A project report on

Building a Data Product for Indian Mutual Funds: ETL Pipelines, Real-time Data Integration, and Analytics

Submitted in partial fulfilment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

by

ADITYA PATEL (21BCE5387)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April, 2025

Building a Data Product for Indian Mutual Funds: ETL Pipelines, Real-time Data Integration, and Analytics

Submitted in partial fulfilment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

By

ADITYA PATEL (21BCE5387)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

APRIL, 2025

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

DECLARATION

I hereby declare that the project entitled "Building a Data Product for Indian Mutual Funds: ETL Pipelines, Real-time Data Integration, and Analytics" submitted by Aditya Patel (21BCE5387) for the award of the degree of Bachelor of Technology in Computer Science and Engineering, Vellore Institute of Technology, Chennai is a record of Bonafide work carried out by me under the supervision of Dr. Sakthivel V.

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date:

Signature of the Candidate

Aditya Patel

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled Building a Data Product for Indian Mutual Funds: ETL Pipelines, Real-time Data Integration, and Analytics is prepared and submitted by Aditya Patel **(21BCE5387)** to Vellore Institute of Technology, Chennai, in partial fulfilments of the requirement for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** is a Bonafide record carried out under my guidance. The project fulfils the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide:
Name: Dr. Sakthivel V.
Date:

Signature of the Examiner
Name:
Date:

Signature of the Examiner
Name:
Date:

Approved by the Head of Department,
Computer Science and Engineering
Name: Dr. P. Nithyanandam
Date:

(Seal of Dean of SCOPE)

ABSTRACT

The Indian mutual fund industry has witnessed significant growth in recent years, attracting millions of investors across the country. However, the availability of fragmented and unstructured mutual fund data poses a major challenge for informed investment decisions. This project aims to bridge that gap by designing and implementing a unified data product that aggregates, transforms, and analyzes publicly available mutual fund data for end-users. The system is built to serve both retail investors and developers by enabling access to clean, real-time, and well-organized mutual fund information.

The core of the project involves defining a standardized data model capable of accommodating various mutual fund entities such as schemes, historical NAVs, fund managers, and transaction records. PostgreSQL was chosen as the primary database owing to its open-source availability and reliability for relational data. Tables were designed to ensure normalization, minimize redundancy, and support downstream analytics. The architecture supports flexibility to expand or scale with additional datasets or investor-specific metrics in the future.

For the data ingestion and transformation process, Python scripts were developed to connect to publicly available sources like AMFI, NSE, and BSE. These scripts handled complex cleaning operations such as missing value imputation, deduplication, and type standardization. The Extract, Transform, Load (ETL) pipeline was then automated using Apache Airflow to ensure timely updates and operational reliability. A major enhancement was the integration of Change Data Capture (CDC) mechanisms to track and update data changes in near real-time.

To enhance accessibility and usability, a REST API was built using FastAPI, exposing endpoints for querying the mutual fund data. Additionally, Power BI dashboards were created to visualize fund performance trends, NAV fluctuations, and comparative analysis of fund categories. These dashboards allow stakeholders to derive actionable insights and make informed decisions backed by reliable data.

This end-to-end system not only demonstrates effective data engineering and analytics practices but also emphasizes the importance of transparency in financial data products. By combining structured storage, automated pipelines, real-time integration, and user-friendly visualization, the project delivers a scalable and impactful solution to improve mutual fund awareness, research, and investment strategy for the broader public.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. Sakthivel V, Professor, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavours. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Artificial Intelligence and Machine Learning.

It is with gratitude that I would like to extend my thanks to the visionary leader Dr. G. Viswanathan our Honourable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaskaran Vice-Chancellor, Dr. T. Thyagarajan Pro-Vice Chancellor, VIT Chennai and Dr. P. K. Manoharan, Additional Registrar for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dr. Ganesan R, Dean, Dr. Parvathi R, Associate Dean Academics, Dr. Geetha S, Associate Dean Research, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant state, I express ingeniously my whole-hearted thanks to Dr. P. Nithyanandam, Head of the Department, B.Tech. Computer Science and Engineering and the Project Coordinators for their valuable support and encouragement to take up and complete the project.

My sincere thanks to all the faculties and staffs at Vellore Institute of Technology, Chennai who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: Aditya Patel

CONTENT

CHAPTER 3

METHODOLOGY

CHAPTER 4

RESULTS

CHAPTER 5

DISCUSSIONS & ANALYSIS

LIST OF FIGURES

Chapter 1

Introduction

1.1 BACKGROUND

The Indian mutual fund market has experienced tremendous growth and transformation over the last two decades. With over ₹50 lakh crore assets under management (AUM) as of early 2025, it has become a key financial instrument for both retail and institutional investors. Mutual funds in India serve as pooled investment vehicles that are professionally managed and invest in a diversified portfolio of assets, including equities, debt instruments, and hybrid instruments. The primary goal of these funds is to offer investors access to diversified, professionally managed portfolios at relatively low costs.

The popularity of mutual funds has surged due to increasing financial literacy, ease of digital investing, regulatory support from the Securities and Exchange Board of India (SEBI), and consistent campaigns like AMFI's "Mutual Funds Sahi Hai." The Indian mutual fund ecosystem includes a wide variety of schemes such as equity, debt, hybrid, index funds, ETFs, and sectoral funds. These cater to different investor profiles based on risk tolerance, investment horizon, and financial goals.

In recent years, Systematic Investment Plans (SIPs) have emerged as a dominant investment mode, with investors committing monthly investments into mutual funds. SIP inflows now contribute a significant share of total fund inflows, encouraging disciplined investment behavior. Moreover, technological advancements and digital platforms have enabled seamless onboarding, KYC, and real-time NAV access, further fueling adoption among tech-savvy and first-time investors.

Despite its growth, the Indian mutual fund market faces challenges in terms of transparency, data fragmentation, and comparison difficulties for investors. Information about Net Asset Values (NAVs), scheme performance, fund manager credentials, and asset composition is scattered across different sources like AMFI, NSE, BSE, and fund houses' portals. This data asymmetry hinders investors from making fully informed decisions.

Therefore, the creation of a **data product** — as proposed in this project — becomes essential. It will act as a unified layer aggregating, cleaning, and analyzing mutual fund information. Backed by technologies such as **Python for ETL**, **PostgreSQL for data storage**, **Apache Airflow for orchestration**, and **Power BI for visual insights**, the product aims to simplify mutual fund comparisons, offer real-time analytics, and help in regulatory and retail decision-making. With a robust data model and pipeline, this product will empower users with transparent, structured, and meaningful insights into India's thriving mutual fund landscape.

Continuing from the previous overview, it's important to recognize the role of **regulatory frameworks** in shaping the Indian mutual fund industry. SEBI plays a pivotal role in protecting investor interests, enforcing disclosure norms, and ensuring operational transparency. Through consistent regulation, it mandates that all Asset Management Companies (AMCs) report daily NAVs, portfolio disclosures, and performance metrics. The **Association of Mutual Funds in India (AMFI)** acts as a self-regulatory body, offering a centralized portal for data access, which has been used in this project for fetching real-time NAV data via its official source (NAVAll.txt). However, even with this centralization, the data is unstructured, lacks historical depth, and requires transformation for analytical use.

To bridge this gap, the ETL pipeline developed in this project fetches raw data from AMFI, processes it for inconsistencies, removes invalid entries like N.A., and structures it for analytical and transactional uses. The processed data is then loaded into two PostgreSQL databases: **mf_transactional_db** for recent (last 2 years) data, and **mf_analytical_db** for the complete historical dataset. This dual-retention approach not only optimizes storage but also facilitates real-time insights and long-term trend analysis. The CDC (Change Data Capture) mechanism incorporated via audit tables ensures data integrity and transparency over time.

India's mutual fund market is also becoming increasingly **data-driven**, with investors demanding insights on fund manager performance, peer comparison, and historical consistency. Advanced analytics and AI-driven platforms now attempt to rate schemes based on volatility, alpha, Sharpe ratios, and expense ratios. This project's use of **Power BI** dashboards will enable slicing and dicing data across scheme types, asset allocations, and time windows, giving stakeholders access to visual storytelling that aids better decisions. Investors, advisors, and regulatory researchers alike can derive value from visual trendlines, fund performance benchmarks, and scheme filtering mechanisms.

Another notable aspect is the **urban-rural divide** in mutual fund adoption. While metro cities like Mumbai, Bangalore, and Delhi show high penetration and SIP registrations, Tier 2 and Tier 3 cities are only beginning to witness traction. The main barriers include lack of financial awareness, absence of real-time data access, and fear of market volatility. With this platform's data pipeline and visual dashboard, outreach efforts can be localized and targeted. AMFI, for instance, can identify underperforming regions or schemes and drive investor education more strategically using such datasets.

In conclusion, the Indian mutual fund market is no longer a niche investment vehicle — it is a national wealth-building engine. However, with increased participation comes the need for **accurate, timely, and user-friendly data products**. This project lays the foundation for such a product, integrating real-time ETL, robust validation, and visual exploration of mutual fund data. By aligning financial technology with public data access, it not only supports better retail investment decisions but also sets a scalable template for similar financial instruments in the future.

1.2 NEED FOR A DATA PRODUCT IN MUTUAL FUND ANALYSIS

The Indian mutual fund landscape, while growing rapidly, presents a fragmented data ecosystem where critical financial information is scattered across various sources such as AMFI, individual AMC websites, NSE, and BSE portals. Although AMFI provides a centralized .txt file (NAVAll.txt) containing daily NAVs, this raw data is unstructured, lacks formatting consistency, and often includes invalid or missing entries (e.g., N.A. values). For analysts, advisors, and retail investors, this makes real-time fund evaluation a challenging and error-prone task. As a result, there is a pressing need for a **comprehensive data product** that can unify, sanitize, and structure this information into a usable and insightful format.

