langgraph

October 25, 2024

1 This is simple implementation of agentic workflow using various LLM endpoints/providers with Langchain and Langraph

Installing dependencies

[1]:
```
!pip install langchain -qU
!pip install langgraph -qU
!pip install langchain-anthropic -qU
!pip install langchain-groq -qU
```

```
                                           50.6/50.6 kB
1.6 MB/s eta 0:00:00
                                        1.0/1.0 MB
15.1 MB/s eta 0:00:00
                                        407.7/407.7 kB
17.6 MB/s eta 0:00:00
                                        296.9/296.9 kB
12.1 MB/s eta 0:00:00
                                        76.4/76.4 kB
5.3 MB/s eta 0:00:00
                                        78.0/78.0 kB
3.9 MB/s eta 0:00:00
                                        144.5/144.5 kB
8.1 MB/s eta 0:00:00
                                        54.5/54.5 kB
3.3 MB/s eta 0:00:00
                                        58.3/58.3 kB
2.7 MB/s eta 0:00:00
                                        113.5/113.5 kB
3.2 MB/s eta 0:00:00
                                        946.0/946.0 kB
14.8 MB/s eta 0:00:00
                                        325.2/325.2 kB
29.1 MB/s eta 0:00:00
                                        106.5/106.5 kB
3.4 MB/s eta 0:00:00
```

Importing dependencies

[96]:
```
import nest_asyncio
from typing import List, TypedDict, Any, Tuple
from pydantic import BaseModel, Field
from langgraph.graph import StateGraph, END, START
from datetime import date, datetime

nest_asyncio.apply()
```

2 API keys

[97]:
```
from google.colab import userdata
CLAUDE_API_KEY = userdata.get('CLAUDEAI_API_KEY')
GROQ_API_KEY = userdata.get('GROQ_API_KEY')
FINANCIAL_MODELING_PREP_API_KEY = userdata.get('FMP_API_KEY')
```

3 Functions

[98]:
```
def convert_to_dbl_qt(input: str) -> str:
  return input.replace("'", '"')
```

4 get_stock_price

[99]:
```
import os
import requests
from pprint import pprint
class StockPrice(BaseModel):
    symbol:str = Field(description="The symbol of the company")
    price:float = Field(description="The price of the company")
    volume:float = Field(description="The volume of the company")
    priceAvg50:float = Field(description="The 50 day average price of the␣
  ↪company")
    priceAvg200:float = Field(description="The 200 day average price of the␣
  ↪company")
    eps:float = Field(description="The EPS of the company")
    pe:float = Field(description="The PE of the company")
    earningsAnnouncement:datetime = Field(description="The earnings␣
  ↪announcement of the company")
# Define the functions that will fetch financial data
def get_stock_price(symbol):
    """
    Fetch the current stock price for the given symbol, the current volume, the␣
  ↪average price 50d and 200d, EPS, PE and the next earnings Announcement.
```

```
    """
    try:
      url = f"https://financialmodelingprep.com/api/v3/quote-order/{symbol}?
↪apikey={FINANCIAL_MODELING_PREP_API_KEY}"
      response = requests.get(url)
      data = response.json()
      stock_price = StockPrice(**data[0])
      return stock_price
    except (IndexError, KeyError):
        return {"error": f"Could not fetch price for symbol: {symbol}"}

## DATA PROVIDED BY THIS ENDPOINT:
# [{'symbol': 'AAPL',
#   'name': 'Apple Inc.',
#   'price': 222.5,
#   'changesPercentage': -0.1212,
#   'change': -0.27,
#   'dayLow': 221.91,
#   'dayHigh': 224.03,
#   'yearHigh': 237.23,
#   'yearLow': 164.08,
#   'marketCap': 3382912250000,
#   'priceAvg50': 223.0692,
#   'priceAvg200': 195.382,
#   'exchange': 'NASDAQ',
#   'volume': 35396922,
#   'avgVolume': 57548506,
#   'open': 223.58,
#   'previousClose': 222.77,
#   'eps': 6.57,
#   'pe': 33.87,
#   'earningsAnnouncement': '2024-10-31T00:00:00.000+0000',
#   'sharesOutstanding': 15204100000,
#   'timestamp': 1726257601}]
```

