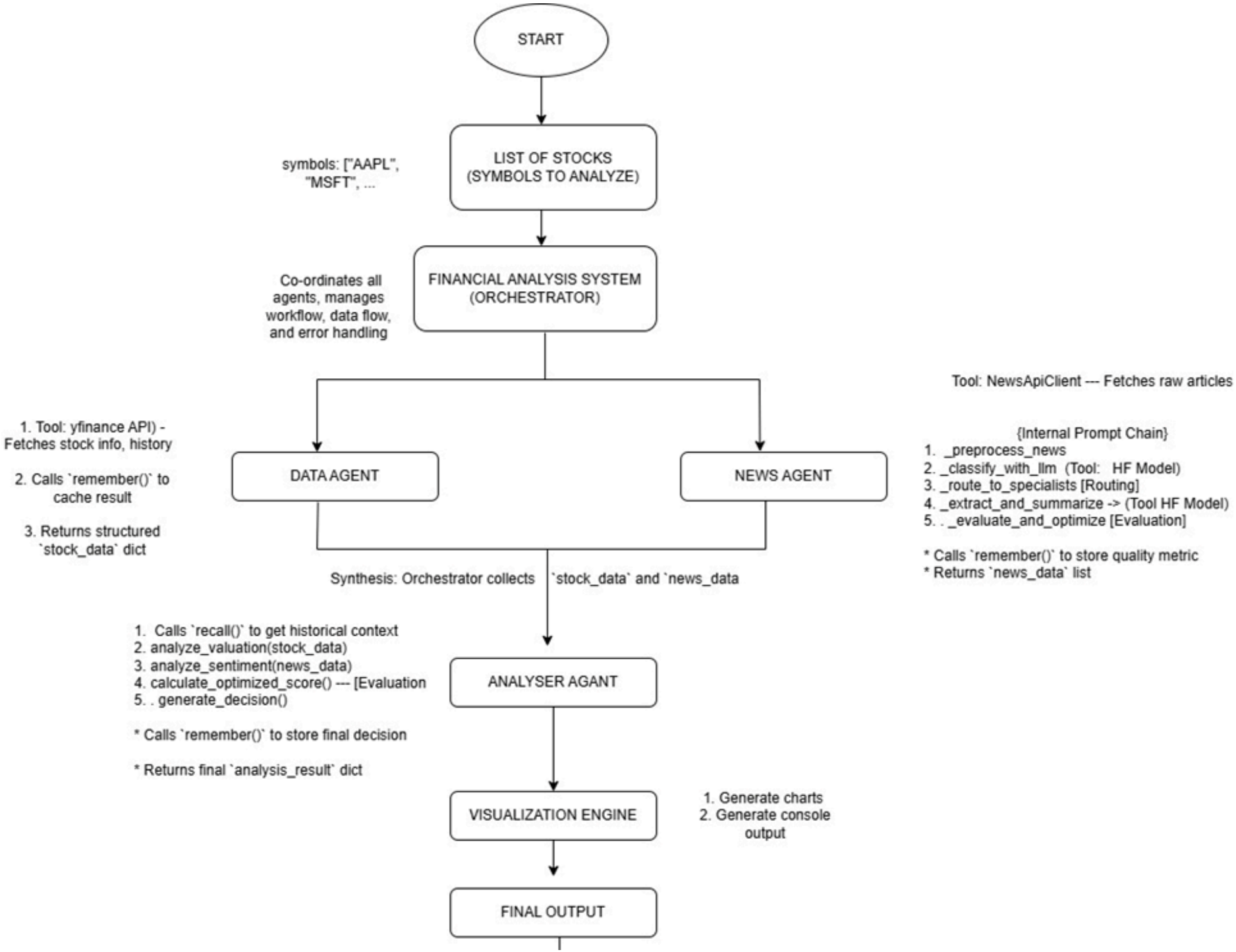
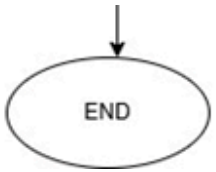
AAI 520 - FINAL TEAM PROJECT: MULTI AGENT FINANCIAL ANALYSIS SYSTEM

GROUP NUMBER: 13

TEAM MEMBER: SETHURAMAN KRISHNASAMY

Work-flow diagram of Multi-Agent Financial Analysis System pipeline


START
symbols: ["AAPL", "MSFT", ...
LIST OF STOCKS (SYMBOLS TO ANALYZE)
Co-ordinates all agents, manages workflow, data flow, and error handling
FINANCIAL ANALYSIS SYSTEM (ORCHESTRATOR)
1. Tool: yfinance API) - Fetches stock info, history
2. Calls `remember()` to cache result
3. Returns structured `stock_data` dict
DATA AGENT
NEWS AGENT
Tool: NewsApiClient --- Fetches raw articles
{Internal Prompt Chain}
1. _preprocess_news
2. _classify_with_llm (Tool: HF Model)
3. _route_to_specialists [Routing]
4. _extract_and_summarize -> (Tool HF Model)
5. . _evaluate_and_optimize [Evaluation]
* Calls `remember()` to store quality metric
* Returns `news_data` list
Synthesis: Orchestrator collects `stock_data` and `news_data
1. Calls `recall()` to get historical context
2. analyze_valuation(stock_data)
3. analyze_sentiment(news_data)
4. calculate_optimized_score() --- [Evaluation
5. . generate_decision()
* Calls `remember()` to store final decision
* Returns final `analysis_result` dict
ANALYSER AGANT
VISUALIZATION ENGINE
1. Generate charts
2. Generate console output
FINAL OUTPUT



1. Start & Orchestration

- `START`: This is the trigger for the entire process.
- `FinancialAnalysisSystem (Orchestrator)`: This is the main "brain" or "project manager" of the system. It receives the initial user input (the list of stock symbols) and begins the workflow by coordinating and delegating tasks to the specialized agents.

---

2. Parallel Data Gathering & Analysis

The Orchestrator initiates two tasks simultaneously to save time:

- `DataAgent` **(The Quantitative Analyst - Left Path):** This agent is a process block. Its sole job is to gather all **quantitative** data. It connects to financial APIs (like Yahoo Finance) to get current prices, P/E ratios, market cap, and historical data for technical calculations (like RSI).
- `NewsAgent` **(The Qualitative Analyst) - Right Path):** This agent is also a process block.This agent is a sophisticated process block that runs concurrently with the DataAgent. Its mission is to gather and analyze all unstructured, qualitative data. It connects to news APIs (like NewsAPI.org) to find the latest articles related to a stock. It then processes these articles through its own internal multi-step pipeline, which uses Large Language Models (LLMs). This pipeline cleans the text, performs advanced sentiment analysis (classifying news as positive, neutral, or negative), summarizes long articles, and even includes a self-evaluation step to score the quality of its own analysis.

---

3. Synthesis & Decision

- `AnalysisAgent`: The parallel paths converge here. This agent takes the structured output from *both* the `DataAgent` (the numbers) and the `NewsAgent` (the sentiment scores). It acts as the synthesizer, combining all these factors, calculating a final `Overall Score`, and generating the `AI Decision` (e.g., "BUY", "HOLD") and the detailed `Reasoning` string.

---

4. Parallel Output Generation

Once the `AnalysisAgent` has its final results, the Orchestrator passes this complete data to two different output components, which also run in parallel:

- **`VisualizationEngine (Charts)` (Left Path):** This is an output block. It takes the final data and generates all the visual elements: the radar chart, the sentiment pie chart, the price history, and the other plots.
- **`Console Display (Table & Reasoning)` (Right Path):** This is also an output block. It formats the data into the main, comprehensive summary table and prints the detailed textual reasoning in console

---

5. End

- **`END`:** Both output paths lead to the end of the process, signifying that the analysis is complete and all results have been delivered to the user.

DESCRIPTION OF PRIMARY DATA SOURCES:

1. Yahoo Finance API: Real-time stock prices, financial statements, historical data
2. Financial News APIs: NewsAPI.org (https://newsapi.org)

Section 1: Install the following required packages

yfinance: To fetch financial market data from Yahoo Finance

newsapi-python: Python client to access NewsAPI.org for news articles

alpha-vantage: API client for Alpha Vantage stock and forex data

fredapi: To access Federal Reserve Economic Data (FRED)

pandas, numpy: Data analytics and numerical computing libraries

matplotlib, seaborn: Plotting and data visualization libraries

```
# Section 1: Install rwquired python packages
# Yahoo Finance data API
# newsapi-python: NewsAPI client
# alpha-vantage: Financial market data API
# fredapi: Federal Reserve economic data client
# pandas, numpy: Data processing libraries
# matplotlib, seaborn: Plotting libraries

!pip install yfinance newsapi-python alpha-vantage fredapi pandas numpy matplotlib seaborn
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.12/dist-packages (0.2.66)
Collecting newsapi-python
  Downloading newsapi_python-0.2.7-py2.py3-none-any.whl.metadata (1.2 kB)
Collecting alpha-vantage
  Downloading alpha_vantage-3.0.0-py3-none-any.whl.metadata (12 kB)
Collecting fredapi
  Downloading fredapi-0.5.2-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.12/dist-packages (from yfinance) (2.32.4)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.12/dist-packages (from yfinance) (0.0.12)
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from yfinance) (4.5.0)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.12/dist-packages (from yfinance) (2025.2)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.12/dist-packages (from yfinance) (2.4.6)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.12/dist-packages (from yfinance) (3.18.2)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.12/dist-packages (from yfinance) (4.13.5)
Requirement already satisfied: curl_cffi>=0.7 in /usr/local/lib/python3.12/dist-packages (from yfinance) (0.13.0)
Requirement already satisfied: protobuf>=3.19.0 in /usr/local/lib/python3.12/dist-packages (from yfinance) (5.29.5)
Requirement already satisfied: websockets>=13.0 in /usr/local/lib/python3.12/dist-packages (from yfinance) (15.0.1)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.12/dist-packages (from alpha-vantage) (3.13.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.1)
```

```
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.5)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist-packages (from beautifulsoup4>=4.11.1->yfinance)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.12/dist-packages (from beautifulsoup4>=4.11.1-
Requirement already satisfied: cffi>=1.12.0 in /usr/local/lib/python3.12/dist-packages (from curl_cffi>=0.7->yfinance) (2.0.0)
Requirement already satisfied: certifi>=2024.2.2 in /usr/local/lib/python3.12/dist-packages (from curl_cffi>=0.7->yfinance) (202
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.31->yfinanc
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.31->yfinance) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.31->yfinance) (2.
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->alpha-vantage)
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->alpha-vantage) (1.4.0)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->alpha-vantage) (25.4.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp->alpha-vantage) (1.8.0
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp->alpha-vantage) (6.7
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->alpha-vantage) (0.4.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->alpha-vantage) (1.22.
Requirement already satisfied: pycparser in /usr/local/lib/python3.12/dist-packages (from cffi>=1.12.0->curl_cffi>=0.7->yfinance
Downloading newsapi_python-0.2.7-py2.py3-none-any.whl (7.9 kB)
Downloading alpha_vantage-3.0.0-py3-none-any.whl (35 kB)
Downloading fredapi-0.5.2-py3-none-any.whl (11 kB)
Installing collected packages: newsapi-python, fredapi, alpha-vantage
Successfully installed alpha-vantage-3.0.0 fredapi-0.5.2 newsapi-python-0.2.7
```

SECTION 1: Install required python packages

This Python code snippet imports a comprehensive set of libraries necessary for financial data analysis, natural language processing (NLP) with large language models (LLMs), and news data retrieval.

a) Imports for Data Handling and Visualization:**

Libraries like `yfinance`, `pandas`, `numpy`, `matplotlib.pyplot`, and `seaborn` are imported for fetching stock market data, manipulating datasets, and creating informative visualizations. Warnings are suppressed to maintain clean output during execution.

b) Time and File Utilities:**

Modules like `datetime`, `os`, `json`, and `re` handle timestamps, file paths, JSON serialization, and regular expressions—essential for data cleaning and persistent storage.

c) Typing and Annotations:**

`typing` imports (`Dict`, `List`, `Any`, etc.) facilitate explicit type hinting for better code clarity and debugging, especially important in larger complex systems.

d) LLM and NLP Integration:**

The `transformers` library is used to bring powerful pretrained models for text classification (`pipeline`, `AutoTokenizer`, `AutoModelForSequenceClassification`) which enable sentiment analysis or summarization tasks based on financial news. The `torch` library supports deep learning model computations.

e) News API Integration:**

The `NewsApiClient` is imported to interact with the NewsAPI service for retrieving real-time news articles related to stock symbols, supporting multi-source news ingestion.

```
 # Library to fetch financial data from Yahoo Finance API
import yfinance as yf

 # Data manipulation and analysis library
import  pandas as pd

# Numerical computing library for array operations
import numpy as np

# HTTP library for API requests
import requests
```

```
# Handle JSON serialization and deserialization
import json

# Operating system interfaces for file handling
import os

# Work with date and time objects
from datetime import datetime

# Regular expressions for text processing
import re

# Type hints for improved code clarity
from typing import Dict, List, Any, Optional, Tuple

# Plotting library for visualizations
import matplotlib.pyplot as plt

# Statistical data visualization library built on matplotlib
import seaborn as sns

import warnings  # Manage warning messages
warnings.filterwarnings('ignore')  # Suppress warnings to keep output clean

# Import components for Large Language Model (LLM) pipelines
from transformers import pipeline, AutoTokenizer, AutoModelForSequenceClassification
import torch  # Deep learning framework for LLM computations
from newsapi import NewsApiClient  # Client to fetch news articles from NewsAPI service
```

SECTION 2: Research Agent Base Architecture with Persistent Memory and Data Serialization

This code defines two main components: a utility class for safely serializing Python data and a base class for research agents with memory persistence and logging capabilities.

1. **DataSerializer Class:**
   This class provides a static method `make_json_serializable` to convert complex Python objects into formats that can be serialized into JSON. It handles nested dictionaries, lists, tuples, pandas DataFrames and Series, datetime objects, and numeric types from pandas and NumPy. The goal is to ensure all data stored or logged can be safely written to JSON without errors, especially useful when dealing with financial or time-series data.
2. **ResearchAgent Class:**
   This abstract base class represents an agent with a name and persistent memory stored as a JSON file. It maintains an internal dictionary called `memory` that stores key-value pairs alongside timestamps and iteration counts. Methods include:
   - `log`: Prints timestamped log messages with severity levels.
   - `remember`: Serializes and saves key-value data in memory, persisting it to a file.
   - `recall`: Retrieves stored values by key, logging the recall action.
   - `load_memory`: Loads prior memory from the JSON file if it exists, handling exceptions gracefully.
   - `save_memory`: Writes the current memory dictionary back to disk, with error reporting.

```
#================================================================================================
# SECTION 2: Research Agent Base Architecture with Persistent Memory and Data Serialization
#================================================================================================

print("--- Section 1: Imports and Setup ---")

# -Plot Configuration
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 12
print("Plot styles configured.")


print("--- Section 2: Research Agent Base Architecture ---")
```

```
print("Core libraries imported.")

# DataSerializer for JSON safe conversion
class DataSerializer:
    @staticmethod
    def make_json_serializable(obj: Any) -> Any:
        if isinstance(obj, dict):
            return {str(k): DataSerializer.make_json_serializable(v) for k, v in obj.items()}
        elif isinstance(obj, (list, tuple)):
            return [DataSerializer.make_json_serializable(item) for item in obj]
        elif isinstance(obj, (pd.Timestamp, datetime)):
            return obj.isoformat()
        elif isinstance(obj, pd.DataFrame):
            # Convert DataFrame to dict (list of records) first
            records = obj.to_dict('records')
            return DataSerializer.make_json_serializable(records) # Recursive call on the list
        elif isinstance(obj, pd.Series):
             # Convert Series to dict first
            series_dict = obj.to_dict()
            return DataSerializer.make_json_serializable(series_dict) # Recursive call on the dict
        elif hasattr(obj, 'dtype') and pd.api.types.is_numeric_dtype(obj):
            # Handle Pandas numeric types explicitly (like Int64 etc.)
            return float(obj) if pd.notna(obj) else None
        elif isinstance(obj, (np.int64, np.float64, np.number)): # Added np.number for broader numpy numeric types
            # Convert numpy numeric types to standard Python types
            return obj.item() if pd.notna(obj) else None
        elif isinstance(obj, (int, float, str, bool)) or obj is None:
             # Handle standard JSON-safe types
            return obj
        else:
            # Fallback for unknown types: convert to string
            return str(obj)

print("✅ DataSerializer class defined successfully.")

# Base ResearchAgent with memory
class ResearchAgent:
```

```
def __init__(self, name: str):
    self.name = name
    self.memory = {}
    self.iteration_count = 0
    # Ensure memory files are stored in a dedicated directory if needed
    memory_dir = "agent_memory"
    os.makedirs(memory_dir, exist_ok=True) # Creates dir if it doesn't exist
    self.memory_file = os.path.join(memory_dir, f"{name}_memory.json")
    self.load_memory()
    # Initialisation confirmation is now part of the successful execution message below

def log(self, message: str, level: str = "INFO"):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    print(f"[{timestamp}] {self.name} ({level}): {message}")

def remember(self, key: str, value: Any):
    try:
        serialized_value = DataSerializer.make_json_serializable(value)
        self.memory[key] = {
            'value': serialized_value,
            'timestamp': datetime.now().isoformat(),
            'iteration': self.iteration_count
        }
        self.save_memory()
        self.log(f"🧠 Stored in memory: '{key}'") # Updated log message
    except Exception as e:
        self.log(f"Error during serialization or saving for key '{key}': {str(e)}", "ERROR")

def recall(self, key: str) -> Optional[Any]:
    if key in self.memory:
        self.log(f"💡 Recalled from memory: '{key}'") # Updated log message
        # Optionally, deserialization logic can be added here if needed in future
        return self.memory[key]['value']
    self.log(f"Key '{key}' not found in memory.", "WARNING") # Added warning for missing key
    return None

def load_memory(self):
    if os.path.exists(self.memory_file):
```

```
            try:
                with open(self.memory_file, 'r') as f:
                    self.memory = json.load(f)
                # Find the max iteration count from loaded memory if exists
                if self.memory:
                    self.iteration_count = max(item.get('iteration', 0) for item in self.memory.values())
                self.log(f"💾 Loaded memory from '{self.memory_file}' (Last iteration: {self.iteration_count})") # Added filena
            except json.JSONDecodeError as e:
                self.log(f"Error decoding JSON from memory file '{self.memory_file}': {str(e)}. Starting with empty memory.", "
                self.memory = {} # Reset memory if file is corrupt
            except Exception as e:
                self.log(f"Error loading memory from '{self.memory_file}': {str(e)}", "ERROR")
                self.memory = {} # Fallback to empty memory
        else:
             self.log(f"No memory file found at '{self.memory_file}'. Starting fresh.", "INFO") # Info if no file exists

    def save_memory(self):
        try:
            with open(self.memory_file, 'w') as f:
                json.dump(self.memory, f, indent=4) # Changed indent for better readability
            # self.log("💾 Saved memory to file") # Removed redundant save log, covered by remember()
        except Exception as e:
            self.log(f"Error saving memory to '{self.memory_file}': {str(e)}", "ERROR")

    # Assuming increment_iteration might be used elsewhere, keeping it
    def increment_iteration(self):
        self.iteration_count += 1
        self.log(f"🏁 Starting Iteration: {self.iteration_count}")

print("✅ ResearchAgent class defined successfully.")
print("\n✅ Section 2: Base Agent Architecture definition completed successfully.")
print("-" * 50 + "\n") # Added separator for clarity
```

```
--- Section 1: Imports and Setup ---
Plot styles configured.
--- Section 2: Research Agent Base Architecture ---
Core libraries imported.
✅ DataSerializer class defined successfully.
✅ ResearchAgent class defined successfully.
```

```
✅ Section 2: Base Agent Architecture definition completed successfully.
------------------------------------------------
```

SECTION 3: IMPLEMENTATION DETAILS OF NEWS AGENT WITH WORKFLOW PATTERNS

---

a. Description: The News Agent Orchestrates a sophisticated, multi-stage pipeline to ingest, process, and analyze financial news. It transforms raw articles into enriched data by executing a chain of LLM-powered analyses, including sentiment classification, summarization, and routing to specialized sub-analyzers.

b. Key Methods/Features:

- **`fetch_news(symbol)`:** The main orchestration method that manages the entire end-to-end pipeline, from ingestion to evaluation.
- **`_ingest_news(symbol)`:** Fetches raw news articles from the `NewsApiClient`, which is more robust than the basic `yf.Ticker` method.
- **`_process_news_pipeline(...)`:** The core **PROMPT CHAINING** method. It sequentially executes the full workflow for a single article:
    1. `_preprocess_news` (cleans text, extracts metadata)
    2. `_classify_with_llm` (generates sentiment)
    3. `_route_to_specialists` (categorizes news)
    4. `_extract_and_summarize` (creates a summary)
- **`_classify_with_llm(...)`:** Uses the `cardiffnlp/twitter-roberta-base-sentiment-latest` model. This is enhanced by:
    - `_add_financial_context`: A **prompt engineering** step that adds guiding text.
    - `_apply_financial_context_adjustments`: A correction layer that uses financial rules to override or boost the LLM's output.
- **`_fallback_sentiment_analysis(...)`:** A robust, keyword-based backup system (with contextual boosts) that provides sentiment if the LLM fails.

- `_route_to_specialists(...)`: Implements the **ROUTING** pattern, directing news to specialized modules like `EarningsAnalyzer` or `MarketAnalyzer`.
- `_extract_and_summarize(...)`: Uses the `facebook/bart-large-cnn` model to generate a dynamic, AI-powered summary of the article.
- `_evaluate_and_optimize(...)`: An **EVALUATOR** function that scores the quality of the sentiment and summaries, enabling a feedback loop for continuous improvement.

c. Pipeline Role: This agent is the primary engine for **PROMPT CHAINING**, **ROUTING**, and **EVALUATION** on unstructured news data. It doesn't just feed raw text; it generates a rich, multi-layered data object (containing summaries, classified sentiment, specialist insights, and quality scores) that is then passed to the `ReportingAgent` for final synthesis.

SECTION 3: EARNINGS ANALYZER (Specialized Agent)

a. Description:

This agent Acts as a specialized sub agent under the Routing Pattern hierarchy of the NewsAgent. It focuses on detecting and interpreting earnings related news such as quarterly results, EPS beats/misses, and guidance updates. Its duty is to extract clear, financially relevant insights that represent corporate performance impact.

b. Key Methods/Features:

i) Embedded Keyword Detection (earnings_keywords):

```
Maintains a dedicated vocabulary set (earnings, EPS, revenue, quarter, guidance, beat, miss etc.) to classify items as earnings r
```

ii) Insight Extraction:

Derives structured insights (e.g., “Earnings exceeded expectations”, “Future guidance provided”) from the headline context.

iii) Impact Scoring (calculate_earnings_impact):

Computes a 0–1 impact value where positive terms (beat, surprise, raise guidance) increase score and negative terms (miss) reduce

iv) Structured Output:

Returns a dictionary with flags – is_earnings, insights, impact_score, and category = "earnings" for integration by NewsAgent.

c. Pipeline Role:

Serves as the Earnings focused Classifier within the NewsAgent Routing system, supporting sentiment and summary derivation for ea

```
# ================================================================================
# SECTION 3: SPECIALIZED AGENTS FOR ROUTING PATTERN
#  EARNINGS ANALYZER
# ================================================================================

class EarningsAnalyzer(ResearchAgent):
    """Specialized agent for earnings-related news analysis"""
    def __init__(self):
        super().__init__("EarningsAnalyzer")
        self.earnings_keywords = {
            'earnings', 'eps', 'revenue', 'profit', 'loss', 'quarter', 'q1', 'q2', 'q3', 'q4',
            'guidance', 'forecast', 'beat', 'miss', 'results', 'financial results'
        }

    def analyze_earnings(self, news_item: Dict) -> Dict[str, Any]:
        """Analyze earnings-specific content"""
        title = news_item.get('title', '').lower()
```

```
        content = news_item.get('content', '').lower()

        # Check if this is earnings-related
        is_earnings_news = any(keyword in title or keyword in content
                               for keyword in self.earnings_keywords)

        if not is_earnings_news:
            return {'is_earnings': False}

        # Extract earnings-specific insights
        insights = []
        if 'beat' in title:
            insights.append("Earnings exceeded expectations")
        if 'miss' in title:
            insights.append("Earnings missed expectations")
        if 'guidance' in title or 'forecast' in title:
            insights.append("Future guidance provided")

        return {
            'is_earnings': True,
            'insights': insights,
            'impact_score': self.calculate_earnings_impact(title),
            'category': 'earnings'
        }

    def calculate_earnings_impact(self, title: str) -> float:
        """Calculate impact score for earnings news"""
        score = 0.5  # Base score

        # Impact modifiers
        if 'beat' in title.lower():
            score += 0.3
        if 'miss' in title.lower():
            score -= 0.3
        if 'surprise' in title.lower():
            score += 0.2
        if 'raise' in title.lower() and 'guidance' in title.lower():
            score += 0.2
```

```
        return max(0.0, min(1.0, score))
```

SECTION 4: MARKET ANALYZER (SPECIEALIZED AGENT)

a. Description:

Dedicated to evaluating macro-level and marketwide news about indices, interest rates, or economic events. It filters content with market-wide s cope (the Dow, NASDAQ, inflation, Fed rate moves) and infers overall market climate sentiment.

b. Key Methods / Features:

i. Market Keyword Registry (market_keywords): Looks for core macroeconomic triggers (“Fed”, “interest rate”, “rally”, “selloff”, “inflation”).
ii. Sentiment Assignment: Labels detected news as Positive (“market rally”, “bull run”) or Negative (“selloff”, “bear”, “drop”).
iii. Insight Collection: Generates short phrases highlighting market momentum (“Positive market momentum”, “Negative market pressure”).
iv. Categorical Output: Returns structured dict {'is_market', 'sentiment', 'insights', 'category':'market'}.

c. Pipeline Role:

Plugs into NewsAgent’s Routing system to contextualize economic and broad market narratives. Its outputs influence the macro component of t he AnalysisAgent’s decision-weight metrics through sentiment aggregation and market tone assessment.

```
# ================================================================================
# SECTION 4: MARKET ANALYZER
# ================================================================================
class MarketAnalyzer(ResearchAgent):
    """Specialized agent for market and economic news"""
    def __init__(self):
        super().__init__("MarketAnalyzer")
        self.market_keywords = {
            'market', 'dow', 's&p', 'nasdaq', 'fed', 'interest', 'rate', 'inflation',
```

```
            'economy', 'trading', 'volatility', 'bull', 'bear', 'rally', 'selloff'
        }

    def analyze_market(self, news_item: Dict) -> Dict[str, Any]:
        """Analyze market and economic content"""
        title = news_item.get('title', '').lower()

        # Check if this is market-related
        is_market_news = any(keyword in title for keyword in self.market_keywords)

        if not is_market_news:
            return {'is_market': False}

        # Extract market insights
        insights = []
        sentiment = "Neutral"

        if any(word in title for word in ['rally', 'bull', 'gain', 'surge']):
            sentiment = "Positive"
            insights.append("Positive market momentum")
        if any(word in title for word in ['selloff', 'bear', 'drop', 'fall']):
            sentiment = "Negative"
            insights.append("Negative market pressure")

        return {
            'is_market': True,
            'sentiment': sentiment,
            'insights': insights,
            'category': 'market'
        }
```

SECTION 5: Company News Analyzer (Specialized Agent)

a. Description:

Analyzes company-specific stories such as product launches, M&A announcements, and leadership changes. Acts as a focused agent for corp orate activity detection within the NewsAgent Routing layer.

b. Key Methods / Features:

i. Domain Categorization:

```
Scans news titles for keywords (spanning “product”, “launch”, “acquisition”, “merger”, “CEO”, “executive”) to label items as pro
```

ii. Relevance Scoring (calculate_relevance)

```
Computes a 0–1 relevance factor weighted by explicit company mention and priority keywords (breakthrough, major, strategic).
```

iii. Output Schema:

```
Provides {'is_company_news', 'categories', 'relevance_score', 'category':'company_news'} for cross-agent integration.
```

c. Pipeline Role:

```
Complements the Earnings and Market Analyzers by handling micro-level, entity-specific information. Feeds critical qualitative si
```

```
    # ================================================================================
    # SECTION 5: COMPANY NEWS ANALYZER
    # ================================================================================
    class CompanyNewsAnalyzer(ResearchAgent):
        """Specialized agent for company-specific news"""
        def __init__(self):
            super().__init__("CompanyNewsAnalyzer")
```

```
def analyze_company_news(self, news_item: Dict, symbol: str) -> Dict[str, Any]:
    """Analyze company-specific developments"""
    title = news_item.get('title', '').lower()

    # Categorize company news
    categories = []
    if any(word in title for word in ['product', 'launch', 'release', 'new']):
        categories.append('product')
    if any(word in title for word in ['acquisition', 'merge', 'deal', 'partnership']):
        categories.append('corporate_action')
    if any(word in title for word in ['ceo', 'executive', 'leadership', 'hire']):
        categories.append('management')

    return {
        'is_company_news': True,
        'categories': categories,
        'relevance_score': self.calculate_relevance(title, symbol),
        'category': 'company_news'
    }

def calculate_relevance(self, title: str, symbol: str) -> float:
    """Calculate relevance score for company news"""
    score = 0.5

    # Company mention boosts relevance
    if symbol.lower() in title.lower():
        score += 0.3

    # Important keywords boost relevance
    important_keywords = ['breakthrough', 'record', 'major', 'strategic', 'transformative']
    if any(keyword in title for keyword in important_keywords):
        score += 0.2

    return max(0.0, min(1.0, score))
```

Section 6: News Agent - Intelligent Financial News Analysis System

The NewsAgent represents a sophisticated AI-powered financial news processing platform that intelligently analyzes, categorizes, and summarizes market-moving information. This system combines advanced machine learning, domain expertise, and software engineering patterns to transform raw news articles into actionable investment insights.

Built on three core architectural patterns, the system implements a multi-stage processing pipeline:

1. PROMPT CHAINING:

Orchestrates sequential AI transformations where each news item flows through preprocessing, sentiment classification, specialized analysis, and summarization stages. Instead of a single, monolithic analysis, each news item undergoes a multi-stage transformation. The output of each stage becomes the refined input for the next, ensuring that the final summary is built upon layers of contextual understanding.This ensures comprehensive and layered understanding of financial content. The sentiment classification is LLM powered. The model's classification is provided by a set of financial heuristics. For example a product launch classified as "Neutral" is automatically elevated to "Positive"). The output is a structured sentiment object, like {'sentiment': 'Positive', 'score': 0.95, 'context_adjusted': True}. It generates a financially relevant sentiment score that goes beyond simple keyword analysis.

2. ROUTING PATTERN

Automatically directs content to domain-specific analyzers - Earnings Analyzer for financial results, market analysts for generic macroeconomic news such as such as "inflation," "interest rates," "GDP," or "Federal Reserve" and Company news Analyzer for corporate developments. This enables precision analysis tailored to each news type's unique characteristics.

The Routing Pattern acts as the system's intelligent triage center, ensuring that the right news gets to the right expert. By automatically directing content to domain-specific analyzers, the system delivers a precise analysis tailored to each news type's unique characteristics.

3. EVALUATOR-OPTIMIZER:

This continuously monitors processing quality through multiple metrics, enabling real-time performance assessment and systematic improvement. The system self-diagnoses issues and suggests optimizations for maintaining high-quality outputs. The Evaluator-Optimizer pattern is what elevates the system from a simple automated tool to an intelligent agent capable of self-reflection. It functions as an internal quality control department, continuously assessing the performance of its own AI-driven analyses. By scoring its outputs against key metrics and generating actionable feedback, this pattern creates a continuous learning loop, allowing the system to identify its own weaknesses and systematically improve its accuracy and reliability over time.

4. EVALUATOR-OPTIMIZER PATTERN:

The Self-Reflective Quality Control System The Evaluator-Optimizer pattern is what elevates the system from a simple automated tool to an intelligent agent capable of self-reflection. It functions as an internal quality control department, continuously assessing the performance of its own AI-driven analyses. By scoring its outputs against key metrics and generating actionable feedback, this pattern creates a continuous learning loop, allowing the system to identify its own weaknesses and systematically improve its accuracy and reliability over time.

This is the mechanism that builds trust in the agent's insights.

The Three Stages of Self-Improvement

The process of evaluation and optimization follows a clear, cyclical path.

i) Stage 1: The Quality Audit (Quantitative Scoring):

After the ReportingAgent synthesizes its analysis, the EvaluationAgent steps in to perform a quantitative audit. It doesn't assess the content itself, but rather the quality of the AI processes used to generate that content. It measures specific, pre-defined metrics

ii) Stage 2: The Actionable Critique (Feedback Generation):

Raw scores are just numbers; their value comes from interpretation. In this stage, the agent translates the quantitative scores from the audit into qualitative, actionable feedback. It compares the average scores against internal benchmarks.

iii) Stage 3: The Optimization Loop (Closing the Feedback Loop):

This is the most critical stage, where reflection leads to action. The feedback generated in Stage 2 is passed back to the OrchestratorAgent, closing the loop and enabling improvement.

The platform demonstrates exceptional contextual intelligence by understanding financial terminology, recognizing product launch significance, and interpreting business events through an investment lens. It enhances generic sentiment analysis with financial domain knowledge, boosting confidence scores for market-moving events like product announcements and earnings reports.

Robust error handling and graceful degradation ensure system reliability, with comprehensive fallback mechanisms that maintain service availability even when individual components fail. The architecture supports continuous learning and adaptation, making it increasingly effective at identifying financially relevant patterns over time.

This system serves as a powerful tool for investors, analysts, and financial institutions seeking to process vast amounts of news data efficiently while extracting meaningful, actionable intelligence for informed decision-making.

Section 6: News Agent - Sentiment Analysis Component: LLM Model and Pipeline details Details:

i) A powerful, pre-trained RoBERTa-based AI model, fine-tuned by Cardiff University on Twitter data is used, to perform highly accurate sentiment analysis (classifying text as positive, negative, or neutral).

ii) The pipeline function from the Hugging Face transformers library is used to for classifying text sentiment.Below are the details

iii) pipeline("sentiment-analysis", ...): This is the core command. The Hugging Face library builds a complete, end-to-end tool for the specific task of "sentiment-analysis". The pipeline automatically handles all the complex steps as mentioned below:

```
a) Tokenization: Takes the raw text (e.g., a news headline) and converts it into a numerical format the model can understand.

b) Inference: Feeds the numbers through the AI model to get a prediction.
```

```
c) Post-processing: Converts the model's numerical output back into a human-readable result (like "Positive" or "Negative").
```

iv) model="cardiffnlp/twitter-roberta-base-sentiment-latest": This specifies exactly which pre-trained model to use.

v) RoBERTa-base: It's built on RoBERTa, a powerful and robust language model developed by Facebook AI.

vi) cardiffnlp: It has been fine-tuned by the NLP research group at Cardiff University.

vii) sentiment-latest: It's specifically trained on a huge dataset of tweets to be exceptionally good at identifying sentiment (Positive, Negative, Neutral) in modern, informal text.

Section 6: NewsAgent - Design Description:

1. Class Architecture & Inheritance

The NewsAgent class inherits from the ResearchAgent base class, following an **object-oriented hierarchical design**. This structure encourages **code reuse**, consistency in memory management, and centralized logging, while allowing NewsAgent to specialize in news gathering and sentiment processing functions.

2. Specialized Analyzer Initialization (Routing Pattern)

Three separate analyzers are initialized to process distinct types of financial content. This follows the **Routing design pattern**, where news articles are automatically directed to the most relevant analyzer, such as general market updates, company earnings, or macroeconomic reports.

3. LLM Pipeline Initialization (Prompt Chaining)

Two domain-tuned LLMs handle different NLP tasks—one focuses on **sentiment classification**, and the other on **text summarization**. These operate in a **Prompt Chaining pipeline**, sequentially transforming news articles from raw text to structured sentiment and summary outputs.

4. NewsAPI Client Configuration

The class connects to an external **News API service** for real-time financial data retrieval. This integration includes API key management, rate-limit handling, and structured error recovery to maintain continuous data flow even during API downtime.

5. Domain-Specific Vocabulary Enhancement

Customized vocabulary libraries extend the LLM's financial understanding. Positive, negative, and neutral keywords (e.g., "beat estimates," "cut guidance") help fine-tune sentiment detection in financial contexts, ensuring relevance to market behavior.

6. Quality Metrics Tracking (Evaluator–Optimizer Pattern)

The system maintains internal performance metrics for sentiment accuracy, summarization quality, and news source reliability. These metrics enable self-optimization under the **Evaluator–Optimizer pattern**, supporting ongoing model tuning.

7. Main Processing Pipeline Orchestration

A central method orchestrates news ingestion, sentiment evaluation, classification, and summarization. Both **sequential task flow** and **error isolation** ensure resilience and graceful degradation when submodules fail.

8. News Ingestion Layer

Handles all external API calls, manages pagination, result filtering, and query optimization for retrieving relevant company news. It emphasizes efficiency and keyword-based relevance scoring for stocks under analysis.

9. Sequential Processing Pipeline (Prompt Chaining)

News content flows through structured processing stages: preprocessing → classification → routing → summarization. The pipeline ensures data enrichment at each step while maintaining processing order and logical consistency.

10. Text Preprocessing & Cleaning

Normalization prepares text for downstream analysis by removing noise, handling encoding inconsistencies, trimming excessive lengths, and extracting metadata like author and source.

11. LLM-Enhanced Sentiment Classification

An LLM-based classifier determines news sentiment while adjusting scores for financial meaning. It goes beyond traditional polarity analysis to interpret investor mood, merging **AI linguistic reasoning** with financial rule-based context.

12. Financial Context Intelligence

The intelligence layer links language patterns to economic events—such as earnings reports or product launches—and infers potential effects on stock movements. This contextual analysis connects semantic interpretation to financial impact.

13. Score Adjustment & Boosting Logic

Rule-based reinforcement adjusts sentiment scores based on detected event importance. For example, earnings “beats” are boosted positively, while “misses” or “downgrades” reduce sentiment strength to reflect realistic market reactions.

14. Robust Fallback Sentiment Analysis

A keyword-based sentiment mechanism activates when the LLM model or API fails. It performs lexicon scoring and context-aware sentiment classification to ensure system continuity and consistent results.

15. Intelligent Content Routing System

Implements **automatic routing** of content to specialized analyzers—corporate, macroeconomic, or market-wide. This guarantees that each article receives expert-level processing based on its linguistic and topical profile.

16. Dynamic Summary Generation

Integrates **AI-driven summarization** that compresses long news articles into concise contextual insights. It adjusts the summary length based on article size and source credibility, ensuring optimized reading time and precision.This dynamic, AI-driven summarization allows the NewsAgent to provide readers concise yet meaningful news digests tailored to article length and complexity.

17. Multi-Source Insight Aggregation

Aggregates insights from multiple analyzers into a coherent intelligence package. This multi-perspective synthesis ensures the final summary covers all key developments without redundancy.

18. Automated Quality Evaluation

Evaluates generated summaries using metrics like **information retention ratio**, **compression efficiency**, and **semantic faithfulness**. These metrics gauge whether summaries convey essential information accurately and concisely.

19. Continuous Optimization System

Implements a self-correcting framework that monitors accuracy, latency, and sentiment consistency. When performance degrades, it triggers retraining or parameter updates to restore desired system behavior.

20. Comprehensive Fallback System

A multi-level fallback ensures user-facing reliability even under multiple failure scenarios. If both API retrieval and LLM analysis fail, the system returns structured neutral outputs with a reliability indicator for graceful degradation.

```
# ================================================================================
# SECTION 6: NEWS AGENT WITH WORKFLOW PATTERNS
# ================================================================================
class NewsAgent(ResearchAgent):
    def __init__(self):
        super().__init__("NewsAgent")

        # Initialize specialized analyzers for ROUTING pattern
        self.earnings_analyzer = EarningsAnalyzer()
        self.market_analyzer = MarketAnalyzer()
        self.company_analyzer = CompanyNewsAnalyzer()

        # Initialize LLM components for PROMPT CHAINING
        try:
            # Sentiment classification model
            self.sentiment_classifier = pipeline(
                "sentiment-analysis",
                model="cardiffnlp/twitter-roberta-base-sentiment-latest",
                tokenizer="cardiffnlp/twitter-roberta-base-sentiment-latest"
            )

            # Text summarization model
            self.summarizer = pipeline(
                "summarization",
```

```
            model="facebook/bart-large-cnn",
            tokenizer="facebook/bart-large-cnn"
        )

        self.log("LLM models initialized successfully")
    except Exception as e:
        self.log(f"Failed to initialize LLM models: {str(e)}", "ERROR")
        self.sentiment_classifier = None
        self.summarizer = None

    # NewsAPI configuration
    self.api_key = "14b44d9019cd4010ae1eead451ee8e2b"
    try:
        self.client = NewsApiClient(api_key=self.api_key)
        self.log("NewsAPI client initialized successfully")
    except Exception as e:
        self.log(f"Failed to initialize NewsAPI client: {str(e)}", "ERROR")
        self.client = None

    # Enhanced positive words with financial context
    self.positive_words = {
        'up', 'gain', 'rise', 'high', 'positive', 'strong', 'beat',
        'increase', 'growth', 'profit', 'bullish', 'outperform',
        # Product and innovation terms
        'new', 'launch', 'release', 'innovation', 'breakthrough',
        'upgrade', 'enhanced', 'improved', 'advanced', 'premium',
        # Financial performance terms
        'record', 'surge', 'rally', 'soar', 'jump', 'climb',
        'demand', 'popular', 'bestseller', 'sold out', 'waitlist',
        # Strategic terms
        'expansion', 'partnership', 'acquisition', 'investment',
        'leadership', 'dominant', 'win', 'success'
    }

    self.negative_words = {
        'down', 'fall', 'drop', 'low', 'negative', 'weak', 'miss',
        'decrease', 'loss', 'sell', 'downgrade', 'crash',
        # Product issues
```

```
            'recall', 'defect', 'faulty', 'delay', 'cancelled', 'recall',
            'lawsuit', 'investigation', 'regulatory', 'fine',
            # Financial troubles
            'bankruptcy', 'default', 'layoff', 'cut', 'reduction',
            'warning', 'concern', 'risk', 'volatile', 'uncertainty'
        }

        # Product launch context boosters
        self.product_launch_indicators = {
            'iphone', 'ipad', 'macbook', 'airpods', 'apple watch',
            'product launch', 'new model', 'next generation', 'unveil',
            'announce', 'introduce', 'release', 'launch event'
        }

        # Financial impact indicators
        self.financial_boosters = {
            'sales', 'revenue', 'earnings', 'profit', 'demand', 'orders',
            'pre-orders', 'shipments', 'market share', 'growth'
        }

        # Evaluation metrics for EVALUATOR-OPTIMIZER pattern
        self.quality_metrics = {
            'sentiment_confidence': [],
            'summary_quality': [],
            'routing_accuracy': []
        }

    def fetch_news(self, symbol: str) -> List[Dict[str, Any]]:
        """Main news processing pipeline implementing all three workflow patterns"""
        try:
            # STEP 1: INGEST - Get raw news data
            raw_news = self._ingest_news(symbol)

            processed_news = []
            for news_item in raw_news:
                # PROMPT CHAINING: Ingest → Preprocess → Classify → Extract → Summarize
                processed_item = self._process_news_pipeline(news_item, symbol)
                processed_news.append(processed_item)
```

```
            # EVALUATOR-OPTIMIZER: Evaluate quality and refine
            self._evaluate_and_optimize(processed_news)

            return processed_news

        except Exception as e:
            self.log(f"Error in news processing pipeline for {symbol}: {str(e)}", "ERROR")
            return self._get_fallback_news(symbol)

    def _ingest_news(self, symbol: str) -> List[Dict]:
        """Step 1: Ingest raw news from NewsAPI"""
        try:
            if self.client is None:
                self.log("NewsAPI client not available, using fallback", "WARNING")
                return []

            response = self.client.get_everything(
                q=symbol,
                language='en',
                sort_by='relevancy',
                page_size=10
            )
            return response.get('articles', [])
        except Exception as e:
            self.log(f"NewsAPI ingestion failed: {str(e)}", "ERROR")
            return []

    def _process_news_pipeline(self, news_item: Dict, symbol: str) -> Dict[str, Any]:
        """PROMPT CHAINING: Complete processing pipeline for each news item"""

        # Step 2: PREPROCESS - Clean and prepare text
        preprocessed_data = self._preprocess_news(news_item, symbol)

        # Step 3: CLASSIFY - Use LLM for sentiment classification
        classification_data = self._classify_with_llm(preprocessed_data)

        # Step 4: ROUTING - Direct to specialized analyzers
```

```
        routing_data = self._route_to_specialists(preprocessed_data, symbol)

        # Step 5: EXTRACT & SUMMARIZE - Generate insights and summary
        summary_data = self._extract_and_summarize(preprocessed_data, routing_data)

        # Combine all processed data
        return {
            **preprocessed_data,
            **classification_data,
            **routing_data,
            **summary_data,
            'processing_timestamp': datetime.now().isoformat(),
            'pipeline_version': '1.0'
        }

    def _preprocess_news(self, news_item: Dict, symbol: str) -> Dict[str, Any]:
        """Step 2: Preprocess news content"""
        title = news_item.get('title', f"Latest news about {symbol}")
        content = news_item.get('content', '') or news_item.get('description', '')
        publisher = news_item.get('source', {}).get('name', 'Unknown Source')

        # Clean and truncate
        title = self._clean_text(title)
        content = self._clean_text(content)

        # Extract metadata
        published = news_item.get('publishedAt')
        if published:
            try:
                published_dt = datetime.strptime(published, '%Y-%m-%dT%H:%M:%SZ')
            except:
                published_dt = datetime.now()
        else:
            published_dt = datetime.now()

        return {
            'title': title[:100] + '...' if len(title) > 100 else title,
            'content': content[:500] + '...' if len(content) > 500 else content,
```

```
            'publisher': publisher,
            'published': published_dt.strftime('%Y-%m-%d %H:%M'),
            'link': news_item.get('url', ''),
            'symbol': symbol
        }

    def _classify_with_llm(self, preprocessed_data: Dict) -> Dict[str, Any]:
        """Step 3: Classify sentiment using LLM"""
        title = preprocessed_data['title']
        content = preprocessed_data['content']

        try:
            if self.sentiment_classifier:
                # Add financial context to improve classification
                enhanced_text = self._add_financial_context(title, content)

                # Classify with enhanced context
                result = self.sentiment_classifier(enhanced_text[:512])[0]

                # Apply financial context adjustments
                adjusted_result = self._apply_financial_context_adjustments(result, title)

                return {
                    'sentiment': adjusted_result['label'],
                    'sentiment_score': adjusted_result['score'],
                    'sentiment_confidence': adjusted_result['confidence'],
                    'llm_analysis_used': True,
                    'financial_context_applied': True
                }

        except Exception as e:
            self.log(f"LLM classification failed: {str(e)}", "WARNING")

        # Fallback to enhanced rule-based analysis
        return self._fallback_sentiment_analysis(title)

    def _add_financial_context(self, title: str, content: str) -> str:
        """Add financial context to help LLM understand investment implications"""
```

```
        # Check for product launch patterns
        text_lower = (title + " " + content).lower()

        product_terms = []
        for product in ['iphone', 'ipad', 'mac', 'airpods', 'watch']:
            if product in text_lower:
                product_terms.append(product)

        # Create enhanced context
        enhanced_parts = [title]

        if product_terms and any(word in text_lower for word in ['new', 'launch', 'release']):
            enhanced_parts.append("This is a product launch announcement which typically has positive implications for company

        if any(word in text_lower for word in ['sales', 'demand', 'revenue', 'earnings']):
            enhanced_parts.append("This news relates to financial performance metrics important for investment decisions.")

        return " ".join(enhanced_parts)

    def _apply_financial_context_adjustments(self, result: Dict, title: str) -> Dict:
        """Adjust LLM results based on financial context"""
        label = result['label']
        score = result['score']
        title_lower = title.lower()

        # Boost scores for product launches and financial news
        if any(indicator in title_lower for indicator in self.product_launch_indicators):
            if label == 'positive':
                score = min(1.0, score * 1.2)  # Boost positive product news
            elif label == 'neutral':
                # Convert neutral product launches to positive
                label = 'positive'
                score = 0.7  # Reasonable confidence for product launches

        return {
            'label': label,
            'score': score,
            'confidence': result['score']
```

```
        }

    def _fallback_sentiment_analysis(self, text: str) -> Dict[str, Any]:
        """Enhanced fallback sentiment analysis with financial context"""
        text_lower = text.lower()

        # Base sentiment calculation
        pos_count = sum(1 for word in self.positive_words if word in text_lower)
        neg_count = sum(1 for word in self.negative_words if word in text_lower)

        # Contextual boosts for product launches
        context_boost = 0.0
        confidence_boost = 0.0

        # Boost for product launches (like new iPhone)
        if any(indicator in text_lower for indicator in self.product_launch_indicators):
            context_boost += 0.3
            confidence_boost += 0.2

            # Additional boost if combined with financial terms
            if any(booster in text_lower for booster in self.financial_boosters):
                context_boost += 0.2
                confidence_boost += 0.1

        # Calculate base sentiment
        total_words = pos_count + neg_count
        if total_words > 0:
            base_sentiment_score = (pos_count - neg_count) / total_words
        else:
            base_sentiment_score = 0.0

        # Apply context boost
        final_sentiment_score = base_sentiment_score + context_boost
        final_sentiment_score = max(-1.0, min(1.0, final_sentiment_score))

        # Calculate confidence
        base_confidence = min(1.0, total_words / 10) if total_words > 0 else 0.3
        final_confidence = min(1.0, base_confidence + confidence_boost)
```

```
        # Determine sentiment category
        if final_sentiment_score > 0.2:
            sentiment = 'Positive'
        elif final_sentiment_score < -0.2:
            sentiment = 'Negative'
        else:
            sentiment = 'Neutral'

        # Convert to 0-1 scale for consistency
        normalized_score = (final_sentiment_score + 1) / 2

        return {
            'sentiment': sentiment,
            'sentiment_score': normalized_score,
            'sentiment_confidence': final_confidence,
            'llm_analysis_used': False,
            'context_boost_applied': context_boost > 0
        }

    def _route_to_specialists(self, preprocessed_data: Dict, symbol: str) -> Dict[str, Any]:
        """ROUTING PATTERN: Direct content to specialized analyzers"""
        routing_results = {}

        # Route to Earnings Analyzer
        earnings_analysis = self.earnings_analyzer.analyze_earnings(preprocessed_data)
        if earnings_analysis['is_earnings']:
            routing_results['earnings_analysis'] = earnings_analysis

        # Route to Market Analyzer
        market_analysis = self.market_analyzer.analyze_market(preprocessed_data)
        if market_analysis['is_market']:
            routing_results['market_analysis'] = market_analysis

        # Route to Company News Analyzer
        company_analysis = self.company_analyzer.analyze_company_news(preprocessed_data, symbol)
        if company_analysis['is_company_news']:
            routing_results['company_analysis'] = company_analysis
```

```
        # Determine primary category
        primary_category = self._determine_primary_category(routing_results)

        return {
            'routing_results': routing_results,
            'primary_category': primary_category,
            'specialist_analyses': len(routing_results)
        }

    def _determine_primary_category(self, routing_results: Dict) -> str:
        """Determine the most relevant category for the news"""
        if not routing_results:
            return 'general'

        # Priority order: earnings > company_news > market
        if 'earnings_analysis' in routing_results:
            return 'earnings'
        elif 'company_analysis' in routing_results:
            return 'company_news'
        elif 'market_analysis' in routing_results:
            return 'market'
        else:
            return 'general'

    def _extract_and_summarize(self, preprocessed_data: Dict, routing_data: Dict) -> Dict[str, Any]:
        """Step 5: Extract key insights and generate summary"""
        title = preprocessed_data['title']
        content = preprocessed_data['content']

        try:
            if self.summarizer and content and len(content) > 50:
                # Calculate appropriate max_length based on content length
                content_length = len(content.split())
                max_length = min(100, max(30, content_length // 2))  # Dynamic max_length
                min_length = min(20, max_length // 2)

                # Generate AI summary with appropriate length
```

```
                summary = self.summarizer(
                    content,
                    max_length=max_length,
                    min_length=min_length,
                    do_sample=False
                )[0]['summary_text']

                summary_quality = self._evaluate_summary_quality(summary, content)
            else:
                summary = title  # Fallback to title as summary
                summary_quality = 0.5

        except Exception as e:
            self.log(f"LLM summarization failed: {str(e)}", "WARNING")
            summary = title
            summary_quality = 0.3

        # Extract key insights based on routing results
        insights = self._extract_key_insights(preprocessed_data, routing_data)

        return {
            'summary': summary,
            'summary_quality': summary_quality,
            'key_insights': insights,
            'insight_count': len(insights)
        }

    def _extract_key_insights(self, preprocessed_data: Dict, routing_data: Dict) -> List[str]:
        """Extract key insights from specialist analyses"""
        insights = []
        routing_results = routing_data.get('routing_results', {})

        # Add insights from earnings analysis
        if 'earnings_analysis' in routing_results:
            earnings_insights = routing_results['earnings_analysis'].get('insights', [])
            insights.extend(earnings_insights)

        # Add insights from market analysis
```

```
        if 'market_analysis' in routing_results:
            market_insights = routing_results['market_analysis'].get('insights', [])
            insights.extend(market_insights)

        # Add sentiment-based insights
        sentiment = preprocessed_data.get('sentiment', 'Neutral')
        if sentiment == 'Positive':
            insights.append("Positive sentiment detected")
        elif sentiment == 'Negative':
            insights.append("Negative sentiment detected")

        # Ensure we have at least one insight
        if not insights:
            insights.append("General market news")

        return insights[:5]  # Limit to top 5 insights

    def _evaluate_summary_quality(self, summary: str, original_content: str) -> float:
        """Evaluate the quality of the generated summary"""
        if not summary or not original_content:
            return 0.0

        # Simple quality metrics
        summary_length = len(summary)
        content_length = len(original_content)

        # Compression ratio (ideal: 10-30%)
        compression_ratio = summary_length / content_length if content_length > 0 else 1.0
        compression_score = 1.0 - abs(compression_ratio - 0.2)  # Peak at 20% compression

        # Content preservation (simple word overlap)
        summary_words = set(summary.lower().split())
        content_words = set(original_content.lower().split()[:100])  # First 100 words
        overlap = len(summary_words.intersection(content_words))
        preservation_score = min(1.0, overlap / 10)  # At least 10 overlapping words

        return (compression_score * 0.4) + (preservation_score * 0.6)
```

```
def _evaluate_and_optimize(self, processed_news: List[Dict]):
    """EVALUATOR-OPTIMIZER PATTERN: Evaluate quality and refine processing"""
    if not processed_news:
        return

    # Collect quality metrics
    for news_item in processed_news:
        self.quality_metrics['sentiment_confidence'].append(
            news_item.get('sentiment_confidence', 0.0)
        )
        self.quality_metrics['summary_quality'].append(
            news_item.get('summary_quality', 0.0)
        )

    # Calculate average metrics
    avg_metrics = {
        metric: np.mean(values) if values else 0.0
        for metric, values in self.quality_metrics.items()
    }

    # Log quality assessment
    self.log(f"Quality Assessment - Sentiment Confidence: {avg_metrics['sentiment_confidence']:.3f}, "
            f"Summary Quality: {avg_metrics['summary_quality']:.3f}")

    # Optimize based on quality (simple threshold-based optimization)
    if avg_metrics['sentiment_confidence'] < 0.6:
        self.log("Low sentiment confidence detected - consider model fine-tuning", "WARNING")

    if avg_metrics['summary_quality'] < 0.5:
        self.log("Low summary quality detected - optimizing summarization parameters", "WARNING")

def _clean_text(self, text: str) -> str:
    """Clean and normalize text"""
    if not text:
        return ""

    # Remove extra whitespace and special characters
    text = re.sub(r'\s+', ' ', text)
```

```
        text = re.sub(r'[^\w\s.,!?-]', '', text)
        return text.strip()

    def _get_fallback_news(self, symbol: str) -> List[Dict[str, Any]]:
        """Generate fallback news when processing fails"""
        return [{
            'title': f"Market updates for {symbol}",
            'content': f"Recent developments and trading activity for {symbol}",
            'publisher': 'Market Analysis',
            'published': datetime.now().strftime('%Y-%m-%d %H:%M'),
            'link': '',
            'symbol': symbol,
            'sentiment': 'Neutral',
            'sentiment_score': 0.5,
            'sentiment_confidence': 0.0,
            'llm_analysis_used': False,
            'primary_category': 'general',
            'summary': f"General market news for {symbol}",
            'summary_quality': 0.3,
            'key_insights': ['General market activity'],
            'insight_count': 1
        }]
```

Section 7: DATA AGENT- Data Acquisition and Feature Extraction : DESIGN DESCRIPTION

This section is responsible for collecting raw financial market data and transforming it into meaningful, actionable metrics. It involves:

a) Fetching historical stock data over a relevant time window from reliable data sources (e.g., Yahoo Finance APIs).

b) Accessing fundamental company information such as current trading price, market capitalization, P/E ratios, analyst price targets, earnings dates, and volume.

c) Computing key technical indicators widely used in financial analysis, including the Relative Strength Index (RSI) and Moving Average Convergence Divergence (MACD). These indicators provide insights into market momentum, overbought/oversold conditions, and price trend directions.

d) The design leverages time series computations on closing prices to yield trend signals that indicate bullish or bearish market behaviors. By integrating both static company fundamentals and dynamic technical signals, this stage prepares a rich dataset for further evaluation.

