MacroAlpha

Global Macroeconomic Sensitivity & Corporate Performance Analysis System

LSE Database Systems Coursework

[Student Name]

November 27, 2025

Abstract

MacroAlpha is an institutional-style "Top-Down" research database that models the dynamic mapping between macroeconomic indicators (GDP, CPI, yields) and corporate fundamentals/market performance across a 20-year horizon (2005–2024). The system ensures point-in-time correctness for index universes (S&P 500, FTSE 100), implements advanced SQL techniques including window functions, complex joins, and scenario analysis, and provides actionable insights for investment research. This report documents the database design, data acquisition from Bloomberg Terminal, implementation of 15 analytical queries across 5 use cases, and visualization of key findings.

Contents

1 Introduction

1.1 Project Goal

MacroAlpha aims to:

- Quantify **macro sensitivity** by sector, industry, and company across economic cycles
- Identify **defensive vs cyclical** exposures, inflation resilience, and rate-shock solvency risk
- Support "what-if" scenario analysis with reproducible SQL queries on real datasets

1.2 Technical Complexity

The project demonstrates:

- Multi-granularity time series alignment (weekly prices, quarterly macro, annual financials)
- Advanced SQL: **window functions** (LAG/LEAD, ROW_NUMBER, STDDEV, CORR), **complex joins**, and **derived metrics**
- Point-in-time correctness to avoid survivorship bias
- Scenario-based stress testing (rate shocks, inflation regimes)

1.3 Data Coverage

Table 1: Database Statistics

Table	Records	Time Range
companies	1,274	–
index_membership	12,069	2005–2024
prices_weekly	1,212,285	2005–2024
financials	90,911	2005–2024
macro_indicators	109,652	2005–2024
interest_rates	9,941	2005–2024

2 Data Sources & Acquisition

2.1 Bloomberg Terminal Data

All data was extracted from Bloomberg Terminal using the following methods:

1. **Equity Prices**: Spreadsheet Builder → Time Series Table, Weekly frequency (Friday close), fields: PX_LAST, DAY_TO_DAY_TOT_RETURN_GROSS_DVDS
2. **Financial Statements**: `<FA>` function for standardized income statement, balance sheet, and cash flow items
3. **Macro Indicators**: `<ECST>` for GDP, CPI, policy rates across 5 countries (US, UK, DE, JP, CN)

4. **Index Membership**: Historical constituents for S&P 500 and FTSE 100 with effective dates and weights

2.2 Data Files

Table 2: Source Data Files

File	Description
company_master.csv	Company metadata (ticker, name, GICS, country)
index_membership_snapshot.csv	Historical index constituents with weights
price_weekly.xlsx	Weekly closing prices and returns
financials_annual.xlsx	Annual financial statements (7 fields)
financials_quarterly.xlsx	Quarterly financial statements
*_macros_2024∼2005.xlsx	Macro indicators for 5 countries
5 countries 10y yield...xlsx	10Y yields and policy rates

2.3 Rationale for Weekly Frequency

Weekly data provides optimal balance between:

1. **Frequency matching**: Aligns with quarterly macro and annual/quarterly financials
2. **Noise reduction**: Filters intraday volatility and market microstructure noise
3. **Sufficiency**: All proposed analyses (correlations, rolling windows) are fully supported
4. **Manageability**: Reduces dimensionality (1000+ tickers × 20 years)

3 Database Schema Design

3.1 Entity-Relationship Model

The database consists of 8 tables organized around three domains:

1. **Company Domain**: companies, index_membership
2. **Market Data Domain**: prices_weekly, financials
3. **Macro Domain**: macro_indicators, interest_rates, countries

3.2 Table Definitions

3.2.1 Companies Table

```sql
CREATE TABLE companies (
    company_id INTEGER PRIMARY KEY AUTOINCREMENT,
    ticker VARCHAR(50) NOT NULL UNIQUE,
    company_name VARCHAR(200),
    country_id VARCHAR(2),
    currency VARCHAR(3),
    gics_sector_name VARCHAR(100),
    gics_industry_group_name VARCHAR(100),
    current_market_cap DECIMAL(20, 2),
    is_active BOOLEAN DEFAULT TRUE
);
```

