
- pd.qcut assigns equal-count groups; labels=False gives integers starting at 0.

Computing Equal-Weighted Portfolio Returns

Compute the mean return of each group each month, then average across months:

```
# Mean return by decile and month
decile_monthly = df.groupby(["month", "mom_decile"])["return"] \
                   .mean().reset_index()

# Time-series average across all months
decile_avg = decile_monthly.groupby("mom_decile")["return"].mean()

print(decile_avg)
```

- The first `groupby` computes the equal-weighted portfolio return each month.
- The second `groupby` averages across months to get the expected return of each portfolio.
- A monotone pattern (D1 low, D10 high) indicates the signal has predictive power.

Data Formats

Claude should automatically save data as parquet files. This is a compact, fast format.

To view the data, ask Claude to:

- **"Convert the file to Excel."** Then open as usual.
- **"Convert the file to csv."** Then double-click in VS Code File Explorer.
- **"Read the data in a Jupyter notebook."** Then work with the data in Python.