5 get_company_financials

[125]:
```
class CompanyFinancials(BaseModel):
    symbol:str =  Field(description="The symbol of the company")
    companyName:str =  Field(description="The name of the company")
    marketCap:float = Field(alias="mktCap", description="The market␣
↪capitalization of the company")
    industry:str =  Field(description="The industry of the company")
    sector:str =  Field(description="The sector of the company")
    website:str =  Field(description="The website of the company")
    beta:float = Field(description="The beta of the company")
    price:float = Field(description="The price of the company")

def get_company_financials(symbol) -> Tuple[Any, CompanyFinancials]:
    """
    Fetch basic financial information for the given company symbol such as the␣
 ↪industry, the sector, the name of the company, and the market capitalization.
    """
    try:
      url = f"https://financialmodelingprep.com/api/v3/profile/{symbol}?
 ↪apikey={FINANCIAL_MODELING_PREP_API_KEY}"
      response = requests.get(url)
      data = response.json()
      financials = CompanyFinancials(**data[0])
      return financials
    except (IndexError, KeyError):
        return {"error": f"Could not fetch financials for symbol: {symbol}"}

## DATA PROVIDED BY THIS ENDPOINT:
# [{'symbol': 'AAPL',
#   'price': 222.5,
#   'beta': 1.24,
#   'volAvg': 57548506,
#   'mktCap': 3382912250000,
#   'lastDiv': 1,
#   'range': '164.08-237.23',
#   'changes': -0.27,
#   'companyName': 'Apple Inc.',
#   'currency': 'USD',
#   'cik': '0000320193',
#   'isin': 'US0378331005',
#   'cusip': '037833100',
#   'exchange': 'NASDAQ Global Select',
#   'exchangeShortName': 'NASDAQ',
#   'industry': 'Consumer Electronics',
#   'website': 'https://www.apple.com',
#   'description': 'Apple Inc. designs, manufactures, and markets smartphones,␣
 ↪personal computers, tablets, wearables, and accessories worldwide. The␣
 ↪company offers iPhone, a line of smartphones; Mac, a line of personal␣
 ↪computers; iPad, a line of multi-purpose tablets; and wearables, home, and␣
 ↪accessories comprising AirPods, Apple TV, Apple Watch, Beats products, and␣
 ↪HomePod. It also provides AppleCare support and cloud services; and operates␣
 ↪various platforms, including the App Store that allow customers to discover␣
 ↪and download applications and digital content, such as books, music, video,␣
 ↪games, and podcasts. In addition, the company offers various services, such␣
 ↪as Apple Arcade, a game subscription service; Apple Fitness+, a personalized␣
 ↪fitness service; Apple Music, which offers users a curated listening␣
 ↪experience with on-demand radio stations; Apple News+, a subscription news␣
 ↪and magazine service; Apple TV+, which offers exclusive original content;␣
 ↪Apple Card, a co-branded credit card; and Apple Pay, a cashless payment␣
 ↪service, as well as licenses its intellectual property. The company serves␣
 ↪consumers, and small and mid-sized businesses; and the education,␣
 ↪enterprise, and government markets. It distributes third-party applications␣
 ↪for its products through the App Store. The company also sells its products␣
 ↪through its retail and online stores, and direct sales force; and␣
 ↪third-party cellular network carriers, wholesalers, retailers, and resellers.
 ↪ Apple Inc. was incorporated in 1977 and is headquartered in Cupertino,␣
 ↪California.',
#   'ceo': 'Mr. Timothy D. Cook',
#   'sector': 'Technology',
#   'country': 'US',
#   'fullTimeEmployees': '161000',
#   'phone': '408 996 1010',
#   'address': 'One Apple Park Way',
#   'city': 'Cupertino',
#   'state': 'CA',
#   'zip': '95014',
#   'dcfDiff': 55.70546,
#   'dcf': 166.79453554058594,
#   'image': 'https://financialmodelingprep.com/image-stock/AAPL.png',
#   'ipoDate': '1980-12-12',
#   'defaultImage': False,
#   'isEtf': False,
#   'isActivelyTrading': True,
#   'isAdr': False,
#   'isFund': False}]
```