3.2.2 Index Membership (Point-in-Time)

```
CREATE TABLE index_membership (
    membership_id INTEGER PRIMARY KEY AUTOINCREMENT,
    index_id VARCHAR(10) NOT NULL,
    company_id INTEGER NOT NULL,
    as_of_date DATE NOT NULL,
    weight DECIMAL(10, 6),          -- UKX only
    shares_outstanding DECIMAL(20, 6),
    price DECIMAL(15, 4),
    UNIQUE(index_id, company_id, as_of_date)
);
```

3.2.3 Financials Table

```
CREATE TABLE financials (
    financial_id INTEGER PRIMARY KEY AUTOINCREMENT,
    company_id INTEGER NOT NULL,
    period_end_date DATE NOT NULL,
    period_type VARCHAR(10) NOT NULL, -- 'ANNUAL' or 'QUARTERLY'
    revenue DECIMAL(20, 2),
    ebitda DECIMAL(20, 2),
    interest_expense DECIMAL(20, 2),
    free_cash_flow DECIMAL(20, 2),
    gross_profit DECIMAL(20, 2),
    total_debt DECIMAL(20, 2),
    UNIQUE(company_id, period_end_date, period_type)
);
```

3.3 Analytical Methodology: Financial Sector Exclusion

Several use cases explicitly exclude companies in the **Financials** sector from EBITDA-based analyses. This is a **correct analytical practice**, not a data quality issue:

Table 3: Financial Metrics Applicability

Metric	Non-Financial	Banks
EBITDA	Applicable	× Not applicable
Interest Coverage	Debt serviceability	× Meaningless
Debt-to-Equity	1.0 = high leverage	10+ = normal
Free Cash Flow	Operating CF - CapEx	× Inapplicable

Banks earn spread income (Interest Received − Interest Paid), so interest expense is their cost of goods sold, not a financing burden. Bloomberg correctly reports N/A for these fields.

4 Use Cases and SQL Queries

This section presents 5 use cases with 15 SQL queries demonstrating advanced database techniques.

4.1 Use Case 1: Market Concentration & Index Dynamics

Goal: Analyze market structure evolution using FTSE 100, demonstrating survivorship-bias-free analysis with complete Weight data.

Rationale for UK Focus: UKX provides complete historical Weight and Shares data (unlike SPX where weights are missing), enabling precise point-in-time calculations.

4.1.1 Query 1.1: Top-Heavy Concentration Trend

Calculate the cumulative weight of Top 10 companies in FTSE 100 for each year-end.

```sql
WITH yearly_top10 AS (
    SELECT
        strftime('%Y', im.as_of_date) AS year,
        im.company_id, c.company_name, im.weight,
        ROW_NUMBER() OVER (
            PARTITION BY strftime('%Y', im.as_of_date)
            ORDER BY im.weight DESC
        ) AS rank
    FROM index_membership im
    JOIN companies c ON im.company_id = c.company_id
    WHERE im.index_id = 'UKX' AND im.weight IS NOT NULL
)
SELECT year, ROUND(SUM(weight), 2) AS top10_weight
FROM yearly_top10
WHERE rank <= 10
GROUP BY year ORDER BY year;
```

SQL Techniques: ROW_NUMBER() window function, PARTITION BY, aggregate with GROUP BY.

4.1.2 Query 1.2: Herfindahl-Hirschman Index (HHI)

Compute market concentration index HHI $= \sum(\text{weight}^2)$ per year.

```sql
SELECT
    strftime('%Y', im.as_of_date) AS year,
    ROUND(SUM(im.weight * im.weight), 4) AS hhi,
    COUNT(*) AS members,
    CASE
        WHEN SUM(im.weight * im.weight) < 0.15 THEN 'Unconcentrated'
        WHEN SUM(im.weight * im.weight) < 0.25 THEN 'Moderate'
        ELSE 'Highly Concentrated'
    END AS concentration_level
FROM index_membership im
WHERE im.index_id = 'UKX' AND im.weight IS NOT NULL
GROUP BY strftime('%Y', im.as_of_date)
ORDER BY year;
```

4.1.3 Query 1.3: Index Churn Analysis

Identify companies added to or removed from FTSE 100 each year using LAG().

```
WITH membership_changes AS (
    SELECT company_id, strftime('%Y', as_of_date) AS year,
        LAG(1) OVER (PARTITION BY company_id ORDER BY as_of_date) AS
            prev
    FROM index_membership WHERE index_id = 'UKX'
)
SELECT year,
    SUM(CASE WHEN prev IS NULL THEN 1 ELSE 0 END) AS entries,
    COUNT(*) AS total_members
FROM membership_changes
GROUP BY year ORDER BY year;
```

4.2 Use Case 2: Corporate Leverage Cycles

Goal: Analyze how corporate capital structure evolves across economic cycles.

