

【AFAC赛题四】智能体赋能的金融多模态报告自动化生成 baseline教程

💡 本文档为 [AFAC2025挑战组赛题四：智能体赋能的金融多模态报告自动化生成](#) 【Baseline教程】，

核心目标是帮助大家——**理解赛题、复现结果**，主要内容为：

- 赛题解读和问题建模思路介绍
- 可复现的baseline代码
- baseline方案详解
- 提供引导理解赛题和提分思路的简单思考题

作者：李秀奇（筱可）

项目简介

本次项目是需要构建一个能够自动撰写多模态呈现、具备专业性和深度、数据融合与事实溯源、规范有逻辑的各类金融研报的智能Agent系统，学完之后大家可以掌握如何构建一个结合RAG思想的多agent协同的系统，以及金融领域的研究报告都有哪些内容，最终能够生成一篇图文并茂的金融研报。具体可以分成三个部分，分别是公司/个股研报、行业/子行业研报、宏观经济/策略研报。

- **公司研报**需要：目标企业以及同行企业三大会计报表与股权结构；
- **行业研报**需要：目标行业及其产业链上下游发展相关数据，协会年报、企业财报，搜索相关的数据，搜索规模数据，相关政策影响、技术演进。
- **宏观经济/策略研报**需要：GDP、CPI、利率、汇率，政策报告，同类行业政策的影响，美联储利率变动的数据。

完成此项目需要掌握以下知识与能力：Python编程能力、理解并构建RAG系统、理解并构建Agent系统



备注：本baseline只实现公司研报的图文研报生成。

其中行业研报和宏观研报思路与公司研报类似，我仅仅给出行业和宏观研报基于pocketflow版本的deepresearch，不带图文详解。

大家可以参考公司研报的方式探讨另外两个小题的研报内容生成，重点是数据收集。

分析赛题、对问题进行建模

基于比赛说明文档，对金融研报生成比赛进行以下分析和建模：



输入可能是三种情况，分别是公司、行业、宏观的研究报告生成请求，输出是对应的研究报告。

首先需要考虑的如何构建这个系统，系统的输入为3种情况，输出也是三种情况，

- 你可以使用一个路由分发的形式给对应的公司研报、行业研报、宏观研报智能体进行内容生成请求。
- 也可以单纯只是使用多个智能体协同合作进行任务拆解，并共享数据协同执行任务，

这可能是最优的，因为三种研报看似分离，但实际上各有关联，如果每次都是使用三种视角，但是只是完成一个任务，他的全局观以及数据分布，数据质量都会超越从一个视角进行的研报生成。

但我们可以简化问题，分别设计三个智能体，或者说工作流对整个任务进行分解，并执行，单独生成对应的研报内容。

这里我主要以公司研报进行思路介绍，因为另外两个研报思路是大概一致的，其中公司研报的部分也是三个里面最典型的一个，涉及到的数据也更多一些。

首先我们来看看题目的任务：



- 自动抽取公司财务三大表与股权结构，生成主营业务、核心竞争力与行业地位的文字分析；
- 支持财务比率计算与行业对比分析（如ROE 分解、毛利率、现金流匹配度），结合同行企业进行横向竞争分析；
- 构建估值与预测模型，模拟关键变量变化对财务结果的影响（如原材料成本、汇率变动）；
- 结合公开数据与管理层信息，评估公司治理结构与发展战略，提出投资建议与风险提醒。

那我们首先可以思考下公司研报部分如何获取所需要的数据：

具体来说我们可以分解成四个小的子问题：



1. 自动抽取公司财务三大表与股权结构，生成主营业务、核心竞争力与行业地位的文字分析；
 - 方法：先保存三大表，获取股东的数据，接着获取公司的信息，以及使用搜索，获取公司的行业地位以及核心竞争力。
 - 用到的数据：多个三大表、股权结构、公司信息、搜索到的信息。
2. 支持财务比率计算与行业对比分析，结合同行企业进行横向竞争分析；
 - 方法：先获取同行的资料，然后一起进队列，然后直接请求获取roe，毛利率，现金流然后判断他们的匹配度，然后，对他们的信息进行竞争分析
 - 用到的数据：获取财务信息，公司信息，搜索到的信息。
3. 构建估值与预测模型，模拟关键变量变化对财务结果的影响；
 - 方法：用行业平均PE/PB计算目标价，用历史增长率预测未来3年业绩，选择成本、汇率等2-3个变量做±20%的敏感性测试，看对估值的影响。
 - 用到的数据：三大表。
4. 结合公开数据与管理层信息，评估公司治理结构与发展战略，提出投资建议与风险提醒。
 - 方法：输入股东的数据、输入三大表的分析、加入搜索到的行业地位以及核心竞争力，还有估值预测模型分析的结果也加入到里面。
 - 用到的数据：三大表、股东数据、搜索到的信息、预测的报告信息。

总结来说公司报表可以分解成这样的处理思路：



1. 先分别获取单个公司的分析报表数据，以及公司两两之间的对比数据报告
2. 接着获取ai整理后的目标公司股权数据报告
3. 然后获取公司估值的数据报告
4. 最后收集搜索的一些信息，还有加入公司的基础信息
5. 一起作为上下文输入到大模型里面，让他生成一份公司金融研报。

这里我们只是讲到了大概的公司分析研报的大概思路，行业研报以及宏观研报的思路是类似的，可以自行展开，这里的分析先到这里。

注意，数据报告的部分在本次教程里面使用的是我自己做的一个数据分析agent完成的，在本次设计中承担分析计算层的任务，具体可以参考：https://github.com/li-xiu-qi/data_analysis_agent

赛题核心分析

赛事背景


当今金融科技正经历前所未有的变革，智能投研、智能投顾等应用展现出广阔前景。金融研究报告（研报）作为金融领域的核心产出，在基金管理、资产管理、投行等机构决策中起着重要作用。然而，实现研报的自动高质量生成面临诸多挑战。

- 一方面，研报类型多样（宏观经济/策略、行业/子行业、公司/个股等），不同报告需要处理不同形式的数据和专业知识；
- 另一方面，尽管大型预训练语言模型的迅猛发展为自动化研报生成提供了新机遇，直接将大模型应用于复杂金融场景仍存在明显不足。

本任务致力于解决通用预训练大模型财务分析能力缺失、信息获取与整合不足以及大模型幻觉与结构化输出困难三大核心问题，通过引入Agent系统、检索增强生成（RAG）技术、工具库、模型上下文协议（MCP）以及Agent2Agent（A2A）协议等前沿技术，提升大模型在金融场景中的可用性，实现自动化生成具有决策价值的专业研报。

问题本质

这是一个**多模态金融内容**生成问题，需要构建能够自动生成专业金融研报的智能Agent系统。核心挑战在于：

- 
 1. 将通用大模型适配到专业金融领域的研报生成任务
 2. 实现多源数据融合和信息获取
 3. 基于获取的信息以及数据进行系统的分析以及处理
 3. 生成结构化、专业化的研报内容
 4. 确保内容的准确性和可追溯性
 5. 图文并茂，排版优雅

公司/个股研报的生成要点：

- **核心数据：**财务报表、股价数据、公司公告
- **分析重点：**财务指标、估值模型、竞争分析
- **输出特色：**投资建议、风险提示

行业/子行业研报的生成要点：

- **核心数据：**行业统计数据、政策文件、企业群体数据
- **分析重点：**行业趋势、竞争格局、政策影响
- **输出特色：**行业前景预测、投资策略建议

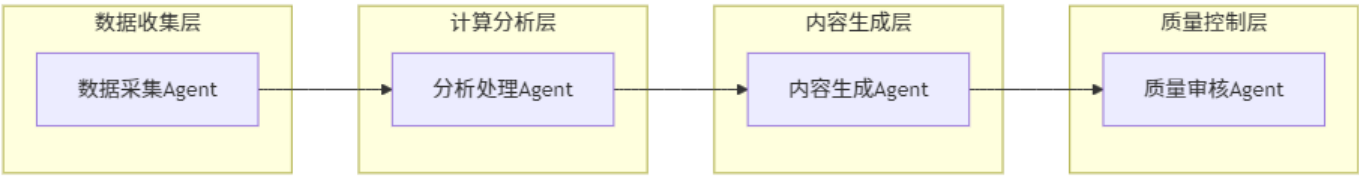
宏观经济/策略研报的生成要点：

- **核心数据：**宏观经济指标、政策文件、国际数据
- **分析重点：**经济周期、政策传导、全球关联
- **输出特色：**宏观预测、资产配置建议

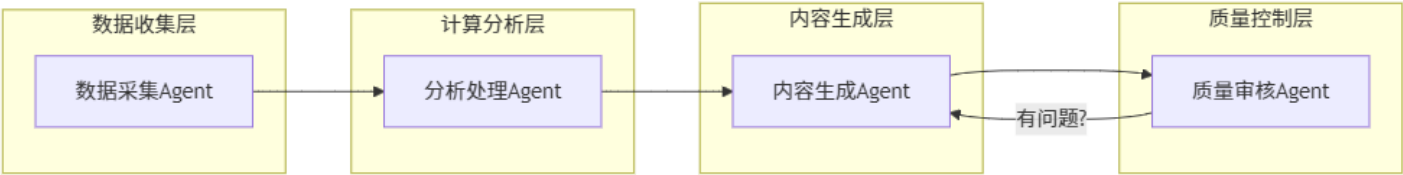
技术建模框架

按照题目的要求，我们可以核心简化成 数据收集层、计算分析层、内容生成层、质量控制层 四个层次：

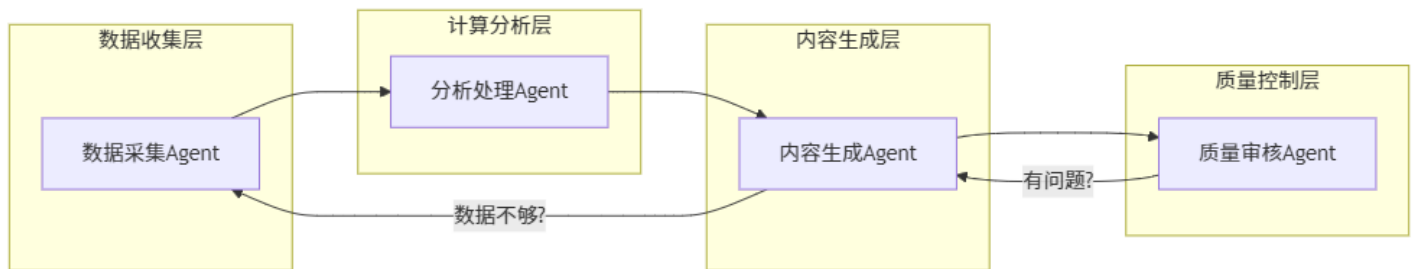
技术上可拆分为4个Agent的多Agent协同系统：



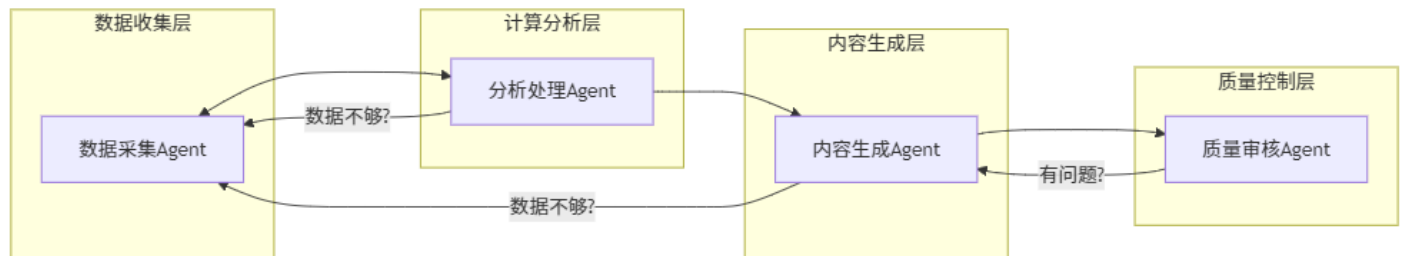
再深入的话，可以加上一个循环，在我们审核发现问题出现的时候，进行**再次生成**的循环操作。



再复杂一些，我们可以考虑**数据质量和数量审查**。



那能不能继续更复杂一些呢？当然可以，比如我们在数据采集的部分以及计算分析的部分可以加入一个双向箭头，数据分析的时候就开始判断数据是否足够，如果不够就继续进行数据收集，直到数据能够满足我们的需求为止。



涉及的核心技术栈如下：

- 🍷 • **RAG (检索增强生成)**：解决金融知识获取问题
- **Aent(智能体)**：模型可以基于对环境的观察而选择合适的外部的插件或者工具增强模型的生成或者控制的能力
- **MCP (模型上下文协议)**：标准化数据交互
- **A2A (Agent2Agent)**：实现Agent间协作
- **工具调用**：集成财务计算、图表生成等功能

可选其中适合的几项，而不是全选，给大家一个建议：核心是实现需求，而不是堆砌技术栈。

提交要求

A榜任务目标

公司：商汤科技（[00020.HK](#)）

行业：智能风控&大数据征信服务

宏观：生成式AI基建与算力投资趋势（2023-2026）

提交格式要求

A榜期间，提交内容格式为ZIP压缩文件，文件命名为 "results.zip"，其中包含三份Microsoft Word文档（.docx），文件命名需 严格遵循 以下规范：公司研报命名为 "Company_Research_Report.docx"，行业研报命名为 "Industry_Research_Report.docx"，宏观研报命名为 "Macro_Research_Report.docx"。提交的研报文档需包含完整的金融研报结构，如包含图表或数据可视化内容，需直接嵌入文档中，不接受外部链接形式。

文件的路径关系：

代码块

```
1  |—— results.zip
2  |  |—— Company_Research_Report.docx
3  |  |—— Industry_Research_Report.docx
4  |  |—— Macro_Research_Report.docx
```