6 get_income_statement

[126]:
```python
class IncomeStatement(BaseModel):
    date_field: date = Field(alias='date', description="The date of the income␣
↪statement")
    revenue:float = Field(description="The revenue of the company")
    gross_profit:float = Field(alias='grossProfit', description="The gross␣
↪profit of the company")
    net_income:float = Field(alias='netIncome', description="The net income of␣
↪the company")
    ebitda:float = Field(description="The EBITDA of the company")
    eps:float = Field(description="The EPS of the company")
    eps_diluted:float = Field(alias='epsdiluted', description="The EPS diluted␣
↪of the company")


def get_income_statement(symbol):
    """
    Fetch last income statement for the given company symbol such as revenue,␣
↪gross profit, net income, EBITDA, EPS.
    """
    try:
      url = f"https://financialmodelingprep.com/api/v3/income-statement/
↪{symbol}?period=annual&apikey={FINANCIAL_MODELING_PREP_API_KEY}"
      response = requests.get(url)
      data = response.json()
      financials = IncomeStatement(**data[0])
      return financials
    except (IndexError, KeyError):
        return {"error": f"Could not fetch financials for symbol: {symbol}"}

## DATA PROVIDED BY THIS ENDPOINT:
# {'date': '2023-09-30',
#   'symbol': 'AAPL',
#   'reportedCurrency': 'USD',
#   'cik': '0000320193',
#   'fillingDate': '2023-11-03',
#   'acceptedDate': '2023-11-02 18:08:27',
#   'calendarYear': '2023',
#   'period': 'FY',
#   'revenue': 383285000000,
#   'costOfRevenue': 214137000000,
#   'grossProfit': 169148000000,
#   'grossProfitRatio': 0.4413112958,
#   'researchAndDevelopmentExpenses': 29915000000,
#   'generalAndAdministrativeExpenses': 0,
#   'sellingAndMarketingExpenses': 0,
#   'sellingGeneralAndAdministrativeExpenses': 24932000000,
#   'otherExpenses': 382000000,
#   'operatingExpenses': 55229000000,
#   'costAndExpenses': 269366000000,
#   'interestIncome': 3750000000,
#   'interestExpense': 3933000000,
#   'depreciationAndAmortization': 11519000000,
#   'ebitda': 125820000000,
#   'ebitdaratio': 0.3282674772,
#   'operatingIncome': 114301000000,
#   'operatingIncomeRatio': 0.2982141227,
#   'totalOtherIncomeExpensesNet': -565000000,
#   'incomeBeforeTax': 113736000000,
#   'incomeBeforeTaxRatio': 0.2967400237,
#   'incomeTaxExpense': 16741000000,
#   'netIncome': 96995000000,
#   'netIncomeRatio': 0.2530623426,
#   'eps': 6.16,
#   'epsdiluted': 6.13,
#   'weightedAverageShsOut': 15744231000,
#   'weightedAverageShsOutDil': 15812547000,
#   'link': 'https://www.sec.gov/Archives/edgar/data/320193/000032019323000106/
↪0000320193-23-000106-index.htm',
#   'finalLink': 'https://www.sec.gov/Archives/edgar/data/320193/
↪000032019323000106/aapl-20230930.htm'}
```

7 Generate Report

[143]:
```
def generate_markdown_report(company_financials: CompanyFinancials,␣
↪income_statement: IncomeStatement, stock_price: StockPrice) -> str:
    """
    Generates a markdown report from the GraphState instance.
    """

    company_info = f"""
      # Financial Report for {company_financials.companyName}␣
↪({company_financials.symbol})

      ## Company Overview
      - **Name**: {company_financials.companyName}
      - **Symbol**: {company_financials.symbol}
      - **Market Capitalization**: {company_financials.marketCap}
      - **Industry**: {company_financials.industry}
      - **Sector**: {company_financials.sector}
      - **Website**: [{company_financials.website}]({company_financials.
↪website})
```

```
      - **Beta**: {company_financials.beta: .3f}
      - **Current Price**: ${company_financials.price: .2f}
      """ if (company_financials) else " No company financials were obtained"

    income_statement = f"""
    ## Income Statement (as of {income_statement.date_field})
    - **Revenue**: ${income_statement.revenue: .2f}
    - **Gross Profit**: ${income_statement.gross_profit: .2f}
    - **Net Income**: ${income_statement.net_income: .2f}
    - **EBITDA**: ${income_statement.ebitda: .2f}
    - **EPS**: {income_statement.eps: .2f}
    - **EPS (Diluted)**: {income_statement.eps_diluted: .2f}
    """ if income_statement else "No income statement was obtained"

    stock_price = f"""
      ## Stock Price Information
    - **Current Price**: ${stock_price.price: .2f}
    - **Volume**: {stock_price.volume: .2f}
    - **50-Day Average Price**: ${stock_price.priceAvg50: .2f}
    - **200-Day Average Price**: ${stock_price.priceAvg200: .2f}
    - **EPS**: {stock_price.eps: .2f}
    - **PE Ratio**: {stock_price.pe: .2f}
    - **Earnings Announcement**: {stock_price.earningsAnnouncement}
    """ if stock_price else "No stock price information was obtained"

    md_report = f"""
    {company_info}

    {income_statement}

    {stock_price}
    """
    return md_report
```