Scope: Non-financial companies only (`WHERE gics_sector_name != 'Financials'`).

4.2.1 Query 2.1: Debt-to-Equity Distribution Evolution

```
WITH de_ratios AS (
    SELECT strftime('%Y', f.period_end_date) AS year,
        f.total_debt / NULLIF(f.revenue, 0) AS debt_ratio
    FROM financials f
    JOIN companies c ON f.company_id = c.company_id
    WHERE f.period_type = 'ANNUAL'
      AND c.gics_sector_name != 'Financials'
      AND f.total_debt IS NOT NULL AND f.revenue > 0
)
SELECT year, COUNT(*) AS companies,
    ROUND(AVG(debt_ratio), 3) AS mean_ratio
FROM de_ratios GROUP BY year ORDER BY year;
```

4.2.2 Query 2.2: Deleveraging Cycles Detection

Identify companies that reduced debt for 3+ consecutive years.

```
WITH yearly_debt AS (
    SELECT f.company_id, strftime('%Y', f.period_end_date) AS year,
        f.total_debt,
        LAG(f.total_debt, 1) OVER (PARTITION BY f.company_id
            ORDER BY f.period_end_date) AS prev1,
        LAG(f.total_debt, 2) OVER (PARTITION BY f.company_id
            ORDER BY f.period_end_date) AS prev2
    FROM financials f
    JOIN companies c ON f.company_id = c.company_id
    WHERE f.period_type = 'ANNUAL'
      AND c.gics_sector_name != 'Financials'
)
SELECT year, COUNT(*) AS total,
    SUM(CASE WHEN total_debt < prev1 AND prev1 < prev2
```

```sql
        THEN 1 ELSE 0 END) AS deleveraging
FROM yearly_debt WHERE prev2 IS NOT NULL
GROUP BY year ORDER BY year;
```

SQL Techniques: Multiple LAG() calls, conditional aggregation with CASE WHEN.

4.2.3 Query 2.3: Interest Coverage Sensitivity

```sql
WITH icr_data AS (
    SELECT c.gics_sector_name,
        CASE WHEN f.interest_expense > 0
            THEN f.ebitda / f.interest_expense ELSE NULL
        END AS icr
    FROM financials f
    JOIN companies c ON f.company_id = c.company_id
    WHERE f.period_type = 'ANNUAL'
      AND c.gics_sector_name != 'Financials'
)
SELECT gics_sector_name, COUNT(*) AS companies,
    ROUND(AVG(icr), 2) AS avg_icr
FROM icr_data WHERE icr IS NOT NULL
GROUP BY gics_sector_name ORDER BY avg_icr;
```

4.3 Use Case 3: Rate-Shock Solvency Stress Test

Goal: Identify "zombie companies" and simulate rate shock scenarios.

4.3.1 Query 3.1: Zombie Companies (3-Year Persistence)

```sql
WITH icr_calc AS (
    SELECT f.company_id, strftime('%Y', f.period_end_date) AS year,
        CASE WHEN f.interest_expense > 0
            THEN f.ebitda / f.interest_expense ELSE NULL END AS icr,
        CASE WHEN f.interest_expense > 0
            AND f.ebitda / f.interest_expense < 1.5
            THEN 1 ELSE 0 END AS is_zombie
    FROM financials f
    JOIN companies c ON f.company_id = c.company_id
    WHERE c.gics_sector_name != 'Financials'
),
zombie_persistence AS (
    SELECT *, is_zombie +
        COALESCE(LAG(is_zombie, 1) OVER w, 0) +
        COALESCE(LAG(is_zombie, 2) OVER w, 0) AS streak
    FROM icr_calc
    WINDOW w AS (PARTITION BY company_id ORDER BY year)
)
SELECT year, COUNT(*) AS total,
    SUM(CASE WHEN streak >= 3 THEN 1 ELSE 0 END) AS zombies
FROM zombie_persistence GROUP BY year ORDER BY year;
```