评测结果字段中可以通过 "score_company"、"score_industry"、"score_macro" 看到小分，分数区间为0-10分。

代码块

```
1  score = 1/3(score_company + score_industry + score_macro)
```

如果有个别报告没有完成，例如公司研报，可以提交为：

代码块

```
1  |—— results.zip
2  |  |—— Industry_Research_Report.docx
3  |  |—— Macro_Research_Report.docx
```

score_company 会返回空字符。

评测脚本实时运行，按照提交顺序逐一评测，如出现过较长时间未出结果（超过30分钟），请先检查提交结果命名及格式，若有问题，请及时联系赛题方。

评分维度

指标类别	判分要素	参考策略	要素说明
<div>客观指标</div> <div>(50%)</div> <div>使用自动评分脚本，基于多个顶级商用LLM对生成研报的关键要素进行联合判分</div>	内容完整与结构规范	建立研报要素检查清单	是否涵盖公司/行业/宏观所要求的关键要素，结构清晰、章节完整，格式符合发布规范，包括段落层次、引用标注和风险提示等；
	数据准确与图文一致	多源数据交叉验证机制	陈述和数据是否正确无误，计算得到的指标是否正确，是否存在事实性错误；
	内容真实性	建立权威数据源引用库	文中的关键论据是否有据可依，即有相应引用支撑，且引用来源权威可靠；
	多模态展现	图表生成质量评估体系	是否合理生成并插入财务图表、走势图、对比表等内容，图文配合是否提升信息传达效果；
	创新性与技术性	设计更高效的架构和方案	是否体现Agent系统任务拆解、多模块协同与工具调用能力，是否合理集成先进技术（如RAG、MCP、A2A、ToolCall），是否具有部署潜力。
<div>主观指标</div> <div>(50%)</div> <div>邀请专家评审团对研报进行质量打分，侧重评估自动指标难以充分衡量的方面</div>	分析深度与专业性	融入金融专业知识	是否具备深入的财务/行业/政策分析能力，逻辑严谨、观点有据、有投资洞察与独到判断；
	可读性与表达风格	优化语言表达	语言是否流畅、专业、符合金融研报写作规范，论点表达清晰、有说服力；
	图表效果	专业级图表设计模板	图表设计是否规范清晰、类型选取恰当，能有效增强分析内容的表达效果与可读性。

任务分层拆解

输入可能是一个问题，比如帮我写一份平安银行的金融研报等等，输出是各类金融研报，包括公司分析研报、行业分析研、宏观经济研报，那么我们可以主要分成四个层次进行展开。

① 数据获取层

1. 实时数据爬取模块

- 股票行情数据（价格、成交量等）
- 财务报表数据（三大报表）
- 新闻资讯数据（使用关键词搜索）
- 宏观经济指标（使用一些api接口，或者是开源的库进行数据收集）

2. 数据标准化处理

- 数据清洗和格式统一（比如html转markdown）
- 时间序列对齐（时间格式需要统一，比如是按照年或者是按照月等等）
- 缺失值处理（有的数据可能存在缺失的情况，一般是设置为0，或者是其他情况也需要做一些特殊处理，比如删除等等）

② 分析计算层

这个部分更侧重于告诉大家让AI帮忙分析的时候能从哪些方向进行分析，我们本次运行的效果实际上是由AI自动进行分析的，我们没有做太多的干涉，输入文件，以及基本的一些要求，接着就会让我们之前设计的数据分析Agent自动进行分析，并在最后生成一份分析报告，后续我们在最终的研究报告中会用到这一份分析报告，当然，分析报告是有多种的，既有同一个公司的分析报告，也有和别的公司对比的分析报告，最终生成公司研究报告的时候是会使用到全部的分析报告，从而实现与同行对比的效果。

1. 财务分析模块

- ROE分解分析
- 财务比率计算
- 现金流分析
- 同行业对比

2. 技术分析模块

- 趋势分析
- 技术指标计算
- 风险评估

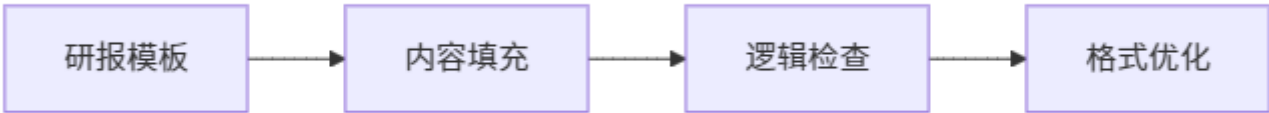
3. 宏观分析模块

- 政策影响分析
- 经济指标关联性分析
- 情景模拟建模

其中金融专业术语的说明，可见 **附录：补充知识**。

③ 内容生成层

1. 结构化生成框架



2. 多模态内容生成

- 文本内容生成（基于模板和数据）
- 图表自动生成（matplotlib/plotly/html/svg渲染等）
- 表格数据展示（以html表格、markdown表格）
- 图文一致性检查（可以借助多模态模型对图片进行检查，但一般我们在创建图片的时候就应该对图片做出对应的描述信息，所以这一步一般是不需要做的）

④ 质量控制层

1. 内容验证机制

- 数据准确性验证
- 逻辑一致性检查
- 引用来源验证

2. 格式规范检查

发布证券研究报告暂行规定：

https://www.gov.cn/gongbao/content/2011/content_1808619.htm

开发建议

1. **渐进式开发**：先实现基础功能，再优化专业性
2. **模块化设计**：便于调试和优化单个组件
3. **数据驱动**：建立评估反馈循环
4. **合规优先**：确保所有数据来源合法合规

如何获得研报需要的数据？

1. 对于公司分析研报

可以通过akshare获取我们需要的公司财务三大表与股权结构，并对数据进行处理，结合大语言模型生成主营业务、核心竞争力与行业地位的文字分析，支持财务比率计算与行业对比分析（如ROE分解、毛利率、现金流匹配度），结合同行企业进行横向竞争分析，构建估值与预测模型，模拟关键变量变化对财务结果的影响（如原材料成本、汇率变动），结合公开数据与管理层信息，评估公司治理结构与发展战略，提出投资建议与风险提醒。

更多数据细节，见 **附录：公司研报核心数据说明**

2. 对于行业分析研报

可以通过akshare以及其他的获取渠道聚合行业发展相关数据（协会年报、企业财报等），输出行业生命周期与结构解读（如集中度、产业链上下游分析），融合趋势分析与外部变量预测能力（如政策影响、技术演进），支持3年以上的行业情景模拟，提供行业进入与退出策略建议，支持关键变量（如上游原材料价格）敏感性分析，自动生成图表辅助说明行业规模变动、竞争格局等核心要素。

更多数据细节，见 **附录：行业研报核心数据说明**

3. 对于宏观研报

可以通过akshare获取与呈现宏观经济核心指标（GDP、CPI、利率、汇率等），对政策报告与关键口径进行解读，构建政策联动与区域对比分析模型，解释宏观变量间的交互影响（如降准对出口与CPI的传导路径），支持全球视野的模拟建模（如美联储利率变动对全球资本流动的影响），提供对潜在“灰犀牛”事件的风险预警机制与指标设计。

更多数据细节，见 **附录：宏观经济研报核心数据说明**

解题思考过程

首先，通过题目我们可以发现这是一个研报生成的任务，而且有需要使用Agent、RAG等技术栈实现，可以先去找找有没有合适的多agent框架可以直接使用，可以发现的是，有类似autogen、openai agents、pocketflow等多智能体/工作量开发框架，我们可以直接使用也可以使用openai 很早之前开发的sdk，目前主流的厂商一般都会支持这个sdk的调用方式。

主要的难点是获取好的数据内容，以及模型的上下文控制,灵活的多模态内容生成、引用。

本题目所涉及到的开发工作比较复杂、建议大家**先确定技术栈**。

首先建立一个文档，列举我们会用到的技术栈的说明文档地址，比如

- pocketflow的说明文档：<https://the-pocket.github.io/PocketFlow/>
- akshare的说明文档：<https://akshare.akfamily.xyz/data/stock/stock.html#id40>等等，

还有其他用到的文档，下面我会在baseline中说明并给出依赖文件，需要查阅的部分可以去网上直接搜索就能找到对应的文档。

接下来，就需要考虑如何实现了，这里主要分为了三个阶段

1. 获取数据，

可以找一些免费的方式获取数据信息，我们本次的baseline会主要以akshare，duckduckgo-search，以及自己写的爬虫获取数据为主。

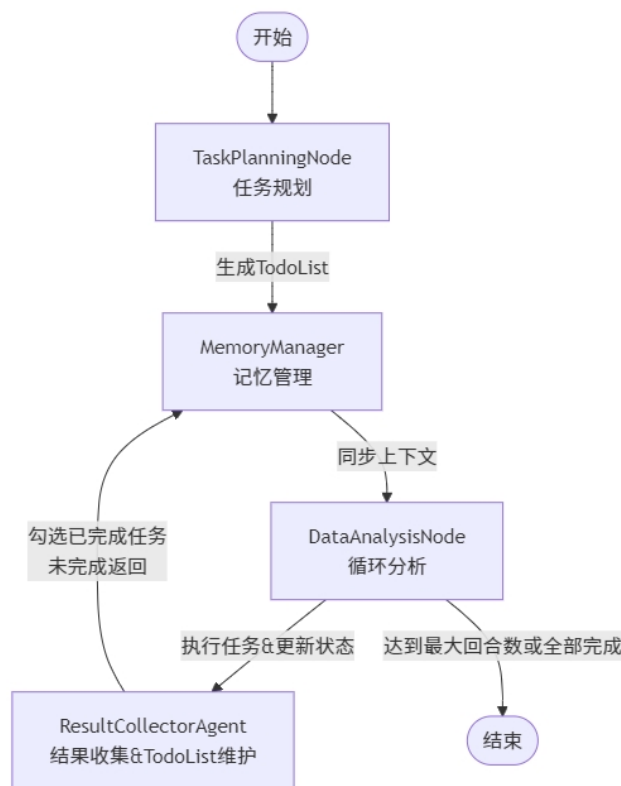
2. 选择模型

本次选用的模型是deepseek v3模型，这是一个开源的模型，满足题目的要求。他的上下文能够达到160k左右，也就是相当于16万字，输出最大是16k，相当于1.6万字，总体上来看是能够满足基本使用的需求的。

3. 逐步实现功能

实际使用的时候，发现在报表的分析环节，有的报表内容非常多，及其庞大，只要打印全表内容模型上下文就会直接爆炸，几乎是没法全放入模型的记忆空间里面的。

所以我们采取的策略是使用notebook的执行环境，有效的利用重复的上下文代码，在表格处理方面给模型提供表头，以及前面五行和后面五行的内容，其他的内容就不全部打印出来，这样的处理使得我们在整个数据分析的过程很少会出上下文记忆溢出的情况出现，如果需要更高级的记忆管理，我建议可以使用类似下图这种方式：



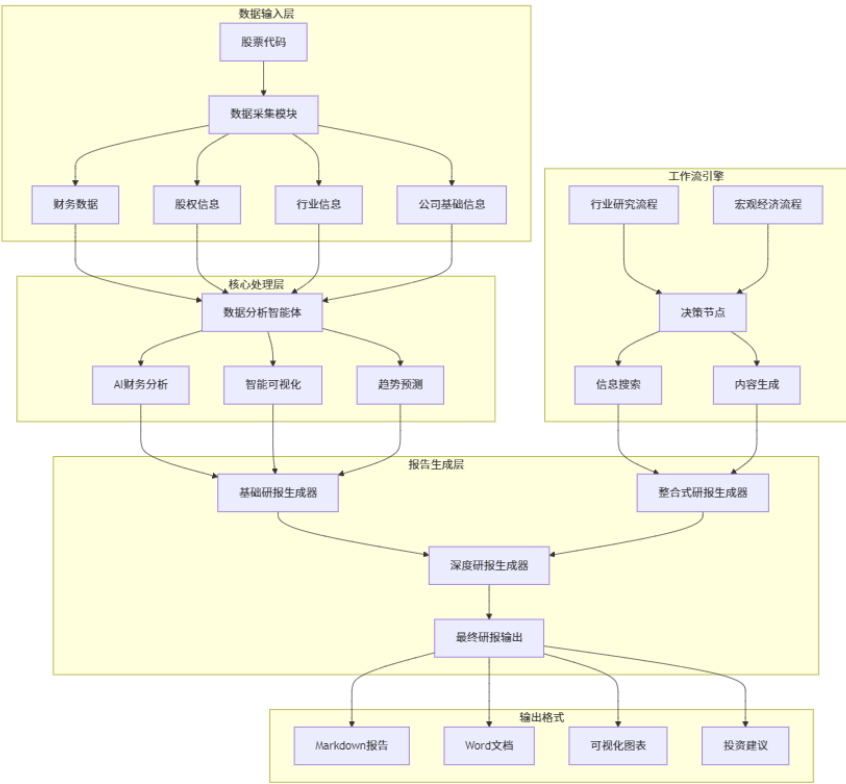
我们继续分析数据，可以设计一个agent，让他能够自动分析我们的输入表格文件、以及分析需求自动化分析数据，并进行数据可视化，生成对应的带图文的分析报告，**数据分析也是本次baseline的核心内容，当然还有一些围绕数据分析内容展开的一些外围部分内容，具体请看baseline核心代码及解释。**

Baseline方案详解

Baseline概况

核心信息	信息详情
赛题任务类型	生成
baseline代码	https://github.com/li-xiu-qi/financial_research_report
baseline涉及的库	Openai sdk、duckduck-search、akshare、pandoc
所需环境和时间	环境：python、cpu 时间：2-4个小时
baseline分数	公司研报4.75分（满分10分），其余为None
baseline所使用的方法	RAG、Agent、工作流

我们Baseline的架构图大致为：



当前的baseline并没有完成架构图的全部功能，最下面的部分暂时没有实现。

在实际的开发中，将右上角结合左上角、可以构建出来一个更优秀的架构。

- 可以一边进行搜索，一边进行内容分析。
- 最终使用搜索到的内容结合分析计算出来的内容，加上一些从网上搜索到的合适的图片。
- 添加到最终的候选内容里。

Baseline核心代码及解释

完整代码详见github仓库：https://github.com/li-xiu-qi/financial_research_report，其运行完成后的文件树如下：

```
代码块
1  financial_research_report/
2  |   └─ 核心生成器
3  |       └─ research_report_generator.py          # 基础研报生成器
4  |       └─ integrated_research_report_generator.py # 整合式研报生成器
```

```

5 |   └─ in_depth_research_report_generator.py    # 深度研报生成器
6 |   └─ 工作流引擎
7 |       └─ industry_workflow.py                # 行业研究 workflow
8 |       └─ macro_workflow.py                  # 宏观经济研究 workflow
9 |   └─ 数据分析智能体
10 |       └─ data_analysis_agent/                # AI 数据分析模块
11 |           └─ __init__.py
12 |           └─ data_analysis_agent.py          # 主分析引擎
13 |           └─ prompts.py                     # 提示词模板
14 |           └─ README.md                      # 模块文档
15 |           └─ config/                         # 配置文件
16 |               └─ __init__.py
17 |               └─ llm_config.py              # LLM 配置
18 |               └─ utils/                      # 工具函数
19 |                   └─ __init__.py
20 |                   └─ code_executor.py        # 代码执行器
21 |                   └─ create_session_dir.py   # 会话目录创建
22 |                   └─ extract_code.py        # 代码提取
23 |                   └─ fallback_openai_client.py # 备用 OpenAI 客户端
24 |                   └─ format_execution_result.py # 结果格式化
25 |                   └─ llm_helper.py          # LLM 助手
26 |   └─ 数据采集工具
27 |       └─ utils/                             # 数据获取工具集
28 |           └─ get_base_info.py               # 基础信息获取
29 |           └─ get_company_info.py            # 公司信息获取
30 |           └─ get_financial_statements.py    # 财务报表获取
31 |           └─ get_shareholder_info.py        # 股东信息获取
32 |           └─ get_stock_intro.py             # 股票介绍获取
33 |           └─ identify_competitors.py        # 竞争对手识别
34 |           └─ search_info.py                 # 信息搜索
35 |   └─ 工作流框架
36 |       └─ pocketflow/                        # 轻量级 workflow 引擎
37 |           └─ __init__.py
38 |   └─ 数据存储
39 |       └─ company_info/                      # 公司基础信息
40 |           └─ 百度_HK_09888_info.txt
41 |           └─ 寒武纪_A_SH688256_info.txt
42 |           └─ 科大讯飞_A_SZ002230_info.txt
43 |           └─ 商汤科技_HK_00020_info.txt
44 |           └─ 云从科技_A_SH688327_info.txt
45 |       └─ download_financial_statement_files/ # 财务报表数据
46 |           └─ [公司名]_balance_sheet_年度.csv # 资产负债表
47 |           └─ [公司名]_cash_flow_statement_年度.csv # 现金流量表
48 |           └─ [公司名]_income_statement_年度.csv # 利润表
49 |       └─ industry_info/                     # 行业信息
50 |           └─ all_search_results.json        # 搜索结果汇总
51 |   └─ 输出结果