Section 7: DATA AGENT- Data Acquisition and Feature Extraction : IM{LEMENTATION DETAILS

a. Description: Serves as the specialized agent responsible for the **ingestion** and **preparation** of structured financial and technical data. It reliably fetches quantitative information (company profile, historical stock prices, analyst recommendations) from external APIs, calculates key **technical indicators** (RSI and MACD), and meticulously **serializes** this data into a standardized format for seamless integration with downstream analytical agents.

b. Key Methods/Features:

```
i) `fetch_stock_data(symbol)`:** The primary orchestration method for this agent. It coordinates the retrieval of stock informati

API Interaction (`yf.Ticker`)**: Uses the `yfinance` library (`yf.Ticker`) to retrieve the raw stock data (profile, price history

calculate_rsi(prices, period=14)`**: Calculates the **Relative Strength Index (RSI)** based on the provided price history DataFra

calculate_macd(prices)`**: Calculates the **Moving Average Convergence Divergence (MACD)** indicator and its signal line based or

Data Structuring & Serialization:** Organizes the fetched data and calculated indicators into a comprehensive dictionary. Crucial
```

c. Pipeline Role: Functions as the primary **Data Ingestion Tool User** within the pipeline, operating in parallel with the `NewsAgent`. Its role is expanded to provide not only foundational **quantitative fundamental data** (like P/E ratio, market cap) but also key **technical**

indicators (RSI, MACD) required by the `AnalysisAgent` and potentially the `ReportingAgent` for generating technical analysis insights and recommendations.

```
#============================================================
##    SECTION 7: DATA AGENT                                 ##
#============================================================

class DataAgent(ResearchAgent):
    def __init__(self):
        super().__init__("DataAgent")

    def fetch_stock_data(self, symbol: str) -> Dict[str, Any]:
        try:
            stock = yf.Ticker(symbol)
            info = stock.info
            hist = stock.history(period="3mo")

            # Calculate technical indicators
            rsi, rsi_trends = self.calculate_rsi(hist)
            macd, macd_signal = self.calculate_macd(hist)

            return {
                'symbol': symbol,
                'current_price': info.get('currentPrice', info.get('regularMarketPrice', 0)),
                'pe_ratio': info.get('trailingPE', 0),
                'forward_pe': info.get('forwardPE', 0),
                'market_cap': info.get('marketCap', 0),
                'volume': info.get('volume', 0),
                '52_week_high': info.get('fiftyTwoWeekHigh', 0),
                '52_week_low': info.get('fiftyTwoWeekLow', 0),
                'earnings_date': info.get('earningsDate', 'N/A'),
                'recommendation': info.get('recommendationKey', 'hold'),
                'target_price': info.get('targetMeanPrice', 0),
                'beta': info.get('beta', 1.0),
                'profit_margins': info.get('profitMargins', 0),
                'revenue_growth': info.get('revenueGrowth', 0),
                'rsi': rsi,
                'rsi_trends': rsi_trends,
```

```
                'macd': macd,
                'macd_signal': macd_signal,
                'price_history': hist.to_dict() if not hist.empty else {}
            }
        except Exception as e:
            self.log(f"Error fetching data for {symbol}: {str(e)}", "ERROR")
            return {'error': str(e)}

    def calculate_rsi(self, prices: pd.DataFrame, period: int = 14) -> Tuple[float, Dict]:
        """Calculate RSI indicator"""
        if prices.empty or len(prices) < period:
            return 50, {}

        close_prices = prices['Close']
        delta = close_prices.diff()
        gain = (delta.where(delta > 0, 0)).rolling(window=period).mean()
        loss = (-delta.where(delta < 0, 0)).rolling(window=period).mean()
        rs = gain / loss
        rsi = 100 - (100 / (1 + rs))

        trends = {
            'current': rsi.iloc[-1] if not rsi.empty else 50,
            'trend': 'Bullish' if rsi.iloc[-1] > 50 else 'Bearish',
            'oversold': rsi.iloc[-1] < 30,
            'overbought': rsi.iloc[-1] > 70
        }

        return rsi.iloc[-1] if not rsi.empty else 50, trends

    def calculate_macd(self, prices: pd.DataFrame) -> Tuple[float, float]:
        """Calculate MACD indicator"""
        if prices.empty:
            return 0, 0

        exp1 = prices['Close'].ewm(span=12).mean()
        exp2 = prices['Close'].ewm(span=26).mean()
        macd = exp1 - exp2
        signal = macd.ewm(span=9).mean()
```

```
        return macd.iloc[-1] if not macd.empty else 0, signal.iloc[-1] if not signal.empty else 0
```

Section 8: Analysis Agent - alytical Evaluation and Decision Support

Building upon the features generated in the data acquisition phase, this component systematically evaluates the stock's attractiveness and market sentiment through multiple lenses:

It applies a weighted scoring framework where dimensions such as valuation (P/E ratios), profitability (profit margins and revenue growth), technical indicators (RSI analysis, momentum), sentiment derived from news, analyst recommendations, and overall market position are quantified.

Sentiment analysis is enhanced by aggregating news-derived sentiment scores, weighted by confidence and whether advanced NLP (like LLMs) was used, ensuring more reliable sentiment inputs.

A composite overall score is calculated by combining component scores adjusted through quality feedback mechanisms to reflect data integrity and processing reliability.

The system generates intuitive decision signals (e.g., "BUY", "HOLD", "SELL") with associated confidence levels based on overall scores.

Detailed reasoning accompanies each decision, explaining contributions from valuation, technical conditions, news sentiment, and analyst outlook, thereby improving transparency.

These two sections together form a powerful pipeline: from raw financial and market data ingestion, through technical and fundamental feature extraction, to multi-faceted analytical evaluation yielding actionable investment recommendations.

Such modular, interpretable, and extensible design enables continuous refinement and integration of new data sources or analytical techniques, advancing AI-driven financial decision support.

SECTION 8: Analysis Agent - Analytical Evaluation and Decision Support - Implementation details

a. Description: Synthesizes structured financial data (from `DataAgent`) and processed news analysis (from `NewsAgent`) into a comprehensive, multi-factor investment assessment. It calculates individual scores for valuation, profitability, technicals, sentiment, analyst ratings, and market position, then combines these into a weighted overall score, potentially adjusted based on the quality of the input news analysis, to generate a final investment decision and supporting rationale.

b. Key Methods/Features:

i. **`analyze_stock(stock_data, news_data)`:** The main orchestration method. It receives inputs from upstream agents and coordinates calls to various internal analysis functions to compute component scores and the final decision.

ii. **`analyze_valuation(stock_data)`:** Evaluates the stock's valuation attractiveness based on metrics like P/E ratio and forward P/E, returning a score (0-100).

iii. **`analyze_profitability(stock_data)`:** Assesses the company's financial health using profit margins and revenue growth, returning a score (0-100).

iv. **`analyze_technical(stock_data)`:** Analyzes technical momentum using indicators like RSI (including oversold/overbought conditions and trend), returning a score (0-100).

v. **`analyze_sentiment(news_data)`:** Calculates an enhanced sentiment score based on the processed news. It weights the sentiment score of each article by the confidence level and whether an LLM was used, providing a more reliable aggregate sentiment score (0-100).

vi. **`analyze_analyst(stock_data)`:** Scores the consensus analyst recommendation (e.g., strong buy, hold) and the potential upside based on the target price versus the current price, returning a score (0-100).

vii. **`analyze_market_position(stock_data)`:** Evaluates the company's size (market cap) and relative volatility (beta), returning a score (0-100).

viii. **`_calculate_optimized_score(...)`:** Implements the **EVALUATOR-OPTIMIZER** pattern. It first calculates a base score by applying predefined weights (`self.decision_weights`) to each component score. It then adjusts this base score using a `_calculate_quality_boost` derived from the average quality metrics (sentiment confidence, summary quality, LLM usage) of the input `news_data`, rewarding analyses based on high-quality inputs.

ix. **`generate_decision(overall_score)`:** Translates the final optimized overall score into a categorical investment decision (e.g., "STRONG BUY", "HOLD", "SELL") and assigns a confidence level.

x. **`generate_detailed_reasoning(...)`:** Creates a human-readable explanation for the final decision by highlighting the key contributing factors identified during the component analysis (e.g., "Attractive valuation", "Negative news sentiment").

c. Pipeline Role: Acts as the **central analysis and decision-making hub**. It sits downstream from the `DataAgent` and `NewsAgent`, receiving their processed outputs. Its primary role is to **synthesize** diverse quantitative and qualitative inputs, apply a weighted scoring model (refined by input quality evaluation), and produce the final, actionable output: an investment recommendation with supporting evidence and reasoning, ready for the `ReportingAgent`.

```
# ================================================================================
# SECTION 8: ANALYSIS AGENT
# ================================================================================

class AnalysisAgent(ResearchAgent):
    def __init__(self):
        super().__init__("AnalysisAgent")
        self.decision_weights = {
            'valuation': 0.25,
            'profitability': 0.20,
            'technical': 0.20,
            'sentiment': 0.20,  # Enhanced weight for LLM sentiment
            'analyst': 0.10,
            'market_position': 0.05
        }

    def analyze_stock(self, stock_data: Dict, news_data: List) -> Dict[str, Any]:
        try:
            # Calculate component scores
            valuation_score = self.analyze_valuation(stock_data)
            profitability_score = self.analyze_profitability(stock_data)
            technical_score = self.analyze_technical(stock_data)
            sentiment_score = self.analyze_sentiment(news_data)
            analyst_score = self.analyze_analyst(stock_data)
            market_score = self.analyze_market_position(stock_data)

            # Calculate weighted overall score with EVALUATOR-OPTIMIZER feedback
            overall_score = self._calculate_optimized_score(
```

```
                valuation_score, profitability_score, technical_score,
                sentiment_score, analyst_score, market_score, news_data
            )

            # Generate decision
            decision, confidence = self.generate_decision(overall_score)

            # Component breakdown for visualization
            component_breakdown = {
                'valuation': valuation_score,
                'profitability': profitability_score,
                'technical': technical_score,
                'sentiment': sentiment_score,
                'analyst': analyst_score,
                'market_position': market_score
            }

            return {
                'decision': decision,
                'confidence': confidence,
                'overall_score': overall_score,
                'component_breakdown': component_breakdown,
                'pe_ratio': stock_data.get('pe_ratio', 0),
                'current_price': stock_data.get('current_price', 0),
                'target_price': stock_data.get('target_price', 0),
                'news_sentiment': self.get_overall_sentiment(news_data),
                'analyst_recommendation': stock_data.get('recommendation', 'hold'),
                'rsi': stock_data.get('rsi', 50),
                'reasoning': self.generate_detailed_reasoning(
                    decision, stock_data, news_data, component_breakdown
                )
            }
        except Exception as e:
            self.log(f"Error analyzing stock: {str(e)}", "ERROR")
            return {'error': str(e)}

    def analyze_valuation(self, stock_data: Dict) -> float:
        """Analyze valuation metrics (0-100 scale)"""
```

```
        pe_ratio = stock_data.get('pe_ratio', 0)
        forward_pe = stock_data.get('forward_pe', 0)

        score = 50  # Neutral

        if 0 < pe_ratio < 15:
            score += 25  # Very attractive
        elif 15 <= pe_ratio < 25:
            score += 10  # Reasonable
        elif pe_ratio >= 40:
            score -= 20  # Expensive

        # Consider forward P/E if available
        if forward_pe > 0 and forward_pe < pe_ratio:
            score += 5  # Improving valuation

        return max(0, min(100, score))

    def analyze_profitability(self, stock_data: Dict) -> float:
        """Analyze profitability metrics"""
        profit_margins = stock_data.get('profit_margins', 0)
        revenue_growth = stock_data.get('revenue_growth', 0)

        score = 50

        if profit_margins > 0.2:
            score += 25
        elif profit_margins > 0.1:
            score += 10
        elif profit_margins <= 0:
            score -= 20

        if revenue_growth > 0.1:
            score += 15
        elif revenue_growth < 0:
            score -= 10

        return max(0, min(100, score))
```

```
def analyze_technical(self, stock_data: Dict) -> float:
    """Analyze technical indicators"""
    rsi = stock_data.get('rsi', 50)
    rsi_trends = stock_data.get('rsi_trends', {})

    score = 50

    # RSI analysis
    if 30 <= rsi <= 70:
        score += 10  # Neutral zone
    elif rsi < 30:
        score += 20  # Oversold - potential buying opportunity
    elif rsi > 70:
        score -= 10  # Overbought - caution

    # RSI trend
    if rsi_trends.get('trend') == 'Bullish':
        score += 5

    return max(0, min(100, score))

def analyze_sentiment(self, news_data: List) -> float:
    """Enhanced sentiment analysis using LLM results with confidence weighting"""
    if not news_data:
        return 50

    weighted_scores = []
    for news in news_data:
        sentiment_score = news.get('sentiment_score', 0.5)
        confidence = news.get('sentiment_confidence', 0.0)
        llm_used = news.get('llm_analysis_used', False)

        # Apply confidence weighting
        if llm_used and confidence > 0.7:
            # High confidence LLM analysis gets full weight
            weight = 1.0
        elif llm_used:
```

```
            # Medium confidence LLM analysis
            weight = 0.7
        else:
            # Rule-based analysis gets lower weight
            weight = 0.5

        # Convert to 0-100 scale and apply weight
        score = 50 + (sentiment_score - 0.5) * 100
        weighted_score = 50 + (score - 50) * weight
        weighted_scores.append(weighted_score)

    avg_sentiment = sum(weighted_scores) / len(weighted_scores) if weighted_scores else 50
    return max(0, min(100, avg_sentiment))

def _calculate_optimized_score(self, valuation_score: float, profitability_score: float,
                               technical_score: float, sentiment_score: float,
                               analyst_score: float, market_score: float,
                               news_data: List) -> float:
    """EVALUATOR-OPTIMIZER: Calculate optimized score with quality feedback"""
    base_score = (
        valuation_score * self.decision_weights['valuation'] +
        profitability_score * self.decision_weights['profitability'] +
        technical_score * self.decision_weights['technical'] +
        sentiment_score * self.decision_weights['sentiment'] +
        analyst_score * self.decision_weights['analyst'] +
        market_score * self.decision_weights['market_position']
    )

    # Apply optimizations based on news quality
    quality_boost = self._calculate_quality_boost(news_data)
    optimized_score = base_score * (1 + quality_boost)

    return max(0, min(100, optimized_score))

def _calculate_quality_boost(self, news_data: List) -> float:
    """Calculate quality boost based on news analysis quality"""
    if not news_data:
        return 0.0
```

```
        # Average quality metrics
        avg_sentiment_confidence = np.mean([news.get('sentiment_confidence', 0.0) for news in news_data])
        avg_summary_quality = np.mean([news.get('summary_quality', 0.0) for news in news_data])
        llm_usage_ratio = np.mean([1.0 if news.get('llm_analysis_used', False) else 0.0 for news in news_data])

        # Quality score (0-1)
        quality_score = (avg_sentiment_confidence * 0.4 +
                         avg_summary_quality * 0.3 +
                         llm_usage_ratio * 0.3)

        # Convert to boost (-0.1 to +0.1)
        return (quality_score - 0.5) * 0.2

    def analyze_analyst(self, stock_data: Dict) -> float:
        """Analyze analyst recommendations"""
        recommendation = stock_data.get('recommendation', 'hold')
        target_price = stock_data.get('target_price', 0)
        current_price = stock_data.get('current_price', 0)

        score = 50

        # Analyst recommendation scoring
        if recommendation == 'strong_buy':
            score += 25
        elif recommendation == 'buy':
            score += 15
        elif recommendation == 'sell':
            score -= 15
        elif recommendation == 'strong_sell':
            score -= 25

        # Price target analysis
        if target_price > 0 and current_price > 0:
            upside = ((target_price - current_price) / current_price) * 100
            if upside > 20:
                score += 10
            elif upside < -10:
```

```
                score -= 10

        return max(0, min(100, score))

    def analyze_market_position(self, stock_data: Dict) -> float:
        """Analyze market position and size"""
        market_cap = stock_data.get('market_cap', 0)
        beta = stock_data.get('beta', 1.0)

        score = 50

        # Market cap scoring (larger companies generally more stable)
        if market_cap > 200e9:  # Large cap
            score += 10
        elif market_cap < 10e9:  # Small cap
            score -= 5

        # Volatility scoring
        if beta < 0.8:  # Low volatility
            score += 5
        elif beta > 1.5:  # High volatility
            score -= 5

        return max(0, min(100, score))

    def generate_decision(self, overall_score: float) -> Tuple[str, float]:
        """Generate investment decision based on overall score"""
        if overall_score >= 80:
            return "STRONG BUY", min(0.95, overall_score/100)
        elif overall_score >= 70:
            return "BUY", min(0.85, overall_score/100)
        elif overall_score >= 60:
            return "HOLD", min(0.75, overall_score/100)
        elif overall_score >= 50:
            return "WEAK HOLD", min(0.65, overall_score/100)
        elif overall_score >= 40:
            return "SELL", min(0.55, overall_score/100)
        else:
```

```
            return "STRONG SELL", min(0.45, overall_score/100)

    def get_overall_sentiment(self, news_data: List) -> str:
        """Get overall sentiment from news data"""
        if not news_data:
            return "Neutral"

        sentiments = [news.get('sentiment', 'Neutral') for news in news_data]
        positive_count = sentiments.count("Positive")
        negative_count = sentiments.count("Negative")

        if positive_count > negative_count:
            return "Positive"
        elif negative_count > positive_count:
            return "Negative"
        else:
            return "Neutral"

    def generate_detailed_reasoning(self, decision: str, stock_data: Dict,
                                    news_data: List, component_breakdown: Dict) -> str:
        """Generate detailed reasoning for the decision"""
        reasons = []

        # Valuation reasoning
        pe_ratio = stock_data.get('pe_ratio', 0)
        if pe_ratio < 15:
            reasons.append("Attractive valuation with low P/E ratio")
        elif pe_ratio > 35:
            reasons.append("High valuation with elevated P/E ratio")

        # Profitability reasoning
        profit_margins = stock_data.get('profit_margins', 0)
        if profit_margins > 0.2:
            reasons.append("Strong profit margins")
        elif profit_margins < 0:
            reasons.append("Negative profit margins")

        # Technical reasoning
```

```
rsi = stock_data.get('rsi', 50)
if rsi < 30:
    reasons.append("Oversold technical condition")
elif rsi > 70:
    reasons.append("Overbought technical condition")

# News sentiment reasoning
overall_sentiment = self.get_overall_sentiment(news_data)
if overall_sentiment == "Positive":
    reasons.append("Positive news sentiment")
elif overall_sentiment == "Negative":
    reasons.append("Negative news sentiment")

# Analyst reasoning
target_price = stock_data.get('target_price', 0)
current_price = stock_data.get('current_price', 0)
if target_price > current_price:
    upside = ((target_price - current_price) / current_price) * 100
    reasons.append(f"Analyst target suggests {upside:.1f}% upside")

if not reasons:
    reasons.append("Balanced fundamentals across metrics")

return "; ".join(reasons)
```

SECTION 9: Visualization Engine (Generative Visualization Agent)

a. Description:

Operates as a non-analytical but highly interactive visual agent transforming outputs from Data, News, and Analysis Agents into intuitive dashbo ards. It helps users trace how sentiments, technical metrics, and recommendations interact across multiple equities.

b. Key Methods / Features:

i. Sentiment Visualization (create_sentiment_analysis_chart)

```
Generates pie/bar comparisons of news sentiment distribution for each symbol.
```

ii. Decision Breakdown (create_decision_breakdown_chart):

```
Depicts weighted component scores on bar and radar charts for each agent score category.
```

iii. Recommendation Summary (create_recommendation_summary):

```
Aggregates buy/hold/sell signals across multiple stocks into pie charts, confidence bars, and heatmaps.
```

iv. Technical and News Category Charts:

```
Render time-series price and RSI visuals plus news category distributions.
```

v. Matplotlib & Seaborn Integration:

```
Implements aesthetic color maps and annotations for clarity.
```

c. Pipeline Role:

Acts as the Presentation Layer Agent, integrating quantitative analytics and qualitative insights into visual formats. It bridges AI outputs to huma n interpretability within the FinancialAnalysis

```
    # ================================================================================
    # SECTION 9: VISUALIZATION ENGINE
```

```
# ============================================================================

class VisualizationEngine:
    """Creates comprehensive visualizations for financial analysis"""

    def __init__(self):
        self.figures = {}

    def create_sentiment_analysis_chart(self, news_data: List, symbol: str):
        """Create sentiment analysis visualization"""
        if not news_data:
            print(f"No news data available for {symbol}")
            return

        sentiments = [news.get('sentiment', 'Neutral') for news in news_data]
        sentiment_counts = pd.Series(sentiments).value_counts()

        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

        # Pie chart
        colors = ['#2ecc71', '#f39c12', '#e74c3c']  # Green, Orange, Red
        ax1.pie(sentiment_counts.values, labels=sentiment_counts.index,
                autopct='%1.1f%%', colors=colors[:len(sentiment_counts)])
        ax1.set_title(f'News Sentiment Distribution - {symbol}')

        # Bar chart with sentiment scores
        sentiment_scores = [news.get('sentiment_score', 0.5) for news in news_data]
        news_titles = [news.get('title', 'No title')[:40] + '...' for news in news_data]

        colors = ['green' if s > 0.6 else 'red' if s < 0.4 else 'orange' for s in sentiment_scores]
        bars = ax2.barh(news_titles, sentiment_scores, color=colors, alpha=0.7)
        ax2.set_xlabel('Sentiment Score')
        ax2.set_title('Individual News Sentiment Scores')
        ax2.set_xlim(0, 1)

        # Add value labels
        for bar, score in zip(bars, sentiment_scores):
            ax2.text(bar.get_width() + 0.01, bar.get_y() + bar.get_height()/2,
```

```
                f'{score:.2f}', va='center')

    plt.tight_layout()
    plt.show()

def create_decision_breakdown_chart(self, analysis_result: Dict, symbol: str):
    """Create decision component breakdown visualization"""
    component_breakdown = analysis_result.get('component_breakdown', {})

    if not component_breakdown:
        return

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

    # Component scores bar chart
    components = list(component_breakdown.keys())
    scores = list(component_breakdown.values())

    colors = plt.cm.viridis(np.linspace(0, 1, len(components)))
    bars = ax1.bar(components, scores, color=colors, alpha=0.7)
    ax1.set_ylabel('Score (0-100)')
    ax1.set_title(f'Decision Component Breakdown - {symbol}')
    ax1.set_ylim(0, 100)
    ax1.tick_params(axis='x', rotation=45)

    # Add value labels on bars
    for bar, score in zip(bars, scores):
        ax1.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 1,
                f'{score:.1f}', ha='center', va='bottom')

    # Radar chart for component comparison
    categories = components
    values = scores

    # Complete the circle
    values += values[:1]
    angles = np.linspace(0, 2*np.pi, len(categories), endpoint=False).tolist()
    angles += angles[:1]
```

```
        ax2.plot(angles, values, 'o-', linewidth=2, label='Scores')
        ax2.fill(angles, values, alpha=0.25)
        ax2.set_yticklabels([])
        ax2.set_xticks(angles[:-1])
        ax2.set_xticklabels(categories)
        ax2.set_title(f'Component Radar Chart - {symbol}')

        plt.tight_layout()
        plt.show()

    def create_recommendation_summary(self, results: Dict, symbols: List[str]):
        """Create recommendation summary visualization"""
        decisions = []
        confidences = []
        overall_scores = []

        for symbol in symbols:
            if symbol in results and 'analysis_result' in results[symbol]:
                analysis = results[symbol]['analysis_result']
                decisions.append(analysis.get('decision', 'HOLD'))
                confidences.append(analysis.get('confidence', 0) * 100)
                overall_scores.append(analysis.get('overall_score', 50))

        fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16, 12))

        # Decision distribution
        decision_counts = pd.Series(decisions).value_counts()
        colors = ['#2ecc71', '#27ae60', '#f39c12', '#e74c3c', '#c0392b']
        ax1.pie(decision_counts.values, labels=decision_counts.index,
                autopct='%1.1f%%', colors=colors[:len(decision_counts)])
        ax1.set_title('Overall Recommendation Distribution')

        # Confidence levels
        ax2.bar(symbols, confidences, color='skyblue', alpha=0.7)
        ax2.set_ylabel('Confidence (%)')
        ax2.set_title('Recommendation Confidence by Stock')
        ax2.tick_params(axis='x', rotation=45)
```

```
        ax2.set_ylim(0, 100)

        # Overall scores
        ax3.bar(symbols, overall_scores, color='lightgreen', alpha=0.7)
        ax3.set_ylabel('Overall Score (0-100)')
        ax3.set_title('Overall Analysis Score by Stock')
        ax3.tick_params(axis='x', rotation=45)
        ax3.set_ylim(0, 100)

        # Decision strength heatmap
        decision_strength = {
            'STRONG BUY': 5, 'BUY': 4, 'HOLD': 3, 'SELL': 2, 'STRONG SELL': 1
        }
        strengths = [decision_strength.get(dec, 3) for dec in decisions]

        im = ax4.imshow([strengths], cmap='RdYlGn', aspect='auto')
        ax4.set_xticks(range(len(symbols)))
        ax4.set_xticklabels(symbols, rotation=45)
        ax4.set_title('Recommendation Strength Heatmap')
        plt.colorbar(im, ax=ax4, label='Strength (1-5)')

        plt.tight_layout()
        plt.show()

    def create_technical_analysis_chart(self, stock_data: Dict, symbol: str):
        """Create technical analysis visualization"""
        if 'price_history' not in stock_data or not stock_data['price_history']:
            return

        # Convert price history back to DataFrame for plotting
        price_data = pd.DataFrame(stock_data['price_history'])

        fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

        # Price chart
        if 'Close' in price_data.columns:
            ax1.plot(price_data.index, price_data['Close'], label='Close Price', linewidth=2)
            ax1.set_ylabel('Price ($)')
```

```
            ax1.set_title(f'Price History - {symbol}')
            ax1.legend()
            ax1.grid(True, alpha=0.3)

        # RSI if available
        rsi = stock_data.get('rsi', 50)
        ax2.axhline(y=70, color='r', linestyle='--', alpha=0.7, label='Overbought (70)')
        ax2.axhline(y=30, color='g', linestyle='--', alpha=0.7, label='Oversold (30)')
        ax2.axhline(y=50, color='gray', linestyle='-', alpha=0.5)
        ax2.bar([0], [rsi], color='orange', alpha=0.7, width=0.4)
        ax2.set_ylabel('RSI')
        ax2.set_title(f'Relative Strength Index - {symbol}')
        ax2.set_ylim(0, 100)
        ax2.set_xticks([])
        ax2.legend()

        plt.tight_layout()
        plt.show()

    def create_news_category_chart(self, news_data: List, symbol: str):
        """Create news category distribution chart"""
        if not news_data:
            return

        categories = [news.get('primary_category', 'general') for news in news_data]
        category_counts = pd.Series(categories).value_counts()

        plt.figure(figsize=(10, 6))
        colors = plt.cm.Set3(np.linspace(0, 1, len(category_counts)))
        bars = plt.bar(category_counts.index, category_counts.values, color=colors, alpha=0.7)

        plt.title(f'News Category Distribution - {symbol}')
        plt.xlabel('News Categories')
        plt.ylabel('Number of Articles')
        plt.xticks(rotation=45)

        # Add value labels on bars
        for bar, count in zip(bars, category_counts.values):
```

```
            plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.1,
                    f'{count}', ha='center', va='bottom')

        plt.tight_layout()
        plt.show()
```

SECTION 10: FINANCIAL ANALYSIS SYSTEM

```
# ================================================================================
# SECTION 10: FINANCIAL ANALYSIS SYSTEM
# ================================================================================

class FinancialAnalysisSystem:
    def __init__(self):
        self.data_agent = DataAgent()
        self.news_agent = NewsAgent()
        self.analysis_agent = AnalysisAgent()
        self.viz_engine = VisualizationEngine()
        print("🚀 Multi-Agent Financial Analysis System Initialized")
        print("=" * 60)

    def analyze_stocks(self, symbols: List[str]) -> Dict[str, Any]:
        results = {}
        for symbol in symbols:
            print(f"🔍 Analyzing {symbol}...")
            try:
                stock_data = self.data_agent.fetch_stock_data(symbol)
                news_data = self.news_agent.fetch_news(symbol)
                analysis_result = self.analysis_agent.analyze_stock(stock_data, news_data)
                results[symbol] = {
                    'stock_data': stock_data,
                    'news_data': news_data,
                    'analysis_result': analysis_result
                }
                print(f"✅ {symbol} analysis completed")
```

```
            # Create visualizations for each stock
            self.create_stock_visualizations(symbol, stock_data, news_data, analysis_result)

        except Exception as e:
            print(f"❌ Error analyzing {symbol}: {str(e)}")
            results[symbol] = {'error': str(e)}

    # Create summary visualizations
    self.create_summary_visualizations(results, symbols)

    return results

def create_stock_visualizations(self, symbol: str, stock_data: Dict,
                                news_data: List, analysis_result: Dict):
    """Create individual stock visualizations"""
    print(f"📊 Creating visualizations for {symbol}...")

    # Sentiment analysis chart
    self.viz_engine.create_sentiment_analysis_chart(news_data, symbol)

    # News category chart
    self.viz_engine.create_news_category_chart(news_data, symbol)

    # Decision breakdown chart
    self.viz_engine.create_decision_breakdown_chart(analysis_result, symbol)

    # Technical analysis chart
    self.viz_engine.create_technical_analysis_chart(stock_data, symbol)

def create_summary_visualizations(self, results: Dict, symbols: List[str]):
    """Create summary visualizations across all stocks"""
    print("\n📈 Creating summary visualizations...")
    self.viz_engine.create_recommendation_summary(results, symbols)

def format_market_cap(self, market_cap: float) -> str:
    if market_cap >= 1e12:
        return f"${market_cap/1e12:.1f}T"
```

```
        elif market_cap >= 1e9:
            return f"${market_cap/1e9:.1f}B"
        elif market_cap >= 1e6:
            return f"${market_cap/1e6:.1f}M"
        else:
            return f"${market_cap:.0f}"

    def get_latest_news_title(self, news_data: List) -> str:
        if not news_data:
            return "No news"
        title = news_data[0].get('title', 'No title')
        return title if len(title) <= 40 else title[:37] + "..."

    def display_comprehensive_table(self, symbols: List[str]):
        results = self.analyze_stocks(symbols)
        table_data = []
        for symbol in symbols:
            res = results.get(symbol, {})
            analysis = res.get('analysis_result', {})
            stock_data = res.get('stock_data', {})
            news_data = res.get('news_data', [])

            table_data.append({
                'Symbol': symbol,
                'Current Price': f"${stock_data.get('current_price', 0):.2f}",
                'P/E Ratio': stock_data.get('pe_ratio', 0),
                'Forward P/E': stock_data.get('forward_pe', 0),
                'Market Cap': self.format_market_cap(stock_data.get('market_cap', 0)),
                'RSI': stock_data.get('rsi', 50),
                'Latest News': self.get_latest_news_title(news_data),
                'News Sentiment': analysis.get('news_sentiment', 'Neutral'),
                'Analyst Target': f"${stock_data.get('target_price', 0):.2f}",
                'Analyst Rec': stock_data.get('recommendation', 'hold').title(),
                'AI Decision': analysis.get('decision', 'HOLD'),
                'Confidence': f"{analysis.get('confidence', 0) * 100:.1f}%",
                'Overall Score': f"{analysis.get('overall_score', 50):.1f}",
                'Reasoning': analysis.get('reasoning', 'Analysis unavailable')
            })
```

```
    df = pd.DataFrame(table_data)
    print("\n" + "—" * 200)
    header_fmt = (f"{'Symbol':<8} {'Price':<10} {'P/E':<8} {'Fwd P/E':<10} {'Market Cap':<12} {'RSI':<6} "
                  f"{'Latest News':<40} {'Sentiment':<12} {'Target':<10} {'Analyst':<10} "
                  f"{'Decision':<12} {'Confidence':<10} {'Score':<8}")
    print(header_fmt)
    print("—" * 200)
    for td in table_data:
        print(f"{td['Symbol']:<8} {td['Current Price']:<10} {td['P/E Ratio']:<8.1f} "
              f"{td['Forward P/E']:<10.1f} {td['Market Cap']:<12} {td['RSI']:<6.1f} "
              f"{td['Latest News']:<40} {td['News Sentiment']:<12} {td['Analyst Target']:<10} "
              f"{td['Analyst Rec']:<10} {td['AI Decision']:<12} {td['Confidence']:<10} "
              f"{td['Overall Score']:<8}")
    print("—" * 200)
    print("\n📋 DETAILED REASONING:")
    print("—" * 100)
    for td in table_data:
        print(f"\n🔍 {td['Symbol']}: {td['Reasoning']}")
    return df
```

SECTION 11: MAIN EXECUTION FUNCTION (main)

a. Description:

Serves as the **primary entry point** and **execution driver** for the Autonomous Investment Research Agent. It defines the scope of the analysis (the list of target stock symbols), initializes the top-level `FinancialAnalysisSystem` orchestrator, triggers the comprehensive analysis workflow, and presents the final consolidated results and a summary report to the user upon completion.

b. Key Methods/Features:

i. **`predefined_symbols`**: Defines a static list of stock tickers (e.g., major tech stocks like "AAPL", "MSFT") that will be the target of the analysis run.

ii. `FinancialAnalysisSystem()` **Instantiation**: Creates the main orchestrator object, which in turn initializes all the specialized agents (`DataAgent`, `NewsAgent`, `AnalysisAgent`, `VisualizationEngine`).

iii. **Calling** `system.analyze_stocks(predefined_symbols)`: Initiates the core multi-agent workflow coordinated by the `FinancialAnalysisSystem` for all specified symbols. This is where the bulk of the data fetching, processing, and analysis occurs.

iv. **Calling** `system.display_comprehensive_table(predefined_symbols)`: Invokes the method responsible for formatting and printing the detailed, comparative results table to the console, providing a structured overview of key metrics and AI decisions for each stock.

v. **Summary Report Generation**: Calculates and prints high-level statistics after the analysis, including the number of stocks successfully analyzed and a distribution count of the final AI recommendations (e.g., number of "BUY", "HOLD", "SELL" decisions).

vi. `if __name__ == "__main__":`: Standard Python construct that ensures the `main()` function is called only when the script is executed directly (not when imported as a module).

c. Pipeline Role:

main() function acts as the **Top-Level Script Executor** and **User Interface Initiator**. It defines the specific task (which stocks to analyze), kicks off the entire orchestrated agentic pipeline by calling the `FinancialAnalysisSystem`, and is responsible for presenting the final, summarized outputs directly to the user in a readable format (console tables and summary statistics). It represents the start and end points of a complete analysis run.

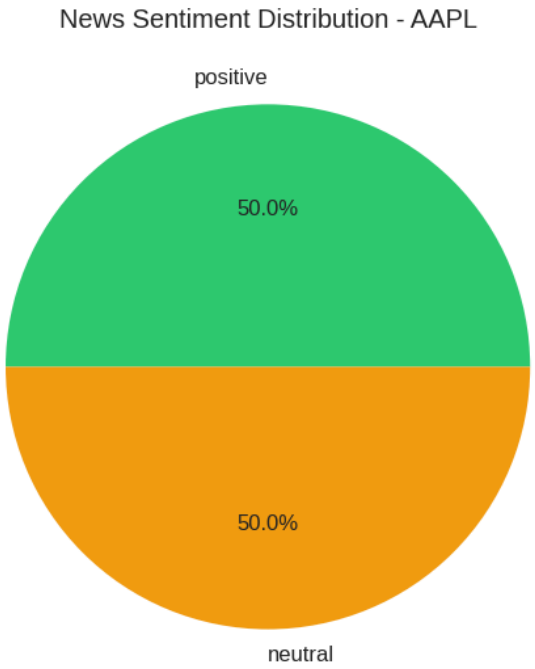
```
# ================================================================================
# SECTION 11: MAIN CALLING FUNCTION
# ================================================================================

def main():
    """Main function to run the complete financial analysis system"""

    # Predefined list of major tech stocks
    predefined_symbols = [
        "AAPL",    # Apple
        "MSFT",    # Microsoft
        "GOOGL"    # Alphabet (Class A)
```

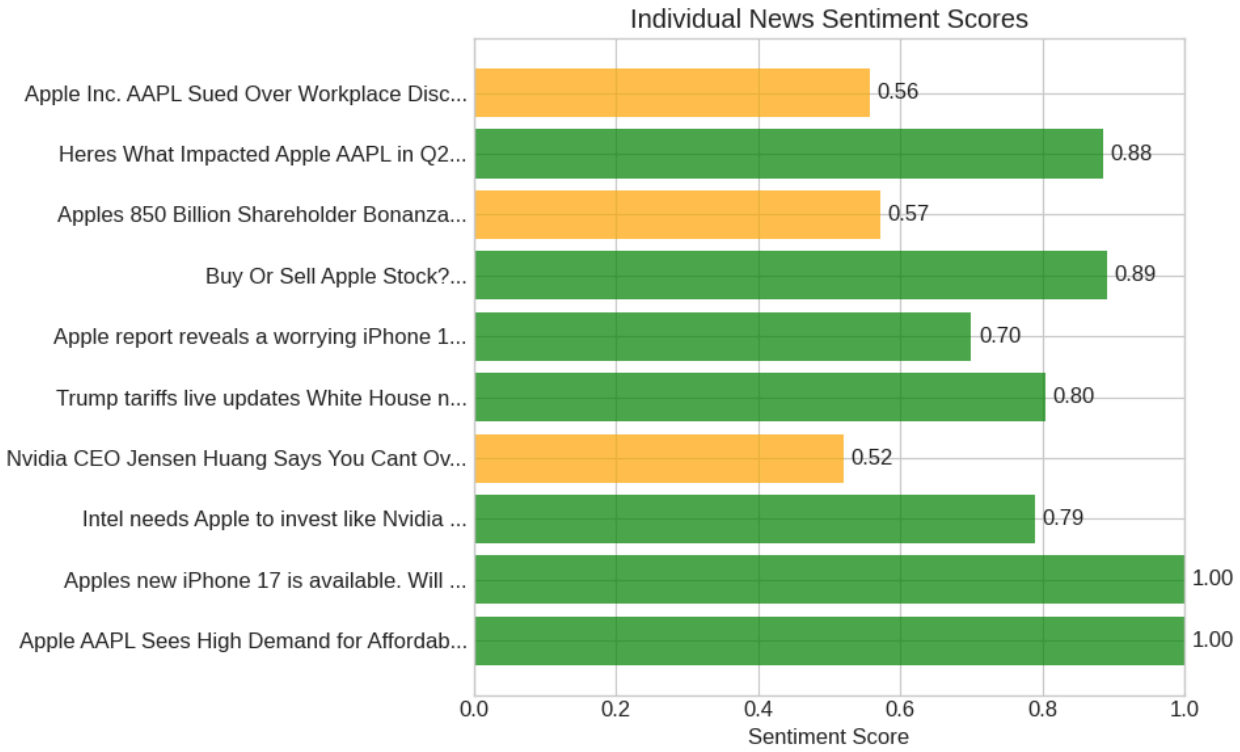
```
    "GOOGL",    # Alphabet (Class A)
    "GOOG",     # Alphabet (Class C)
    "NVDA",     # NVIDIA
    "ORCL",     # Oracle
    "AMD",      # Advanced Micro Devices
    "INTC",     # Intel
    "AMZN",     # Amazon
    "META"      # Meta Platforms
]

print("\n" + "💎" * 60)
print("🚀 AUTONOMOUS INVESTMENT RESEARCH AGENT")
print("💎" * 60)
print(f"Analyzing {len(predefined_symbols)} major tech stocks...")
print("Stocks:", ", ".join(predefined_symbols))
print("\n" + "=" * 80)

# Initialize the system
system = FinancialAnalysisSystem()

# Run comprehensive analysis
print("\n🎯 Starting Comprehensive Analysis...")
results = system.analyze_stocks(predefined_symbols)

# Display results table
print("\n" + "📊" * 60)
print("COMPREHENSIVE ANALYSIS RESULTS")
print("📊" * 60)
system.display_comprehensive_table(predefined_symbols)

# Generate summary report
print("\n" + "📈" * 60)
print("ANALYSIS SUMMARY")
print("📈" * 60)

# Calculate overall statistics
total_stocks = len(predefined_symbols)
successful_analyses = sum(1 for symbol in predefined_symbols if 'error' not in results.get(symbol, {}))
```

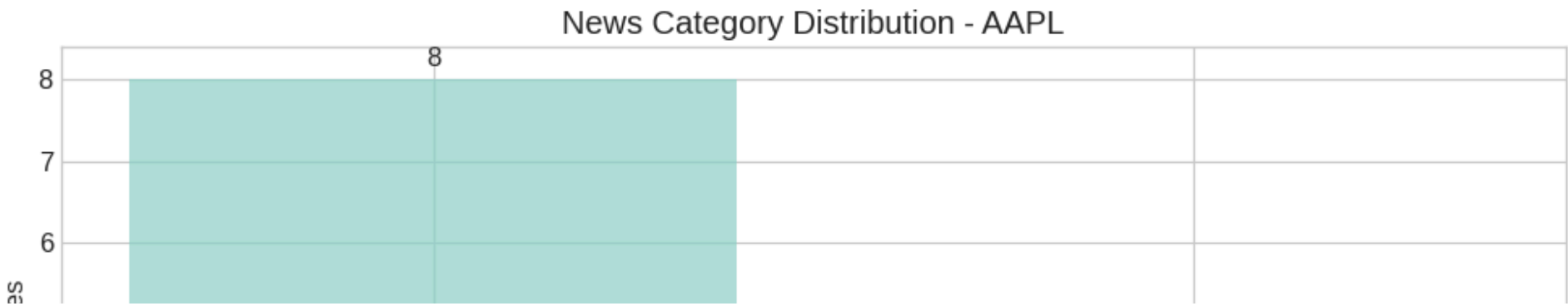
```
    print(f"\n📊 Analysis Completion: {successful_analyses}/{total_stocks} stocks successfully analyzed")

    # Count recommendations
    decisions = []
    for symbol in predefined_symbols:
        if symbol in results and 'analysis_result' in results[symbol]:
            decision = results[symbol]['analysis_result'].get('decision', 'HOLD')
            decisions.append(decision)

    if decisions:
        decision_counts = pd.Series(decisions).value_counts()
        print("\n🎯 Recommendation Distribution:")
        for decision, count in decision_counts.items():
            percentage = (count / len(decisions)) * 100
            print(f"   {decision}: {count} stocks ({percentage:.1f}%)")

    print("\n✅ Analysis completed successfully!")
    print("💡 Check the generated visualizations for detailed insights")

if __name__ == "__main__":
    # Run the main analysis
    main()
```

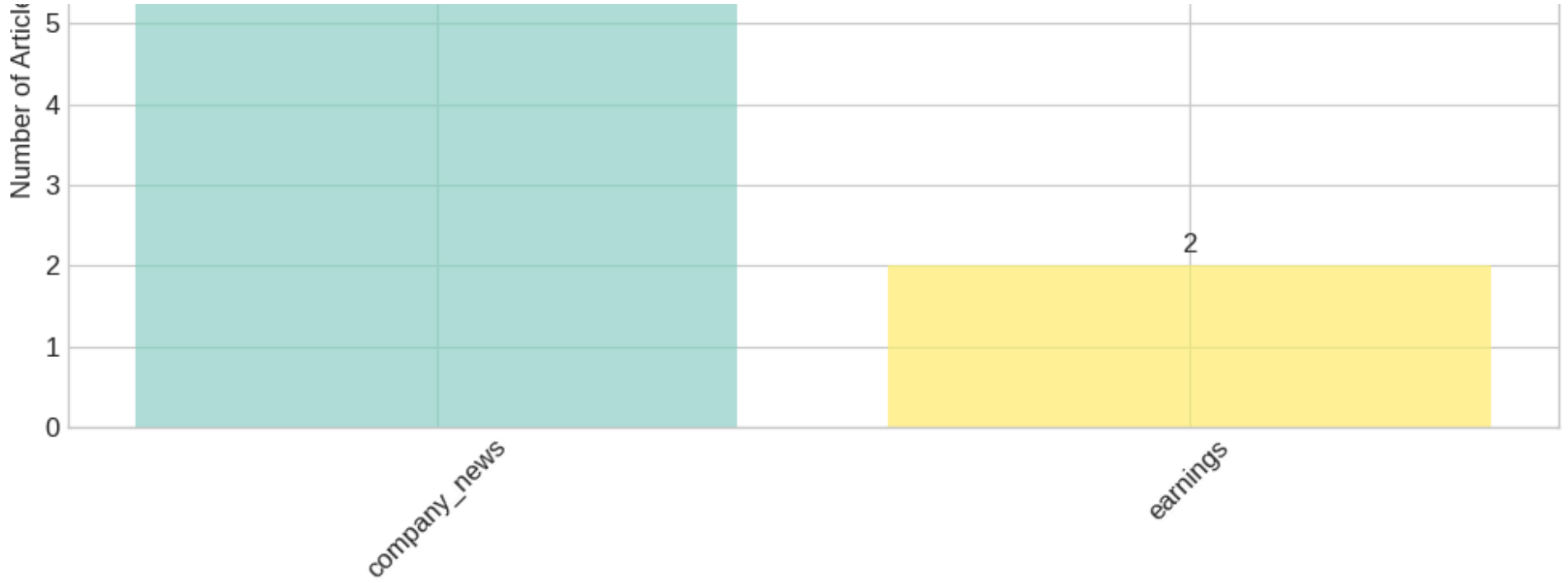
```
💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎
🚀 AUTONOMOUS INVESTMENT RESEARCH AGENT
💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎💎
Analyzing 10 major tech stocks...
Stocks: AAPL, MSFT, GOOGL, GOOG, NVDA, ORCL, AMD, INTC, AMZN, META

================================================================================
[2025-10-19 13:51:58] DataAgent (INFO): No memory file found at 'agent_memory/DataAgent_memory.json'. Starting fresh.
[2025-10-19 13:51:58] NewsAgent (INFO): No memory file found at 'agent_memory/NewsAgent_memory.json'. Starting fresh.
[2025-10-19 13:51:58] EarningsAnalyzer (INFO): No memory file found at 'agent_memory/EarningsAnalyzer_memory.json'. Starting fre
[2025-10-19 13:51:58] MarketAnalyzer (INFO): No memory file found at 'agent_memory/MarketAnalyzer_memory.json'. Starting fresh.
[2025-10-19 13:51:58] CompanyNewsAnalyzer (INFO): No memory file found at 'agent_memory/CompanyNewsAnalyzer_memory.json'. Starti
config.json: 100%    929/929 [00:00<00:00, 94.2kB/s]
pytorch_model.bin: 100%    501M/501M [00:09<00:00, 59.8MB/s]
Some weights of the model checkpoint at cardiffnlp/twitter-roberta-base-sentiment-latest were not used when initializing Roberta
- This IS expected if you are initializing RobertaForSequenceClassification from the checkpoint of a model trained on another ta
- This IS NOT expected if you are initializing RobertaForSequenceClassification from the checkpoint of a model that you expect t
vocab.json:    899k/? [00:00<00:00, 41.5MB/s]
merges.txt:    456k/? [00:00<00:00, 29.2MB/s]
model.safetensors: 100%    501M/501M [00:14<00:00, 31.6MB/s]
special_tokens_map.json: 100%    239/239 [00:00<00:00, 28.2kB/s]
Device set to use cuda:0
config.json:    1.58k/? [00:00<00:00, 62.1kB/s]
model.safetensors: 100%    1.63G/1.63G [00:46<00:00, 91.5MB/s]
generation_config.json: 100%    363/363 [00:00<00:00, 42.7kB/s]
vocab.json:    899k/? [00:00<00:00, 29.9MB/s]
merges.txt:    456k/? [00:00<00:00, 22.8MB/s]
tokenizer.json:    1.36M/? [00:00<00:00, 41.4MB/s]
Device set to use cuda:0
[2025-10-19 13:53:07] NewsAgent (INFO): LLM models initialized successfully
[2025-10-19 13:53:07] NewsAgent (INFO): NewsAPI client initialized successfully
```

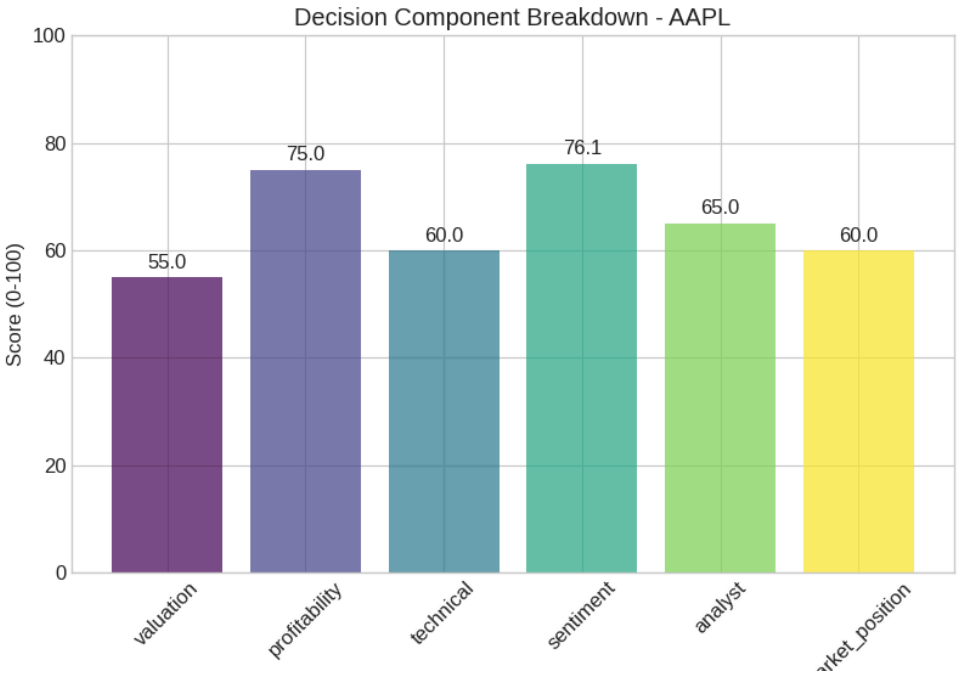
```
[2025-10-19 13:53:07] AnalysisAgent (INFO): No memory file found at 'agent_memory/AnalysisAgent_memory.json'. Starting fresh.
🚀 Multi-Agent Financial Analysis System Initialized
============================================================

🎯 Starting Comprehensive Analysis...
🔍 Analyzing AAPL...
[2025-10-19 13:53:13] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.734, Summary Quality: 0.828
✅ AAPL analysis completed
📊 Creating visualizations for AAPL...
```

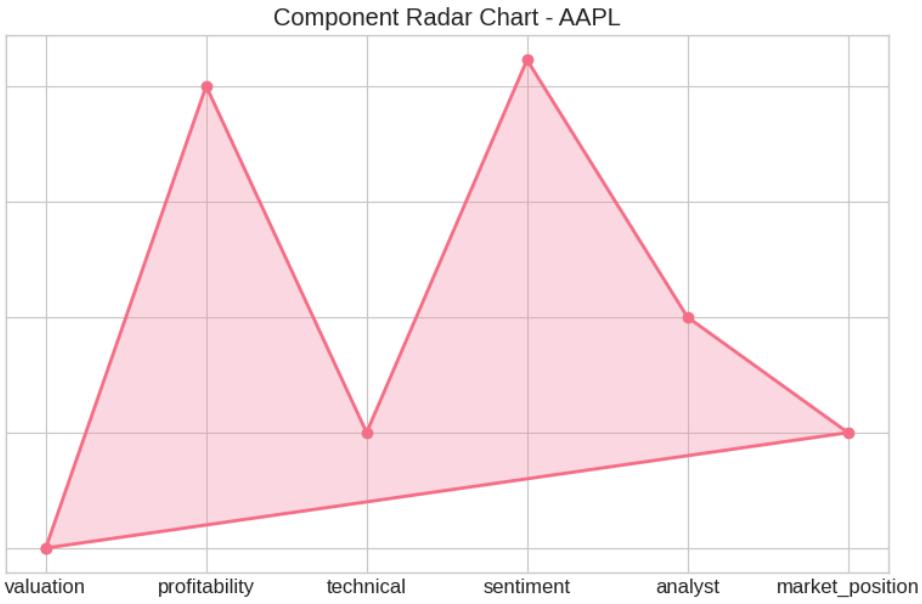






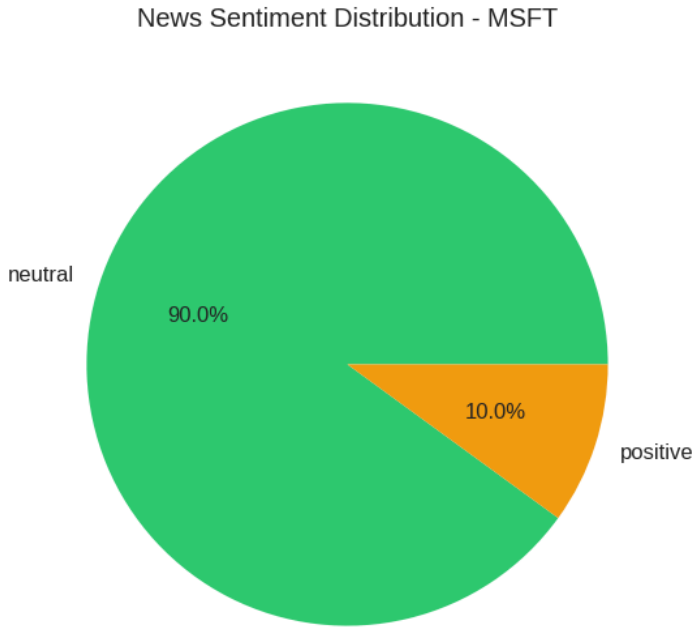

Number of Articl
5
4
3
2
1
0
2
company_news
earnings
News Categories

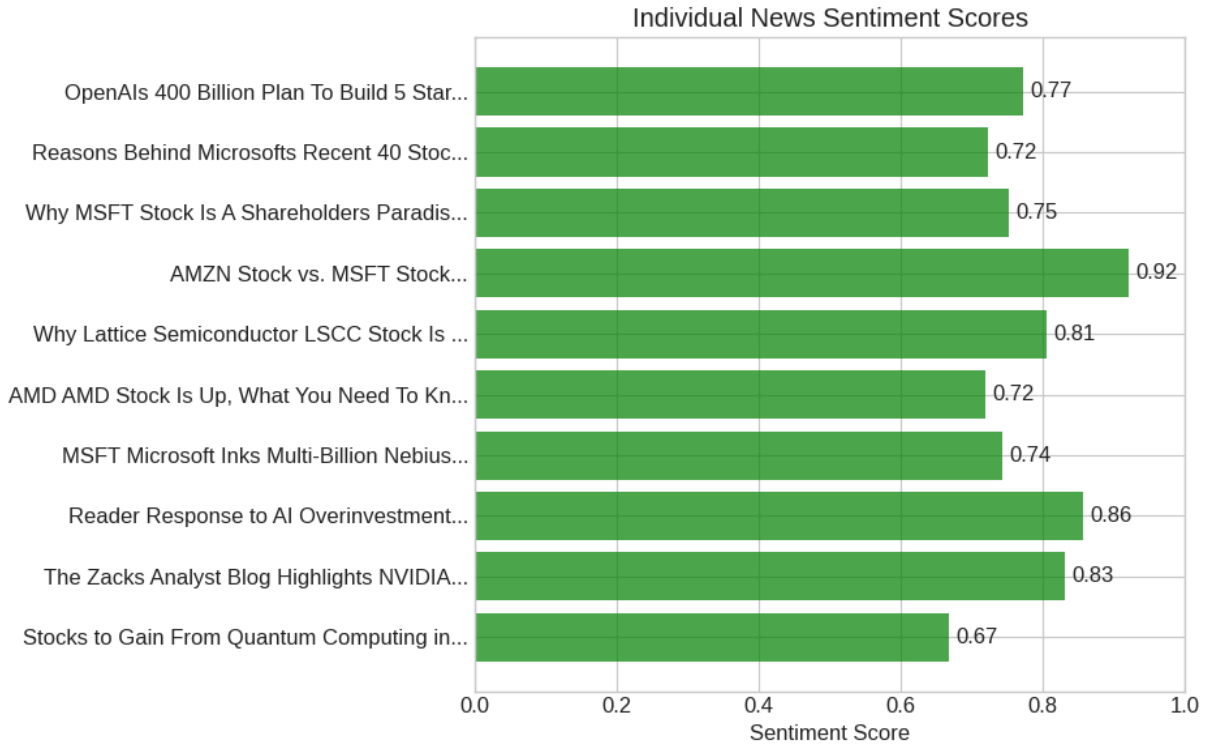

Decision Component Breakdown - AAPL
Score (0-100)
100
80
60
40
20
0
55.0
75.0
60.0
76.1
65.0
60.0
valuation
profitability
technical
sentiment
analyst

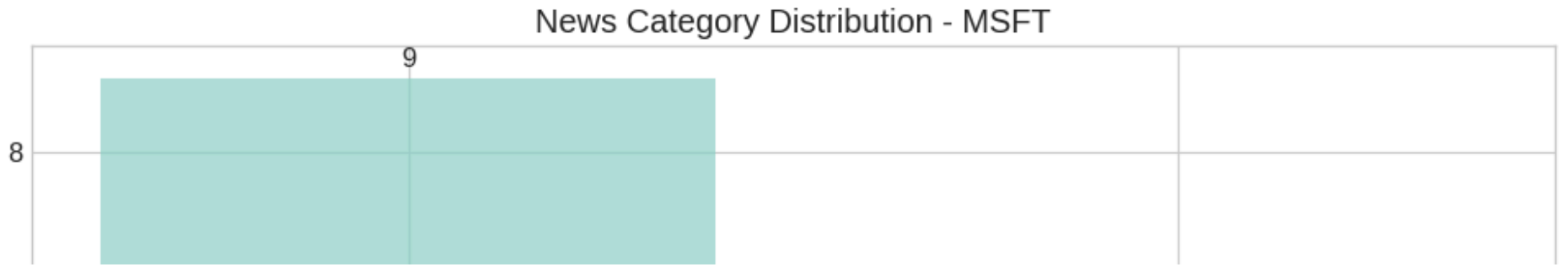

Component Radar Chart - AAPL
valuation
profitability
technical
sentiment
analyst
market_position

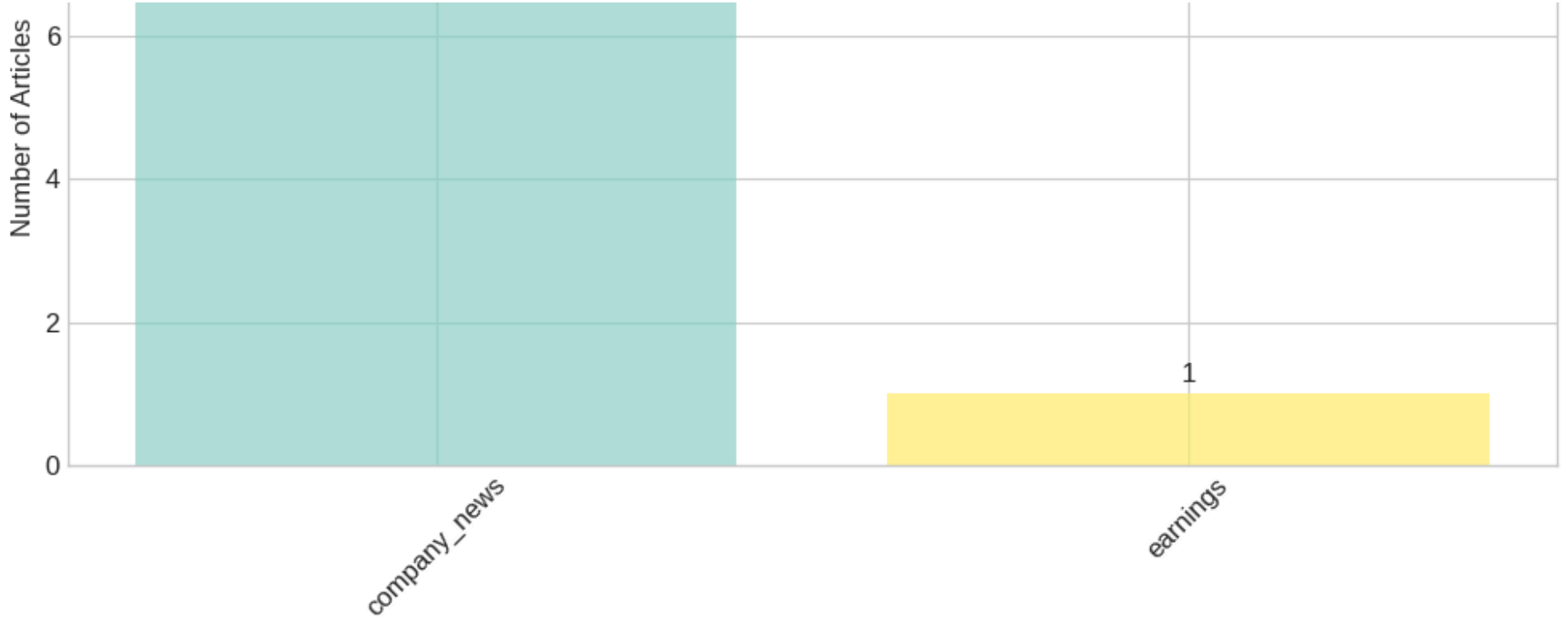

Price History - AAPL
Close Price
Price ($)
260
250
240
230
220
210
200
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15
Relative Strength Index - AAPL
Overbought (70)
Oversold (30)
RSI
100
80
60
40

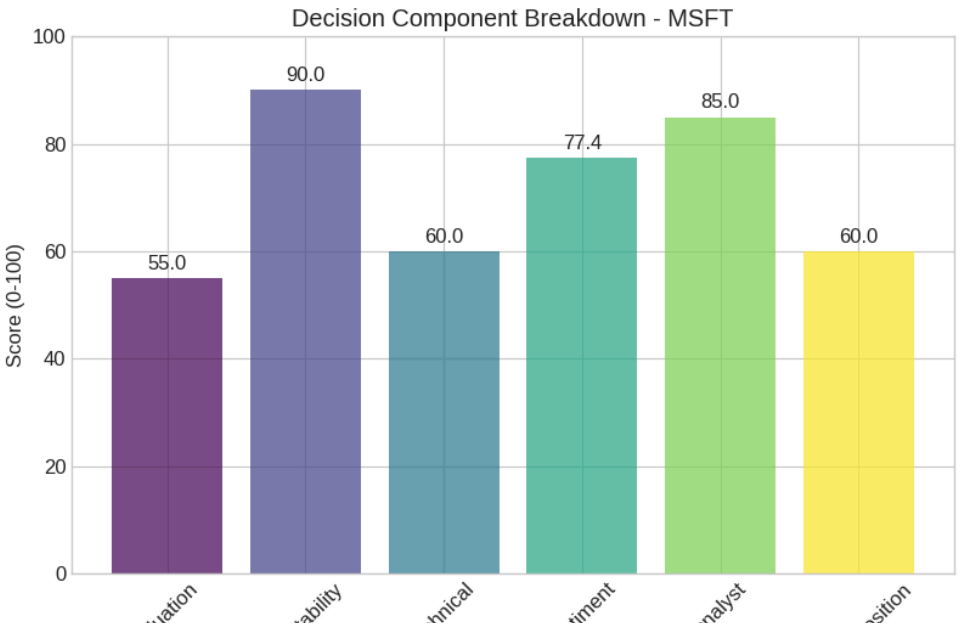
🔍 Analyzing MSFT...
You seem to be using the pipelines sequentially on GPU. In order to maximize efficiency please use a dataset
[2025-10-19 13:53:20] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.756, Summary Quality: 0.823
✅ MSFT analysis completed
📊 Creating visualizations for MSFT...