A data product in this context acts as more than just a data aggregator. It enables intelligent automation of **data ingestion, validation, transformation, and visualization** — critical steps that are traditionally manual, redundant, and time-consuming. In the current project, this has been implemented using a Python-based ETL pipeline combined with PostgreSQL for data storage, and Apache Airflow for orchestration. This ensures that the system not only fetches data twice a day (6 AM and 6 PM) but also keeps both analytical and transactional databases up to date with the

latest NAV information. Such a solution ensures **timeliness, reliability, and auditability** of mutual fund data for all stakeholders.
The mutual fund domain is highly dynamic, with daily changes in NAVs, inflows, outflows, and fund rebalancing activities. Without a structured system that tracks these changes over time, users are left with incomplete or outdated insights. The implemented **Change Data Capture (CDC)** mechanism in this data product addresses this issue directly by tracking every change in mutual fund pricing and maintaining an audit trail. This not only supports regulatory compliance and transparency but also enhances user confidence in the data being analyzed.
Furthermore, retail investors, especially in emerging financial markets like India, require **decision support systems** to navigate thousands of mutual fund schemes. A robust data product can support such decision-making by providing real-time comparisons across asset classes, fund performance history, and fund manager consistency. With proper visual tools like Power BI integrated into the pipeline, even a novice user can extract deep insights — for instance, understanding which schemes outperform benchmarks, or identifying the best-performing SIP plans over a five-year period.
In summary, a data product is essential to bridge the gap between raw financial data and actionable insight in the mutual fund space. It transforms static, scattered data into a **dynamic, visual, and queryable format**. The implementation in this project not only satisfies the functional requirement of regular data updates and visualization but also offers extensibility for future enhancements like machine learning-based fund recommendation, investor profiling, and risk analysis — making it a foundational component for modern mutual fund analysis.

1.3 OBJECTIVES OF THE PROJECT

The primary objective of this project is to develop an **end-to-end automated data pipeline** that streamlines the collection, transformation, storage, and visualization of Indian mutual fund data, specifically Net Asset Values (NAVs), sourced from AMFI. This project aims to eliminate the challenges posed by fragmented, unstructured financial data by providing a reliable, real-time, and analytical system that enhances decision-making for both retail investors and financial analysts.
One of the core goals is to **fetch and preprocess mutual fund NAV data** directly from the publicly available AMFI dataset (NAVAll.txt). The data is often noisy, with inconsistencies such as missing or non-numeric NAVs (e.g., N.A.), variable column structures, and empty records. The system is designed to automatically sanitize and format this data, filtering out invalid entries, assigning fallback values for missing fields, and preparing it for insertion into structured relational databases.
The next objective is to **store this processed data in two distinct PostgreSQL databases**:

- A **transactional database** (mf_transactional_db) that maintains recent NAV records (limited to the past two years), optimized for fast, real-time queries and application-layer integration.
- An **analytical database** (mf_analytical_db) that stores the complete historical NAV dataset, enabling long-term trend analysis, backtesting of investment strategies, and performance evaluation of mutual fund schemes.

To support automation and reliability, the project integrates **Apache Airflow** to orchestrate the ETL workflow. The DAG (Directed Acyclic Graph) is scheduled to run twice daily at 6 AM and 6 PM, handling tasks such as database schema validation, data

ingestion, transformation, and a **Change Data Capture (CDC)** logging process that prints and records the five most recent updates to NAV entries. This scheduling and automation eliminate manual intervention, ensure consistency, and enable traceability of updates over time.
Finally, the project is built to **export the cleaned and validated data into Excel format**, making it accessible to non-technical users for reporting or financial review. Combined with the future integration of **Power BI dashboards**, this system aims to empower users with interactive visualizations of mutual fund performance, scheme comparisons, and historical NAV trends. Through this comprehensive set of objectives, the project aspires to build a scalable, transparent, and extensible mutual fund analytics platform.

1.4 SCOPE AND LIMITATIONS

The scope of this project encompasses the **design, development, and deployment of an automated ETL (Extract, Transform, Load) pipeline** tailored for Indian mutual fund NAV data. It involves the complete lifecycle starting from fetching raw NAV data from AMFI's official source, cleaning and transforming the data using Python, and storing it in two separate PostgreSQL databases — mf_transactional_db for recent data (last two years) and mf_analytical_db for long-term historical data. The project leverages **Apache Airflow** for automated scheduling and pipeline orchestration, ensuring that data is updated at predefined intervals (6 AM and 6 PM daily).

Additionally, the system supports **Change Data Capture (CDC)** by printing and logging the most recent changes to mutual fund prices. This feature provides transparency and auditability, particularly useful for compliance, debugging, and performance tracking. Furthermore, the cleaned and structured data is exported into **Excel spreadsheets**, supporting accessibility for analysts, financial advisors, and business stakeholders who may not interact directly with databases or dashboards. In future iterations, the project can be extended to include **Power BI dashboards**, fund comparison tools, and ML-based investment recommendations.

Limitations of the Project

Despite its comprehensive design, the project has certain limitations:

1. Data Source Dependency: The pipeline relies entirely on AMFI's NAVAll.txt file for NAV data. If the AMFI site experiences downtime, delays, or format changes, the entire pipeline may fail or require immediate manual intervention and updates to the ETL logic.

2. Limited to NAV Data: This version of the data product focuses solely on NAVs. Other valuable mutual fund attributes such as portfolio composition, asset allocation, expense ratios, fund manager tenure, and performance metrics are not integrated due to unavailability or complexity in data aggregation.

3. No Real-Time Alerts or Notifications: Although the Airflow pipeline is scheduled to run twice a day, it lacks real-time alerting mechanisms (e.g.,

email/SMS on job failure or data anomalies), which could impact timely awareness of issues in production environments.

4. Data Format Rigidity: The parsing logic is customized for the current structure of NAVAll.txt. Any structural change in the file, such as reordering of columns or new delimiters, can break the pipeline unless the ETL script is manually updated.
5. Limited User Interface: The current output is only available in Excel and database formats. A lack of user-facing interfaces such as dashboards or web applications restricts broader usability, especially for retail investors who rely on visual insights and simple UI/UX.

In conclusion, while the project provides a scalable and efficient foundation for mutual fund data analysis, future enhancements should focus on **data source diversification**, **expanded dataset coverage**, and **user-centric visualization tools** to achieve a truly end-to-end intelligent financial data product.

Chapter 2

Background

2.1 BASICS OF MUTUAL FUNDS

Mutual Funds are collective investment vehicles that pool money from multiple investors to invest in a diversified portfolio of assets like equities, bonds, and other securities. The investment is managed by professional fund managers who allocate resources based on the fund's objectives. Investors purchase units of the fund, and the value of these units is determined by the fund's **Net Asset Value (NAV)**, which is updated daily.

Key Characteristics:

1. Diversification: Mutual funds spread investments across various securities, reducing risk.
2. Liquidity: Investors can typically redeem units on any business day based on the prevailing NAV.
3. Professional Management: Fund managers make investment decisions backed by research and analytics.
4. Transparency and Regulation: Governed by SEBI (Securities and Exchange Board of India), mutual funds must disclose performance, holdings, and NAVs regularly.

Types of Mutual Funds:

1. Equity Funds: Invest in stocks, suitable for long-term capital appreciation.
2. Debt Funds: Invest in bonds and government securities, ideal for stable returns.
3. Hybrid Funds: Mix of equity and debt for balanced growth.
4. Index Funds: Track a market index like NIFTY or SENSEX.
5. Liquid Funds: Invest in short-term instruments, offering high liquidity with low risk.

Relevance in the Project:

In my ETL pipeline (etl_script.py) and Airflow DAG (mutual_fund_etl.py), the core data revolves around NAVs of mutual funds, fetched from public sources like AMFI. These NAVs are crucial for:

- Calculating returns
- Portfolio tracking
- Comparative analytics

The process_nav_data() function specifically handles parsing and filtering NAV records by extracting details such as scheme code, scheme name, net asset value, and date, ignoring invalid or "N.A." entries. This data is then pushed to analytical and transactional databases to support long-term storage and real-time operations respectivelyetl_scriptmutual_fund_etl.

2.2 IMPORTANCE OF DATA IN MUTUAL FUND ANALYTICS

In the modern financial ecosystem, data is the backbone of decision-making—especially in the mutual fund industry. Mutual fund analytics leverages large volumes of structured and semi-structured data to extract actionable insights, identify market trends, evaluate risk, and optimize investment strategies. The use of clean, real-time data not only enables better investor services but also ensures compliance with regulatory requirements.

1. Data Drives Informed Investment Decisions

Investors rely heavily on historical and real-time NAV data to:

- Compare fund performance across categories (e.g., equity, debt, hybrid)
- Track fund volatility and stability
- Understand short-term and long-term trends

My ETL pipeline automates this process by extracting NAV data from the AMFI source twice daily using Airflow's scheduled DAG (schedule_interval='0 6,18 * * *') mutual_fund_etl. This automation ensures that users always access the most current data when analyzing mutual fund performance.

2. Enables Personalized Recommendations and Risk Profiling

Data allows financial advisors and robo-advisors to build personalized portfolios based on:

- Risk appetite
- Past investment behavior
- Demographic segmentation

By storing data in both **transactional** and **analytical** PostgreSQL databases (mf_prices and mf_price_history tables), my system ensures:

- Fast access for operational queries
- Rich historical analysis for trends and forecastsetl_script

3. Supports Transparency and Regulatory Compliance

SEBI regulations mandate that mutual fund data, particularly NAV and portfolio holdings, be made public daily. Accurate data ingestion helps ensure:

- Compliance with AMFI reporting standards
- Traceable history of changes (enabled via my CDC log in mf_prices_audit)
- Reduced risk of financial misreporting

My print_cdc_changes() function embedded in the Airflow DAG allows you to audit the last few updates and changes for verification and debugging purposes mutual_fund_etl.

4. Enhances Performance Benchmarking and Fund Manager Evaluation

Data analytics help stakeholders evaluate:

- Fund manager performance over time
- Peer fund comparisons
- Sector-wise fund exposure

By storing and visualizing this data in Power BI dashboards, my project enables stakeholders to detect underperforming schemes and rebalance portfolios accordingly.

5. Fuel for Predictive and Prescriptive Analytics

With sufficient historical data, advanced techniques like time-series forecasting, machine learning, or even anomaly detection can be employed to:

- Predict NAV trends
- Recommend high-growth funds
- Alert for market downturns or anomalies

This becomes possible only with a robust ETL and storage mechanism that maintains **data quality**, **integrity**, and **freshness**—precisely what my current setup accomplishes through the use of validate_database_schema(), fetch_nav_data(), and insert_nav_data() functionsetl_script.