```

```

52 |     └─ outputs/                                # 生成的报告和图表
53 |         └─ session_[ID]/                      # 按会话分组的输出
54 |             └─ 经营活动现金流趋势.png
55 |             └─ 净利润趋势.png
56 |             └─ 营业收入趋势.png
57 |             └─ 资产与权益趋势.png
58 | └─ 📄 配置文件
59 |     └─ requirements.txt                        # Python依赖包
60 |     └─ LICENSE                                # 开源许可证
61 | └─ 📄 文档
62 |     └─ README.md                             # 项目说明文档

```

代码执行器

代码块

```

1  # -*- coding: utf-8 -*-
2  """
3  安全的代码执行器，基于 IPython 提供 notebook 环境下的代码执行功能
4  """
5
6  import os
7  import sys
8  import ast
9  import traceback
10 import io
11 from typing import Dict, Any, List, Optional, Tuple
12 from contextlib import redirect_stdout, redirect_stderr
13 from IPython.core.interactiveshell import InteractiveShell
14 from IPython.utils.capture import capture_output
15 import matplotlib
16 import matplotlib.pyplot as plt
17 import matplotlib.font_manager as fm
18
19 class CodeExecutor:
20     """
21     安全的代码执行器，限制依赖库，捕获输出，支持图片保存与路径输出
22     """
23     ALLOWED_IMPORTS = {
24         'pandas', 'pd',
25         'numpy', 'np',
26         'matplotlib', 'matplotlib.pyplot', 'plt',
27         'duckdb', 'scipy', 'sklearn',
28         'plotly', 'dash', 'requests', 'urllib',

```



```

29         'os', 'sys', 'json', 'csv', 'datetime', 'time',
30         'math', 'statistics', 're', 'pathlib', 'io',
31         'collections', 'itertools', 'functools', 'operator',
32         'warnings', 'logging', 'copy', 'pickle', 'gzip', 'zipfile',
33         'typing', 'dataclasses', 'enum', 'sqlite3'
34     }
35
36     def __init__(self, output_dir: str = "outputs"):
37         """
38         初始化代码执行器
39
40         Args:
41             output_dir: 输出目录，用于保存图片 and 文件
42         """
43         self.output_dir = os.path.abspath(output_dir)
44         os.makedirs(self.output_dir, exist_ok=True)
45
46         # 初始化 IPython shell
47         self.shell = InteractiveShell.instance()
48
49         # 设置中文字体
50         self._setup_chinese_font()
51
52         # 预导入常用库
53         self._setup_common_imports()
54
55         # 图片计数器
56         self.image_counter = 0
57
58     def _setup_chinese_font(self):
59         """设置matplotlib中文字体显示"""
60         try:
61             # 设置matplotlib使用Agg backend避免GUI问题
62             matplotlib.use('Agg')
63
64             # 设置matplotlib使用simhei字体显示中文
65             plt.rcParams['font.sans-serif'] = ['SimHei', 'DejaVu Sans',
66             'Arial Unicode MS']
67             plt.rcParams['axes.unicode_minus'] = False
68             # 在shell中也设置
69             self.shell.run_cell("""
70 import matplotlib
71 matplotlib.use('Agg')
72 import matplotlib.pyplot as plt
73 plt.rcParams['font.sans-serif'] = ['SimHei', 'DejaVu Sans', 'Arial Unicode
74 MS']
75 plt.rcParams['axes.unicode_minus'] = False

```

```

74     """
75         except Exception as e:
76             print(f"设置中文字体失败: {e}")
77
78     def _setup_common_imports(self):
79         """预导入常用库"""
80         common_imports = """
81 import pandas as pd
82 import numpy as np
83 import matplotlib.pyplot as plt
84 import duckdb
85 import os
86 import json

```

当需要让 AI 模型动态执行代码，传统的做法是先将代码写入文件再运行。这种方式不仅操作上有些繁琐，也可能带来不必要的风险。一个更优的思路是构建一个类似 Notebook 的交互式执行环境，CodeExecutor 就是这样一个组件。

CodeExecutor 为 Python 代码片段的执行提供了一个安全且受控的平台，特别适合数据分析、自动化报告生成这类需要动态代码能力的场景。通过内部集成的 IPython shell，它模拟了 Jupyter Notebook 那种交互式、能保持状态的执行体验。这样，代码可以在一个持久化的环境中运行，变量和上下文状态也能在不同代码块执行之间顺畅传递。

安全机制是一个核心考量。在代码执行前，会利用抽象语法树（AST）对代码进行静态分析。这个步骤确保只有白名单 ALLOWED_IMPORTS 里的库才能被导入，同时禁止了像 exec、eval 这样的高风险内置函数。从源头上控制风险，阻止了潜在的恶意代码执行，有助于保障模型输出的安全性，减少对系统的意外影响。

除了安全，CodeExecutor 在执行流程和环境管理方面也提供了便利。它能捕获执行过程中的标准输出和错误信息，对 Pandas DataFrame 这样的表格数据也会进行格式化，提升了可读性。此外，还会预加载常用库，配置中文字体以确保 matplotlib 图表正常显示，并能追踪执行期间新产生的变量。这些特性为 AI 模型提供了一个功能全面的代码执行环境。

代码提取器

代码块

```

1 from typing import Optional
2 import yaml
3

```

```

4  def extract_code_from_response(response: str) -> Optional[str]:
5      """从LLM响应中提取代码"""
6      try:
7          # 尝试解析YAML
8          if '```yaml' in response:
9              start = response.find('```yaml') + 7
10             end = response.find('```', start)
11             yaml_content = response[start:end].strip()
12         elif '```' in response:
13             start = response.find('```') + 3
14             end = response.find('```', start)
15             yaml_content = response[start:end].strip()
16         else:
17             yaml_content = response.strip()
18
19             yaml_data = yaml.safe_load(yaml_content)
20             if 'code' in yaml_data:
21                 return yaml_data['code']
22     except:
23         pass
24
25     # 如果YAML解析失败，尝试提取```python代码块
26     if '```python' in response:
27         start = response.find('```python') + 9
28         end = response.find('```', start)
29         if end != -1:
30             return response[start:end].strip()
31     elif '```' in response:
32         start = response.find('```') + 3
33         end = response.find('```', start)
34         if end != -1:
35             return response[start:end].strip()
36
37     return None

```

当 AI 模型返回一大段文本，里面可能夹杂着我们需要的代码时，这个函数就派上用场了。它的任务就是从这些文本中把代码提取出来。

函数主要尝试两种策略来提取代码：

- 第一种是解析 YAML 格式。它会先找文本里有没有被 `yaml```` 这样的标记包围的部分。如果找到了，就尝试把这部分内容当作 YAML 来解析。如果解析成功，并且发现里面有个叫 `code` 的字段，太棒了，它就把这个字段对应的内容作为代码返回。如果直接标明 `yaml`` 的块没找到，它还会尝试找通用的 ````` 块，或者干脆把整个响应都当成 YAML 试试看。

- 如果第一种方法没成功，比如文本不是 YAML 格式，或者 YAML 里没有 code 字段，那它就启动第二套方案：直接找 Python 代码块。这时候，它会寻找被 python ``` 包围的代码。如果这种明确标示的 Python 代码块找不到，它也会退一步，查找通用的 ``` 代码块，并假定里面的内容是代码。

要是这两种方法都失败了，那函数就只好放弃，返回一个 None，表示没能成功提取出代码。这种逐步尝试、有备用方案的设计，提高了从不同格式的 LLM 响应中成功提取代码的概率。

做两种提取方式也是为了兼容其他的代码提取的时候进行使用，有的时候我们是让模型直接输出代码，但是可能存在 ```python对代码进行包围，也有时候存在代码块的外面有其他内容，比如一些问候语，或者提示语，比如“下面是您所需要的代码 ```python.....”，所以我们直接提取代码块的内容是一种更优雅的处理办法，对于yaml的部分我们只是提取code的部分内容主要仅仅是为了我们当前的场景使用。

Agent部分涉及到的提示词

代码块

```
1  data_analysis_system_prompt = """你是一个专业的数据分析助手，运行在Jupyter
    Notebook环境中，能够根据用户需求生成和执行Python数据分析代码。
2
3  🎯 **重要指导原则**：
4  - 当需要执行Python代码（数据加载、分析、可视化）时，使用 `generate_code` 动作
5  - 当需要收集和分析已生成的图表时，使用 `collect_figures` 动作
6  - 当所有分析工作完成，需要输出最终报告时，使用 `analysis_complete` 动作
7  - 每次响应只能选择一种动作类型，不要混合使用
8
9  目前jupyter notebook环境下有以下变量：
10 {notebook_variables}
11
12 ✨ 核心能力：
13 1. 接收用户的自然语言分析需求
14 2. 按步骤生成安全的Python分析代码
15 3. 基于代码执行结果继续优化分析
16
17 🛠️ Notebook环境特性：
18 - 你运行在IPython Notebook环境中，变量会在各个代码块之间保持
19 - 第一次执行后，pandas、numpy、matplotlib等库已经导入，无需重复导入
20 - 数据框(DataFrame)等变量在执行后会保留，可以直接使用
21 - 因此，除非是第一次使用某个库，否则不需要重复import语句
22
23 📌 重要约束：
24 1. 仅使用以下数据分析库：pandas, numpy, matplotlib, duckdb, os, json,
    datetime, re, pathlib
```

25 2. 图片必须保存到指定的会话目录中, 输出绝对路径, 禁止使用`plt.show()`

26 4. 表格输出控制: 超过15行只显示前5行和后5行

27 5. 强制使用SimHei字体: `plt.rcParams['font.sans-serif'] = ['SimHei']`

28 6. 输出格式严格使用YAML

29 7. 不能使用上述库之外的任何库

30 📁 输出目录管理:

31 - 本次分析使用UUID生成的专用目录 (16进制格式), 确保每次分析的输出文件隔离

32 - 会话目录格式: `session_[32位16进制UUID]`, 如
`session_a1b2c3d4e5f6789012345678901234ab`

33 - 图片保存路径格式: `os.path.join(session_output_dir, '图片名称.png')`

34 - 使用有意义的中文文件名: 如 '营业收入趋势.png', '利润分析对比.png'

35 - 每个图表保存后必须使用`plt.close()`释放内存

36 - 输出绝对路径: 使用`os.path.abspath()`获取图片的完整路径

37

38 📊 数据分析工作流程 (必须严格按顺序执行):

39

40 ****阶段1: 数据探索 (使用 generate_code 动作) ****

41 - 首次数据加载时尝试多种编码: `['utf-8', 'gbk', 'gb18030', 'gb2312']`

42 - 使用`df.head()`查看前几行数据

43 - 使用`df.info()`了解数据类型和缺失值

44 - 使用`df.describe()`查看数值列的统计信息

45 - ****强制要求: 必须先打印所有列名****: `print("实际列名:", df.columns.tolist())`

46 - ****严禁假设列名****: 绝对不要使用任何未经验证的列名, 所有列名必须从`df.columns`中获取

47 - ****列名使用规则****: 在后续代码中引用列名时, 必须使用`df.columns`中实际存在的列名

48

49 ****阶段2: 数据清洗和检查 (使用 generate_code 动作) ****

50 - ****列名验证****: 再次确认要使用的列名确实存在于`df.columns`中

51 - 检查关键列的数据类型 (特别是日期列)

52 - 查找异常值和缺失值

53 - 处理日期格式转换

54 - 检查数据的时间范围和排序

55 - ****错误预防****: 每次使用列名前, 先检查该列是否存在

56

57 ****阶段3: 数据分析和可视化 (使用 generate_code 动作) ****

58 - ****列名安全使用****: 只使用已经验证存在的列名进行计算

59 - ****动态列名匹配****: 如果需要找特定含义的列, 使用模糊匹配或包含关键字的方式查找

60 - ****智能图表选择****: 根据数据类型和分析目标选择最合适的图表类型

61 - 时间序列数据: 使用线图(plot)展示趋势变化

62 - 分类比较: 使用柱状图(bar)比较不同类别的数值

63 - 分布分析: 使用直方图(hist)或箱型图(boxplot)

64 - 相关性分析: 使用散点图(scatter)或热力图(heatmap)

65 - 占比分析: 使用饼图(pie)或堆叠柱状图(stacked bar)

66 - ****图表设计原则****:

67 - 确保图表清晰易读, 合理设置图表尺寸(`figsize=(10,6)`或更大)

68 - 使用有意义的中文标题、坐标轴标签和图例

69 - 对于金融数据, 优先展示趋势、对比和关键指标

70 - 数值过大时使用科学计数法或单位转换(如万元、亿元)

```
71     - 合理设置颜色和样式，提高可读性
72     - 图片保存到会话专用目录中
73     - 每生成一个图表后，必须打印绝对路径
74
75     **阶段4：图片收集和分析（使用 collect_figures 动作）**
76     - 当已生成2-3个图表后，使用 collect_figures 动作
77     - 收集所有已生成的图片路径和信息
78     - 对每个图片进行详细的分析和解读
79
80     **阶段5：最终报告（使用 analysis_complete 动作）**
81     - 当所有分析工作完成后，生成最终的分析报告
82     - 包含对所有图片和分析结果的综合总结
83
84     🔧 代码生成规则：
85     1. 每次只专注一个阶段，不要试图一次性完成所有任务
```

首先，最核心的是 `data_analysis_system_prompt`。你可以把它想象成给 AI 数据分析助手设定的一套非常严格的“规章制度”和“工作流程”。它告诉模型：