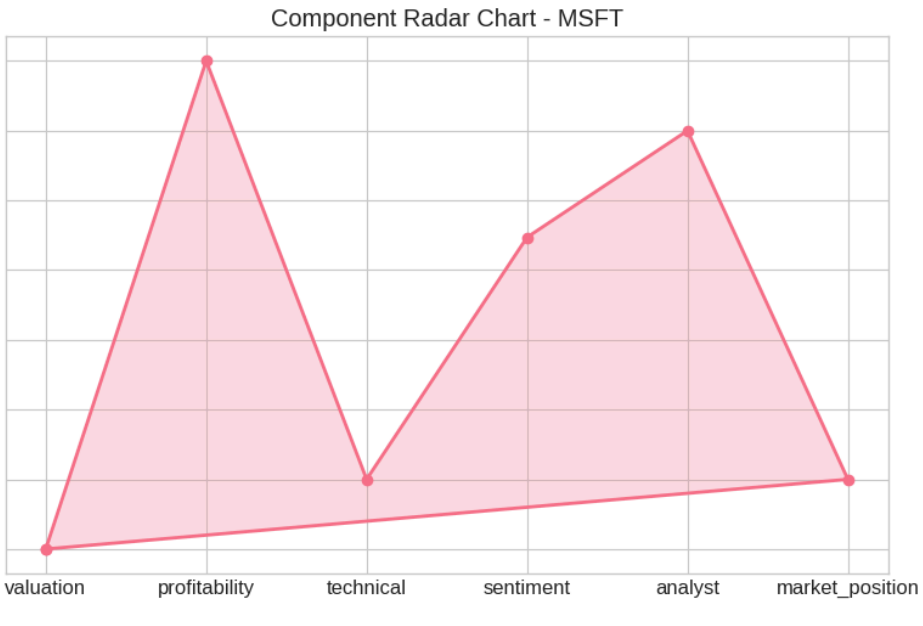




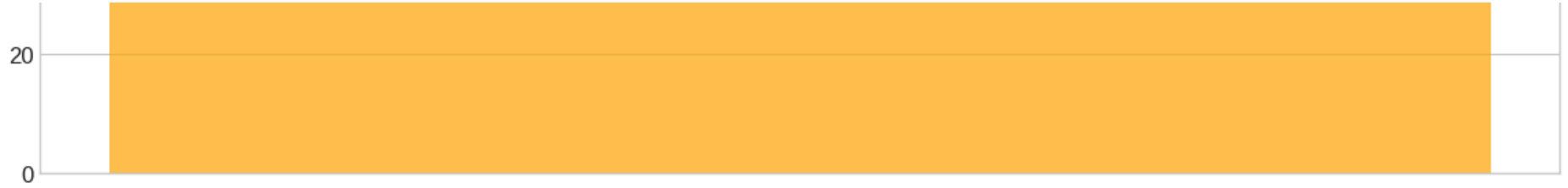



Number of Articles
6
4
2
0
1
company_news
earnings
News Categories

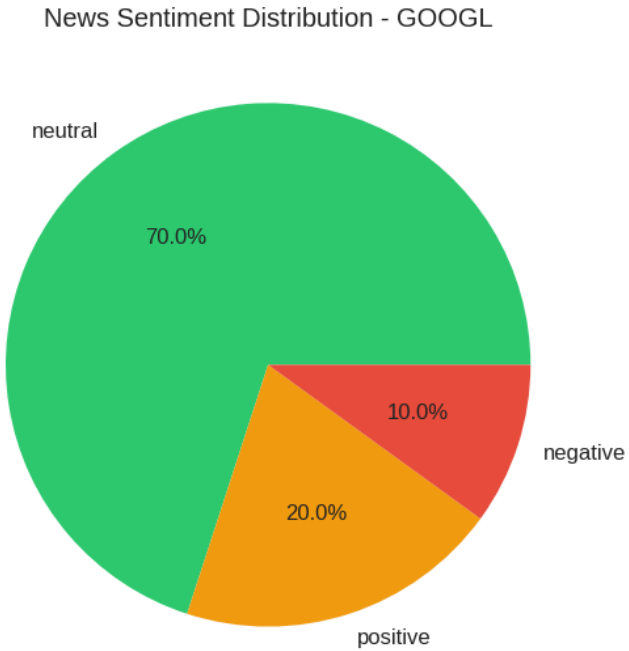

Decision Component Breakdown - MSFT
Score (0-100)
100
80
60
40
20
0
55.0
90.0
60.0
77.4
85.0
60.0

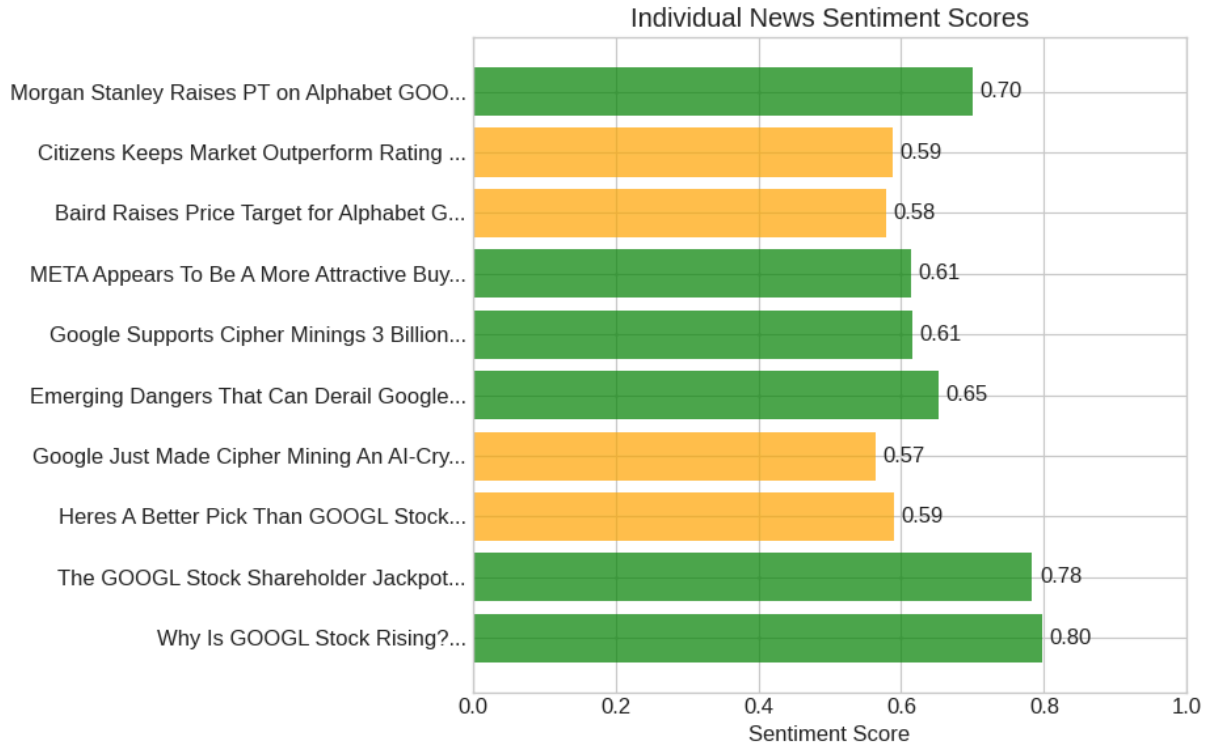

Component Radar Chart - MSFT
valuation
profitability
technical
sentiment
analyst
market_position

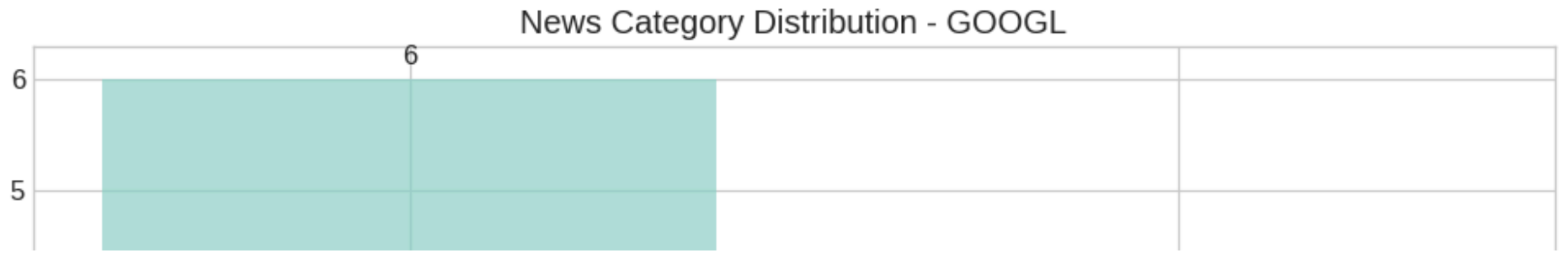

Price History - MSFT
Close Price
Price ($)
535
530
525
520
515
510
505
500
495
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15
Relative Strength Index - MSFT
Overbought (70)
Oversold (30)
RSI
100
80
60
40

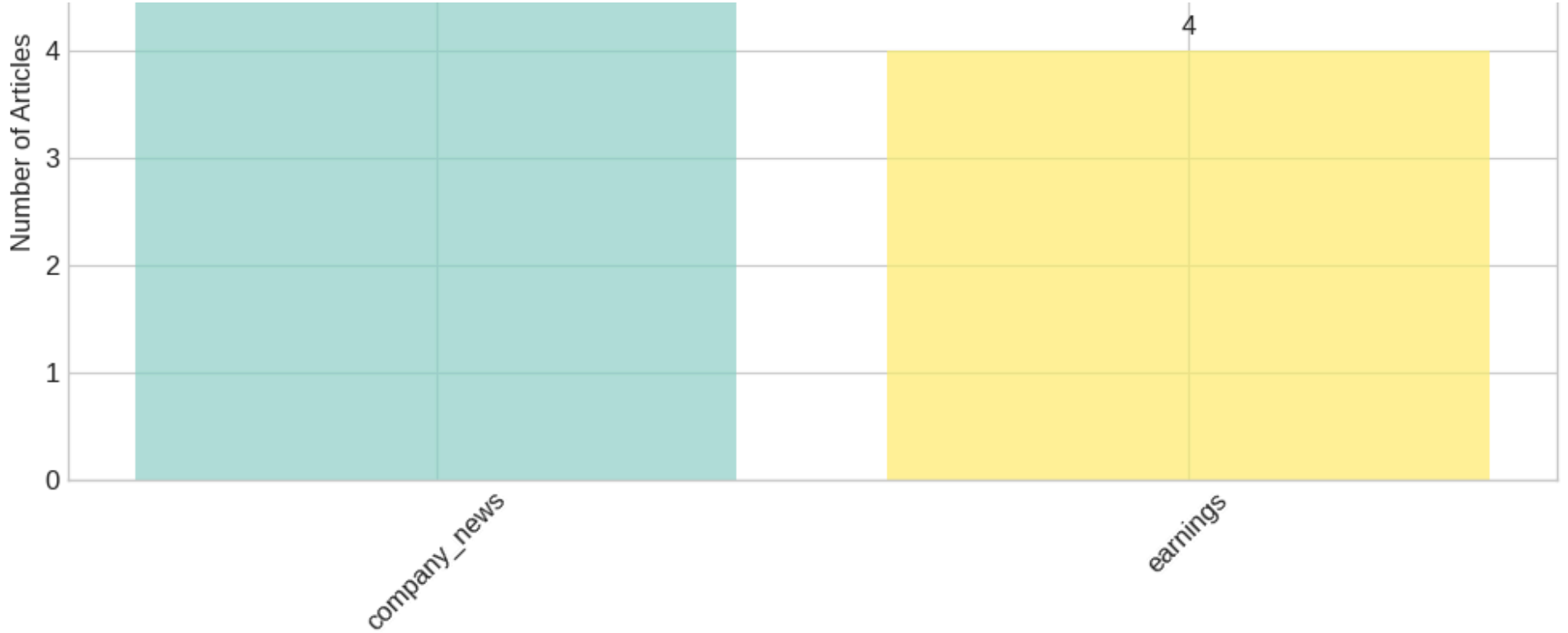


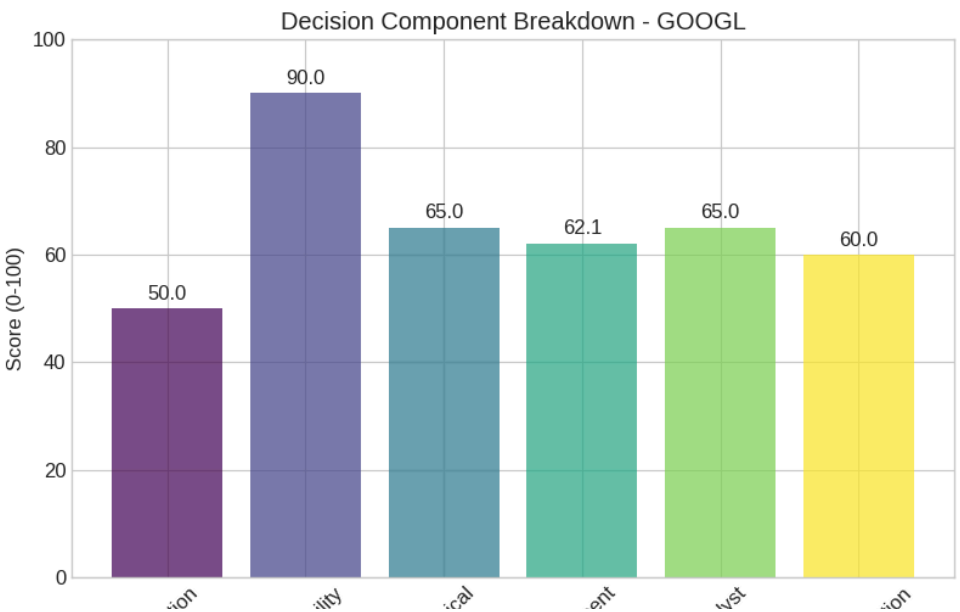
🔍 Analyzing GOOGL...
[2025-10-19 13:53:27] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.720, Summary Quality: 0.831
✅ GOOGL analysis completed
📊 Creating visualizations for GOOGL...

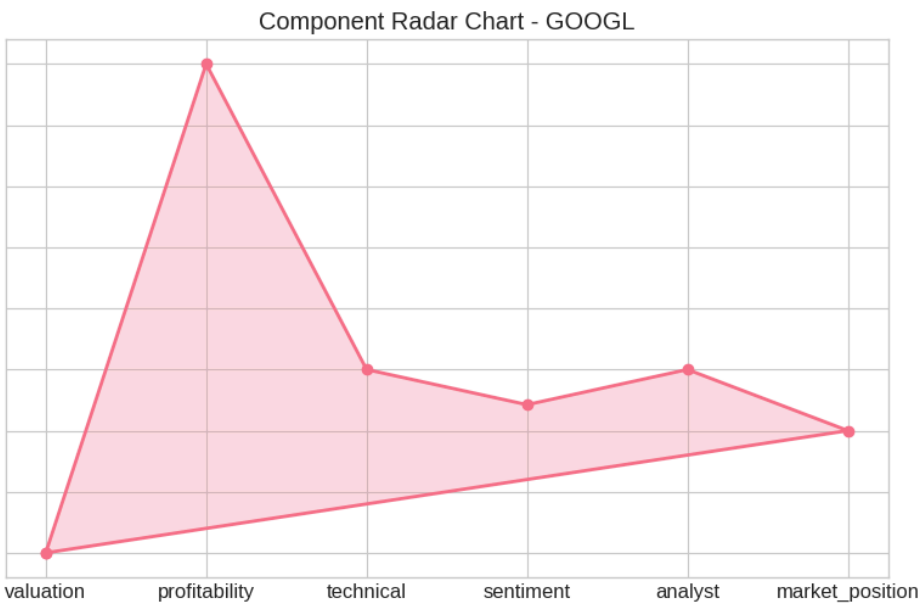


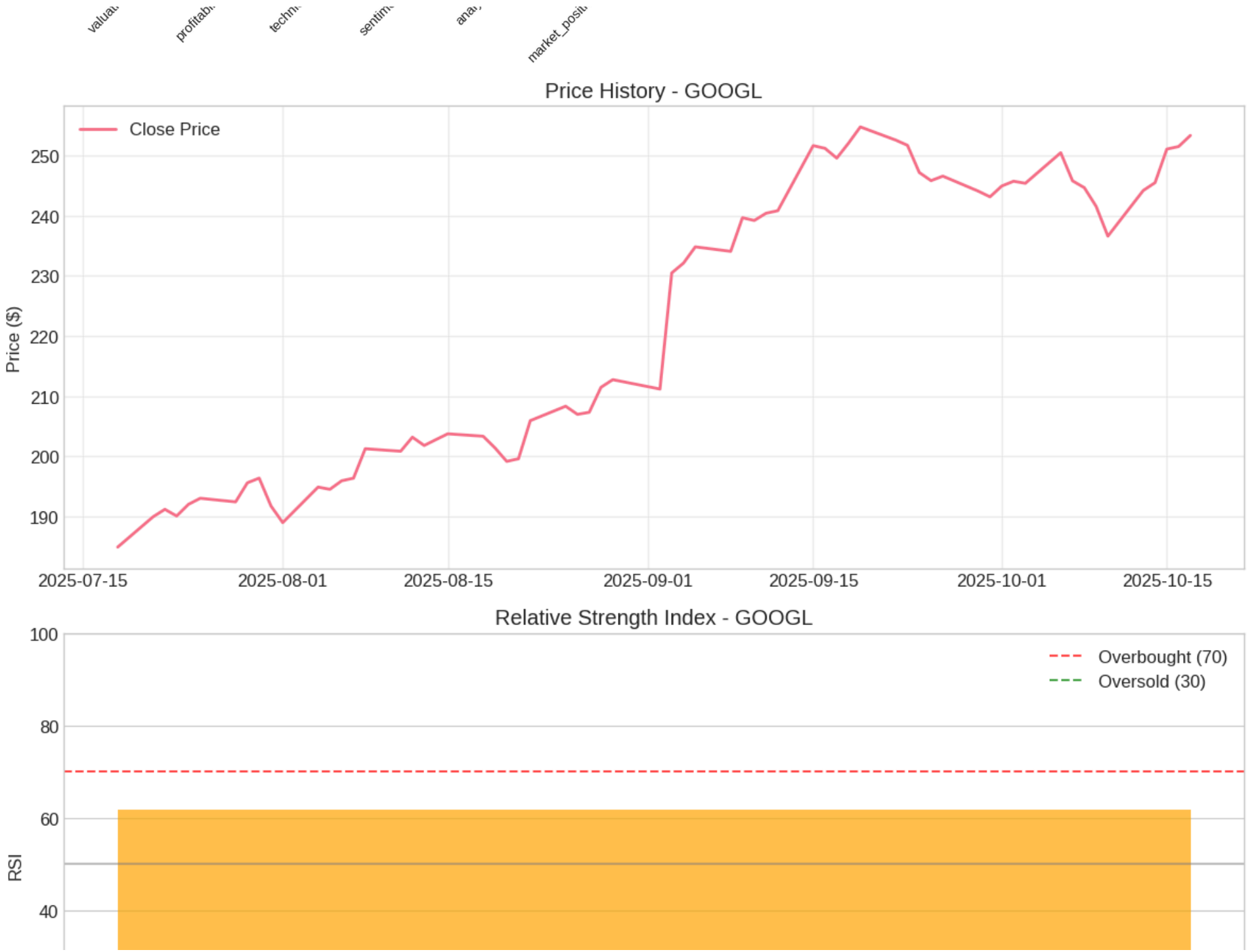




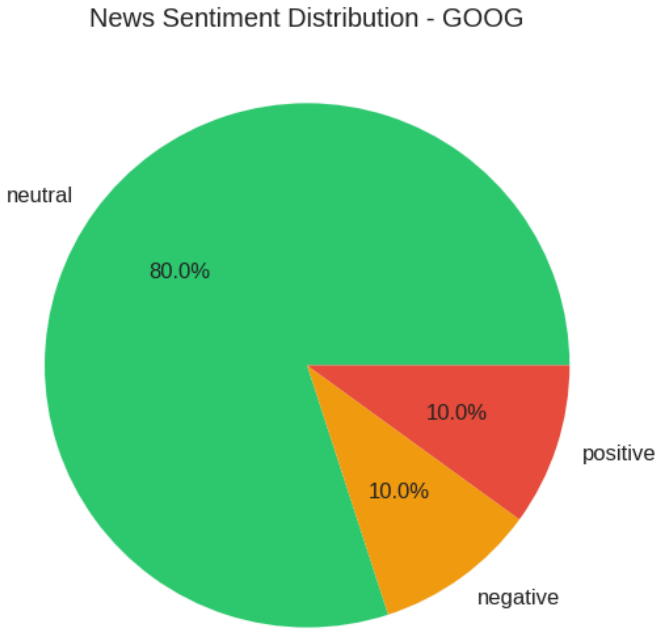

Number of Articles
0
1
2
3
4
4
company_news
earnings
News Categories

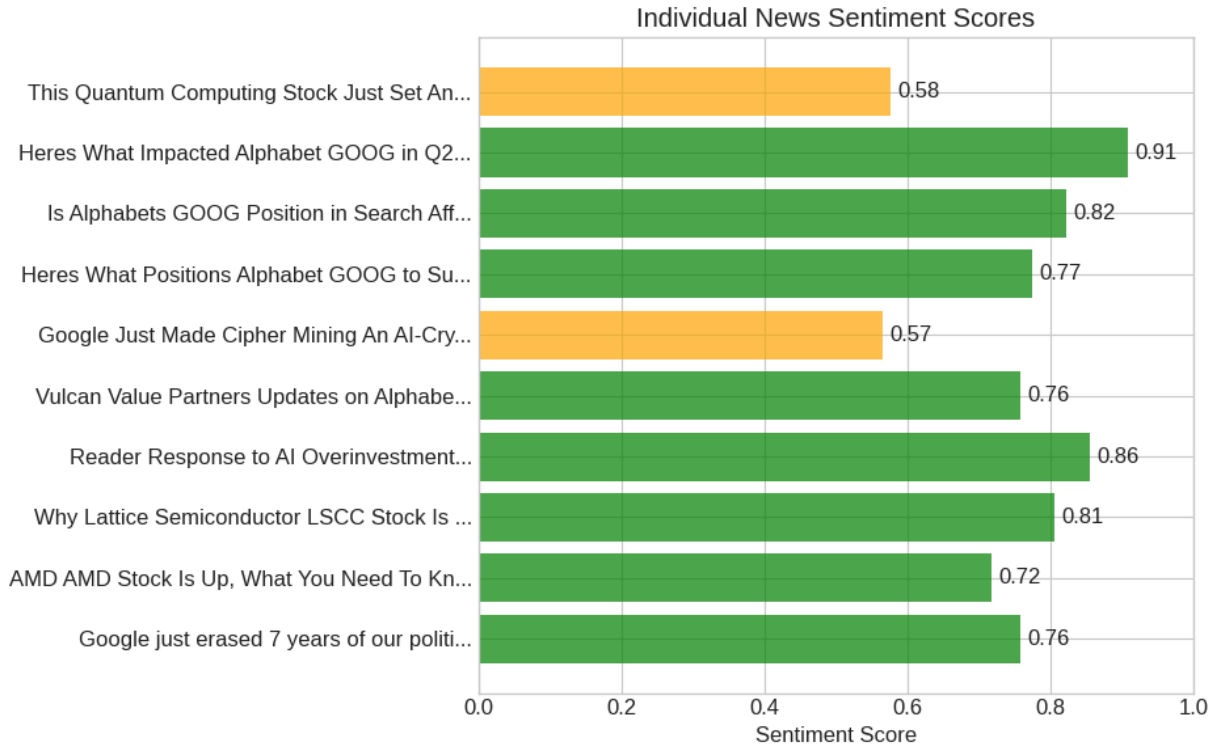

Decision Component Breakdown - GOOGL
Score (0-100)
0
20
40
60
80
100
50.0
90.0
65.0
62.1
65.0
60.0

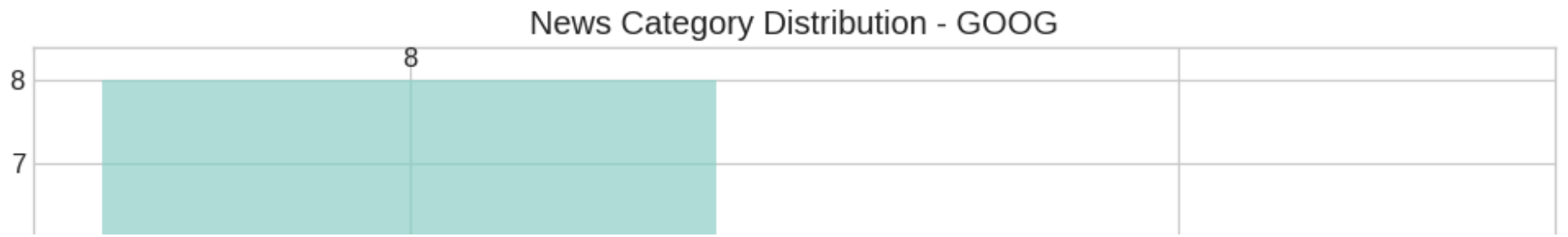

Component Radar Chart - GOOGL
valuation
profitability
technical
sentiment
analyst
market_position

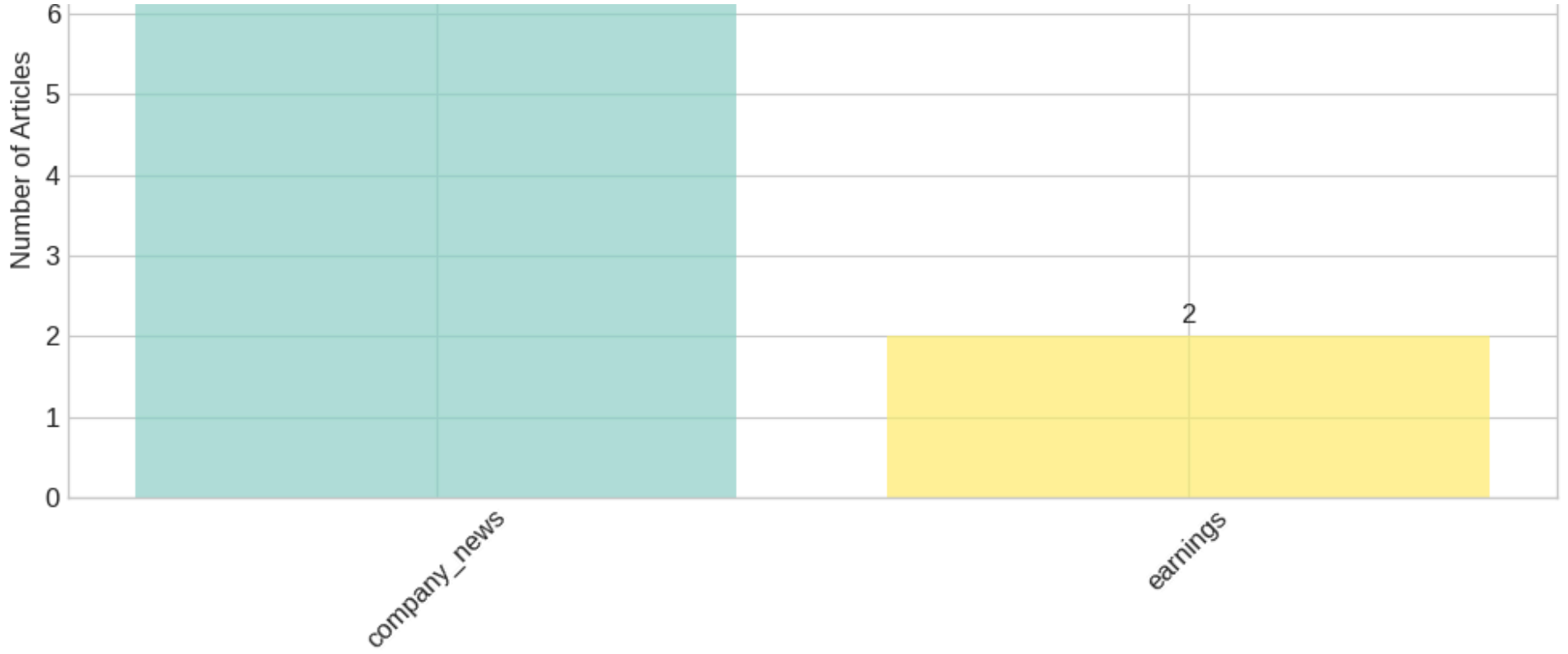

valuat
profitab
techn
sentim
anal
market_posit
Price History - GOOGL
Close Price
Price ($)
250
240
230
220
210
200
190
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15
Relative Strength Index - GOOGL
Overbought (70)
Oversold (30)
RSI
100
80
60
40

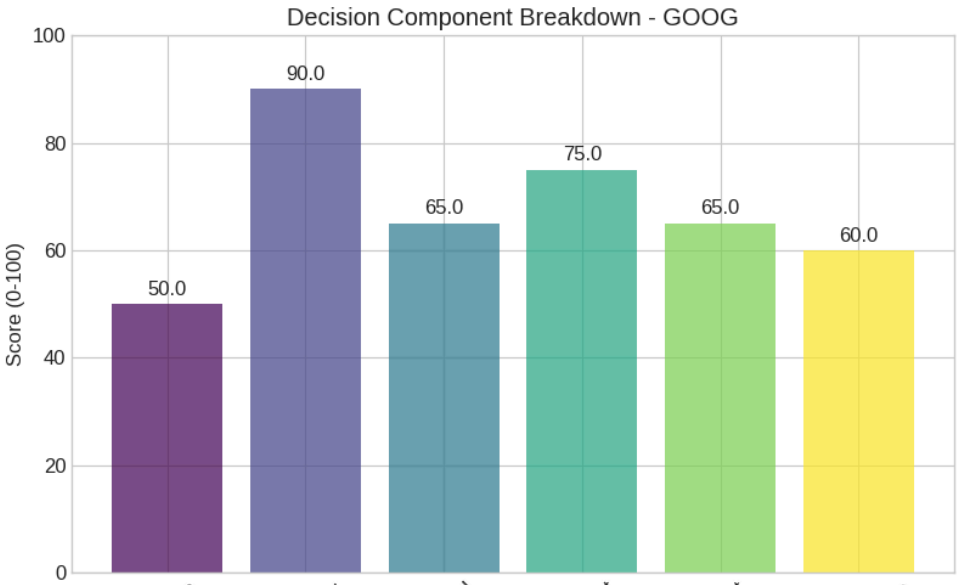
🔍 Analyzing GOOG...
[2025-10-19 13:53:33] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.729, Summary Quality: 0.831
✅ GOOG analysis completed
📊 Creating visualizations for GOOG...

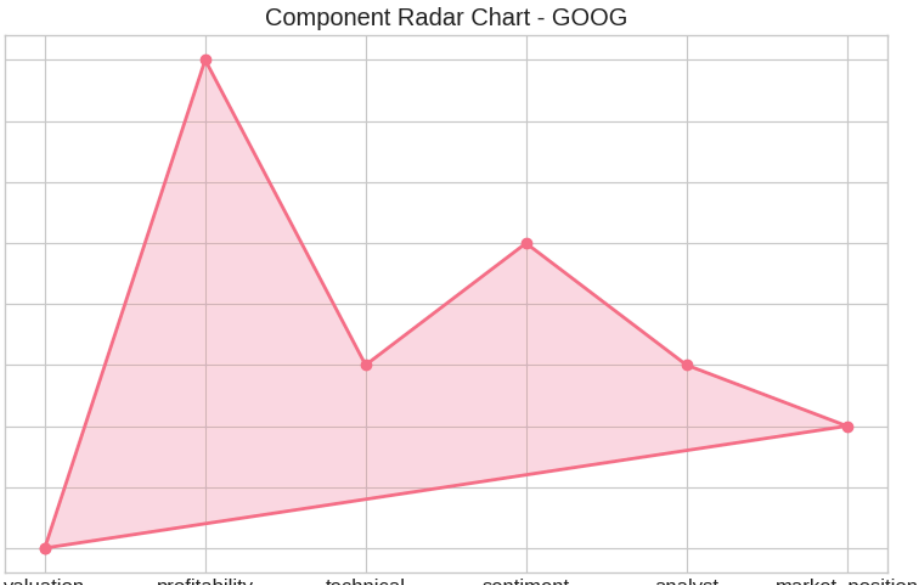


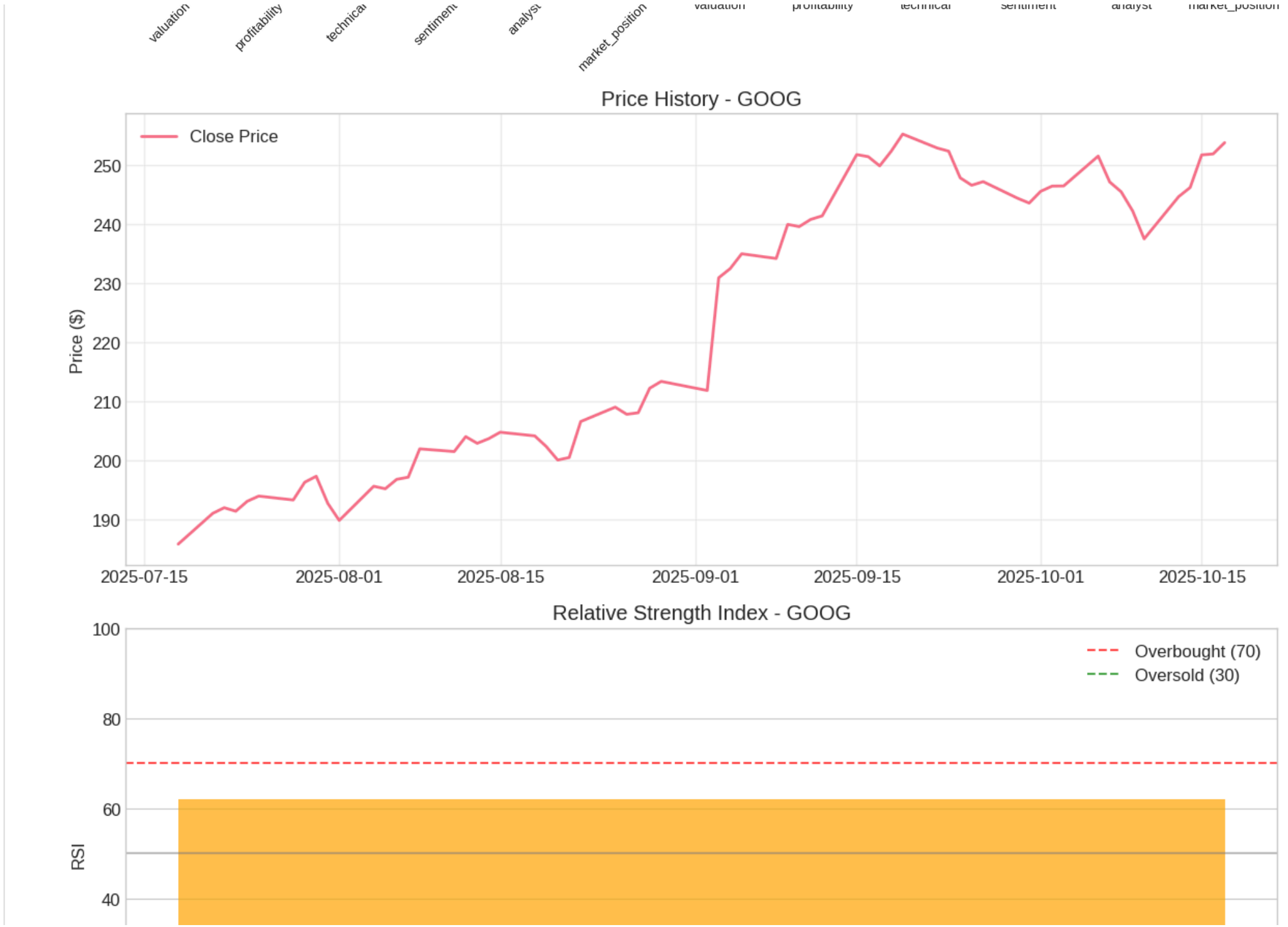


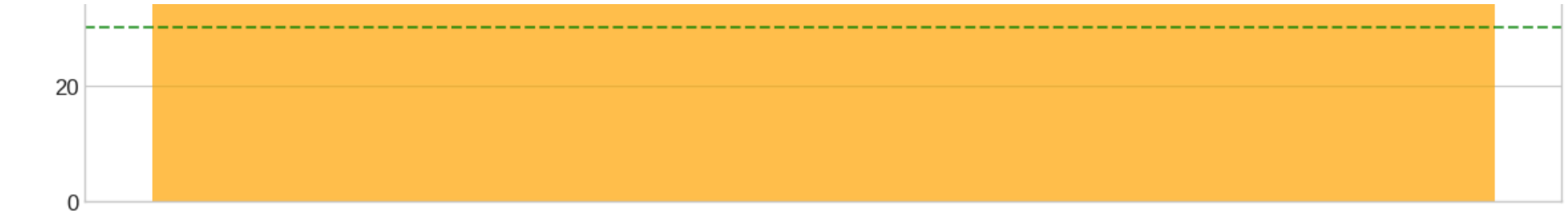



Number of Articles
0
1
2
3
4
5
6
2
company_news
earnings
News Categories

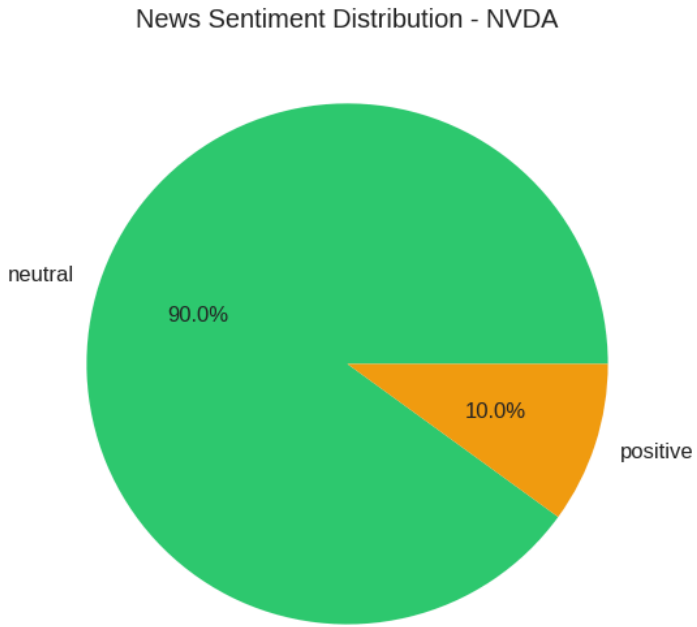

Decision Component Breakdown - GOOG
Score (0-100)
0
20
40
60
80
100
50.0
90.0
65.0
75.0
65.0
60.0

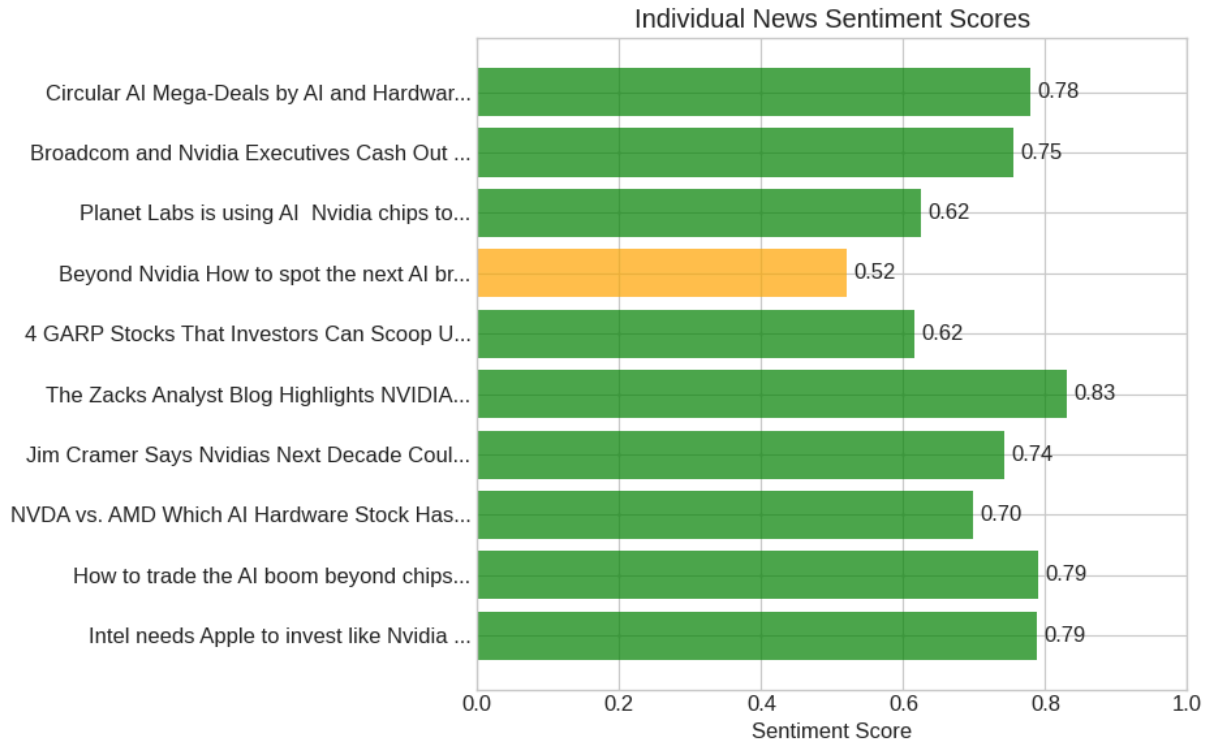

Component Radar Chart - GOOG

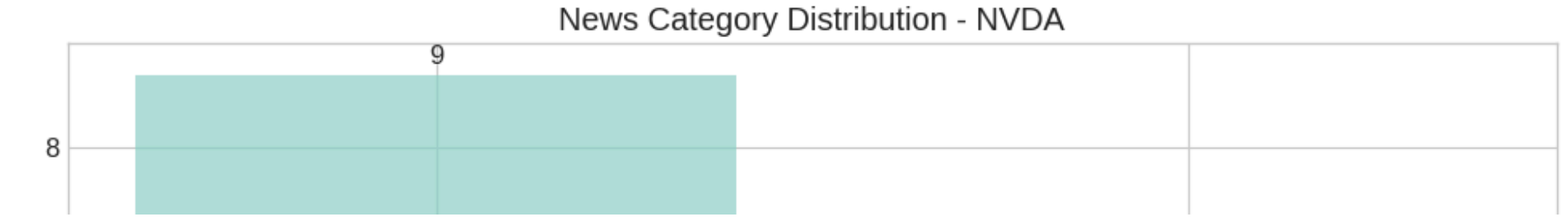

valuation
profitability
technical
sentiment
analyst
market_position
Price History - GOOG
Close Price
Price ($)
250
240
230
220
210
200
190
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15
Relative Strength Index - GOOG
Overbought (70)
Oversold (30)
RSI
100
80
60
40

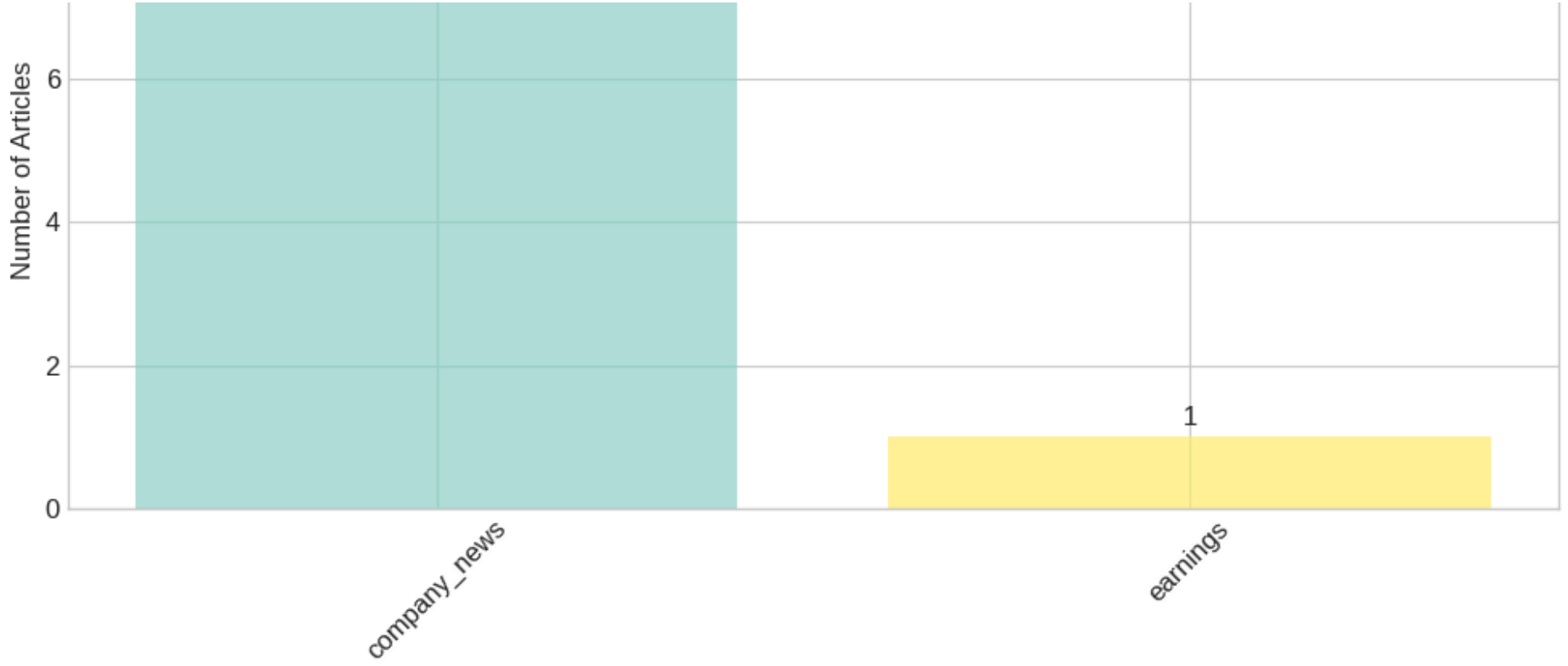


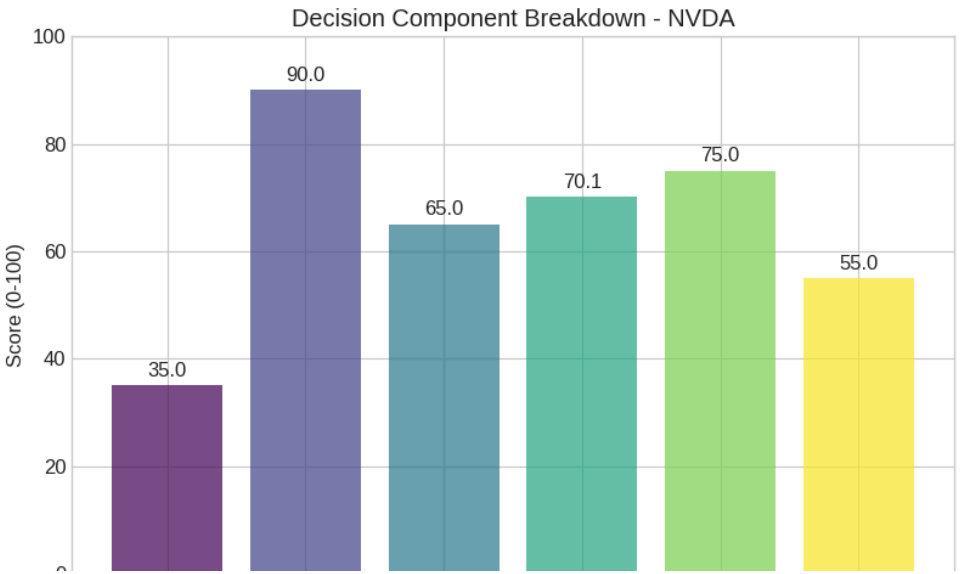
🔍 Analyzing NVDA...
[2025-10-19 13:53:39] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.726, Summary Quality: 0.831
✅ NVDA analysis completed
📊 Creating visualizations for NVDA...

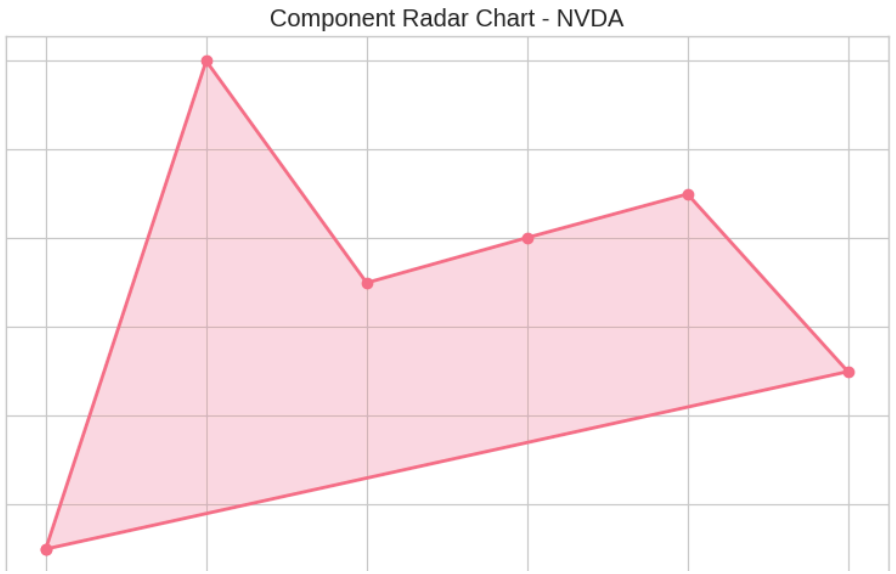






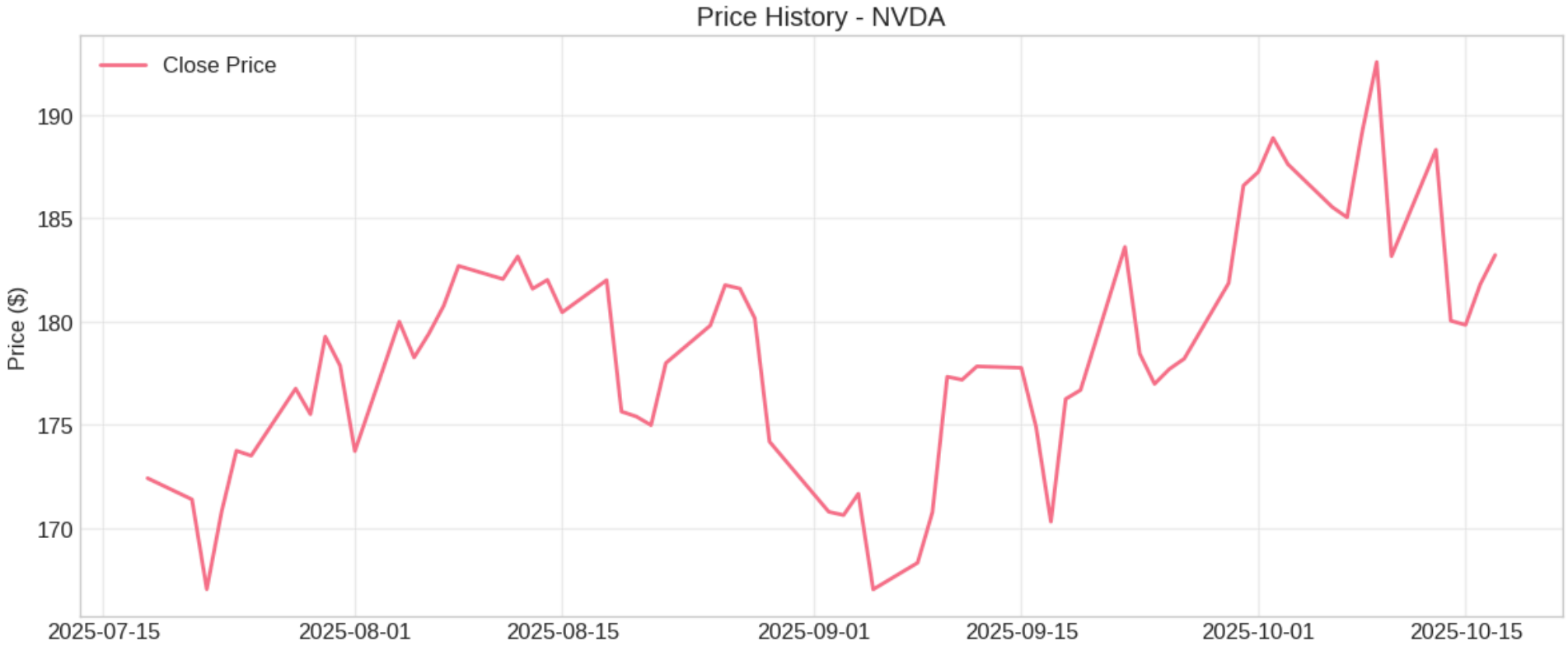

Number of Articles
0
2
4
6
1
company_news
earnings
News Categories

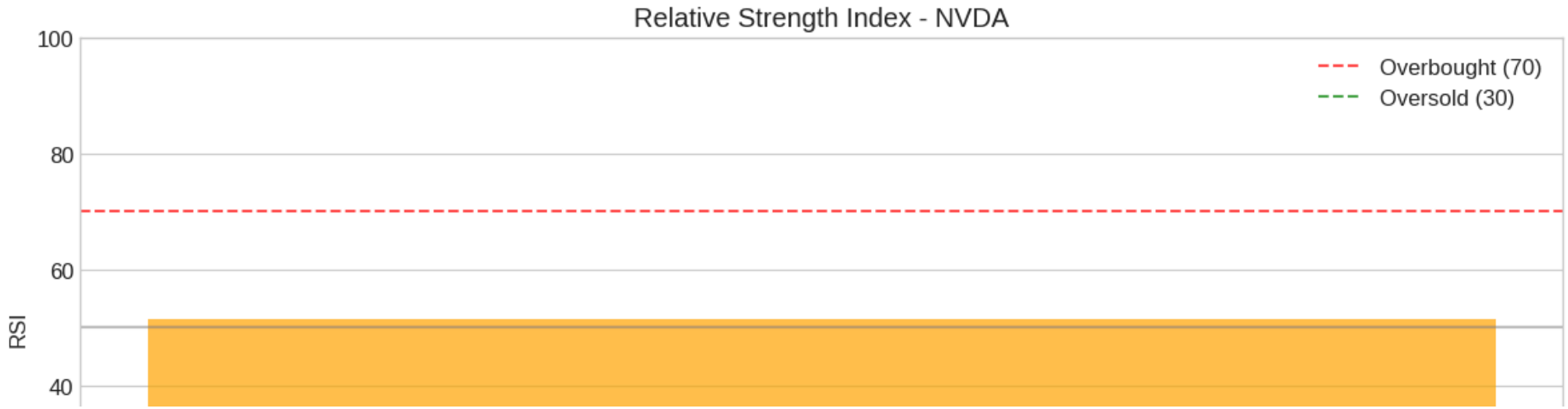

Decision Component Breakdown - NVDA
Score (0-100)
0
20
40
60
80
100
35.0
90.0
65.0
70.1
75.0
55.0

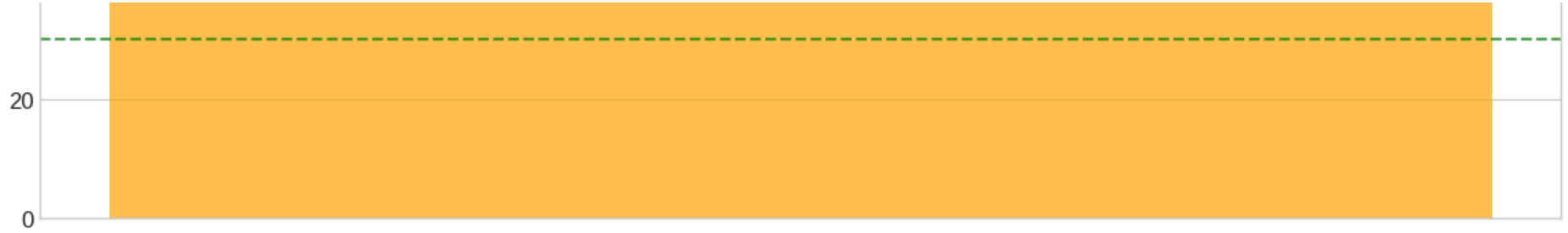

Component Radar Chart - NVDA


0
valuation
profitability
technical
sentiment
analyst
market_position

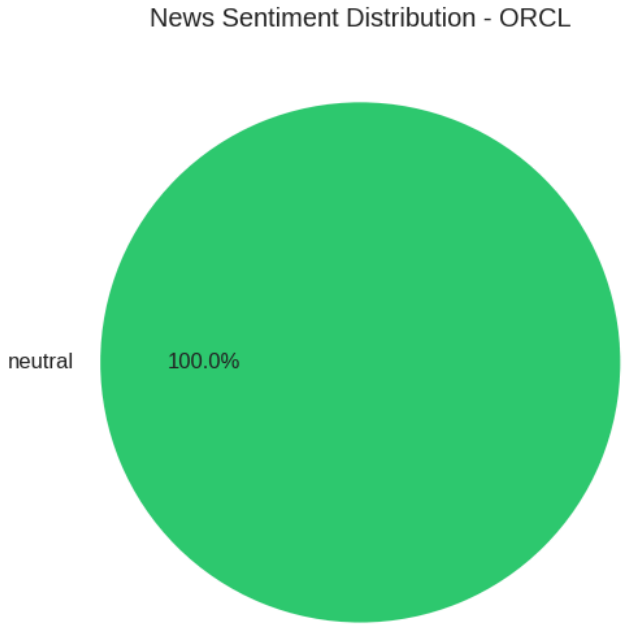

valuation
profitability
technical
sentiment
analyst
market_position

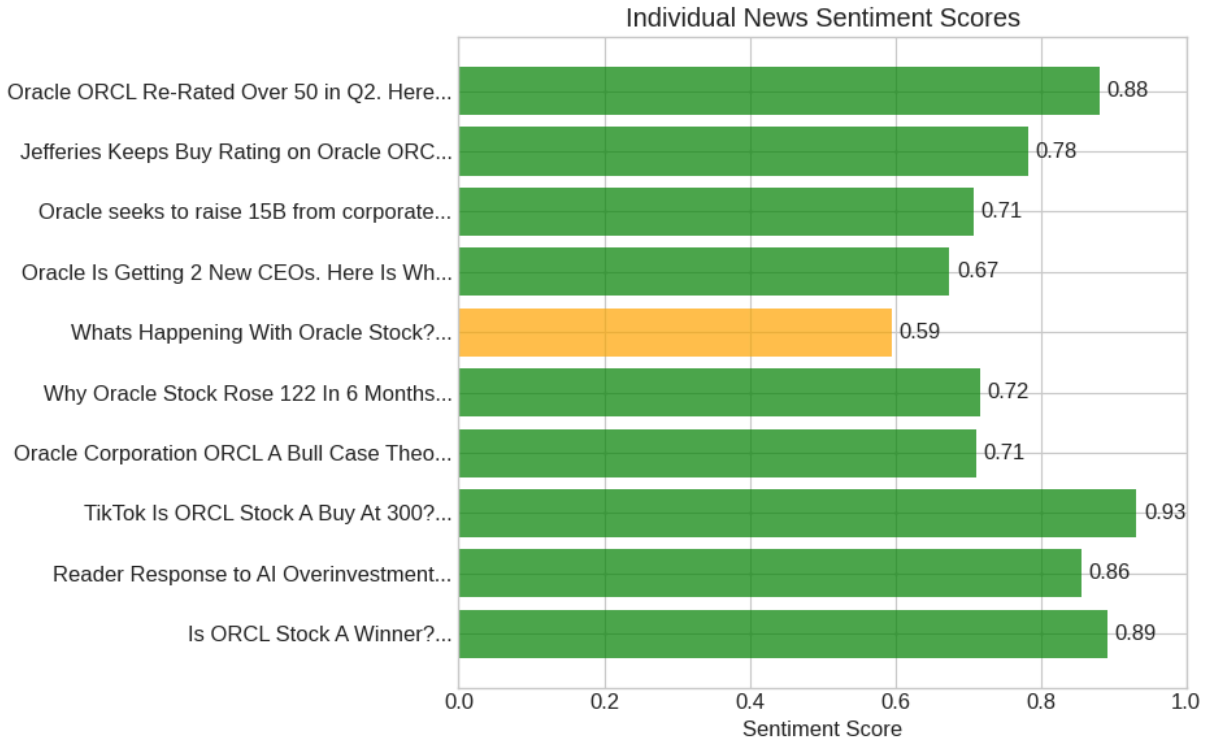

Price History - NVDA
Close Price
Price ($)
190
185
180
175
170
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15

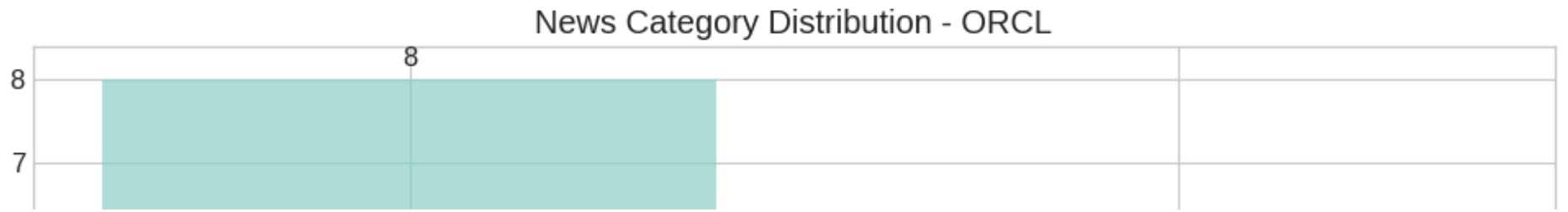

Relative Strength Index - NVDA
Overbought (70)
Oversold (30)
RSI
100
80
60
40



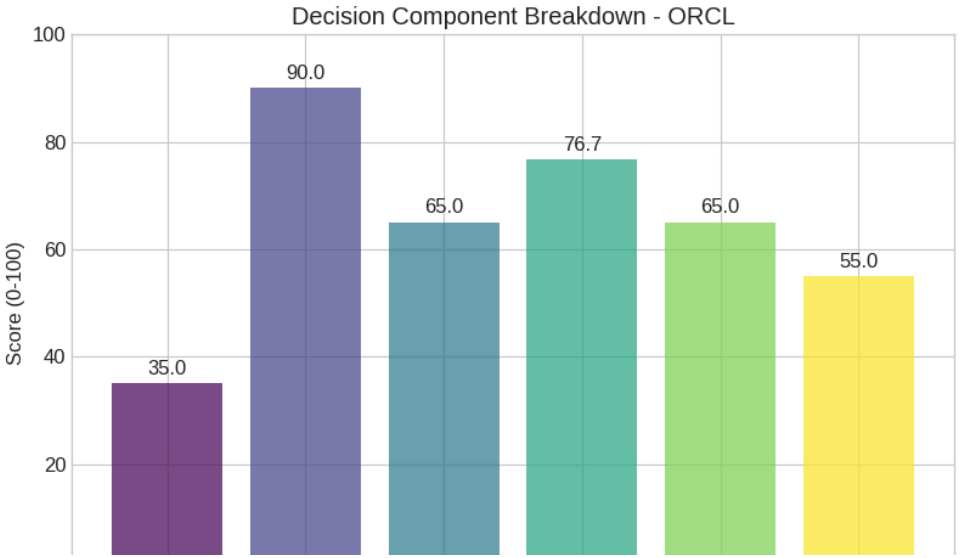
🔍 Analyzing ORCL...
[2025-10-19 13:53:45] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.734, Summary Quality: 0.833
✅ ORCL analysis completed
📊 Creating visualizations for ORCL...

