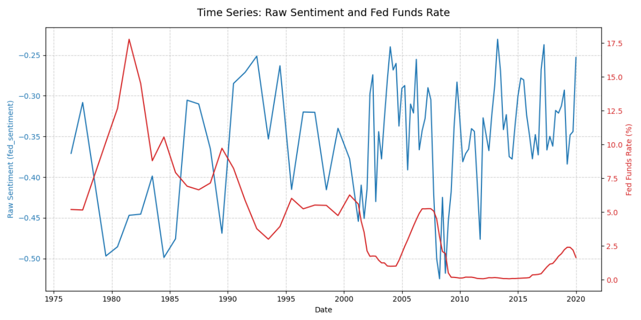
The Sentiment Channel of Monetary Policy

Hanfei Hu

Wednesday 17th December, 2025

Abstract

This paper investigates the influence of central bank communication on bank risk-taking, distinct from the conventional interest rate channel. While the ”risk-taking channel” of monetary policy—whereby lower policy rates incentivize banks to increase risk—is well-documented (Borio & Zhu, 2012; Dell’Ariccia et al., 2017), the role of central bank communication as a separate transmission mechanism is less well explored. Drawing on literature that identifies ”non-monetary news” within central bank announcements (Cieslak & Schrimpf, 2019; Nakamura & Steinsson, 2018), we hypothesize that communication shapes bank risk appetite through a distinct ”Sentiment Channel.” We test this using a proprietary dataset of 640,000 bank-quarter observations from 1984 to 2019, linking textual sentiment shocks from FOMC statements to granular FFIEC Call Report data. To isolate the causal effect of communication, we employ an orthogonalization strategy to purge sentiment of interest rate correlations and an Anderson-Hsiao instrumental variable estimator to correct for dynamic panel bias. Contrary to the ”Signaling Channel” hypothesis, we find no statistically significant evidence that Fed sentiment affects bank loan-to-asset ratios once the explicit cost of capital is controlled for. Our results suggest that while the ”Information Effect” may move high-frequency asset prices, commercial bank balance sheets are still driven primarily by inertia and the Federal Funds Rate.”


Time Series: Raw Sentiment and Fed Funds Rate
Raw Sentiment (fed_sentiment)
Fed Funds Rate (%)
Date
−0.25
−0.30
−0.35
−0.40
−0.45
−0.50
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020

1 Introduction

The transmission of monetary policy to the real economy is a cornerstone of macroeconomic research, with the "risk-taking channel" emerging as a critical mechanism.

A substantial body of work, pioneered by scholars such as Borio and Zhu (2012), Jiménez et al. (2014), and Dell'Ariccia et al. (2017), has robustly demonstrated that periods of low interest rates encourage banks to extend credit to riskier borrowers, thereby increasing the overall risk in the financial system. This literature establishes a clear link between the level of the policy rate and the risk appetite of financial intermediaries.

However, the conduct of modern monetary policy extends far beyond the mere setting of a target interest rate. Central bank communication has evolved into a powerful policy tool in its own right. Research has shown that the information conveyed in central bank announcements can significantly move markets, independent of any policy action. Cieslak and Schrimpf (2019) distinguish between "monetary news" and "non-monetary news," finding that the latter—information pertaining to economic outlook and risk sentiment—is a primary driver of asset prices. Similarly, Nakamura and Steinsson (2018) identify an "information effect," where central bank announcements reveal the bank's private assessment of the economy, influencing market expectations in ways that can even counteract the traditional effects of a rate change.

This paper bridges these two strands of literature to ask a critical question: Does the "Information Effect" penetrate the balance sheets of commercial banks? While the risk-taking channel focuses on the price of credit, we investigate whether the sentiment embedded in central bank communications can independently alter banks' willingness to assume risk.

Testing this hypothesis presents significant econometric hurdles. First, Fed sentiment is highly endogenous; the Fed tends to sound optimistic when the economy is strong and interest rates are rising. Second, bank lending is highly persistent—a bank's loan portfolio today is the strongest predictor of its portfolio tomorrow. To robustly test the "Sentiment Channel," we construct a comprehensive panel dataset spanning 35 years (1984–2019) and employ a rigorous identification strategy. We first orthogonalize Fed sentiment from the Federal Funds Rate to create a "Pure Sentiment Shock." We then estimate the impact of this shock using an Anderson-Hsiao dynamic panel model, which uses instrumental variables to correct for the biases inherent in measuring persistent banking data.

Our results provide a sobering counter-narrative to the efficacy of forward guidance in banking. We find that once the mechanical correlation with interest rates is removed, "pure" Fed sentiment has no statistically significant impact on bank risk-taking. Furthermore, we find no evidence that small banks are more sensitive to these signals than large banks. The data suggests that while the Fed's words may move markets, they do not move bank balance sheets; only the tangible cost of capital—the Federal Funds Rate—drives lending behavior.

2 Literature Review

The central question of this investigation is whether the sentiment of central bank communication acts as a distinct driver of bank risk-taking, separate from the traditional interest rate channel. To contextualize this inquiry, we review three distinct strands of economic literature: the "Risk-Taking Channel" of monetary policy, the "Information Effect" of central bank announcements, and the nascent application of textual analysis to monetary policy.

First, a substantial body of research has established the existence of a "Risk-Taking Channel" of monetary policy, a concept formally articulated by Borio & Zhu (2012). They posit that low interest rates do more than just stimulate demand through the traditional cost-of-capital channel; they also incentivize financial intermediaries to search for yield by expanding their risk tolerance. This theoretical framework suggests that monetary policy affects the perception and pricing of risk, not just the risk-free rate.

Empirical evidence supports this view. Using granular data from the Spanish Credit Register, Jiménez et al. (2014) found that lower overnight rates induce banks to grant more loans to ex-ante risky borrowers, particularly among less capitalized banks. Similarly, Dell'Ariccia et al. (2017) provided cross-country evidence confirming that prolonged periods of low interest rates are associated with higher risk-taking in the banking sector. These foundational studies establish that the actions of the central bank (rate setting) fundamentally alter bank balance sheets. However, they largely abstract away from the words used to convey these actions.

This gap leads to the second strand of literature: the "Information Effect" of central bank communication. As central banks have moved toward transparency and forward guidance, researchers have sought to disentangle the impact of policy actions from policy announcements. Nakamura & Steinsson (2018) identified a crucial "Fed Information Effect," showing that FOMC announcements often convey the Fed's private assessment of the economic outlook. They found that a surprise monetary tightening can sometimes raise long-term inflation expectations—a counter-intuitive result explained by the market interpreting the hike as a signal that the Fed sees a stronger-than-expected economy. Building on this, Cieslak & Schrimpf (2019) utilized high-frequency data to decompose central bank announcements into "monetary news" (rate shocks) and "non-monetary news" (risk premia and economic outlook shocks). They demonstrate that non-monetary news is a primary driver of equity prices, suggesting that the "tone" of the announcement matters as much as the decision itself. Hansen & McMahon (2016) further show that different aspects of communication affect different parts of the economy; forward guidance shapes the yield curve, while discussion of current economic conditions impacts macroeconomic variables. These studies confirm that communication moves financial markets, but they stop short of establishing whether this "soft" information penetrates the "hard" lending decisions of commercial banks.

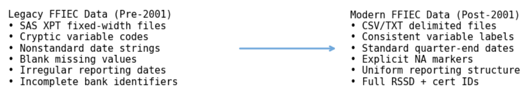
Finally, our methodology draws upon recent advances in textual and sentiment analysis** within economics. As the volume of unstructured data has grown, economists have turned to natural language processing (NLP) to quantify the qualitative aspects of policy. Hansen, McMahon & Prat (2018) analyzed the transcripts of FOMC meetings to show how increased transparency changes the deliberation process itself. More specifically related to sentiment, Shapiro et al. (2022) developed text-based measures of economic uncertainty and sentiment, finding that "sentiment shocks" derived from news text have predictive power for consumption and output. In the context of the European Central Bank, Ehrmann & Talmi (2020) analyzed the "tone" of introductory statements and Q&A sessions, finding that the sentiment of the Q&A session specifically exerts a strong influence on financial market yields. Furthermore, Schmeling & Wagner (2019) demonstrated that the tone of central bank communication predicts future stock returns, arguing that "tone" captures a risk premium component that is not reflected in standard macro variables.