💡 你的角色：你是一个在 Jupyter Notebook 环境里工作的专业数据分析助手。

你的工具箱：你只能使用像 pandas, numpy, matplotlib, duckdb 等白名单里的库。

你的行动指南：什么时候该生成代码 (generate_code)，什么时候该收集图表信息 (collect_figures)，什么时候该给出最终报告 (analysis_complete)，都规定得清清楚楚。而且，每次回应只能做一个动作。

工作流程：从数据加载、探索、清洗，到分析、可视化，再到图表收集和最终报告，每个阶段都有明确的步骤和要求。比如，加载数据要尝试多种编码，画图必须保存到特定目录，文件名要有意义，图表要有中文标题和标签，并且用指定的 SimHei 字体。

安全第一：特别强调了列名使用的安全性，严禁假设列名存在，必须先从 df.columns 中验证。

输出格式：所有的思考和行动都必须以严格的 YAML 格式输出，这样后续程序才能准确解析和执行。

然后，还有两个用于生成最终报告的提示词： `final_report_system_prompt` 和 `final_report_system_prompt_absolute`。

这两个提示词指导模型如何将整个分析过程（包括生成的图表、代码执行结果等）汇总成一份专业的 Markdown 格式的分析报告。它们详细规定了报告的结构，比如要有分析概述、过程总结、关键发现、图表分析、结论建议等。

两者主要的区别在于报告中引用图片的方式：

- `final_report_system_prompt` 要求使用相对路径（例如 `./图表.png`），这样报告更容易在不同环境下迁移和查看。
- `final_report_system_prompt_absolute` 则要求使用绝对路径，这在不同图片分别在不同的地址上的时候是有用的，方便我们后续在同一个markdown文件中引用多个多个目录下的图片，以及转移处理。

实现数据分析功能

代码块

```
1  # -*- coding: utf-8 -*-
2  """
3  简化的 Notebook 数据分析智能体
4  仅包含用户和助手两个角2. 图片必须保存到指定的会话目录中，输出绝对路径，禁止使用
   plt.show()
5  3. 表格输出控制：超过15行只显示前5行和后5行
6  4. 强制使用SimHei字体：plt.rcParams['font.sans-serif'] = ['SimHei']
7  5. 输出格式严格使用YAML共享上下文的单轮对话模式
8  """
9
10 import os
11 import json
12 import yaml
13 from typing import Dict, Any, List, Optional
14 from .utils.create_session_dir import create_session_output_dir
15 from .utils.format_execution_result import format_execution_result
16 from .utils.extract_code import extract_code_from_response
17 from .utils.llm_helper import LLMHelper
18 from .utils.code_executor import CodeExecutor
19 from .config.llm_config import LLMConfig
20 from .prompts import data_analysis_system_prompt,
   final_report_system_prompt, final_report_system_prompt_absolute
21
22 class DataAnalysisAgent:
23     """
24     数据分析智能体
25
26     职责：
27     - 接收用户自然语言需求
28     - 生成Python分析代码
29     - 执行代码并收集结果
```

```

30     - 基于执行结果继续生成后续分析代码
31     """
32     def __init__(self, llm_config: LLMConfig = None,
33                   output_dir: str = "outputs",
34                   max_rounds: int = 20,
35                   absolute_path: bool = False):
36         """
37         初始化智能体
38
39         Args:
40             config: LLM配置
41             output_dir: 输出目录
42             max_rounds: 最大对话轮数
43         """
44         self.config = llm_config or LLMConfig()
45         self.llm = LLMHelper(self.config)
46         self.base_output_dir = output_dir
47         self.max_rounds = max_rounds
48         # 对话历史和上下文
49         self.conversation_history = []
50         self.analysis_results = []
51         self.current_round = 0
52         self.session_output_dir = None
53         self.executor = None
54         self.absolute_path = absolute_path
55
56     def _process_response(self, response: str) -> Dict[str, Any]:
57         """
58         统一处理LLM响应，判断行动类型并执行相应操作
59
60         Args:
61             response: LLM的响应内容
62
63         Returns:
64             处理结果字典
65         """
66         try:
67             yaml_data = self.llm.parse_yaml_response(response)
68             action = yaml_data.get('action', 'generate_code')
69
70             print(f"🔍 检测到动作: {action}")
71
72             if action == 'analysis_complete':
73                 return self._handle_analysis_complete(response, yaml_data)
74             elif action == 'collect_figures':
75                 return self._handle_collect_figures(response, yaml_data)
76             elif action == 'generate_code':

```



```

77         return self._handle_generate_code(response, yaml_data)
78     else:
79         print(f"⚠️ 未知动作类型: {action}, 按generate_code处理")
80         return self._handle_generate_code(response, yaml_data)
81
82     except Exception as e:
83         print(f"⚠️ 解析响应失败: {str(e)}, 按generate_code处理")
84         return self._handle_generate_code(response, {})
85
86 def _handle_analyze_complete(self, response: str, yaml_data: Dict[str,

```

DataAnalysisAgent 的主要职责是接收用户的自然语言需求（比如“帮我分析一下A公司的营收趋势”），然后通过与大语言模型（LLM）的多轮对话，一步步地生成 Python 代码进行数据分析，执行这些代码，收集结果，并最终形成一份分析报告。

初始化与准备：

当创建一个 DataAnalysisAgent 实例时，它会做一些准备工作：

1. 设置好与 LLM 通信的助手 (LLMHelper)。
2. 准备好一个代码执行器 (CodeExecutor)，负责安全地运行 Python 代码。
3. 设定一个基础的输出目录，并且能够为每一次分析任务创建一个独立的、带有唯一ID的会话目录 (session_output_dir)，确保不同分析任务的产出（如图表、报告）互相隔离。
4. 初始化对话历史、分析结果列表等内部状态。

核心分析流程 (analyze 方法)：

这是智能体的核心方法，驱动整个分析过程：

1. 接收用户输入：用户给出分析需求和相关的数据文件。
2. 创建会话环境：为本次分析创建一个专属的输出目录，并初始化代码执行器，将这个会话目录设置为代码执行环境中的一个变量 (session_output_dir)，这样生成的图表就能保存到正确的地方。

多轮对话循环：

智能体进入一个循环，与 LLM 进行多轮对话，直到达到预设的最大轮数或者分析任务完成。

在每一轮，它会构建一个包含当前对话历史和系统指令（来自 prompts.py 的 data_analysis_system_prompt，并动态填入当前 Notebook 环境中的变量信息）的提示，发送给 LLM。

LLM 返回一个包含“思考过程”和“建议动作”的响应（通常是 YAML 格式）。

响应处理 (`_process_response` 方法):

1. 智能体解析 LLM 的响应，判断 LLM 想要执行哪种动作，比如是 `generate_code`（生成代码）、`collect_figures`（收集图表信息）还是 `analysis_complete`（分析完成）。
2. 代码生成与执行 (`_handle_generate_code`): 如果 LLM 要求生成代码，智能体会提取代码，交给 CodeExecutor 执行。执行后，将结果（包括输出、错误信息、新生成的变量等）格式化后反馈给 LLM，作为下一轮对话的输入。这里还有一个细节：它会检查代码执行后声称已保存的图片是否真的存在于文件系统中，如果图片“丢失”了，它会向 LLM 反馈这个问题，并可能建议重新分析或修正代码。
3. 图片收集 (`_handle_collect_figures`): 如果 LLM 决定收集已生成的图表信息，智能体会根据 LLM 提供的图表列表（包含文件名、描述、分析等），整理这些信息，并验证图片文件是否真实存在。
4. 分析完成 (`_handle_analysis_complete`): 当 LLM 判断所有分析已完成，智能体就准备结束流程并生成最终报告。
5. 记录与迭代：每一轮的对话、代码、执行结果、收集的图表信息都会被记录下来。
6. 最终报告生成 (`_generate_final_report` 方法):

当分析流程结束（无论是达到最大轮数还是 LLM 主动完成），智能体会：

1. 汇总在整个分析过程中收集到的所有图表信息和成功的代码执行结果。
2. 使用 prompts.py 中的 `final_report_system_prompt`（或 `final_report_system_prompt_absolute`，取决于初始化时是否指定使用绝对路径）作为模板，将汇总的信息填入，构建一个生成最终报告的提示。
3. 调用 LLM 生成 Markdown 格式的最终分析报告。
4. 将生成的报告保存到当前会话的输出目录中。

分析报告生成以及资料收集部分

代码块

```
1  """
2  金融研报生成器
3  整合财务分析、股权分析、估值模型和行业信息，生成完整的金融研报
4  """
5
6  import os
7  import glob
8  import time
9  import json
```

```
10 from datetime import datetime
11 from dotenv import load_dotenv
12 import importlib
13
14 from data_analysis_agent import quick_analysis
15 from data_analysis_agent.config.llm_config import LLMConfig
16 from data_analysis_agent.utils.llm_helper import LLMHelper
17 from utils.get_shareholder_info import get_shareholder_info, get_table_content
18 from utils.get_financial_statements import get_all_financial_statements,
    save_financial_statements_to_csv
19 from utils.identify_competitors import identify_competitors_with_ai
20 from utils.get_stock_intro import get_stock_intro, save_stock_intro_to_txt
21 from duckduckgo_search import DDGS
22
23 # ===== 环境变量与全局配置 =====
24 load_dotenv()
25 api_key = os.getenv("OPENAI_API_KEY")
26 base_url = os.getenv("OPENAI_BASE_URL", "https://api.openai.com/v1")
27 model = os.getenv("OPENAI_MODEL", "gpt-4")
28
29 target_company = "商汤科技"
30 target_company_code = "00020"
31 target_company_market = "HK"
32 data_dir = "./download_financial_statement_files"
33 os.makedirs(data_dir, exist_ok=True)
34
35 company_info_dir = "./company_info"
36 os.makedirs(company_info_dir, exist_ok=True)
37
38 llm_config = LLMConfig(
39     api_key=api_key,
40     base_url=base_url,
41     model=model,
42     temperature=0.7,
43     max_tokens=16384,
44 )
45 llm = LLMHelper(llm_config)
46
47 # ===== 1. 获取目标公司及竞争对手的财务数据 =====
48 # 获取竞争对手列表
49 other_companies = identify_competitors_with_ai(api_key=api_key,
50                                                 base_url=base_url,
51                                                 model_name=model,
52                                                 company_name=target_company)
53 listed_companies = [company for company in other_companies if
54                      company.get('market') != "未上市"]
```

```

55 # 获取目标公司财务数据
56 print("\n" + "="*80)
57 print(f"获取目标公司 {target_company}({target_company_market}:
    {target_company_code}) 的财务数据")
58 target_financials = get_all_financial_statements(
59     stock_code=target_company_code,
60     market=target_company_market,
61     period="年度",
62     verbose=False
63 )
64 save_financial_statements_to_csv(
65     financial_statements=target_financials,
66     stock_code=target_company_code,
67     market=target_company_market,
68     company_name=target_company,
69     period="年度",
70     save_dir=data_dir
71 )
72
73 # 获取竞争对手的财务数据
74 print("\n" + "="*80)
75 print("获取竞争对手的财务数据")
76 competitors_financials = {}
77 for company in listed_companies:
78     company_name = company.get('name')
79     company_code = company.get('code')
80     market_str = company.get('market', '')
81     if "A" in market_str:
82         market = "A"
83     if not (company_code.startswith('SH') or
84             company_code.startswith('SZ')):
85         if company_code.startswith('6'):

```

这个脚本是整个金融研报生成流程的调度中心，它串联了数据获取、信息整理、多维度分析以及最终报告输出等一系列复杂的步骤。

1. 数据准备与信息搜集：

脚本首先会进行一系列的数据准备工作：

- **环境配置：**加载环境变量（如 API 密钥、模型名称等），并设置目标公司（这里是“商汤科技”）和数据存储目录。
- **识别竞争对手：**调用 `identify_competitors_with_ai` 函数，利用 AI 识别出目标公司的主要竞争对手。
- **获取财务数据：**

- 调用 `get_all_financial_statements` 函数，分别获取目标公司及其已上市竞争对手的年度财务报表（资产负债表、利润表、现金流量表）。
- 获取到的财务数据会以 CSV 格式保存到 `data_dir` 目录下，方便后续的 `DataAnalysisAgent` 进行分析。
- **获取公司基础信息：**
 - 调用 `get_stock_intro` 函数，获取目标公司、所有已识别的竞争对手，以及一些特定公司（如脚本中写明的百度）的公司概况信息。
 - 这些信息会保存为文本文件到 `company_info_dir` 目录。
- **搜索行业信息：**
 - 使用 `duckduckgo_search` 库（`DDGS`）针对目标公司和竞争对手，搜索相关的行业地位、市场份额、竞争分析、业务模式等信息。
 - 搜索结果会汇总并保存为一个 JSON 文件（`all_search_results.json`）到 `industry_info_dir` 目录。

2. 信息整理与初步分析：

在收集完原始数据后，脚本会进行初步的整理和分析：

- **公司信息整理：**读取 `company_info_dir` 目录下所有公司的基础信息文本文件，将它们汇总，然后调用 LLM 对这些信息整理，使其更清晰易读，并保留关键信息。
- **股权信息分析：**调用 `get_shareholder_info` 和 `get_table_content` 获取目标公司（商汤科技）的股东信息表格，然后让 LLM 对这些表格内容进行分析。
- **行业信息搜索结果整理：**读取之前保存的 `all_search_results.json` 文件，将其中的搜索结果（标题、链接、摘要）整合成一段文本。

3. 深度财务数据分析（核心环节）：

这是脚本中最为核心和复杂的部分，主要依赖 `DataAnalysisAgent`（我们之前讨论过的那个智能体）来完成：

- **单公司财务分析：**
 - `analyze_companies_in_directory` 函数会遍历 `data_dir` 目录下的所有公司的财务数据 CSV 文件。
 - 对每家公司，它会调用 `quick_analysis` 函数（实际上就是启动一个 `DataAnalysisAgent` 实例），让智能体基于该公司的一系列财务报表进行分析，生成图表，并产出一份针对该公司的分析报告。默认的分析指令是“基于表格的数据，分析有价值的内容，并绘制相关图表。最后生成汇报给我。”
- **两两对比分析：**
 - `run_comparison_analysis` 函数会选取目标公司（商汤科技），并将其与 `data_dir` 中的其他每一家公司进行两两对比分析。

- 同样是调用 `quick_analysis`，但此时会传入两家公司的财务数据文件，并给出特定的分析指令：“基于两个公司的表格的数据，分析有共同点的部分，绘制对比分析的表格，并绘制相关图表。最后生成汇报给我。”

- **特定公司估值与预测分析：**

- `analyze_sensetime_valuation` 函数专门针对目标公司（商汤科技）进行更深入的估值与预测分析。
- 它会收集商汤科技的所有财务数据文件，然后调用 `quick_analysis`，分析指令是：“基于三大表的数据，构建估值与预测模型，模拟关键变量变化对财务结果的影响,并绘制相关图表。最后生成汇报给我。”