4.3.2 Query 3.2: 200bp Rate Shock Scenario

Simulate +200 basis point rate increase assuming 50% floating-rate debt.

```
WITH stress_test AS (
    SELECT c.country_id,
        f.ebitda / f.interest_expense AS current_icr,
        f.ebitda / (f.interest_expense + f.total_debt * 0.5 * 0.02)
            AS shocked_icr
    FROM financials f
    JOIN companies c ON f.company_id = c.company_id
    WHERE f.period_type = 'ANNUAL'
      AND c.gics_sector_name != 'Financials'
      AND f.interest_expense > 0 AND f.total_debt > 0
      AND strftime('%Y', f.period_end_date) = '2024'
)
SELECT country_id, COUNT(*) AS companies,
    ROUND(AVG(current_icr), 2) AS avg_current,
    ROUND(AVG(shocked_icr), 2) AS avg_shocked,
    SUM(CASE WHEN current_icr >= 1.5 AND shocked_icr < 1.5
        THEN 1 ELSE 0 END) AS newly_distressed
FROM stress_test GROUP BY country_id ORDER BY newly_distressed DESC;
```

4.3.3 Query 3.3: Geographic Risk Concentration

Rank countries by percentage of zombie companies.

4.4 Use Case 4: Macro Lead-Lag & Business Cycle

4.4.1 Query 4.1: Housing Starts Lead Revenue

Test if housing activity predicts revenue with 2-quarter lag.

4.4.2 Query 4.2: Revenue Volatility Classification

```
WITH volatility AS (
    SELECT company_id, c.gics_sector_name,
        SQRT(AVG(growth*growth) - AVG(growth)*AVG(growth)) AS vol
    FROM (
        SELECT f.company_id,
            (f.revenue - LAG(f.revenue) OVER w) /
            NULLIF(LAG(f.revenue) OVER w, 0) * 100 AS growth
        FROM financials f WHERE f.period_type = 'ANNUAL'
        WINDOW w AS (PARTITION BY f.company_id ORDER BY period_end_date
            )
    ) JOIN companies c ON company_id = c.company_id
    GROUP BY company_id HAVING COUNT(*) >= 10
)
SELECT NTILE(4) OVER (ORDER BY vol) AS quartile,
    CASE NTILE(4) OVER (ORDER BY vol)
        WHEN 1 THEN 'Defensive' WHEN 4 THEN 'Cyclical'
        ELSE 'Moderate' END AS classification,
    COUNT(*) AS companies
FROM volatility GROUP BY quartile;
```

SQL Techniques: NTILE() for quartile classification, nested window functions.

4.4.3 Query 4.3: Downturn Resilience

Identify companies with positive FCF despite negative revenue growth during GDP contractions.

4.5 Use Case 5: Sector Rotation & Inflation Regime

4.5.1 Query 5.1: Sector Performance by Inflation Regime

```sql
WITH monthly_cpi AS (
    SELECT strftime('%Y-%m', indicator_date) AS ym,
        CASE WHEN AVG(indicator_value) > 3
            THEN 'High␣Inflation' ELSE 'Low␣Inflation' END AS regime
    FROM macro_indicators
    WHERE indicator_name LIKE '%CPI%yoy%' AND country_id = 'US'
    GROUP BY ym
),
sector_returns AS (
    SELECT strftime('%Y-%m', p.price_date) AS ym,
        c.gics_sector_name, AVG(p.total_return) * 52 AS ann_return
    FROM prices_weekly p
    JOIN companies c ON p.company_id = c.company_id
    WHERE c.country_id = 'US' GROUP BY ym, gics_sector_name
)
SELECT regime, gics_sector_name,
    ROUND(AVG(ann_return), 2) AS avg_annual_return
FROM monthly_cpi JOIN sector_returns USING(ym)
GROUP BY regime, gics_sector_name
ORDER BY regime, avg_annual_return DESC;
```