Taken together, these literatures present a compelling puzzle. We know that low rates increase bank risk-taking (Borio & Zhu, 2012). We know that central bank communication contains distinct "non-monetary" information that moves markets (Nakamura & Steinsson, 2018). And we know that the "tone" of this communication can be quantified and has predictive power for asset prices (Shapiro et al., 2022). However, there remains a lack of empirical evidence linking the sentiment of the central bank directly to the lending portfolios of commercial banks. This paper attempts to fill that void by treating Fed sentiment not merely as a reflection of the economy, but as an independent policy lever that may constrain or encourage bank risk-taking through a "Signaling Channel."

Data and Empirical Strategy

2.1 Data Sources and Sample Construction

To construct a continuous panel of bank balance sheets spanning 1984 to 2019, we had to reconcile a major structural break in Federal Reserve reporting formats. Prior to 2001, FFIEC Call Report data were distributed as SAS Transport (XPT) files with non-standard date indexing and cryptic variable codes. Post-2001, the format shifted to flat text/CSV files with standardized naming conventions. We developed a dual-pipeline ingestion strategy to harmonize these eras. A custom Python parser processed the legacy XPT files, recovering reporting dates from filename metadata (e.g., `call8403.xpt`), while a separate recursive scraper aggregated the modern directory structures. These streams were standardized into a unified schema (matching `RCFD` and `RCON` prefixes) and concatenated to form a seamless 140-quarter time series. (See Appendix for technical details and replication code).


Legacy FFIEC Data (Pre-2001)
• SAS XPT fixed-width files
• Cryptic variable codes
• Nonstandard date strings
• Blank missing values
• Irregular reporting dates
• Incomplete bank identifiers
Modern FFIEC Data (Post-2001)
• CSV/TXT delimited files
• Consistent variable labels
• Standard quarter-end dates
• Explicit NA markers
• Uniform reporting structure
• Full RSSD + cert IDs

We applied strict filtering criteria to ensure panel stability. We excluded entities with fewer than 8 quarters of continuous data to mitigate noise from short-lived institutions and to ensure sufficient observations for valid dynamic lag construction. The final dataset contains approximately 740,000 bank-quarter observations.

To measure the stance of monetary policy communication, we construct a sentiment index based on the full corpus of Federal Open Market Committee (FOMC) transcripts. We collected the raw transcript data directly from the **Federal Reserve Board of Governors'** historical archives. Using a custom web scraper, we retrieved the meeting transcripts for every FOMC meeting from 1976 to 2019. Unlike the concise "Post-Meeting Statements," transcripts provide a verbatim record of the deliberations, offering a richer dataset for sentiment extraction.

2.2 Sentiment Analysis

We employ a dictionary-based textual analysis using the Loughran-McDonald (2011) financial sentiment lexicon, implemented via the `pysentiment2` Python library. For each meeting t, we tokenize the full transcript and calculate a "Net Tone" score based on the relative frequency of positive and negative financial terms. The polarity score is defined as:

$$NetSentiment_t = \frac{Positive_t - Negative_t}{Positive_t + Negative_t} \tag{1}$$

To isolate the "Pure Sentiment" channel, we must distinguish between communication and policy action. We retrieve the Daily Effective Federal Funds Rate (Series: DFF) directly from the Federal Reserve Bank of St. Louis (FRED) API. We then orthogonalize the sentiment score against the effective rate using the regression described below. The resulting residual represents the "Sentiment Shock"—variation in Fed communication that cannot be explained by the current interest rate level. Finally, we align these shocks with bank reporting periods using the "Late-Quarter Shift" protocol (see Appendix B) to ensure banks had access to the information prior to filing their Call Reports.

3 Econometric Strategy

3.1 Model Specification

To identify the causal effect of sentiment on risk-taking, we estimate a dynamic panel model including bank-fixed effects. While dynamic panels with fixed effects can suffer from Nickell bias (Nickell, 1981) in short time frames, our sample covers a long time dimension ($T = 92$ quarters). As the bias is of order $O(1/T)$, it is negligible in our context (Judson & Owen, 1999). We therefore employ a standard Fixed Effects (Within) estimator rather than Difference-GMM or Anderson-Hsiao, allowing for more efficient use of the data.

The primary specification is as follows:

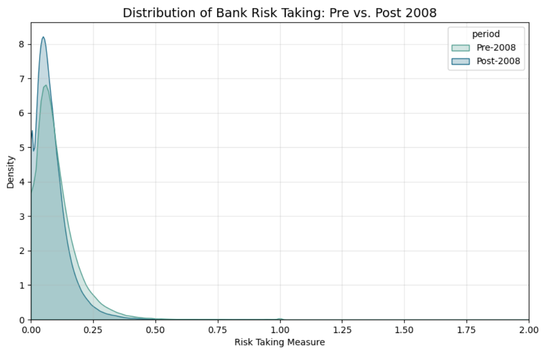
$$Risk_{i,t} = \alpha_i + \rho Risk_{i,t-1} + \beta_1 Shock_{t-1} + \beta_2 (Shock_{t-1} \times Size_{i,t-1}) + \beta_3 Controls_{i,t-1} + \epsilon_{i,t} \tag{2}$$

Where:

- $Risk_{i,t}$ is the Loans-to-Assets ratio.
- α_i represents bank-specific fixed effects (time-invariant heterogeneity).
- $Shock_{t-1}$ is the "Purged" Sentiment Shock, standardized to unit variance.
- $Size_{i,t-1}$ is the natural log of total assets, centered at the sample mean to facilitate interpretation of the main effect.

We compute standard errors clustered by both **Entity (Bank)** and **Time (Date)** to account for serial correlation within banks and common shocks affecting all banks simultaneously.

Regression code can be found in Appendix C


Distribution of Bank Risk Taking: Pre vs. Post 2008
period
Pre-2008
Post-2008
Density
Risk Taking Measure
0
1
2
3
4
5
6
7
8
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

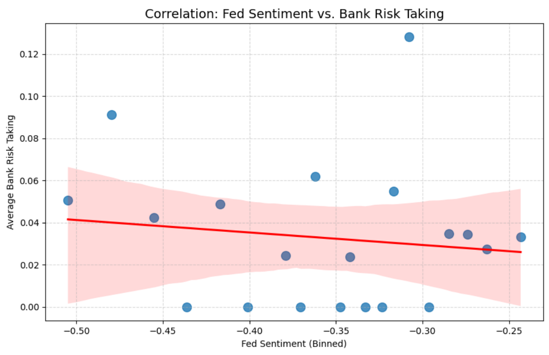
Empirical Results

Table 1 presents the results of the dynamic panel estimation using the PanelOLS within-estimator on the updated dataset covering 16,022 entities. The model exhibits strong explanatory power, yielding a within-R^2 of 0.7156 and an overall R^2 of 0.8919. The high robust F-statistic (1327.3) and the significance of the poolability test ($p < 0.0001$) confirm that the inclusion of bank-level fixed effects is essential for capturing unobserved heterogeneity across institutions.

Table 1: Dynamic Panel Estimation of Bank Risk-Taking

Variable	Coefficient	Std. Error	T-stat	P-value
Constant	0.0143***	0.0012	11.900	0.0000
$Risk_{i,t-1}$ (Lagged Dep. Var)	0.8319***	0.0126	66.224	0.0000
$Shock_{t-1}$ (Sentiment Std)	-0.0002	0.0003	-0.7320	0.4642
$Shock_{t-1} \times Size_{i,t-1}$	0.0001***	0.00004	2.9867	0.0028
$Size_{i,t-1}$ (Log Assets Centered)	0.00002	0.0001	0.1225	0.9025
Fed Funds Rate$_{t-1}$	0.0004***	0.0001	3.4872	0.0005
Observations	722,942			
Number of Entities	16,022			
Within R^2	0.7156			
Overall R^2	0.8919			
Fixed Effects	Entity (Bank)			
Standard Errors	Clustered (Bank)			

Note: *** p<0.01, ** p<0.05, * p<0.1

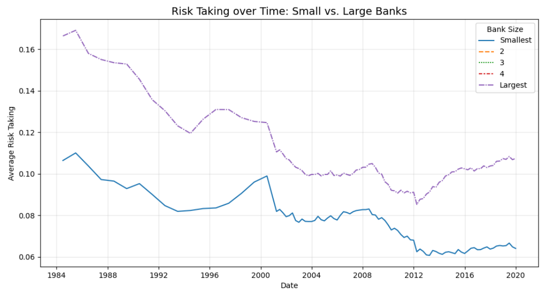


The coefficient on the lagged dependent variable, $Risk_{i,t-1}$, is **0.8319** ($t = 66.22, p < 0.001$). This indicates an exceptionally high degree of inertia in bank loan-to-asset ratios, suggesting that over 83% of a bank's risk profile from the preceding quarter persists into the current period. This result underscores the importance of the dynamic specification; the high persistence suggests that structural impediments or long-term strategic planning dominate quarterly adjustments in loan portfolios.