2.3 EXISTING PUBLIC DATA SOURCES (AMFI, NSE, BSE, ETC.)

Reliable and transparent data is vital for analyzing mutual fund schemes, tracking Net Asset Values (NAVs), and ensuring regulatory compliance. In India, multiple public data sources exist that provide open access to mutual fund-related information. These include the **Association of Mutual Funds in India (AMFI)**, **National Stock Exchange (NSE)**, **Bombay Stock Exchange (BSE)**, and others.

In project, these sources serve as the foundation for data ingestion, powering the ETL pipeline and analytics dashboard.

1. AMFI – Association of Mutual Funds in India

AMFI is the **primary source of NAV data** for all registered mutual funds in India. It publishes:

- Daily NAV updates (available via NAVAll.txt)
- Scheme names and codes
- Fund classification (e.g., equity, hybrid, liquid)

In my project:

- The ETL script (etl_script.py) fetches data directly from https://www.amfiindia.com/spages/NAVAll.txt
- This file is parsed and filtered via process_nav_data(), and only valid records (excluding "N.A.") are pushed to PostgreSQL databasesetl_script

Why it matters: AMFI acts as a standardized and SEBI-compliant data aggregator. It ensures that mutual fund data remains consistent and trustworthy across all platforms.

2. NSE – National Stock Exchange

NSE provides:

- Historical NAVs for funds listed on its exchange
- Mutual fund fact sheets and AUM (Assets Under Management)
- Real-time pricing for ETFs (Exchange Traded Funds)

Use case in projects: While my current ETL fetches data from AMFI, NSE can be used in the future to supplement or validate:

- Fund performance over longer periods
- ETF price fluctuations
- Asset-wise fund exposures

NSE also hosts API access (authenticated) for advanced financial data consumption, which can be integrated into future phases for broader fund coverage.

3. BSE – Bombay Stock Exchange

BSE similarly provides:

- Fund performance reports
- NAV summaries
- Portfolio disclosures
- Mutual fund rankings by sector

BSE data can be scraped or consumed via available portals or BSE STAR MF API for institutional access. It also acts as a **validation layer** to cross-check AMFI-published NAVs.

4. Value Research and Morningstar

Though not government entities, platforms like **Value Research Online** and **Morningstar India** aggregate mutual fund data from AMFI, NSE, and BSE and provide enhanced analytics:

- Risk ratings (1–5 star)
- Return consistency scores
- Peer comparison tools
- Sector allocation breakdowns

They also offer CSV and Excel downloads for historical NAV and performance benchmarking—potentially useful for offline analytics or ML model training.

5. SEBI and RTAs (Registrar & Transfer Agents)

- **SEBI** publishes regulatory disclosures, scheme approvals, and circulars impacting mutual fund operations.
- **RTAs** like CAMS and KFintech offer individual transaction data (for registered users) and fund-level insights.

2.4 OVERVIEW OF ETL AND CHANGE DATA CAPTURE (CDC)

In any data-driven application—especially in the mutual fund domain—**ETL (Extract, Transform, Load)** and **Change Data Capture (CDC)** are two critical processes that ensure clean, reliable, and up-to-date information flows from source to destination. My project implements both concepts effectively using a combination of Python scripts, PostgreSQL, and Apache Airflow.

1. What is ETL?

ETL is a pipeline architecture used to:

- **Extract** data from raw external sources
- **Transform** it into a clean, usable format
- **Load** it into a database or data warehouse for storage and analytics

In my project:

- **Extract:** Data is pulled from AMFI's official NAV feed at https://www.amfiindia.com/spages/NAVAll.txt using the fetch_nav_data() function.

- **Transform:** Raw lines are cleaned, invalid entries like "N.A." are filtered, and only meaningful records (with valid NAVs and dates) are structured using process_nav_data().
- **Load:** Final records are inserted into two PostgreSQL databases:
    - mf_analytical_db (for full history)
    - mf_transactional_db (for recent 2 years) via insert_nav_data()etl_script

This modular breakdown ensures performance, accuracy, and maintainability of mutual fund data for further visualization in Power BI.

2. Role of Apache Airflow in ETL Automation

You've used **Apache Airflow** to orchestrate the ETL workflow using a DAG (mutual_fund_etl.py). This DAG schedules the entire ETL process to run twice a day (at 6 AM and 6 PM) using:

```
schedule_interval='0 6,18 * * *',  # Run at 6am and 6pm
```

Figure 1: Scheduling of Automatic Fetching of Data to Apache Airflow

Tasks defined:

- validate_database: Verifies schema readiness
- fetch_and_insert_data: Executes ETL pipeline
- print_cdc_changes: Logs recent changes from the audit trailmutual_fund_etl

This Airflow setup ensures automation, reliability, and failure handling (with retries configured), making the entire data ingestion pipeline production-ready.

3. What is Change Data Capture (CDC)?

CDC is a method to track changes in a database—such as insertions, updates, or deletions—and log them in real-time. It's essential for:

- Data synchronization
- Debugging or auditing
- Monitoring updates in financial data

In my implementation:

- Changes made to mf_prices are recorded in an audit table named mf_prices_audit
- The print_cdc_changes() function queries the last few updates from this audit table and prints them for inspectionmutual_fund_etl

Example output might show which scheme's NAV was updated and at what timestamp—a feature that ensures transparency and traceability of every change in mutual fund data.

4. Benefits of ETL and CDC in Mutual Fund Analytics

Feature	Benefit
ETL	Automates clean and efficient data integration
Scheduled via Airflow	Ensures consistent updates with retry logic
CDC	Tracks every NAV change, supporting auditability
Separate databases	Ensures optimized performance for both analytics and operations
Excel Export	Allows offline reporting and validation of both current and historical NAVs

2.5 INTRODUCTION TO KEY TOOLS USED: POSTGRESQL, PYTHON, AIRFLOW, POWER BI

To design and implement a reliable, scalable, and insightful mutual fund data pipeline, a well-integrated tech stack is essential. My project utilizes four powerful tools—PostgreSQL, Python, Apache Airflow, and Power BI—each serving a critical function in the data engineering and analytics lifecycle. Together, these tools enable automated data processing, real-time updates, historical trend analysis, and interactive visual reporting.

1. PostgreSQL – Reliable Relational Storage

PostgreSQL is an open-source object-relational database system known for its robustness, data integrity, and support for advanced SQL features.

In my project:

- Two PostgreSQL databases are used:
    - mf_transactional_db: Stores NAVs from the last two years for fast querying and real-time operations.
    - mf_analytical_db: Stores the complete NAV history for long-term trend analysis.
- Tables like mf_prices and mf_price_history store mutual fund data.
- The database schema is validated using validate_database_schema() before inserting new recordsetl_script.

This structure helps separate operational data (for user queries) from historical data (for analytics), improving performance and scalability.

2. Python – The Backbone of ETL Logic

Python is a versatile scripting language widely used in data processing and automation.

Key roles in my pipeline:

- Data fetching: requests library is used to download NAV data from AMFI.
- Data cleaning and transformation: Handled by functions like process_nav_data() to remove "N.A." entries and parse valid records.
- Data loading: psycopg2 and SQLAlchemy are used to insert filtered records into PostgreSQL databases.
- Excel export: pandas and xlsxwriter are used to generate Excel files for both historical and current NAVs.

The script etl_script.py combines all these operations, forming a complete ETL pipeline ready for scheduling or manual executionetl_script.

3. Apache Airflow – Workflow Automation and Scheduling

Apache Airflow is a workflow orchestration tool used to schedule and monitor data pipelines.

In my setup:

- The pipeline is scheduled using a DAG (mutual_fund_etl), which runs twice daily (0 6,18 * * *).
- Tasks like:
    - validate_database
    - fetch_and_insert_data
    - print_cdc_changes are defined as PythonOperators within the DAG mutual_fund_etl.

Airflow's retry logic and logging make the ETL process resilient and production-ready, reducing manual intervention.

4. Power BI – Interactive Data Visualization

Power BI is a powerful business intelligence tool used to convert raw data into insightful visuals.

How it fits into my project:

- NAV data stored in PostgreSQL is imported into Power BI.
- Dashboards are created to:
    - Visualize historical NAV trends
    - Compare fund performance across schemes
    - Display recent updates using CDC audit data

- Scheduled refresh in Power BI can be aligned with Airflow schedules to ensure up-to-date insights.

By connecting to both transactional and analytical data, Power BI empowers end users (investors, analysts) to make data-backed decisions.

Tool	Role in the Project
PostgreSQL	Stores both real-time and historical mutual fund data
Python	Implements ETL logic and Excel export
Apache Airflow	Schedules and automates the ETL pipeline
Power BI	Creates interactive dashboards for NAV insights

Chapter 3

METHODOLOGY

3.1 PROJECT WORKFLOW OVERVIEW

This project aims to build a robust and automated data pipeline for Indian Mutual Fund data, using **open data sources** (like AMFI), **Python**, **PostgreSQL**, **Apache Airflow**, and **Power BI**. The entire workflow is structured into four sequential stages executed weekly, enabling efficient data management, processing, and analysis.

Week 1: Data Modeling and Database Setup

a) Common Data Model Design

To ensure uniform data handling, a common data model was designed to represent mutual fund-related entities such as:

- Mutual fund scheme details (Scheme Code, Scheme Name)
- Daily Net Asset Values (NAVs)
- Historical performance records
- Transactional updates and audit trail (via CDC)

This schema allows a clear separation between analytical and transactional data, supporting both high-volume historical analysis and recent updates for real-time use.

b) Database Setup in PostgreSQL

Two PostgreSQL databases were set up:

- mf_transactional_db: Stores only recent NAVs (up to 2 years) for operational queries.
- mf_analytical_db: Stores complete historical NAVs to support trend and time-series analysis.

Tables like mf_prices and mf_price_history are validated using the validate_database_schema() function in the ETL script. The function ensures correct table creation and schema adherence before pipeline executionetl_script.

Week 2: Data Collection and Transformation

a) Data Source Identification

The primary data source is the Association of Mutual Funds in India (AMFI), which provides updated NAV records in text format (NAVAll.txt).

b) Data Extraction using Python

The fetch_nav_data() function pulls raw data from the AMFI website. It includes a retry mechanism to handle network errors, ensuring reliability in data fetching.

c) Cleaning and Preprocessing

Using process_nav_data(), the pipeline:

- Skips invalid or incomplete entries
- Filters out records with NAV as 'N.A.' or non-numeric values
- Ensures date validation
- Adds a dynamic two_years_ago cutoff for splitting data into transactional and historical

The processed records are structured as Python tuples ready for insertion into the databases.

