Building an Agentic Financial Analyst with LangGraph and OpenAI

6 min read · Nov 25, 2024

Abhinav Kumar Follow

Listen Share

Image using monica

In the world of stock trading, investors rely on various tools and methods to make informed decisions. One such approach is **fundamental analysis**, which evaluates a company's financial health and stock performance to provide actionable insights. With the advancement of AI and machine learning, stock analysis can now be automated to a great extent. In this post, we will explore how to create a **stock performance analysis agent** using LangChain, LangGraph, and Yahoo Finance, leveraging real-time stock data and key technical indicators.

Whether you're a finance enthusiast, developer, or data scientist, this step-by-step tutorial will empower you to create your own intelligent agent. Let's dive in!

What this Agentic Financial Analyst will do?

- **Fetches stock price data** using Yahoo Finance.
- **Calculates technical indicators** like RSI, MACD, VWAP, and more.
- **Evaluate financial metrics** such as P/E ratio, Debt-to-Equity, and Profit Margins.
- **Provides a structured, AI-generated analysis** using OpenAI's powerful language models.

Tools We'll Use

1. **LangGraph:** A library for orchestrating tools and building conversational agents.
2. **OpenAI GPT-4:** For generating intelligent and structured financial insights.
3. **yfinance:** To retrieve stock prices and financial ratios.
4. **ta (Technical Analysis Library):** For calculating key technical indicators.
5. **Python libraries:** `pandas` , `dotenv` , and `datetime` for data manipulation and environment setup.

Step 1: Setting Up the Environment

Start by installing the required libraries:

```
pip install -U langgraph langchain langchain_openai pandas ta python-dotenv yfin
```

Set up a `.env` file to securely store your OpenAI API key:

```
OPENAI_API_KEY=your_openai_api_key_here
```

Step 2: Tools for Analyst

Fetching Stock Prices: This tool fetches the stock's historical data and computes several technical indicators.

```
from typing import Union, Dict, Set, List, TypedDict, Annotated
import pandas as pd
from langchain_core.tools import tool
```

```
import yfinance as yf
from ta.momentum import RSIIndicator, StochasticOscillator
from ta.trend import SMAIndicator, EMAIndicator, MACD
from ta.volume import volume_weighted_average_price

@tool
def get_stock_prices(ticker: str) -> Union[Dict, str]:
    """Fetches historical stock price data and technical indicator for a given t
    try:
        data = yf.download(
            ticker,
            start=dt.datetime.now() - dt.timedelta(weeks=24*3),
            end=dt.datetime.now(),
            interval='1wk'
        )
        df= data.copy()
        data.reset_index(inplace=True)
        data.Date = data.Date.astype(str)

        indicators = {}

        rsi_series = RSIIndicator(df['Close'], window=14).rsi().iloc[-12:]
        indicators["RSI"] = {date.strftime('%Y-%m-%d'): int(value)
                    for date, value in rsi_series.dropna().to_dict().items()}

        sto_series = StochasticOscillator(
            df['High'], df['Low'], df['Close'], window=14).stoch().iloc[-12:]
        indicators["Stochastic_Oscillator"] = {
                    date.strftime('%Y-%m-%d'): int(value)
                    for date, value in sto_series.dropna().to_dict().items()}

        macd = MACD(df['Close'])
        macd_series = macd.macd().iloc[-12:]
        indicators["MACD"] = {date.strftime('%Y-%m-%d'): int(value)
                    for date, value in macd_series.to_dict().items()}

        macd_signal_series = macd.macd_signal().iloc[-12:]
        indicators["MACD_Signal"] = {date.strftime('%Y-%m-%d'): int(value)
                    for date, value in macd_signal_series.to_dict().items()}

        vwap_series = volume_weighted_average_price(
            high=df['High'], low=df['Low'], close=df['Close'],
            volume=df['Volume'],
        ).iloc[-12:]
        indicators["vwap"] = {date.strftime('%Y-%m-%d'): int(value)
                    for date, value in vwap_series.to_dict().items()}

        return {'stock_price': data.to_dict(orient='records'),
                'indicators': indicators}

    except Exception as e:
        return f"Error fetching price data: {str(e)}"
```

Financial Ratios: This tool retrieves key financial health ratios.

```python
@tool
def get_financial_metrics(ticker: str) -> Union[Dict, str]:
    """Fetches key financial ratios for a given ticker."""
    try:
        stock = yf.Ticker(ticker)
        info = stock.info
        return {
            'pe_ratio': info.get('forwardPE'),
            'price_to_book': info.get('priceToBook'),
            'debt_to_equity': info.get('debtToEquity'),
            'profit_margins': info.get('profitMargins')
        }
    except Exception as e:
        return f"Error fetching ratios: {str(e)}"
```

Step 3: Building the LangGraph

LangGraph allows us to orchestrate tools and manage conversational logic efficiently.