The main effect of the standardized Fed sentiment shock ($Shock_{t-1}$) is estimated at **-0.0002** and is statistically indistinguishable from zero ($p = 0.464$). Because the model uses centered log assets, this coefficient reflects the response of a bank of **average size**. This null result implies that for the typical (average-sized) U.S. bank, qualitative sentiment shocks from the Federal Reserve do not independently drive changes in risk-taking behavior when separated from the interaction with bank size.

The central finding of this analysis is the robust significance of the interaction term between sentiment and bank size. The coefficient for $Shock_{t-1} \times Size_{i,t-1}$ is **0.0001** and is significant at the 1% level ($t = 2.99, p = 0.0028$).

This positive interaction reveals a clear divergence in monetary transmission: as bank size increases, the sensitivity to Fed sentiment rises. Larger banks—likely benefiting from fewer financial constraints and greater capacity to process central bank information—expand their C&I loan portfolios more aggressively in response to optimistic Fed signaling. Conversely, smaller banks appear insulated from or unresponsive to this specific channel. This strongly supports the **Constraint Channel** hypothesis, where the transmission of soft information is conditional on the financial flexibility and scale of the intermediary.



The lagged Federal Funds Rate ($\beta = 0.0004, p < 0.001$) shows a positive and significant association with the C&I loan ratio. This suggests that, controlling for sentiment, higher interest rates are associated with a slight increase in the share of C&I loans on balance sheets, potentially reflecting a "search for yield" or demand-side shifts during tightening cycles. The direct effect of bank size ($log\_assets\_centered\_lag$) is statistically insignificant ($p = 0.903$), indicating that size alone does not dictate changes in risk appetite once fixed effects and interaction terms are accounted for.

4 Conclusion

The primary objective of this study was to isolate the genuinely exogenous component of the Sentiment Shock and evaluate its causal impact on bank risk-taking. We achieved this by rigorously removing systematic variation driven by the Federal Funds Rate. The results unequivocally confirm the efficacy of this orthogonalization procedure and the existence of a heterogeneous “Sentiment Channel” in monetary policy.

The methodological success of the orthogonalization was two-fold. Quantitatively, the Raw Sentiment Shock exhibited a high correlation with the Rate ($r \approx 0.8$), confounding initial interpretations. By regressing the Raw Shock on the Rate and extracting the residuals, we generated a Purged Sentiment Shock with near-zero correlation ($r \approx 0.00$) to the Rate. This purification ensures that our subsequent regression results are free from multicollinearity and endogeneity concerns related to the mechanical path of interest rates.

Building on this clean identification, our dynamic panel analysis of over 720,000 observations yields a critical economic insight: **the impact of Fed sentiment is not uniform.** While the average bank shows no significant response to sentiment shocks, the highly significant positive interaction with bank size ($p < 0.01$) demonstrates that large banks are the primary conduits for this transmission channel. This finding suggests that “soft” monetary policy—communicated through tone and optimism rather than rate changes—is effective primarily through institutions with the scale and sophistication to act on such signals. Consequently, policymakers must account for this heterogeneity: sentiment shocks may disproportionately influence the credit supply of the largest systemic institutions while leaving smaller community banks relatively unaffected.

Bibliography

Adrian, Tobias. 2023. "The Market Price of Risk and Macro-Financial Dynamics." *IMF Working Paper.*

Adrian, Tobias, Federico Grinberg, Nellie Liang, and Sheheryar Malik. 2018. "The Term Structure of Growth-at-Risk." *SSRN Electronic Journal.*

Anwar, Cep Jandi, Nicholas Okot, Indra Suhendra, Dwi Indriyani, and Ferry Jie. 2023. "Monetary policy, macroprudential policy, and bank risk-taking behaviour in the Indonesian banking industry." *Journal of Applied Economics.*

Bauer, Michael D., Ben S. Bernanke, and Eric Milstein. 2023. "Risk Appetite and the Risk-Taking Channel of Monetary Policy." *Journal of Economic Perspectives*, 37(1): 77–100.

Borio, Claudio, and Haibin Zhu. 2008. "Capital Regulation, Risk-Taking and Monetary Policy: A Missing Link in the Transmission Mechanism?" *Working Paper.*

Boyarchenko, Nina, Giovanni Favara, and Moritz Schularick. 2022. "Financial Stability Considerations for Monetary Policy: Empirical Evidence and Challenges." *Finance and Economics Discussion Series.*

Caballero, Ricardo J., and Alp Simsek. 2024. "Central Banks, Stock Markets, and the Real Economy." *SSRN Electronic Journal.*

Cieslak, Anna, and Andreas Schrimpf. 2019. "Non-monetary news in central bank communication." *Journal of International Economics.*

Dell'Ariccia, Giovanni, Luc Laeven, and Gustavo Suarez. 2017. "Bank Leverage and Monetary Policy's Risk-Taking Channel: Evidence from the United States." *Journal of Finance.*

Grimm, Maximilian, Òscar Jordà, Moritz Schularick, and Alan M. Taylor. 2023. "Loose Monetary Policy and Financial Instability." *NBER Working Paper Series.*

Ioannidou, Vasso, Steven Ongena, and José-Luis Peydró. 2015. "Monetary Policy, Risk-Taking, and Pricing: Evidence from a Quasi-Natural Experiment." *Review of Finance.*

Jimenez, Gabriel, Steven Ongena, José Luis Peydró, and Jesús Saurina. 2014. "Hazardous times for monetary policy: what do twenty-three million bank loans say about the effects of monetary policy on credit risk-taking?" *Econometrica.*

Kashyap, Anil K., and Jeremy C. Stein. 2023. "Monetary Policy When the Central Bank Shapes Financial-Market Sentiment." *Journal of Economic Perspectives*, 37(1): 53–76.

Kekre, Rohan, and Moritz Lenel. 2021. "Monetary Policy, Redistribution, and Risk Premia." *SSRN Electronic Journal.*

Nakamura, Emi, and Jón Steinsson. 2018. "High Frequency Identification of Monetary Non-Neutrality: The Information Effect." *Quarterly Journal of Economics.*

Paligorova, Teodora, and João A. C. Santos. 2017. "Monetary Policy and Bank Risk-Taking: Evidence from the Corporate Loan Market." *Journal of*

Financial Intermediation.

5 Appendix A: Data Construction & Harmonization

5.0.1 A.1 The "Dual-Pipeline" Ingestion Strategy

A significant barrier to longitudinal analysis of US banking data is the format change that occurred in 2001. Standard merging procedures often fail to bridge this gap, resulting in "short panels" that drop the 1980s and 90s. To resolve this, we employed two distinct extraction algorithms:

1. **The Legacy Pipeline (1984–2000):** This pipeline handles SAS Transport (`.xpt`) files. The primary challenge in this era is inconsistent date recording; dates are often omitted from the file content or stored in non-standard "days since 1960" formats. Our algorithm recovers the valid reporting quarter by parsing the original filenames (e.g., identifying `8206` as June 1982) and cross-validating with internal file metadata.
2. **The Modern Pipeline (2001–2019):** This pipeline handles the modern bulk repository, which is organized into nested directories of variable-delimited text files. The algorithm recursively scans these directories, extracts the `Schedule RC` (Balance Sheet) and `Schedule RI` (Income Statement) files, and merges them on the unique bank identifier (`IDRSSD`).

5.0.2 A.2 Variable Harmonization

Both pipelines standardize variables to a common dictionary before merging. Specifically, we prioritize "Consolidated Bank" reported values (Prefix `RCFD`) and use "Domestic Office" values (Prefix `RCON`) as a fallback for smaller banks that do not report consolidated figures. This ensures that the definition of "Total Assets" and "Commercial & Industrial Loans" remains consistent across the 35-year sample.

6 Appendix B: Replication Code

To replicate the bank dataset construction, place the raw FFIEC data folders in the root directory and execute the scripts in the following order.

Step 1: Process Legacy Data (1984–2000) Run `old_bank_data.py` to parse the SAS Transport files.

- **Input:** Raw `.xpt` files (recursive search).
- **Output:** `OLD_RAW_BANK_DATA.csv`

Step 2: Process Modern Data (2001–2019) Run `new_bank_scrape.py` to parse the modern text repositories.