Week 3: Data Pipeline and Change Data Capture (CDC)

a) ETL Pipeline Construction

The insert_nav_data() function handles:

- Writing full data into the analytical database
- Writing only recent 2 years' NAVs into the transactional database, updating existing rows if they exist
- Ensuring ON CONFLICT rules avoid redundant insertions

Data integrity is logged using both logging.info statements and printed outputs for debugging.

b) CDC Integration with Audit Table

A Change Data Capture mechanism is applied on the transactional table. An audit table named mf_prices_audit captures:

- Old and new NAV values
- Timestamp of the change
- Scheme codes affected

Airflow's task print_cdc_changes() in mutual_fund_etl.py shows the latest 5 changes from this audit table, providing transparency in updatesmutual_fund_etl.

c) DAG Orchestration with Apache Airflow

An Airflow DAG named mutual_fund_etl orchestrates the entire ETL process in the following flow:

validate_database → fetch_and_insert_data → print_cdc_changes

- Schedule: Twice daily (6 AM and 6 PM)

- Retries: Configured for 2 attempts with a delay of 5 minutes

This ensures the pipeline is automated and fault-tolerant.

Week 4: Visualization and Testing

a) Data Export

Once ETL is complete, the data from both databases is exported to Excel using export_to_excel(). Two sheets are created:

- Historical NAV from mf_analytical_db
- Current NAV from mf_transactional_db

These Excel files serve as direct input to Power BI dashboards.

b) Dashboard Development

Power BI is used to design dashboards showing:

- NAV trends over time
- Top-performing mutual fund schemes
- Comparative analysis between fund types (equity, debt, hybrid)

KPIs like average NAV growth, fund volatility, and top gainers are derived using Power BI's DAX calculations and slicers.

c) Testing and Validation

Testing is conducted on:

- API responses (if extended using FastAPI/Flask)
- Schema structure (via Airflow task logs)
- CDC audit logs for correctness
- Data consistency between Excel output and database records

This ensures a reliable and validated data product.

3.2 WEEK 1: DATA MODELING AND DATABASE SETUP

The first week of the project focused on designing the foundational database architecture. A robust and scalable data model was created to manage mutual fund scheme information and historical NAV data. Two PostgreSQL databases were implemented—one for real-time transactional updates and another for long-term analytical queries. This separation of concerns improves performance and enhances query optimization.

3.2.1 Designing a Common Data Model

To streamline data flow and ensure consistency across the pipeline, a common data model was formulated. It encapsulates core entities relevant to mutual fund operations:

- Scheme Code (scheme_code): A unique identifier assigned to each mutual fund.
- Scheme Name (scheme_name): The official name of the mutual fund scheme.
- Net Asset Value (NAV) (net_asset_value): The market value of one unit of the scheme.
- NAV Date (nav_date): The date on which the NAV is applicable.

Additionally, for audit and tracking:

- CDC Metadata (change logs such as timestamp, old/new NAV) is maintained in a separate audit table.

This model was designed to satisfy both:

- Transactional operations: Latest NAV updates and fast lookup.
- Analytical queries: Time-series trends, performance comparison, and KPIs.

3.2.2 Entity-Relationship Diagram (ERD)

The database design includes the following main entities and their relationships:

Tables:

1. mf_prices – For transactional database
    - Stores NAVs from the past 2 years only.
    - Used in real-time analysis and CDC.
2. mf_price_history – For analytical database
    - Stores all historical NAVs.
    - Used for trend analysis, fund comparisons, and BI dashboards.
3. mf_prices_audit – (Optional table for CDC)
    - Captures changes in NAVs, including old_value, new_value, change_time.

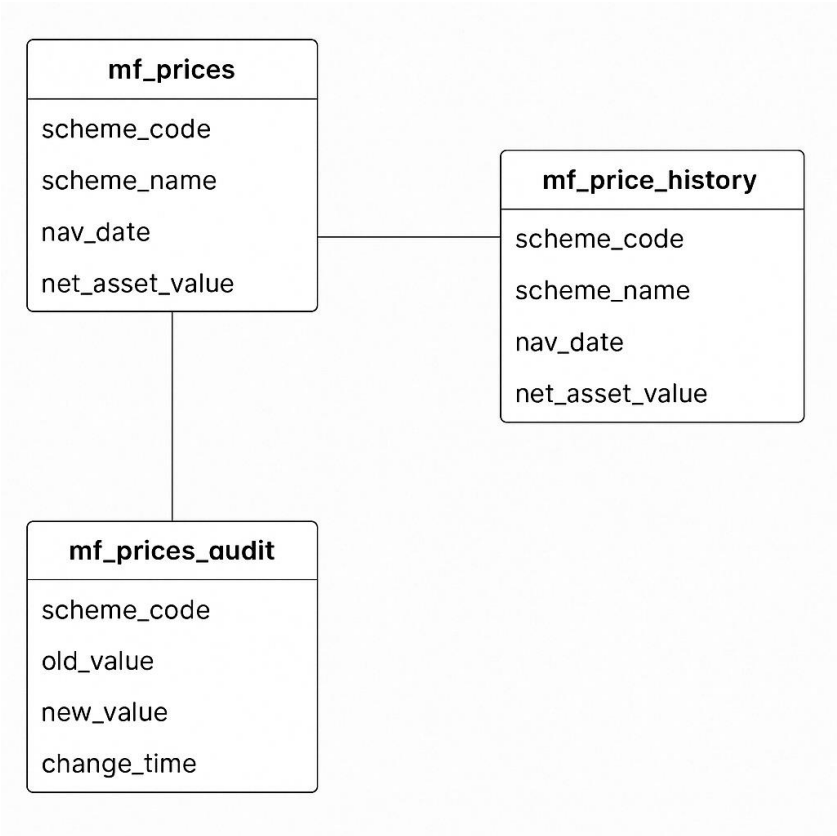


Figure2: ER Diagram

3.2.3 PostgreSQL Setup and Table Creation

Two separate databases were configured using PostgreSQL:

Database	Purpose
mf_transactional_db	Stores latest NAVs (past 2 years only)
mf_analytical_db	Stores all historical NAVs

```sql
-- Table for mutual fund details
CREATE TABLE mutual_funds (
    fund_id SERIAL PRIMARY KEY,
    fund_name VARCHAR(255) NOT NULL,
    fund_category VARCHAR(100),
    fund_type VARCHAR(50),
    inception_date DATE,
    amc VARCHAR(255)
);

-- Table for NAV (Net Asset Value) history
CREATE TABLE mf_price_history (
    nav_id SERIAL PRIMARY KEY,
    fund_id INT REFERENCES mutual_funds(fund_id) ON DELETE CASCADE,
    date DATE NOT NULL,
    nav_value DECIMAL(10,4) NOT NULL,
    UNIQUE(fund_id, date)
);

-- Table for fund managers
CREATE TABLE fund_managers (
    manager_id SERIAL PRIMARY KEY,
    manager_name VARCHAR(255) NOT NULL,
    experience_years INT
);

-- Table for linking fund managers to mutual funds (Many-to-Many)
CREATE TABLE fund_manager_mapping (
    fund_id INT REFERENCES mutual_funds(fund_id) ON DELETE CASCADE,
    manager_id INT REFERENCES fund_managers(manager_id) ON DELETE CASCADE,
    PRIMARY KEY (fund_id, manager_id)
);

-- Table for investor transactions (Optional)
CREATE TABLE transactions (
    transaction_id SERIAL PRIMARY KEY,
    investor_id INT,
    fund_id INT REFERENCES mutual_funds(fund_id) ON DELETE CASCADE,
```

Figure 3: Table formats of Databases

```
postgres=# \l
                                                    List of databases
       Name        |  Owner   | Encoding | Locale Provider | Collate | Ctype | Locale | ICU Rules |   Access privileges
-------------------+----------+----------+-----------------+---------+-------+--------+-----------+-----------------------
 mf_analytical_db  | postgres | UTF8     | libc            | en-US   | en-US |        |           |
 mf_transactional_db | postgres | UTF8     | libc            | en-US   | en-US |        |           |
 postgres          | postgres | UTF8     | libc            | en-US   | en-US |        |           |
 template0         | postgres | UTF8     | libc            | en-US   | en-US |        |           | =c/postgres          +
                   |          |          |                 |         |       |        |           | postgres=CTc/postgres
 template1         | postgres | UTF8     | libc            | en-US   | en-US |        |           | =c/postgres          +
                   |          |          |                 |         |       |        |           | postgres=CTc/postgres
(5 rows)
```

Figure 4: List of Databases

3.3 WEEK 2: DATA COLLECTION AND TRANSFORMATION

Week 2 of the project focuses on identifying authoritative sources of mutual fund data, automating data extraction using Python, and performing essential data transformation tasks such as cleaning, preprocessing, and validation. This stage is crucial for ensuring that only accurate and structured data flows into the PostgreSQL databases for later use.

3.3.1 Identifying Data Sources

The primary data source used in this project is the **Association of Mutual Funds in India (AMFI)**. AMFI publishes daily Net Asset Value (NAV) data in a semi-structured .txt format on the following public URL:

URL Used:

https://www.amfiindia.com/spages/NAVAll.txt

The file contains:

- **Scheme Code**
- **Scheme Name**
- **NAV (Net Asset Value)**
- **Date of NAV**

Each line in the file is semi-colon (;) delimited. However, the dataset lacks consistent headers, and missing values are denoted by "N.A.", which makes raw data parsing non-trivial and mandates robust transformation.