- **结果合并：**所有单个公司的分析报告和两两对比的分析报告最终会被合并到一个字典 `merged_results` 中。

4. 最终报告生成与输出：

在所有分析完成后，脚本会将收集和分析得到的所有信息整合成一份完整的金融研报：

- **格式化输出：**`format_final_reports` 函数会将 `merged_results` 中的各个分析报告（主要是文本内容）整合成一个大的字符串。
- **保存文本报告：**将上述整合后的文本报告保存为一个带时间戳的 `.txt` 文件。
- **保存 Markdown 报告：**这是最终的成果。脚本会将前面整理好的公司基础信息、股权信息分析、行业信息搜索结果、所有财务数据分析与对比报告，以及商汤科技的估值与预测分析报告，全部写入一个带时间戳的 Markdown (`.md`) 文件。Markdown 格式方便后续转换为 Word 文档或直接阅读，并且可以内嵌 `DataAnalysisAgent` 生成的图片（通过相对路径引用）。

主程序入口 (`if __name__ == "__main__":`)

这部分代码负责按顺序调用上述的各个功能模块，从数据获取开始，一步步进行到最终报告的生成和保存。它还会在控制台打印一些进度信息和最终产出文件的路径。

公司分析报告生成部分

代码块

```
1  """
2  深入财务研报分析与生成脚本（简洁版）
3  基于自动化采集与分析的财务研报汇总，结合大模型能力，生成详细的公司财务、股权、行业、估值、
   治理结构等多维度深度分析与投资建议。
4  """
5
6  import os
```

```

7  import yaml
8  from datetime import datetime
9  from data_analysis_agent.config.llm_config import LLMConfig
10 from data_analysis_agent.utils.llm_helper import LLMHelper
11 import re
12 import shutil
13 import requests
14 from urllib.parse import urlparse
15
16 def load_report_content(md_path):
17     with open(md_path, "r", encoding="utf-8") as f:
18         return f.read()
19
20 def get_background():
21     return '''
22 本报告基于自动化采集与分析流程，涵盖如下环节：
23  - 公司基础信息等数据均通过akshare、公开年报、主流财经数据源自动采集。
24  - 财务三大报表数据来源：东方财富-港股-财务报表-三大报表
25    (https://emweb.securities.eastmoney.com/PC\_HKF10/FinancialAnalysis/index)
26  - 主营业务信息来源：同花顺-主营介绍
27    (https://basic.10jqka.com.cn/new/000066/operate.html)
28  - 股东结构信息来源：同花顺-股东信息
29    (https://basic.10jqka.com.cn/HK0020/holder.html) 通过网页爬虫技术自动采集
30  - 行业信息通过DuckDuckGo等公开搜索引擎自动抓取，引用了权威新闻、研报、公司公告等。
31  - 财务分析、对比分析、估值与预测均由大模型（如GPT-4）自动生成，结合了行业对标、财务比率、
32    治理结构等多维度内容。
33  - 相关数据与分析均在脚本自动化流程下完成，确保数据来源可追溯、分析逻辑透明。
34  - 详细引用与外部链接已在正文中标注。
35  - 数据接口说明与免责声明见文末。
36  '''
37
38 def get_llm():
39     api_key = os.environ.get("OPENAI_API_KEY")
40     base_url = os.environ.get("OPENAI_BASE_URL", "https://api.openai.com/v1")
41     model = os.environ.get("OPENAI_MODEL", "gpt-4")
42     llm_config = LLMConfig(api_key=api_key, base_url=base_url, model=model)
43     return LLMHelper(llm_config)
44
45 def generate_outline(llm, background, report_content):
46     outline_prompt = f"""
47 你是一位顶级金融分析师和研报撰写专家。请基于以下背景和财务研报汇总内容，生成一份详尽的《商
48 汤科技公司研报》分段大纲，要求：
49  - 以yaml格式输出，务必用```yaml和```包裹整个yaml内容，便于后续自动分割。
50  - 每一项为一个主要部分，每部分需包含：
51    - part_title: 章节标题
52    - part_desc: 本部分内容简介

```

```

48 - 章节需覆盖公司基本面、财务分析、行业对比、估值与预测、治理结构、投资建议、风险提示、数据来源等。
49 - 只输出yaml格式的分段大纲，不要输出正文内容。
50
51 【背景说明开始】
52 {background}
53 【背景说明结束】
54
55 【财务研报汇总内容开始】
56 {report_content}
57 【财务研报汇总内容结束】
58 """
59     outline_list = llm.call(
60         outline_prompt,
61         system_prompt="你是一位顶级金融分析师和研报撰写专家，善于结构化、分段规划输出，
分段大纲必须用```yaml包裹，便于后续自动分割。",
62         max_tokens=4096,
63         temperature=0.3
64     )
65     print("\n===== 生成的分段大纲如下 =====\n")
66     print(outline_list)
67     try:
68         if '```yaml' in outline_list:
69             yaml_block = outline_list.split('```yaml')[1].split('```')[0]
70         else:
71             yaml_block = outline_list
72         parts = yaml.safe_load(yaml_block)
73         if isinstance(parts, dict):
74             parts = list(parts.values())
75     except Exception as e:
76         print(f"[大纲yaml解析失败] {e}")
77         parts = []
78     return parts
79

```

这个脚本的目标是基于 `research_report_generator.py` 生成的初步研报汇总（一个 Markdown 文件），利用大语言模型（LLM）的能力，将其深化和扩展，生成一份更详尽、结构更清晰的深度研究报告。

1. 预处理与加载：

- 图片路径本地化 (`extract_images_from_markdown`)：
 - 这是非常关键的一步。脚本首先会处理输入的 Markdown 文件（ `raw_md_path` ，这里写死为 "财务研报汇总_20250608_074539.md"，实际使用中应该是一个动态的输入）。
 - 它会查找 Markdown 中所有的图片引用（ `` 格式）。

- 如果图片是网络链接 (URL)，它会尝试下载图片到本地一个新建的 `images` 目录下（与输入的 Markdown 文件同级）。
- 如果图片是本地路径，它会尝试复制图片到这个 `images` 目录。
- 所有成功处理的图片，在 Markdown 中的路径会被统一修改为相对路径，如 `./images/图片名.png`。
- 如果图片下载或复制失败（比如链接失效或本地文件不存在），该图片的引用会从 Markdown 中被移除。
- 处理后的新 Markdown 内容会保存到一个新的文件（`new_md_path`，例如 "财务研报汇总_20250608_074539_images.md"）。
- **目的：** 确保所有图片资源都本地化，并且路径统一，这对于后续 LLM 处理（避免访问外部链接）以及最终生成可移植的报告（如转换为 DOCX）至关重要。

◦ **加载内容与背景：**

- `load_report_content`：读取经过图片路径处理后的 Markdown 文件内容。
- `get_background`：提供一段固定的背景信息文本，说明原始数据的来源和处理方式（如数据采集自 akshare、东方财富，分析由大模型生成等）。这有助于 LLM 理解输入材料的上下文。
- `get_llm`：初始化并返回一个 `LLMHelper` 实例，用于后续与大语言模型的交互。

2. 生成研报大纲 (`generate_outline`):

- 脚本会构建一个提示 (`outline_prompt`)，要求 LLM 基于提供的背景信息和已加载的研报汇总内容，为《商汤科技公司研报》生成一个分段大纲。
- 这个大纲被要求以 YAML 格式输出，包含每个部分的标题 (`part_title`) 和简介 (`part_desc`)。
- 大纲的章节需要覆盖公司基本面、财务分析、行业对比、估值与预测、治理结构、投资建议、风险提示、数据来源等关键领域。
- LLM 返回的大纲 (YAML 格式) 会被解析成一个列表，其中每个元素代表报告的一个主要部分。

3. 分段生成研报内容 (`generate_section` 和 `main` 循环):

- 脚本进入一个循环，遍历上一步生成的大纲中的每一个部分。
- 对于每个部分：

- 它会调用 `generate_section` 函数，并传入当前部分的标题、之前已生成的所有报告内容（`prev_content`，用于上下文保持）、背景信息、原始研报汇总内容，以及一个标记（`is_last`）来判断是否是最后一个部分。
- `generate_section` 函数会构建另一个更具体的提示（`section_prompt`），要求 LLM 针对当前部分标题，直接输出该部分的完整研报内容。
- 对 LLM 的严格要求：
 - 直接输出 Markdown 格式的正文，以 `## 章节标题` 开头。
 - 在引用数据、图片时，使用特定的引用符号（如 `[1][2][3]`）并在文末（如果是最后一部分）列出参考文献。
 - **极其严格的图片引用规则：**只允许引用原始研报汇总内容中真实存在的、且路径已被预处理为 `./images/图片名.png` 格式的图片。严禁虚构、改编或猜测图片地址。如果 LLM 引用了不存在的图片，会被视为错误。
 - 禁止输出任何提示性语言（如“建议补充”）或编造内容。
- LLM 生成的该部分内容会被追加到 `full_report` 列表中。
- `prev_content` 会更新为当前已生成的所有报告内容，以便下一部分的生成能够参考前面的内容，因为模型的上下文通常要比最大输出内容长很多，使用输出的内容加入到之前的记忆里面反复让模型进行输出就能超过单次让模型直接输出的内容长度。

4. 汇总与输出：

- 所有部分的内容都生成完毕后，`full_report` 列表中的所有文本段落会用换行符连接起来，形成最终的深度研报 Markdown 文本。
- `save_markdown`：将这份完整的深度研报保存为一个新的 Markdown 文件，文件名包含时间戳（如“深度财务研报分析_YYYYMMDD_HHMMSS.md”）。
- `format_markdown`（可选）：尝试使用 `mdformat` 工具对生成的 Markdown 文件进行格式化，使其更规范。
- `convert_to_docx`（可选）：尝试使用 `pandoc` 工具将 Markdown 文件转换为 Word（`.docx`）文档。这里特别注意了 `--resource-path=.` 和 `--extract-media=.` 选项，以帮助 `pandoc` 正确处理本地图片。

主函数（`main`）

`main` 函数负责按顺序执行上述所有步骤：从图片预处理开始，到加载内容，生成大纲，然后逐段生成报告内容，最后保存、格式化和转换报告。

核心思想：

in_depth_research_report_generator.py 的核心思想是分治策略和精细化的提示工程。它不是让 LLM 一口气生成整个深度报告（这通常很难控制质量和结构），而是：

1. 先让 LLM 生成一个结构化的大纲。
2. 然后针对大纲的每一个小部分，让 LLM 集中精力生成该部分的内容，同时提供足够的上下文（前文、背景、原始材料）和非常严格的输出规范（尤其是图片引用）。

这种方式使得生成长篇、结构复杂且内容要求精确的报告成为可能，并通过预处理（如图片本地化）和后处理（格式化、转换）来提升最终输出的质量和可用性。

集成分析报告生成、资料收集、公司分析报告

上面两个部分是分离的，大家使用的时候会不方便，所以，我给大家准备了集成好两部分的代码内容，下面是集成了上面提到的两部分内容的代码。

代码块

```
1  """
2  整合的金融研报生成器
3  包含数据采集、分析和深度研报生成的完整流程
4  - 第一阶段：数据采集与基础分析
5  - 第二阶段：深度研报生成与格式化输出
6  """
7
8  import os
9  import glob
10 import time
11 import json
12 import yaml
13 import re
14 import shutil
15 import requests
16 from datetime import datetime
17 from dotenv import load_dotenv
18 import importlib
19 from urllib.parse import urlparse
20
21 from data_analysis_agent import quick_analysis
22 from data_analysis_agent.config.llm_config import LLMConfig
23 from data_analysis_agent.utils.llm_helper import LLMHelper
24 from utils.get_shareholder_info import get_shareholder_info, get_table_content
25 from utils.get_financial_statements import get_all_financial_statements,
26     save_financial_statements_to_csv
27 from utils.identify_competitors import identify_competitors_with_ai
```

```

27 from utils.get_stock_intro import get_stock_intro, save_stock_intro_to_txt
28 from duckduckgo_search import DDGS
29
30 class IntegratedResearchReportGenerator:
31     """整合的研报生成器类"""
32
33     def __init__(self, target_company="商汤科技", target_company_code="00020",
34 target_company_market="HK"):
35         # 环境变量与全局配置
36         load_dotenv()
37         self.api_key = os.getenv("OPENAI_API_KEY")
38         self.base_url = os.getenv("OPENAI_BASE_URL",
39 "https://api.openai.com/v1")
40         self.model = os.getenv("OPENAI_MODEL", "gpt-4")
41
42         self.target_company = target_company
43         self.target_company_code = target_company_code
44         self.target_company_market = target_company_market
45
46         # 目录配置
47         self.data_dir = "./download_financial_statement_files"
48         self.company_info_dir = "./company_info"
49         self.industry_info_dir = "./industry_info"
50
51         # 创建必要的目录
52         for dir_path in [self.data_dir, self.company_info_dir,
53 self.industry_info_dir]:
54             os.makedirs(dir_path, exist_ok=True)
55
56         # LLM配置
57         self.llm_config = LLMConfig(
58             api_key=self.api_key,
59             base_url=self.base_url,
60             model=self.model,
61             temperature=0.7,
62             max_tokens=16384,
63         )
64         self.llm = LLMHelper(self.llm_config)
65
66         # 存储分析结果
67         self.analysis_results = {}
68
69     def stage1_data_collection(self):
70         """第一阶段：数据采集与基础分析"""
71         print("\n" + "="*80)
72         print("🚀 开始第一阶段：数据采集与基础分析")
73         print("="*80)

```

```

71
72     # 1. 获取竞争对手列表
73     print("🔍 识别竞争对手...")
74     other_companies = identify_competitors_with_ai(
75         api_key=self.api_key,
76         base_url=self.base_url,
77         model_name=self.model,
78         company_name=self.target_company
79     )
80     listed_companies = [company for company in other_companies if
81                          company.get('market') != "未上市"]
82
83     # 2. 获取目标公司财务数据
84     print(f"\n📊 获取目标公司 {self.target_company} 的财务数据...")