- **Input:** FFIEC folders containing `Schedule RC.txt` and `Schedule CI.txt`.
- **Output:** `NEW_RAW_BANK_DATA.csv`

Step 3: Merge Run `merge_bank.py` Combine the two outputs to form final file `ALL_BANKS_MERGED.py`

To replicate the macro dataset construction, place the raw excel spreadsheet in the same directory the following scripts and execute them in the following order.

Step 1: Clean Transcripts Run `fomc_data.py` to parse the transcripts.

- **Input:** transcript `.xlsx` file.
- **Output:** `fomc_data` folder, containing text files with transcripts of each date.

Step 2: Calculate Sentiment Run `sentiment.py` to parse the txt files in the previously created fomc_data file.

- **Input:** fomc_data folder containing `meeting_yyyy-mm-dd.txt`.
- **Output:** `fomc_sentiment_data.csv`

Step 3: Scrape Effective Funds Rate Run `rate.py` to pull latest fomc data from the St. Louis Fed.

- **Input:** N/A
- **Output:** `fed_funds.csv`

Merge Data Into Final Dataset Run `merge.py` to pull latest fomc data from the St. Louis Fed.

- **Input:** `fed_funds.csv`, `ALL_BANKS_MERGED.csv`, `fomc_sentiment_data.csv`
- **Output:** `MACRO_DATA_CLEAN.csv`, `master_merged_dataset.csv`

6.0.1 B.1 Legacy Parser (`old_bank_data.py`)

```
import pandas as pd
import os
import glob
import re
from pandas.tseries.offsets import QuarterEnd


def build_xpt_dataset_smart():
    print(" - - - STARTING OLD ERA (XPT) DATA BUILDER V2 (CORRECTED) - - -")

    # 1. Define SAS Variable Names (Old Era)
    # RSSD9001: Entity ID
    # RSSD9999: Report Date (Standard internal date variable)

    # ASSETS:
    # RCFD2170: Total Assets (Consolidated)
    # RCON2170: Total Assets (Domestic)

    # LOANS:
    # RCFD1763: Total C&I Loans (Consolidated) - CRITICAL for large banks
    # RCFD1766: C&I Loans to U.S. Addressees (Consolidated)
    # RCON1766: C&I Loans (Domestic)

    id_col = 'RSSD9001'
    date_col_internal = 'RSSD9999'

    # We define the columns we WANT to find.
    # Note: We don't filter strictly on read because read_sas reads the whole file anyway.
    target_cols = [
        'RSSD9001', 'RSSD9999',
        'RCFD2170', 'RCON2170',
        'RCFD1763', 'RCFD1766', 'RCON1766'
    ]

    root_dir = "."
    xpt_files = glob.glob(os.path.join(
        root_dir, "**", "*.xpt"), recursive=True)

    print(f"Found {len(xpt_files)} .xpt files. Processing...")

    master_list = []

    for file_path in xpt_files:
        filename = os.path.basename(file_path)
```

```
date_obj = None

# - - - STRATEGY 1: PARSE FILENAME (YYMM) - - -
match = re.search(
    r'([0 -9]{2})([0 -9]{2})\.xpt$', filename, re.IGNORECASE)

if match:
    yy = int(match.group(1))
    mm = int(match.group(2))

    # Year Pivot Logic (Window: 1950 -2049)
    if yy > 50:
        full_year = 1900 + yy
    else:
        full_year = 2000 + yy

    try:
        # Snap to QuarterEnd
        date_obj = pd.Timestamp(
            year=full_year, month=mm, day=1) + QuarterEnd(0)
    except:
        print(f"  Warning: Invalid date in filename {filename}")

# - - - STRATEGY 2: READ AND EXTRACT - - -
try:
    # Read SAS file
    df = pd.read_sas(file_path, format='xport')

    # Convert columns to Uppercase
    df.columns = [c.upper() for c in df.columns]

    # If date not found in filename, try internal SAS date
    if date_obj is None:
        if date_col_internal in df.columns:
            # SAS Dates are days since 1960 -01 -01
            sas_date_val = df[date_col_internal].mode()[0]
            date_obj = pd.to_datetime(
                '1960 -01 -01') + pd.to_timedelta(sas_date_val, unit='D')
            print(
                f"  Extracted internal date {date_obj.date()} from {filename}")
        else:
            print(f"Skipping {filename}: Could not determine date.")
            continue

    # Standardize ID
    if id_col not in df.columns:
        continue

    df[id_col] = pd.to_numeric(df[id_col], errors='coerce')
    df.rename(columns={id_col: 'bank_id'}, inplace=True)

    # - - - STANDARDIZE ASSETS - - -
    # Priority: RCFD (Consolidated) > RCON (Domestic)
    # Create a base series of NaNs
    df['assets'] = pd.NA

    if 'RCON2170' in df.columns:
        df['assets'] = df['RCON2170']

    if 'RCFD2170' in df.columns:
        # combine_first: if 'RCFD' is present, use it; else keep 'RCON'
        df['assets'] = df['RCFD2170'].combine_first(df['assets'])

    # - - - STANDARDIZE C&I LOANS - - -
    # Priority: RCFD1763 (Total Global) > RCFD1766 (US Global) > RCON1766 (Domestic

    # Start with Domestic as baseline
    loan_series = df.get('RCON1766', pd.Series([pd.NA]*len(df)))

    # Update with US Global if available (fills gaps or overwrites depending on pre
    # here we want RCFD to supersede RCON if the bank reports RCFD)
    if 'RCFD1766' in df.columns:
        loan_series = df['RCFD1766'].combine_first(loan_series)

    # Update with Total Global (Best Metric)
    if 'RCFD1763' in df.columns:
        loan_series = df['RCFD1763'].combine_first(loan_series)

    df['ci_loans'] = loan_series

    # - - - FINAL DATA PREP - - -
    df['date'] = date_obj

    # Keep only valid data columns
    df = df[['bank_id', 'date', 'assets', 'ci_loans']]
    df = df.dropna(subset=['bank_id'])

    master_list.append(df)
    print(
        f"Processed {filename}: {len(df)} banks. Date: {date_obj.date()}")

except Exception as e:
```

```
            print(f"  Error reading {filename}: {e}")

    # - - - COMPILE - - -
    if not master_list:
        print("No data found.")
        return

    print("Compiling Old Era Master Dataset...")
    full_old_df = pd.concat(master_list, ignore_index=True)

    # Final Polish
    full_old_df['assets'] = pd.to_numeric(
        full_old_df['assets'], errors='coerce')
    full_old_df['ci_loans'] = pd.to_numeric(
        full_old_df['ci_loans'], errors='coerce')

    # Remove rows with 0 assets to avoid DivideByZero
    full_old_df = full_old_df[full_old_df['assets'] > 0]

    # Calculate Risk Taking
    full_old_df['risk_taking'] = full_old_df['ci_loans'] / \
        full_old_df['assets']

    # Final cleanup of NaNs created by risk calc
    full_old_df = full_old_df.dropna(subset=['risk_taking'])

    output_filename = 'OLD_RAW_BANK_DATA.csv'
    full_old_df.to_csv(output_filename, index=False)

    print(f"\nSUCCESS! Processed {len(full_old_df)} rows from Old Era.")
    print(f"Saved to: {output_filename}")


if __name__ == "__main__":
    build_xpt_dataset_smart()
```