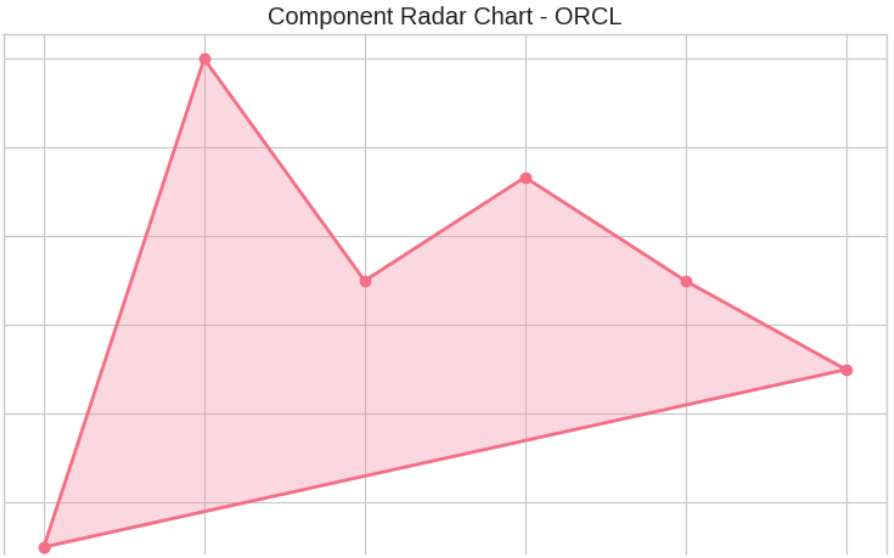


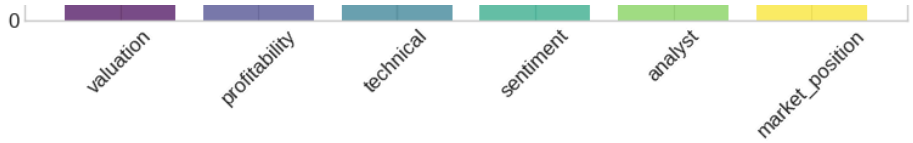




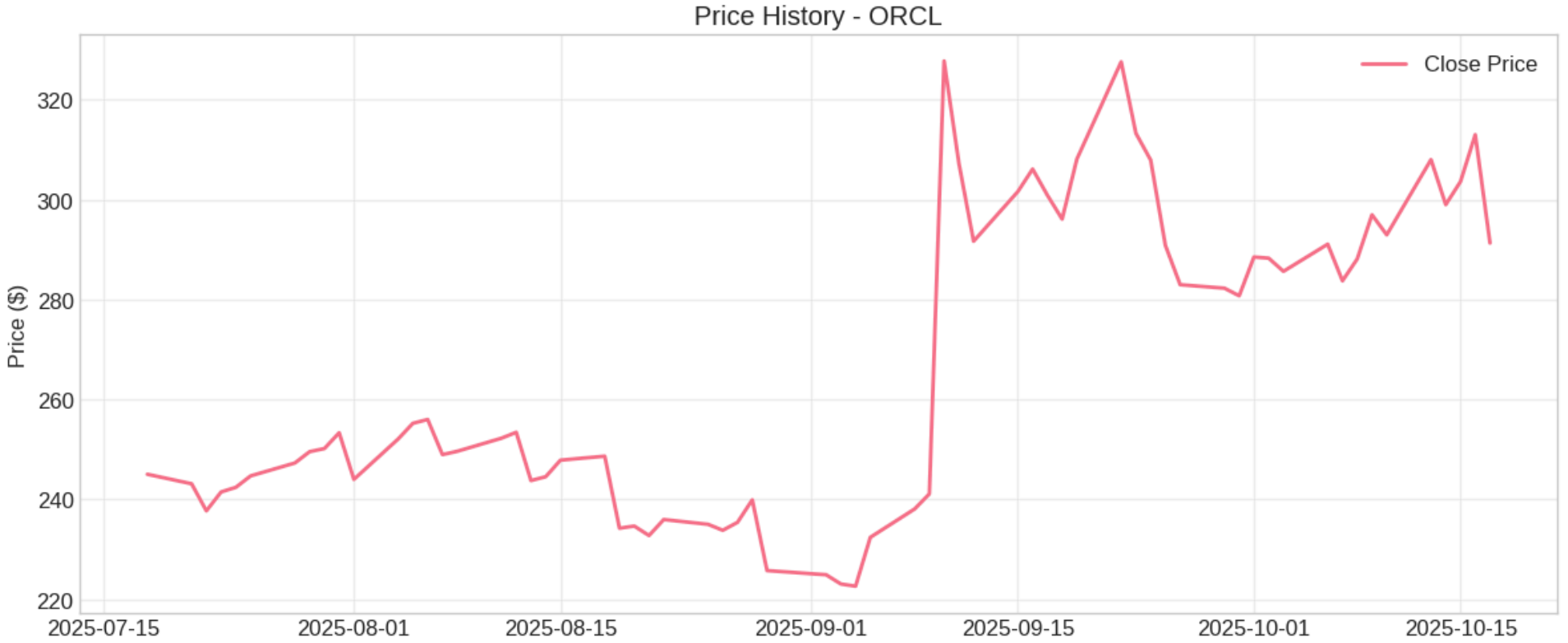

Number of Articles
0
1
2
3
4
5
6
2
company_news
earnings
News Categories

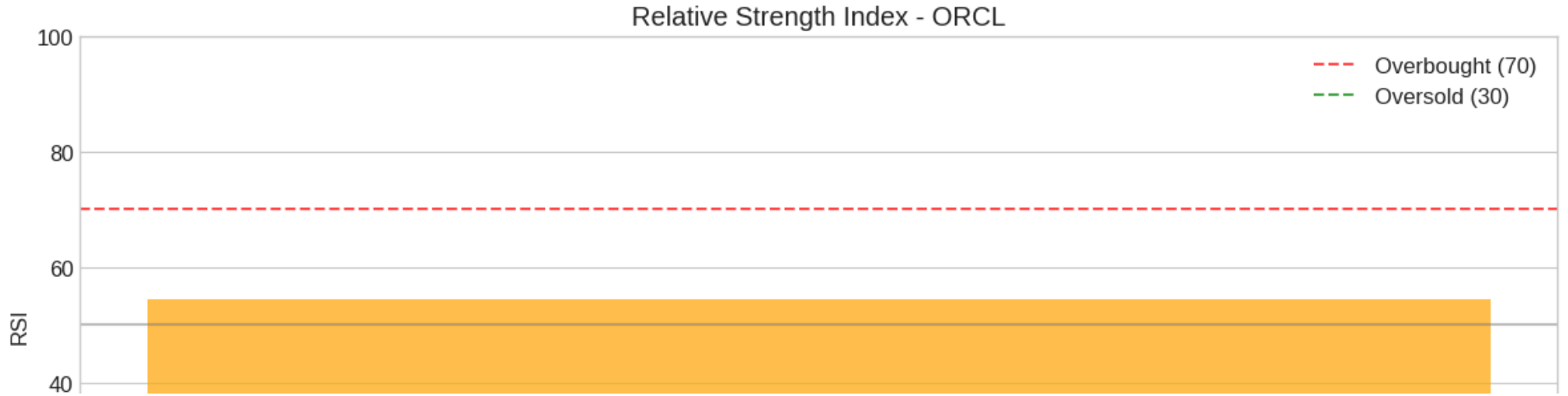

Decision Component Breakdown - ORCL
Score (0-100)
20
40
60
80
100
35.0
90.0
65.0
76.7
65.0
55.0

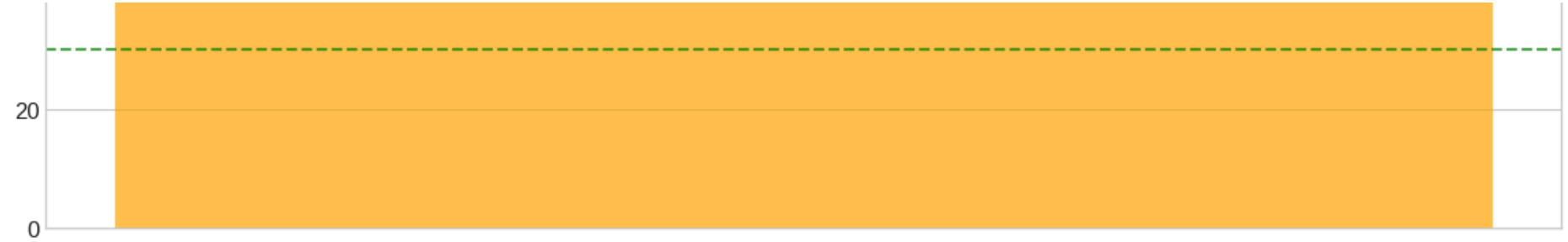

Component Radar Chart - ORCL


0
valuation
profitability
technical
sentiment
analyst
market_position

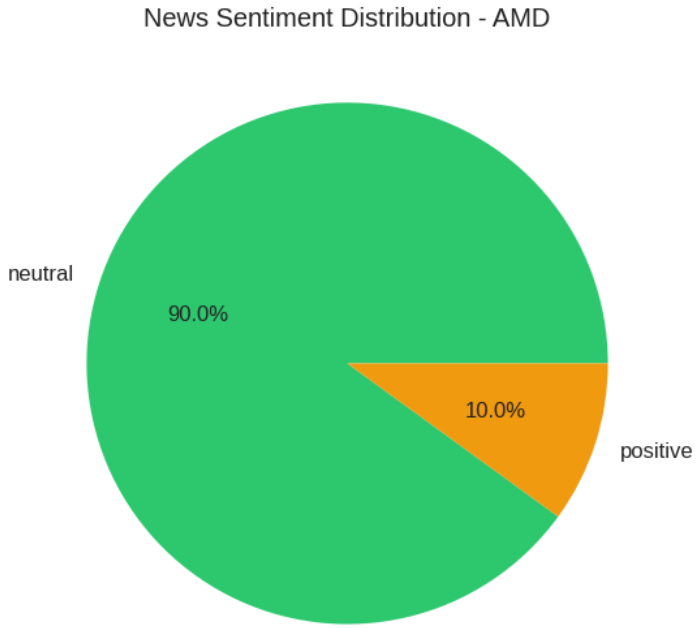

valuation
profitability
technical
sentiment
analyst
market_position

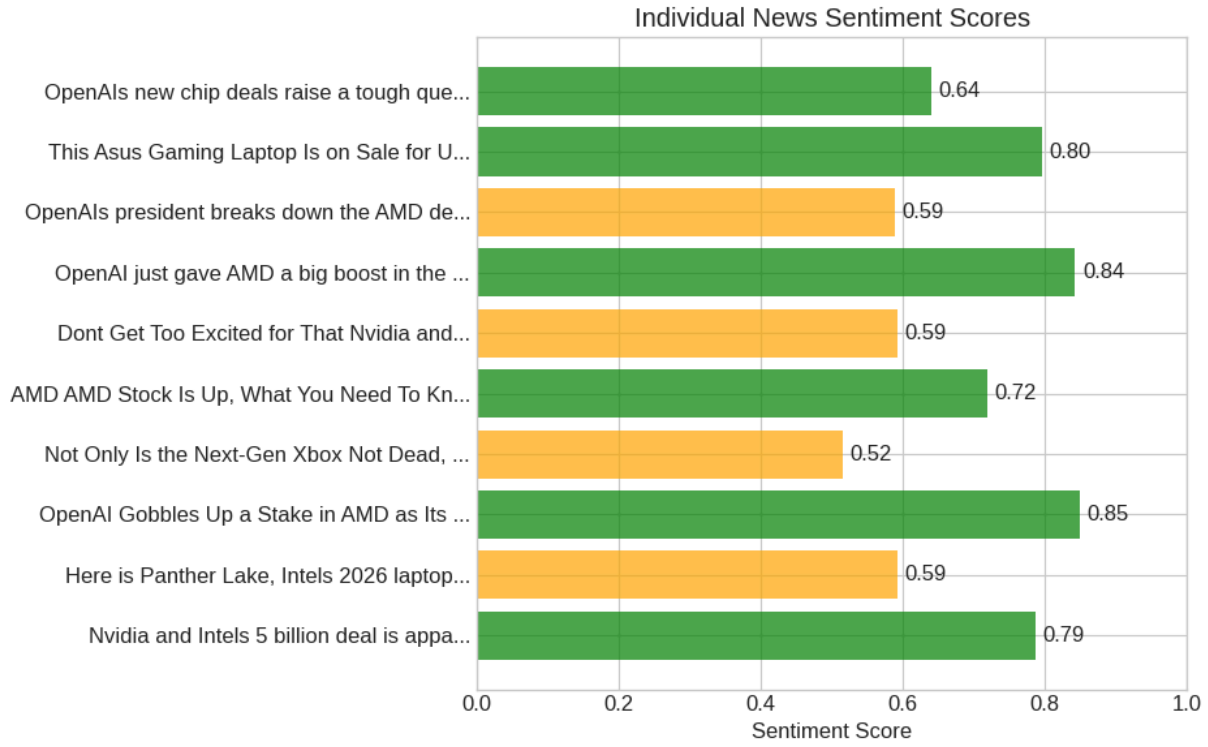

Price History - ORCL
Close Price
Price ($)
320
300
280
260
240
220
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15

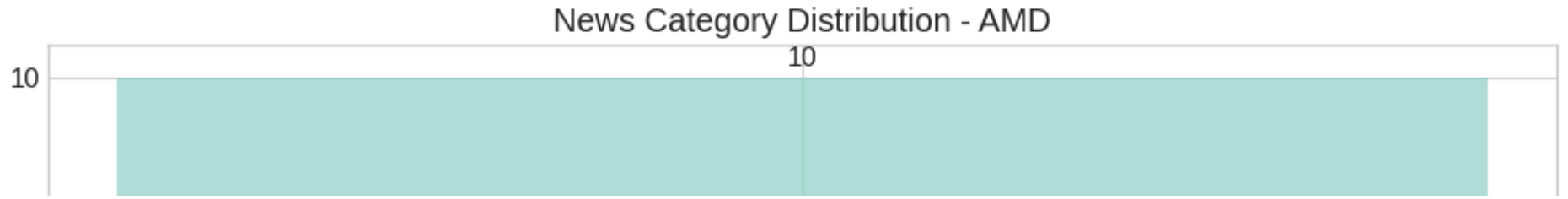

Relative Strength Index - ORCL
Overbought (70)
Oversold (30)
RSI
100
80
60
40

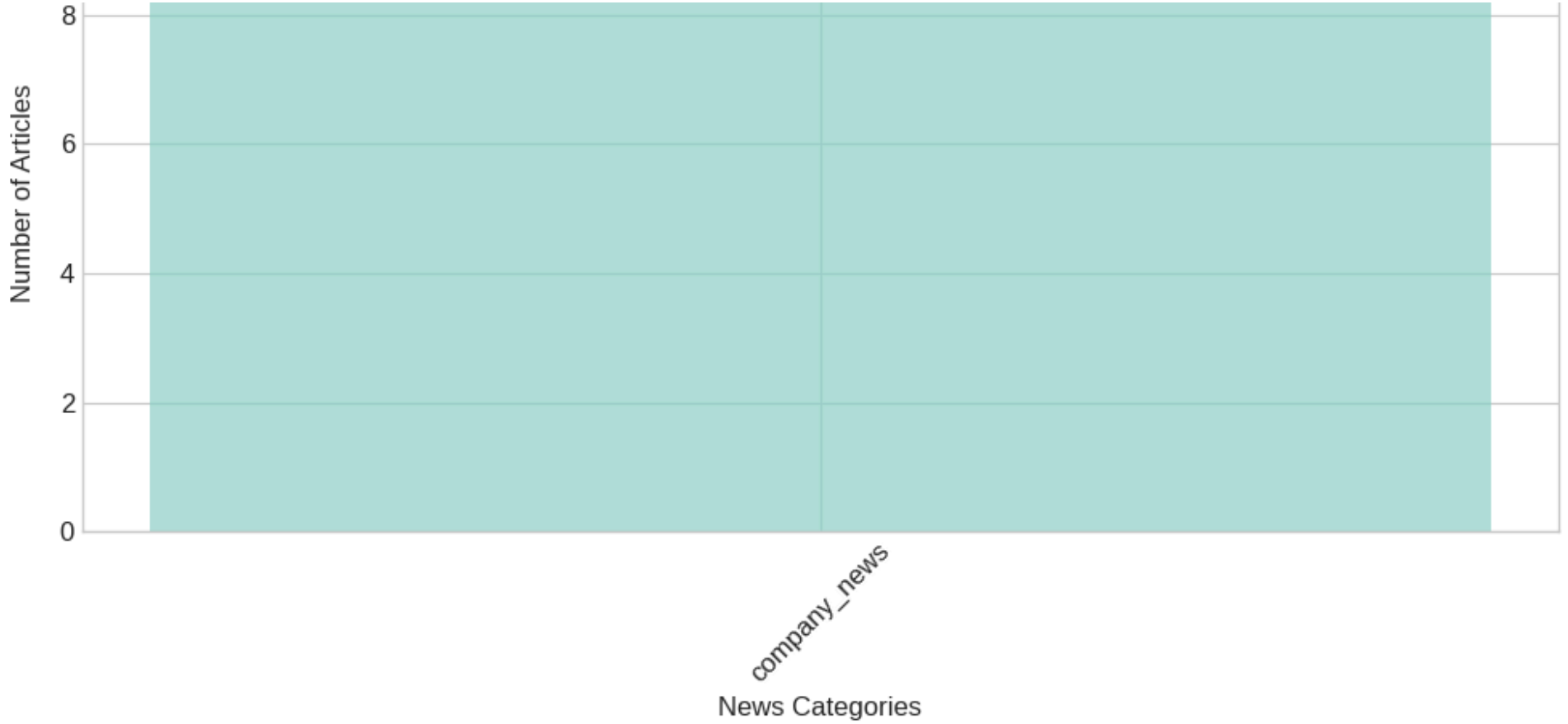


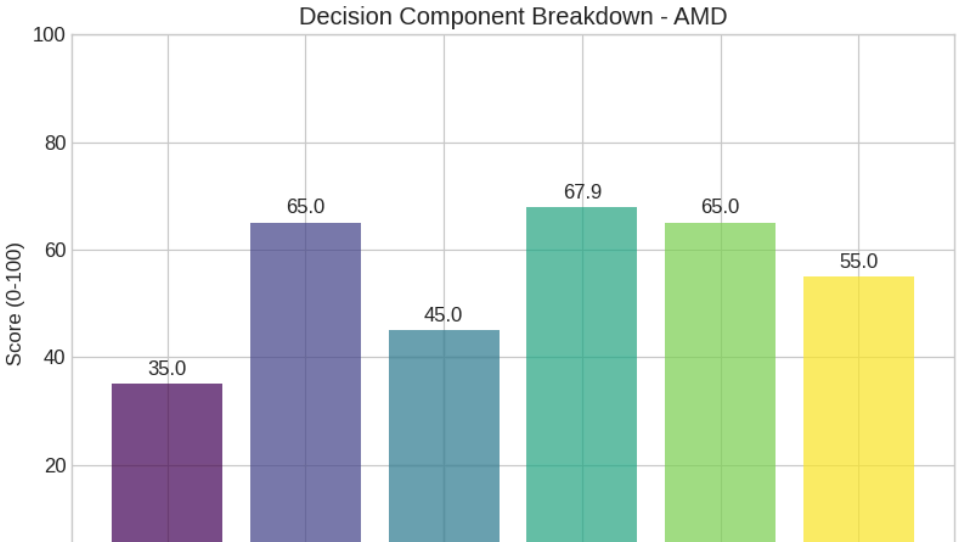
🔍 Analyzing AMD...
[2025-10-19 13:53:51] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.728, Summary Quality: 0.835
✅ AMD analysis completed
📊 Creating visualizations for AMD...

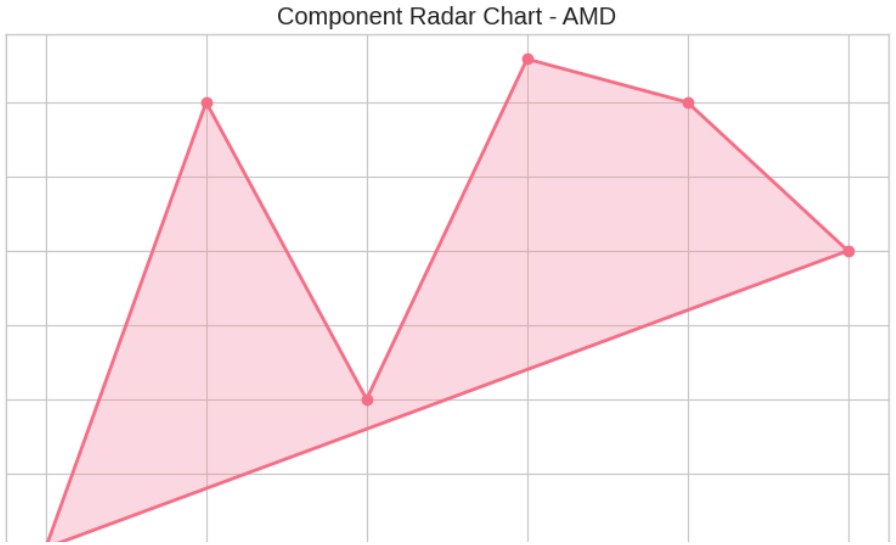


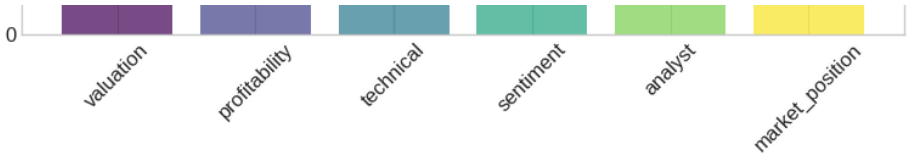




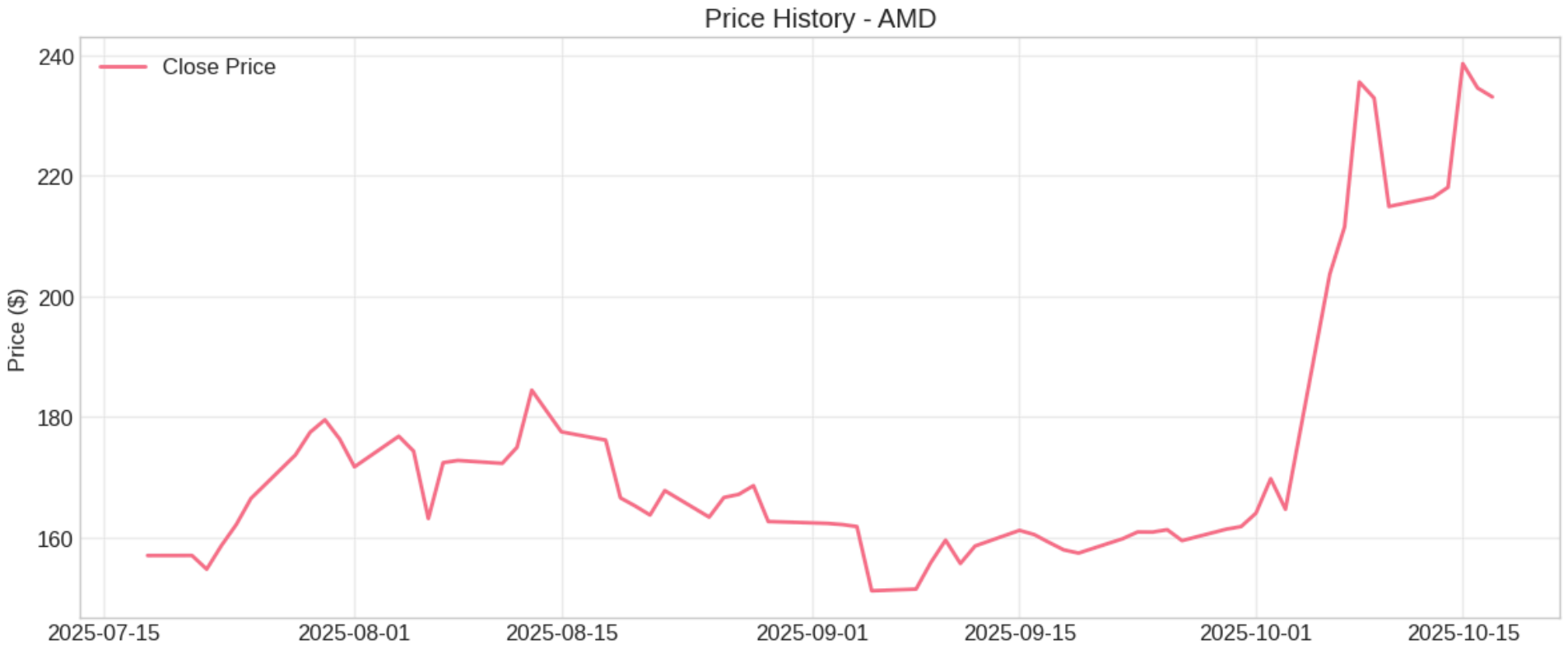

Number of Articles
0
2
4
6
8
company_news
News Categories

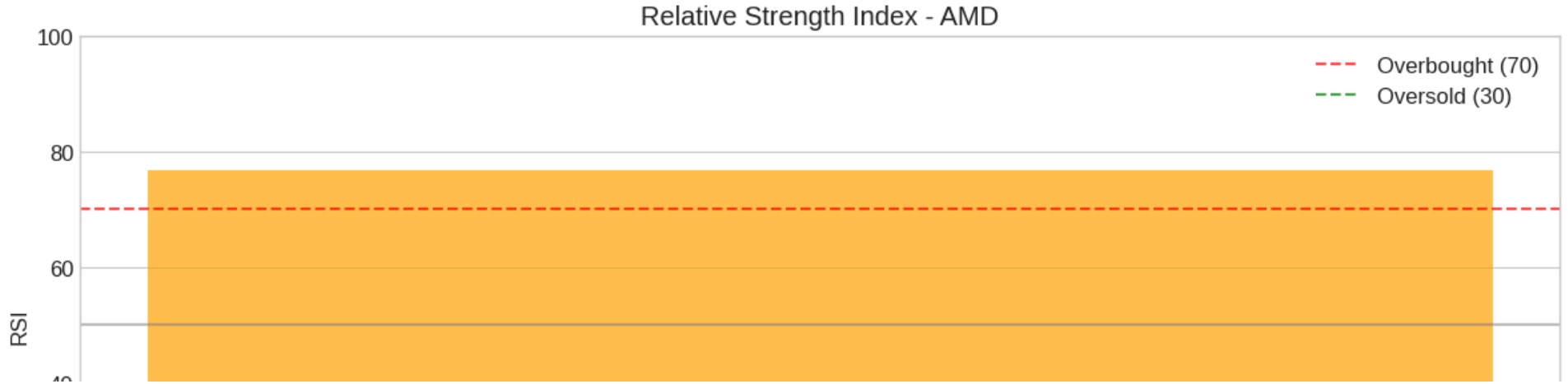

Decision Component Breakdown - AMD
Score (0-100)
20
40
60
80
100
35.0
65.0
45.0
67.9
65.0
55.0

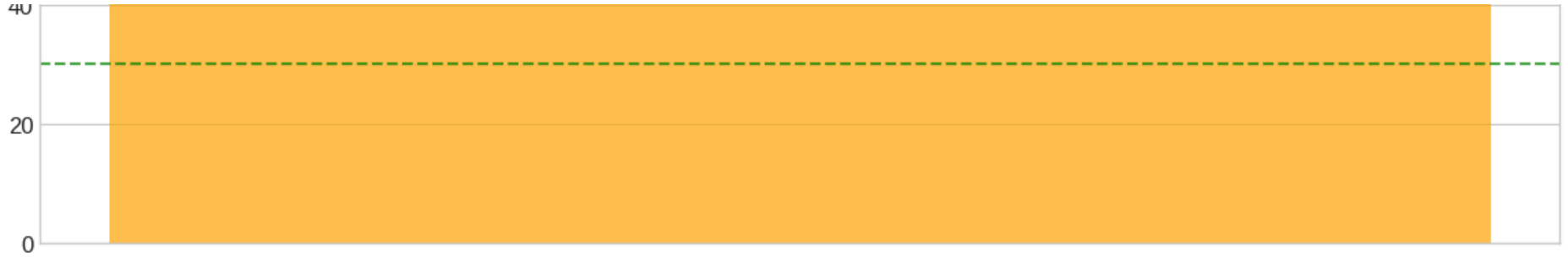

Component Radar Chart - AMD


0
valuation
profitability
technical
sentiment
analyst
market_position

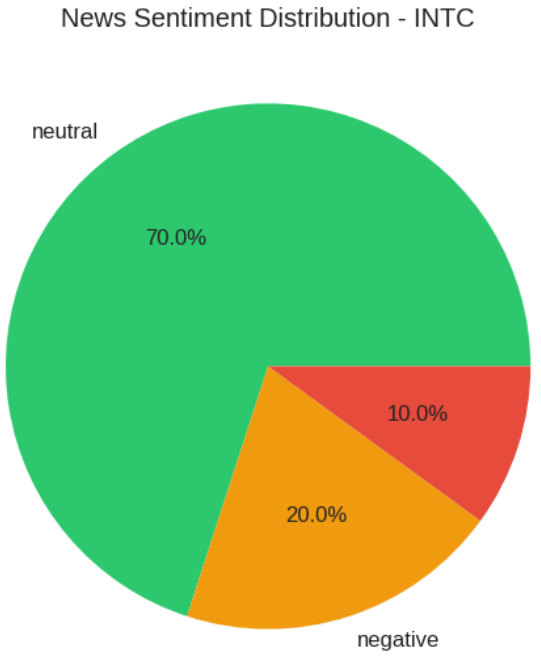

valuation
profitability
technical
sentiment
analyst
market_position

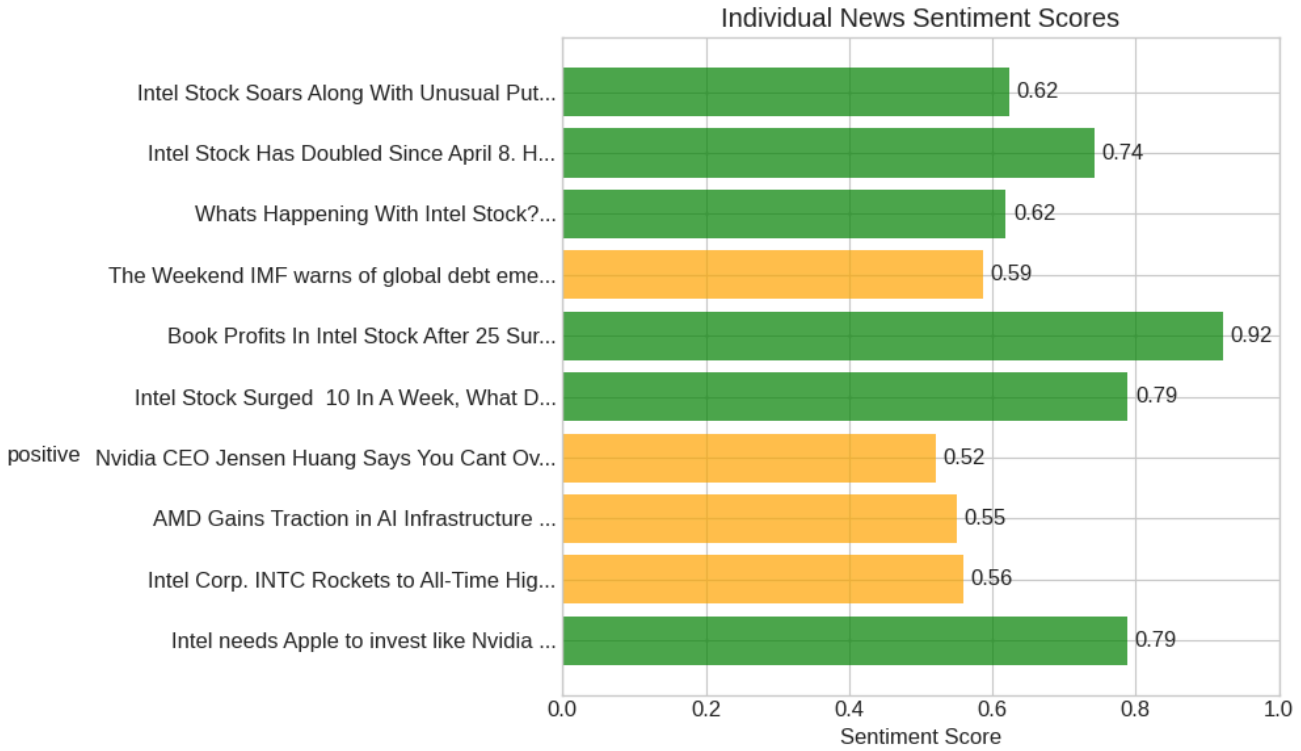

Price History - AMD
Close Price
Price ($)
240
220
200
180
160
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15

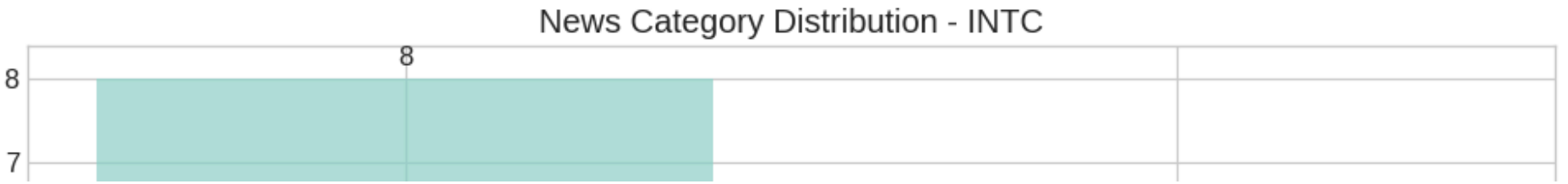

Relative Strength Index - AMD
Overbought (70)
Oversold (30)
RSI
100
80
60

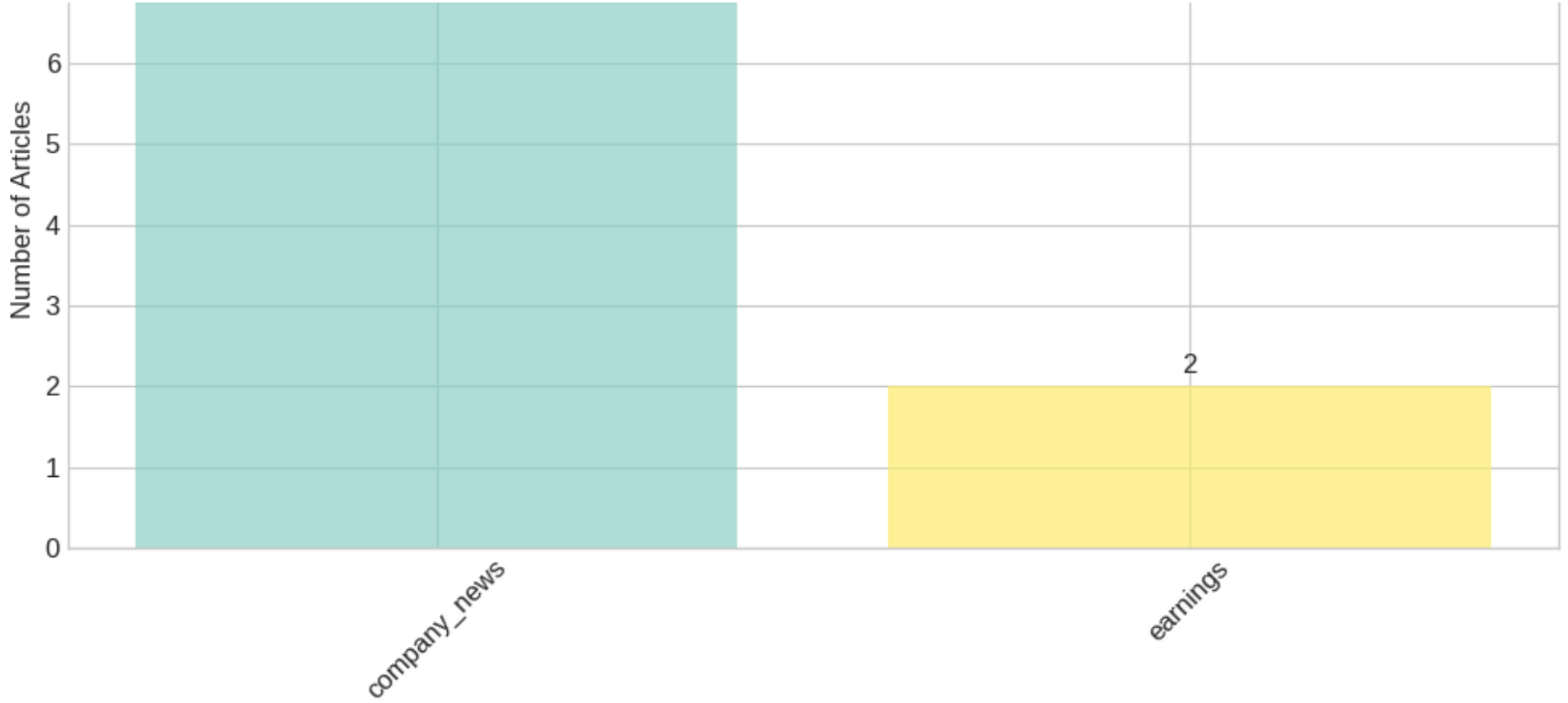


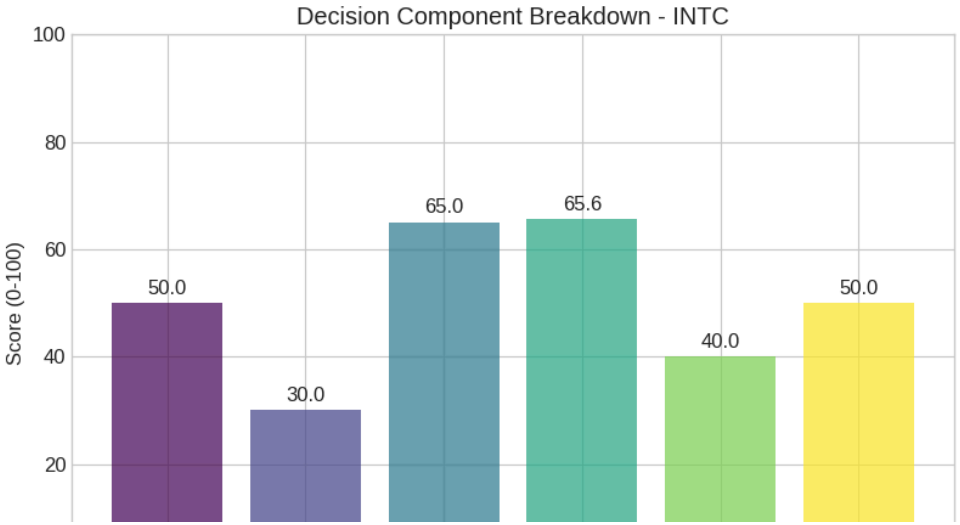
🔍 Analyzing INTC...
[2025-10-19 13:53:56] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.721, Sentiment Quality: 0.828
✅ INTC analysis completed
📊 Creating visualizations for INTC...

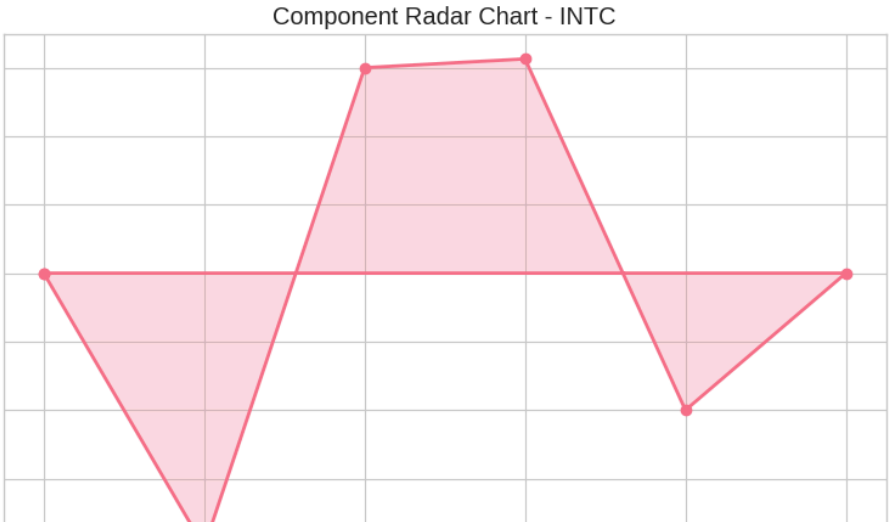


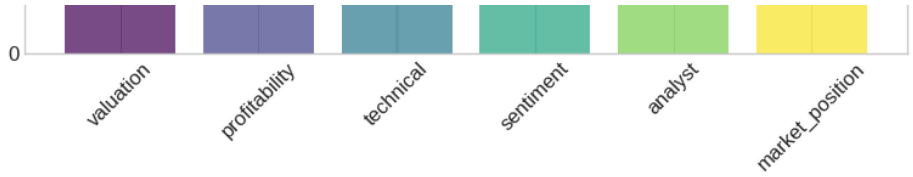


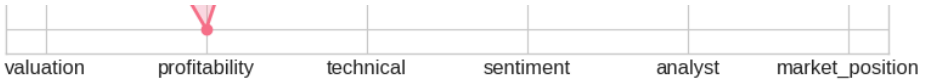


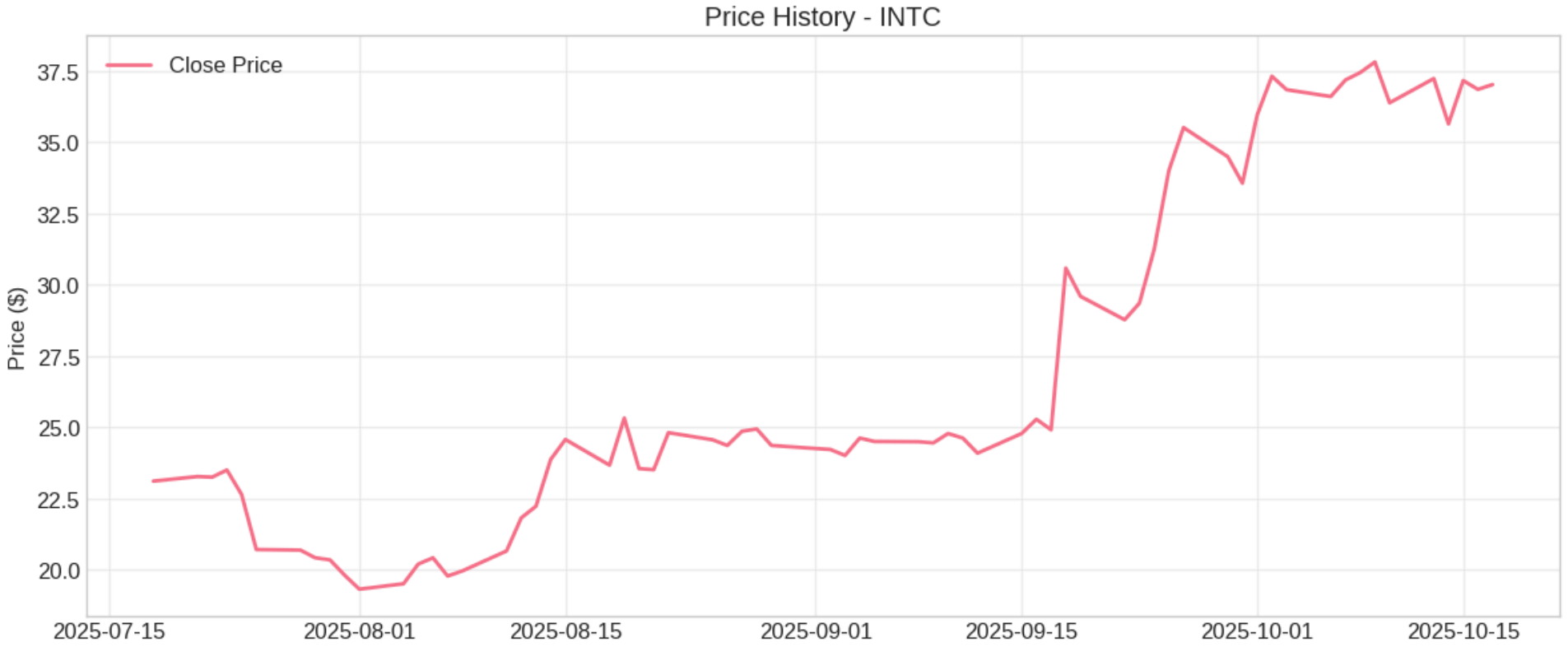

Number of Articles
0
1
2
3
4
5
6
2
company_news
earnings
News Categories

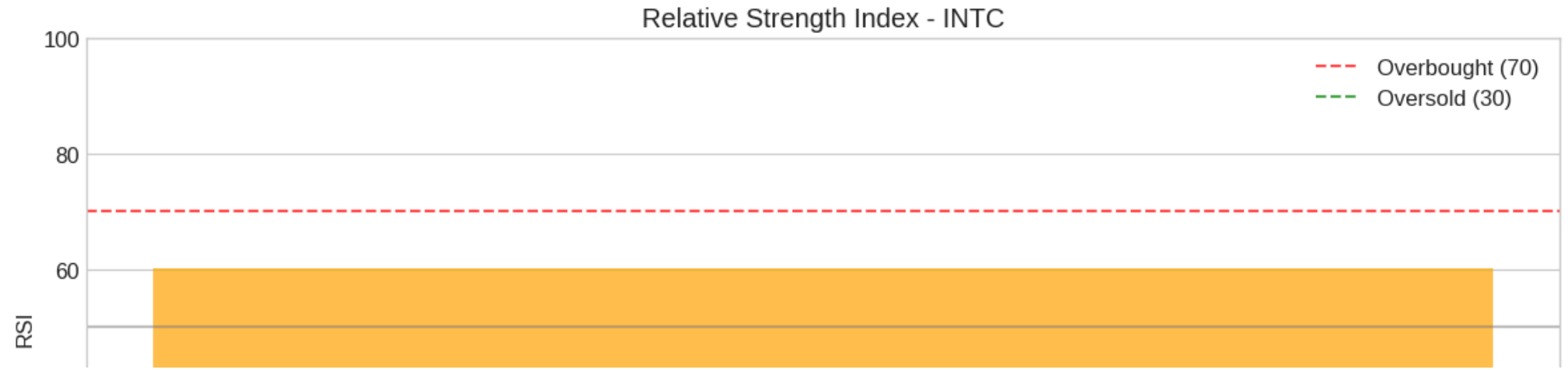

Decision Component Breakdown - INTC
Score (0-100)
20
40
60
80
100
50.0
30.0
65.0
65.6
40.0
50.0

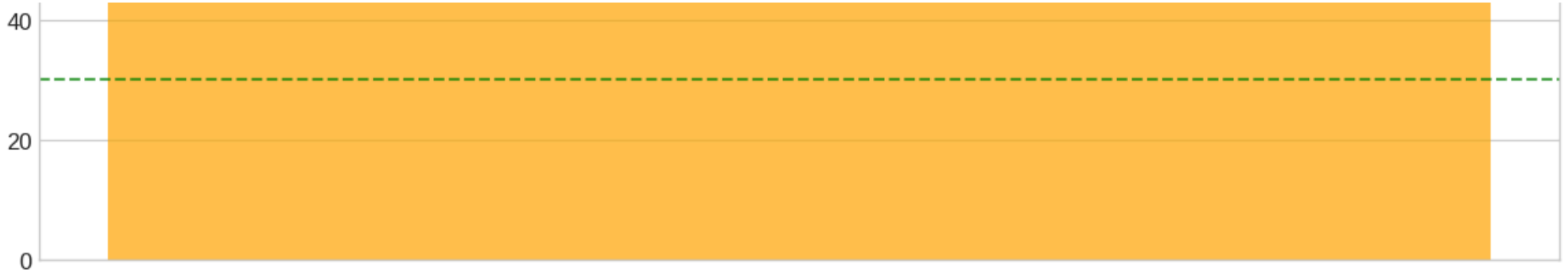

Component Radar Chart - INTC


0
valuation
profitability
technical
sentiment
analyst
market_position

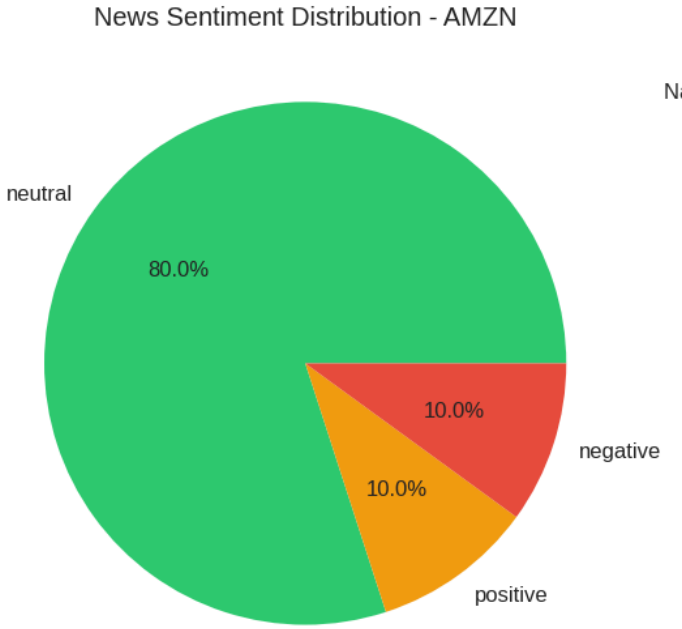

valuation
profitability
technical
sentiment
analyst
market_position

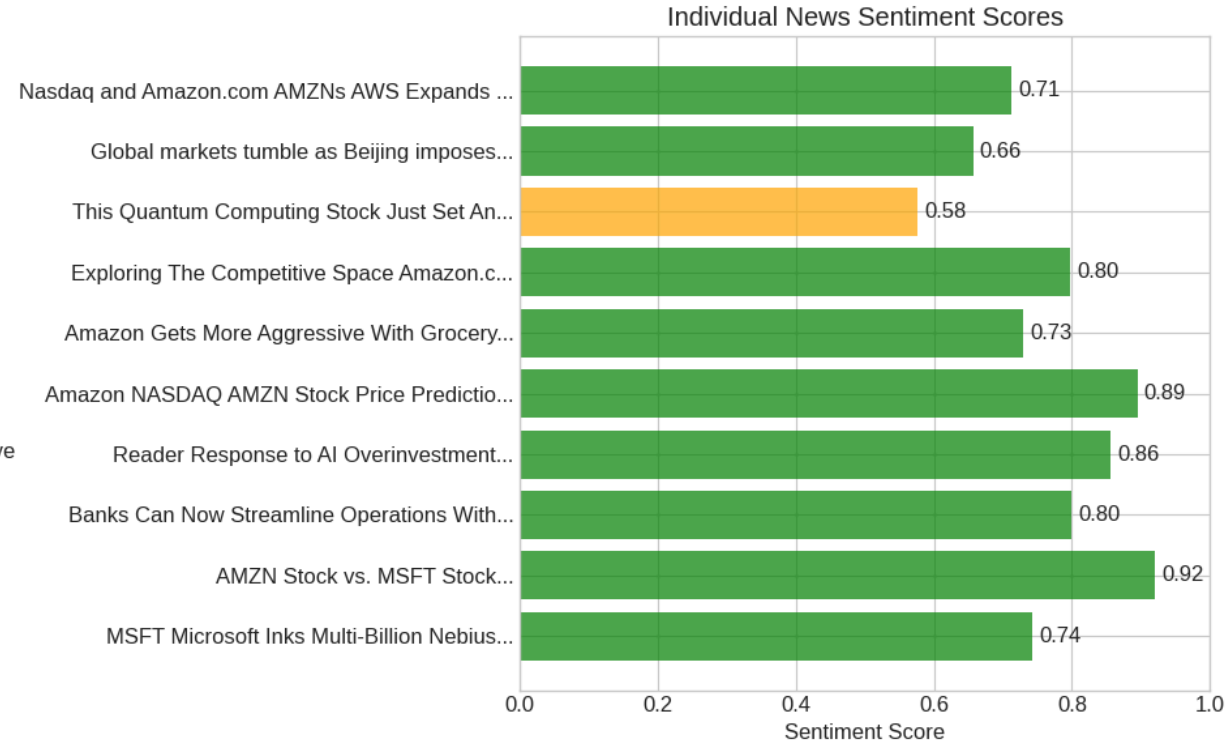

Price History - INTC
Close Price
Price ($)
37.5
35.0
32.5
30.0
27.5
25.0
22.5
20.0
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15

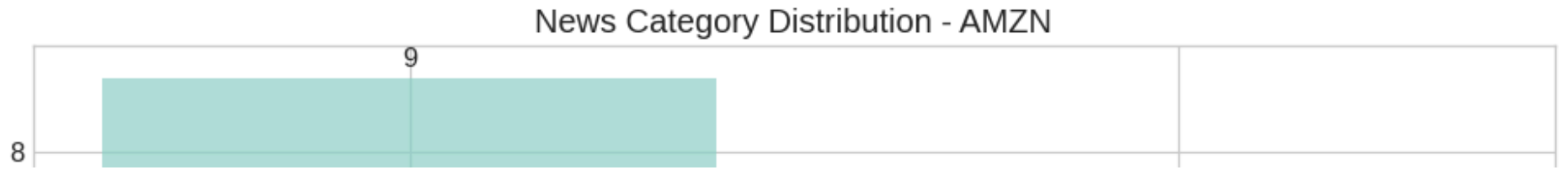

Relative Strength Index - INTC
Overbought (70)
Oversold (30)
RSI
100
80
60

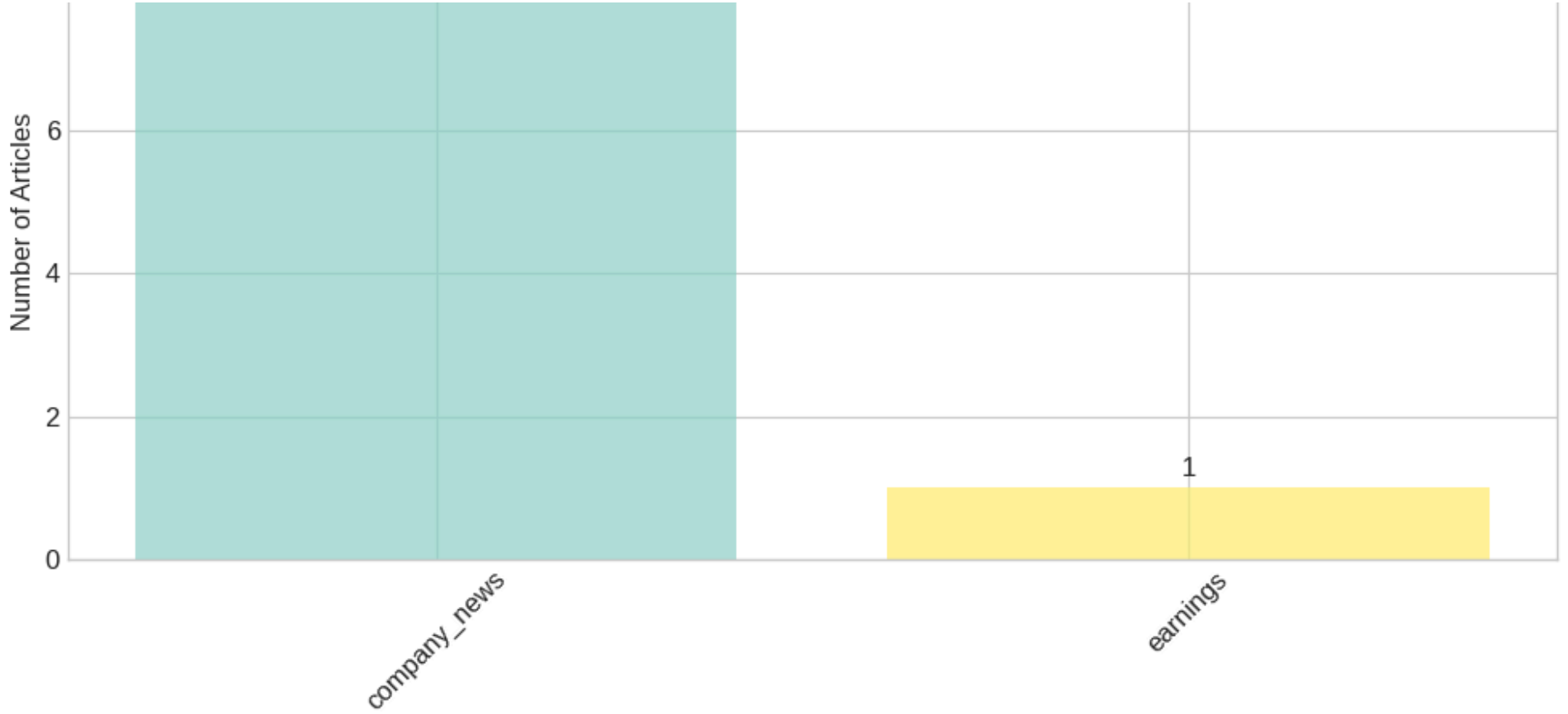


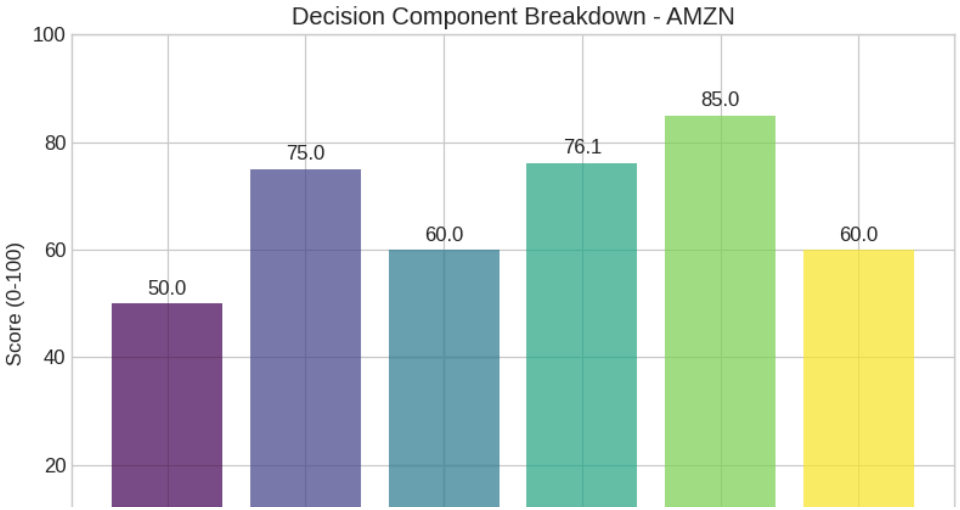
🔍 Analyzing AMZN...
[2025-10-19 13:54:03] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.726, Summary Quality: 0.829
✅ AMZN analysis completed
📊 Creating visualizations for AMZN...