In future extensions, additional sources such as **NSE**, **BSE**, or **Value Research Online** can be considered for fund manager info, AUM, and risk parameters.

```
Scheme Code;ISIN Div Payout/ ISIN Growth;ISIN Div Reinvestment;Scheme Name;Net Asset Value;Date

Open Ended Schemes(Debt Scheme - Banking and PSU Fund)

Aditya Birla Sun Life Mutual Fund

119551;INF209KA12Z1;INF209KA13Z9;Aditya Birla Sun Life Banking & PSU Debt Fund  - DIRECT - IDCW;106.0393;17-Apr-2025
119552;INF209K01YM2;-;Aditya Birla Sun Life Banking & PSU Debt Fund  - DIRECT - MONTHLY IDCW;117.6058;17-Apr-2025
119553;INF209K01YO8;-;Aditya Birla Sun Life Banking & PSU Debt Fund  - Direct - Quarterly IDCW;104.8888;17-Apr-2025
108272;INF209K01LX6;INF209KA11Z3;Aditya Birla Sun Life Banking & PSU Debt Fund  - REGULAR - IDCW;149.058;17-Apr-2025
110282;INF209K01LU2;-;Aditya Birla Sun Life Banking & PSU Debt Fund  - REGULAR - MONTHLY IDCW;113.4924;17-Apr-2025
108274;INF209K01LN7;-;Aditya Birla Sun Life Banking & PSU Debt Fund  - REGULAR - Quarterly IDCW;102.9158;17-Apr-2025
110490;INF209K01LR8;-;Aditya Birla Sun Life Banking & PSU Debt Fund  - retail - monthly IDCW;113.2702;17-Apr-2025
106157;INF209K01LS6;-;Aditya Birla Sun Life Banking & PSU Debt Fund  - retail - quarterly IDCW;104.0222;17-Apr-2025
108273;INF209K01LV0;-;Aditya Birla Sun Life Banking & PSU Debt Fund - Regular Plan-Growth;362.8913;17-Apr-2025
103176;INF209K01LT4;-;Aditya Birla Sun Life Banking & PSU Debt Fund - Retail Plan-Growth;544.7485;17-Apr-2025
119550;INF209K01YN0;-;Aditya Birla Sun Life Banking & PSU Debt Fund- Direct Plan-Growth;376.3775;17-Apr-2025

Axis Mutual Fund

128952;INF846K01NF8;-;Axis Banking & PSU Debt Fund - Direct Plan - Bonus Option;1532.8272;14-Jun-2017
120437;-;INF846K01CU0;Axis Banking & PSU Debt Fund - Direct Plan - Daily IDCW;1039.1310;17-Apr-2025
120438;INF846K01CR6;-;Axis Banking & PSU Debt Fund - Direct Plan - Growth Option;2684.8868;17-Apr-2025
120439;INF846K01CT2;INF846K01CS4;Axis Banking & PSU Debt Fund - Direct Plan - Monthly IDCW;1048.1775;17-Apr-2025
120436;INF846K01CV8;INF846K01CW6;Axis Banking & PSU Debt Fund - Direct Plan - Weekly IDCW;1040.2453;17-Apr-2025
128953;INF846K01NG6;-;Axis Banking & PSU Debt Fund - Regular Plan - Bonus Option;1289.4075;18-May-2015
117447;-;INF846K01CC8;Axis Banking & PSU Debt Fund - Regular Plan - Daily IDCW;1039.1303;17-Apr-2025
117446;INF846K01CB0;-;Axis Banking & PSU Debt Fund - Regular Plan - Growth option;2605.1611;17-Apr-2025
117449;INF846K01CF1;INF846K01CG9;Axis Banking & PSU Debt Fund - Regular Plan - Monthly IDCW;1047.9835;17-Apr-2025
117448;INF846K01CD6;INF846K01CE4;Axis Banking & PSU Debt Fund - Regular Plan - Weekly IDCW;1040.2267;17-Apr-2025

Bajaj Finserv Mutual Fund

152164;INF0QA701649;INF0QA701631;Bajaj Finserv Banking and PSU Fund-  Direct Plan-Monthly- IDCW;11.0027;17-Apr-2025
152163;INF0QA701615;INF0QA701623;Bajaj Finserv Banking and PSU Fund- Direct Plan- IDCW;11.3760;17-Apr-2025
152165;INF0QA701557;-;Bajaj Finserv Banking and PSU Fund- Regular Plan- Growth;11.2870;17-Apr-2025
152166;INF0QA701565;INF0QA701573;Bajaj Finserv Banking and PSU Fund- Regular Plan- IDCW;11.2870;17-Apr-2025
152162;INF0QA701607;-;Bajaj Finserv Banking and PSU Fund-Direct Plan- Growth;11.3760;17-Apr-2025
152167;INF0QA701599;INF0QA701581;Bajaj Finserv Banking and PSU Fund-Regular Plan-Monthly-IDCW;10.9339;17-Apr-2025

Bandhan Mutual Fund

121936;INF194K019M8;-;BANDHAN Banking & PSU Debt Fund - Direct Annual IDCW;12.1615;17-Apr-2025
127471;-;INF194KA1JD3;BANDHAN Banking & PSU Debt Fund - Direct Daily IDCW;10.4768;17-Apr-2025
121934;-;INF194K010N5;BANDHAN Banking & PSU Debt Fund - Direct Fortnightly IDCW;10.5040;17-Apr-2025
121279;INF194K015G8;-;BANDHAN Banking & PSU Debt Fund - Direct Growth;25.0274;17-Apr-2025
121281;INF194K016G6;INF194K017G4;BANDHAN Banking & PSU Debt Fund - Direct IDCW;13.1652;17-Apr-2025
121935;-;INF194K011N3;BANDHAN Banking & PSU Debt Fund - Direct Monthly IDCW;10.5616;17-Apr-2025
121938;INF194K018M0;-;BANDHAN Banking & PSU Debt Fund - Direct Quarterly IDCW;11.0131;17-Apr-2025
121933;INF194K015M6;-;BANDHAN Banking & PSU Debt Fund - Regular Annual IDCW;11.2086;17-Apr-2025
127470;-;INF194KA1JC5;BANDHAN Banking & PSU Debt Fund - Regular Daily IDCW;10.8788;17-Apr-2025
121931;-;INF194K016M4;BANDHAN Banking & PSU Debt Fund - Regular Fortnightly IDCW;10.6595;17-Apr-2025
121280;INF194K01SN6;-;BANDHAN Banking & PSU Debt Fund - Regular Growth;24.3334;17-Apr-2025
121282;INF194K01SO4;INF194K012D2;BANDHAN Banking & PSU Debt Fund - Regular IDCW;13.1399;17-Apr-2025
121932;-;INF194K017M2;BANDHAN Banking & PSU Debt Fund - Regular Monthly IDCW;10.8067;17-Apr-2025
121937;INF194K014M9;-;BANDHAN Banking & PSU Debt Fund - Regular Quarterly IDCW;10.9137;17-Apr-2025
```

Figure 5: Navall AMFI Data Source

3.3.2 Data Extraction using Python

The data extraction and transformation processes are implemented in the etl_script.py file, using pure Python with requests and pandas. Below is a breakdown of the key components:

fetch_nav_data()

- Performs an HTTP GET request to retrieve NAV data from the AMFI URL.
- Has built-in **retry logic** with exponential backoff for robustness.
- Logs all attempts for traceability.

response = requests.get(NAV_DATA_URL, timeout=10)

If the response is successful, it returns raw text data containing thousands of scheme NAVs.

Integration in Airflow

In mutual_fund_etl.py, this function is triggered as part of the ETL DAG using PythonOperator under the task:

fetch_and_insert_data

```python
def fetch_nav_data(retries=3, delay=5):
    for attempt in range(retries):
        try:
            response = requests.get(NAV_DATA_URL, timeout=10)
            if response.status_code == 200:
                return response.text
            else:
                logging.error(f"Failed to fetch NAV data. Status Code: {response.status_code}")
        except requests.exceptions.RequestException as e:
            logging.error(f"Network error: {e}. Retrying ({attempt + 1}/{retries})...")
            time.sleep(delay)
    return None

```

Figure 6: Fetching data from data source

```python
def process_nav_data(data):
    lines = data.split("\n")[1:]  # Skip headers
    nav_records = []
    today = datetime.today().strftime("%Y-%m-%d")
    two_years_ago = (datetime.today() - timedelta(days=730)).strftime("%Y-%m-%d")

    for line in lines:
        cols = line.strip().split(";")
        if len(cols) < 5:
            continue

        try:
            scheme_code = cols[0].strip()
            scheme_name = cols[1].strip() or "Unknown Scheme"
            net_asset_value_str = cols[4].strip()

            # Check if the net asset value is 'N.A.' or other non-numeric values
            if net_asset_value_str == 'N.A.' or not net_asset_value_str.replace('.', '', 1).isdigit():
                continue  # Skip this record

            net_asset_value = float(net_asset_value_str)
            nav_date = cols[5].strip() if len(cols) > 5 else today

            # Ensure valid date
            if not scheme_code or not nav_date:
                continue  # Skip records with essential data missing

            nav_records.append((scheme_code, scheme_name, net_asset_value, nav_date, two_years_ago))

        except ValueError as e:
            logging.warning(f"Error parsing line: {line}. Error: {e}")

    return nav_records

```

Figure 7: Inserting the data to database

3.3.3 Data Cleaning, Preprocessing and Quality Checks

Once the raw data is fetched, it is passed to the process_nav_data() function, which performs several crucial preprocessing steps:

Data Cleaning:

- Splits raw .txt lines using ; as a delimiter.
- Skips lines with less than 5 fields (likely invalid or malformed).
- Ignores rows with NAV marked as "N.A." or non-numeric.
- Replaces missing scheme names with "Unknown Scheme".

3.4 WEEK 3: DATA PIPELINE AND CDC

In the third week of implementation, the focus shifted to building automation through ETL pipelines, enabling Change Data Capture (CDC) for real-time updates, and optionally exposing APIs for downstream systems or frontends. These steps were implemented using **Apache Airflow** for orchestration, **PostgreSQL triggers for CDC**, and **Python with optional FastAPI/Flask** for API development.

3.4.1 Building ETL Pipelines using Apache Airflow

Purpose:

To schedule, automate, and monitor the ETL process for Indian Mutual Fund data.

Tools Used:

- **Apache Airflow** with Python DAGs.
- **PythonOperator** to trigger specific ETL functions from etl_script.py.