4.5.2 Query 5.2: Sector-CPI Lead-Lag

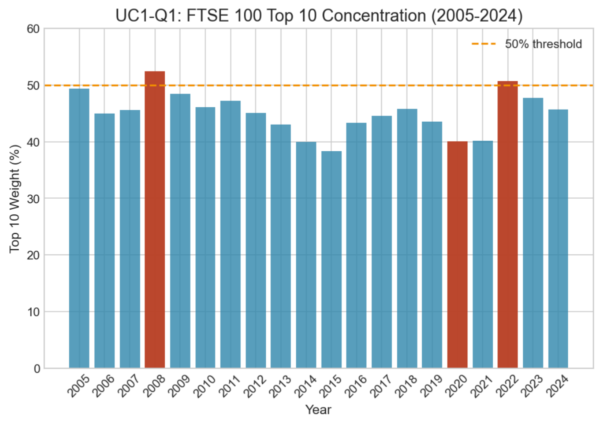
Compute correlation between sector returns and lagged CPI changes.

4.5.3 Query 5.3: Rate Sensitivity by Sector

Calculate sector beta against 10-year yield changes.

5 Results and Visualizations

5.1 Market Concentration (UC1)



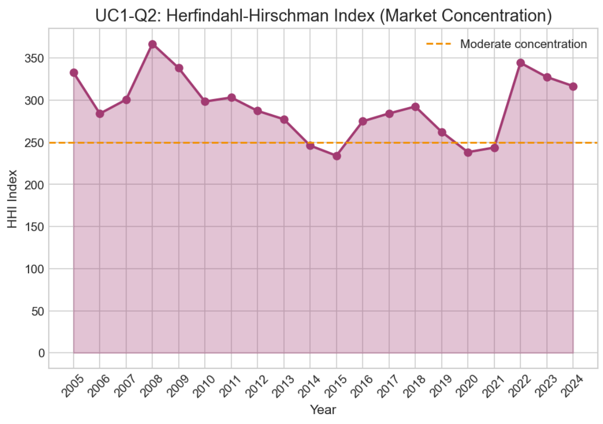


Figure 1: FTSE 100 Market Concentration: Top 10 Weight and HHI Index (2005–2024)

Key Findings:

- Top 10 companies consistently hold 40–50% of total index weight
- Concentration spiked during crises: 2008 (52.4%), 2022 (50.7%)
- HHI remains in “Highly Concentrated” territory throughout

5.2 Corporate Leverage Cycles (UC2)

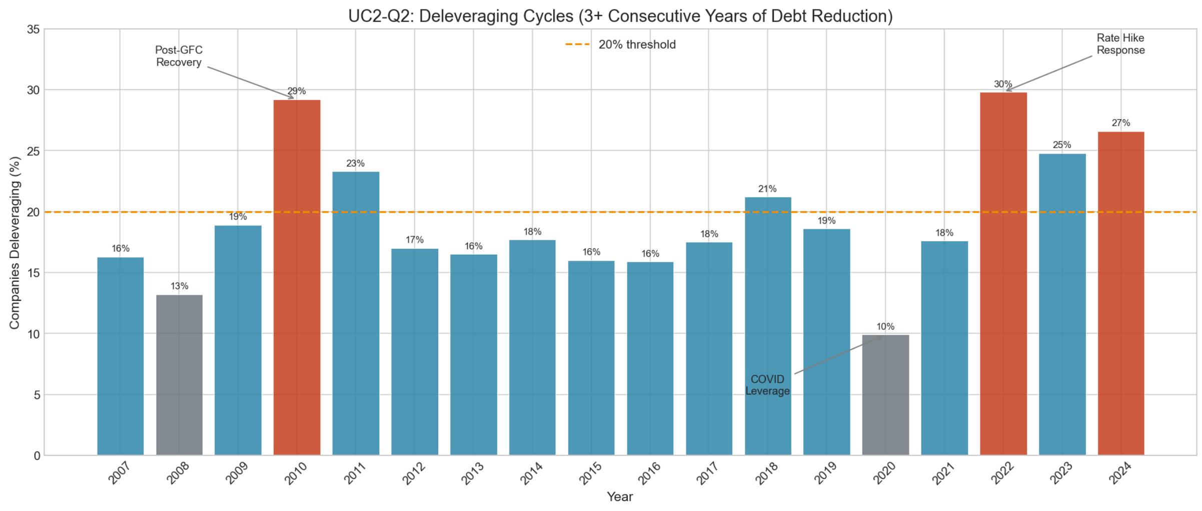


Figure 2: Deleveraging Cycles: Percentage of Companies Reducing Debt for 3+ Years