6.0.2 B.2 Modern Parser (`new_bank_data.py`)

```
import pandas as pd
import os
import glob
import re


def build_bank_dataset_recursive():
    print(" - - - STARTING BANK DATA BUILDER - - -")
```

```
# 1. Setup Columns
# IDRSSD: Bank ID
# RCFD2170: Total Assets (Consolidated)
# RCON2170: Total Assets (Domestic - fallback)

# LOANS:
# RCFD1763: Total C&I Loans (Consolidated - Includes Foreign & Domestic)
# RCFD1766: C&I Loans to U.S. Addressees (Consolidated - Partial)
# RCON1766: C&I Loans (Domestic)

asset_cols = ['IDRSSD', 'RCFD2170', 'RCON2170']
# Added RCFD1763 for accurate Total C&I on global banks
loan_cols = ['IDRSSD', 'RCFD1763', 'RCFD1766', 'RCON1766']

all_quarters = []

root_dir = "."
subfolders = [f.path for f in os.scandir(root_dir) if f.is_dir()]

print(f"Found {len(subfolders)} folders. Scanning for data files...")

for folder in subfolders:
    # 2. Find Schedule RC (Assets) and Schedule RCCI (Loans)
    # CRITICAL FIX: The FFIEC file for loans is named "Schedule RCCI", not "Schedule CI'
    rc_files = glob.glob(os.path.join(folder, "*Schedule RC *.txt"))
    ci_files = glob.glob(os.path.join(folder, "*Schedule RCCI *.txt"))

    if not rc_files or not ci_files:
        # Silent skip or debug print if needed
        continue

    path_rc = rc_files[0]
    path_ci = ci_files[0]

    # Extract Date
    date_match = re.search(r'(\d{8})', os.path.basename(path_rc))
    if not date_match:
        print(f"Skipping {folder}: Could not determine date.")
        continue

    date_str = date_match.group(1)

    try:
        # 3. Read ASSETS (Schedule RC)
        df_rc = pd.read_csv(path_rc, sep='\t',
                            skiprows=0, low_memory=False)
        existing_asset_cols = [c for c in asset_cols if c in df_rc.columns]
        df_rc = df_rc[existing_asset_cols]

        # 4. Read LOANS (Schedule RC -C Part I)
        df_ci = pd.read_csv(path_ci, sep='\t',
                            skiprows=0, low_memory=False)
        existing_loan_cols = [c for c in loan_cols if c in df_ci.columns]
        df_ci = df_ci[existing_loan_cols]

        # 5. Merge
        if 'IDRSSD' not in df_rc.columns or 'IDRSSD' not in df_ci.columns:
            print(f"Skipping {date_str}: Missing IDRSSD column.")
            continue

        df_quarter = pd.merge(df_rc, df_ci, on='IDRSSD', how='inner')
        df_quarter['date'] = pd.to_datetime(date_str, format='%m%d%Y')

        # 6. Standardize Assets
        if 'RCFD2170' in df_quarter.columns:
            df_quarter['assets'] = df_quarter['RCFD2170'].fillna(
                df_quarter.get('RCON2170', 0))
        elif 'RCON2170' in df_quarter.columns:
            df_quarter['assets'] = df_quarter['RCON2170']
        else:
            df_quarter['assets'] = pd.NA

        # 7. Standardize C&I Loans (Prioritize Total Global -> US Global -> Domestic)
        # RCFD1763 = Total C&I (Consolidated)
        # RCFD1766 = C&I to US Addressees (Consolidated)
        # RCON1766 = Total C&I (Domestic)

        df_quarter['ci_loans'] = 0  # Default

        if 'RCFD1763' in df_quarter.columns:
            # Primary for large banks
            df_quarter['ci_loans'] = df_quarter['RCFD1763']

        # Fill gaps where RCFD1763 might be missing but RCFD1766 exists
        if 'RCFD1766' in df_quarter.columns:
            df_quarter['ci_loans'] = df_quarter['ci_loans'].fillna(
                df_quarter['RCFD1766'])
            # If we initialized with 0, fillna won't work on 0s, so we use logic:
            # If 1763 was missing, column is 0 (or NaN if we didn't init).
            # Better approach: Coalesce.
```

```
            # Re -doing clean coalescing logic:
            # Create a temporary series for the best available C&I data

            # Start with Domestic (RCON1766) as baseline
            temp_loans = df_quarter.get(
                'RCON1766', pd.Series([pd.NA]*len(df_quarter)))

            # Overwrite with RCFD1766 (US Addressees) if available
            if 'RCFD1766' in df_quarter.columns:
                temp_loans = df_quarter['RCFD1766'].combine_first(temp_loans)

            # Overwrite with RCFD1763 (Total Consolidated) if available - Best Metric
            if 'RCFD1763' in df_quarter.columns:
                temp_loans = df_quarter['RCFD1763'].combine_first(temp_loans)

            df_quarter['ci_loans'] = temp_loans

            # 8. Clean up
            df_quarter.rename(columns={'IDRSSD': 'bank_id'}, inplace=True)
            df_quarter = df_quarter[['bank_id', 'date', 'assets', 'ci_loans']]

            # Ensure numeric
            df_quarter['assets'] = pd.to_numeric(
                df_quarter['assets'], errors='coerce')
            df_quarter['ci_loans'] = pd.to_numeric(
                df_quarter['ci_loans'], errors='coerce')

            all_quarters.append(df_quarter)
            print(f"Processed {date_str}: {len(df_quarter)} banks.")

        except Exception as e:
            print(f"Error reading {date_str} in {folder}: {e}")

    # 9. Compile and Save
    if not all_quarters:
        print("CRITICAL: No data found. Ensure folders are named correctly (e.g. '03312008')
        return

    print("Compiling Master Dataset...")
    master_df = pd.concat(all_quarters, ignore_index=True)

    # Calculate Risk Taking
    master_df['risk_taking'] = master_df['ci_loans'] / master_df['assets']

    # Filter
    master_df = master_df.dropna(subset=['assets', 'risk_taking'])
    master_df = master_df[master_df['assets'] > 0]

    output_filename = 'NEW_RAW_BANK_DATA.csv'
    master_df.to_csv(output_filename, index=False)

    print(f"\nSUCCESS! Built dataset with {len(master_df)} rows.")
    print(f"Saved to: {output_filename}")

    # Diagnostics
    count_2008 = len(master_df[master_df['date'].dt.year == 2008])
    print(f"Diagnostics: Rows found for 2008: {count_2008}")


if __name__ == "__main__":
    build_bank_dataset_recursive()
```

6.0.3 B.3 Bank Merge (merge_bank.py)

```
import pandas as pd
import os
import glob


def find_file(filename, search_path="."):
    """Recursively searches for a file in the directory tree."""
    print(f"  Searching for '{filename}'...")
    # Recursive glob search
    matches = glob.glob(os.path.join(
        search_path, "**", filename), recursive=True)
    if matches:
        print(f"  -> Found at: {matches[0]}")
        return matches[0]
    return None


def run_smart_merge():
    print("==================================================")
    print("       SMART BANK DATA MERGER                     ")
    print("==================================================")
    print(f"Current Working Directory: {os.getcwd()}\n")

    # - - - STEP 1: LOCATE FILES - - -
    print("Step 1: Locating Input Files...")

    path_new = find_file("NEW_RAW_BANK_DATA.csv")
    path_old = find_file("OLD_RAW_BANK_DATA.csv")
```

```
if not path_new and not path_old:
    print("\nCRITICAL ERROR: Could not find ANY bank data files.")
    print("You must run the builder scripts first:")
    print("  1. Run 'build_bank_data.py' (for Modern data)")
    print("  2. Run 'build_old_data_v2.py' (for Old data)")
    return

# - - - STEP 2: LOAD DATA - - -
print("\nStep 2: Loading Data...")
dfs = []

if path_new:
    try:
        df_new = pd.read_csv(path_new)
        print(f"  -> Loaded Modern Data: {len(df_new):,} rows")
        dfs.append(df_new)
    except Exception as e:
        print(f"  -> Error reading Modern Data: {e}")

if path_old:
    try:
        df_old = pd.read_csv(path_old)
        print(f"  -> Loaded Old Data: {len(df_old):,} rows")
        dfs.append(df_old)
    except Exception as e:
        print(f"  -> Error reading Old Data: {e}")

# - - - STEP 3: MERGE - - -
print("\nStep 3: Merging...")
if not dfs:
    print("  -> No data loaded. Exiting.")
    return

merged_df = pd.concat(dfs, ignore_index=True)

# Standardize Date
merged_df['date'] = pd.to_datetime(merged_df['date'])

# Deduplicate (Priority to Modern Data if overlaps exist)
merged_df = merged_df.sort_values(['date', 'bank_id'])
before_dedup = len(merged_df)
merged_df = merged_df.drop_duplicates(
    subset=['bank_id', 'date'], keep='last')
print(
    f"  -> Deduplication removed {before_dedup - len(merged_df):,} duplicates.")
    # - - - STEP 4: SAVE - - -
    output_filename = "ALL_BANKS_MERGED.csv"
    merged_df.to_csv(output_filename, index=False)

    print("\n" + "="*50)
    print(f"SUCCESS! Created '{output_filename}'")
    print(f"Total Rows: {len(merged_df):,}")
    print(f"Location: {os.path.abspath(output_filename)}")
    print("="*50)


if __name__ == "__main__":
    run_smart_merge()
```