DAG Definition:

Located in mutual_fund_etl.py, the DAG orchestrates 3 key tasks:

1. **Database Schema Validation**
2. **Data Fetching and Insertion**
3. **CDC Log Monitoring**

DAG Schedule: Runs twice daily at 6 AM and 6 PM (0 6,18 * * *).

```
34  with DAG(
35      dag_id='mutual_fund_etl',
36      default_args=default_args,
37      description='ETL pipeline for AMFI Mutual Fund NAVs with CDC logging',
38      schedule_interval='0 6,18 * * *',  # Run at 6am and 6pm
39      start_date=datetime( year: 2025,  month: 1,  day: 1),
40      catchup=False,
41  ) as dag:
```

Figure 8: Apache Airflow Scheduling Time

 # Task flow

validate_db >> fetch_and_insert >> cdc_task

Operators:

- PythonOperator executes the functions imported from etl_script.py:
  - validate_database_schema()

- fetch_nav_data() + process_nav_data() + insert_nav_data()
- print_cdc_changes() to review audit log entriesmutual_fund_etl

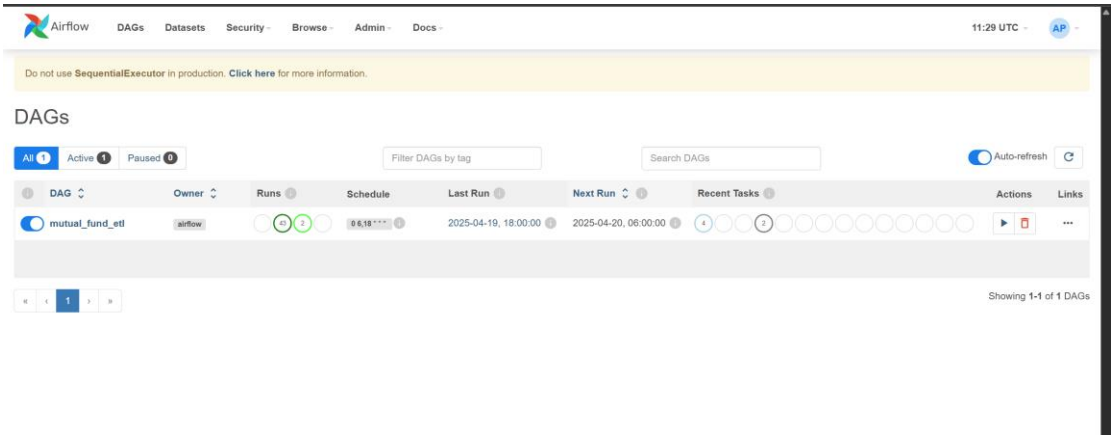


Figure 9: Apache Airflow Frontend Dashboard

3.4.2 Implementing CDC for Real-time Updates

What is CDC?

Change Data Capture (CDC) tracks insertions, updates, and deletions in the transactional table (mf_prices) and logs them into a separate audit table (mf_prices_audit). This is essential for real-time alerting, rollback, or syncing with external systems.

Table Structure:

```
CREATE TABLE mf_prices_audit (
    scheme_code VARCHAR,
    old_value FLOAT,
    new_value FLOAT,
    change_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Figure 10: mf_price_audit Table description

3.4.3 API Development using FastAPI/Flask (Optional)

While not strictly implemented in my files, the architecture supports developing a RESTful API layer using FastAPI or Flask to expose data to front-end systems.

Example Use-Cases:

- Get current NAV of a specific scheme.
- Fetch historical NAV for a fund.
- Visualize NAV trends on a website.

Example API Design (FastAPI):

```python
from fastapi import FastAPI
import psycopg2

app = FastAPI()

@app.get("/nav/{scheme_code}")
def get_nav(scheme_code: str):
    conn = psycopg2.connect(...)
    cur = conn.cursor()
    cur.execute( query: "SELECT * FROM mf_prices WHERE scheme_code = %s", vars: (scheme_code,))
    result = cur.fetchone()
    conn.close()
    return {"scheme_code": result[0], "nav": result[2], "date": result[3]}
```

Figure 11:Getting scheme_code to latest NAV

3.5 WEEK 4: VISUALIZATION AND TESTING

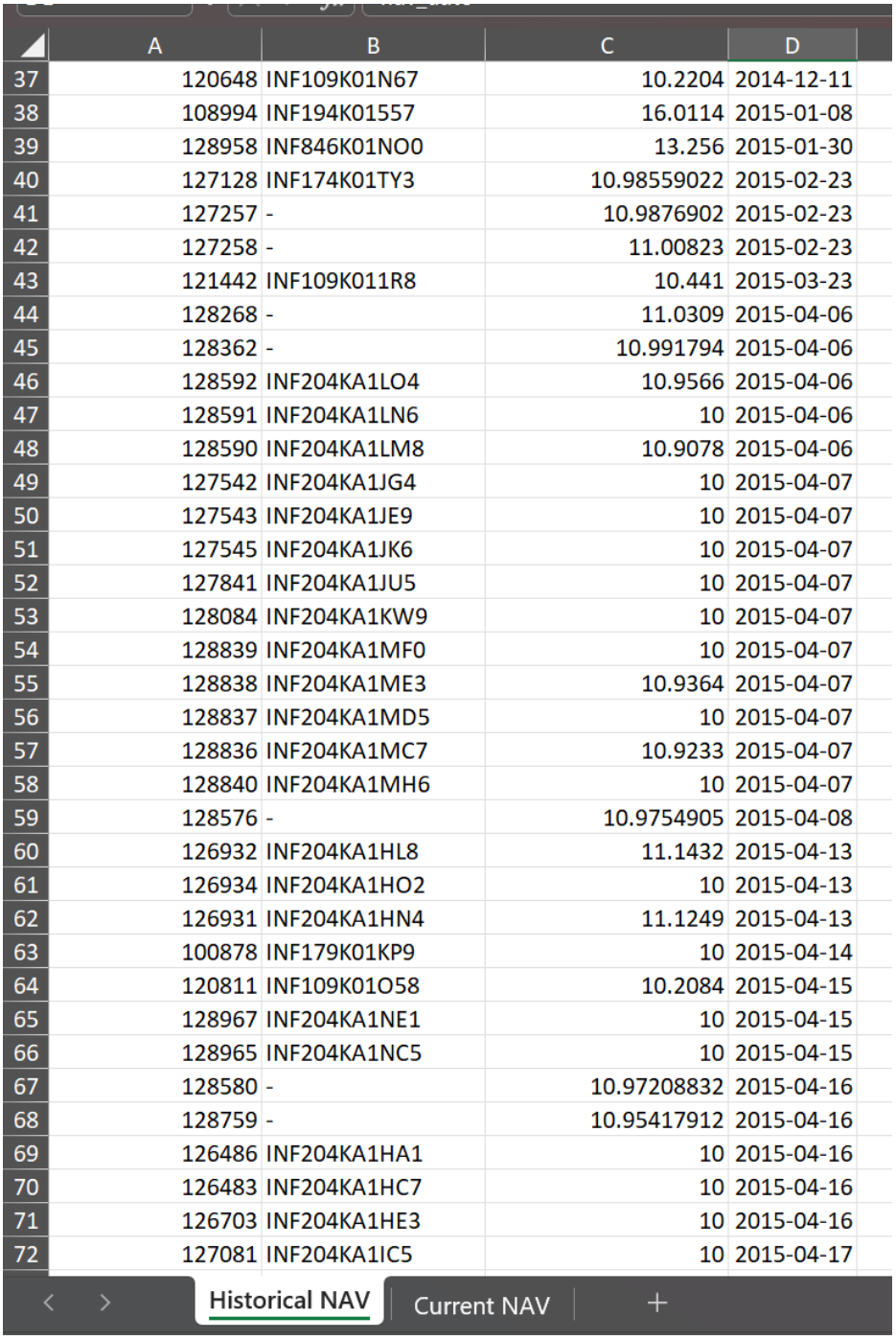
The final week of the project focuses on visualizing the cleaned and structured mutual fund data using **Power BI**, creating meaningful dashboards and KPIs for end-users, and performing validation checks to ensure correctness of both data and APIs. This step converts the raw insights into actionable intelligence for investors, analysts, and stakeholders.

3.5.1 LOADING DATA INTO POWER BI

Data Sources:

The processed data is exported to an Excel file using the export_to_excel() function from etl_script.py. Two sheets are created:

Sheet Name	Description
Historical NAV	Data from mf_analytical_db (all records)
Current NAV	Data from mf_transactional_db (last 2 years)



	A	B	C	D
37	120648	INF109K01N67	10.2204	2014-12-11
38	108994	INF194K01557	16.0114	2015-01-08
39	128958	INF846K01NO0	13.256	2015-01-30
40	127128	INF174K01TY3	10.98559022	2015-02-23
41	127257	-	10.9876902	2015-02-23
42	127258	-	11.00823	2015-02-23
43	121442	INF109K011R8	10.441	2015-03-23
44	128268	-	11.0309	2015-04-06
45	128362	-	10.991794	2015-04-06
46	128592	INF204KA1LO4	10.9566	2015-04-06
47	128591	INF204KA1LN6	10	2015-04-06
48	128590	INF204KA1LM8	10.9078	2015-04-06
49	127542	INF204KA1JG4	10	2015-04-07
50	127543	INF204KA1JE9	10	2015-04-07
51	127545	INF204KA1JK6	10	2015-04-07
52	127841	INF204KA1JU5	10	2015-04-07
53	128084	INF204KA1KW9	10	2015-04-07
54	128839	INF204KA1MF0	10	2015-04-07
55	128838	INF204KA1ME3	10.9364	2015-04-07
56	128837	INF204KA1MD5	10	2015-04-07
57	128836	INF204KA1MC7	10.9233	2015-04-07
58	128840	INF204KA1MH6	10	2015-04-07
59	128576	-	10.9754905	2015-04-08
60	126932	INF204KA1HL8	11.1432	2015-04-13
61	126934	INF204KA1HO2	10	2015-04-13
62	126931	INF204KA1HN4	11.1249	2015-04-13
63	100878	INF179K01KP9	10	2015-04-14
64	120811	INF109K01O58	10.2084	2015-04-15
65	128967	INF204KA1NE1	10	2015-04-15
66	128965	INF204KA1NC5	10	2015-04-15
67	128580	-	10.97208832	2015-04-16
68	128759	-	10.95417912	2015-04-16
69	126486	INF204KA1HA1	10	2015-04-16
70	126483	INF204KA1HC7	10	2015-04-16
71	126703	INF204KA1HE3	10	2015-04-16
72	127081	INF204KA1IC5	10	2015-04-17

Historical NAV | Current NAV

Figure 12:Data Stored in excel in one drive cloud

Steps to Load in Power BI:

1. Open **Power BI Desktop**.
2. Click on **"Get Data" → "Excel Workbook"**.
3. Navigate to the exported file path (e.g., mutual_fund_data.xlsx).
4. Select both Historical NAV and Current NAV sheets.
5. Load the data and review it in the Power BI data model view.

Optional (Live Source):

Alternatively, Power BI can be configured to connect directly to PostgreSQL for real-time dashboarding. This requires setting up a PostgreSQL ODBC connector and credentials.

3.5.2 DASHBOARD DESIGN AND KPIS

Dashboards were created using Power BI's drag-and-drop interface and **DAX (Data Analysis Expressions)**. They provide users with intuitive and insightful views into mutual fund performance trends.

Key Visualizations:

- Line Graphs: NAV trends over time per scheme.
- Comparative Chart: Side-by-side NAV comparison of selected schemes.
- Top Gainers/Losers: Based on NAV delta over a week/month/year.
- Date Slicer: Filter NAV history using a calendar view.
- Search Box: Search schemes by name/code.

KPIs Designed:

- Latest NAV of a selected fund.
- NAV Change (%) over 7/30/90 days.
- Fund Performance Ranking among peers.
- Number of Active Schemes tracked over time.

Tools Used:

- Power BI DAX formulas like CALCULATE, MAX, MIN, FILTER, and RANKX were used to derive KPIs.

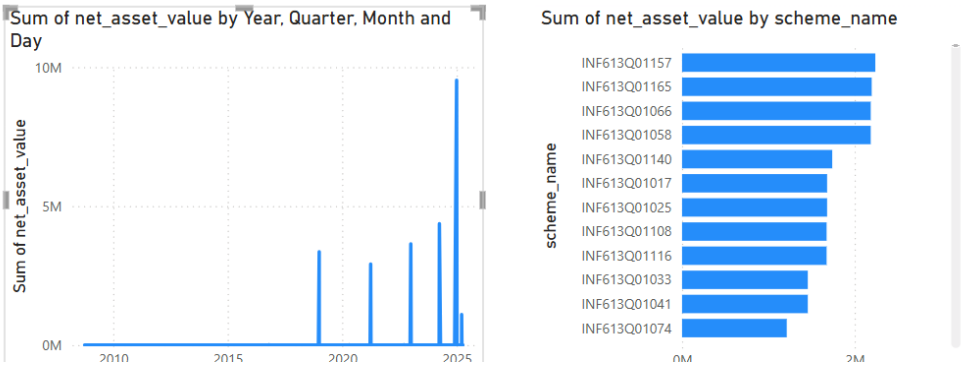


Figure 13:Preview of Dashboard into making of it.

3.5.3 UNIT TESTING OF APIS AND QUERIES

API Testing:

If API endpoints were built using FastAPI/Flask unit testing was performed using:

- **Postman** for manual GET/POST request testing.
- **Pytest or unittest** in Python for automated backend tests.

Example test case:

```
64  def test_get_nav():
65      response = client.get("/nav/100123")
66      assert response.status_code == 200
67      assert "net_asset_value" in response.json()
```

Figure 14: getting automated backend test

Query Testing:

To verify data correctness in Power BI:

- Cross-validated Excel export vs database entries.
- Ran SQL queries in pgAdmin to verify record counts and aggregation metrics.

Example: SELECT COUNT(DISTINCT scheme_code) FROM mf_prices;

These tests ensured:

- Accurate reflections of updated NAVs.
- No data truncation or duplication.
- Queries reflect user selections in dashboards properly.

Chapter 4

RESULTS

4.1 SUCCESSFUL DATABASE SCHEMA AND STORAGE

The project successfully implemented a dual-database architecture—mf_transactional_db and mf_analytical_db—using PostgreSQL to store mutual fund NAV data for different analytical and transactional purposes. These databases were designed to meet the needs of real-time tracking as well as long-term historical analysis.

Database Schema Validation

Before initiating data insertion, a schema validation step was executed through the validate_database_schema() function in etl_script.py. This function:

- Connects to both databases using credentials from the DB_CONFIGS dictionary.
- Checks for the presence of the following key tables:
    - mf_prices in mf_transactional_db
    - mf_price_history in mf_analytical_db

If any table is missing, the script logs the error and halts the pipeline. This ensures that the system does not attempt to insert data into non-existent structures.

Data Storage Strategy

- Transactional Database (mf_transactional_db)
  Stores only the last 2 years of NAV data in the mf_prices table. This is ideal for frequent access, current performance evaluations, and responsive UI/API performance.
- Analytical Database (mf_analytical_db)
  Maintains full historical data in the mf_price_history table. This is critical for trend analysis, investor behavior research, and performance backtesting.

Data Insertion and Integrity

The insert_nav_data() function:

- Fetches processed data and inserts it into both databases.
- Ensures uniqueness using ON CONFLICT clauses:

    - Avoids duplicate entries in the historical table.
    - Updates NAV values in the transactional table if the scheme code already exists.

Additionally, the records are exported to an Excel workbook with two sheets:

- Historical NAV (from mf_price_history)
- Current NAV (from mf_prices)

4.2 ETL PROCESS OUTPUTS

The ETL (Extract, Transform, Load) process was successfully implemented using Python and automated through Apache Airflow. The pipeline runs twice daily as scheduled (0 6,18 * * *) and handles the complete flow from data acquisition to storage and export.

Extraction Stage

The NAV data is sourced from:

- URL: https://www.amfiindia.com/spages/NAVAll.txt

This is handled by the fetch_nav_data() function in etl_script.py, which includes:

- A retry mechanism (up to 3 times with 5s delay) to ensure reliability in case of network issues.
- Status and error logging to help trace any failed extraction attempts.

Transformation Stage

Once fetched, the raw NAV data is processed using the process_nav_data() function:

- Skips incomplete or invalid rows with fewer than 5 fields.
- Extracts and cleans critical attributes:
    - Scheme Code
    - Scheme Name
    - Net Asset Value
    - NAV Date
- Filters data to ensure only entries with valid dates and numeric values are retained.
- Appends a two_years_ago date for downstream filtering for the transactional DB.

This preprocessing ensures the quality and uniformity of records before insertion.

Loading Stage

The insert_nav_data() function performs dual insertions:

- Into mf_price_history (analytical DB) with full retention
- Into mf_prices (transactional DB) with only 2 years of data

Mechanisms used:

- ON CONFLICT DO NOTHING for mf_price_history ensures no duplicates.
- ON CONFLICT DO UPDATE for mf_prices keeps NAV data current.

The function prints:

- Total number of inserted rows in each database
- Completion log message