```

Baseline运行说明

1. 克隆项目

代码块

```

1  git clone https://github.com/li-xiu-qi/financial_research_report
2  cd financial_research_report

```

2. 安装依赖

代码块

```

1  pip install -r requirements.txt

```

3. 环境配置

创建 `.env` 文件并配置以下变量：

代码块

```

1  OPENAI_API_KEY=your_openai_api_key
2  OPENAI_BASE_URL=https://api.openai.com/v1
3  OPENAI_MODEL=gpt-4

```

注意，本次baseline使用的是deepseek v3，如果你需要和作者达到一样的效果，那么可以选择如下配置

```
代码块
2 # 火山引擎配置
3 OPENAI_API_KEY=
4 OPENAI_BASE_URL=https://ark.cn-beijing.volces.com/api/v3
5 # 文本模型
6 OPENAI_MODEL=deepseek-v3-250324
7 # OPENAI_MODEL=deepseek-r1-250528
8
```

4. 生成基础研报

```
代码块
1 from research_report_generator import generate_report
2
3 # 配置目标公司
4 target_company = "商汤科技"
5 target_company_code = "00020"
6 target_company_market = "HK" # 生成研报
7 generate_report(target_company, target_company_code, target_company_market)
```

5. 生成整合式研报：

```
代码块
1 from integrated_research_report_generator import
  IntegratedResearchReportGenerator
2
3 # 创建生成器实例
4 generator = IntegratedResearchReportGenerator() # 运行完整流程
5 generator.run_full_pipeline()
```

6. 执行行业研究工作流：

```
代码块
1 from industry_workflow import IndustryResearchFlow
2 from pocketflow import Flow
3
4 # 创建行业研究流程
5 flow = Flow()
6 flow.add_node("industry_research", IndustryResearchFlow()) # 设置研究参数
7 shared_data = {"industry": "人工智能", "context": []} # 执行研究流程
8 flow.run(shared_data)
```

baseline方案思路

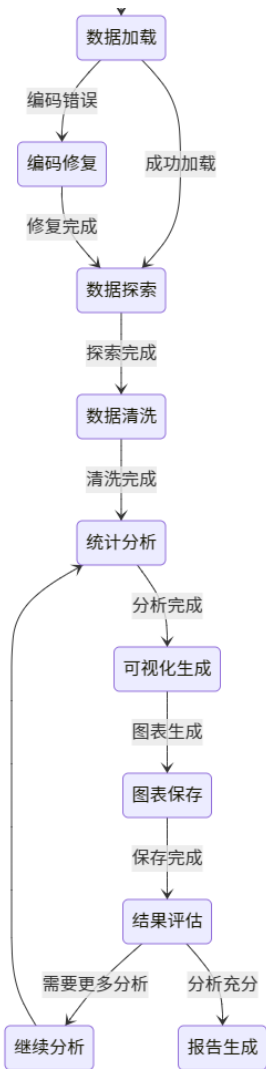
本次的baseline没有使用多智能体的方式实现，只使用工作流结合智能体的逻辑实现公司研报的生成。

主要是参考pocketflow的一些官方的示例思路。这样做

- 优点是实现快，容易修改，方便理解，
- 缺点是智能体的功能比较局限，泛化性比较差，运行的时候经常会出现一些bad case，比如模型记忆爆炸导致报错，格式化输出控制出现问题，导致工作流没法很好稳定输出每次都一样的研究报告，其中数据分析智能体发挥也存在不稳定的情况，比如两次分析生成的报告会有很大差异也是常有的。

baseline核心逻辑

核心逻辑是数据分析的智能体，以及代码执行器的逻辑。数据分析智能体的整体流程是这样的：



是一种基于观察反复调整结果的思路，参考单个智能体的ReAct思路实现。

代码细节的部分还设计到一些记忆管理的设计，比如专门设计了一个图片收集，分析结果收集，对话历史交互记录，分别是这些记忆，每个不同的环节会用到不同的记忆，也有混合一起使用的部分，这样能够使得模型在非无限上下文的情况下能够有效提升模型在多个领域知识的泛化性以及记忆溢出的问题（超过了最大上下文）。

当然，还有最重要的一个整体生成逻辑流程，具体是分成两个部分。

阶段一：数据采集与基础分析 (stage1_data_collection)

1. 初始化:

- 加载环境变量 (API密钥、模型名称等)。
- 设置目标公司信息 (名称、代码、市场)。
- 配置数据存储目录 (download_financial_statement_files, company_info, industry_info) 并创建它们。
- 初始化大语言模型 (LLM) 配置和助手。

2. 数据采集:

- **识别竞争对手:** 使用 AI (如 GPT-4) 识别目标公司的竞争对手。
- **获取财务数据:**
 - 获取目标公司的财务报表 (年度)。
 - 获取已上市竞争对手的财务报表 (年度)。
 - 将所有财务报表保存为 CSV 文件到 download_financial_statement_files 目录。
- **获取公司基础信息:**
 - 获取目标公司和竞争对手的公司简介。
 - 将公司简介保存为 TXT 文件到 company_info 目录。
- **搜索行业信息:**
 - 使用 DuckDuckGo 搜索目标公司和竞争对手的行业地位、市场份额、业务模式等信息。
 - 将搜索结果保存为 JSON 文件到 industry_info 目录。

3. 基础分析:

- **运行财务分析:**
 - 对每个公司单独进行财务数据分析 (使用 `quick_analysis` 函数)。
 - 对目标公司与每个竞争对手进行两两对比分析。
 - 对特定公司（如商汤科技）进行估值与预测分析。

- **整理分析结果:**
 - 整理从 `company_info` 目录中读取的公司信息。
 - 获取并分析目标公司的股东信息。
 - 整理从 `industry_info` 目录中读取的行业搜索结果。

4. 保存阶段一报告:

- 将所有收集和分析的信息汇总，格式化并保存为一个 Markdown 文件 (`财务研报汇总_{timestamp}.md`)。
- 将阶段一的关键结果存储在 `self.analysis_results` 中，供阶段二使用。

阶段二：深度研报生成 (stage2_deep_report_generation)

1. 准备:

- **处理图片路径:** 从阶段一生成的 Markdown 文件中提取图片，将图片统一存放到 `images` 目录，并更新 Markdown 文件中的图片路径。
- 加载处理图片路径后的 Markdown 文件内容。
- 获取报告的背景信息 (数据来源说明等)。

2. 生成深度研报:

- **生成大纲:** 使用 LLM 根据背景信息和阶段一的报告内容，生成一个详细的研报大纲 (YAML 格式)。
- **分段生成研报:**
 - 遍历大纲中的每个部分。
 - 使用 LLM 针对每个部分标题，结合已生成的前文、背景信息和阶段一报告内容，生成该部分的详细研报内容。
 - 在生成过程中，会强调引用【财务研报汇总内容】中真实存在的图片，并要求在文末列出引用文献。

3. 保存与格式化:

- 将所有分段生成的研报内容合并，保存为一个新的 Markdown 文件 (`深度财务研报分析_{timestamp}.md`)。
- 使用 `mdformat` 工具格式化生成的 Markdown 文件。
- 使用 `pandoc` 工具将 Markdown 文件转换为 Word 文档 (.docx)。

整体流程 (run_full_pipeline)

1. 调用 `stage1_data_collection` 完成数据采集和基础分析，并获取基础分析报告的路径。
2. 调用 `stage2_deep_report_generation`，传入基础分析报告的路径，生成深度研报。

辅助方法:

- 获取和处理公司信息、财务文件。
- 调用 `quick_analysis` 进行单公司分析和对比分析。
- 格式化和合并报告内容。
- 加载报告内容、获取背景信息、生成大纲和章节。
- 保存 Markdown 文件、格式化 Markdown、转换为 Word 文档。
- 处理图片：确保目录存在、判断是否为 URL、下载图片、复制图片、从 Markdown 中提取和替换图片路径。

- 创建一个 `IntegratedResearchReportGenerator` 实例，指定目标公司。
- 调用 `run_full_pipeline` 方法执行完整的研报生成流程。
- 打印最终生成的报告文件路径。

该脚本通过一个自动化的两阶段流程，实现了从数据采集、基础分析到深度研报生成的完整功能。它利用了外部库 (如 `requests` , `duckduckgo_search` , `pandoc` , `mdformat`) 和大语言模型来完成各项任务，旨在高效地生成结构化、内容详实的金融研究报告。

1. 如何优化输入数据的质量?

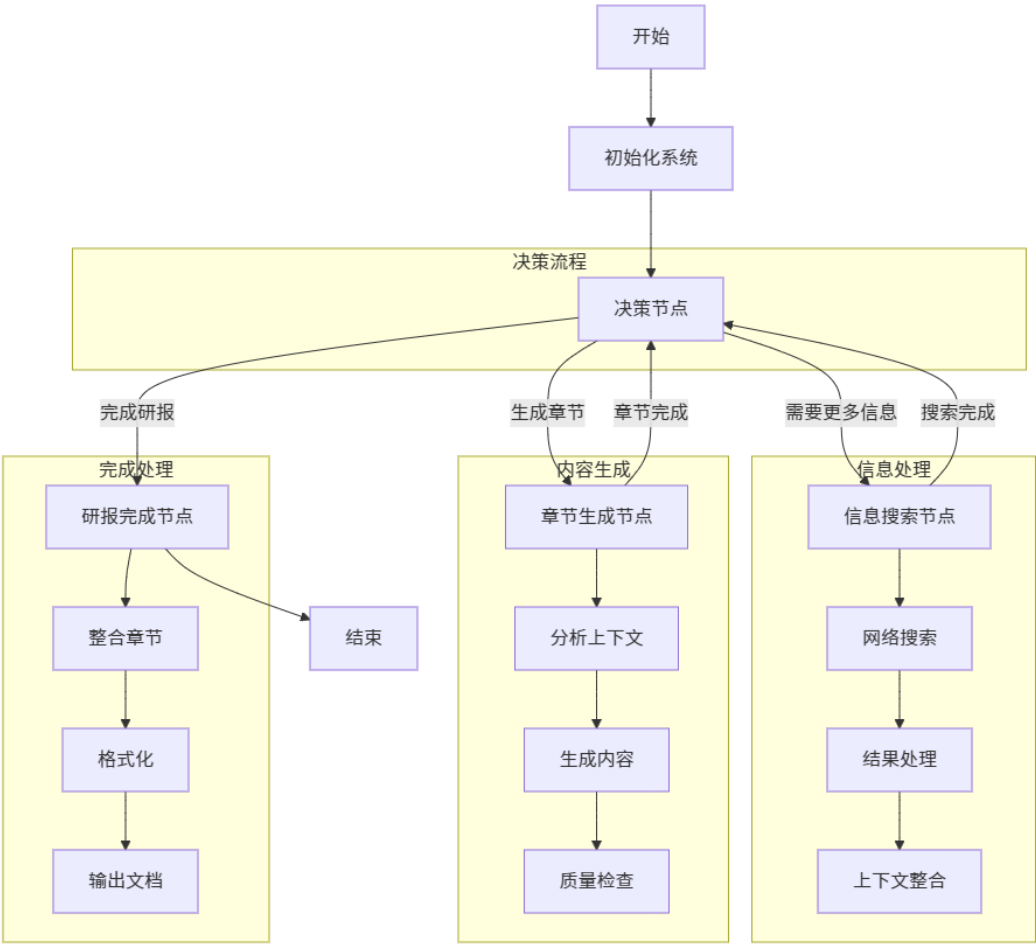
对于想要继续提示我们的研报生成系统的生成质量部分，我建议可以从数据源入手，没有优秀的质量源，高质量的研报也就无从而谈，就像我们常说的高质量入，高质量出，输入的全是垃圾，输出自然难以好到哪里去。

在本次的baseline里面，我们给出了多种获取数据的途径，只要你能够认真从我给出的数据源里面找到合适的数据源，我觉得一定是没有问题的。当然了，你也可以自己尝试找找有没有更优秀的数据源。

2. 如何优化处理流程？

在公司研报的部分，我们只是使用了一个自主决策的Agent进行数据分析并在最后的部分生成分析报告，并结合一些搜索的内容以及额外爬取的一些信息，这样看起来似乎是有些简陋的，还有非常大的优化空间，进一步的处理流程是[优化Agent的调度，分析，记忆管理，生成管理](#)等等，

这里我给出一个典型的deepresearch结构的Agent流程给大家参考：



参考代码如下：

代码块

```

1  import os
2  import time
3  import yaml
4  import openai
5  from duckduckgo_search import DDGS
6
7  from dotenv import load_dotenv
8  # 加载环境变量
9  load_dotenv()
10
11 # 从环境变量中初始化 OpenAI API 密钥
12 openai.api_key = os.getenv("OPENAI_API_KEY")
13 openai.api_base = os.getenv("OPENAI_BASE_URL", "https://api.openai.com/v1")
14 import asyncio, warnings, copy, time
15
16 class BaseNode:
17     def __init__(self): self.params, self.successors={}, {}
18     def set_params(self, params): self.params=params
19     def next(self, node, action="default"):
20         if action in self.successors: warnings.warn(f"Overwriting successor
for action '{action}'")
21         self.successors[action]=node; return node
22     def prep(self, shared): pass
23     def exec(self, prep_res): pass
24     def post(self, shared, prep_res, exec_res): pass
25     def _exec(self, prep_res): return self.exec(prepare_res)
26     def _run(self, shared): p=self.prep(shared); e=self._exec(p); return
self.post(shared, p, e)
27     def run(self, shared):
28         if self.successors: warnings.warn("Node won't run successors. Use
Flow.")
29         return self._run(shared)
30     def __rshift__(self, other): return self.next(other)
31     def __sub__(self, action):
32         if isinstance(action, str): return _ConditionalTransition(self, action)
33         raise TypeError("Action must be a string")
34
35 class _ConditionalTransition:
36     def __init__(self, src, action): self.src, self.action=src, action
37     def __rshift__(self, tgt): return self.src.next(tgt, self.action)
38
39 class Node(BaseNode):
40     def __init__(self, max_retries=1, wait=0): super().__init__();
self.max_retries, self.wait=max_retries, wait
41     def exec_fallback(self, prep_res, exc): raise exc
42     def _exec(self, prep_res):
43         for self.cur_retry in range(self.max_retries):

```

```

44         try: return self.exec(prepare_res)
45         except Exception as e:
46             if self.cur_retry==self.max_retries-1: return
self.exec_fallback(prepare_res,e)
47             if self.wait>0: time.sleep(self.wait)
48
49     class BatchNode(Node):
50         def _exec(self,items): return [super(BatchNode,self)._exec(i) for i in
(items or [])]
51
52     class Flow(BaseNode):
53         def __init__(self,start=None): super().__init__(); self.start_node=start
54         def start(self,start): self.start_node=start; return start
55         def get_next_node(self,curr,action):
56             nxt=curr.successors.get(action or "default")
57             if not nxt and curr.successors: warnings.warn(f"Flow ends: '{action}'
not found in {list(curr.successors)}")
58             return nxt
59         def _orch(self,shared,params=None):
60             curr,p,last_action =copy.copy(self.start_node),(params or
{**self.params}),None
61             while curr: curr.set_params(p); last_action=curr._run(shared);
curr=copy.copy(self.get_next_node(curr,last_action))
62             return last_action
63         def _run(self,shared): p=self.prepare(shared); o=self._orch(shared); return
self.post(shared,p,o)
64         def post(self,shared,prepare_res,exec_res): return exec_res
65
66     class BatchFlow(Flow):
67         def _run(self,shared):
68             pr=self.prepare(shared) or []
69             for bp in pr: self._orch(shared,{**self.params,**bp})
70             return self.post(shared,pr,None)
71
72     class AsyncNode(Node):
73         async def prepare_async(self,shared): pass
74         async def exec_async(self,prepare_res): pass
75         async def exec_fallback_async(self,prepare_res,exc): raise exc
76         async def post_async(self,shared,prepare_res,exec_res): pass
77         async def _exec(self,prepare_res):
78             for i in range(self.max_retries):