8 Anthropic LLM

[129]:
```
# from langchain_anthropic import ChatAnthropic
# llm = ChatAnthropic(api_key=CLAUDE_API_KEY,␣
↪model_name='claude-3-sonnet-20240229', temperature=0.0)
```

9 GROQ LLM

[130]:
```
from langchain_groq import ChatGroq

llm = ChatGroq(api_key=GROQ_API_KEY,
    model="mixtral-8x7b-32768",
    temperature=0,
    max_tokens=None,
    timeout=None,
    max_retries=2,
)
```

10 Generation Chain

[131]:
```
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
```

[132]:
```
class Extraction(BaseModel):
  symbol: str = Field(description="The symbol of the company")
```

[133]:
```
UNKNOWN = 'UNKNOWN'
system = f"""You are very helpful Financial assistant.User will request␣
 ↪financial data/information for a company.You are to return company's symbol␣
 ↪on a stock market.
            do not make anything up. if you do not know reply {UNKNOWN}.
"""

extraction_prompt = ChatPromptTemplate.from_messages(
     [
        ("system", system),
        ("human", "{request}"),
    ]
)
```

[134]:
```
extraction_chain = extraction_prompt | llm.with_structured_output(Extraction)
```

11 Testing extraction chain

[135]:
```
result: Extraction = extraction_chain.invoke({"request": "What is the stock␣
 ↪price of Apple?"})
print(result.symbol)
```

```
AAPL
```

12 Graph State

[136]:
```
class GraphState(TypedDict):
  """
   Represents the state of our graph.

   Attributes:
       symbol: The symbol of the company.
       income_statement: The income statement of the company.
       company_financials: The company financials of the company.
       stock_price: The stock price of the company.
  """
  symbol: str
  request: str
  income_statement: IncomeStatement
  company_financials: CompanyFinancials
  stock_price: StockPrice
```

[137]:
```
from langgraph.graph import END, StateGraph
```

13 Graph Nodes

[138]:
```
def save_md_report_to_file(md_report: str, filename: str = "financial_report.
↪md"):
  """Saves the provided markdown report to a file."""
  with open(filename, "w") as f:
    f.write(md_report)
  print(f"Markdown report saved to {filename}")
```

[145]:
```
def extraction_node(state: GraphState):
  print('extraction_node')
  print('State', state)
  try:
    result: Extraction = extraction_chain.invoke(state['request'])
    state['symbol'] = result.symbol
  except Exception as e:
    print('Error:', e)
    state['symbol'] = UNKNOWN

  print('Symbol:', state['symbol'])
  return state

def get_income_statement_node(state: GraphState):
  print('get_income_statement_node')
  print('Symbol:', state['symbol'])
  result: IncomeStatement = get_income_statement(state['symbol'])
  return {'income_statement': result}

def get_company_financials_node(state: GraphState):
  print('get_company_financials_node')
  print('Symbol:', state['symbol'])
  result: CompanyFinancials = get_company_financials(state['symbol'])
  return {'company_financials': result}

def get_stock_price_node(state: GraphState):
  print('get_stock_price_node')
  print('Symbol:', state['symbol'])
  result: StockPrice = get_stock_price(state['symbol'])
  return {'stock_price': result}

def error_node(state: GraphState) -> str:
    return f"""
    Unknown Symbol: {state['symbol']}
    Can not produce report for this symbol.
    """


def generate_markdown_report_node(state: GraphState) -> str:
    """
    Generates a markdown report from the GraphState instance.
    """
    company_financials = state['company_financials'] if 'company_financials' in␣
  ↪state else None
    income_statement = state['income_statement'] if ('income_statement' in␣
  ↪state) else None
    stock_price = state['stock_price'] if ('stock_price' in state) else None
    md_report = generate_markdown_report(company_financials=company_financials,␣
  ↪income_statement=income_statement, stock_price=stock_price)
    file_name = f"{state['symbol']}_financial_report.md"
    save_md_report_to_file(md_report, filename= file_name)
    return state
```