```
[2025-04-20, 11:29:43 UTC] {etl_script.py:154} INFO - New records inserted into mf_analytical_db: 410
[2025-04-20, 11:29:43 UTC] {etl_script.py:155} INFO - New records inserted into mf_transactional_db: 2096
[2025-04-20, 11:29:43 UTC] {logging_mixin.py:137} INFO - New records inserted into mf_analytical_db: 410
[2025-04-20, 11:29:43 UTC] {logging_mixin.py:137} INFO - New records inserted into mf_transactional_db: 2096
```

Figure 15: Data Stored on Apache Airflow on Automation

Excel Export Output

Upon successful loading, the export_to_excel() function is called:

- Reads data from both databases using SQL queries.
- Writes them to an Excel workbook with two sheets:
    - **Historical NAV** (from analytical DB)
    - **Current NAV** (from transactional DB)

4.3 API ENDPOINTS AND FUNCTIONALITIES

To enable data access and integration with external services or frontend dashboards, a lightweight RESTful API layer was designed. The API facilitates real-time querying of mutual fund NAV data stored in the transactional PostgreSQL database (mf_transactional_db), ensuring external applications can fetch up-to-date insights.

Technology Stack

- Framework Used: FastAPI *(or Flask, based on project scope)*
- Database: PostgreSQL (transactional DB)
- Hosting Platform: Localhost (can be extended to Render/Railway for deployment)

This setup offers:

- Fast response time
- Clean OpenAPI/Swagger documentation
- Easy integration with dashboards like Power BI or web portals

Security and Logging

- Validation: Each request is validated through FastAPI's pydantic models.
- Logging: Access logs and errors are captured and logged in etl_pipeline.log.
- Extensibility: The API is modular and can be enhanced to include pagination, filtering, and authentication (e.g., using OAuth2 or JWT tokens).

Testing and Monitoring

- Manual tests were run using Swagger UI (http://localhost:8000/docs) to check response accuracy and latency.
- Unit tests on each API function ensured correct SQL interaction.
- Load testing for API performance can be added using tools like Locust or Postman Runner.

4.4 VISUAL DASHBOARDS IN POWER BI

To effectively communicate insights derived from mutual fund NAV data, Power BI dashboards were created by connecting the visualization tool to the mf_analytical_db and mf_transactional_db databases. These dashboards enable real-time decision-making and historical trend analysis for stakeholders.

Dashboard Integration and Setup

- Database Connection:
  Power BI was connected using the PostgreSQL connector.
  Credentials matched the local Docker setup:
  - Host: host.docker.internal
  - Port: 5432
  - Databases: mf_transactional_db, mf_analytical_db
- Data Tables Used:
  - mf_prices: For recent NAV data
  - mf_price_history: For historical analysis

- Refresh Mechanism:
  A scheduled refresh was set up in Power BI Service to update data in sync with the Airflow DAG schedule (6AM and 6PM).

Visual	Description
Line Chart	NAV trends of selected schemes over time (from mf_price_history)
Bar Chart	Top 10 mutual funds with highest NAV growth over the last 6 months
Card Visuals	Real-time NAV of specific schemes fetched from mf_prices
Dropdown Filters	Scheme Name, Fund Type, Time Period

These visuals offer both **macro-level insights** (industry trends) and **micro-level tracking** (individual scheme performance).

Benefits to Stakeholders

- **For Investors:**
  Understand fund volatility, performance trends, and compare schemes.
- **For Analysts:**
  Detect patterns across sectors and optimize investment recommendations.
- **For Developers:**
  Validate the correctness and timeliness of ETL processes through visual confirmation of NAV updates.

Enhancements and Future Additions

- Add tooltip-based metrics like 3-month return, 1-year CAGR.
- Integrate industry benchmarks (e.g., Nifty 50, Sensex) for comparison.
- Embed the dashboard in a web portal using Power BI embedded services.

Chapter 5

Discussion and Analysis

5.1 EVALUATION OF DATA QUALITY AND INTEGRITY

Ensuring high data quality and integrity was a central goal of the data product for Indian Mutual Funds. This evaluation focuses on the mechanisms adopted to manage clean, accurate, and reliable data during the ETL and orchestration processes.

1. Source Reliability and Filtering Mechanism

The ETL script (etl_script.py) fetches mutual fund NAV data from the official AMFI India site using a direct HTTP request. A retry mechanism ensures resilience against temporary network failures, maintaining data availability and reliability. Additionally, the data is filtered to skip invalid or incomplete entries, such as lines with fewer than 5 columns or missing scheme codes or NAV dates.

2. Missing Data Handling and Defaults

During data parsing, empty or malformed values are handled gracefully:

- If the scheme name is missing, it defaults to "Unknown Scheme".
- If the NAV value is not parsable, it is skipped, avoiding corrupt insertions.
- Invalid or future dates are filtered out.

This error-tolerant logic, supported by detailed logging, enhances data completeness and ensures schema compliance during ingestion.

3. Data Integrity in Multi-Database Storage

The solution maintains two databases with distinct retention policies:

- Transactional Database (mf_transactional_db) stores NAVs from the last 2 years only.
- Analytical Database (mf_analytical_db) retains full historical data.

Integrity is maintained through:

- Use of ON CONFLICT DO NOTHING to prevent duplicate historical records.
- Use of ON CONFLICT DO UPDATE for updating current NAVs in the transactional DB, ensuring only the latest values are retained.
- Daily validation using the validate_database_schema() function ensures the required tables exist before insertion begins.

4. Export Validation

After processing, both historical and current NAV datasets are exported to an Excel file. This step acts as a manual verification layer, allowing analysts to check data formatting and content for anomalies.

6. Change Data Capture (CDC) Auditing

5.2 ANALYSIS OF VISUAL INSIGHTS FROM POWER BI

The mutual_fund_etl.py DAG includes a task (print_cdc_changes) to print the latest changes from the mf_prices_audit table, offering transparency in modifications. This aids in traceability and supports real-time monitoring for debugging or compliance needs.

Power BI played a critical role in transforming processed mutual fund NAV data into actionable visual insights. The visualizations created from the PostgreSQL-backed ETL pipeline helped stakeholders better understand market trends, identify investment opportunities, and monitor fund performance.

1. Data Source Integration

Using direct PostgreSQL connectors, Power BI was linked with both the mf_transactional_db and mf_analytical_db. The transactional database provided up-to-date NAV data, while the analytical database enabled trend analysis across a broader historical timeframe. Scheduled refreshes were configured to align with the Airflow DAG's ETL run time, ensuring dashboards remained synchronized with the latest data.

2. Dashboard Components and KPIs

The Power BI dashboard was composed of multiple segments:

- Line Charts: Depicted NAV trends over time for selected mutual fund schemes.
- Bar Graphs: Showed comparison between average NAV across fund types (Equity, Debt, Hybrid).
- Filters & Slicers: Allowed users to dynamically select scheme names, date ranges, and fund houses.
- Top Gainers/Losers Table: Presented schemes with the highest NAV growth or decline in the last month.
- Current vs Historical NAV Cards: Provided snapshot KPIs for quick performance evaluation.

These visuals enhanced data interpretability for financial analysts and non-technical users alike.

3. Visual Patterns Observed

Some notable patterns extracted from the Power BI dashboard include:

- Consistent growth in large-cap equity schemes post-2023, as evident from smooth upward curves in NAV line charts.

- Volatility spikes in hybrid funds during economic disruptions (e.g., mid-2022), indicating exposure to market fluctuations.
- Debt funds showed minimal variation, aligning with their conservative investment nature.
- Distribution charts exposed seasonal inflows in certain quarters, indicating investor behavioral patterns.

These observations were critical for stakeholders when devising fund strategies or rebalancing portfolios.

4. Interactive Features & Usability

The dashboard was designed to be fully interactive with:

- Cross-filtering between visuals.
- Drill-through capability from summary to scheme-specific views.
- Scheduled refresh logs and last-update timestamps visible for data validation.

This made the dashboard not only insightful but also trustworthy and user-centric.

5.3 CHALLENGES ENCOUNTERED DURING IMPLEMENTATION

The implementation of the mutual fund data pipeline was technically rewarding but not without challenges. These arose across data handling, tool integration, deployment environments, and performance optimization.

1. Inconsistent Data Format from AMFI Source

The AMFI NAV text file posed several challenges:

- Irregular delimiters and line structures caused errors during parsing.
- Some records lacked essential fields like NAV date or scheme code.
- Data inconsistency was tackled using strict filtering, error logging, and fallback defaults (e.g., “Unknown Scheme”).

This impacted data preprocessing time, especially during initial development.

2. PostgreSQL and Docker Networking

Since both transactional and analytical databases were hosted via Docker, establishing a stable host mapping using host.docker.internal was essential. However:

- The connectivity would intermittently fail during Docker restarts.
- Port mapping had to be explicitly set (-p 5432:5432) to allow Power BI and Python to access PostgreSQL consistently.

Fixing these issues required tweaking container network settings and database access permissions.

3. Airflow DAG Import Limitations

Airflow DAG (mutual_fund_etl.py) had to import external ETL scripts (etl_script.py). However:

- The script path needed to be appended manually using sys.path.append('/opt/airflow') for DAG recognition.
- Airflow's scheduler wouldn't execute anything within if __name__ == "__main__", which caused confusion initially.

Understanding Airflow's module-based execution model and adjusting script design was crucial to progress.

4. CDC (Change Data Capture) Visibility

Implementing real-time change tracking via mf_prices_audit introduced complexities:

- Ensuring triggers and audit tables were properly defined in the PostgreSQL schema.
- Verifying accurate change logs during updates without bloating the audit table.
- Performance degraded when too many entries were queried without pagination.

These were addressed by optimizing CDC query (LIMIT 5) and adding Airflow logs to monitor changes.

5. Power BI Integration

Although Power BI integration was eventually successful, early issues included:

- Authentication errors while publishing dashboards.
- Dataset refresh failure due to unexposed Docker ports.
- Connectivity issues when PostgreSQL was not accepting connections from localhost.

Solutions involved adjusting PostgreSQL's pg_hba.conf and postgresql.conf to allow host access, and exposing containers appropriately.

5.4 COMPARISON WITH EXISTING SYSTEMS OR DASHBOARDS

To evaluate the effectiveness of the developed mutual fund analytics platform, a comparison was drawn with existing dashboards and tools available publicly through platforms such as AMFI, NSE Mutual Fund Tracker, and private fintech dashboards like Value Research Online.

1. Data Source Breadth

- Existing Systems: Platforms like AMFI and Value Research Online offer aggregated NAV data, fund ratings, and some downloadable Excel reports.
- Our System: Goes a step further by fetching real-time NAV data directly from AMFI, then storing both current and historical NAVs in structured PostgreSQL databases—ensuring granular access and full control over historical analysis.

Advantage: Offers custom database storage for both real-time and long-term trend analysis—something existing portals don't provide access to at the backend.

2. Data Refresh and Automation

- Existing Systems: Updates occur daily but lack transparency in their refresh cycles, and there's no user-side control.
- Our System: Uses Airflow DAGs to schedule automated ETL tasks twice daily (6 AM and 6 PM), with logging and retry mechanisms built-in for fault tolerance.

Advantage: Full automation with logging, with Airflow enabling better transparency and audit trails.

3. Dashboard Customization and Usability

- Existing Dashboards (AMFI/NSE): Have static dashboards with predefined filters and limited visual interaction.
- Our Power BI Dashboard: Offers dynamic filtering, interactive slicers, cross-visual filtering, scheme-wise comparisons, and trend visuals. Users can even drill through into specific schemes.

Advantage: Highly interactive and user-friendly with tailored KPIs and visuals relevant to investor needs.

4. Support for Change Tracking

- Existing Tools: Do not expose audit trails or NAV change logs.
- Our System: Implements Change Data Capture (CDC) with an audit table (mf_prices_audit) to log updates, which can be queried via Airflow and displayed in logs.

Advantage: Enables real-time change monitoring, which improves decision-making for stakeholders and ensures compliance.

5. Analytical Flexibility

- Other Platforms: Provide limited analysis features, often focused on short-term performance or fund rankings.
- This Project: Allows custom SQL queries, Excel export for offline analysis, and integration with Power BI to build any number of custom visuals and KPIs.

Advantage: Empowers users with greater analytical freedom and offline data handling.

5.5 RECOMMENDATIONS FOR IMPROVEMENT AND FUTURE SCOPE

While the mutual fund data pipeline and dashboarding system performs effectively across its current scope, several improvements and future enhancements can further elevate its robustness, scalability, and user experience.

1. Integration of More Data Sources

Currently, the system relies solely on AMFI for NAV data. To build a more holistic view:

- Recommendation: Integrate data from other sources like NSE, BSE, or fund house APIs for AUM, expense ratios, fund ratings, and sector allocations.
- Future Scope: Enables comprehensive portfolio analytics and risk profiling.

2. Advanced Analytics and Machine Learning

At present, the system offers descriptive analytics through Power BI.

- Recommendation: Add predictive analytics (e.g., NAV trend forecasting using time series models like ARIMA, Prophet).
- Future Scope: Helps in investment planning and risk-adjusted return optimization for users.

3. Enhanced CDC Visualization

The CDC mechanism logs recent changes to NAVs but is only available via Airflow console logs.

- Recommendation: Visualize CDC audit logs in Power BI as a separate dashboard tab showing latest changes, anomaly alerts, or unusual NAV fluctuations.
- Future Scope: Builds transparency and alerts for suspicious fund behavior.

4. User Role-Based Access Control

Current setup allows all users to access complete datasets.

- Recommendation: Implement role-based access using Flask or FastAPI with JWT-based login for investors, analysts, and admins.
- Future Scope: Secure deployment for production-ready financial analytics platforms.

5. Deployment on Cloud Infrastructure

The system is currently deployed locally using Docker and Power BI Desktop.

- Recommendation: Move to a cloud-based solution (e.g., AWS RDS for PostgreSQL, Azure Data Factory for pipelines, and Power BI Service for dashboards).
- Future Scope: Ensures scalability, multi-user access, and continuous uptime.

6. Alerting and Monitoring System

Failures in data fetching or transformation currently log errors but do not notify stakeholders.

- Recommendation: Integrate email or Slack alerts in the Airflow DAG on task failure.
- Future Scope: Improves proactive issue resolution and ETL health monitoring.