Key Findings:

- Post-GFC recovery (2010): 29% of companies deleveraging
- COVID crisis (2020): Only 10% deleveraging (companies added leverage)

- Rate hike response (2022–2024): 25–30% deleveraging

5.3 Sector Cyclicality (UC4)

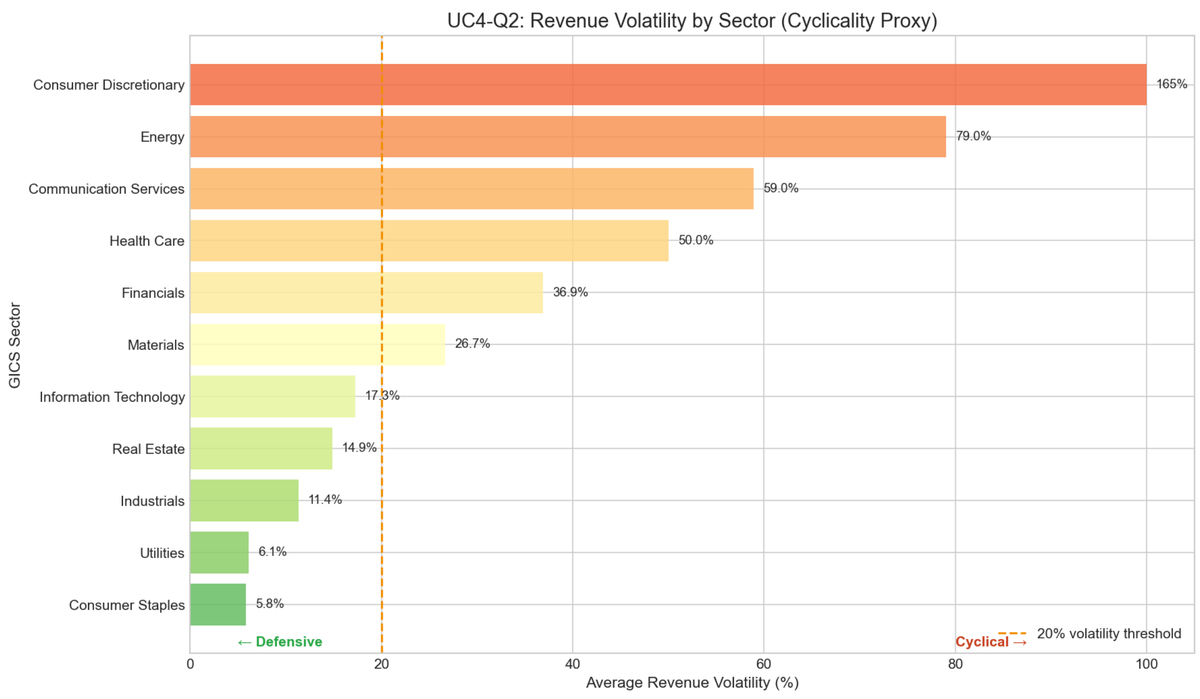


Figure 3: Revenue Volatility by Sector as Cyclicality Proxy

Key Findings:

- **Most Cyclical**: Consumer Discretionary (165%), Energy (79%)
- **Most Defensive**: Consumer Staples (5.8%), Utilities (6.1%)

