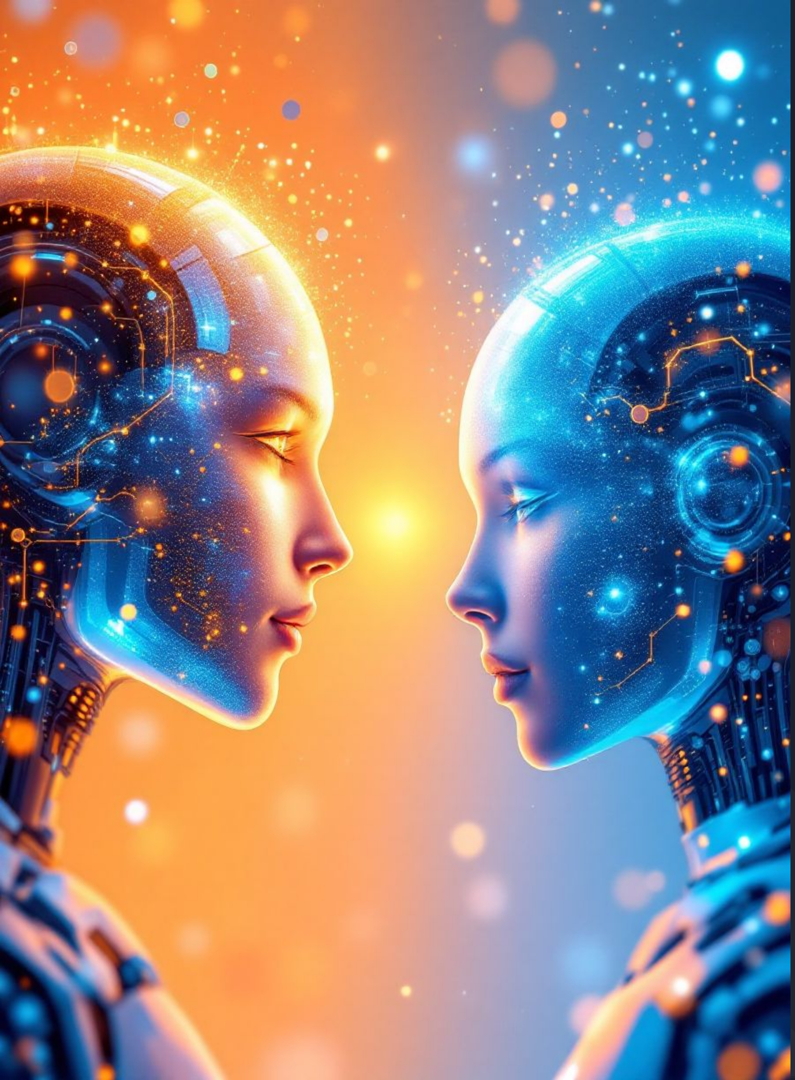


Open AI API v Customized Financial Model: Group 2 Final Project

Our project compares two approaches for extracting insights from financial or sales data: an OpenAI-powered financial insight generator and a customized Python-based financial model. Each offers unique capabilities and limitations in handling and analyzing data.

We explore their features, integrations, limitations, and optimization opportunities to understand their strengths and weaknesses in financial data processing and reporting

Open A.I Financial Insight Generator Overview

Core Feature:

Generates AI-powered financial reports including earnings, forecasts, and CEO tone analysis. Performs sentiment analysis with scores and natural language explanations.

How It Works:

Uses two LangChain LLMChain pipelines: one for analyst-style financial summaries based on company name and report type, and another for sentiment classification and scoring of the

Integration:

Powered by OpenAI GPT API, LangChain prompt chains, Streamlit UI, and Dotenv for secure environment management.

Open AI Financial Insight Generator: Limitations & Optimizations

Limitations:

- Requires stable internet and API access
- Knowledge cutoff may omit latest earnings
- Cannot process raw numerical sales data or files directly

Optimization Opportunities:

- Add memory and user history for better follow-up questions
- Fine-tune prompt templates to improve accuracy
- Incorporate data visualization like charts to complement text