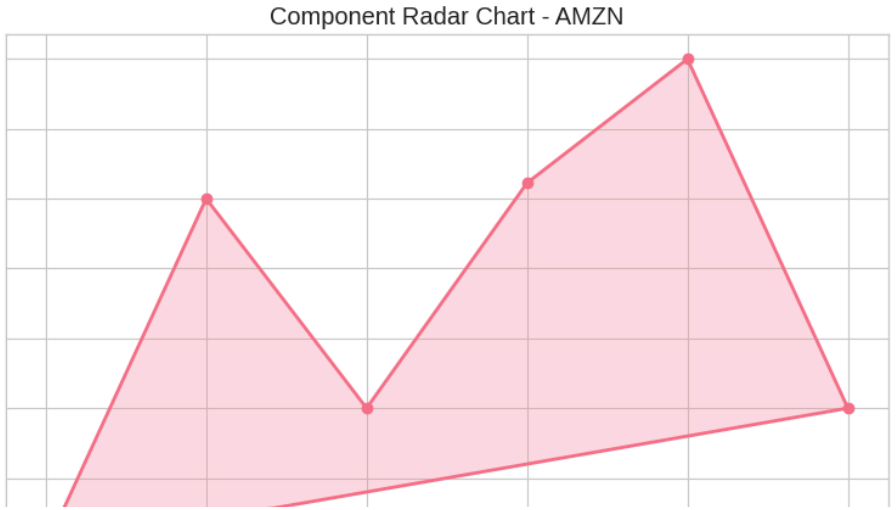


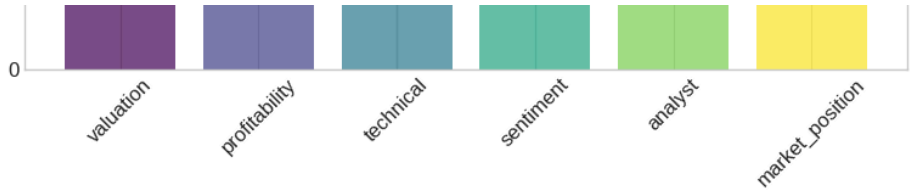


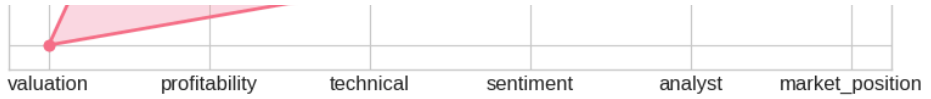


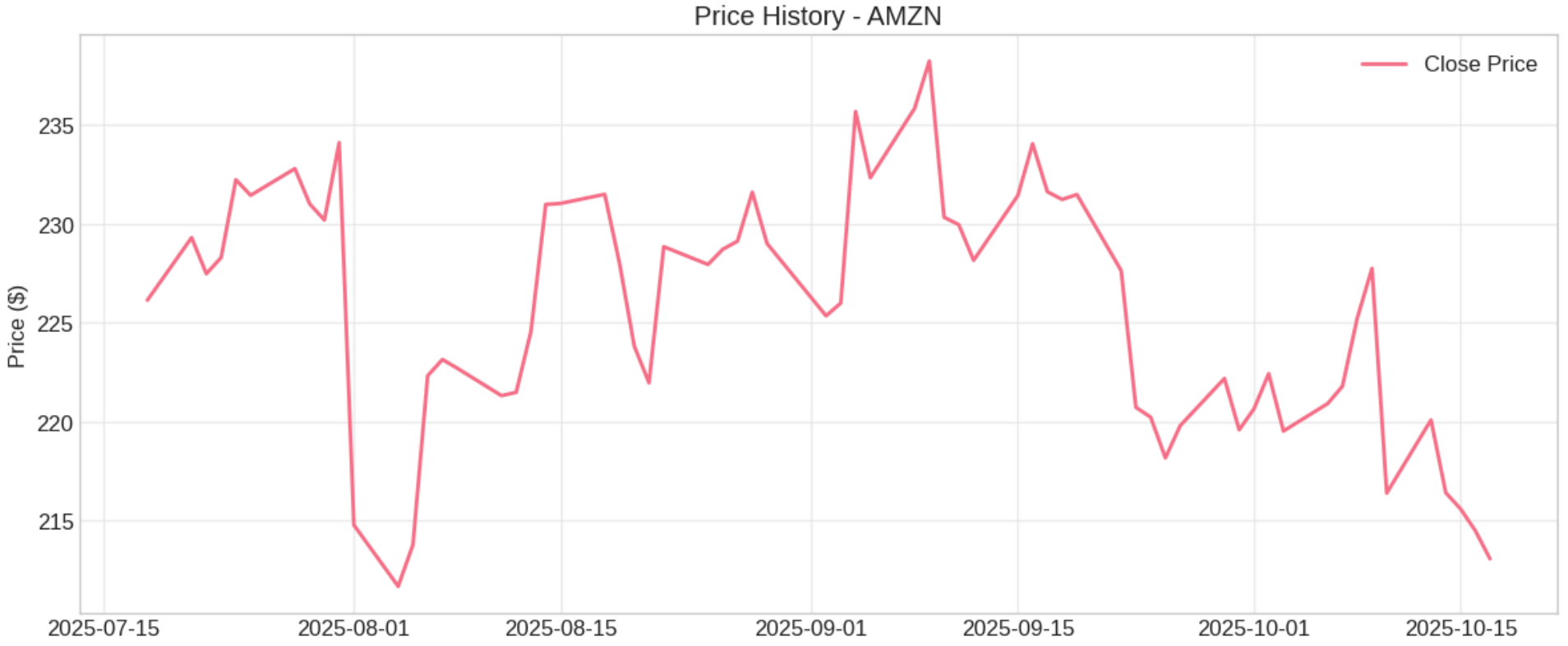

Number of Articles
0
2
4
6
1
company_news
earnings
News Categories


Decision Component Breakdown - AMZN
Score (0-100)
20
40
60
80
100
50.0
75.0
60.0
76.1
85.0
60.0

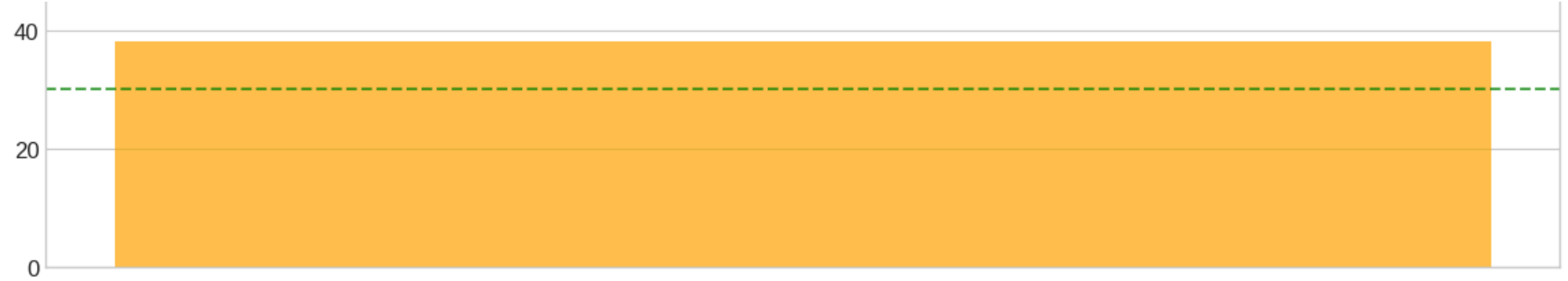

Component Radar Chart - AMZN


0
valuation
profitability
technical
sentiment
analyst
market_position

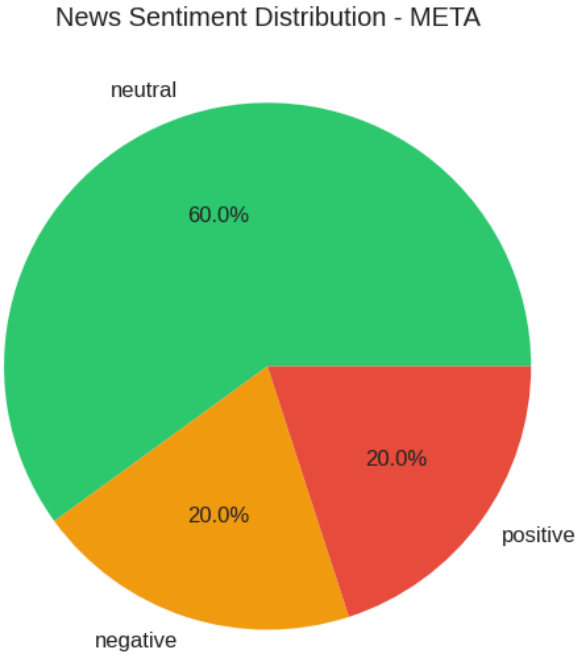

valuation
profitability
technical
sentiment
analyst
market_position

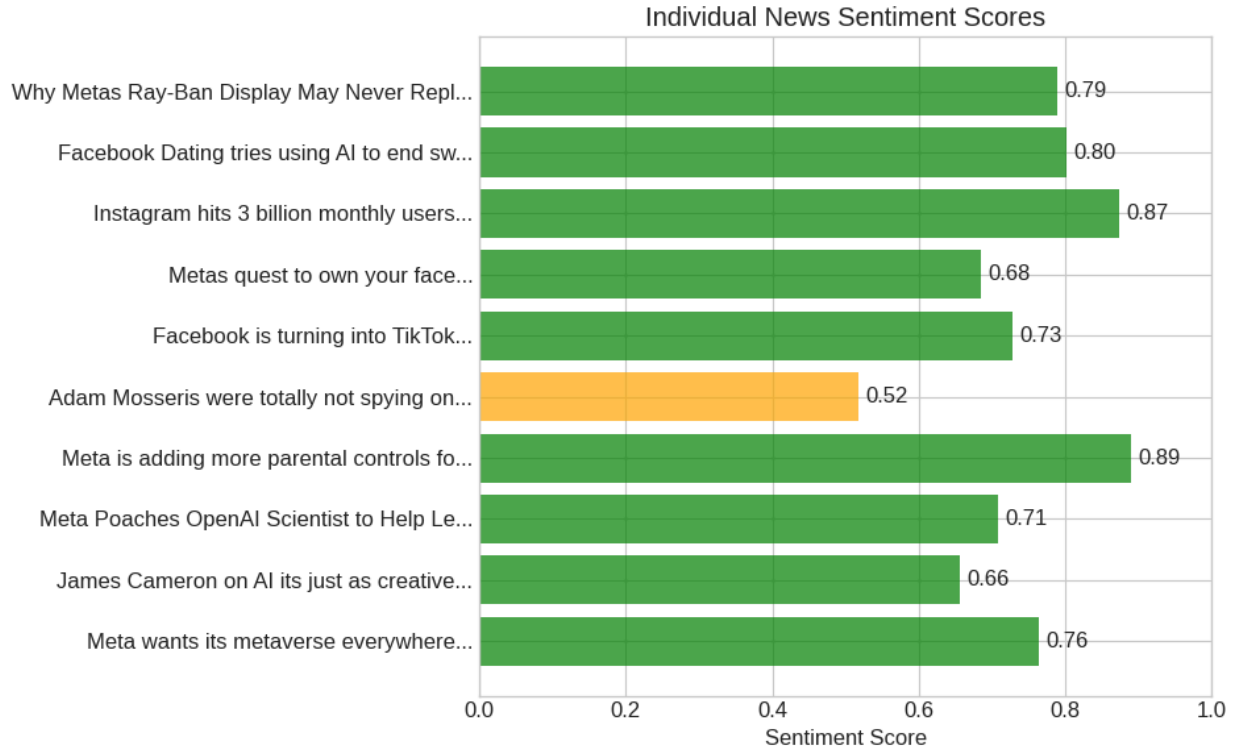

Price History - AMZN
Close Price
Price ($)
235
230
225
220
215
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15

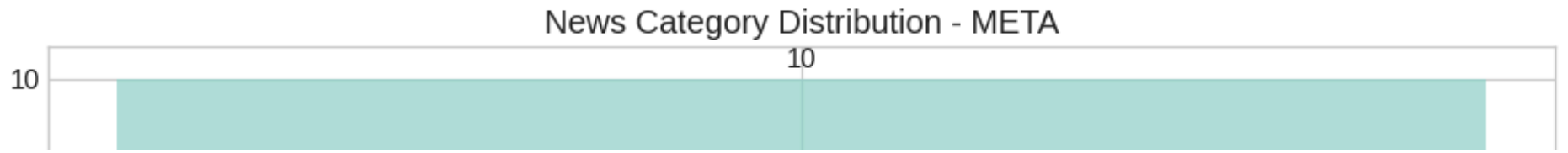

Relative Strength Index - AMZN
Overbought (70)
Oversold (30)
RSI
100
80
60

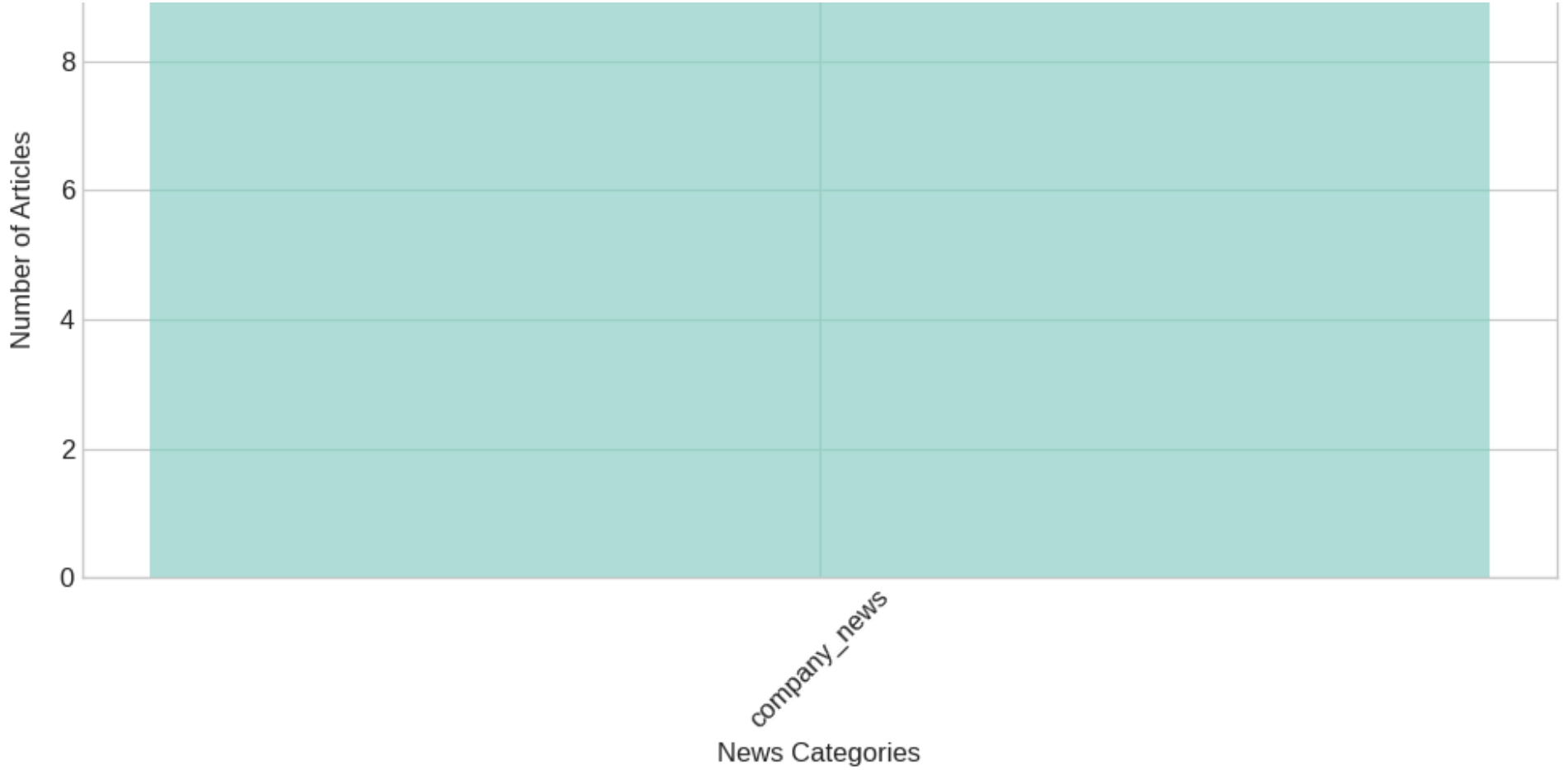


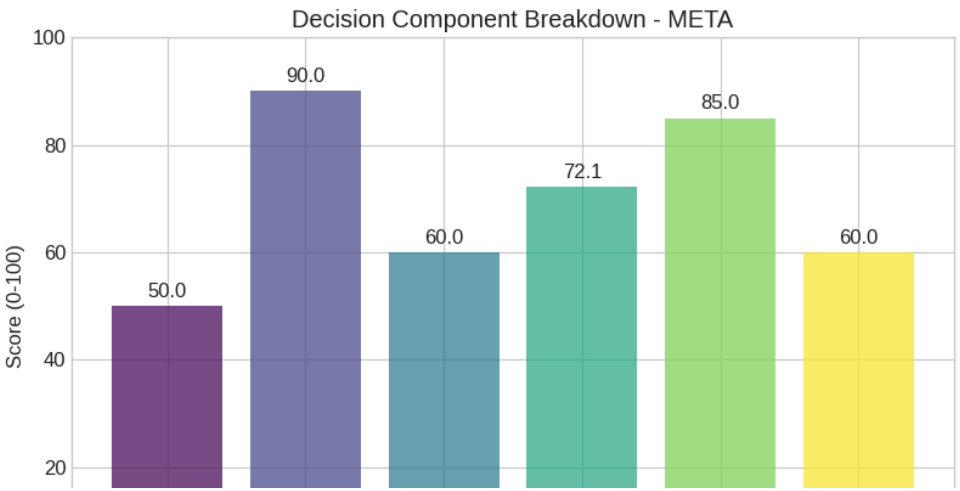
🔍 Analyzing META...
[2025-10-19 13:54:08] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.726, Summary Quality: 0.832
✅ META analysis completed
📊 Creating visualizations for META...

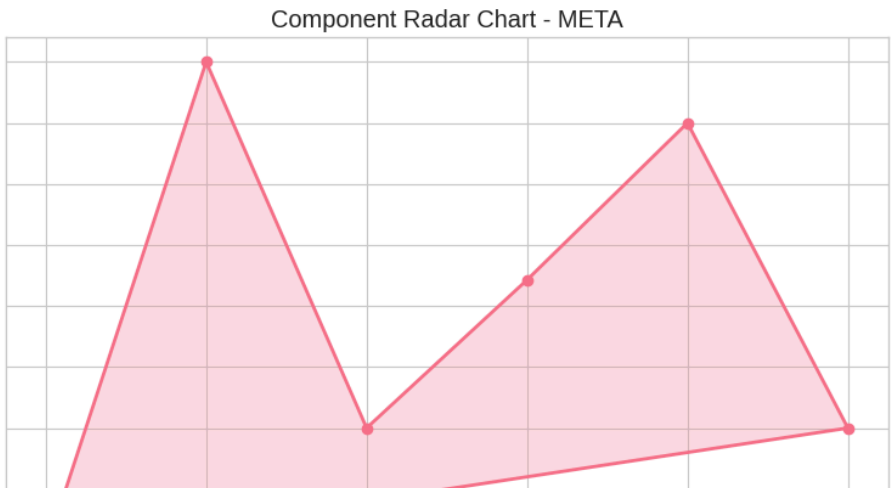


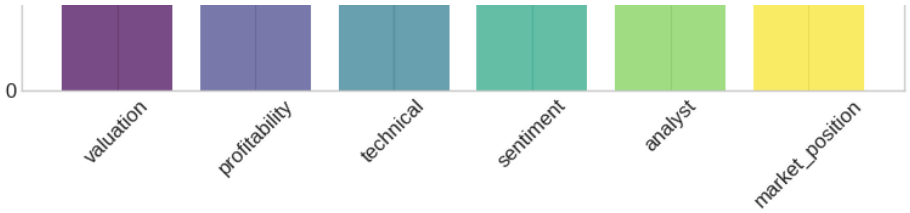




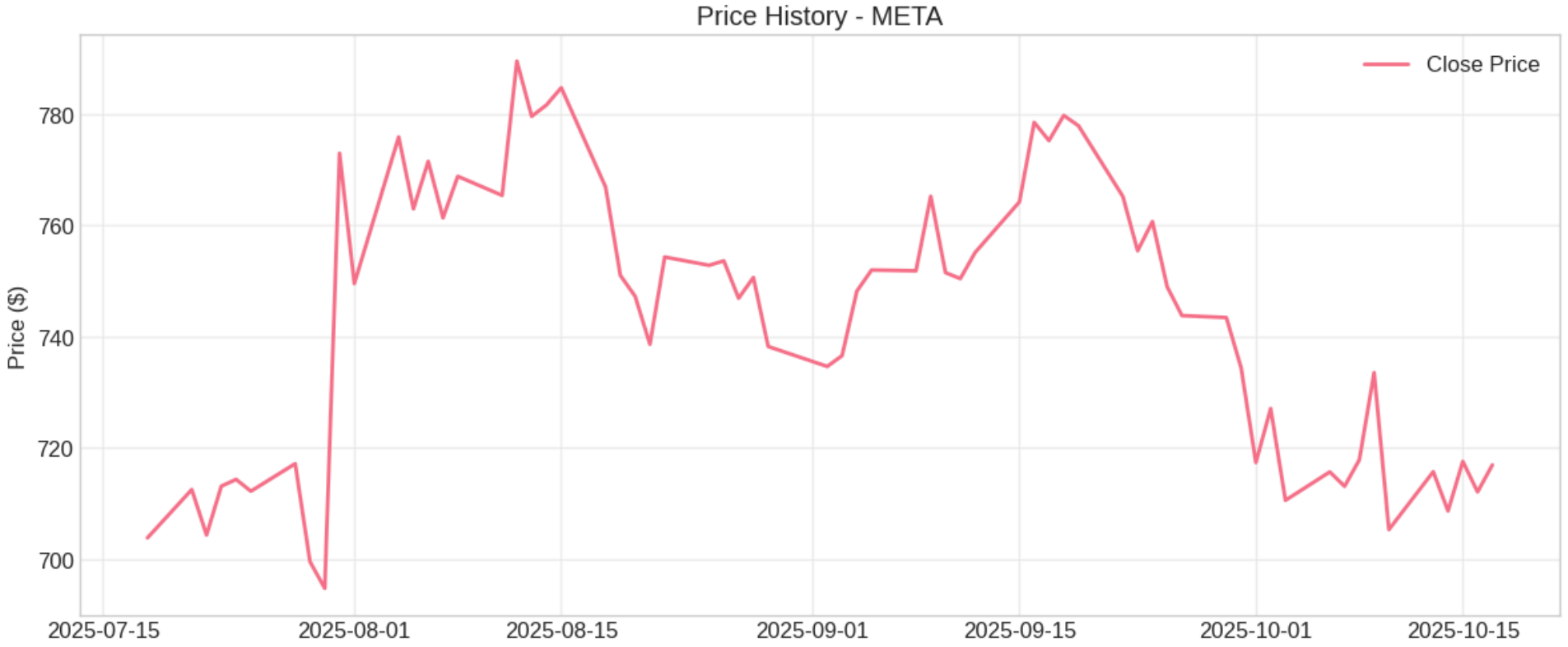

Number of Articles
0
2
4
6
8
company_news
News Categories


Decision Component Breakdown - META
Score (0-100)
20
40
60
80
100
50.0
90.0
60.0
72.1
85.0
60.0

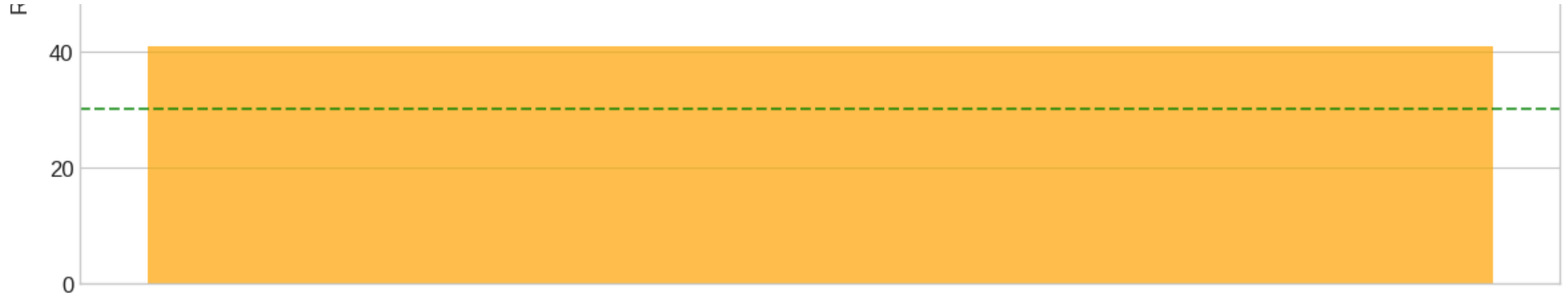

Component Radar Chart - META


0
valuation
profitability
technical
sentiment
analyst
market_position

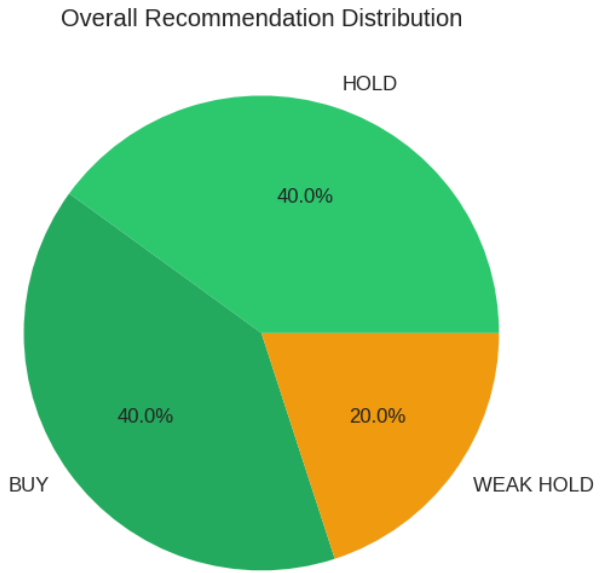

valuation
profitability
technical
sentiment
analyst
market_position

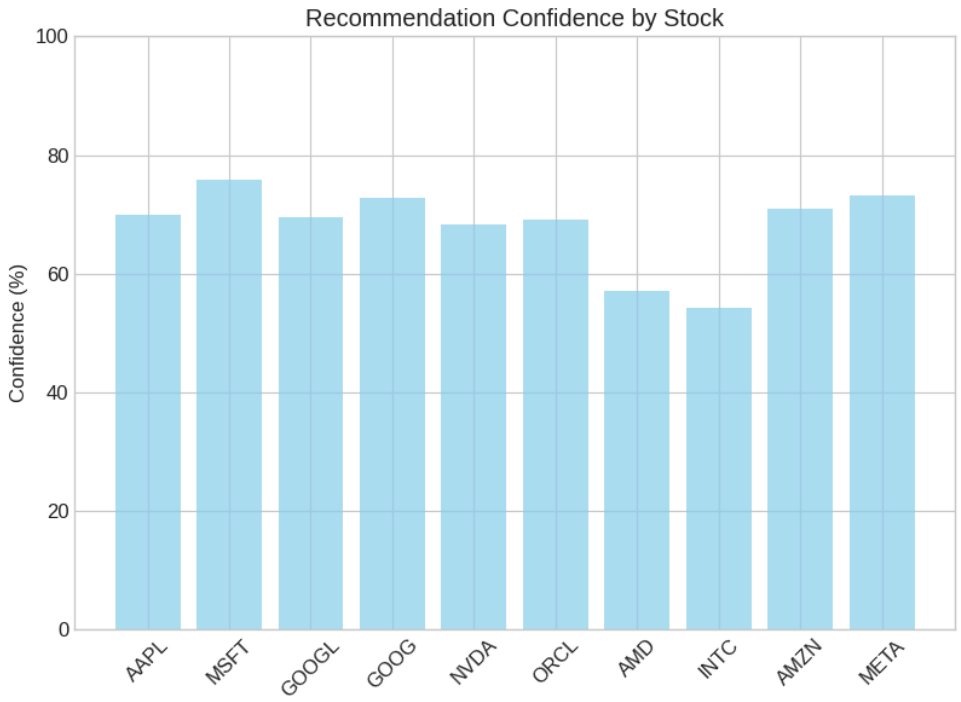

Price History - META
Close Price
Price ($)
780
760
740
720
700
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15

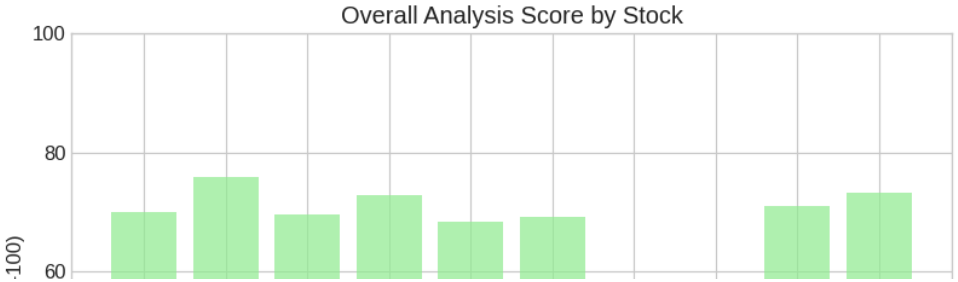

Relative Strength Index - META
Overbought (70)
Oversold (30)
100
80
60

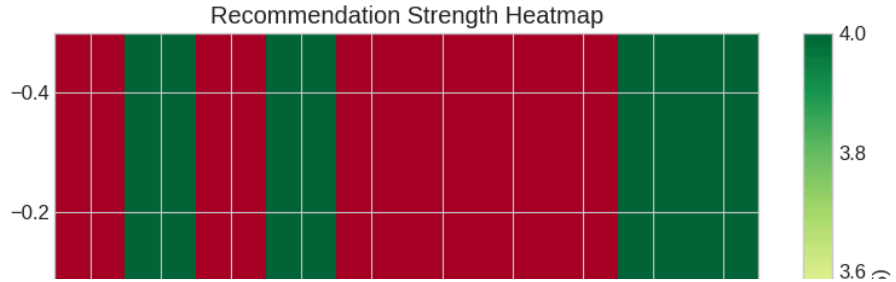


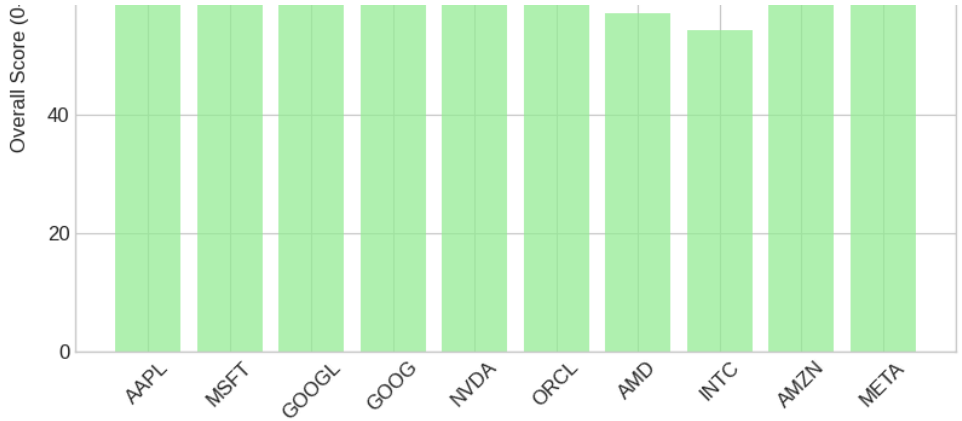
📈 Creating summary visualizations...

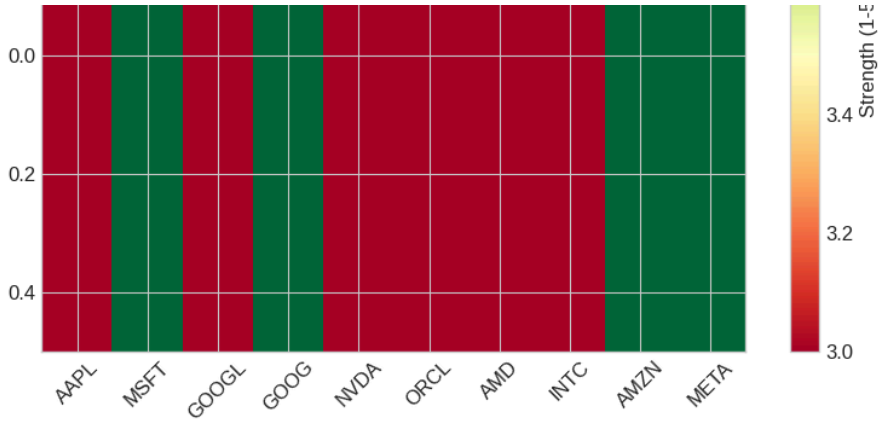












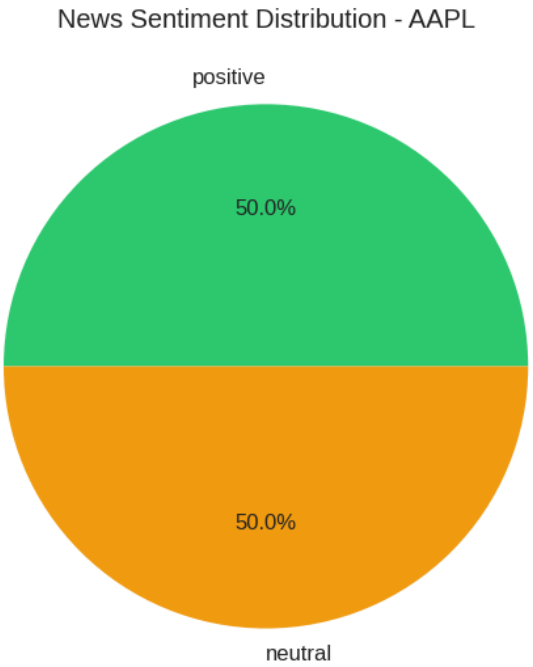
COMPREHENSIVE ANALYSIS RESULTS

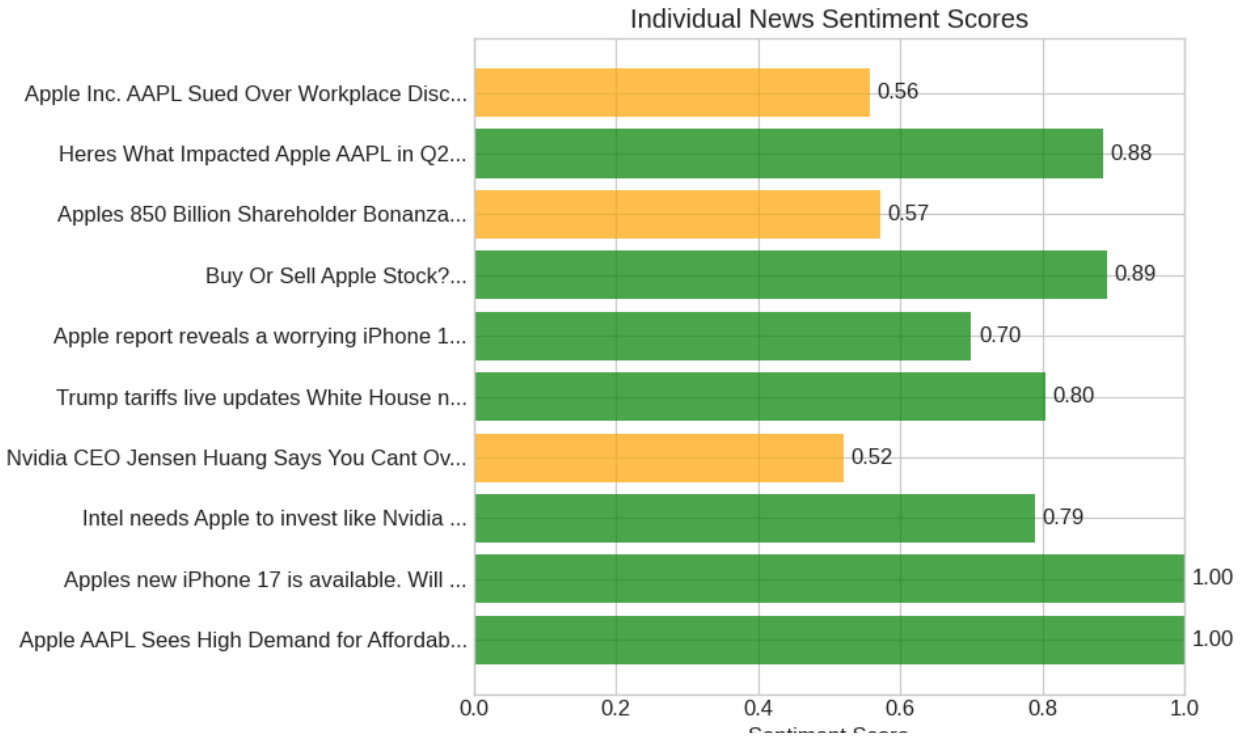
🔍 Analyzing AAPL...

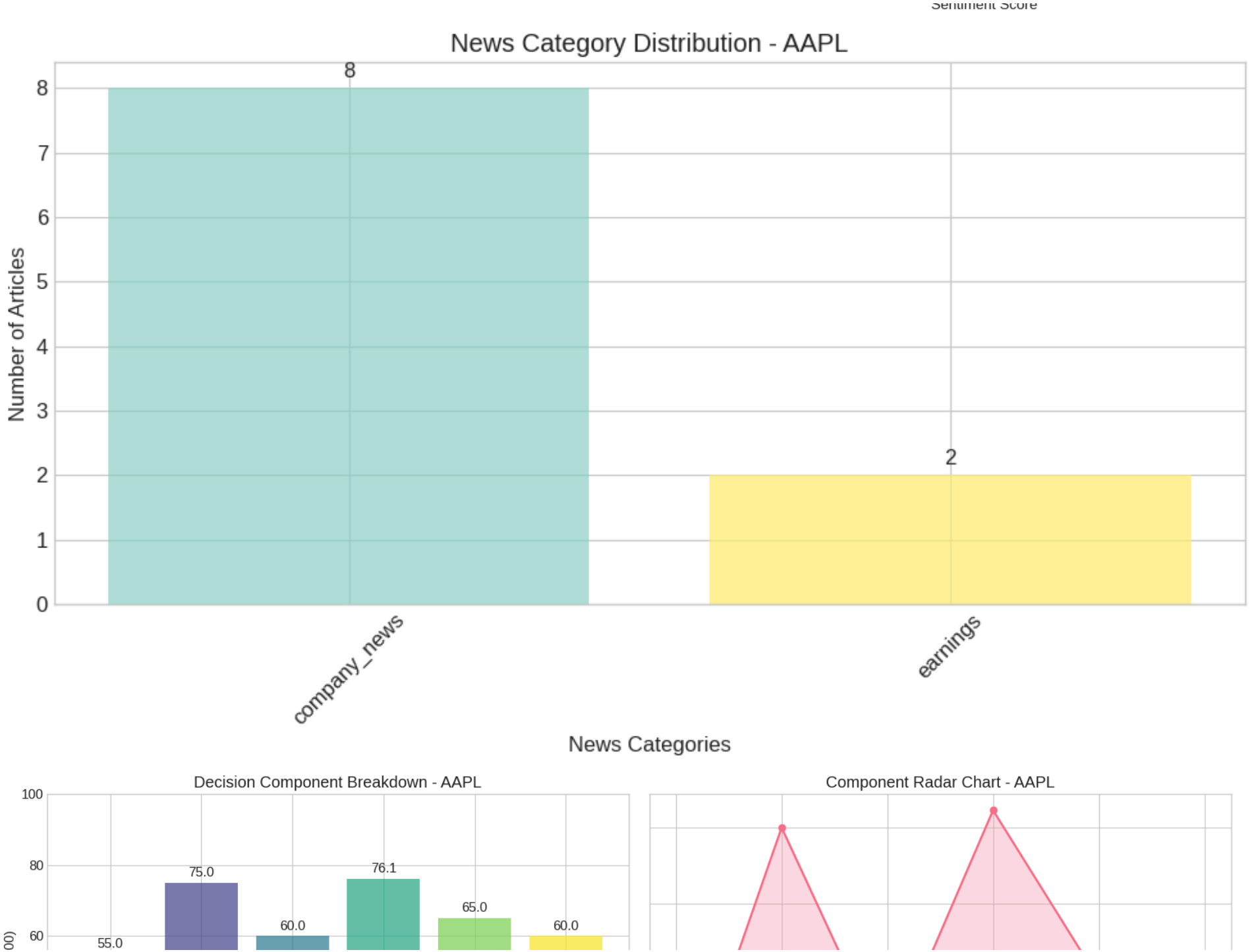
[2025-10-19 13:54:14] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.727, Summary Quality: 0.832

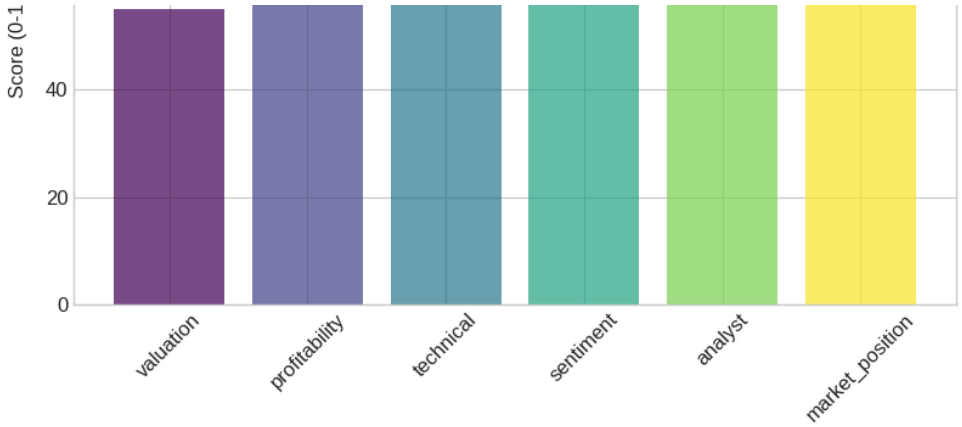
✅ AAPL analysis completed

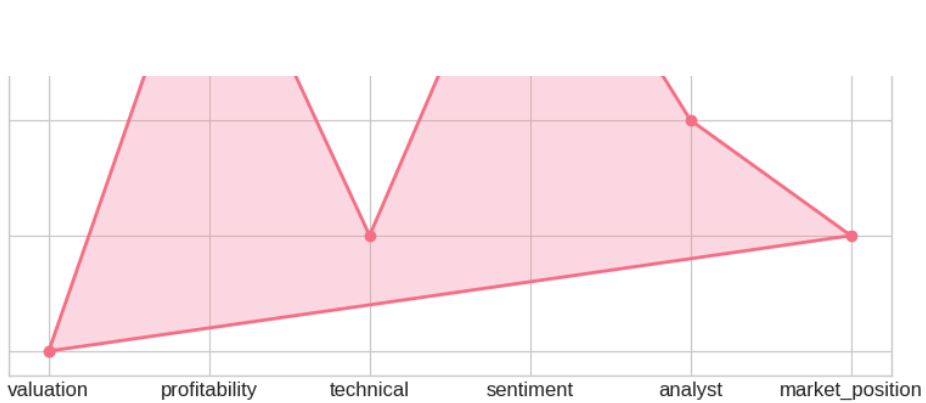
📊 Creating visualizations for AAPL...

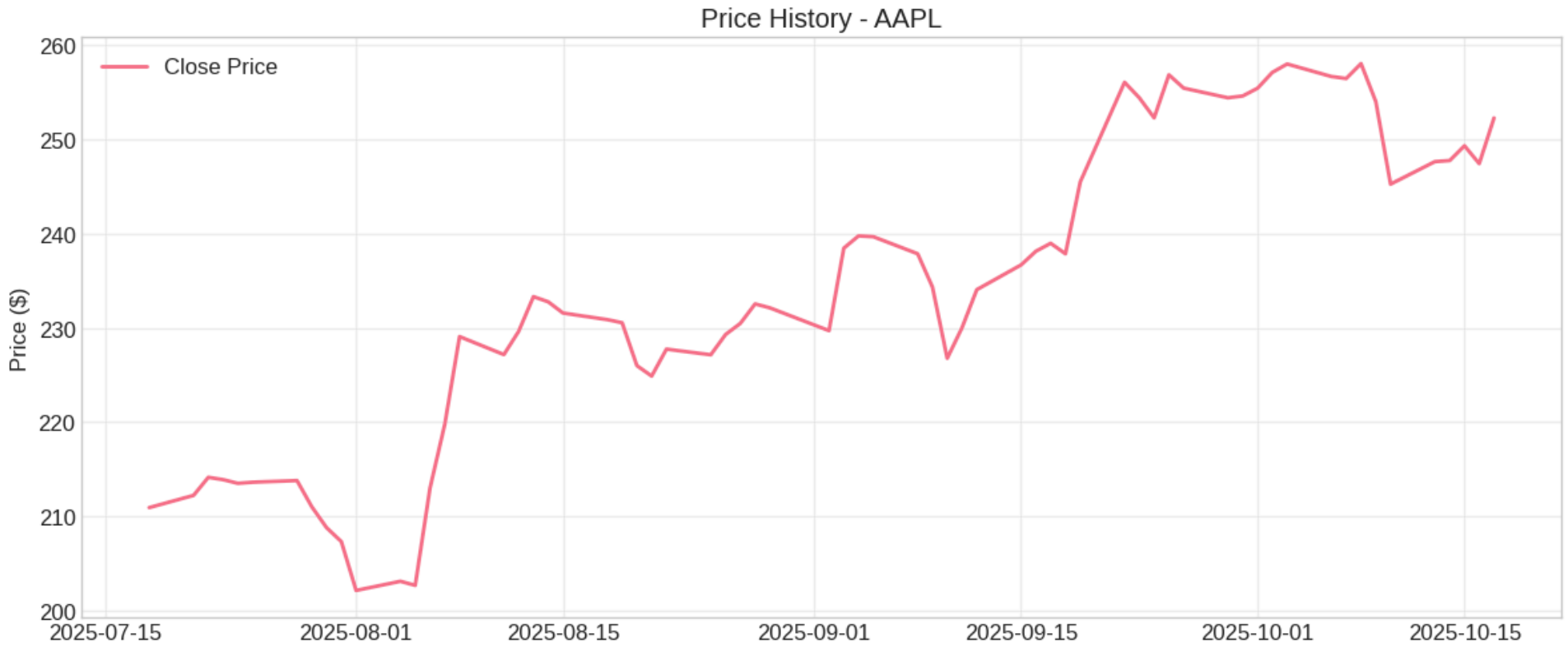


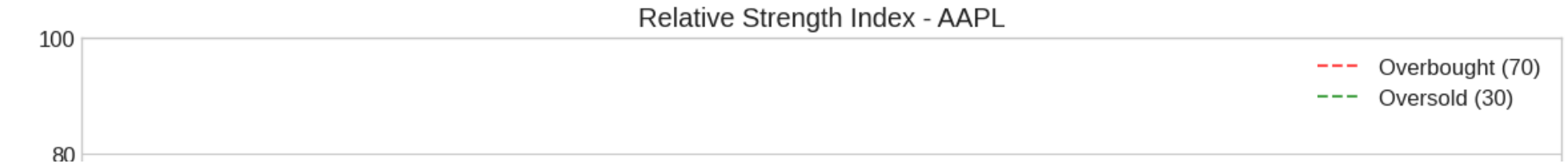


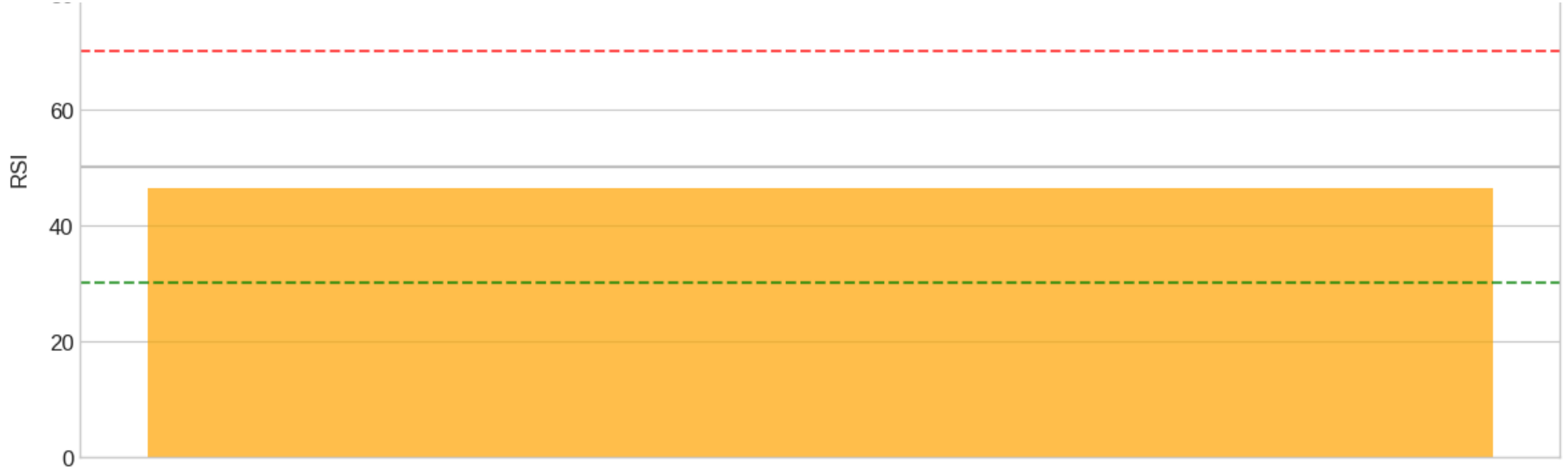

Sentiment Score
News Category Distribution - AAPL
Number of Articles
0
1
2
3
4
5
6
7
8
8
2
company_news
earnings
News Categories
Decision Component Breakdown - AAPL
100
80
60
00)
55.0
75.0
60.0
76.1
65.0
60.0
Component Radar Chart - AAPL


Score (0-1
40
20
0
valuation
profitability
technical
sentiment
analyst
market_position

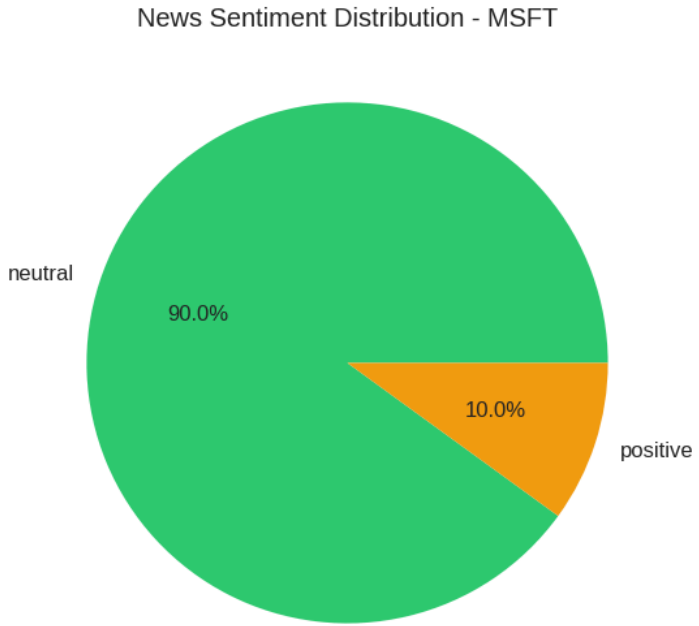

valuation
profitability
technical
sentiment
analyst
market_position

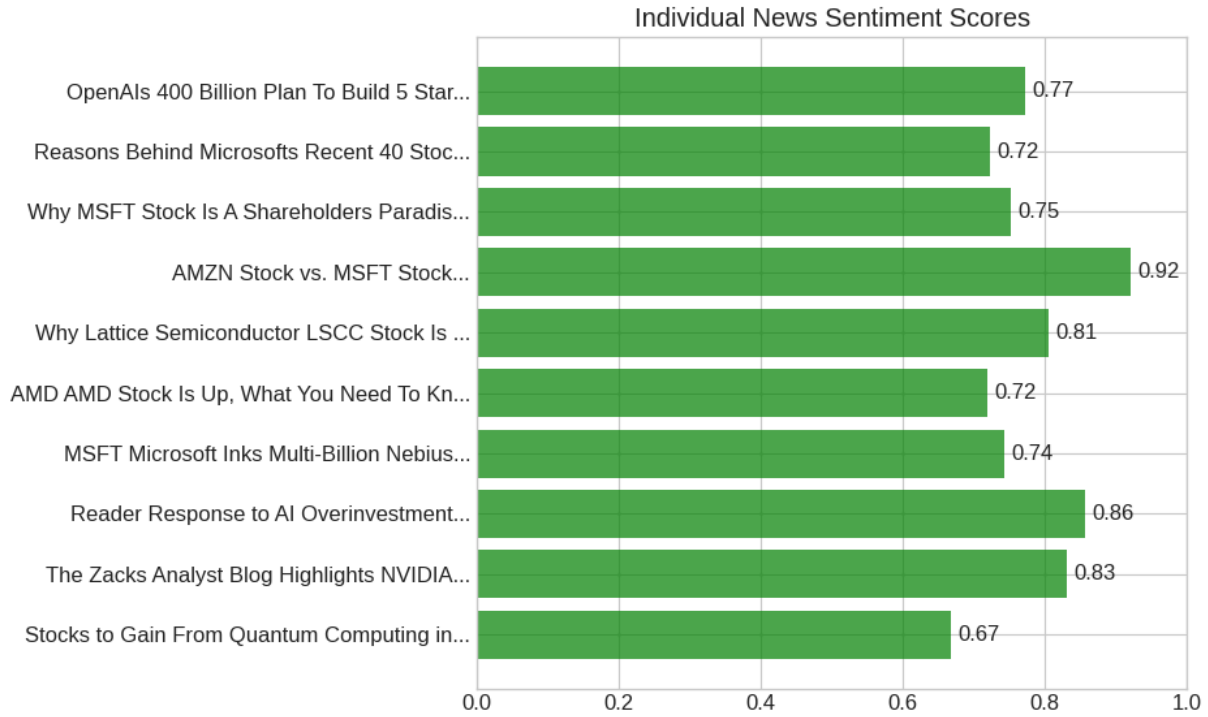

Price History - AAPL
Close Price
Price ($)
260
250
240
230
220
210
200
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15

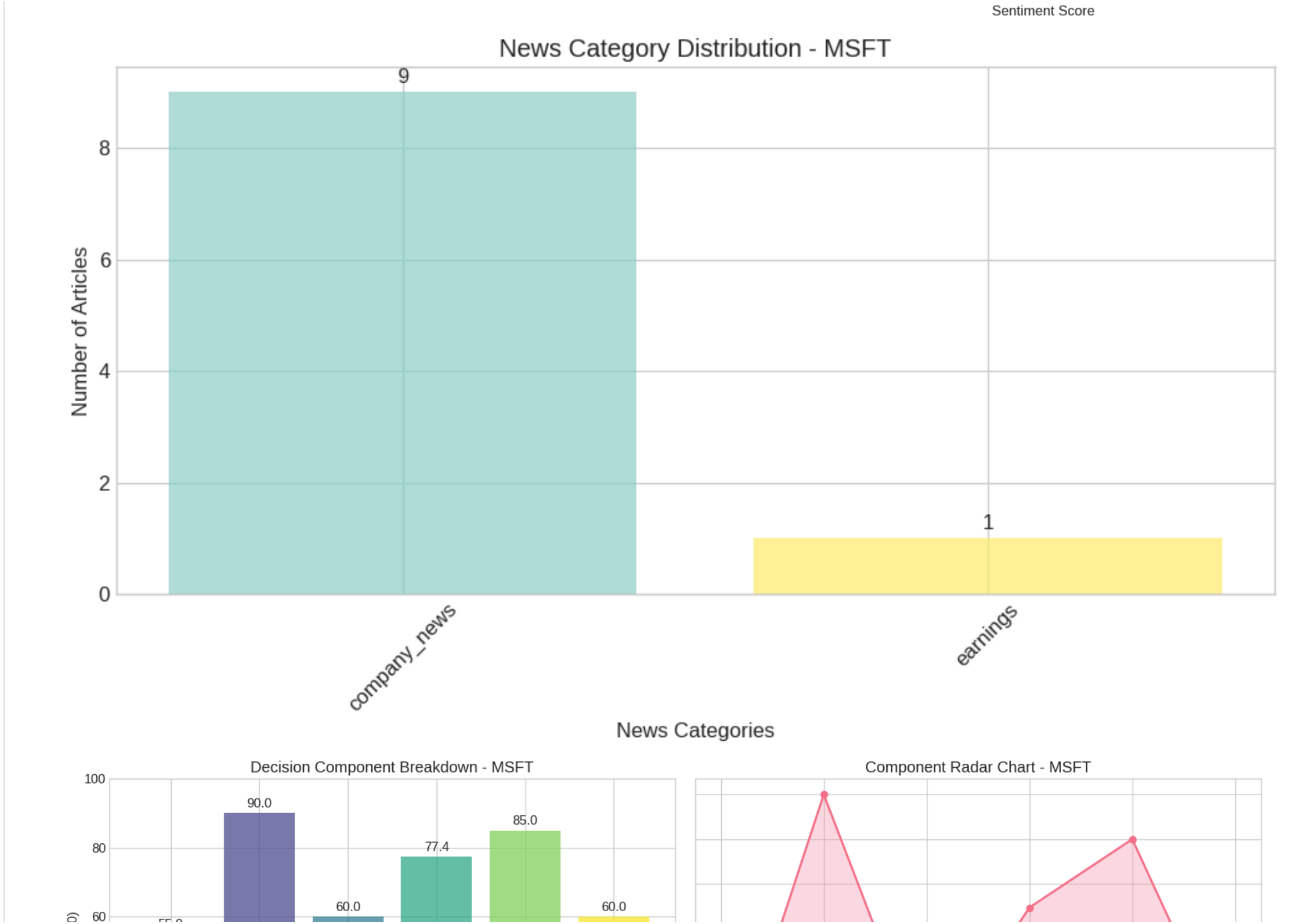

Relative Strength Index - AAPL
100
80
Overbought (70)
Oversold (30)

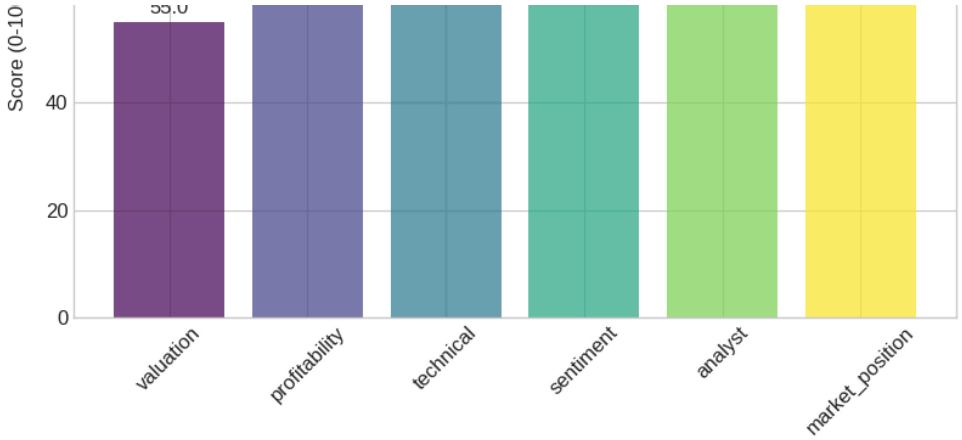


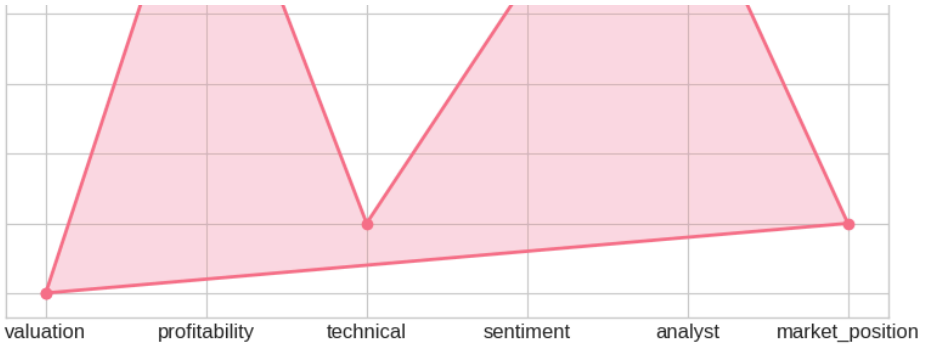
🔍 Analyzing MSFT...
[2025-10-19 13:54:20] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.731, Summary Quality: 0.831
✅ MSFT analysis completed
📊 Creating visualizations for MSFT...