```

当然，还有宏观研报的版本，主要是调整了提示词：

代码块

```
1 import os
```

```

2 import time
3 import yaml
4 import openai
5 from duckduckgo_search import DDGS
6 # from pocketflow import Node, Flow 可以通过pip install pocketflow, 也可以自己复制
   源码
7 from dotenv import load_dotenv
8 # 加载环境变量
9 load_dotenv()
10
11 # 从环境变量中初始化 OpenAI API 密钥
12 openai.api_key = os.getenv("OPENAI_API_KEY")
13 openai.api_base = os.getenv("OPENAI_BASE_URL", "https://api.openai.com/v1")
14
15 class IndustryResearchFlow(Node): # 研报生成的决策节点
16     def prep(self, shared):
17         context = shared.get("context", [])
18         context_str = yaml.dump(context, allow_unicode=True)
19         industry = shared["industry"] # 行业名称
20         focus_areas = shared.get("focus_areas", [])
21         return industry, context_str, focus_areas
22
23     def exec(self, inputs):
24         industry, context, focus_areas = inputs
25         print(f"\n正在分析{industry}研究进度...")
26         prompt = f"""
27 针对宏观经济研究，基于已有信息：{context}
28 关注指标：{'，' .join(focus_areas)}
29
30 请分析并判断下一步行动：
31 1) 搜索更多信息
32 2) 开始生成某个章节内容
33 3) 完成研报生成
34
35 要特别关注：
36 - 核心经济指标数据（GDP、CPI、利率、汇率）的最新数据和趋势
37 - 重要政策文件与解读（如货币政策、财政政策等）
38 - 区域经济对比数据与分析
39 - 全球经济形势与联动影响
40 - 潜在风险因素与预警信号
41
42 请以 YAML 格式输出：
43 ```yaml
44 action: search/generate/complete # search表示继续搜索，generate表示生成章节，
   complete表示完成
45 reason: 做出此判断的原因
46 search_terms: # 如果是search，列出要搜索的关键词列表

```

```

47     - 关键词1
48     - 关键词2
49     section: # 如果是generate, 指定要生成的章节名称
50         name: 章节名称 # 如: 宏观经济概况/政策解读/风险分析等
51         focus: 重点关注内容 # 具体要分析的要点
52     """
53     resp = call_llm(prompt)
54     yaml_str = resp.split("`yaml`")[1].split("`", 1)[0].strip()
55     result = yaml.safe_load(yaml_str)
56
57     # 打印决策结果
58     print(f"决策结果: {result['action']}")
59     print(f"决策原因: {result['reason']}")
60     if result['action'] == 'search':
61         print("需要搜索的关键词:", result['search_terms'])
62     elif result['action'] == 'generate':
63         print(f"即将生成章节: {result['section']['name']}")
64
65     return result
66
67     def post(self, shared, prep_res, exec_res):
68         action = exec_res.get("action")
69         if action == "search":
70             shared["search_terms"] = exec_res.get("search_terms", [])
71             print("\n=== 开始信息搜索阶段 ===")
72         elif action == "generate":
73             shared["current_section"] = exec_res.get("section", {})
74             print("\n=== 开始章节生成阶段 ===")
75         elif action == "complete":
76             print("\n=== 开始完成研报阶段 ===")
77         return action
78
79     class SearchInfo(Node): # 信息搜索节点
80         def prep(self, shared):
81             return shared.get("search_terms", [])
82
83         def exec(self, search_terms):
84             all_results = []
85             total = len(search_terms)
86             for i, term in enumerate(search_terms, 1):

```

小提示：这两个部分的内容其实是可以结合公司研报的部分使用的数据分析进行结合，构建出来一个更优秀的研报生成管道，感兴趣的同学可以试试啦~

或者你还有更好的思路吗？

附录：补充知识

金融术语：ROE分解之三步杜邦分析法

三步杜邦分析法 (Three-Step DuPont Analysis)

这是最经典和最常用的ROE分解方式。它将ROE分解为三个核心部分：**盈利能力**、**资产管理效率**和**财务杠杆**。

计算公式为：

$$\text{ROE} = \text{净利润率} \times \text{总资产周转率} \times \text{权益乘数}$$

财务比率通常分为以下几个类别：

1. 盈利能力比率 (Profitability Ratios)

这类比率衡量公司创造利润的能力。

◦ 销售毛利率 (Gross Profit Margin)

- 公式: $\text{销售毛利率} = \frac{\text{营业收入} - \text{营业成本}}{\text{营业收入}} \times 100$

- 说明: 反映了公司产品或服务的初始获利空间。

◦ 销售净利率 (Net Profit Margin)

- 公式: $\text{销售净利率} = \frac{\text{净利润}}{\text{营业收入}} \times 100$

- 说明: 衡量公司最终的盈利能力，即扣除所有成本和费用（包括税收）后的利润水平。

◦ 净资产收益率 (ROE - Return on Equity)

- 公式: $\text{ROE} = \frac{\text{净利润}}{\text{股东权益平均余额}} \times 100$

- 说明: 衡量股东投入资本的回报水平，是股东最关心的指标之一。

◦ 总资产回报率 (ROA - Return on Assets)

- 公式: $\text{ROA} = \frac{\text{净利润}}{\text{总资产平均余额}} \times 100$

- 说明: 衡量公司利用其所有资产（无论来自股东还是债权人）来创造利润的效率。

2. 偿债能力比率 (Solvency Ratios)

这类比率衡量公司偿还债务的能力，分为短期和长期。

A. 短期偿债能力

◦ 流动比率 (Current Ratio)

- 公式: $\text{流动比率} = \frac{\text{流动资产}}{\text{流动负债}}$

- 说明: 衡量公司用流动资产偿还短期债务的能力。通常认为该比率在2左右较为理想。

- **速动比率 (Quick Ratio / Acid-Test Ratio)**

- **公式:** $\text{速动比率} = \frac{\text{流动资产} - \text{存货}}{\text{流动负债}}$

- **说明:** 比流动比率更严格，剔除了变现能力较慢的存货。通常认为该比率在1左右较为安全。

B. 长期偿债能力

- **资产负债率 (Debt-to-Asset Ratio)**

- **公式:** $\text{资产负债率} = \frac{\text{总负债}}{\text{总资产}} \times 100$

- **说明:** 反映公司的总资产中有多少是通过负债筹集的，衡量公司的财务风险。

- **产权比率 (Debt-to-Equity Ratio)**

- **公式:** $\text{产权比率} = \frac{\text{总负债}}{\text{股东权益}}$

- **说明:** 衡量公司的负债相对于股东权益的水平，是另一个衡量财务杠杆的重要指标。

- **利息保障倍数 (Interest Coverage Ratio)**

- **公式:** $\text{利息保障倍数} = \frac{\text{息税前利润 (EBIT)}}{\text{利息费用}}$

- **说明:** 衡量公司盈利覆盖长期债务利息的能力。倍数越高，偿付利息的能力越强。

3. 运营效率比率 (Efficiency Ratios)

这类比率衡量公司利用资产进行运营管理的效率。

- **总资产周转率 (Total Asset Turnover)**

- **公式:** $\text{总资产周转率} = \frac{\text{营业收入}}{\text{总资产平均余额}}$

- **说明:** 反映公司利用其全部资产创造销售收入的效率。

- **应收账款周转率 (Accounts Receivable Turnover)**

- **公式:** $\text{应收账款周转率} = \frac{\text{营业收入}}{\text{应收账款平均余额}}$

- **说明:** 衡量公司收回应收账款的速度。周转率越高，资金回笼越快。

- **存货周转率 (Inventory Turnover)**

- **公式:** $\text{存货周转率} = \frac{\text{营业成本}}{\text{存货平均余额}}$

- **说明:** 衡量公司销售存货的速度。周转率越高，存货占用资金的时间越短，管理效率越高。

4. 市场价值比率 (Market Value Ratios)

这类比率主要用于上市公司，将公司股价与财务数据关联起来。

- **每股收益 (EPS - Earnings Per Share)**

- **公式:** $\text{EPS} = \frac{\text{净利润} - \text{优先股股利}}{\text{发行在外的普通股加权平均数}}$

- **说明:** 衡量每一普通股所能获得的利润。

- **市盈率 (P/E Ratio - Price-to-Earnings Ratio)**

- **公式:**

$$\text{市盈率} = \frac{\text{每股市价}}{\text{每股收益}}$$

- **说明:** 最常用的估值指标之一，反映了投资者愿意为每一元利润支付的价格。

- **市净率 (P/B Ratio - Price-to-Book Ratio)**

- **公式:**

$$\text{市净率} = \frac{\text{净利润}}{\text{营业收入}} \times 100$$

- **说明:** 反映了投资者愿意为每一元净资产支付的价格。

金融术语：趋势分析

趋势分析的目标就是判断市场方向，顺势而为。

定义与识别趋势：

- **上升趋势 (Uptrend):** 价格走势出现一系列“更高的高点 (Higher Highs)”和“更高的低点 (Higher Lows)”。市场情绪乐观，策略以逢低做多为主。
- **下降趋势 (Downtrend):** 价格走势出现一系列“更低的高点 (Lower Highs)”和“更低的低点 (Lower Lows)”。市场情绪悲观，策略以逢高做空或持币观望为主。
- **盘整/横盘趋势 (Sideways/Ranging):** 价格在一定区间内波动，没有明确的向上或向下方向。策略以高抛低吸或等待趋势突破为主。

趋势核心分析工具：

- **趋势线与通道 (Trendlines & Channels):**
 - **画法:** 在上升趋势中，连接两个或多个关键的低点；在下降趋势中，连接两个或多个关键的高点。
 - **作用:** 趋势线是动态的支撑位或阻力位。价格回踩趋势线不破，是潜在的入场信号。价格有效跌破上升趋势线或突破下降趋势线，是趋势可能反转的信号。通道则定义了趋势运行的上下轨。
- **K线形态 (Candlestick Patterns):**
 - **反转形态:** 如锤子线 (Hammer)、看涨吞没 (Bullish Engulfing)、黄昏之星 (Evening Star)等，预示着当前趋势可能即将结束。
 - **持续形态:** 如上升三法 (Rising Three Methods)，表明趋势在短暂休整后大概率会继续。
- **经典图表形态 (Classic Chart Patterns):**

- **反转形态:** 头肩顶/底 (Head and Shoulders)、双重顶/底 (Double Top/Bottom)、V形反转等，通常出现在趋势的顶部或底部。
- **持续形态:** 三角形 (Triangles)、旗形 (Flags)、矩形 (Rectangles)等，表示市场在为下一波行情蓄力。

金融术语：技术指标计算

技术指标的计算目标是获取量化趋势的强度、市场的超买超卖状态以及波动性，从而找到精确的入场和出场时机。

趋势型指标 (Trend-Following Indicators)

- 用于确认趋势
- **移动平均线 (Moving Average, MA):**
 - **计算:** 一定周期内收盘价的算术平均值 (SMA) 或指数加权平均值 (EMA)。
 - **应用:** MA的方向指示趋势方向。短期MA上穿长期MA形成“**金叉**”（买入信号），反之形成“**死叉**”（卖出信号）。价格在MA上方得到支撑，在下方受到压制。
- **MACD (平滑异同移动平均线):**
 - **计算:** 基于两条EMA（通常是12日和26日）的差值（DIF），再对DIF进行平滑处理得到信号线（DEA）。
 - **应用:** DIF上穿DEA形成金叉；DIF和DEA在零轴上方为多头市场，下方为空头市场；**顶背离**（价格新高，指标未新高）和**底背离**（价格新低，指标未新低）是强烈的反转预警信号。

动量/震荡型指标 (Momentum/Oscillator Indicators)

- 用于判断超买超卖
- **RSI (相对强弱指数):**
 - **计算:** 通过比较一定时期内上涨和下跌的幅度来测量动能。
 - **应用:** 数值在0-100之间。通常高于70-80为**超买区**（Overbought），价格可能回调；低于20-30为**超卖区**（Oversold），价格可能反弹。同样，**背离**也是其重要信号。
- **KDJ (随机指标):**
 - **计算:** 基于统计学原理，通过当前收盘价在近期价格区间的相对位置来预测短期走势。
 - **应用:** 对价格短期波动非常敏感。K线在80以上为超买，20以下为超卖。K线上穿D线形成金叉。

波动率指标 (Volatility Indicators) - 用于衡量市场波动幅度

- **布林带 (Bollinger Bands, BOLL):**
 - **计算:** 由一条中轨（通常是20日SMA）和根据标准差计算出的上下轨构成。

- **应用:** 通道收窄 (**Squeeze**) 预示着即将出现大的单边行情。价格触及上轨是阻力, 触及下轨是支撑。价格沿着上轨或下轨运行, 表明趋势非常强劲。

金融术语: 风险评估

风险评估 (Risk Assessment), 目标是**保护本金**。这是决定长期能否在市场中生存和盈利的关键。交易不是为了预测百分百正确, 而是为了在正确时赚得足够多, 在错误时亏损得足够少。

设定止损 (Stop-Loss):

- **为什么:** 这是风险管理的核心。任何交易都有失败的可能, 止损是在入场前就已确定的最大可接受亏损。
- **如何设定:**
 - **技术位止损:** 设在关键的支撑位下方 (做多时) 或阻力位上方 (做空时), 例如前低、趋势线、MA等。
 - **百分比止损:** 设定一个固定的亏损比例, 如本金的2%。
 - **波动性止损:** 使用ATR (平均真实波幅) 指标来设定适应市场波动的止损位。

设定止盈 (Take-Profit):

- **为什么:** 将浮动盈利转化为实际利润, 克服贪婪。
- **如何设定:**
 - **目标位止盈:** 设在下一个关键阻力位 (做多时) 或支撑位 (做空时)。
 - **形态目标位:** 根据图表形态 (如头肩底的颈线到头部的距离) 来测算目标价位。
 - **移动止盈:** 随着价格向有利方向移动, 不断调高止损位, 以锁定利润。

仓位管理 (Position Sizing):

- **核心思想:** 每次交易投入多少资金? 这应该基于你的风险承受能力和止损位置来决定。
- **计算方法:**
 - **公式:** $\text{仓位数量} = (\text{总资金} \times \text{单次可接受风险百分比}) / (\text{入场价} - \text{止损价})$
 - **举例:** 10万元资金, 愿意承担2%的风险 (即2000元)。如果股票入场价20元, 止损价18元, 每股风险为2元。那么可以买入的仓位就是 $2000 \text{元} / 2 \text{元/股} = 1000 \text{股}$ 。