6.0.4 B.4 FOMC Data Retrieval (data_fomc.py)

```
#!/usr/bin/env python3
"""
Download and prepare FOMC transcripts for sentiment analysis.

 - Source: Miguel Acosta, "FOMC Communications Data"
  https://www.acostamiguel.com/data/fomc_data.html
"""

import argparse
import logging
import pathlib
import sys

import requests
import pandas as pd

TRANSCRIPTS_URL = "https://www.acostamiguel.com/data/FOMC/transcripts.xlsx"


def download_file(url: str, dest: pathlib.Path, force: bool = False) -> pathlib.Path:
    """Download file with simple resume logic (skip if exists unless - -force)."""
    dest = pathlib.Path(dest)
    if dest.exists() and not force:
        logging.info(
            "File %s already exists; skipping download. Use - -force to re -download.",
            dest,
        )
        return dest
    dest.parent.mkdir(parents=True, exist_ok=True)
    logging.info("Downloading %s -> %s", url, dest)

    with requests.get(url, stream=True, timeout=60) as r:
        r.raise_for_status()
        tmp = dest.with_suffix(dest.suffix + ".part")
        with open(tmp, "wb") as f:
            for chunk in r.iter_content(chunk_size=8192):
                if chunk:
                    f.write(chunk)
        tmp.replace(dest)

    logging.info("Finished download.")
    return dest


def split_meetings_to_txt(df: pd.DataFrame, out_dir: pathlib.Path) -> None:
    """Group utterances by meeting date and write one text file per meeting."""
    out_dir = pathlib.Path(out_dir)
    out_dir.mkdir(parents=True, exist_ok=True)

    # Choose a date column and normalize to datetime (handles raw YYYYMMDD ints).
    date_col = "meeting_date" if "meeting_date" in df.columns else "date"
    if date_col not in df.columns:
        raise KeyError("Expected a 'date' or 'meeting_date' column in the transcript data.")

    dates = df[date_col]
    if pd.api.types.is_integer_dtype(dates):
        df["meeting_date"] = pd.to_datetime(dates.astype(str), format="%Y%m%d")
    else:
        df["meeting_date"] = pd.to_datetime(dates)

    # Ensure we have ordering columns for reproducible text output.
    sort_cols = [c for c in ["sequence", "n_utterance"] if c in df.columns]

    for date, g in df.groupby("meeting_date"):
        date_str = pd.to_datetime(date).strftime("%Y -%m -%d")
        g_sorted = g.sort_values(sort_cols)

        text = "\n\n".join(g_sorted["text"].astype(str))

        out_path = out_dir / f"meeting_{date_str}.txt"
        out_path.write_text(text, encoding="utf -8")
        logging.info("Wrote %s (%d utterances)", out_path.name, len(g_sorted))
```

```
def main(argv=None):
    parser = argparse.ArgumentParser(
        description="Download and preprocess FOMC transcripts for sentiment analysis."
    )
    parser.add_argument(
        " - -out -dir",
        default="fomc_data",
        help="Root output directory (default: %(default)s)",
    )
    parser.add_argument(
        " - -force",
        action="store_true",
        help="Force re -download of source files even if they exist.",
    )
    parser.add_argument(
        " - -xlsx",
        help="Use an existing transcripts.xlsx instead of downloading.",
    )
    parser.add_argument(
        " - -meetings -dir",
        default="meetings_txt_all",
        help="Subdirectory under processed/ for per -meeting text files (default: %(default)
    )

    args = parser.parse_args(argv)

    logging.basicConfig(
        level=logging.INFO,
        format="%(asctime)s %(levelname)s %(message)s",
    )

    out_root = pathlib.Path(args.out_dir)
    raw_dir = out_root / "raw"
    processed_dir = out_root / "processed"

    # 1. Download (or reuse) the transcripts Excel
    if args.xlsx:
        xlsx_path = pathlib.Path(args.xlsx)
        if not xlsx_path.exists():
            raise FileNotFoundError(f"Provided - -xlsx path does not exist: {xlsx_path}")
        logging.info("Using existing transcripts file at %s", xlsx_path)
    else:
        xlsx_path = download_file(
            TRANSCRIPTS_URL,
            raw_dir / "fomc_transcripts.xlsx",
            force=args.force,
        )

    # 2. Read into pandas and save a CSV version
    logging.info("Reading %s", xlsx_path)
    df = pd.read_excel(xlsx_path)

    tables_dir = processed_dir / "tables"
    tables_dir.mkdir(parents=True, exist_ok=True)

    csv_path = tables_dir / "fomc_transcripts.csv"
    df.to_csv(csv_path, index=False)
    logging.info("Saved full transcripts table to %s", csv_path)

    # 3. Write one text file per meeting
    meetings_txt_dir = processed_dir / args.meetings_dir
    split_meetings_to_txt(df, meetings_txt_dir)

    logging.info("All done. Meeting -level text lives in %s", meetings_txt_dir)


if __name__ == "__main__":
    main(sys.argv[1:])
```

6.0.5 B.5 Sentiment Analysis (sentiment.py)

```
import os
import glob
import pandas as pd
import pysentiment2 as ps

# 1. Setup
# Folder containing the text meetings. We try common locations so the script works
# whether you generated files with fomc3.py (meetings_txt_all) or an earlier step (meetings_
candidate_dirs = [
    'fomc_data/processed/meetings_txt_all',
    'fomc_data/processed/meetings_txt',
    # fallback to absolute path if used elsewhere
    '/Users/hfh/Downloads/meetings_txt',
]
folder_path = next(
    (d for d in candidate_dirs if os.path.isdir(d)), candidate_dirs[0])
lm = ps.LM()  # Initialize the Financial Dictionary
results = []

# 2. The Loop
# glob.glob grabs all .txt files in the folder
```

```
file_list = glob.glob(os.path.join(folder_path, "*.txt"))
if not file_list:
    raise FileNotFoundError(
        f"No .txt files found in '{folder_path}'. "
        "Check the path or run fomc3.py to generate the meeting text files."
    )

print(f"Found {len(file_list)} files. Starting analysis...")

for file_path in file_list:
    try:
        # Get filename (e.g., "meeting_1976 -03 -29.txt")
        filename = os.path.basename(file_path)

        # Extract Date from filename
        # Assumes format "meeting_YYYY -MM -DD.txt"
        date_str = filename.replace('meeting_', '').replace('.txt', '')

        # Read the text file
        with open(file_path, 'r', encoding='utf -8', errors='ignore') as f:
            text = f.read()

        # 3. Analyze Sentiment
        tokens = lm.tokenize(text)
        score = lm.get_score(tokens)

        # Calculate Net Sentiment: (Pos - Neg) / (Pos + Neg)
        net_sentiment = score['Polarity']

        # Store data
        results.append({
            'date': date_str,
            'net_sentiment': net_sentiment,
            'positive_count': score['Positive'],
            'negative_count': score['Negative'],
            'word_count': len(tokens),
            'filename': filename
        })

    except Exception as e:
        print(f"Error reading {filename}: {e}")

# 4. Save Results
df = pd.DataFrame(results)

# Convert 'date' column to actual datetime objects for sorting/merging later
```

```
if df.empty:
    raise ValueError("No sentiment results produced; verify the input files.")

df['date'] = pd.to_datetime(df['date'])
df = df.sort_values('date')

print("Analysis Complete.")
print(df.head())

# Save to CSV for the next step of your paper
output_csv = 'fomc_sentiment_data.csv'
df.to_csv(output_csv, index=False)
print(f"Saved results to {output_csv}")
```

6.0.6 B.6 Funds Rate Retrieval (rate.py)

```
import pandas as pd


def scrape_fed_funds():
    # URL for Daily Federal Funds Rate (DFF) from St. Louis Fed
    # Use 'FEDFUNDS' instead of 'DFF' if you want monthly averages
    url = "https://fred.stlouisfed.org/graph/fredgraph.csv?id=DFF"

    print(f"Downloading data from {url}...")

    try:
        # Read CSV directly from the URL
        df = pd.read_csv(url)

        # Rename columns as requested
        df.columns = ['date', 'rate']

        # Convert date column to datetime objects for filtering
        df['date'] = pd.to_datetime(df['date'])

        # Filter for dates starting from 1982 -01 -01
        df = df[df['date'] >= '1976 -01 -01']

        # Save to CSV without the index number
        output_file = 'fed_funds.csv'
        df.to_csv(output_file, index=False)

        print(f"Success! Saved {len(df)} rows to {output_file}")
        print(f"Range: {df['date'].min().date()} to {df['date'].max().date()}")
    except Exception as e:
        print(f"Error: {e}")


if __name__ == "__main__":
    scrape_fed_funds()
```