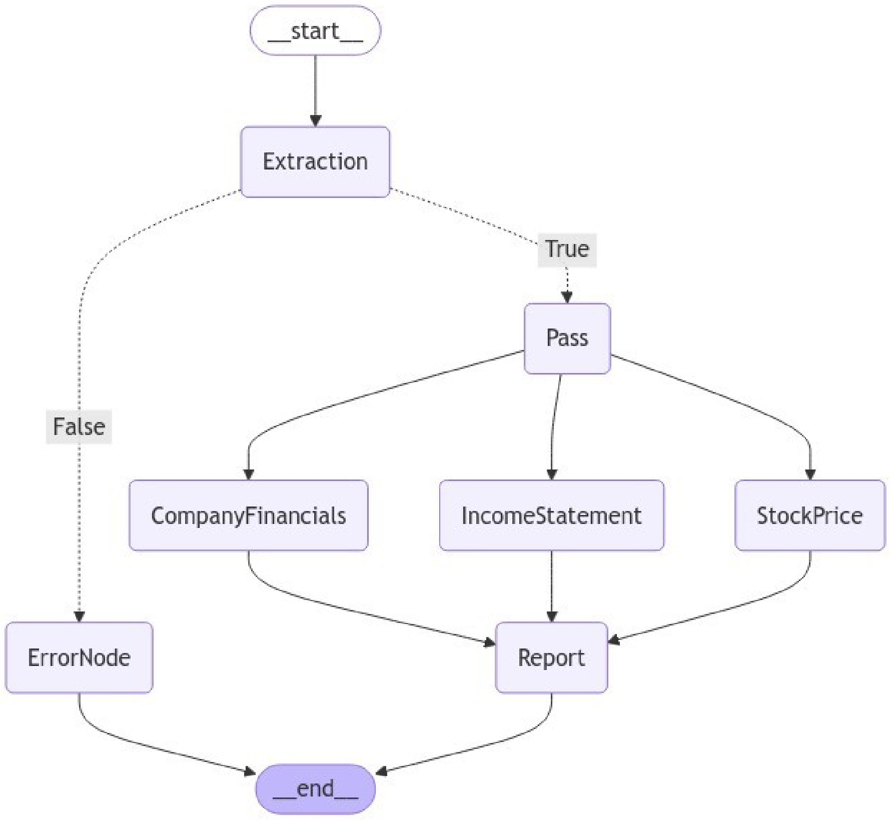
[146]:
```
def is_there_symbol(state: GraphState):
  print('is_there_symbol')
  print('State', state)
  if state['symbol']  == UNKNOWN:
    print('Symbol:', UNKNOWN)
    return False

  return True
```

[147]:
```
EXTRACTION = 'Extraction'
STOCK_PRICE = 'StockPrice'
INCOME_STATEMENT ='IncomeStatement'
```

```
COMPANY_FINANCIALS ='CompanyFinancials'
ERROR_NODE='ErrorNode'
REPORT = 'Report'
PASS = 'Pass'

workflow = StateGraph(GraphState)
workflow.add_node(EXTRACTION, extraction_node)
workflow.add_node(PASS, lambda state: state)
workflow.add_node(INCOME_STATEMENT, get_income_statement_node)
workflow.add_node(COMPANY_FINANCIALS, get_company_financials_node)
workflow.add_node(STOCK_PRICE, get_stock_price_node)
workflow.add_node(REPORT, generate_markdown_report_node)
workflow.add_node(ERROR_NODE, error_node)

workflow.set_entry_point(EXTRACTION)

workflow.add_conditional_edges(EXTRACTION, is_there_symbol, {True:PASS, False:␣
 ↪ERROR_NODE})


workflow.add_edge(PASS,INCOME_STATEMENT)
workflow.add_edge(PASS,COMPANY_FINANCIALS)
workflow.add_edge(PASS,STOCK_PRICE)

workflow.add_edge(INCOME_STATEMENT,REPORT)
workflow.add_edge(COMPANY_FINANCIALS,REPORT)
workflow.add_edge(STOCK_PRICE,REPORT)

workflow.add_edge(REPORT, END)
workflow.add_edge(ERROR_NODE, END)


app = workflow.compile()

# app.debug = True

app.get_graph().draw_mermaid_png(output_file_path="financial_data_report_graph.
 ↪png")
```