1. Defining the Graph

We start by defining a `StateGraph` to manage the flow:

```python
from langgraph.graph import StateGraph, START, END

class State(TypedDict):
    messages: Annotated[list, add_messages]
    stock: str

graph_builder = StateGraph(State)
```

2. Defining OpenAI and Binding Tools

We integrate the tools into LangGraph and create a feedback loop for analysis

```python
import dotenv
dotenv.load_dotenv()
```

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model='gpt-4o-mini')

tools = [get_stock_prices, get_financial_metrics]
llm_with_tool = llm.bind_tools(tools)
```

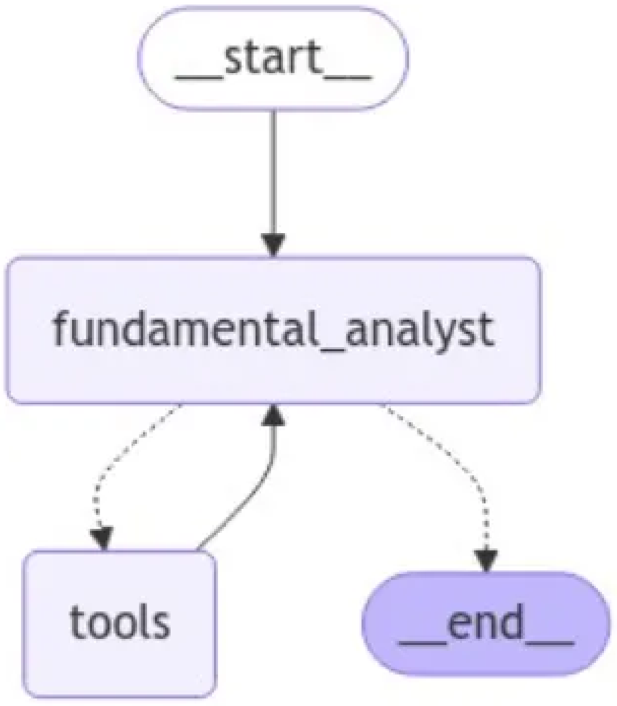
3. Analyst Node

The prompt ensures the AI understands its role and delivers structured output.

```
FUNDAMENTAL_ANALYST_PROMPT = """
You are a fundamental analyst specializing in evaluating company (whose symbol i

You have access to the following tools:
1. **get_stock_prices**: Retrieves the latest stock price, historical price data
2. **get_financial_metrics**: Retrieves key financial metrics, such as revenue,

### Your Task:
1. **Input Stock Symbol**: Use the provided stock symbol to query the tools and
2. **Analyze Data**: Evaluate the results from the tools and identify potential
3. **Provide Summary**: Write a concise, well-structured summary that highlights
    - Recent stock price movements, trends and potential resistance.
    - Key insights from technical indicators (e.g., whether the stock is overbou
    - Financial health and performance based on financial metrics.

### Constraints:
- Use only the data provided by the tools.
- Avoid speculative language; focus on observable data and trends.
- If any tool fails to provide data, clearly state that in your summary.

### Output Format:
Respond in the following format:
"stock": "<Stock Symbol>",
"price_analysis": "<Detailed analysis of stock price trends>",
"technical_analysis": "<Detailed time series Analysis from ALL technical indicat
"financial_analysis": "<Detailed analysis from financial metrics>",
"final Summary": "<Full Conclusion based on the above analyses>"
"Asked Question Answer": "<Answer based on the details and analysis above>"

Ensure that your response is objective, concise, and actionable."""

def fundamental_analyst(state: State):
    messages = [
        SystemMessage(content=FUNDAMENTAL_ANALYST_PROMPT.format(company=state['s
    ]  + state['messages']
    return {
        'messages': llm_with_tool.invoke(messages)
    }
graph_builder.add_node('fundamental_analyst', fundamental_analyst)
graph_builder.add_edge(START, 'fundamental_analyst')
```

4. Adding tool to graph and compile

```
graph_builder.add_node(ToolNode(tools))
graph_builder.add_conditional_edges('fundamental_analyst', tools_condition)
graph_builder.add_edge('tools', 'fundamental_analyst')

graph = graph_builder.compile()
```



graph

5. Executing the Graph

```
events = graph.stream({'messages':[('user', 'Should I buy this stock?')],
 'stock': 'TSLA'}, stream_mode='values')
for event in events:
    if 'messages' in event:
        event['messages'][-1].pretty_print()
```

Sample Output

```
{
  "stock": "TSLA",
  "price_analysis": "The recent stock price for TSLA has shown volatility, with
  "technical_analysis": "The technical indicators present a mixed outlook. The R
  "financial_analysis": "TSLA's financial metrics indicate a high valuation rela
  "final Summary": "In summary, TSLA shows strong recent price recovery with pot
  "Asked Question Answer": "Given the current overbought indicators and high val
}
```

Future Improvements

Incorporating the idea of a **full portfolio manager agent** is a fantastic improvement to the project! With multiple specialized teams working under one umbrella, the agent can be significantly enhanced to cover a wide range of areas and provide a comprehensive portfolio management tool.

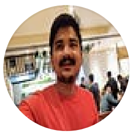
Conclusion

Building an Agentic Financial Analyst is not only a great way to learn about AI and financial analysis but also a stepping stone to creating powerful, real-world applications. Try it yourself and see the power of automation in action!

Check out the complete project on GitHub. Fork it, experiment, and let me know your thoughts!

What features would you add to this project? Drop a comment below, and let's discuss!

Finance Llm Stock Market Langgraph Langchain



Follow

Written by Abhinav Kumar

184 followers · 142 following

AI enthusiast, designing and deploying AI agents that extract meaningful insights and automate complex tasks is my idea of a good time.