Customized Financial Model Overview

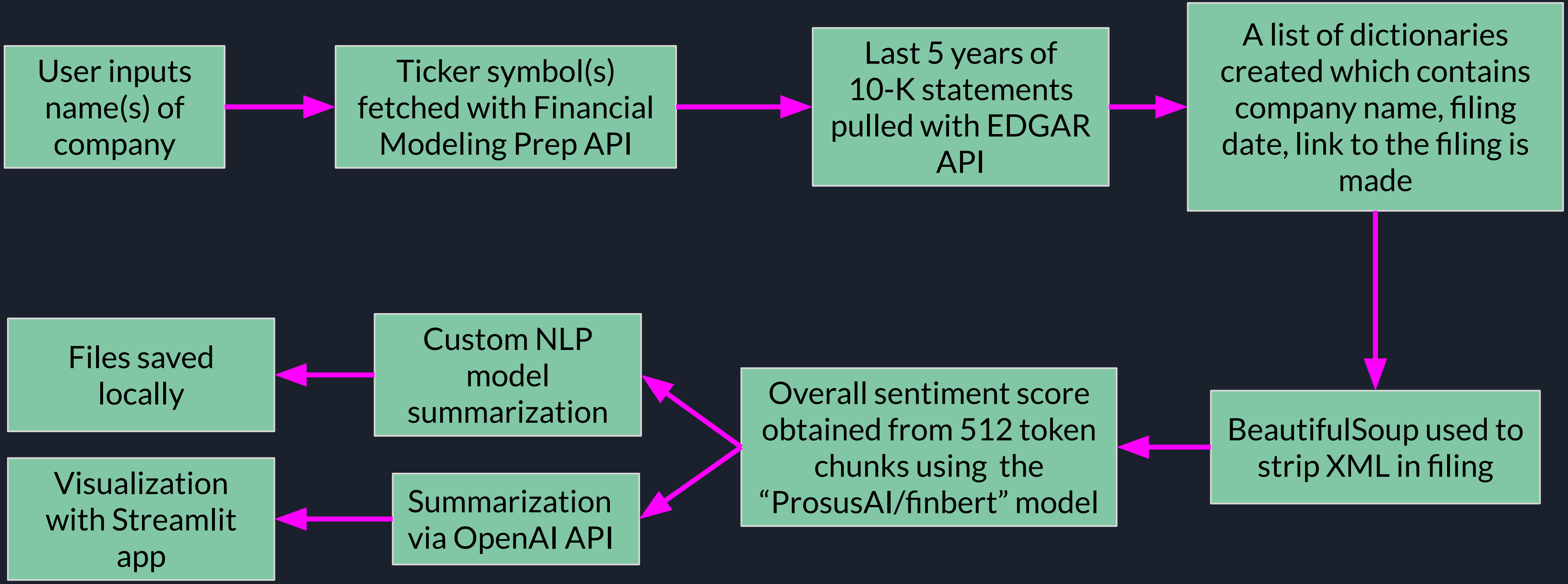
Implementation:

Python 3.x script using os for file system access and pandas for data transformation and merging.

Main Feature:

- Recursively scans folders for sales-related CSV files
- Standardizes inconsistent column names
- Converts sales and order dates to numeric and datetime formats
- Creates sales categories based on thresholds
- Optional filtering by year and source file tracking

Customized Financial Model Overview


User inputs name(s) of company
Ticker symbol(s) fetched with Financial Modeling Prep API
Last 5 years of 10-K statements pulled with EDGAR API
A list of dictionaries created which contains company name, filing date, link to the filing is made
BeautifulSoup used to strip XML in filing
Overall sentiment score obtained from 512 token chunks using the "ProsusAI/finbert" model
Custom NLP model summarization
Summarization via OpenAI API
Files saved locally
Visualization with Streamlit app

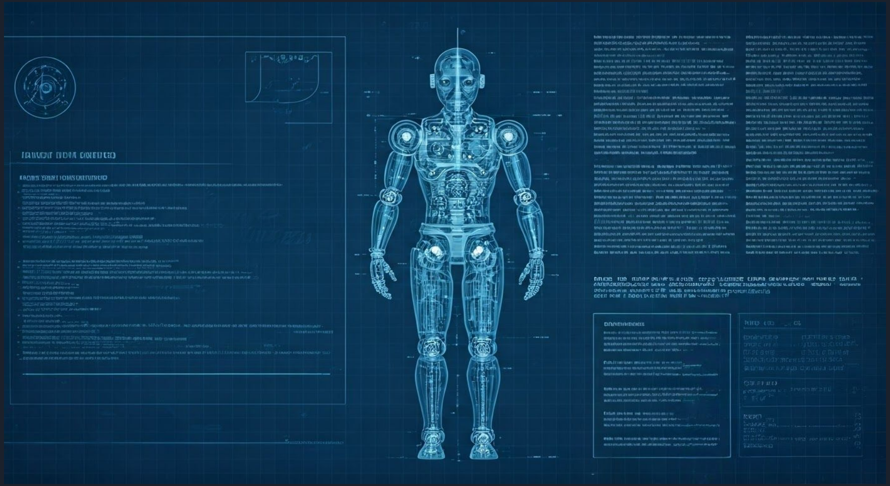
Customized Financial Model: Integration & Unique Capability

Integration:

Automates file system scanning with os.listdir(), parses data using pandas, and supports optional year filtering logic.