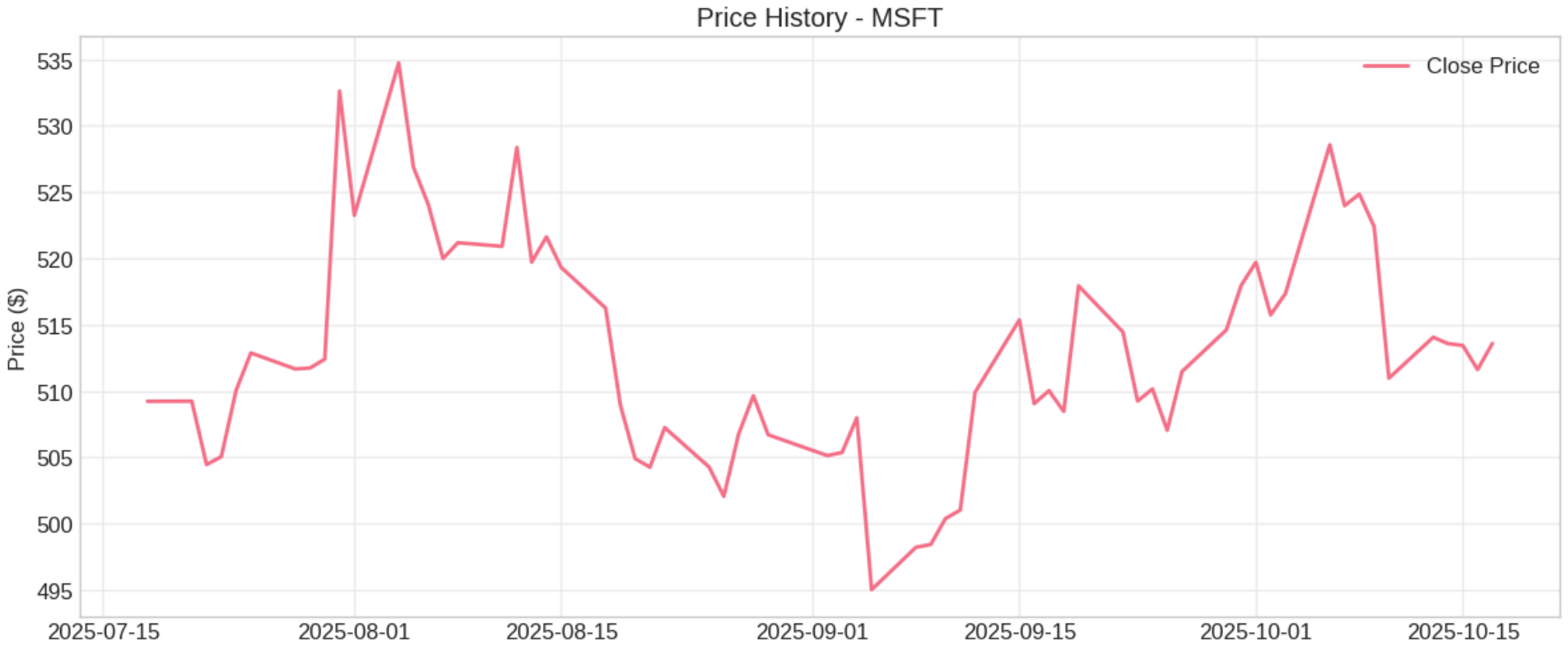


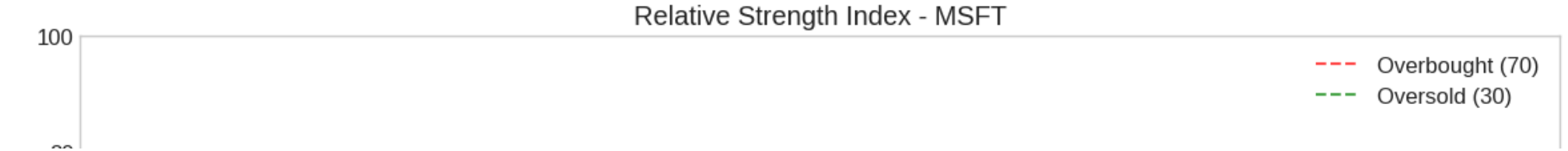


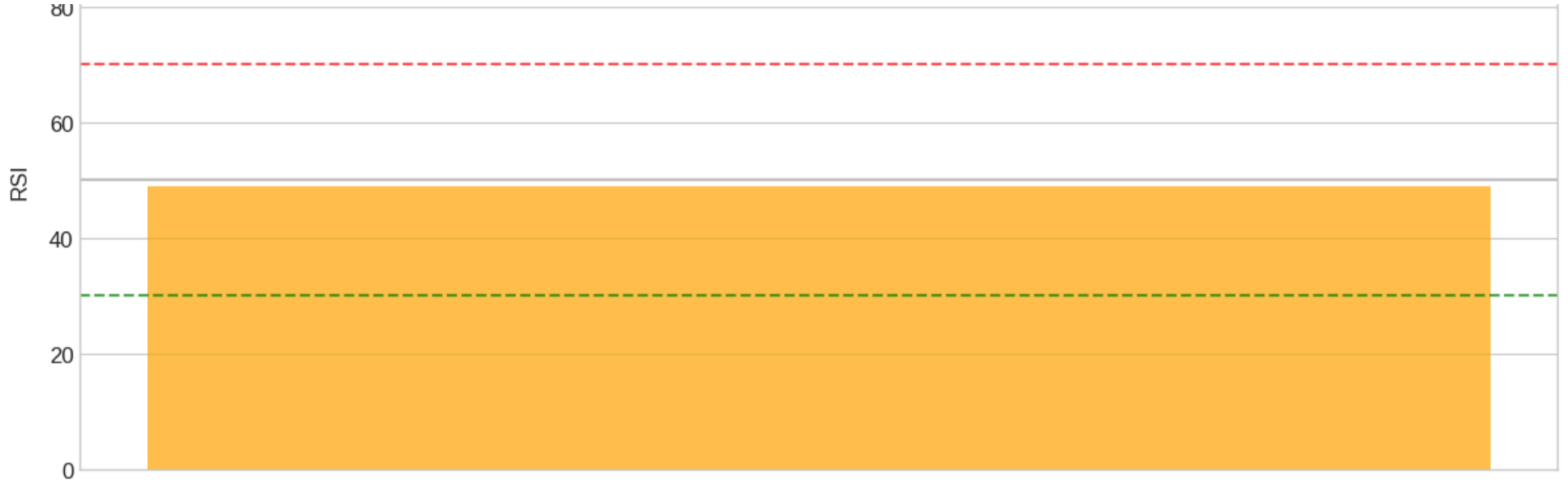

Sentiment Score
News Category Distribution - MSFT
Number of Articles
0
2
4
6
8
9
1
company_news
earnings
News Categories
Decision Component Breakdown - MSFT
100
80
60
90.0
60.0
77.4
85.0
60.0
Component Radar Chart - MSFT


Score (0-10
55.0
40
20
0
valuation
profitability
technical
sentiment
analyst
market_position

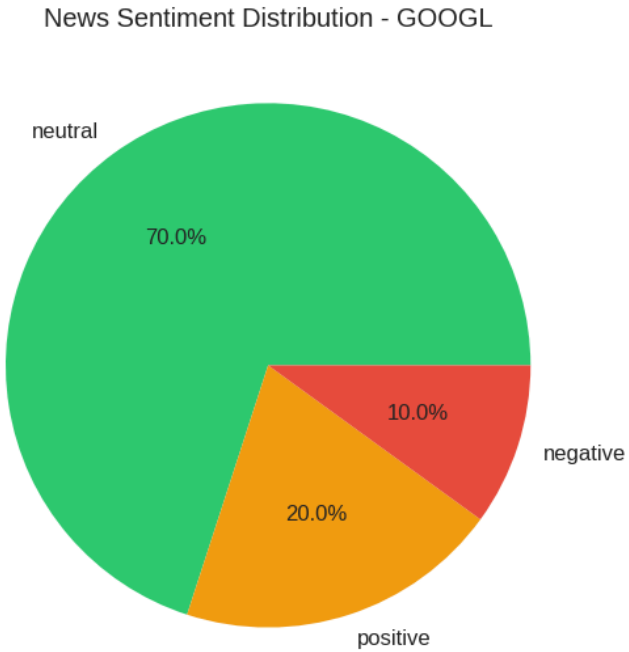

valuation
profitability
technical
sentiment
analyst
market_position

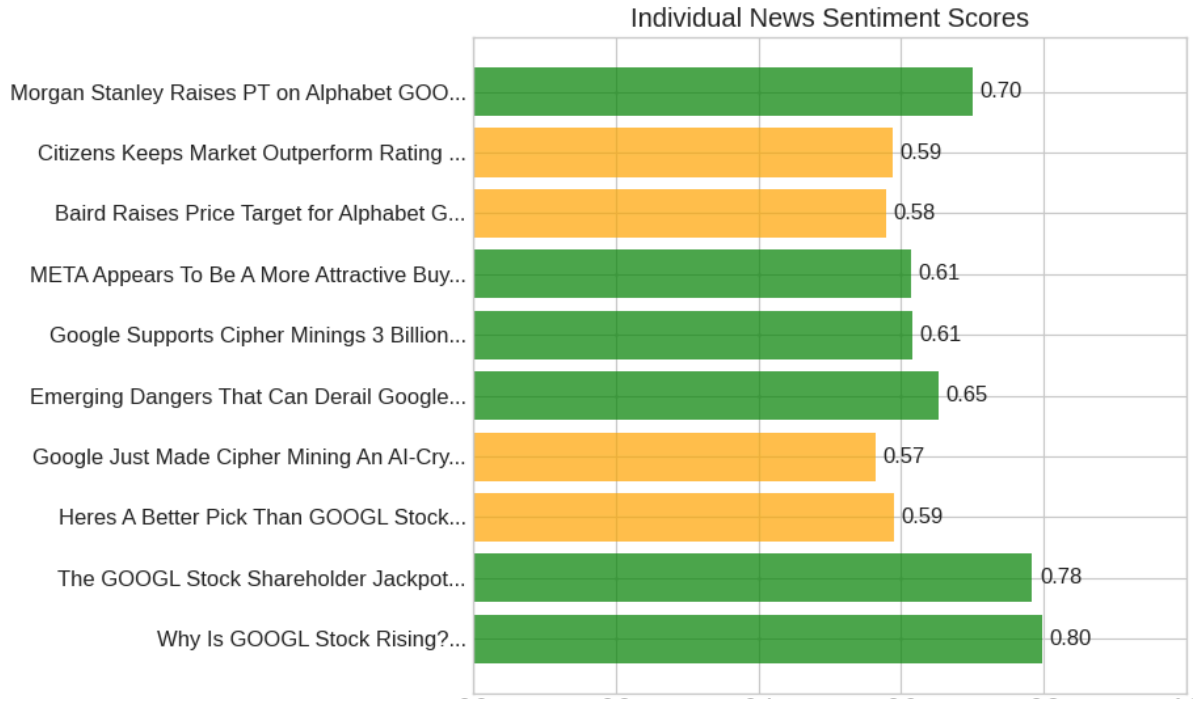

Price History - MSFT
Close Price
Price ($)
535
530
525
520
515
510
505
500
495
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15

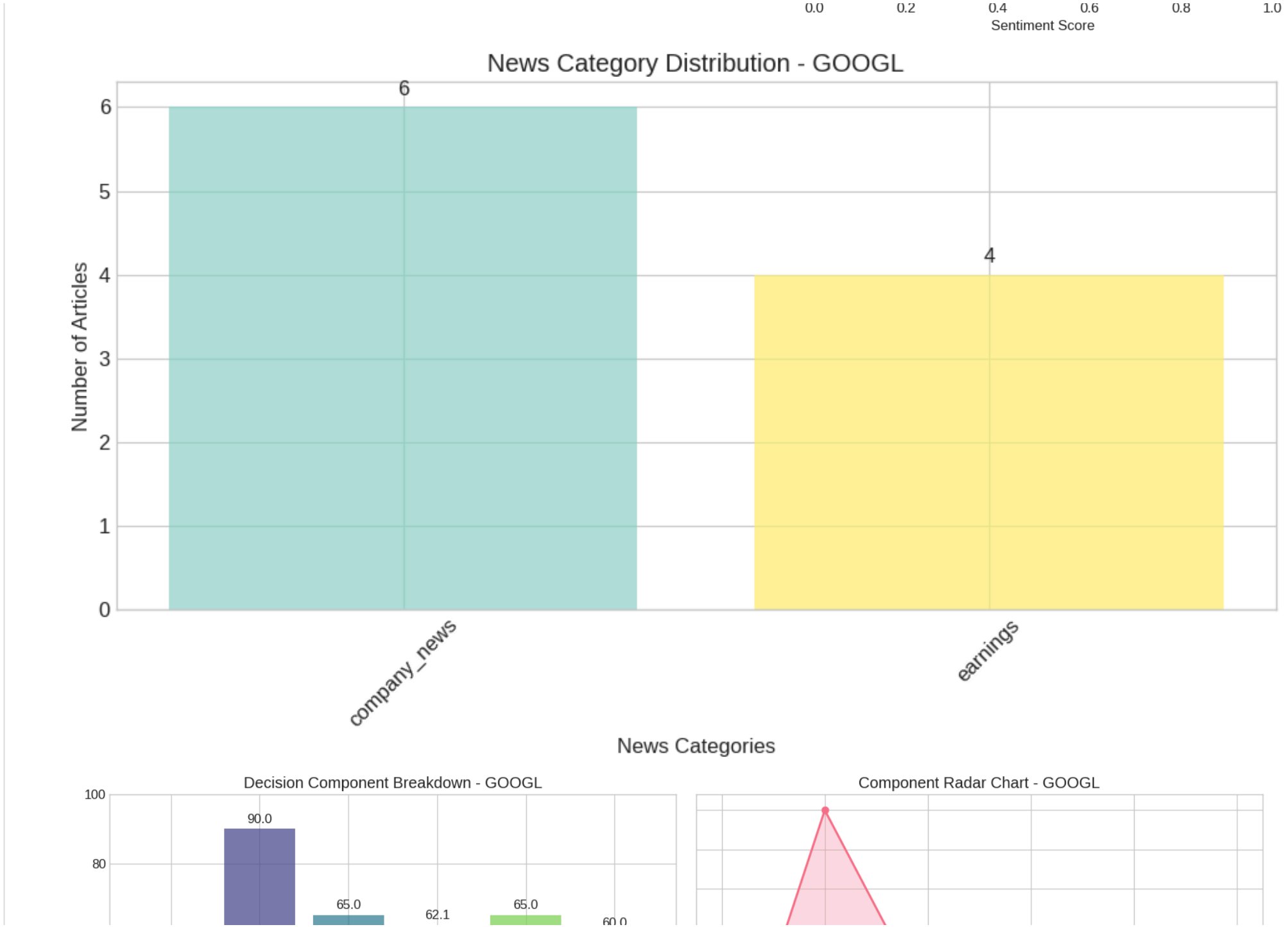

Relative Strength Index - MSFT
100
Overbought (70)
Oversold (30)

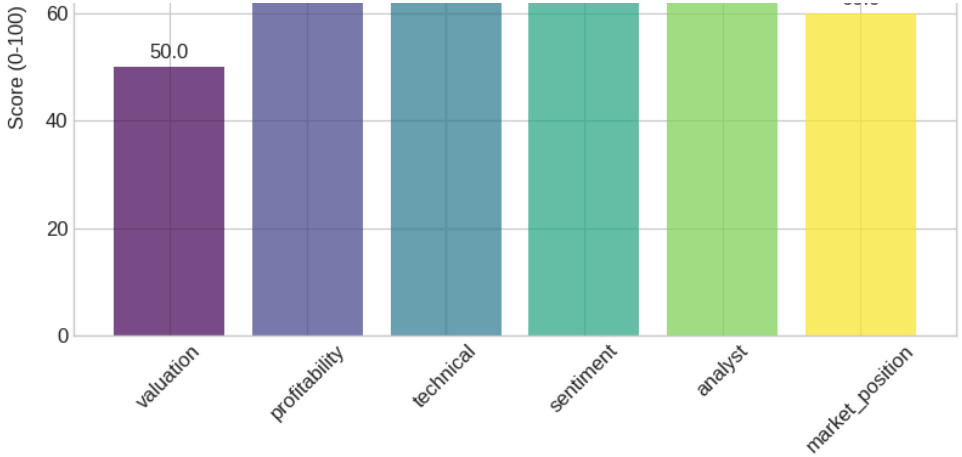


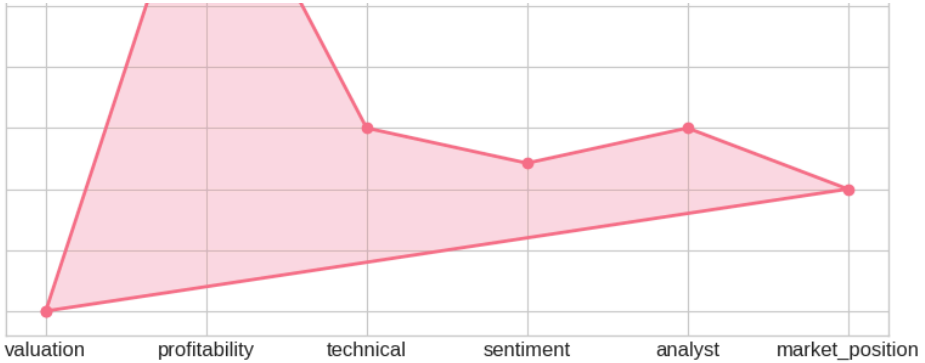
🔍 Analyzing GOOGL...
[2025-10-19 13:54:26] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.725, Summary Quality: 0.832
✅ GOOGL analysis completed
📊 Creating visualizations for GOOGL...

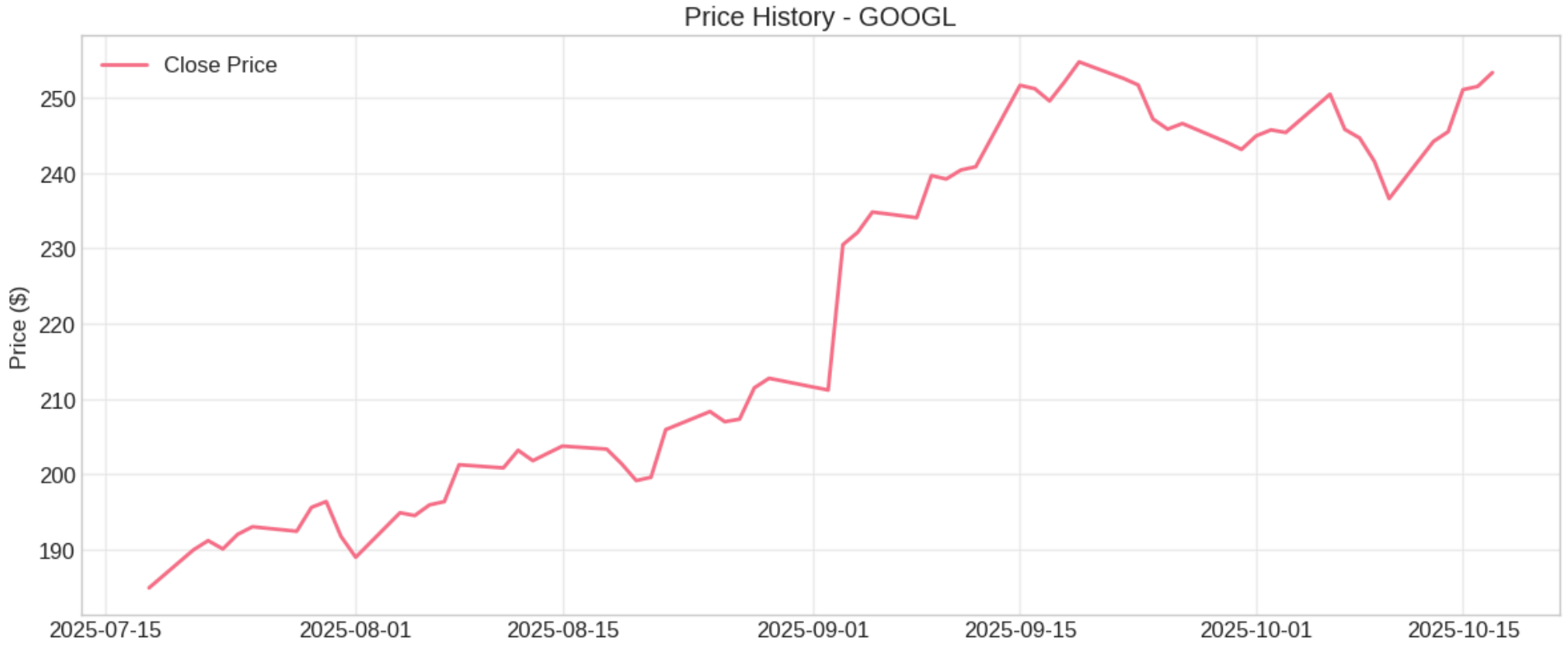


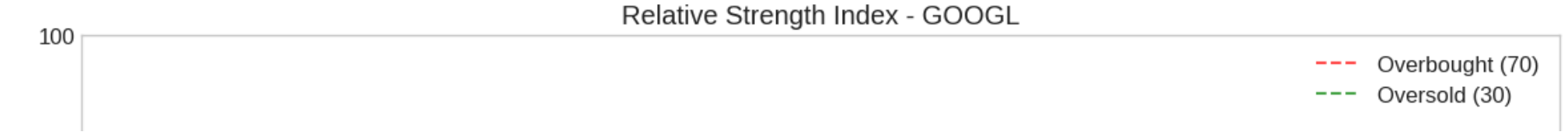


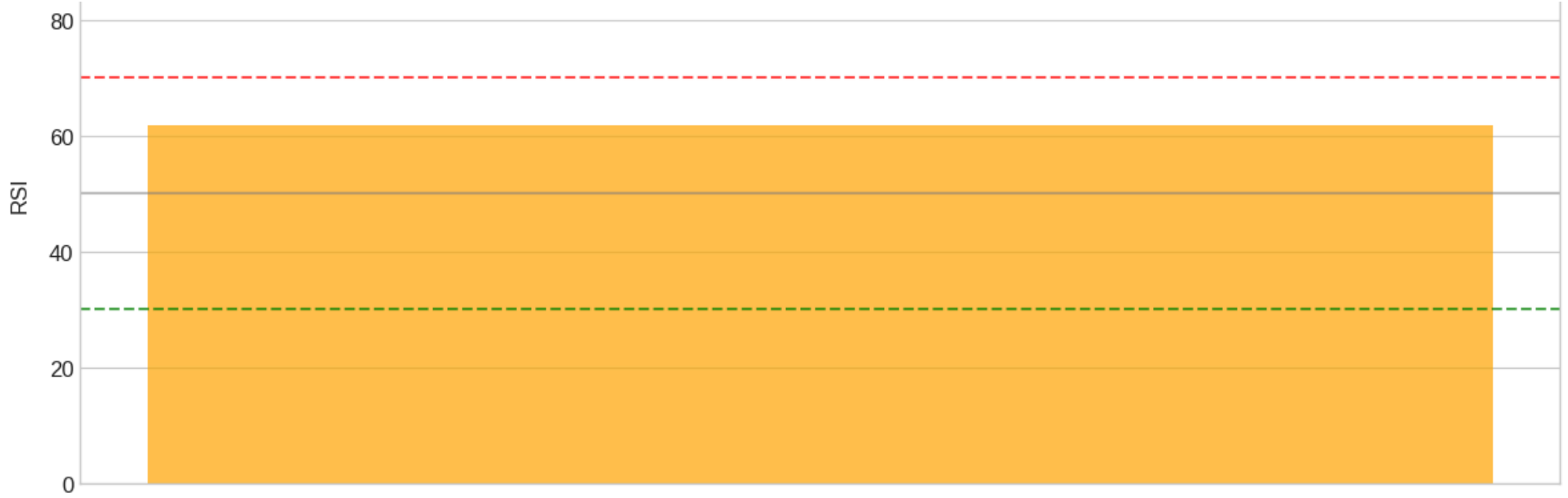

0.0
0.2
0.4
0.6
0.8
1.0
Sentiment Score
News Category Distribution - GOOGL
6
4
Number of Articles
0
1
2
3
4
5
6
company_news
earnings
News Categories
Decision Component Breakdown - GOOGL
100
80
90.0
65.0
62.1
65.0
60.0
Component Radar Chart - GOOGL


Score (0-100)
60
40
20
0
50.0
valuation
profitability
technical
sentiment
analyst
market_position

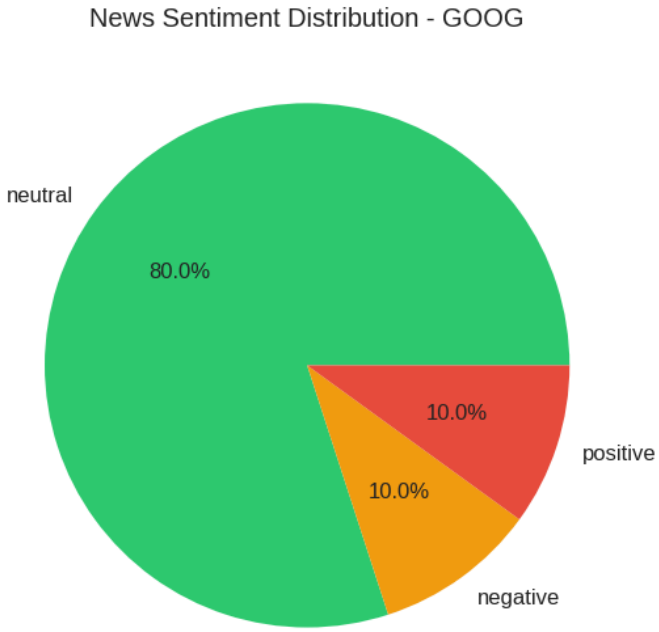

valuation
profitability
technical
sentiment
analyst
market_position

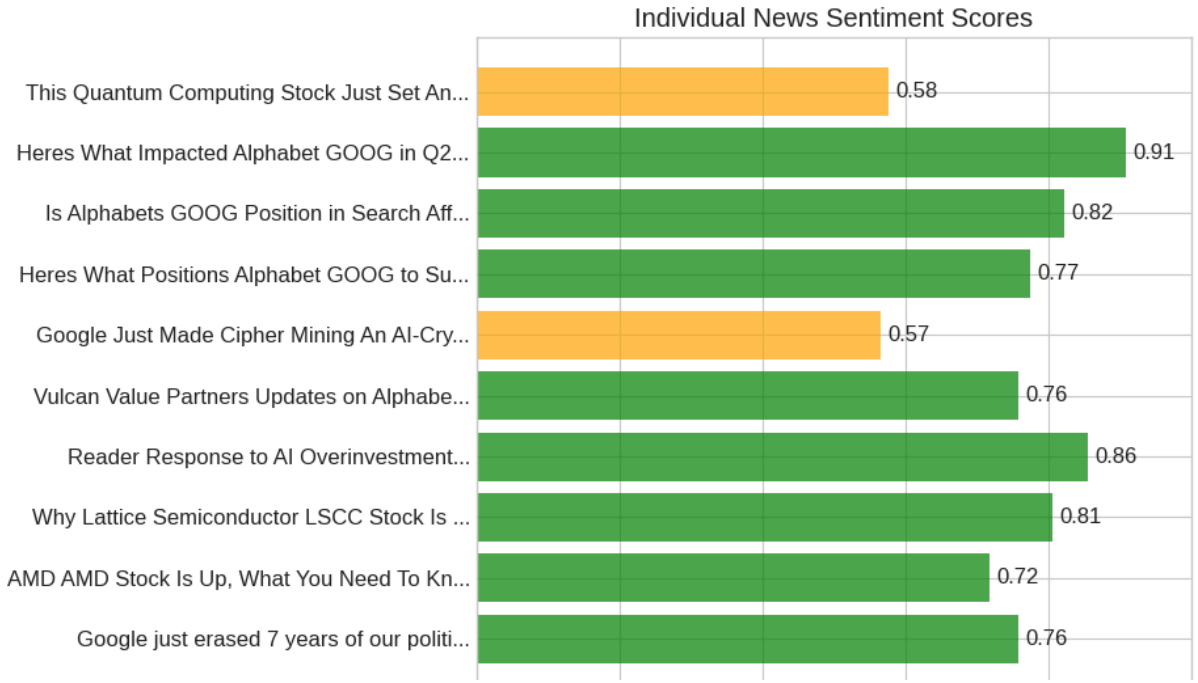

Price History - GOOGL
Close Price
Price ($)
250
240
230
220
210
200
190
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15

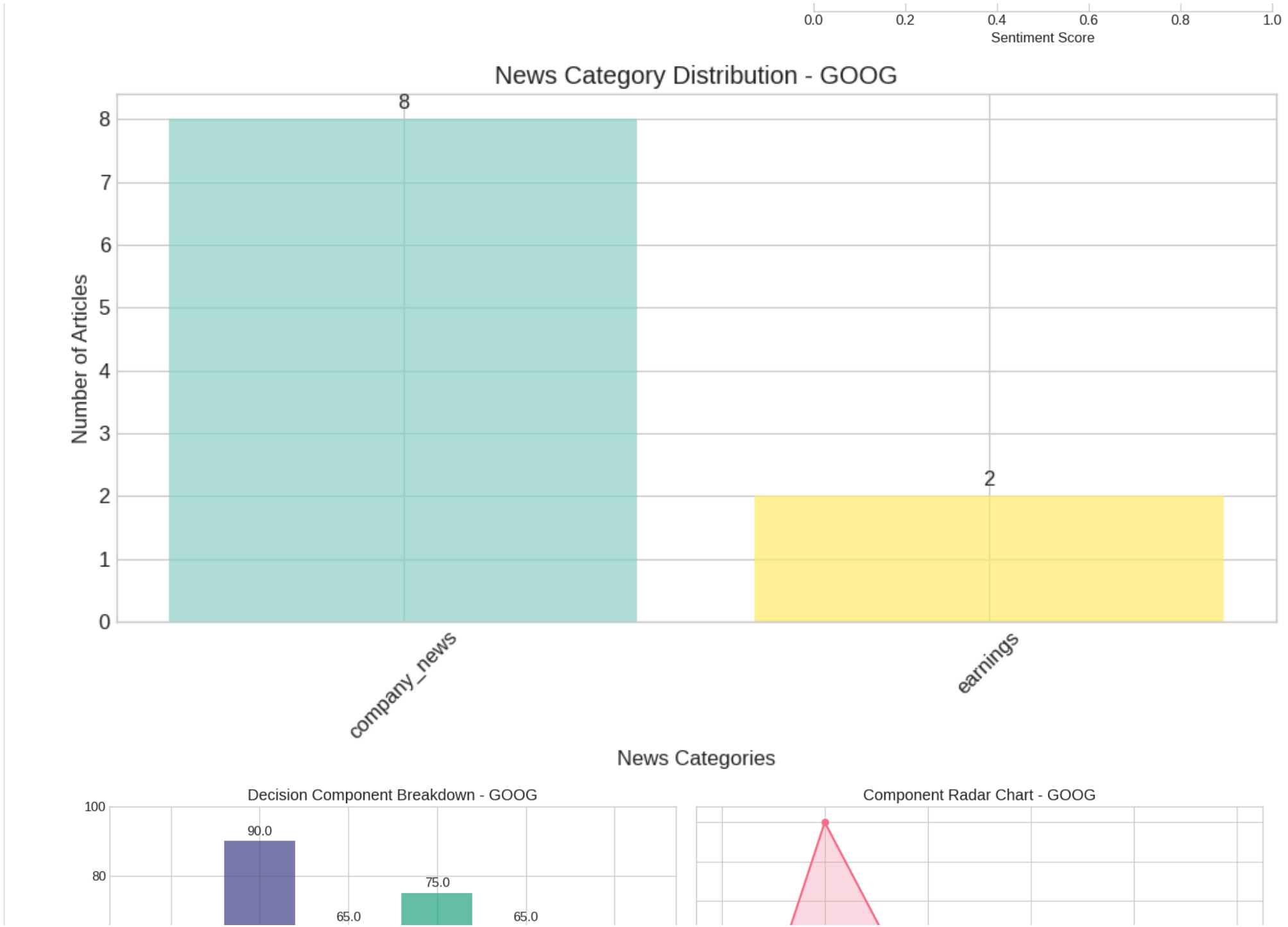

Relative Strength Index - GOOGL
100
Overbought (70)
Oversold (30)

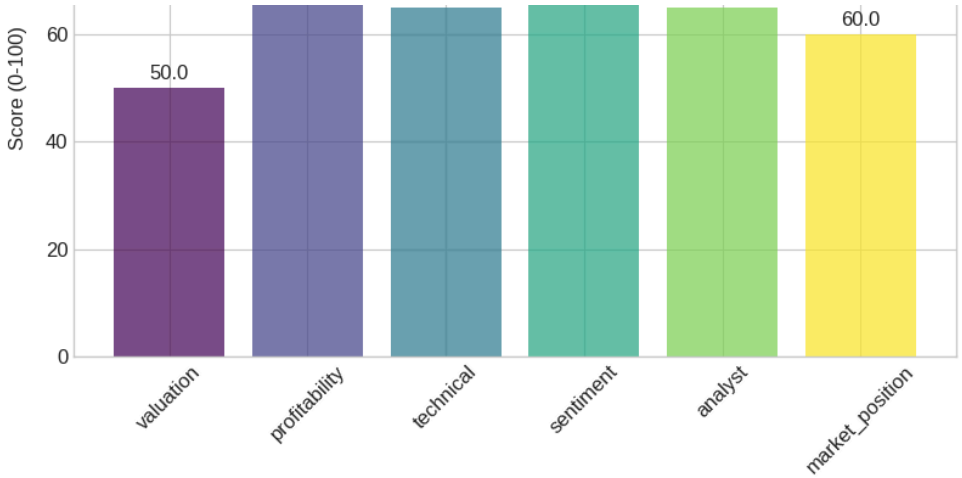


🔍 Analyzing GOOG...
[2025-10-19 13:54:32] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.727, Summary Quality: 0.832
✅ GOOG analysis completed
📊 Creating visualizations for GOOG...

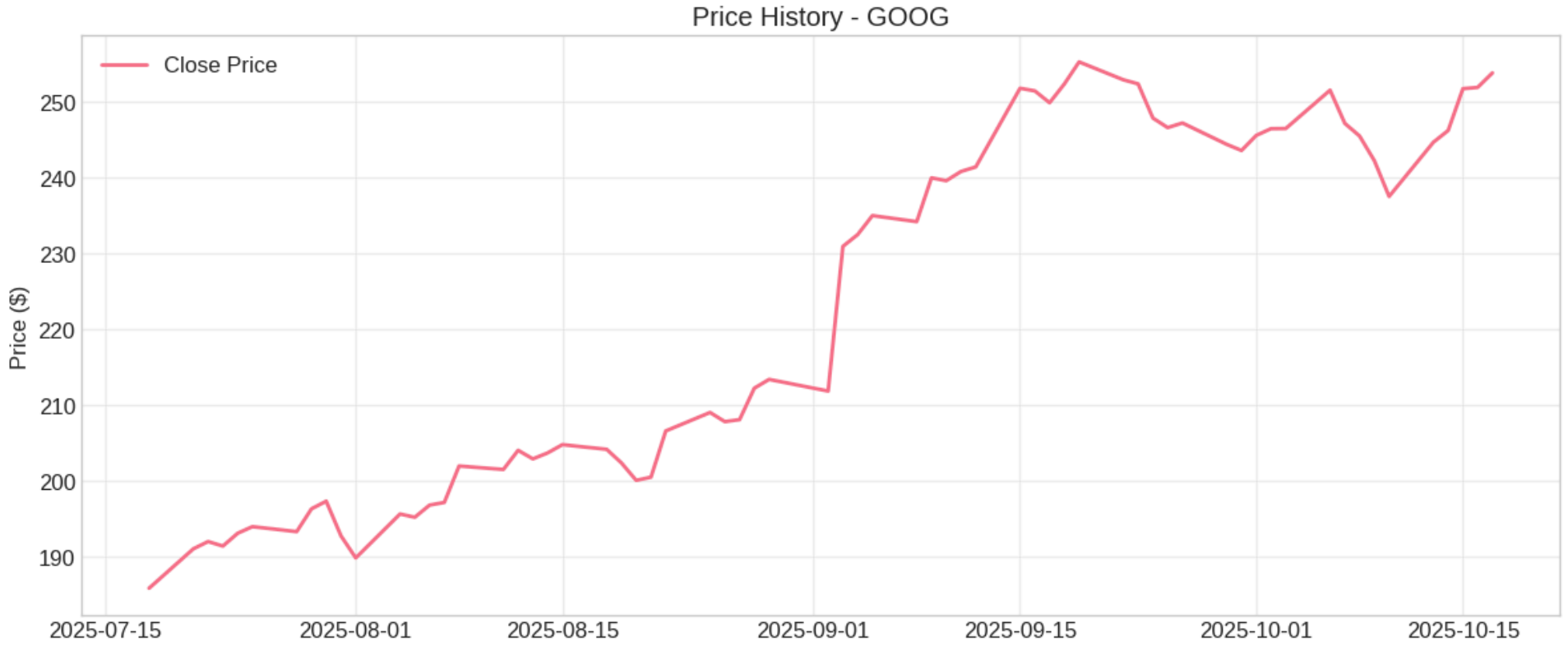


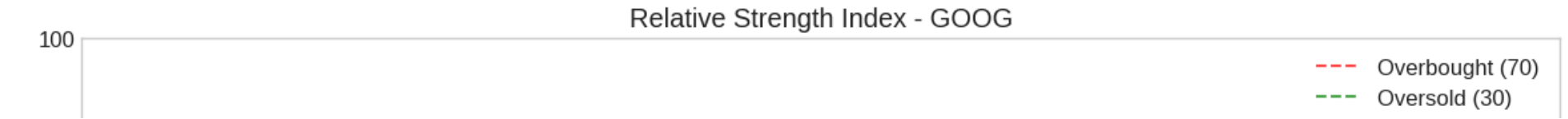


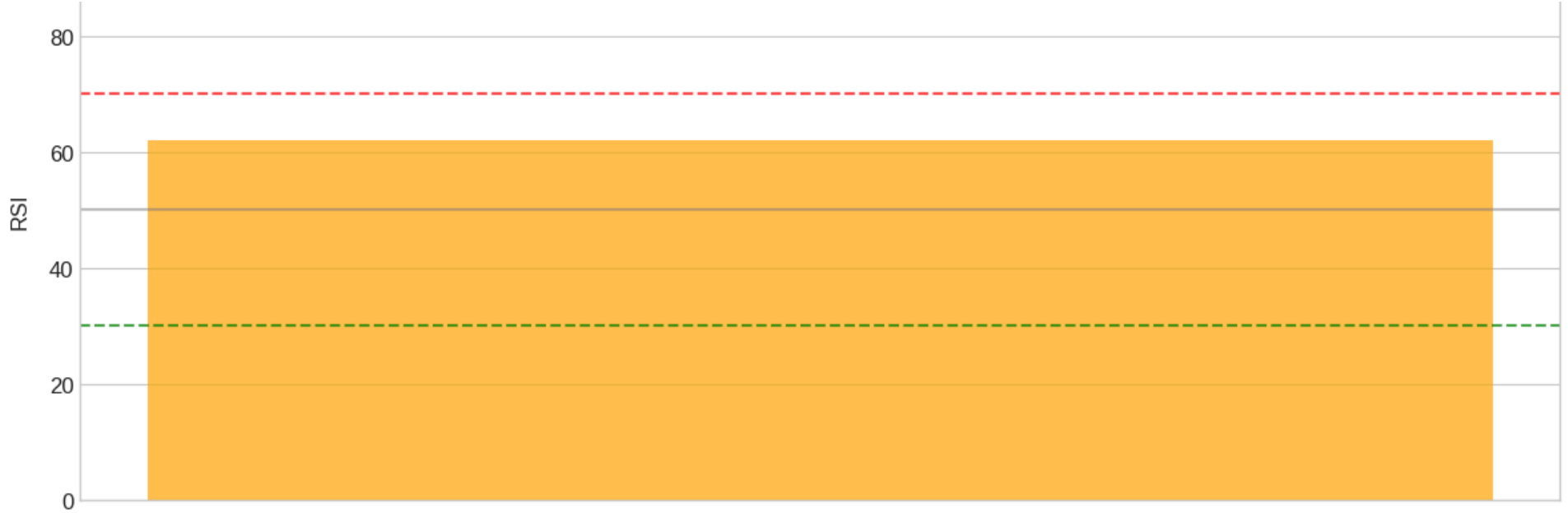

0.0
0.2
0.4
0.6
0.8
1.0
Sentiment Score
News Category Distribution - GOOG
Number of Articles
0
1
2
3
4
5
6
7
8
8
2
company_news
earnings
News Categories
Decision Component Breakdown - GOOG
100
80
90.0
65.0
75.0
65.0
Component Radar Chart - GOOG


Score (0-100)
0
20
40
60
50.0
60.0
valuation
profitability
technical
sentiment
analyst
market_position

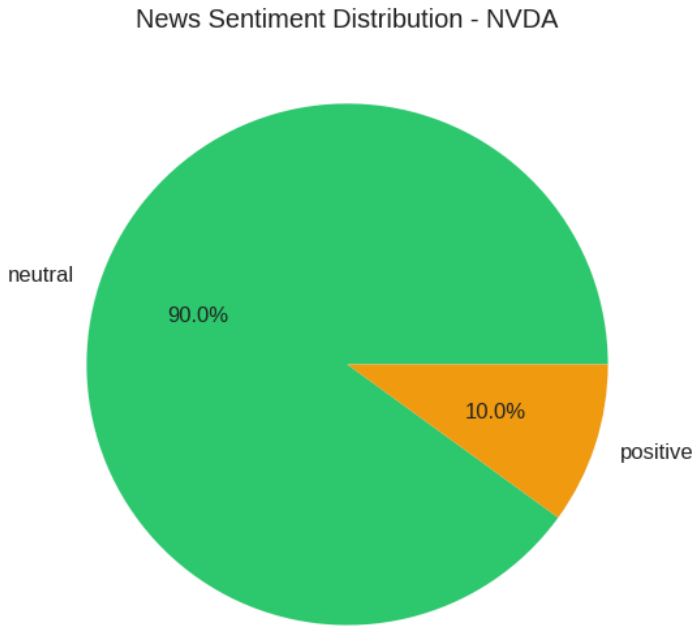

valuation
profitability
technical
sentiment
analyst
market_position

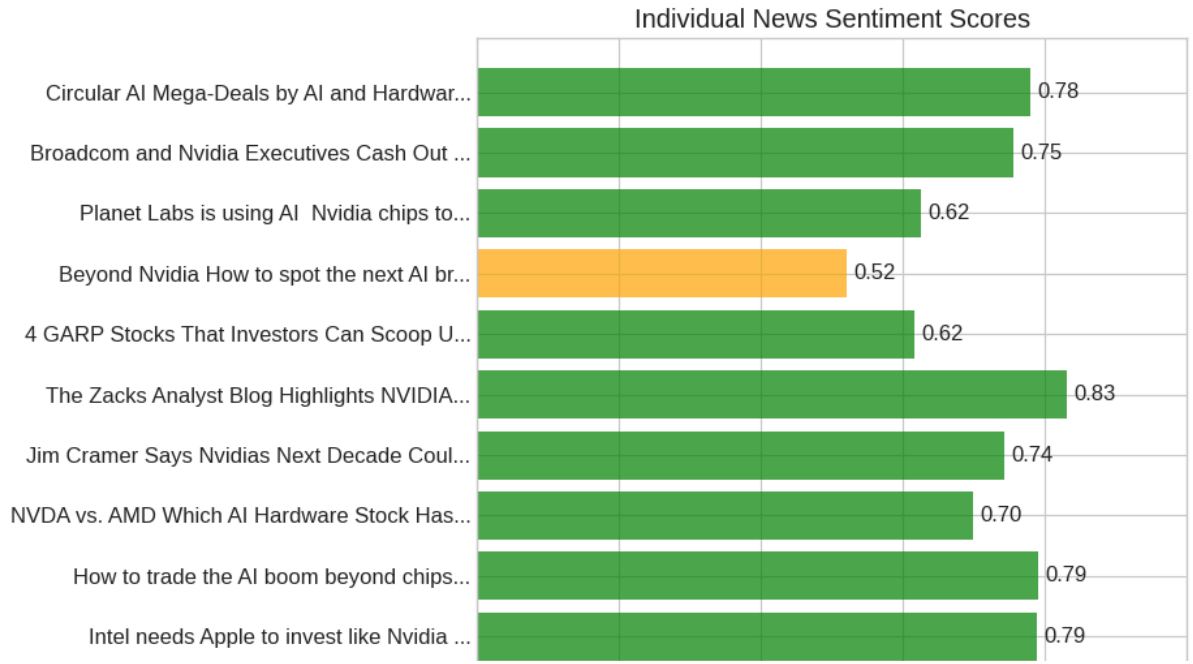

Price History - GOOG
Close Price
Price ($)
190
200
210
220
230
240
250
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15

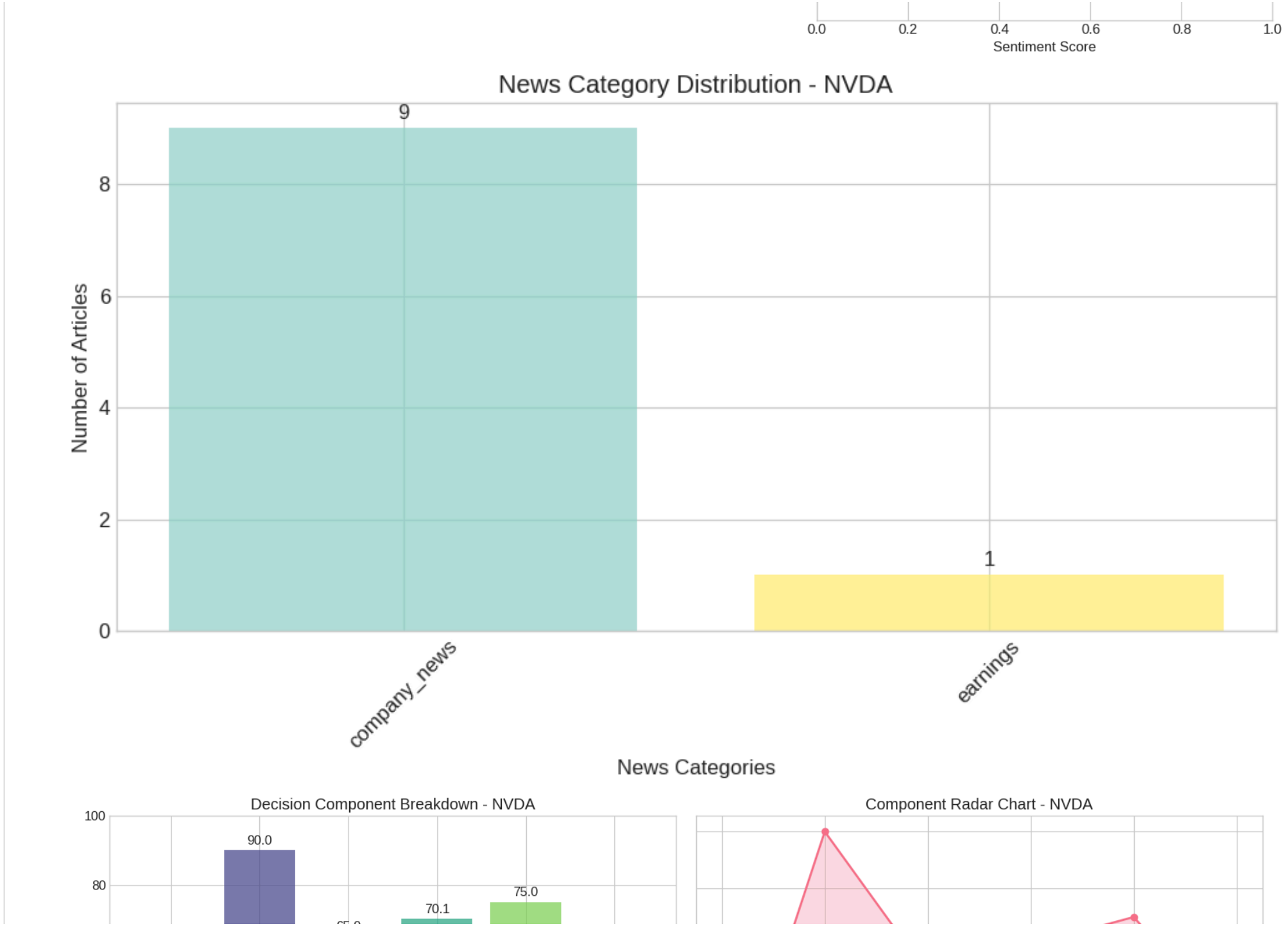

Relative Strength Index - GOOG
100
Overbought (70)
Oversold (30)

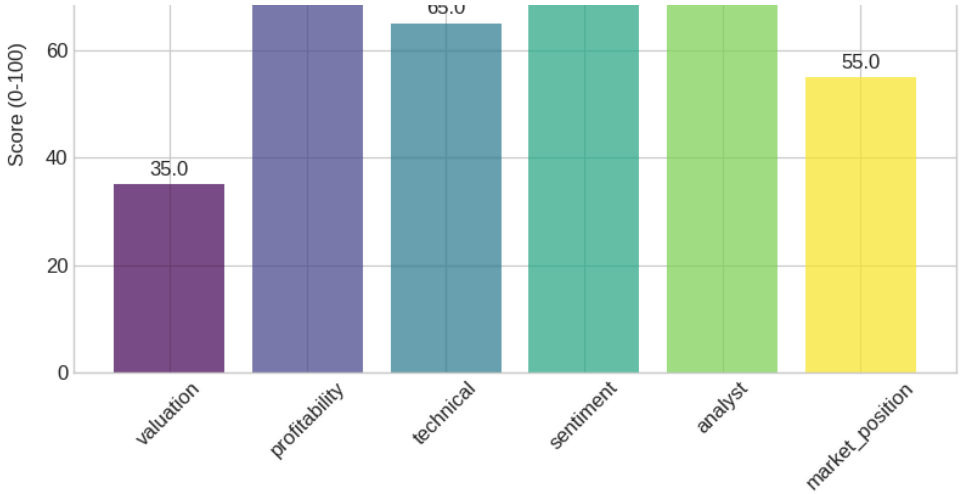


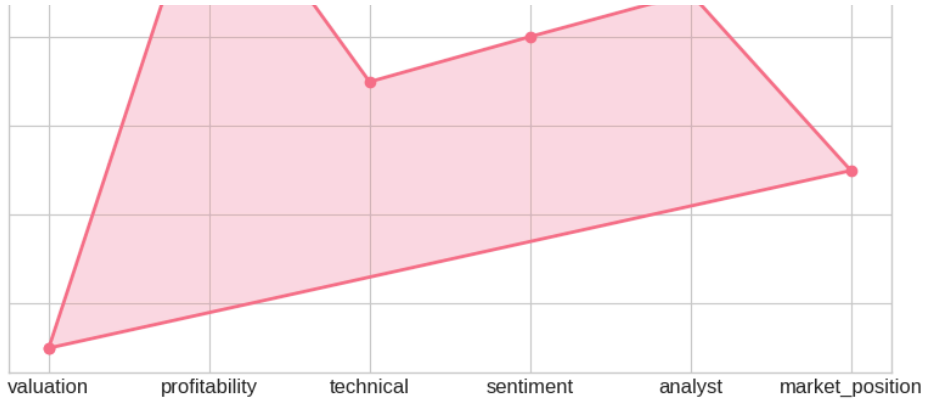
🔍 Analyzing NVDA...
[2025-10-19 13:54:38] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.726, Summary Quality: 0.832
✅ NVDA analysis completed
📊 Creating visualizations for NVDA...