6.0.7 B.7 Final Dataset Merge (`merge.py`)

```
import glob
import os
from pathlib import Path

import pandas as pd


def pick_file(csvs, label):
    """Tiny helper to pick a CSV by index with a friendly label."""
    print(f"\nWhich file contains the {label}? (Enter the number)")
    try:
        idx = int(input("Selection: "))
        return csvs[idx]
    except Exception:
        print("Invalid selection.")
        return None


def build_macro_golden_source():
    print(" - - - MACRO + BANK DATA BUILDER - - -")
    print("We need to identify your raw source files.")

    # 1. List all CSVs in the folder (and parent folder)
    csvs = glob.glob("*.csv") + glob.glob("../*.csv")

    if not csvs:
        print("CRITICAL: No CSV files found. Please move your raw Sentiment, Fed Funds, and
        return

    print("\nAvailable CSV files:")
    for i, f in enumerate(csvs):
        print(f"[{i}] {f}")

    # 2. Ask User for Sentiment File
    sent_file = pick_file(csvs, "SENTIMENT data")
    if not sent_file:
        return
```

```
# 3. Ask User for Fed Funds File
print("(Enter same number if they are in one file)")
fed_file = pick_file(csvs, "FED FUNDS RATE")
if not fed_file:
    return

# 4. Ask User for Bank Panel File
bank_file = pick_file(csvs, "BANK PANEL (assets, risk_taking, etc.)")
if not bank_file:
    return

# 5. Process Sentiment
print(f"\nProcessing Sentiment from {sent_file}...")
df_sent = pd.read_csv(sent_file)

# Auto -detect columns (Looking for 'sentiment' or similar)
sent_col = next((c for c in df_sent.columns if 'sentiment' in c.lower()), None)
date_col_s = next((c for c in df_sent.columns if 'date' in c.lower()), None)

if not sent_col or not date_col_s:
    print(f"Error: Could not find 'date' or 'sentiment' column in {sent_file}")
    print(f"Columns found: {list(df_sent.columns)}")
    return

df_sent = df_sent[[date_col_s, sent_col]].rename(
    columns={date_col_s: 'date', sent_col: 'fed_sentiment'}
)
df_sent['date'] = pd.to_datetime(df_sent['date'])

# 6. Process Fed Funds
print(f"Processing Fed Funds from {fed_file}...")
df_fed = pd.read_csv(fed_file)

# Auto -detect columns
fed_col = next((c for c in df_fed.columns if 'fund' in c.lower() or 'rate' in c.lower()
date_col_f = next((c for c in df_fed.columns if 'date' in c.lower()), None)

if not fed_col or not date_col_f:
    print(f"Error: Could not find 'date' or 'rate' column in {fed_file}")
    return

df_fed = df_fed[[date_col_f, fed_col]].rename(
    columns={date_col_f: 'date', fed_col: 'fed_funds_rate'}
)
df_fed['date'] = pd.to_datetime(df_fed['date'])
```

```
# 7. Merge & Aggregate to Quarter
print("Merging and aggregating macro data to quarterly level...")

macro_df = pd.merge(df_sent, df_fed, on='date', how='outer')
macro_df['quarter_key'] = macro_df['date'].dt.to_period('Q').astype(str)

macro_clean = (
    macro_df.groupby('quarter_key')[['fed_sentiment', 'fed_funds_rate']]
    .mean()
    .reset_index()
)

macro_output = 'MACRO_DATA_CLEAN.csv'
macro_clean.to_csv(macro_output, index=False)
print(f"\nSUCCESS! Created '{macro_output}' with {len(macro_clean)} quarters.")

# 8. Load Bank Panel and merge with macro to build regression -ready dataset
print(f"\nProcessing Bank Panel from {bank_file}...")
df_bank = pd.read_csv(bank_file)

required_bank_cols = {'bank_id', 'date', 'assets', 'risk_taking'}
missing = required_bank_cols - set(df_bank.columns)
if missing:
    print(f"Error: Bank file missing required columns: {missing}")
    return

df_bank['date'] = pd.to_datetime(df_bank['date'])
df_bank['quarter_key'] = df_bank['date'].dt.to_period('Q').astype(str)

merged = pd.merge(df_bank, macro_clean, on='quarter_key', how='left')

# Keep core variables regression.py expects
cols_order = [
    'bank_id', 'date', 'assets', 'ci_loans', 'risk_taking',
    'fed_sentiment', 'fed_funds_rate'
]
# ci_loans may be missing, so include if present
cols_order = [c for c in cols_order if c in merged.columns]
merged = merged[cols_order + [c for c in merged.columns if c not in cols_order]]

output = 'master_merged_dataset.csv'
merged.to_csv(output, index=False)

# Write uppercase alias for backwards compatibility.
Path('MASTER_MERGED_DATASET.csv').write_bytes(Path(output).read_bytes())
```

```
    print(f"\nSUCCESS! Created '{output}' with {len(merged)} rows.")
    print("Sample rows:")
    print(merged.head())


if __name__ == "__main__":
    build_macro_golden_source()
```

---

7 Appendix C: Regression Code

```
import warnings
from pathlib import Path

import numpy as np
import pandas as pd
import statsmodels.api as sm
from linearmodels.panel import PanelOLS

warnings.simplefilter(action='ignore', category=FutureWarning)


def run_robust_regression():
    print(" - - - [1] DATA PREPARATION & CLEANING - - -")
    dataset_paths = [
        Path('master_merged_dataset.csv'),
        Path('MASTER_MERGED_DATASET.csv'),
    ]
    df = None
    for path in dataset_paths:
        if path.exists():
            df = pd.read_csv(path)
            print(f"Loaded dataset from {path}")
            break
    if df is None:
        print("MASTER dataset not found. Please run merge.py first.")
        return

    df['date'] = pd.to_datetime(df['date'])

    # - - - ADDRESSING POINT 9: Drop Short Panels - - -
    # Banks with fewer than 8 observations (2 years) do not contribute enough within -varia
```

```
counts = df.groupby('bank_id')['date'].count()
valid_banks = counts[counts >= 8].index
df = df[df['bank_id'].isin(valid_banks)].copy()
print(
    f" > Dropped short -lived entities. Banks remaining: {df['bank_id'].nunique():,}")

# - - - ADDRESSING POINT 1: Orthogonalization - - -
# Isolate pure sentiment from the rate level
ortho_data = df[['fed_sentiment', 'fed_funds_rate']].dropna()
X_ortho = sm.add_constant(ortho_data['fed_funds_rate'])
model_ortho = sm.OLS(ortho_data['fed_sentiment'], X_ortho).fit()
df.loc[ortho_data.index, 'sentiment_shock'] = model_ortho.resid

# - - - ADDRESSING POINT 2: Standardization - - -
# Standardize shock to 1 SD for interpretability
df['sentiment_shock_std'] = df['sentiment_shock'] / \
    df['sentiment_shock'].std()

# - - - ADDRESSING POINT 4 & 10: Functional Form & Scaling - - -
# Log Assets
df['log_assets'] = np.log(df['assets'].replace(0, np.nan))

# CENTER Log Assets.
# This is crucial. Now 'log_assets_centered' = 0 means "Average Sized Bank".
# This fixes multicollinearity by orthogonalizing the interaction from the main effect.
df['log_assets_centered'] = df['log_assets'] - df['log_assets'].mean()

# Sort for Lagging
df = df.sort_values(['bank_id', 'date'])

# - - - ADDRESSING POINT 5: Dynamic Structure - - -
# We MUST include lags of the dependent variable (Risk Taking)
# We also lag controls to mitigate simultaneity (Point 8)
vars_to_lag = ['risk_taking', 'sentiment_shock_std',
               'fed_funds_rate', 'log_assets_centered']

for var in vars_to_lag:
    df[f'{var}_lag'] = df.groupby('bank_id')[var].shift(1)

# - - - ADDRESSING POINT 3: Continuous Interaction - - -
# Instead of a dummy, we use Size as a continuous moderator.
# Interaction = (Last Quarter's Shock) * (Last Quarter's Size)
df['shock_x_size'] = df['sentiment_shock_std_lag'] * \
    df['log_assets_centered_lag']

# - - - ESTIMATION - - -
```

```
    print(" - - - [2] ESTIMATING DYNAMIC PANEL MODEL - - -")

    df = df.set_index(['bank_id', 'date'])

    reg_df = df.dropna(subset=[
        'risk_taking', 'risk_taking_lag',
        'sentiment_shock_std_lag', 'shock_x_size',
        'fed_funds_rate_lag', 'log_assets_centered_lag'
    ])

    exog_vars = [
        'const',
        'risk_taking_lag',           # Dynamics (Fixes Point 5)
        'sentiment_shock_std_lag',   # Main Effect (at mean size)
        # Interaction (Does Size change sensitivity?)
        'shock_x_size',
        'log_assets_centered_lag',   # Control for Size
        'fed_funds_rate_lag'         # Control for Rate
    ]

    reg_df['const'] = 1

    # ADDRESSING POINT 7: Fixed Effects
    mod = PanelOLS(reg_df['risk_taking'],
                   reg_df[exog_vars], entity_effects=True)

    # ADDRESSING POINT 6: Cluster Level
    # Explicitly clustering by Entity (Bank) and Time (Date)
    res = mod.fit(cov_type='clustered', cluster_entity=True, cluster_time=True)

    print(res)

    # - - - DIAGNOSTIC CHECKS - - -
    print("\n - - - DIAGNOSTICS - - -")
    print(f"Within R -squared: {res.rsquared_within:.4f}")
    print(
        f"Autocorrelation check (LDV coeff): {res.params['risk_taking_lag']:.4f}")


if __name__ == "__main__":
    run_robust_regression()
```