Unique Capabilities

Handles real structured data files, automates batch ingestion of many CSVs, adds custom numeric categorization logic, and operates offline without internet or external APIs.



Customized Financial Model: Limitation & Optimization

Limitations:

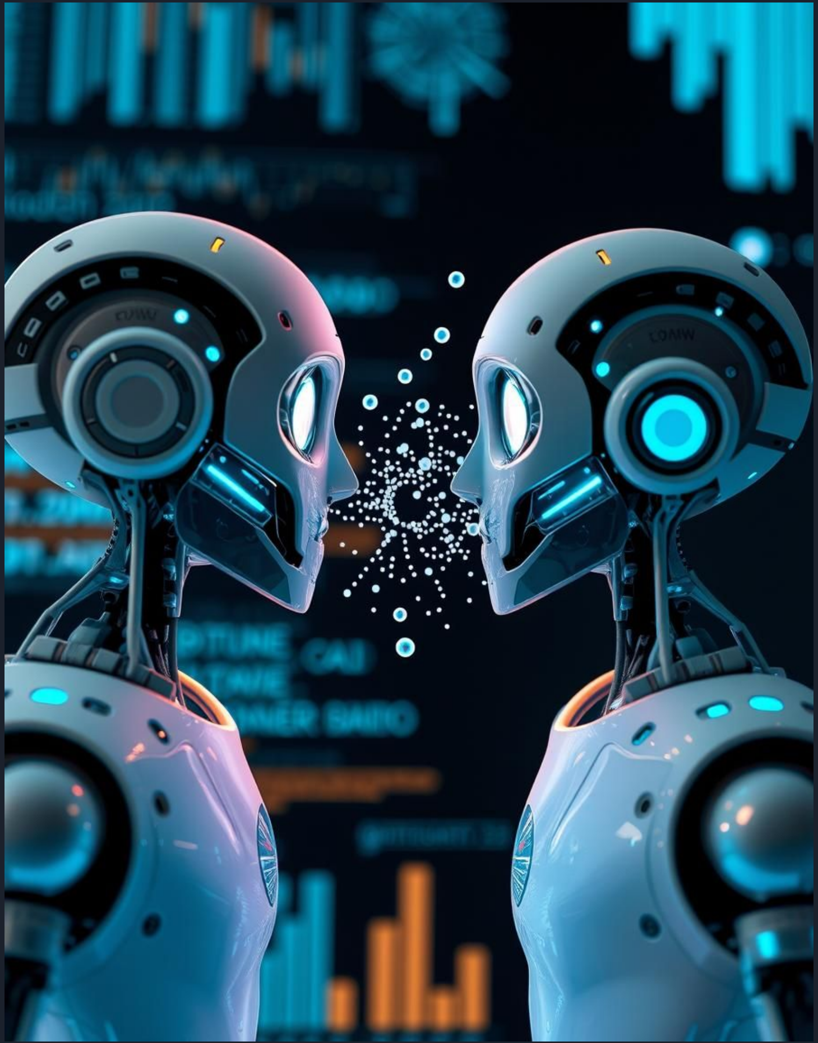
- No user interface; requires command line or script runner
- Lacks built-in predictive or forecasting features
- Minimal error handling and logging in current form

Optimization Opportunities:

- Modularize functions for reuse and testing
- Add logging and exception handling
- Create a GUI using Tkinter or Streamlit for better accessibility

Performance Results: Reading & Summarization

Area	Both Excel	Both can Improve
Data Parsing	Clean, Standardize inputs	Error handling is minimal
Panda Usage	Use powerful transformations	No modular testing
Reusability	Generate final CSV or display	Hardcoded logic, could be config-driven
Output Ready	Extendable by developers	Comments and docstriing could be clearer



Summary and Next Steps

Key Takeaways:

The OpenAI model excels in generating professional financial narratives and sentiment analysis but depends on API access and has limited raw data handling. The customized model efficiently processes real sales data offline with batch automation but lacks UI and predictive features.

Next Steps:

The Enhance OpenAI integration with memory and visualization. Improve the customized model by modularizing code, adding error handling, and developing a user-friendly interface for broader accessibility.