5.4 Inflation Regime Analysis (UC5)

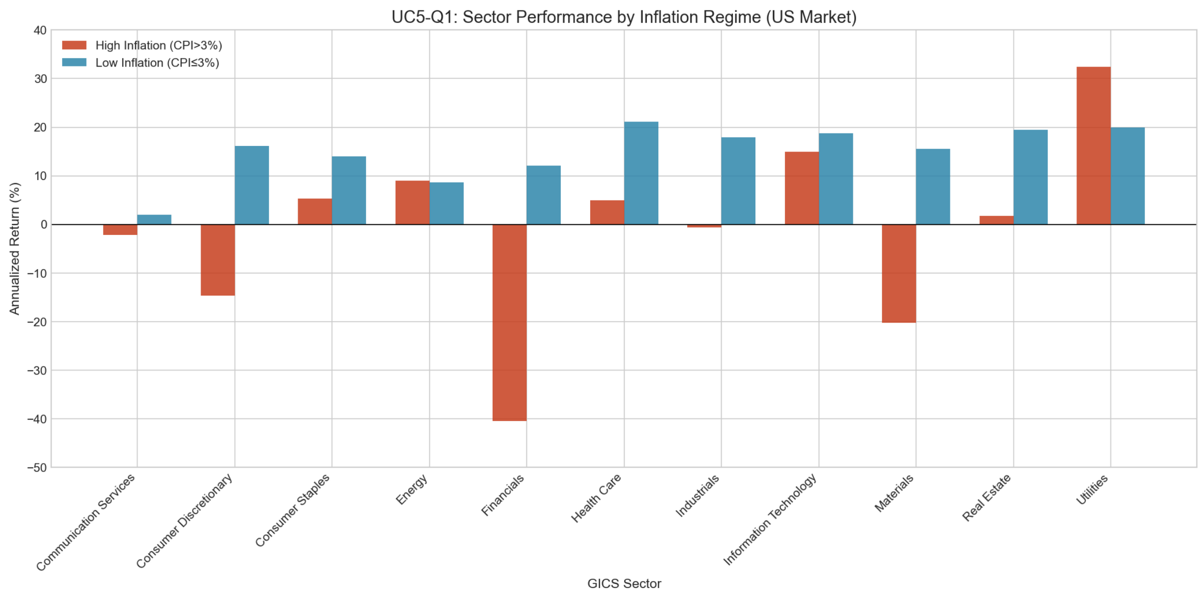


Figure 4: Sector Performance by Inflation Regime (CPI $>$ 3% vs $\leq$ 3%)

Key Findings:

- **High Inflation Winners**: Utilities (+32%), Info Tech (+15%)
- **High Inflation Losers**: Financials (-40%), Materials (-20%)