[147]:

```
b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00\x00\x01\x00\x01\x00\x00\xff\xe2\x
01\xd8ICC_PROFILE\x00\x01\x01\x00\x00\x01\xc8\x00\x00\x00\x00\x040\x00\x00mntrRG
B XYZ \x07\xe0\x00\x01\x00\x01\x00\x00\x00\x00\x00\x00acsp\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x01\x00\x00\xf6\xd6\x00\x01\x00\x00\x00\x00\xd3-
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\tdesc\x00\x00\x00\xf0\x00\x00\x00$r
```



__start__
Extraction
True
False
Pass
CompanyFinancials
IncomeStatement
StockPrice
ErrorNode
Report
__end__

```
d7\xfa\xdfRcq\xeed\xb8\x9cmZr\xc5#\xabK\x04\xfd\xa4\xbd\xaf;^\xd96 \x00\xc6\x91\
xe4\x8e\xff\x00>\xe1k\xcc\xfe\x98\xcd\xc1\x94\xcfe\xa2\xc1\xde\xbf0\x1d\xae+\xe6
]F(O=\xba\xed\xa9\x1b\x1d$-
;\t\x0b^C\x80\x1d\xe5\x87\xce\x15\xcf\x87V\xec\xe7x\xa9\xac\xb3g\r\x95\xc5\xe3\x
edc\xf1\xd1@\xfc\xa5\'\xd62\xb9\x86\xc7>\xc1\xc3~\x9c\xc3q\xdf\xd4t\xd8\x8d\xf1\
xa7\x1b\x12\xacJi\x9dZ\xf7m\xd5:\xfc\x86\xd1DE\xe9#\xe2x\x8c\xd19\x82G\xc4\xee\x
f6\xc9\x19\xd9\xccp\xea\x1c\x0f\x98\x83\xb1\x1f2\xde\x18|V\x03\x8b\x9a#\x17wQ`qy
\x97O\x0f,\xd1\xde\xa9\x1c\xedl\xad%\x92\x01\xcc\x0e\xc0=\xaeZIn\xee\n\xd6}n\x1b
b\xcb\xc6\xdd\xbb\xecYg\xa3\x92I\xe4\x91\x87\xe9k\x81\xfaW\xce|r\x8ag\x02\x9a\xe
7lO\xd6&\xff\x00Hg\x1b\x16]5\xa61\x1a7\x07W\r\x81\xc6T\xc3\xe2j\x82 \xa5J\x16\xc
5\x14{\xb8\xb9\xdb5\xa0\r\xcb\x8b\x9cO\x9c\x92OR\xa4\xd1\x17\xc5\x02" ""\x02"
""\x02" ""\x02" ""\x02" ""\x02" ""\x02" ""\x02" ""\x02" ""\x02" ""\x02" ""\x02"
""\x02" ""\x02" ""\x02" ""\x02" ""\x02" \xc7\xc8P\x83+B\xcd+Q\x89\xabX\x8d\xd1K\
x19\xfb\xa6\xb8lG\xe6+\x9b\xf3\xdar\xee\x8e\xca\x9c]\xfey6\x1b\xd6\xb6\xef{j1\xf
7@\xfd\xf0\xe9\xcc\xde\xf0z\xf7\x16\x93\xd3\x0b\x037\x82\xc7\xea<{\xe9d\xeaEr\xa
b\xce\xe69F\xfb\x1f3\x81\xef\x04y\x88\xd8\x85\xea\xf4\x0e\x9fWB\xaeo\x17\xa6v\xc
7\xde\x0f7!f8c\xa4u\x0eJl\x86SL\xe2r\x17\xa6\xdb\xb4\xb3f\x9crH\xfd\x80h\xdd\xc4
nv\x00\x0f\x98,?q\x8d\x04\x7f\xfb\x1b\x83;\x7f\xf7\x08\xbfet.C\x80\xd5_+\x9d\x8e
\xce\xde\xa6\xc2w\x11N\xc6N\xd6\xfc\x80\x90\x1d\xb7\xce\xe2\xb0}\xc0\xee\xfcj?\x
ee\xf6\xfe\xda\xfax\xf8\x87\xc3\xea\xfcS1\x7f\x9d3\xcaK|\xda\xa7\x03\xa71Z^\x8f\
x81a\xf1\xd5qu9\xcb\xfb\n\x9166s\x1e\xf3\xb0\x1bo\xd0)\x15\xb1}\xc0\xee\xfcj?\xe
e\xf6\xfe\xda{\x81\xdd\xf8\xd4\x7f\xdd\xed\xfd\xb5\xba>)\xd0\xa2-
\x15\xfbO#\'\xe6\xd7J3Pi|>\xac\xa6\xca\x99\xaceL\xadh\xe4\x12\xb2\x1b\x90\xb6V5\
xe0\x10\x1c\x03\x81\xeb\xb1#\x7f\x94\xad\xb1\xee\x07w\xe3Q\xff\x00w\xb7\xf6\xd3\
xdc\x0e\xef\xc6\xa3\xfe\xefo\xed\xa4\xfcO\xa1U\x16\x9a\xfd\xa7\x91\x93\xf3h\x96\
xf0kA\xb48\r\x1d\x83\x01\xc3b\x05\x08\xba\x8d\xf7\xeb\xd3\xe4\x0b3\x0b\xc3=%\xa7
21\xdf\xc5i\xacV:\xf4`\x86X\xabM\x91\xc8\xd0F\xc7g\x01\xb8\xdc\x12\x16\xea\xf7\x
03\xbb\xf1\xa8\xff\x00\xbb\xdb\xfbk>\x87\x01i6@\xec\x96n\xfd\xd8\xc1\xdc\xc3\x00
ev\xbb\xe4%\xa0\xbf\xf38-
3\xf1\x1f\x87\xd1\xae&\'\xca\x99\xe5\x06O\xcd\xaf4\xe6\x96\xb5\xad\xb2\x87\x1bW\
x9e:\xe3o\x0c\xb6\xc3\xb0\xae\xc3\xe8?~Gp\xfaO@\xba>\xa5Xh\xd5\x86\xb5x\xdb\x14\
x10\xb1\xb1\xc7\x1b{\x9a\xd06\x00|\xc0/\x0c>\x12\x86\x9f\xa1\x1d,mHiUf\xe4E\x0bC
F\xe7\xbc\x9fI\'\xa9\'\xa9=\xeb5|\xbfO\xe9\xd5t\xda\xe3U\xa9\x8d\x91\xf7\x93\xe4
""\xf2\xc1\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x
11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x
11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x
11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x
11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x
11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x
11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x
11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x01\x11\x10\x11\x11\x07\xff\xd9'
```