评估风险回报比 (Risk/Reward Ratio):

- **定义:** $(\text{止盈价} - \text{入场价}) / (\text{入场价} - \text{止损价})$
- **应用:** 只参与那些潜在回报至少是潜在风险2倍或3倍以上的交易 (即R/R Ratio $\geq 2:1$ 或 $3:1$)。这能确保即使胜率不高 (比如只有50%), 长期下来依然可以实现盈利。

公司研报核心数据说明

1. 公司基础财务数据

◦ 三大财务报表 (Financial Statements)

- **利润表 (Income Statement):** 用于计算盈利能力指标，如毛利率、净利率、营业收入增长率等。需要获取**历年及各季度**的数据，以便进行趋势分析。
 - **关键字段:** 营业总收入、营业成本、销售费用、管理费用、研发费用、财务费用、营业利润、净利润等。
- **资产负债表 (Balance Sheet):** 用于分析公司的资产结构、偿债能力和运营效率。同样需要**历年及各季度**数据。
 - **关键字段:** 货币资金、应收账款、存货、流动资产合计、固定资产、总资产、短期借款、应付账款、流动负债合计、总负债、所有者权益等。
- **现金流量表 (Cash Flow Statement):** 用于评估公司的现金获取能力、偿债能力和投资活动。
 - **关键字段:** 经营活动产生的现金流量净额、投资活动产生的现金流量净额、筹资活动产生的现金流量净额、现金及现金等价物净增加额等。

◦ 财务报表附注 (Notes to Financial Statements)

- 虽然 `akshare` 可能无法直接提供结构化的附注数据，但这部分信息对于理解会计政策、重大交易、关联方关系等至关重要。必要时，需要从公司年报（PDF/HTML格式）中进行**手动或通过NLP技术提取**。

2. 股权与公司治理数据

3. 市场与行业数据

4. 估值与预测所需数据

这些数据是构建估值模型和进行未来财务预测的基础。

◦ 历史股价数据 (Historical Stock Prices)

- 用于计算 Beta 值（CAPM模型）、历史波动率（期权定价模型）以及进行技术分析。
`akshare` 提供日度、周度、月度的**前复权或后复权**股价数据。

◦ 关键变量数据 (Key Variables)

- 用于敏感性分析。例如：
 - **大宗商品价格:** 如果公司是制造业，需要获取其主要原材料（如石油、铜、锂）的历史和期货价格。
 - **汇率:** 对于有大量进出口业务的公司，需要主要交易货币（如美元、欧元）的汇率数据。

行业研报核心数据说明

核心数据：

- 1. 行业基础与宏观数据聚合 🌐
- 2. 行业结构与生命周期解读 🔄
- 3. 趋势分析与情景模拟 🎯
- 4. 策略建议与可视化 💡

分析的最终目的是为了指导决策。

- 行业进入与退出策略 (Entry & Exit Strategy)
 - **进入策略:** 基于情景分析，建议在何种条件下（如技术突破、政策利好）、以何种方式（如自主研发、收购兼并）进入该行业最有价值。
 - **退出策略:** 识别行业转向衰退的早期信号，为现有参与者提供退出或转型的时机与路径建议。
- 敏感性分析 (Sensitivity Analysis)
 - 量化分析**单一关键变量**（如原材料成本、汇率）的变动对行业整体利润率或龙头企业盈利能力的影响。例如：“上游A材料价格每上涨10%，行业平均净利率下降多少个百分点？”

数据获取渠道：

1. 企业财务报告

企业财报的获取途径因企业是否上市以及所在地区而有很大差异。

主要是上市公司财务报告。

这类报告通常信息最透明，获取渠道也相对规范。

- 证券交易所官方网站：
 - 中国大陆：
 - 上海证券交易所 (SSE) 官网 (www.sse.com.cn)：提供沪市上市公司公告及定期报告。
 - 深圳证券交易所 (SZSE) 官网 (www.szse.cn)：提供深市上市公司公告及定期报告。
 - 巨潮资讯网 (www.cninfo.com.cn)：中国证监会指定的上市公司信息披露平台，包含沪深两市、新三板、港股等市场的公告和定期报告（年报、半年报、季报）。这是查找中国大陆上市公司财报最权威、最全面的渠道之一。
 - 香港：
 - 香港交易所披露易 (HKEXnews) (www.hkexnews.hk)：香港上市公司法定信息披露平台。

- **公司官方网站：**
 - 绝大多数上市公司会在其官网的“投资者关系” (Investor Relations) 或类似板块发布年度报告、中期报告等财务文件。
- **商业财经数据库（通常付费，数据更结构化、全面，并提供分析工具）：**
 - **万得 (Wind) 资讯金融终端：** 中国市场占有率领先的金融数据和分析工具提供商，覆盖股票、债券、基金、宏观行业等各类数据，财报数据非常详尽。
 - **东方财富 Choice 数据：** 东方财富网推出的金融数据服务终端，提供全面的财经数据，包括上市公司财报。
 - **同花顺 iFinD 金融数据终端：** 另一家主流的金融数据服务商，提供详细的上市公司财务数据。

2. 协会年度报告

协会（如行业协会、商会、学会、基金会等非营利组织）的年度报告透明度和获取难度差异很大，大多是需要付费才能获取的。

- **协会官方网站：**
 - 许多规模较大、运作规范的协会会在其官方网站上发布年度工作报告、财务报告或审计报告，尤其是有公开募捐资格或接受政府资助的协会。

宏观经济研报核心数据说明

核心数据：

1. 宏观经济核心指标数据

2. 宏观经济建模

3. 全球视野模拟与风险预警

- **全球模拟建模模拟场景：**
 - “若美联储超预期加息100个基点，对人民币汇率、中国资本外流规模以及出口企业利润会产生多大的冲击？”
 - “若地缘政治冲突导致全球油价上涨50%，将如何影响中国的PPI，并最终传导至CPI？”
- **“灰犀牛”事件风险预警机制：**
 - “灰犀牛”指那些概率大、冲击力强，但却容易被忽视的风险。预警机制不是预测“黑天鹅”，而是监控“灰犀牛”的逼近。
 - **指标设计与监控：**

- **房地产风险:** 关键城市的“房价收入比”、“租售比”、房地产开发贷与个人房贷的违约率变化。
- **地方政府债务风险:** 地方融资平台（LGFV）债券的信用利差、土地财政依赖度、区域债务率。
- **资本外流风险:** 外汇储备变化、银行代客结售汇数据、离岸与在岸人民币汇差。

宏观经济数据获取渠道：

1. 官方及权威机构：

- **国家统计局 (中国):** 是获取中国宏观经济数据最权威的官方机构。其官方网站会定期发布GDP、CPI、PPI（工业生产者出厂价格指数）、PMI（采购经理指数）等重要经济指标。
- **中国人民银行 (中国):** 发布中国的利率政策、货币供应量、存贷款利率、社会融资规模等金融数据。
- **国家外汇管理局 (中国):** 发布中国的外汇储备、人民币汇率中间价、跨境资金流动等数据。
- **政府官方网站 (如中国政府网):** 常常会汇总发布重要的经济数据和解读。
- **其他国家央行和统计机构:** 例如美联储（Federal Reserve）等，会发布经济数据。

2. 第三方数据服务商：

CEIC: 提供覆盖全球210多个国家和地区的超过450万时间序列数据，包含宏观经济数据、行业数据等，这是一个付费的服务。

彭博 (Bloomberg)、路透 (Reuters)、万得 (Wind): 这些是专业的金融数据终端，提供非常全面和及时的全球宏观经济数据、市场行情和分析工具，但通常需要付费订阅。

财经门户网站: 如新浪财经、东方财富网、和讯网等，也会整合发布一些宏观经济数据，通常来源于官方机构。

政策影响数据获取渠道：

政策影响信息主要来源于政府部门、研究机构、新闻媒体以及行业协会等。

1. 政府官方渠道：

- **中央及地方政府门户网站:** 例如中国的“中华人民共和国中央人民政府”官网 (www.gov.cn) 及其下属各部委网站（如发改委、工信部、科技部、财政部等），会发布最新的政策法规、解读文件、规划纲要以及部分政策执行情况和评估报告。地方政府网站也会发布地方性政策及影响。
- **政策数据库/信息公开平台:** 一些国家和地区建有专门的法律法规数据库和政策公开平台，如中国的“国家法律法规数据库”、“中国政府法制信息网”等。

- **立法与听证记录：** 人大、政协等机构的官方网站会公布立法进程、草案征求意见稿及相关的讨论和说明，这些是理解政策出台背景和预期影响的重要参考。

2. 研究机构与智库 (Think Tanks):

- **官方背景研究机构：** 如国务院发展研究中心、中国社会科学院及其下属各研究所、地方社科院等，会发布大量关于宏观经济、产业政策、社会政策及其影响的深度研究报告。
- **高校研究中心：** 许多大学设有专门的政策研究中心或学院（如公共管理学院、经济学院等），其学术研究成果和发布的报告具有较高参考价值。

3. 新闻媒体与专业资讯平台：

- **主流财经媒体：** 如《人民日报》、《经济日报》、《财新》、《第一财经》、《华尔街日报》、《金融时报》等，通常设有政策解读和深度分析栏目。
- **行业媒体与资讯平台：** 针对特定行业的媒体会更深入地报道相关政策对对应行业的影响。

4. 行业协会与商会：

- 各行业协会会密切关注并分析相关政策对本行业企业的影响，并可能发布研究报告、行业动态或组织研讨会。

5. 企业与咨询公司报告：

- 大型企业（尤其是上市公司）在其年报或专项报告中可能会分析政策环境对其业务的影响。
- 咨询公司（如麦肯锡、波士顿咨询、普华永道、德勤等）会发布关于特定政策对产业或市场影响的分析报告。

技术演进信息数据获取渠道：

技术演进信息主要来源于科研机构、科技公司、行业报告、专利数据库、学术出版物等。

1. 学术与科研机构：

- **顶尖大学与研究机构：** 如麻省理工学院 (MIT)、斯坦福大学、清华大学、北京大学以及各国国家级实验室和科学院，其发布的研究成果、论文和技术报告是前沿技术演进的重要来源。
- **学术数据库与期刊：**
 - **中国知网 (CNKI)：** 收录大量中文学术期刊、学位论文、会议论文。
 - **万方数据知识服务平台、维普资讯**
 - **国际：** IEEE Xplore (电子电气工程)、ACM Digital Library (计算机科学)、ScienceDirect (Elsevier)、SpringerLink、Nature、Science 等顶级期刊和数据库。
 - **预印本服务器：** 如 [arXiv.org](https://arxiv.org)，可以快速获取最新的研究进展（但未经同行评审）。

2. 科技公司与行业领导者：

- 大型科技公司（如华为、阿里、腾讯、谷歌、微软、苹果、英伟达等）的官方博客、技术白皮书、开发者大会、年度报告等会披露技术研发布局和对未来技术趋势的看法。

- 初创科技公司的技术发布和融资信息也是观察新兴技术方向的窗口。

3. 专利数据库：

- 通过分析专利申请和授权情况，可以洞察技术研发的热点领域和未来趋势。
- **各国专利局数据库：** 如中国国家知识产权局 (CNIPA)、美国专利商标局 (USPTO)、欧洲专利局 (EPO)、世界知识产权组织 (WIPO) 的 Patentscope。
- **商业专利数据库：** 如 Derwent Innovation, PatSnap (智慧芽) 等提供更强大的检索和分析功能。

4. 行业会议与展览：

- 重要的行业会议（如 CES 消费电子展、MWC 世界移动通信大会、RSA Conference 信息安全会议等）是了解最新技术产品和行业趋势的绝佳场所。会议论文集和演讲视频也是重要信息来源。

5. 专业科技媒体与博客：

- 如 MIT Technology Review (麻省理工科技评论)、Wired (连线)、TechCrunch、36氪、虎嗅、机器之心、量子位等，它们会快速报道和解读最新的科技突破和产业动态。

6. 开源社区与技术论坛：

- 如 GitHub、Stack Overflow 等，可以观察到实际技术应用和开发者关注的热点。

参考资料

1. [GitHub - The-Pocket/PocketFlow: Pocket Flow: 100-line LLM framework. Let Agents build Agents!](#)
2. <https://openai.github.io/openai-agents-python/tools/>
3. <https://github.com/microsoft/autogen>
4. <https://akshare.akfamily.xyz/data/stock/stock.html#id40>
5. https://github.com/li-xiu-qi/data_analysis_agent
6. https://github.com/li-xiu-qi/financial_research_report
7. Python 学习文档 (中文)
 - **Python 官方教程 (简体中文版):** <https://docs.python.org/zh-cn/3/tutorial/index.html>
 - 这是学习 Python 最权威的起点，详细介绍了 Python 的基础语法、标准库以及许多高级特性。
 - **廖雪峰的 Python 教程:** <https://www.liaoxuefeng.com/wiki/1016959663602400>
 - 非常受欢迎的中文 Python 教程，内容通俗易懂，适合初学者。

- **菜鸟教程 - Python 3 教程:** <https://www.runoob.com/python3/python3-tutorial.html>
 - 提供了大量的实例，方便快速上手，适合查询和入门。
- **W3CSchool - Python 教程:** <https://www.w3cschool.cn/python/>
 - 另一个提供基础教程和实例的中文网站。

8. RAG (Retrieval Augmented Generation) 学习文档

一篇全景介绍RAG的公众号文章: <https://mp.weixin.qq.com/s/HH9OWCDMPHJocqVjslqooA>

9. OpenAI SDK 学习文档 🤖

- **OpenAI API 官方文档:** <https://platform.openai.com/docs/api-reference>
 - 虽然 OpenAI 的官方文档主要是英文，但由于其广泛应用，有很多中文的教程和实践指南。
- **OpenAI Cookbook :** <https://cookbook.openai.com/>
 - 官方提供的代码示例库，虽然是英文注释，但代码本身是 Python，结合具体场景，比较容易理解。可以配合翻译工具阅读说明。

10. Agent (AI Agent) 学习文档 🧠

- **微软 Autogen :** <https://microsoft.github.io/autogen/>
 - **autogen**是一个多智能体框架，里面有很多设计思想可以参考，特点是提供了一个可以自主执行代码的代理也有上下文管理的模块。
- **Openai agents sdk:** <https://openai.github.io/openai-agents-python/>
 - 由openai官方出品的多agent框架，特点是多agent任务传递移交比较方便。
- **PocketFlow :** <https://github.com/The-Pocket/PocketFlow>
 - **pocketflow**是一个使用100行代码构建的智能体开发框架，主要是以代码形态的工作流展现，**核心实现简洁且强大**，能够让你快速上手，且能理解其核心原理，没有过多的心理、认知负担，极简的抽象，代码非常优雅。