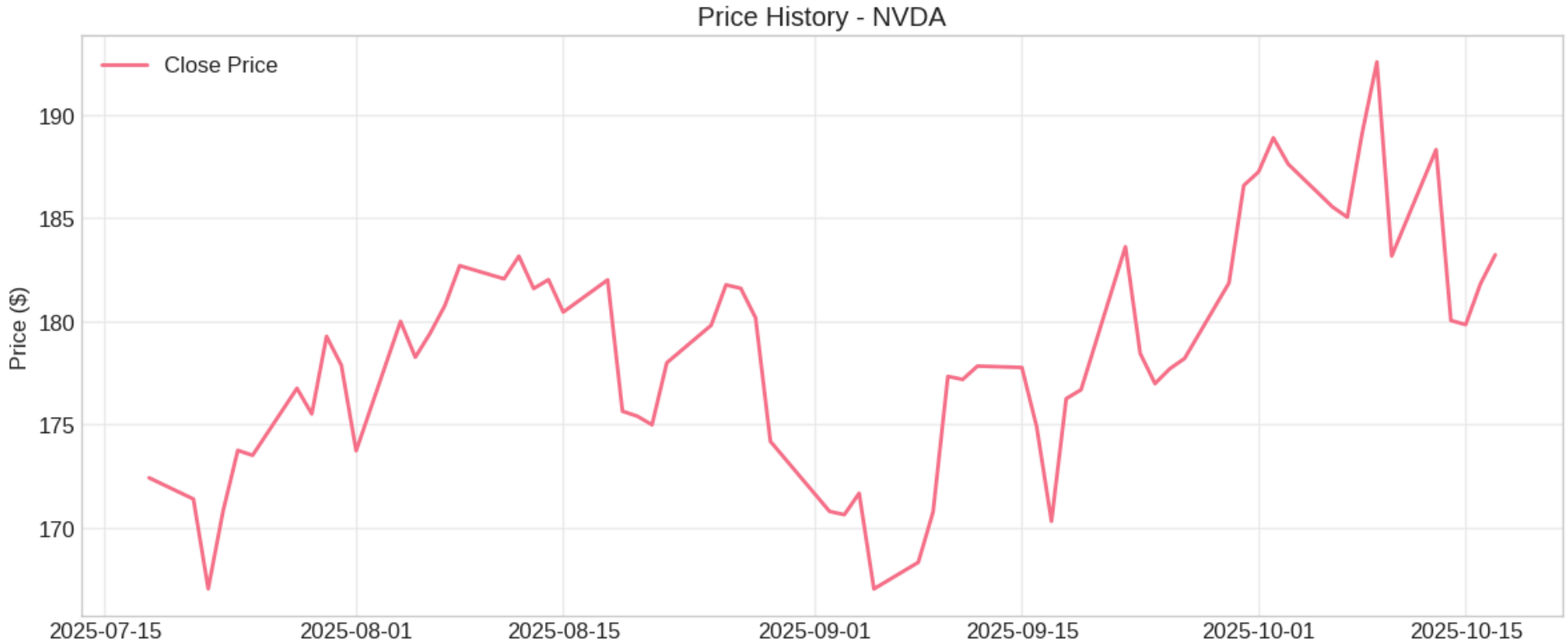


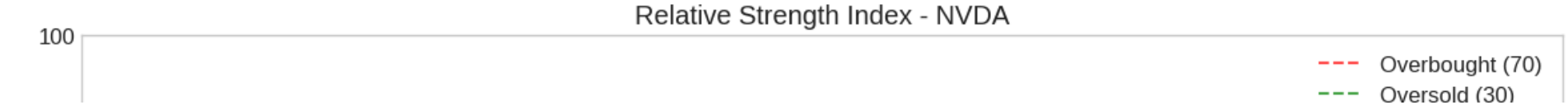


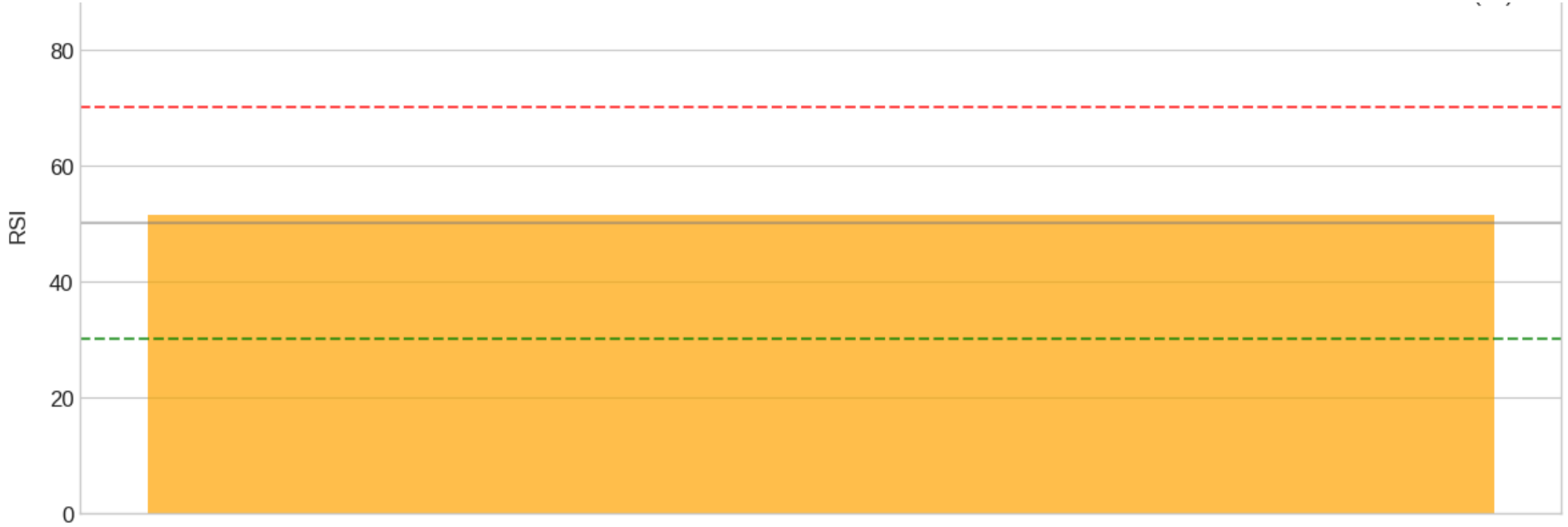

0.0
0.2
0.4
0.6
0.8
1.0
Sentiment Score
News Category Distribution - NVDA
Number of Articles
0
2
4
6
8
9
1
company_news
earnings
News Categories
Decision Component Breakdown - NVDA
100
80
90.0
70.1
75.0
Component Radar Chart - NVDA


Score (0-100)
0
20
40
60
35.0
65.0
55.0
valuation
profitability
technical
sentiment
analyst
market_position

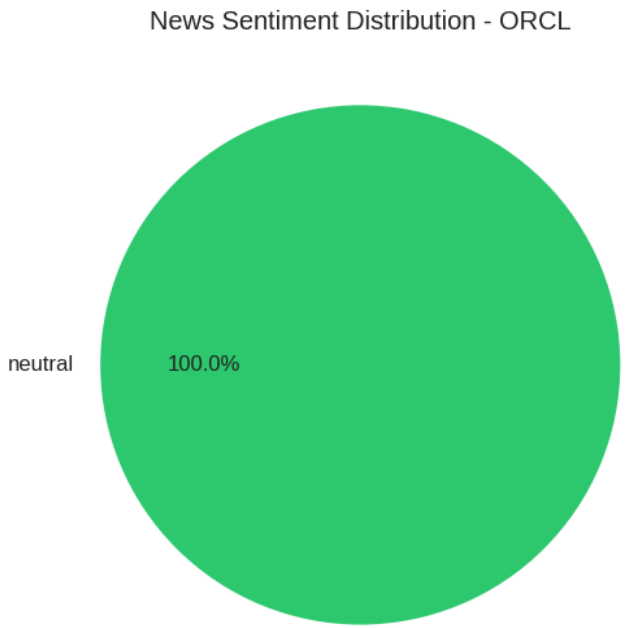

valuation
profitability
technical
sentiment
analyst
market_position

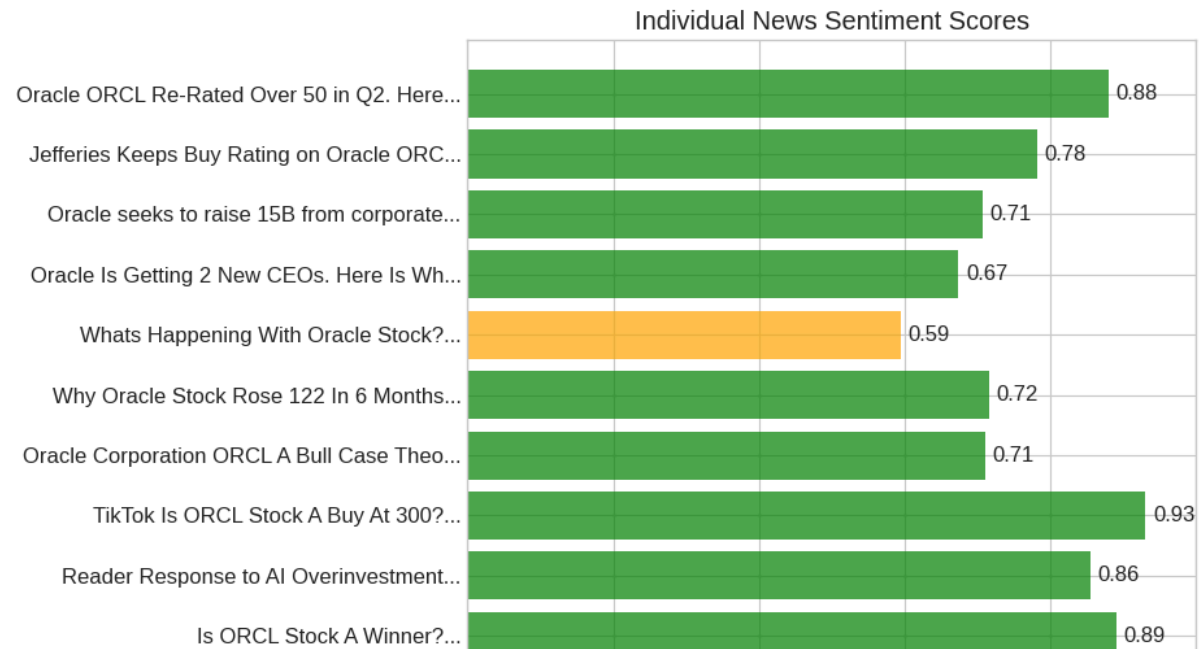

Price History - NVDA
Close Price
Price ($)
170
175
180
185
190
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15

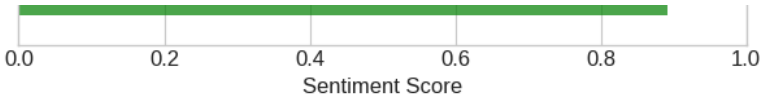

Relative Strength Index - NVDA
100
Overbought (70)
Oversold (30)

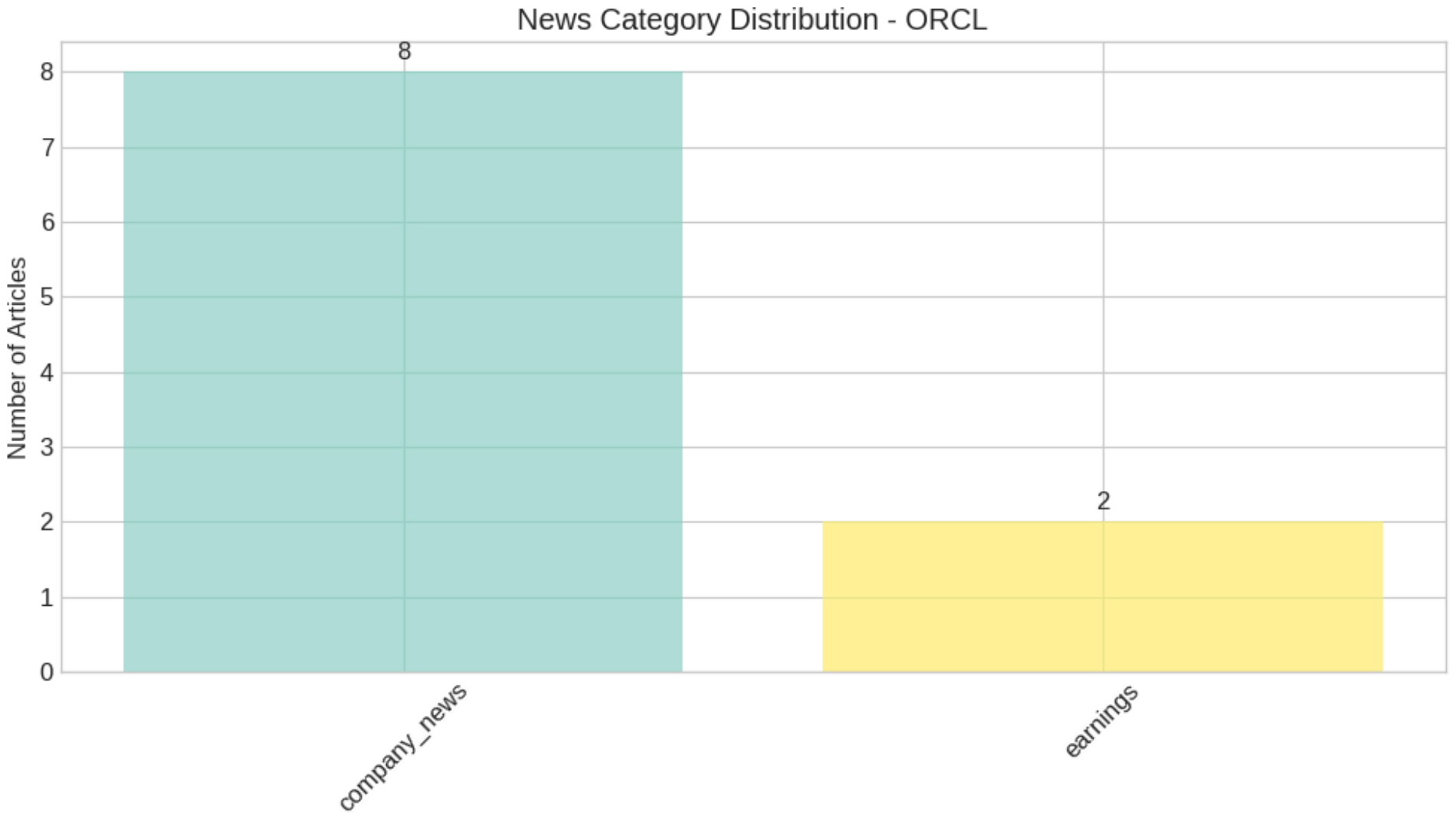


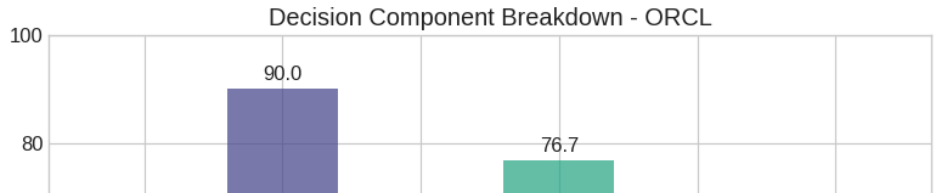
🔍 Analyzing ORCL...
[2025-10-19 13:54:44] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.729, Summary Quality: 0.833
✅ ORCL analysis completed
📊 Creating visualizations for ORCL...

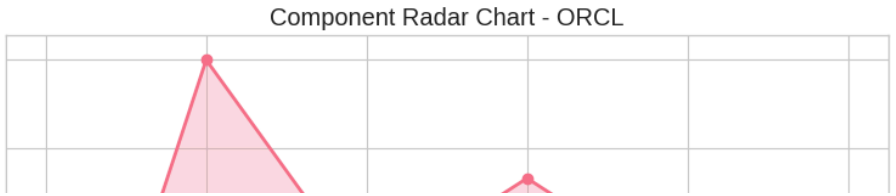


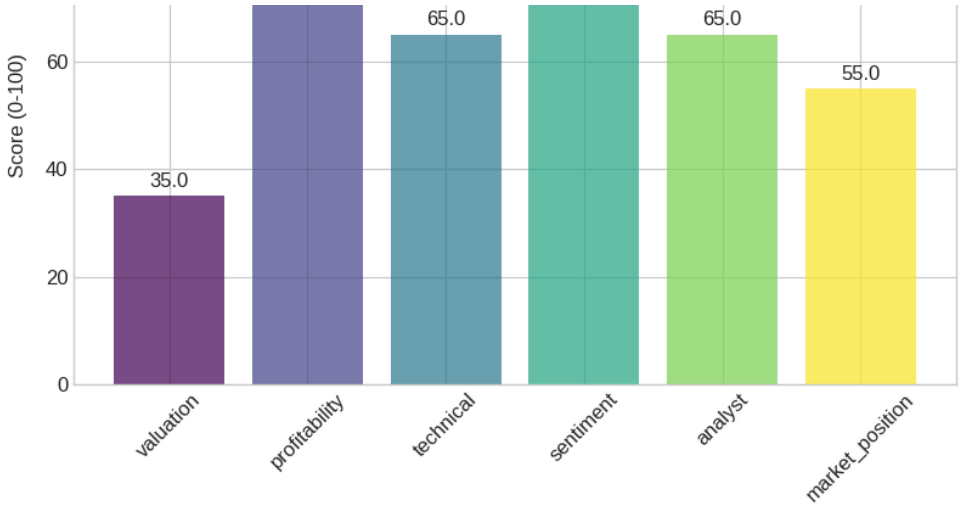


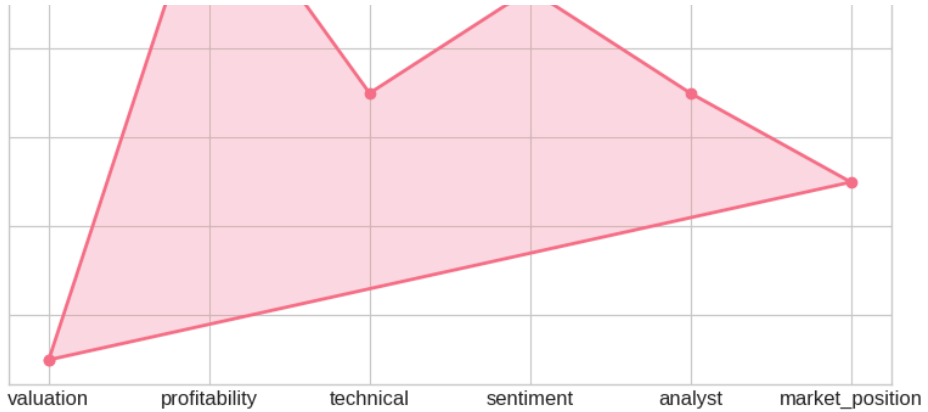

0.0
0.2
0.4
0.6
0.8
1.0
Sentiment Score


News Category Distribution - ORCL
Number of Articles
0
1
2
3
4
5
6
7
8
8
2
company_news
earnings
News Categories

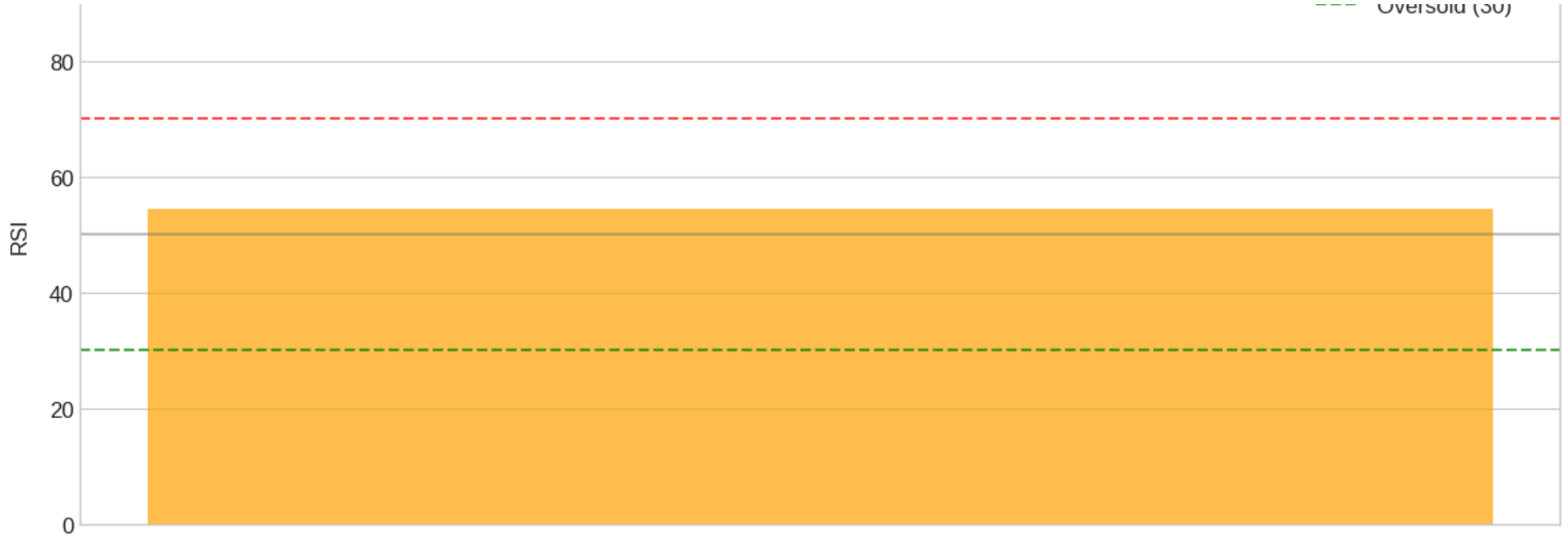

Decision Component Breakdown - ORCL
100
80
90.0
76.7


Component Radar Chart - ORCL

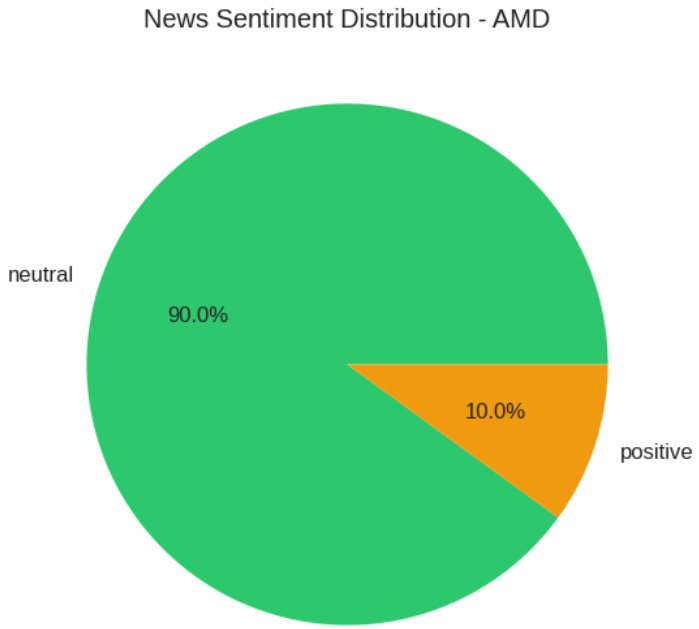

Score (0-100)
0
20
40
60
35.0
65.0
65.0
55.0
valuation
profitability
technical
sentiment
analyst
market_position

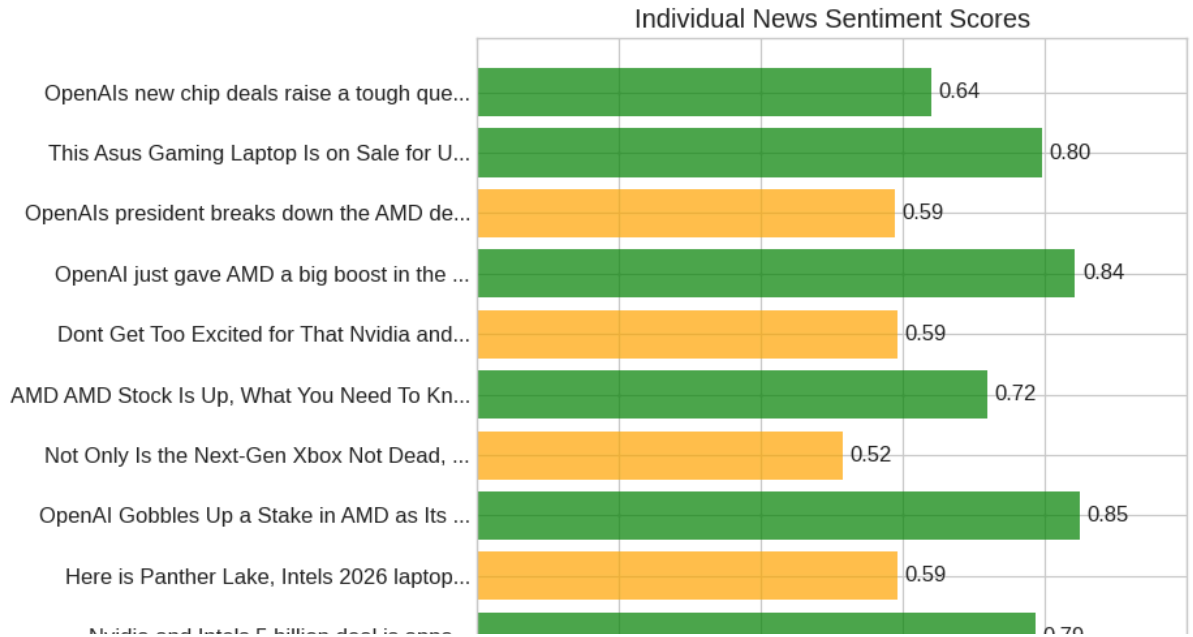

valuation
profitability
technical
sentiment
analyst
market_position

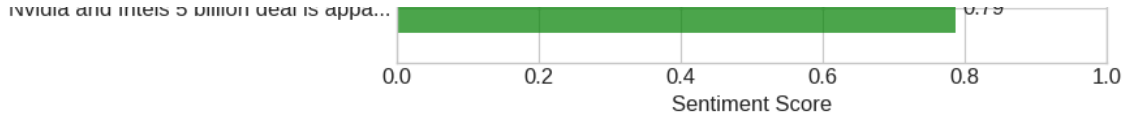

Price History - ORCL
Close Price
Price ($)
220
240
260
280
300
320
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15
Relative Strength Index - ORCL
100
Overbought (70)

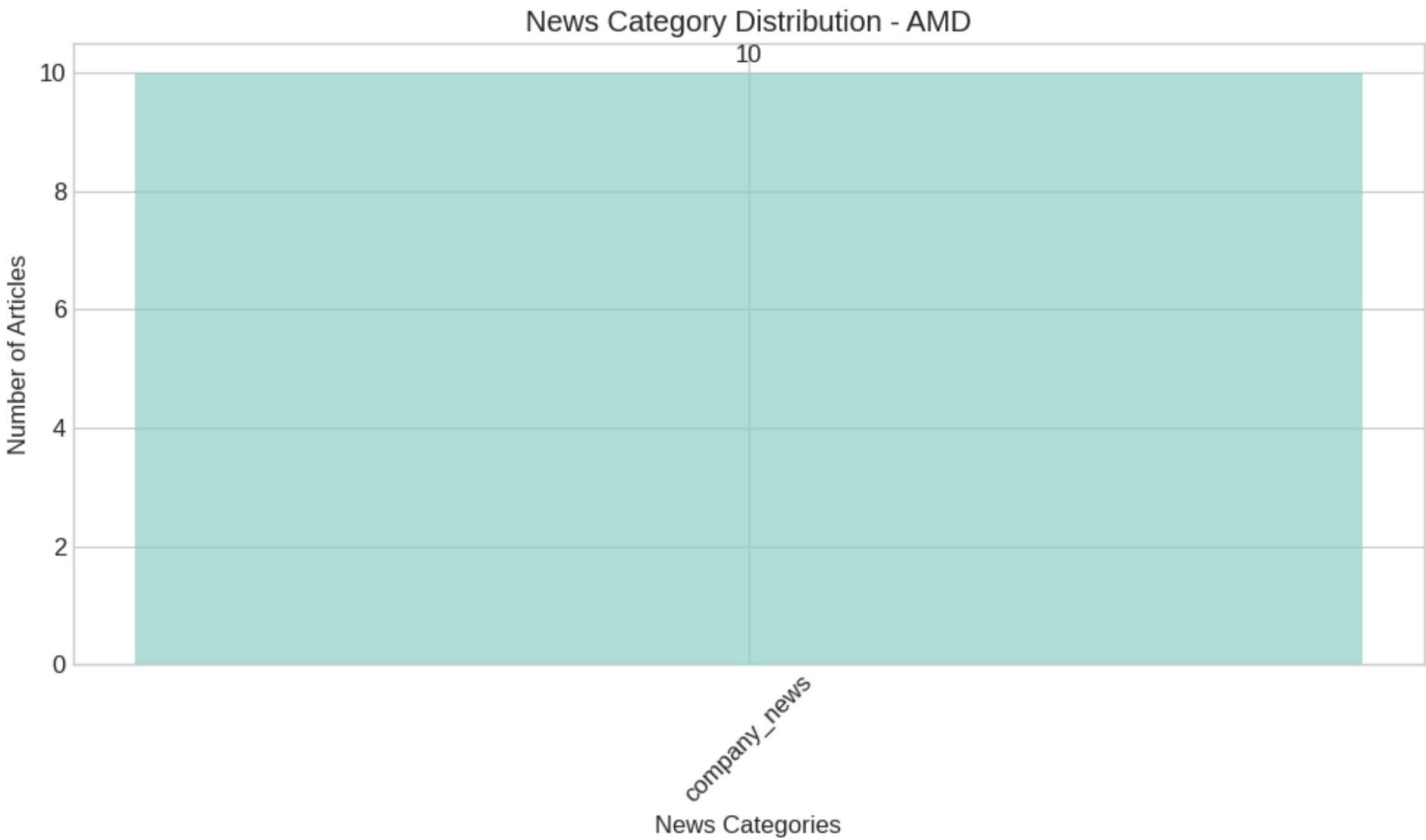


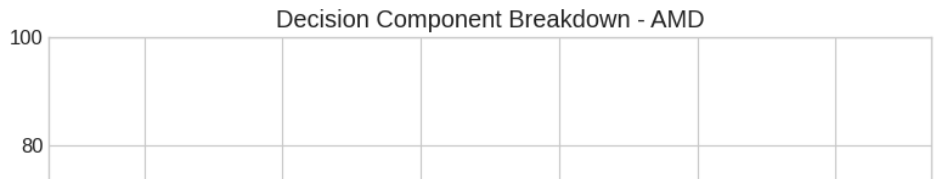
🔍 Analyzing AMD...
[2025-10-19 13:54:50] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.727, Summary Quality: 0.834
✅ AMD analysis completed
📊 Creating visualizations for AMD...

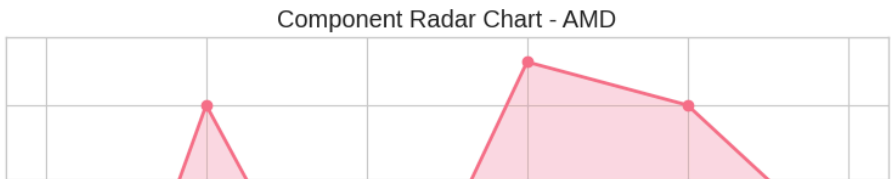


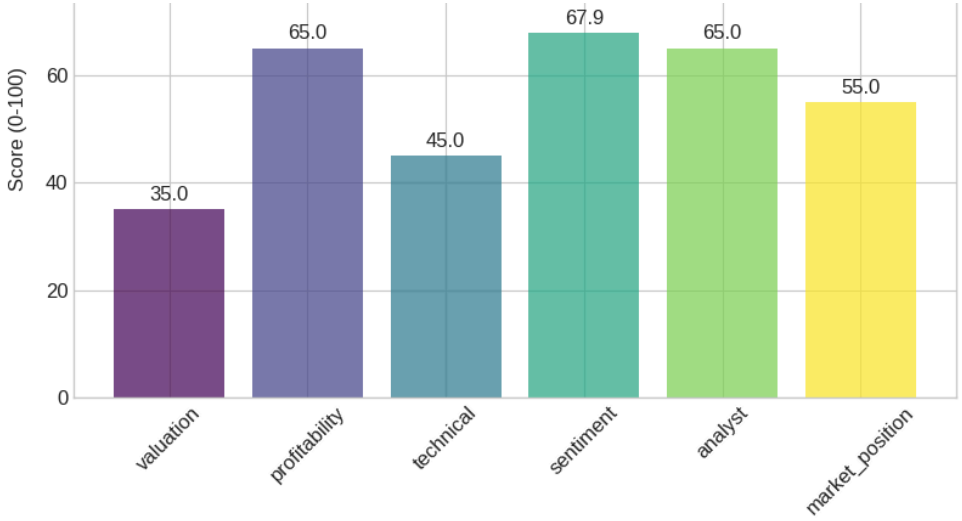


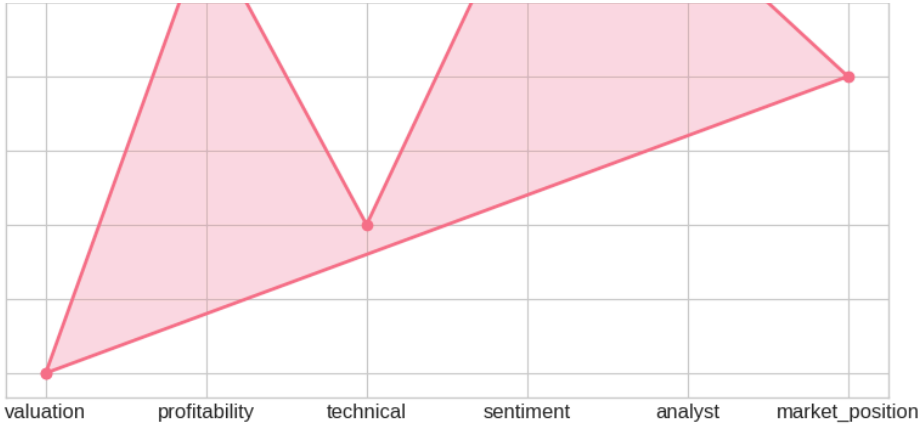

Nvidia and Intel's $5 billion deal is appa...
0.79
0.0
0.2
0.4
0.6
0.8
1.0
Sentiment Score

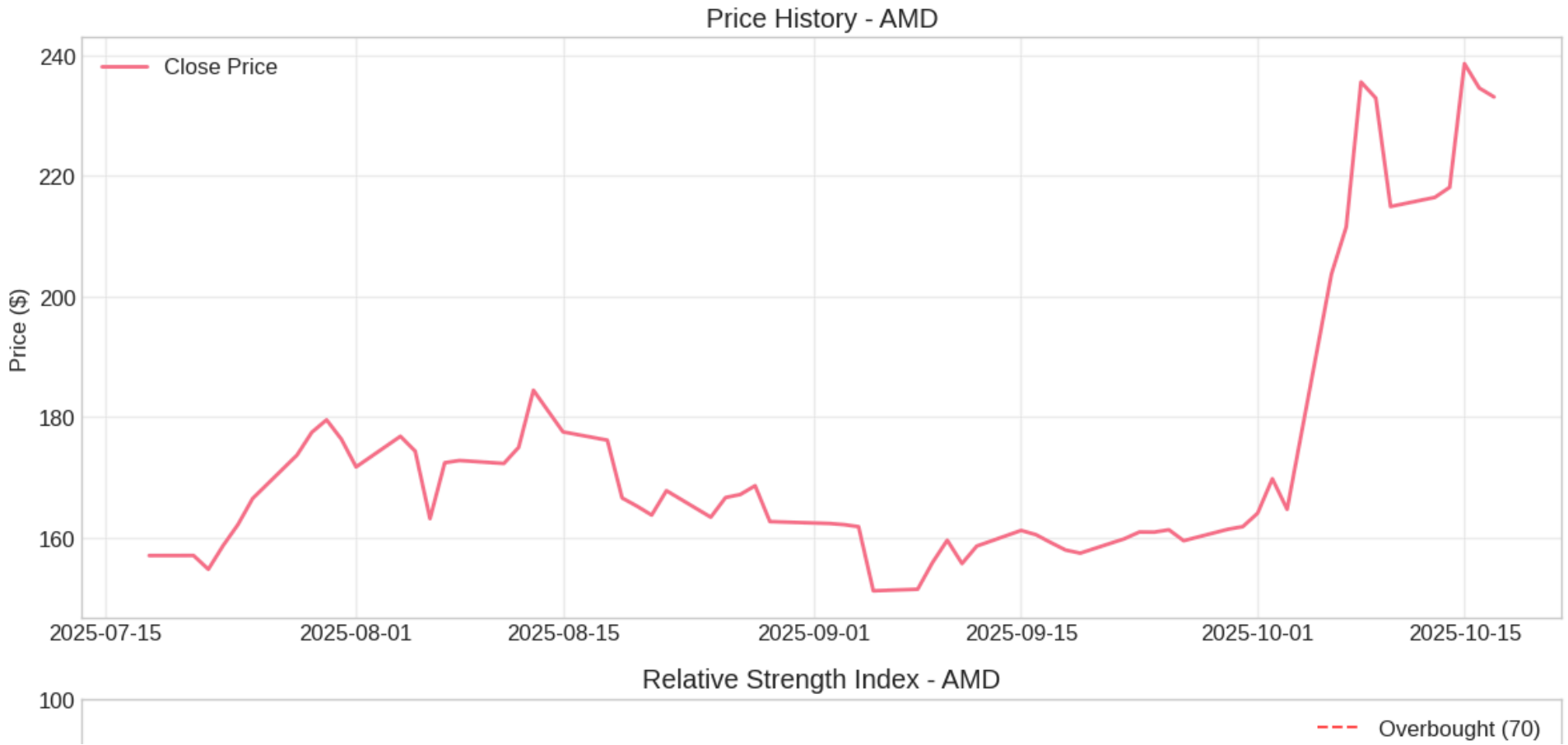

News Category Distribution - AMD
10
10
8
6
4
2
0
Number of Articles
company_news
News Categories

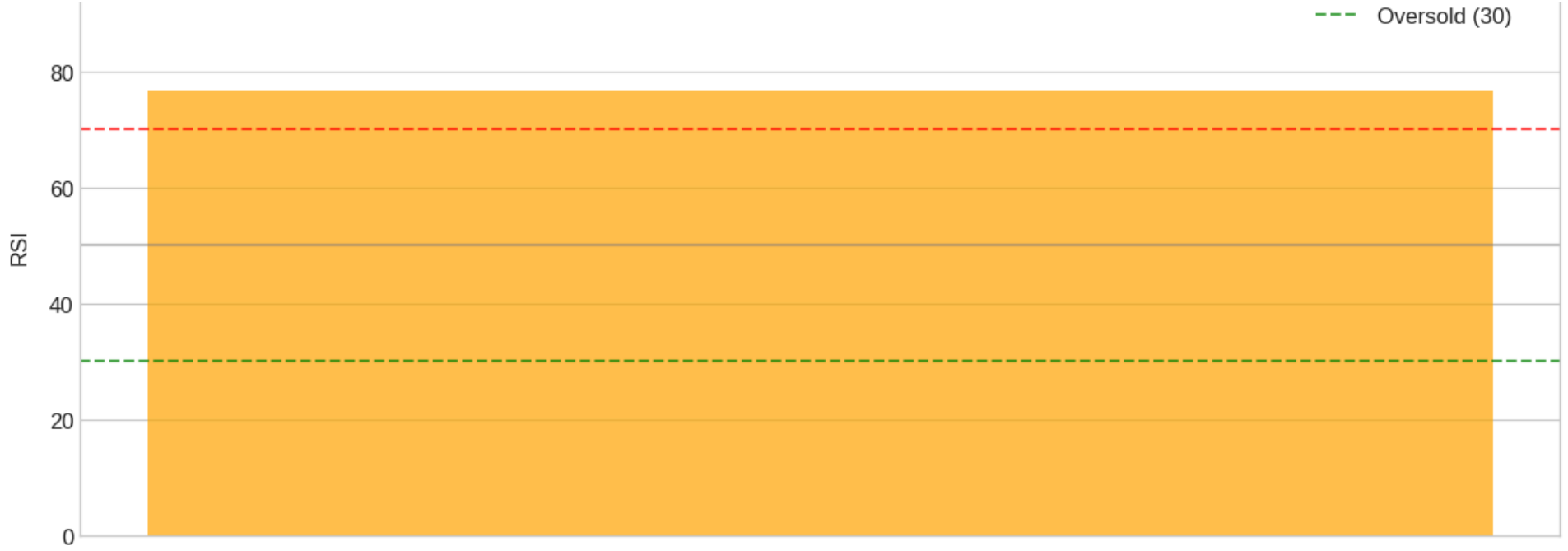

Decision Component Breakdown - AMD
100
80


Component Radar Chart - AMD

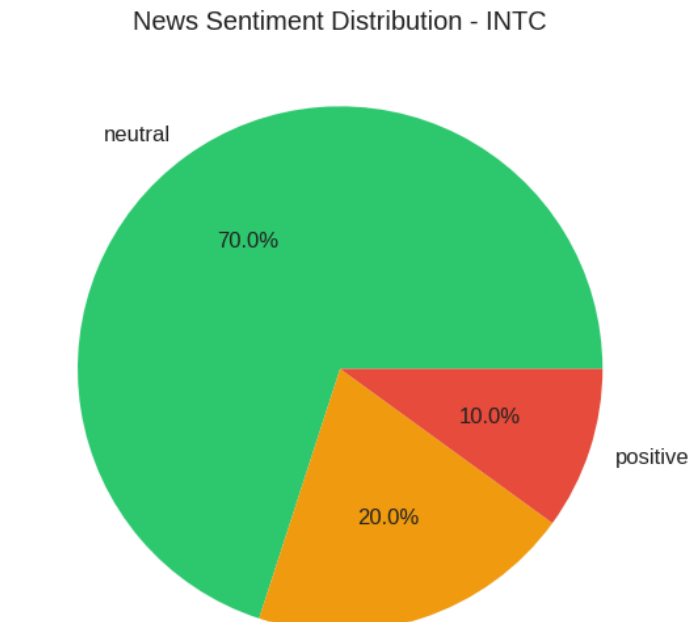

Score (0-100)
0
20
40
60
35.0
65.0
45.0
67.9
65.0
55.0
valuation
profitability
technical
sentiment
analyst
market_position

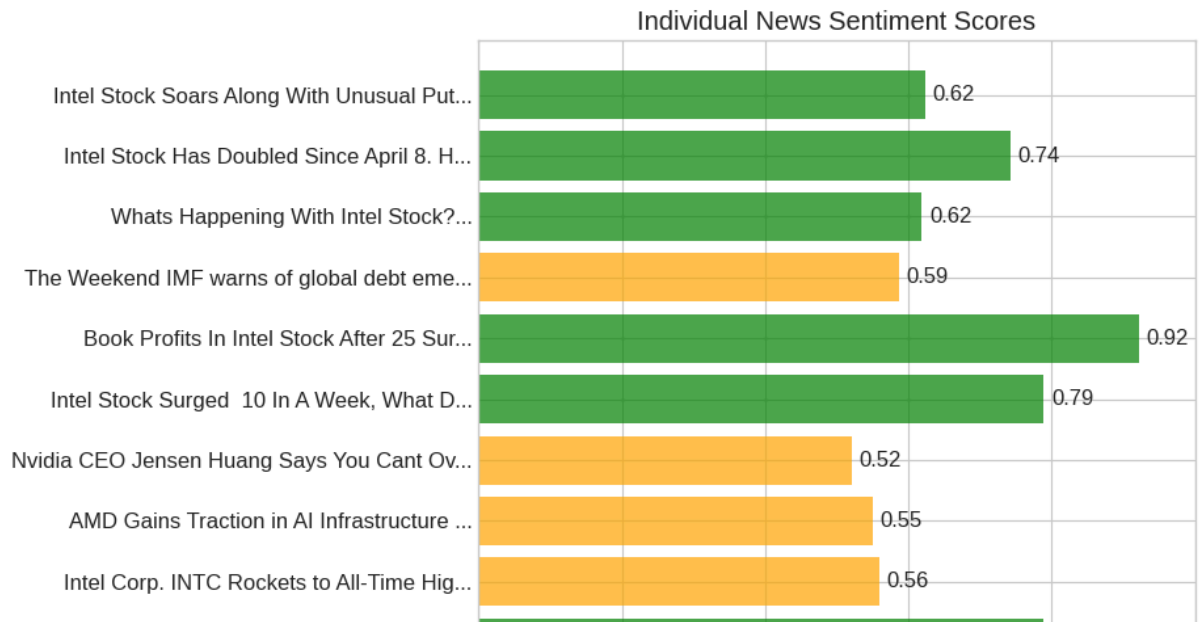

valuation
profitability
technical
sentiment
analyst
market_position


Price History - AMD
Close Price
Price ($)
240
220
200
180
160
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15
Relative Strength Index - AMD
100
Overbought (70)

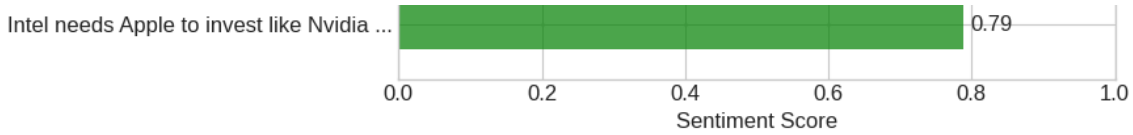


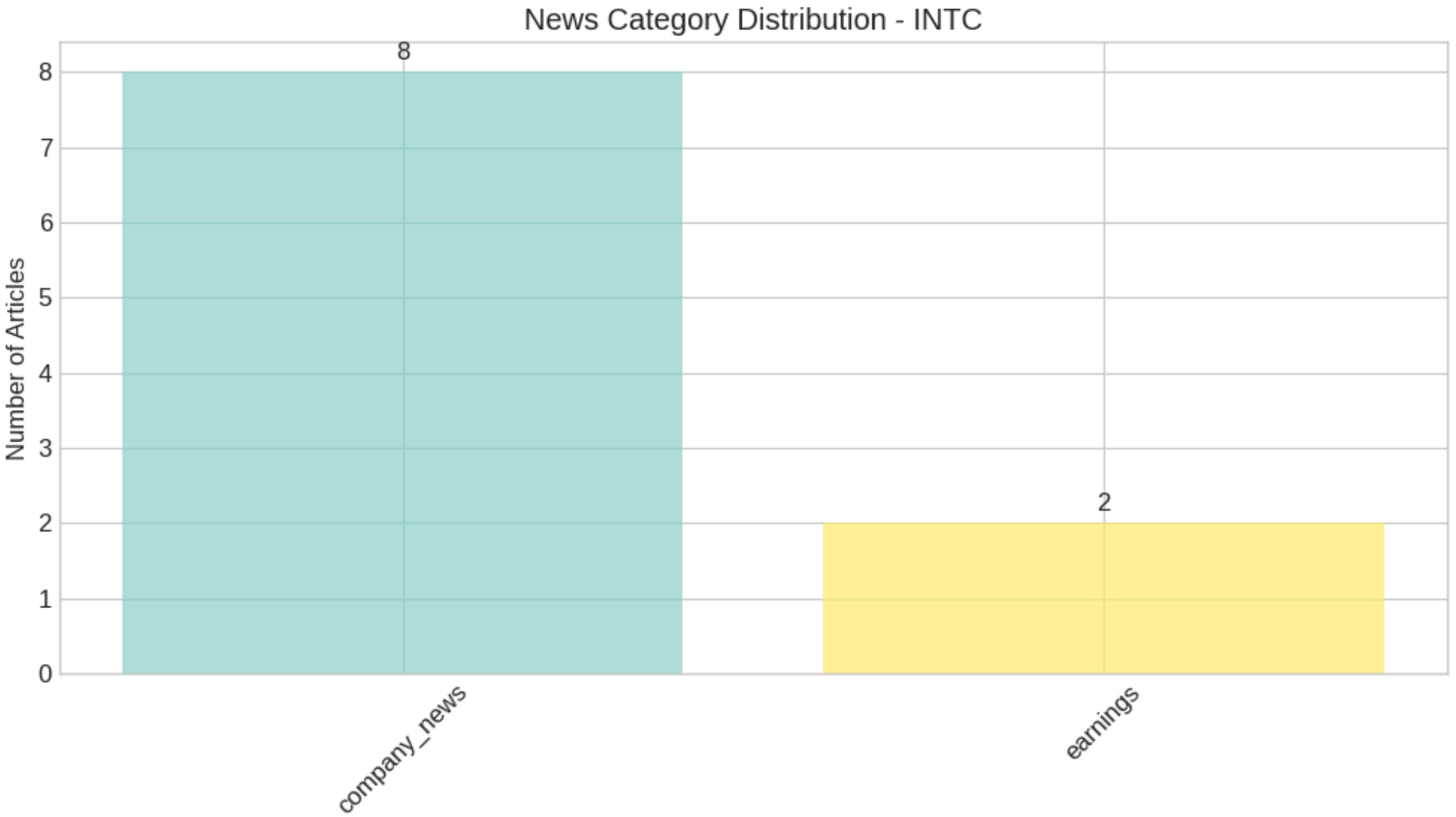
🔍 Analyzing INTC...
[2025-10-19 13:54:55] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.724, Summary Quality: 0.831
✅ INTC analysis completed
📊 Creating visualizations for INTC...

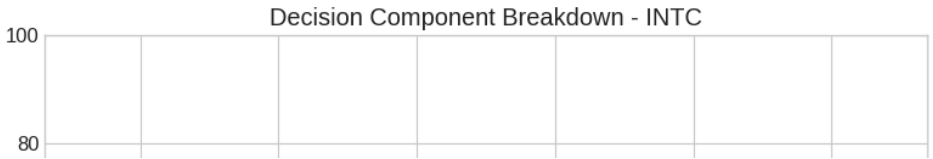


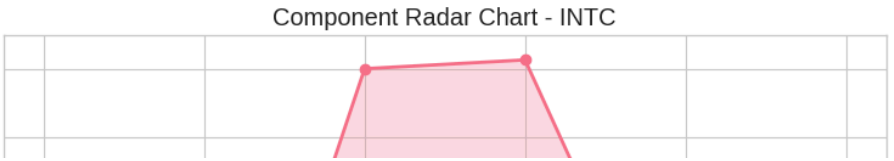


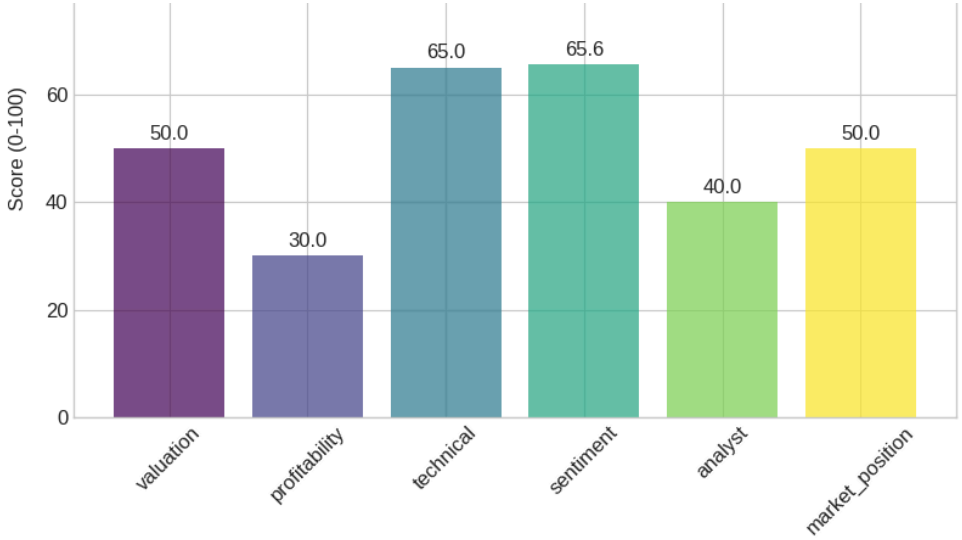

negative

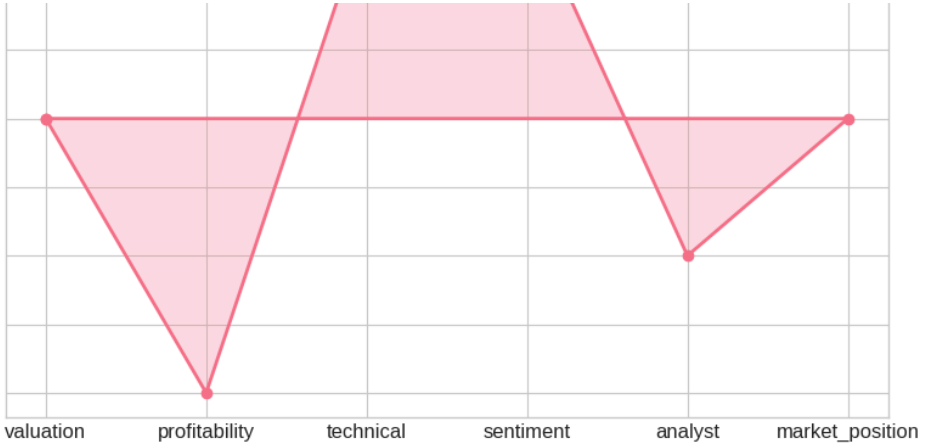

Intel needs Apple to invest like Nvidia ...
0.79
0.0
0.2
0.4
0.6
0.8
1.0
Sentiment Score


News Category Distribution - INTC
Number of Articles
0
1
2
3
4
5
6
7
8
8
2
company_news
earnings
News Categories

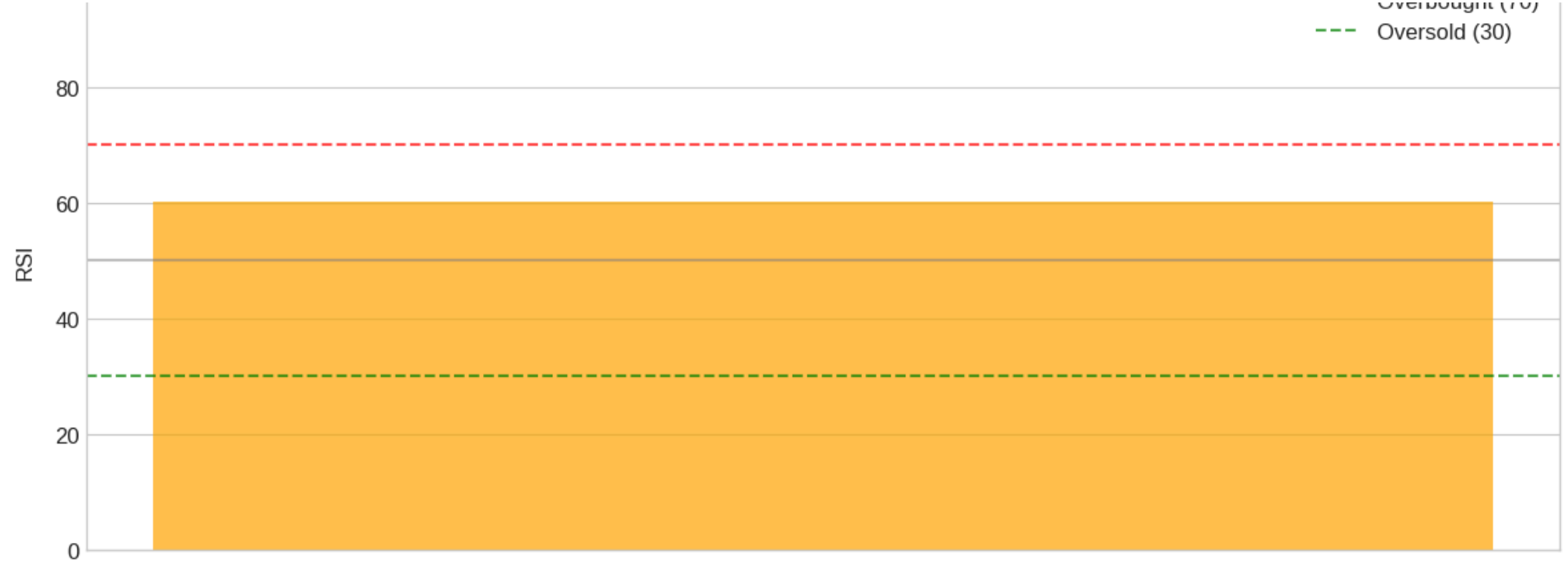

Decision Component Breakdown - INTC
100
80


Component Radar Chart - INTC

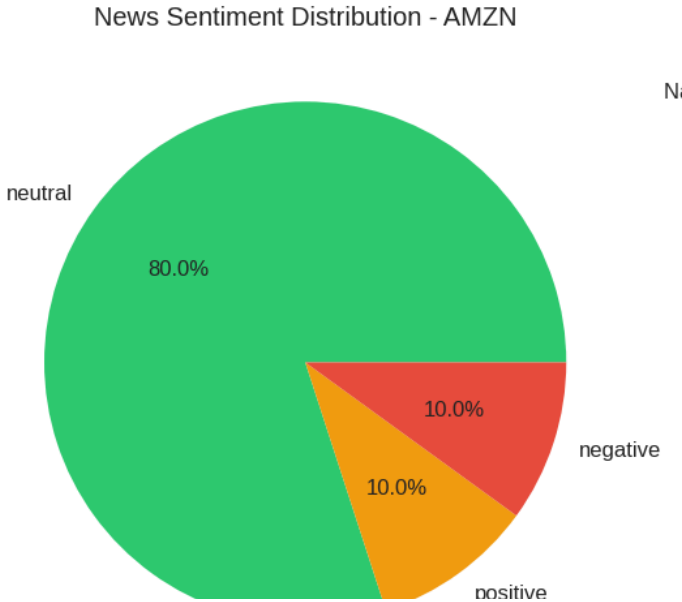

Score (0-100)
0
20
40
60
50.0
30.0
65.0
65.6
40.0
50.0
valuation
profitability
technical
sentiment
analyst
market_position

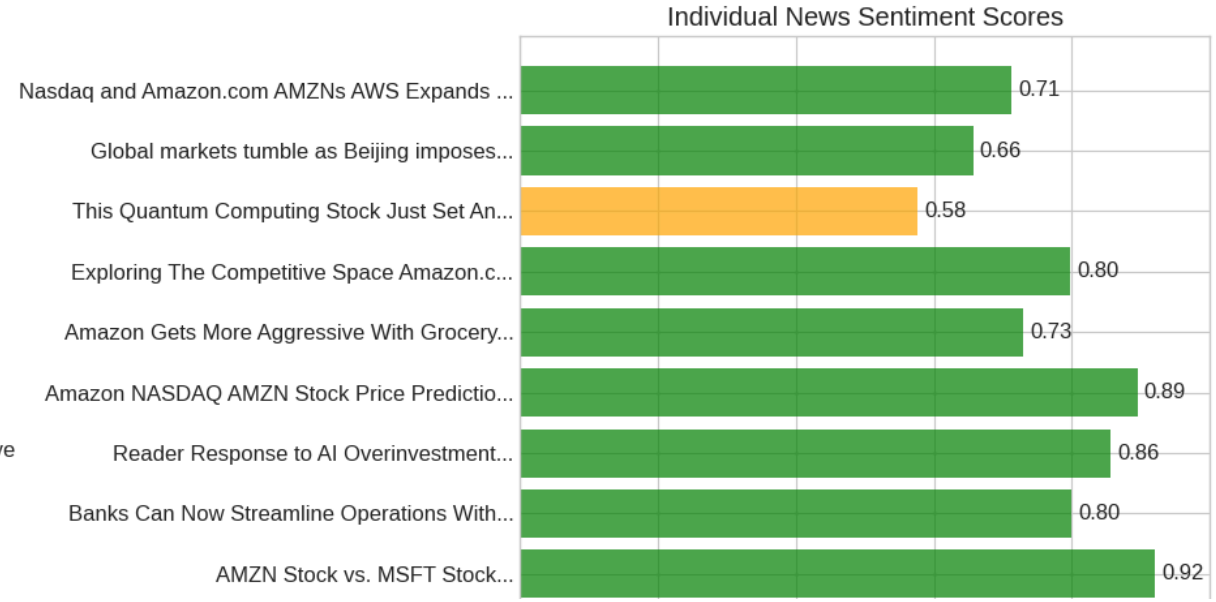

valuation
profitability
technical
sentiment
analyst
market_position

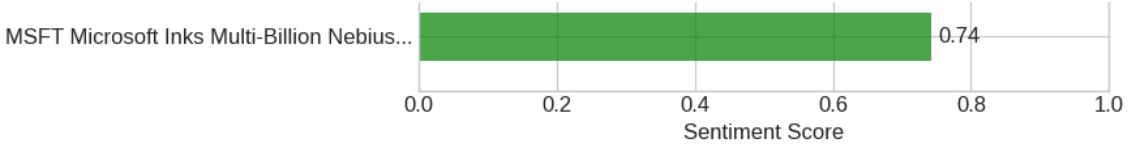

Price History - INTC
Close Price
Price ($)
37.5
35.0
32.5
30.0
27.5
25.0
22.5
20.0
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15
Relative Strength Index - INTC
100
Overbought (70)

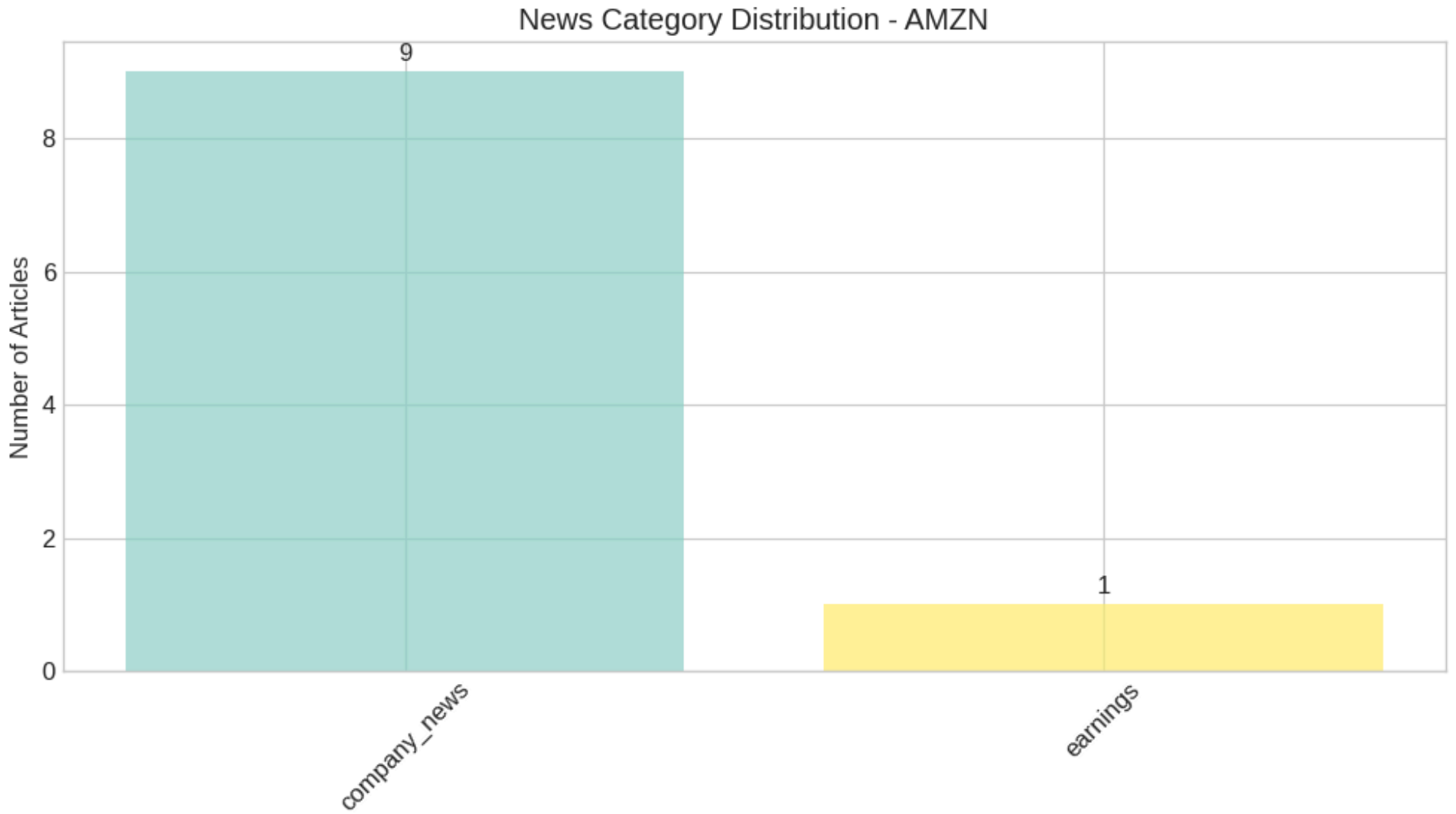


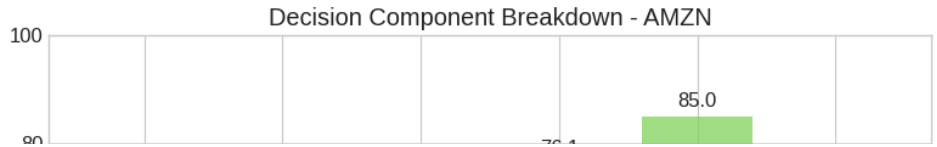
🔍 Analyzing AMZN...
[2025-10-19 13:55:01] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.726, Summary Quality: 0.831
✅ AMZN analysis completed
📊 Creating visualizations for AMZN...