[148]:
```python
# request = "tell me about Apple?"
request = "tell me about Uber?"
app.invoke(input = {"request": request})
```

```
extraction_node
```

```
State {'request': 'tell me about Uber?'}
Symbol: UBER
is_there_symbol
State {'request': 'tell me about Uber?', 'symbol': 'UBER'}
get_income_statement_node
Symbol: UBER
get_company_financials_node
Symbol: get_stock_price_node
Symbol:UBER UBER

Markdown report saved to UBER_financial_report.md
```

```
[148]: {'symbol': 'UBER',
        'request': 'tell me about Uber?',
        'income_statement': IncomeStatement(date_field=datetime.date(2023, 12, 31),
       revenue=37281000000.0, gross_profit=14824000000.0, net_income=1887000000.0,
       ebitda=2219000000.0, eps=0.93, eps_diluted=0.87),
        'company_financials': CompanyFinancials(symbol='UBER', companyName='Uber
       Technologies, Inc.', marketCap=164671677200.0, industry='Software -
       Application', sector='Technology', website='https://www.uber.com', beta=1.331,
       price=78.38),
        'stock_price': StockPrice(symbol='UBER', price=78.38, volume=13655049.0,
       priceAvg50=74.847, priceAvg200=71.54585, eps=0.92, pe=85.2,
       earningsAnnouncement=datetime.datetime(2024, 10, 31, 0, 0, tzinfo=TzInfo(UTC)))}
```