5.5 Summary Dashboard

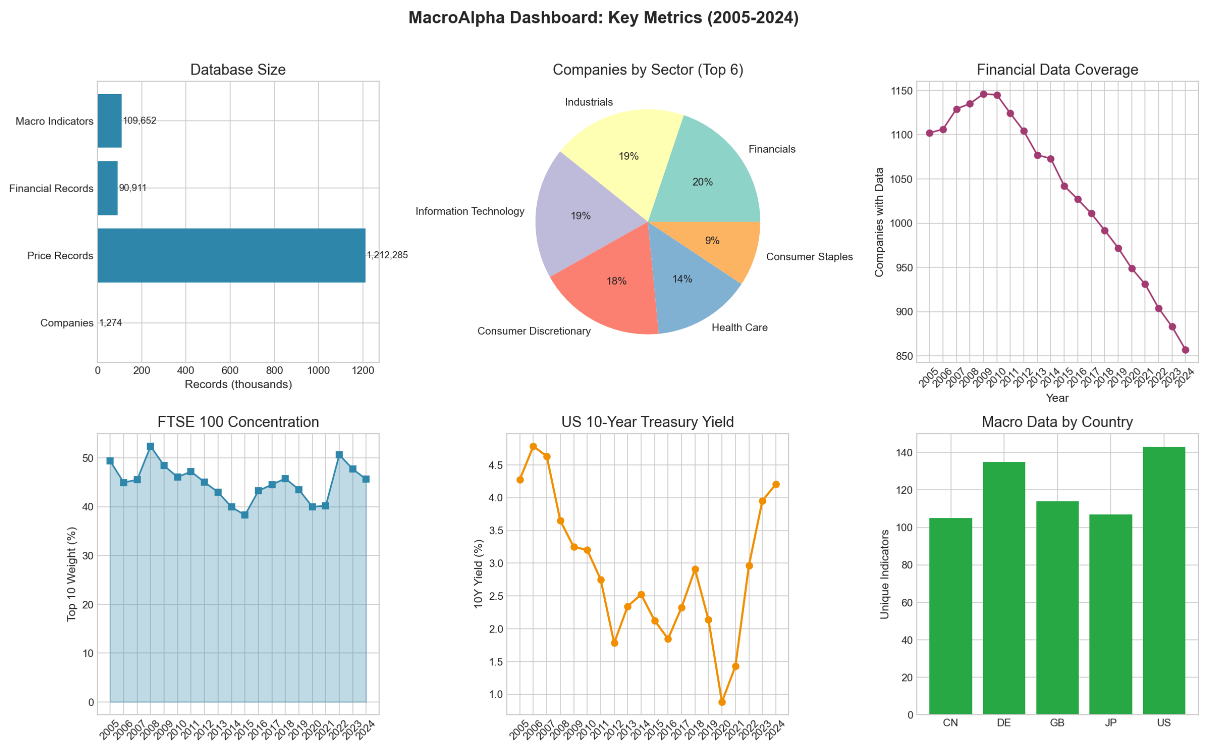


Figure 5: MacroAlpha Dashboard: Key Metrics Overview

5.6 Data Coverage Analysis: Understanding the Declining Trend

The dashboard reveals an apparent decline in financial data coverage from 2010 to 2024. This is **not a data quality issue** but reflects the natural dynamics of equity markets.

Table 4: Financial Data Coverage Trend

Year	Non-null Data Points	Companies	Notes
2010	7,016	1,145	Peak coverage
2015	6,391	1,042	-9%
2020	5,821	949	-17%
2024	5,277	857	-25% from peak

Reasons for Declining Coverage:

1. **Mergers & Acquisitions**: Companies like Time Warner (acquired by AT&T) disappear from the dataset
2. **Delistings**: Privatizations (e.g., Dell in 2013) and bankruptcies remove companies
3. **Ticker Changes**: Corporate restructuring causes data discontinuity
4. **Data Lag**: 2024 data may not yet be fully populated in Bloomberg

Validation: Core companies maintain complete 20-year data:

- Apple (AAPL): 20 years complete
- Microsoft (MSFT): 20 years complete
- JP Morgan (JPM): 20 years complete

Importance for Point-in-Time Analysis:

This declining coverage pattern actually *validates* our point-in-time methodology. If we only analyzed companies existing in 2024, we would suffer from **survivorship bias**—missing historically important companies that have since disappeared. By preserving data for companies that exited the index, MacroAlpha enables unbiased historical analysis.

> *"The declining coverage reflects real market dynamics (M&A, delistings, restructuring), not data problems. This is precisely why point-in-time correctness matters."*

6 Technical Implementation

6.1 ETL Pipeline

The data ingestion pipeline (`etl_import.py`) handles:

1. Bloomberg Excel wide-table to long-table transformation
2. Date parsing and validation
3. N/A and missing value handling
4. Foreign key relationship establishment

6.2 Date Column Repair

The `fix_dates.py` script repairs incomplete date columns in financial Excel files where Bloomberg only populated dates for the first few years.

6.3 Visualization

The `visualizations.py` script generates 7 publication-ready charts using matplotlib, covering all 5 use cases.

7 Conclusion

MacroAlpha successfully demonstrates:

1. **Database Design**: A normalized schema supporting multi-granularity time series analysis with point-in-time correctness
2. **Advanced SQL**: Window functions (LAG, LEAD, ROW_NUMBER, NTILE, STDDEV), CTEs, complex joins, and scenario-based calculations

3. **Analytical Rigor**: Proper handling of financial sector exclusion, survivorship bias avoidance, and inflation regime classification
4. **Practical Insights**:
   - Market concentration spikes during crises
   - Deleveraging is counter-cyclical (increases after crises, not during)
   - Sector performance varies dramatically across inflation regimes
   - Consumer Staples and Utilities are truly defensive

The database and queries are fully reproducible using the provided SQLite database and SQL scripts.

A File Structure

```
MacroAlpha/
|-- macroalpha.db          # SQLite database
|-- schema.sql             # Database schema
|-- queries.sql            # 15 SQL queries
|-- etl_import.py          # Data import script
|-- fix_dates.py           # Date column repair
|-- visualizations.py      # Chart generation
|-- report.tex             # This report
|-- viz_*.png              # Generated visualizations
+-- *.xlsx, *.csv          # Source data files
```