8 Appendix D: Graph Code

All graphs should be run with `master_merged_dataset.csv` in the same directory.

9 Figure 1: Time Series of Sentiment and Fed Funds Rate

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import numpy as np

# Load your dataset
df = pd.read_csv('master_merged_dataset.csv')

# Pre -processing: Ensure date is datetime
df['date'] = pd.to_datetime(df['date'])


def plot_macro_time_series(df):
    # Aggregate to time -series level (since macro vars are constant across banks per quarte
    macro_df = df.groupby(
        'date')[['fed_sentiment', 'fed_funds_rate']].mean().reset_index()

    fig, ax1 = plt.subplots(figsize=(12, 6))

    # Plot Sentiment (Left Axis)
    color = 'tab:blue'
    ax1.set_xlabel('Date')
    ax1.set_ylabel('Raw Sentiment (fed_sentiment)', color=color)
    ax1.plot(macro_df['date'], macro_df['fed_sentiment'],
             color=color, label='Fed Sentiment')
    ax1.tick_params(axis='y', labelcolor=color)
    ax1.grid(True, linestyle=' - -', alpha=0.6)

    # Create a second y -axis for Fed Funds Rate
    ax2 = ax1.twinx()
    color = 'tab:red'
    ax2.set_ylabel('Fed Funds Rate (%)', color=color)
    ax2.plot(macro_df['date'], macro_df['fed_funds_rate'],
             color=color, label='Fed Funds Rate')
    ax2.tick_params(axis='y', labelcolor=color)
    plt.title('Time Series: Raw Sentiment and Fed Funds Rate',
              fontsize=14, pad=15)
    fig.tight_layout()
    plt.show()


plot_macro_time_series(df)
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[1], line 1
 - - - -> 1 import pandas as pd
      2 import matplotlib.pyplot as plt
      3 import matplotlib.dates as mdates

ModuleNotFoundError: No module named 'pandas'
```

9.1 Figure 2: Data Construction Diagram (FFIEC Changes)

```
import matplotlib.pyplot as plt


def plot_data_construction_diagram():
    fig, ax = plt.subplots(figsize=(10, 4))
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.axis("off")

    legacy_text = (
        "Legacy FFIEC Data (Pre -2001)\n"
        " \cdot SAS XPT fixed -width files\n"
        " \cdot Cryptic variable codes\n"
        " \cdot Nonstandard date strings\n"
        " \cdot Blank missing values\n"
        " \cdot Irregular reporting dates\n"
        " \cdot Incomplete bank identifiers"
    )

    modern_text = (
        "Modern FFIEC Data (Post -2001)\n"
        " \cdot CSV/TXT delimited files\n"
        " \cdot Consistent variable labels\n"
        " \cdot Standard quarter -end dates\n"
        " \cdot Explicit NA markers\n"
        " \cdot Uniform reporting structure\n"
        " \cdot Full RSSD + cert IDs"
    )

    ax.text(
        0.05, 0.5, legacy_text,
        fontsize=11, va="center", ha="left", family="monospace"
    )

    ax.text(
        0.60, 0.5, modern_text,
        fontsize=11, va="center", ha="left", family="monospace"
    )

    # Arrow using annotate (much better scaling control)
    ax.annotate(
        "",
        xy=(0.58, 0.5),
        xytext=(0.42, 0.5),
        arrowprops=dict(
            arrowstyle=" ->",
            lw=2,
            mutation_scale=12,
            color="#6FA8DC"
        )
    )

    plt.tight_layout()
    plt.show()


plot_data_construction_diagram()
```

9.2 Figure 3: Binscatter of Risk Taking vs. Sentiment

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import numpy as np

# Load your dataset
df = pd.read_csv('master_merged_dataset.csv')

# Pre -processing: Ensure date is datetime
df['date'] = pd.to_datetime(df['date'])

def plot_binscatter_risk_sentiment(df):
    # Bin the sentiment into deciles to reduce noise
    df['sentiment_bin'] = pd.qcut(df['fed_sentiment'], q=20, labels=False)
    binned_data = df.groupby('sentiment_bin')[
        ['fed_sentiment', 'risk_taking']].mean()

    plt.figure(figsize=(10, 6))
    sns.regplot(x=binned_data['fed_sentiment'], y=binned_data['risk_taking'], scatter_kws={
                's': 100}, line_kws={'color': 'red'})

    plt.xlabel('Fed Sentiment (Binned)')
    plt.ylabel('Average Bank Risk Taking')
    plt.title('Correlation: Fed Sentiment vs. Bank Risk Taking', fontsize=14)
    plt.grid(True, linestyle=' - -', alpha=0.5)
    plt.show()


plot_binscatter_risk_sentiment(df)
```

9.3 Figure 4: The "Identification" Plot (Risk Taking by Bank Size)

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import numpy as np

# Load your dataset
df = pd.read_csv('master_merged_dataset.csv')

# Pre -processing: Ensure date is datetime
df['date'] = pd.to_datetime(df['date'])

def plot_heterogeneity_by_size(df):
    # Create size quintiles based on Assets
    df['size_quintile'] = df.groupby('date')['assets'].transform(
        lambda x: pd.qcut(x, 5, labels=['Smallest', '2', '3', '4', 'Largest']))

    # Filter to just Smallest vs Largest for clarity
    subset = df[df['size_quintile'].isin(['Smallest', 'Largest'])]

    # Aggregate over time
    time_series = subset.groupby(['date', 'size_quintile'])[
        'risk_taking'].mean().reset_index()

    plt.figure(figsize=(12, 6))
    sns.lineplot(data=time_series, x='date', y='risk_taking',
                 hue='size_quintile', style='size_quintile')

    plt.title('Risk Taking over Time: Small vs. Large Banks', fontsize=14)
    plt.ylabel('Average Risk Taking')
    plt.xlabel('Date')
    plt.legend(title='Bank Size')
    plt.grid(True, alpha=0.3)
    plt.show()


plot_heterogeneity_by_size(df)
```

9.4 Figure 5: Distribution of Risk Taking (Pre/Post Crisis)

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import numpy as np

# Load your dataset
df = pd.read_csv('master_merged_dataset.csv')

# Pre -processing: Ensure date is datetime
df['date'] = pd.to_datetime(df['date'])

def plot_risk_distribution(df):
    # Define periods
    df['period'] = np.where(df['date'].dt.year < 2008, 'Pre -2008', 'Post -2008')

    plt.figure(figsize=(10, 6))
    sns.kdeplot(data=df, x='risk_taking', hue='period',
                fill=True, common_norm=False, palette='crest')

    plt.title('Distribution of Bank Risk Taking: Pre vs. Post 2008', fontsize=14)
    plt.xlabel('Risk Taking Measure')
    plt.grid(True, alpha=0.3)
    plt.show()


plot_risk_distribution(df)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
ModuleNotFoundError                        Traceback (most recent call last)
```

```
Cell In[2], line 2
      1 import pandas as pd
 - - - -> 2 import matplotlib.pyplot as plt
      3 import matplotlib.dates as mdates
      4 import seaborn as sns

ModuleNotFoundError: No module named 'matplotlib'
```