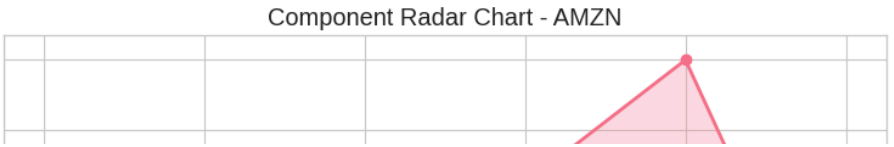


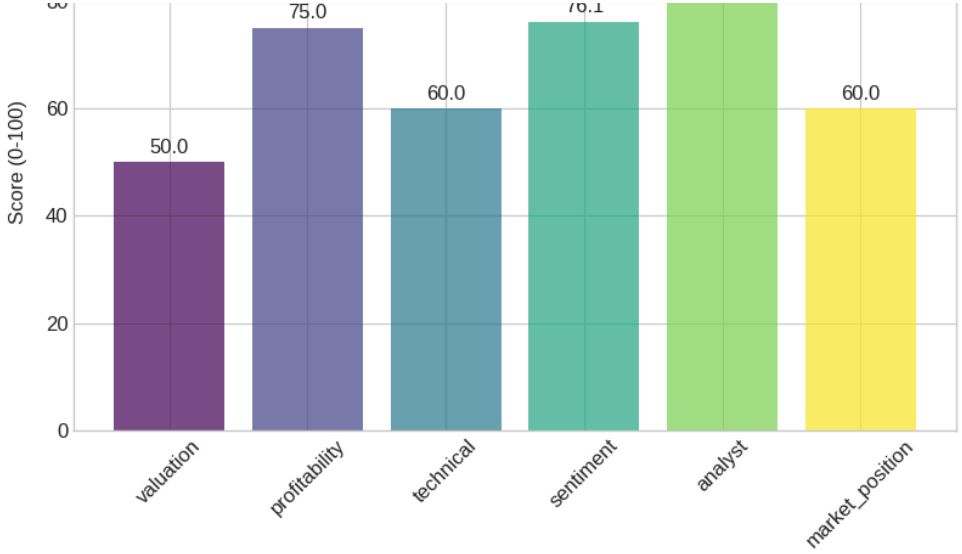


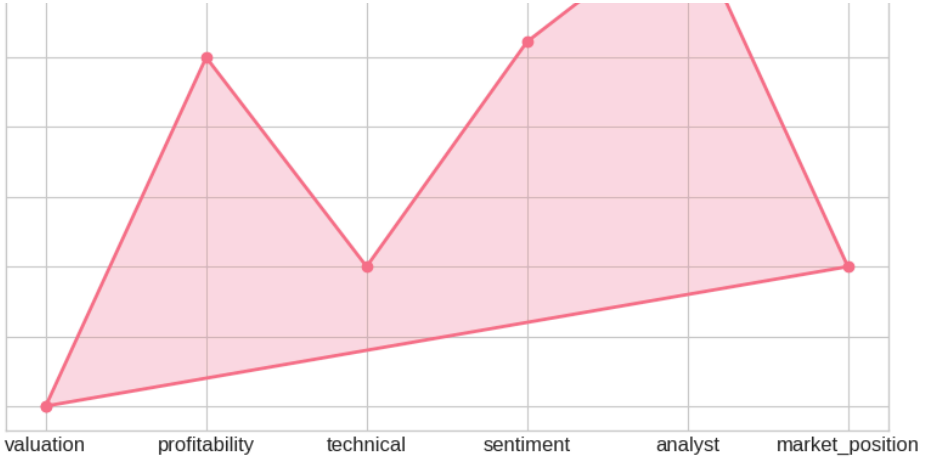

MSFT Microsoft Inks Multi-Billion Nebius...
0.74
0.0
0.2
0.4
0.6
0.8
1.0
Sentiment Score

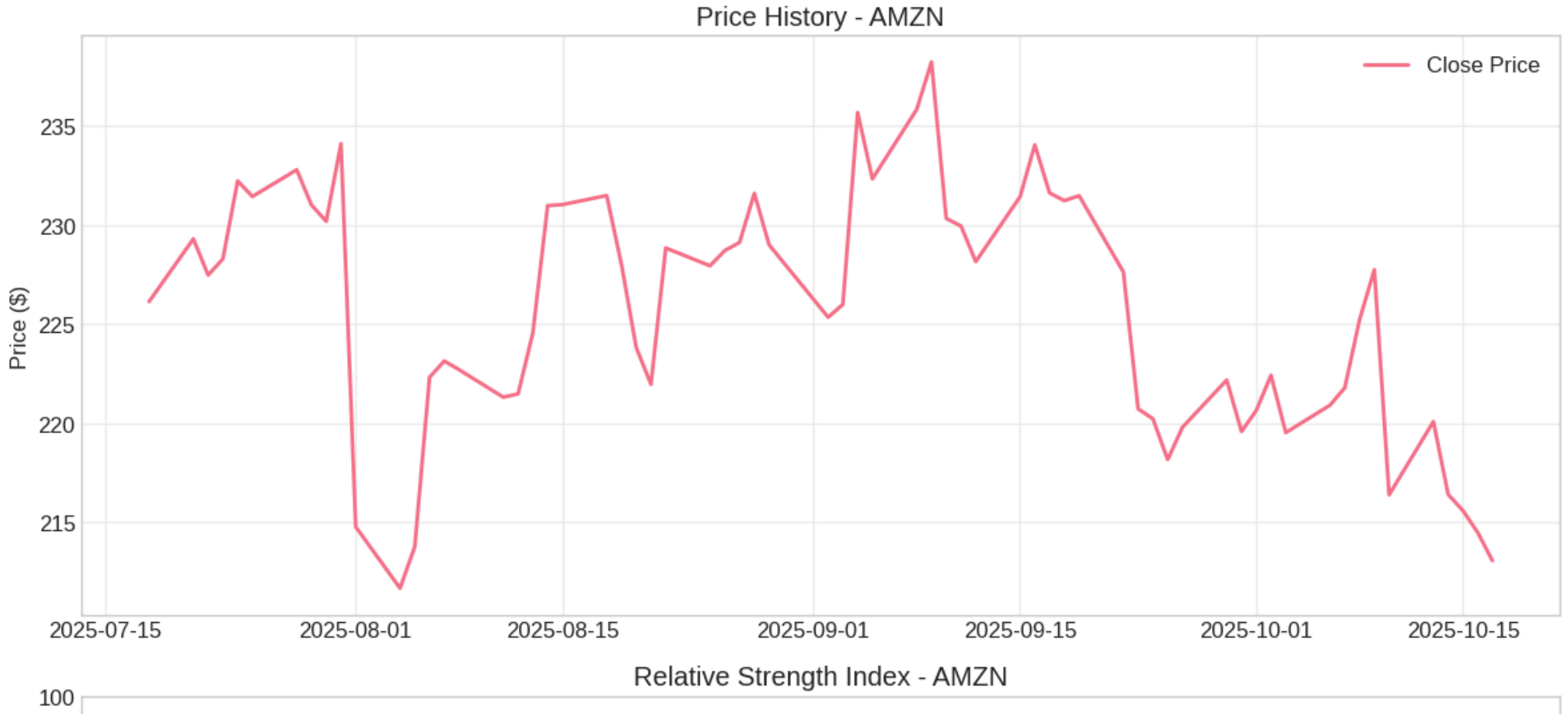

News Category Distribution - AMZN
Number of Articles
0
2
4
6
8
9
1
company_news
earnings
News Categories

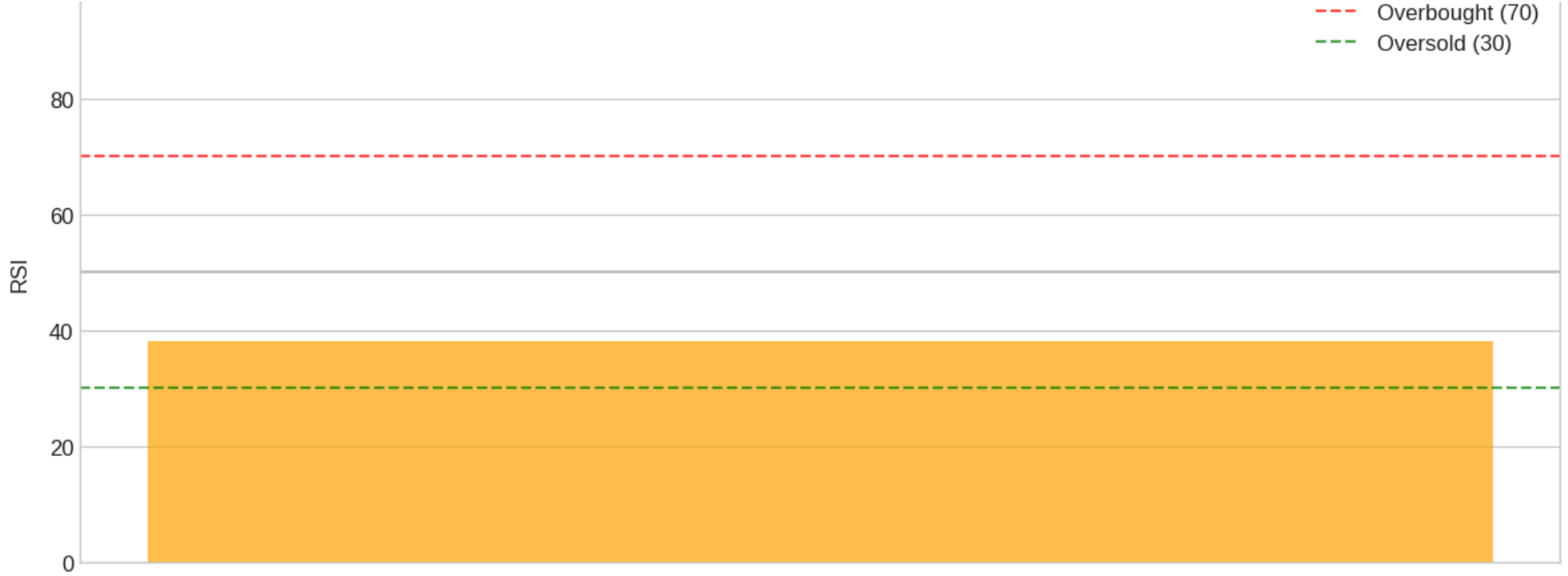

Decision Component Breakdown - AMZN
100
85.0


Component Radar Chart - AMZN

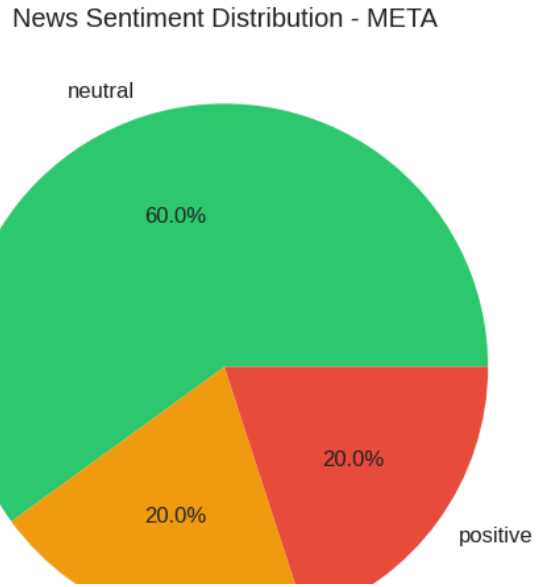

Score (0-100)
0
20
40
60
50.0
75.0
60.0
76.1
60.0
valuation
profitability
technical
sentiment
analyst
market_position

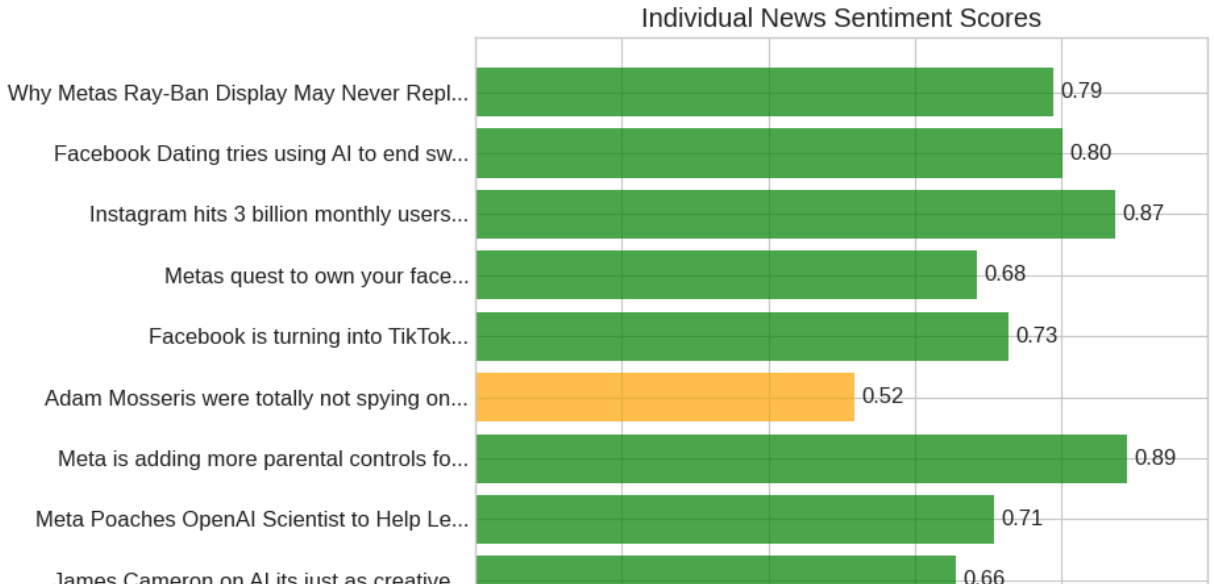

valuation
profitability
technical
sentiment
analyst
market_position


Price History - AMZN
Close Price
Price ($)
235
230
225
220
215
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15
Relative Strength Index - AMZN
100

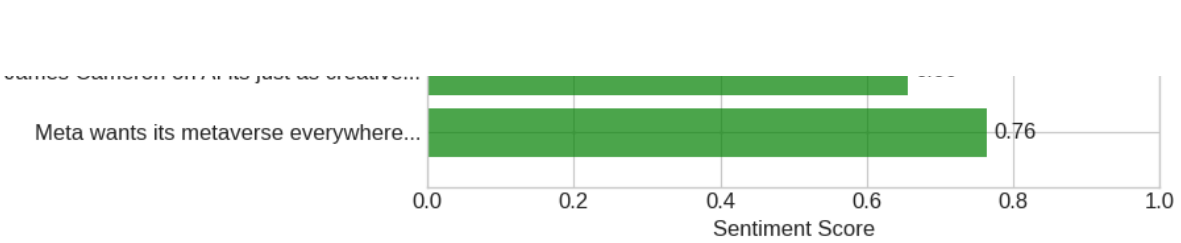


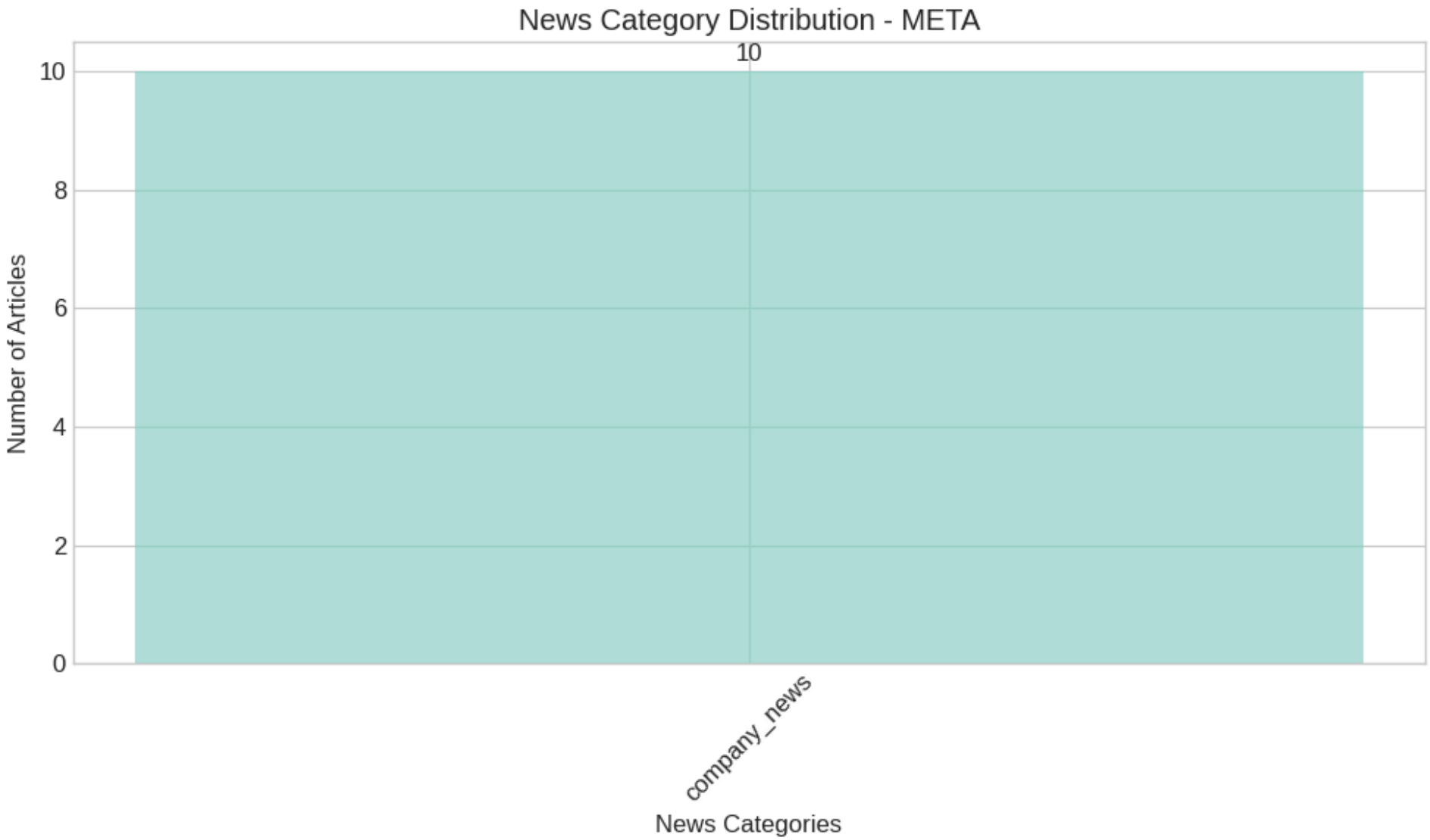
🔍 Analyzing META...
[2025-10-19 13:55:07] NewsAgent (INFO): Quality Assessment - Sentiment Confidence: 0.726, Summary Quality: 0.832
✅ META analysis completed
📊 Creating visualizations for META...



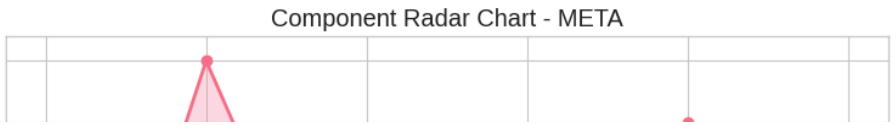


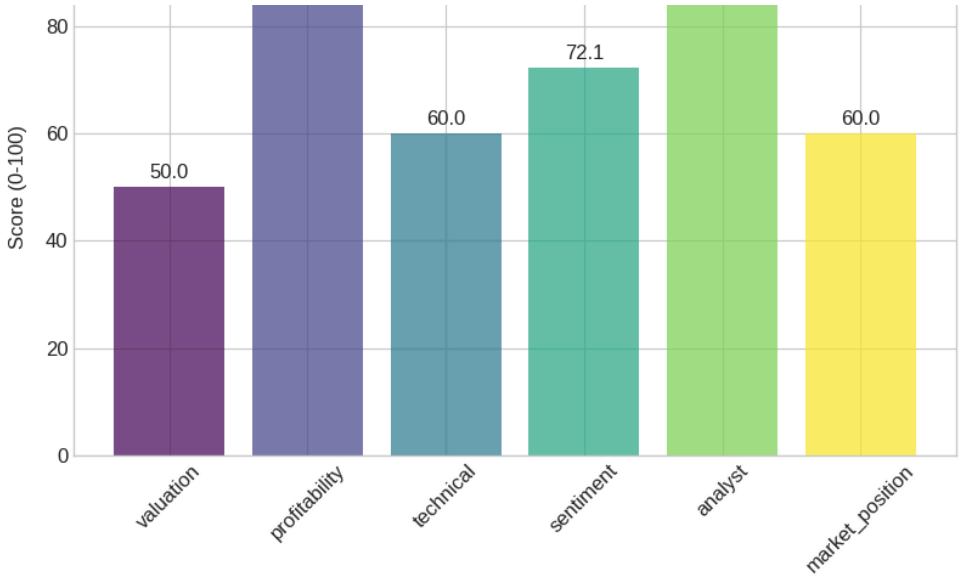

negative

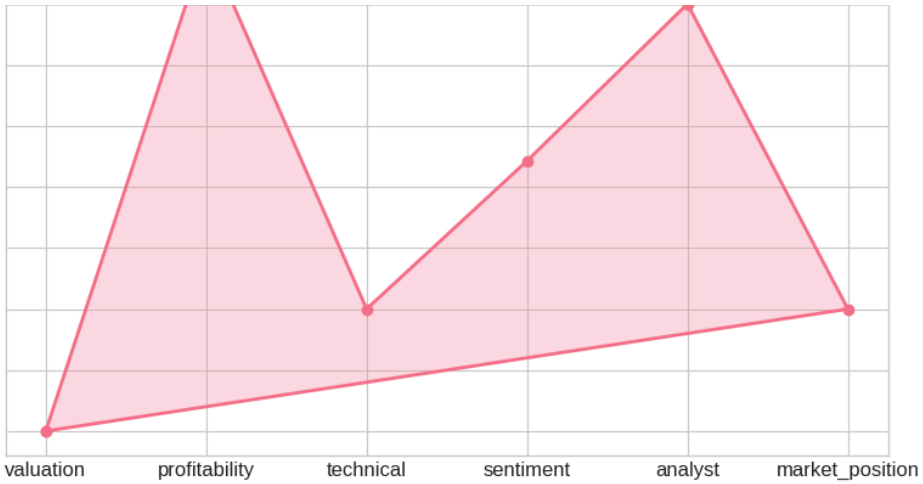

Meta wants its metaverse everywhere...
0.76
0.0
0.2
0.4
0.6
0.8
1.0
Sentiment Score

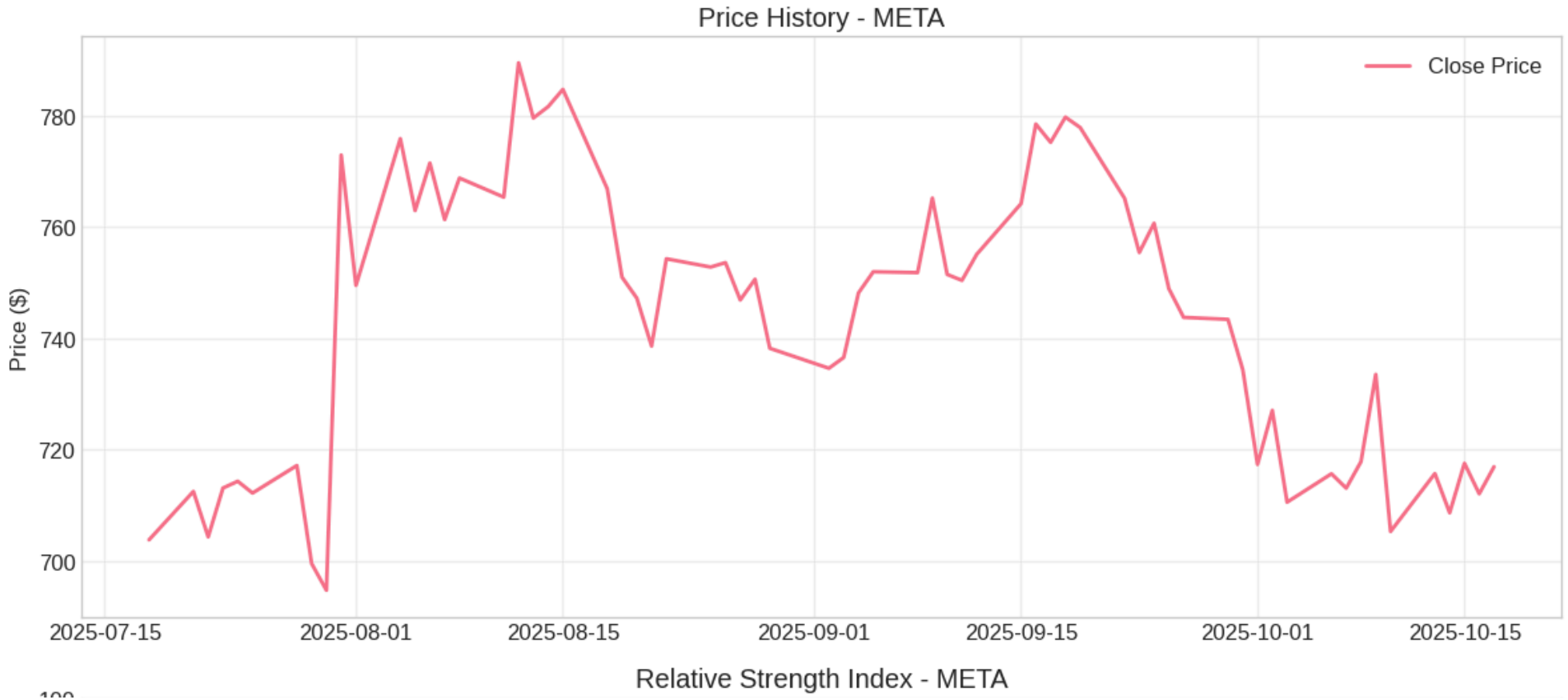

News Category Distribution - META
10
10
8
6
4
2
0
Number of Articles
company_news
News Categories

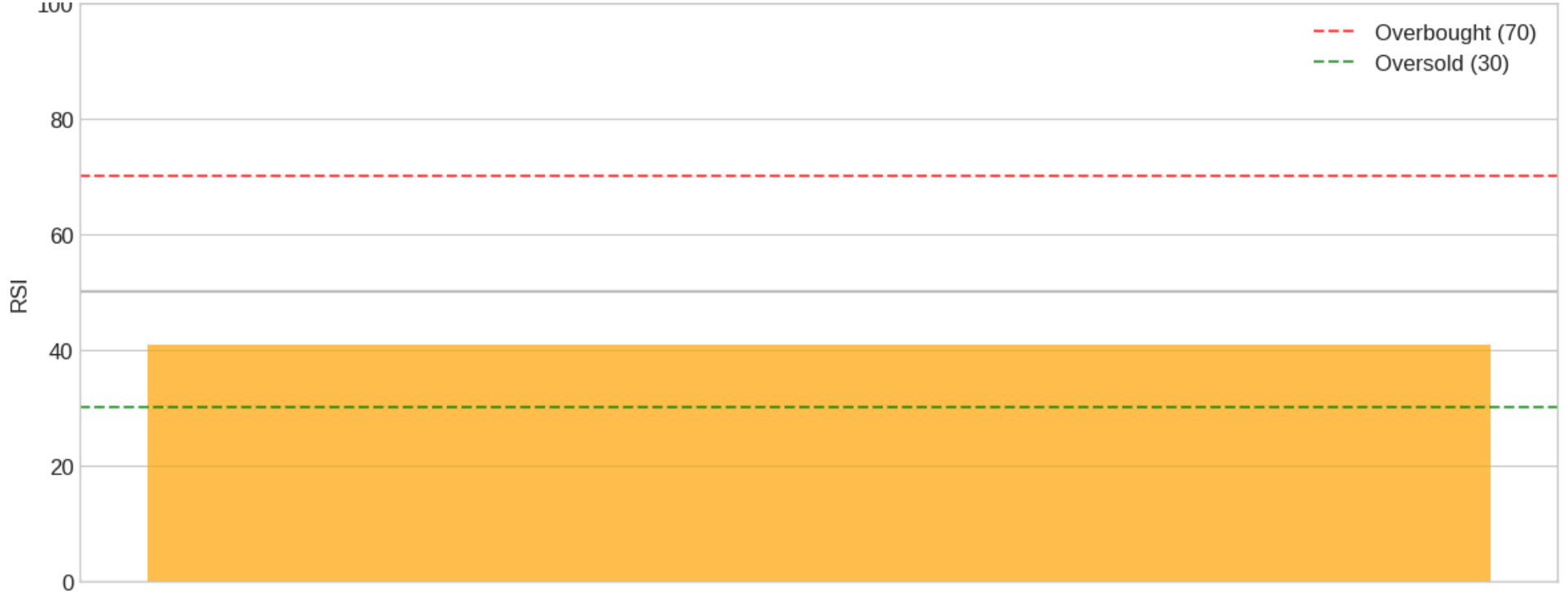

Decision Component Breakdown - META
100
90.0
85.0


Component Radar Chart - META

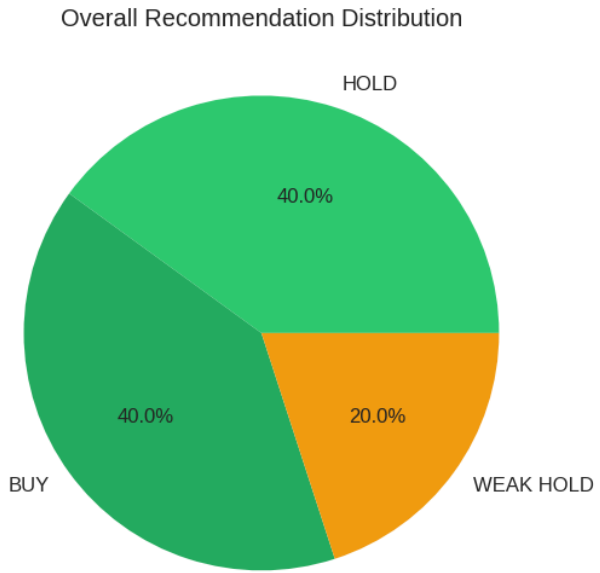

Score (0-100)
0
20
40
60
80
50.0
72.1
60.0
60.0
valuation
profitability
technical
sentiment
analyst
market_position

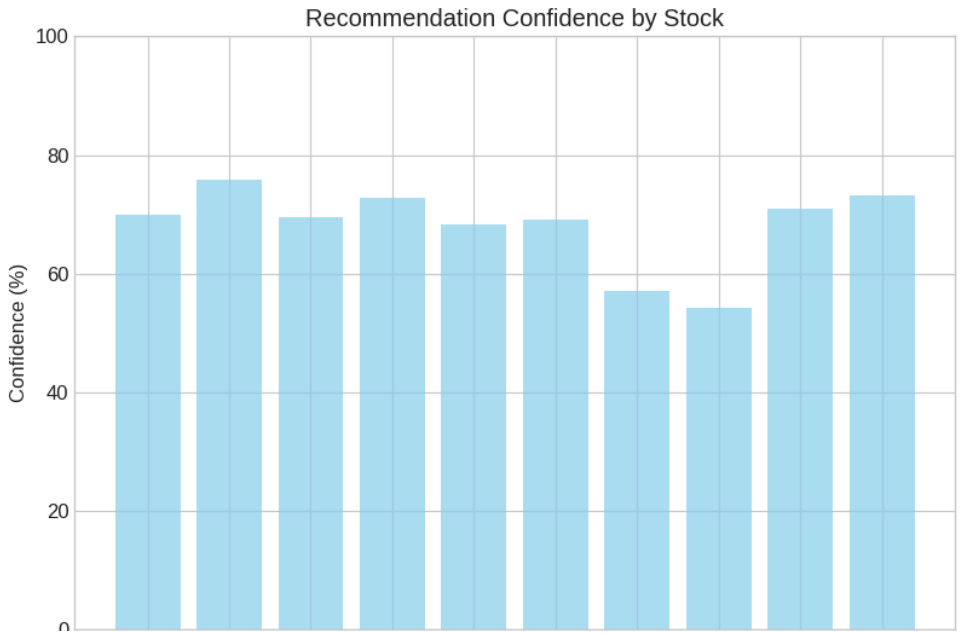

valuation
profitability
technical
sentiment
analyst
market_position

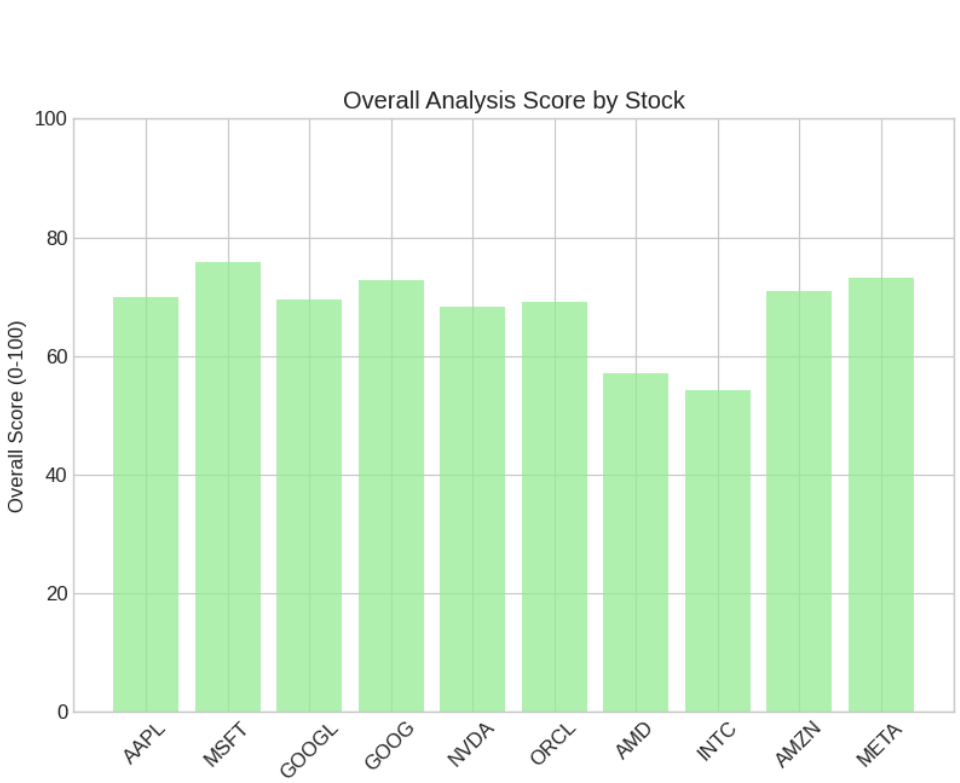

Price History - META
Close Price
Price ($)
780
760
740
720
700
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15
2025-10-01
2025-10-15
Relative Strength Index - META

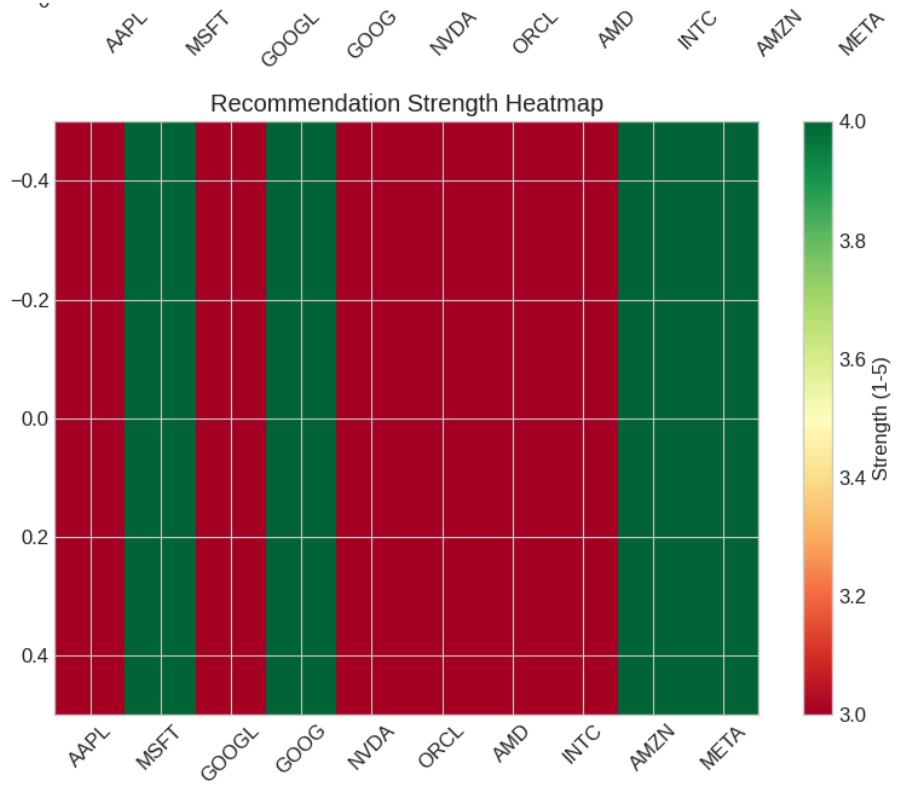


📈 Creating summary visualizations...









Symbol	Price	P/E	Fwd P/E	Market Cap	RSI	Latest News	Sentiment	Target	Ana
AAPL	$252.29	38.3	30.4	$3.7T	46.5	Apple AAPL Sees High Demand for Affor...	Neutral	$248.12	Buy
MSFT	$513.58	37.6	34.4	$3.8T	49.0	Stocks to Gain From Quantum Computing...	Neutral	$621.03	Str
GOOGL	$253.30	27.0	28.3	$3.1T	61.7	Why Is GOOGL Stock Rising?	Neutral	$252.73	Buy
GOOG	$253.79	27.0	28.4	$3.1T	62.1	Google just erased 7 years of our pol...	Neutral	$246.09	Buy
NVDA	$183.22	52.1	44.5	$4.5T	51.5	Intel needs Apple to invest like Nvid...	Neutral	$218.51	Str
ORCL	$291.31	67.6	40.7	$830.5B	54.4	Is ORCL Stock A Winner?	Neutral	$345.19	Buy
AMD	$233.08	138.7	45.7	$378.3B	76.7	Nvidia and Intels 5 billion deal is a...	Neutral	$239.11	Buy
INTC	$37.01	0.0	38.2	$176.1B	60.1	Intel needs Apple to invest like Nvid...	Neutral	$27.72	Hol
AMZN	$213.04	32.5	34.6	$2.3T	38.2	MSFT Microsoft Inks Multi-Billion Neb...	Neutral	$266.56	Str
META	$716.91	26.0	28.3	$1.8T	40.9	Meta wants its metaverse everywhere	Neutral	$868.26	Str

📋 DETAILED REASONING:

🔍 AAPL: High valuation with elevated P/E ratio; Strong profit margins

🔍 MSFT: High valuation with elevated P/E ratio; Strong profit margins; Analyst target suggests 20.9% upside

🔍 GOOGL: Strong profit margins

🔍 GOOG: Strong profit margins

🔍 NVDA: High valuation with elevated P/E ratio; Strong profit margins; Analyst target suggests 19.3% upside

🔍 ORCL: High valuation with elevated P/E ratio; Strong profit margins; Analyst target suggests 18.5% upside

🔍 AMD: High valuation with elevated P/E ratio; Overbought technical condition; Analyst target suggests 2.6% upside

🔍 INTC: Attractive valuation with low P/E ratio; Negative profit margins

🔍 AMZN: Analyst target suggests 25.1% upside

🔍 META: Strong profit margins; Analyst target suggests 21.1% upside

📈📈
ANALYSIS SUMMARY
📈📈

📊 Analysis Completion: 10/10 stocks successfully analyzed

🎯 Recommendation Distribution:
HOLD: 4 stocks (40.0%)
BUY: 4 stocks (40.0%)
WEAK HOLD: 2 stocks (20.0%)

✅ Analysis completed successfully!
💡 Check the generated visualizations for detailed insights

SECTION 12.1: OUTPUT DESCRIPTION AAND ANALYSIS

The detailed analysis of the chart outputs for Apple (AAPL) is provided below.

Introduction to Output Analysis

The following graphical analysis summarizes the findings of the multi-agent financial analysis system. The system was tasked with a comprehensive evaluation of a predefined list of 10 major technology stocks: **AAPL** (Apple), **MSFT** (Microsoft), **GOOGL** (Alphabet A), **GOOG** (Alphabet C), **NVDA** (NVIDIA), **ORCL** (Oracle), **AMD** (Advanced Micro Devices), **INTC** (Intel), **AMZN** (Amazon), and **META** (Meta Platforms).

For each stock, the system's `VisualizationEngine` generates a standardized suite of charts based on the processed data from the `DataAgent`, `NewsAgent`, and `AnalysisAgent`. This suite includes:

1. **Decision Component Breakdown** (Bar Chart)
2. **Component Radar Chart**
3. **Price History** (Line Chart)
4. **Relative Strength Index (RSI)** (Gauge Chart)
5. **News Category Distribution** (Bar Chart)
6. *News Sentiment Distribution (Pie Chart)*
7. *Individual News Sentiment Scores (Bar Chart)*

Chart 1: News Sentiment Distribution - AAPL (The Macro View)

- **What it is:** This pie chart shows the **aggregated sentiment** of all news articles processed for Apple.
- **Analysis:** The chart indicates a perfectly balanced sentiment distribution among the analyzed articles:
    - **50.0% Positive (Green):** Half of the news items were classified as having a positive sentiment.
    - **50.0% Neutral (Orange):** The other half were classified as neutral.

    - **Noteworthy Absence:** There were **no articles classified as "Negative"** in this batch.
- **Agent Contribution:** This chart visualizes the *summary* that the `AnalysisAgent` would compute. The `AnalysisAgent` takes the full list of processed articles from the `NewsAgent`, counts the occurrences of each sentiment label ("positive", "neutral", "negative"), and calculates these percentage breakdowns.

Chart 2: Individual News Sentiment Scores (The Micro View)

- **What it is:** This horizontal bar chart provides a detailed, **article-by-article breakdown** of the sentiment scores that were aggregated to create the pie chart.
- **Analysis:** It plots each news headline against its calculated "Sentiment Score" (from 0.0 to 1.0).
    - **`NewsAgent`'s Core Output:** This is a direct visualization of the output from the `NewsAgent`'s `_classify_with_llm` and `_apply_financial_context_adjustments` methods. Each bar represents one processed news item.
    - **High-Scoring Positive (Green) Articles:** Headlines like "Apples new iPhone 17 is available..." and "Apple AAPL Sees High Demand..." received perfect 1.00 scores, indicating the `NewsAgent`'s high confidence in their positive nature.
    - **Mid-Scoring Neutral (Orange) Articles:** Headlines like "Apple Inc. Sued Over Workplace Disc..." (0.56) and "Nvidia CEO Jensen Huang Says..." (0.52) received scores close to 0.5, which the `AnalysisAgent` (or the underlying `NewsAgent`'s classification) correctly interprets as "Neutral".
    - **Connecting the Charts:**This bar chart feeds the pie chart: there are 5 green bars (Positive) and 5 orange bars (Neutral), leading to the 50/50 split.

Overall Insight from the Agents' Output as per Charts (1 and 2) on News Sentiment

This visualization powerfully combines the high-level summary from the **`AnalysisAgent`** with the granular evidence from the **`NewsAgent`**.

A user can instantly see the **"What"** (50/50 positive/neutral split) from the pie chart and then immediately investigate the **"Why"** by looking at the bar chart to see *which specific articles* were positive (e.g., "High Demand") and which were neutral (e.g., "Sued Over Workplace Disc."). This provides both a quick summary and the necessary detail for a human to validate the AI's findings.

Chart 3. News Category Distribution

This chart shows the output of the **`NewsAgent`'s Routing** pattern (`_route_to_specialists` function).

- **Analysis:** Out of the 10 articles processed for AAPL, the `NewsAgent` successfully categorized 8 as "company_news" (e.g., product updates, partnerships, legal news) and 2 as "earnings" (related to financial results).
- **Significance:** This demonstrates the agent's ability to differentiate news types, ensuring that earnings reports (which are highly significant) are tagged appropriately. This classification directly feeds the `AnalysisAgent`'s context

Chart 4. Decision Component & Radar Charts

These two charts visualize the final scores (0-100) calculated by the **`AnalysisAgent`** for each key factor.

- **Analysis:** The charts reveal a classic "tug-of-war" scenario. The strongest drivers for Apple are its **Profitability (75.0)** and **Sentiment (76.1)**. This indicates that the `DataAgent` found strong profit margins and revenue growth, while the `NewsAgent`'s LLM-driven analysis of recent news was overwhelmingly positive.
- **Balancing Factor:** However, these strengths are being moderated by a mediocre **Valuation score (55.0)**, which is the lowest component. This suggests the `DataAgent` determined that AAPL's stock price is high relative to its earnings (P/E ratio), making it less attractive from a value perspective.
- **Other Components:** The Technical (60.0), Analyst (65.0), and Market Position (60.0) scores are all neutral-to-positive, providing a stable floor but not driving a strong "buy" signal. This combination leads to the balanced **"HOLD"** recommendation seen in the summary table.

Chart 5. Price History & Relative Strength Index (RSI)

These charts are generated from data fetched and calculated by the **`DataAgent`**.

- **Price History:** The line chart shows AAPL's stock price over the last three months (approx. July-Oct 2025). It displays a clear uptrend, peaking around the beginning of October before a recent, minor pullback.
- **RSI:** The gauge chart) shows the current RSI value is approximately **46.5**. This value is firmly in the neutral territory (between the "Oversold" threshold of 30 and the "Overbought" threshold of 70).

- **Combined Insight:** The technical picture supports the `AnalysisAgent`'s neutral **Technical score (60.0)**. The stock is neither over-extended nor oversold; it is in a "holding" pattern after its recent run-up, justifying a neutral technical stance.

SECTION 12.1: MULTI-AGENT WORK FLOW AND FINAL TABLE OUTPUT DESCRIPTION OF MULTI-AGENT FINANCIAL SYSTEM:

Final Output Recommendation of Stock: Multi-Agent Workflow

The output table summarizes the final results of an **autonomous Investment Research System** where several collaborative agents interact to generate actionable financial recommendations. Below, each agent's contribution and function in the pipeline are highlighted.

A. FINANCIAL ANALYSIS SYSTEM (ORCHESTRATOR)

Role: Orchestrator, Data Flow Manager, Output Formatter

- **Initiation & Coordination:**
  Starts the analysis for each symbol (e.g., `analyze_stocks` loop).
- **Data Flow Management:**
  Passes the stock symbol to DataAgent and NewsAgent, receives their structured outputs (`stock_data`, `news_data`), and forwards them to the AnalysisAgent.
- **Result Consolidation:**
  Collects the final analysis result along with all inputs.
- **Output Formatting & Display:**
  Calls helper functions (like `format_market_cap`, `get_latest_news_title`) and prints or displays the table seen above.

B. DATA AGENT

Role: Data Ingestion, Feature Computation, Serialization

- **Data Ingestion (Tool Use):**
  Fetches raw quantitative data for a symbol using APIs such as `yf.Ticker`.
- **Technical Calculation:**
  Computes technical indicators like RSI and MACD.
- **Serialization:**
  Prepares all data in Python dictionaries (easy for further processing).
- **Output Contribution:**
  Directly provides:
  - Current Price
  - P/E Ratio and Forward P/E
  - Market Cap
  - RSI
  - Analyst Target
  - Analyst Recommendation
    for each symbol in the final output table.

C. NEWS AGENT

Role: News Ingestion, Prompt Chaining, Sentiment/Insight Generation, Routing, Internal Evaluation

1. **Data Ingestion & Preprocessing:**
   - Fetches news articles (`NewsApiClient`)
   - Cleans news text (`_clean_text`)
   - *Example*: For AAPL, "Apple unveils the new iPhone..." is the ingested and preprocessed headline.
2. **LLM Analysis (Summarization):**
   - Summarizes each article (`_extract_and_summarize`)
   - Produces a summary quality score (not shown in table, but passed along)
   - *Example*: High summary quality can increase the final overall score.

3. **LLM Analysis (Sentiment Classification):**
    - Uses transformer pipeline for sentiment
    - Applies financial context corrections and enhancements
    - Aggregates sentiment for each symbol (seen as "Sentiment" in final table)
    - *Example*: For AAPL, "Positive"; for MSFT, "Neutral".
4. **Prompt Chaining:**
    - Combines classification, context enhancement, and summarization in a chained workflow.
5. **Routing:**
    - Routes each article to domain specialists (Earnings, Market, CompanyNews analyzers)
    - Extracts key insights (used by AnalysisAgent for the reasoning string)
    - *Example*: Reasoning "Positive news sentiment" can originate here.
6. **Evaluation:**
    - Assesses quality metrics: `sentiment_confidence`, `summary_quality_score`
    - Passes these to AnalysisAgent for scoring adjustment
    - *Example*: High news analysis quality boosts scoring for AAPL.
7. **Confidence Score: Description and Derivation:**

    The Confidence score in the output table represents the system's degree of certainty in its final AI Decision (e.g., "BUY", "HOLD", "SELL"). It's essentially a numerical measure of how strongly the combined analytical factors support the given recommendation. A higher confidence score indicates that the various components analyzed (valuation, sentiment, technical, etc.) predominantly align with the final decision.

    The Confidence score is derived directly within the AnalysisAgent, specifically in the generate_decision method. This method takes the final, potentially optimized overall_score (which ranges from 0 to 100) as its input.

a. Thresholds Determine Decision:

The overall_score is first compared against predefined thresholds (80, 70, 60, 50, 40) to determine the categorical AI Decision.

b. Score Normalization and Capping:

The overall_score is normalized (divided by 100 to get a value between 0 and 1). This normalized score is then compared with a maximum confidence cap specific to that decision category (e.g., 0.95 for "STRONG BUY", 0.85 for "BUY"). The lower of these two values is chosen as the final Confidence score.

c. Output:

The method returns both the decision string and the calculated confidence score (as a float between 0 and 1), which is then formatted as a percentage in the final table. Example: • If overall_score is 75.8 , the decision is "STRONG BUY". The confidence is min(0.85.3, 75.6/100) = min(0.95, 0.758) = 0.758 • If overall_score is 57.1, the decision is "WEAK HOLD". The confidence is min(0.65, 57.1/100) = min(0.65, 0.571) = 0.571. The table shows 57.1%.

Impacted Agentic AI Section

The Confidence score is primarily generated and impacted by the AnalysisAgent. • It's a direct output of the AnalysisAgent's decision-making logic (generate_decision). • It's fundamentally tied to the overall_score, which the AnalysisAgent calculates by synthesizing inputs from the DataAgent and NewsAgent and potentially applying optimizations based on the NewsAgent's evaluation metrics (part of the Evaluator-Optimizer pattern implemented within the AnalysisAgent). While the inputs from DataAgent and NewsAgent influence the overall_score and thus indirectly affect the confidence, the calculation and capping of the confidence value itself happen entirely within the AnalysisAgent.

8. **Output Contribution:**
    - **Direct:** Inserts latest news headline and overall sentiment into table.
    - **Indirect:** Passes summary scores, sentiment, and confidence to AnalysisAgent for optimized scoring, decisions, and reasoning.
    - **Examples:**
        - For AAPL: *Reasoning* can include "Strong profit margins; Positive news sentiment; Analyst target suggests 13.2% upside."
        - For MSFT: *Reasoning* can include "Reasonable valuation; Strong profit margins; Analyst target suggests 9.7% upside."

D. ANALYSIS AGENT

Role: Weighted Integration, AI Decision-Making, Confidence Evaluation, Explanation

- Synthesizes outputs from DataAgent (fundamentals/technicals) and NewsAgent (news/sentiment/quality)
- Applies evaluator-optimizer algorithms:
    - Scores each component
    - Applies quality-based boosts and adjustments
    - Aggregates into a final confidence-weighted recommendation/decision ("BUY", "HOLD", etc.)
- Generates **human-readable reasoning** using routing and sentiment insights.

Conclusion

This final, user-facing output is a direct product of **orchestrated collaboration** among specialized agents:

- Each agent runs advanced computations using prompt-chaining, tool-based data ingestion, routing, and evaluator-optimizer patterns.
- The AnalysisAgent synthesizes all numerical and textual signals with quality reflection to produce robust, explainable investment intelligence, viewable as scores, decisions, and explanations for each stock.

SECTION 12.2: FUTURE ENHANCEMENTS FOR BETTER ACCURACY:

Future Enhancements & Next Steps

The current multi-agent system provides a robust foundation for automated financial analysis. Future enhancements can focus on deepening the analytical capabilities, improving adaptability, expanding data sources, and enhancing user interaction.

Enhancing Agent Capabilities

- `DataAgent` **- Broader Data Integration:**

    - **More Sources:** Integrate additional APIs for macroeconomic data (e.g., interest rates, inflation from FRED), alternative data (e.g., social media trends, satellite imagery), competitor data, and more granular financial statement details (e.g., cash flow statements, balance sheets).
    - **Data Quality Handling:** Implement more sophisticated methods for handling missing or inconsistent data from APIs.
- `NewsAgent` **- Deeper Contextual Understanding:**
    - **Advanced LLMs/Fine-tuning:** Utilize larger, more powerful LLMs or fine-tune existing models specifically on financial news datasets to better understand nuance, causal relationships, and sector-specific jargon.
    - **Retrieval-Augmented Generation (RAG):** Implement RAG to allow the `NewsAgent` to pull in relevant historical context or specific company information when analyzing a new article, leading to richer summaries and sentiment analysis.
    - **Multilingual Support:** Extend capabilities to process news from multiple languages.
- `AnalysisAgent` **- Sophisticated Modeling & Risk Assessment:**
    - **Dynamic Weighting:** Move beyond fixed weights. Implement models where the importance of factors (valuation vs. sentiment vs. technicals) can change based on market regime (e.g., high volatility vs. stable growth) or sector.
    - **Machine Learning Models:** Incorporate ML models (e.g., regression, classification) trained on historical data to predict potential price movements or risk levels, complementing the rule-based scoring.
    - **Dedicated Risk Analysis:** Introduce specific risk scoring based on factors like volatility (beta), debt levels, geopolitical news, and regulatory changes.
- `EvaluationAgent` **- Autonomous Optimization:**
    - **Richer Metrics:** Develop more nuanced quality metrics, potentially using LLMs to score the coherence and insightfulness of the reasoning.
    - **Automated Parameter Tuning:** Enable the `EvaluationAgent`'s feedback to directly trigger adjustments in other agents' parameters (e.g., automatically adjusting summary length in `NewsAgent` if quality scores are consistently low).
- **New Agents:**
    - `RiskAgent`**:** Dedicated agent to assess various risk factors (market, credit, operational, geopolitical).
    - `MacroeconomicAgent`**:** Focuses on analyzing broader economic trends and their potential impact on specific stocks or sectors.

- `PortfolioAgent`: Manages a virtual portfolio based on the system's recommendations, tracking performance and suggesting rebalancing.

Improving System Architecture & User Experience

- **Asynchronous Processing:** Re-architect parts of the system for asynchronous operation, allowing agents like `DataAgent` and `NewsAgent` to work concurrently more effectively, speeding up analysis, especially for multiple stocks.
- **Enhanced Error Handling & Recovery:** Implement more granular error tracking and potentially add mechanisms for agents to retry failed tasks or adapt the plan if a specific data source is unavailable.
- **Scalability:** Optimize API usage (caching), memory management, and potentially distribute agent workloads for analyzing larger numbers of stocks.
- **User Interface (UI):** Develop a web-based dashboard for easier interaction, allowing users to input stocks, view results and visualizations dynamically, and potentially customize analysis parameters (like decision weights). 💻

Next Steps (Logical Progression)

1. **Refine Core Agents:** Enhance the existing `DataAgent` (more data points) and `NewsAgent` (fine-tuning LLMs/RAG) as they provide the foundational input.
2. **Develop UI:** Create a simple web interface for better usability.
3. **Implement `EvaluationAgent` Optimization:** Make the feedback loop actively tune parameters.
4. **Introduce New Specialized Agents:** Add agents like `RiskAgent` or `MacroeconomicAgent` incrementally.

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit