北　　京　　大　　学

大模型实习生测试项目

基于大模型的 CSV 数据分析系统

姓名：罗功成

2024 年 12 月 18 日

一、 任务描述

仿照 OpenAI 的 code interpreter 思想, 构建一个基于大模型的 CSV 数据分析系统。该系统需要具备以下功能:

1. 读取 CSV 数据: 系统应能指定路径并读取, 解析 CSV 文件中的数据。

2. 读取数据分析问题: 系统应能够接受命令行输入的数据分析请求, 数据分析请求以自然语言表达。且系统在响应完当前请求后, 具备基于当前历史继续响应下一个请求的能力。

3. 基于大模型的 Python 代码生成: 系统应使用大模型 (如 GPT, Qwen 等) 生成相应的 Python 代码来解决输入的数据分析问题。

4. 代码纠错: 当大模型生成代码有误而无法运行时, 系统应能将当前对话历史, 错误信息等反馈给大模型, 让大模型改正错误, 重新生成正确代码.

5. 代码运行: 系统应能够执行生成的 Python 代码而不发生崩溃, 并返回执行结果。

6. 基于运行结果的解释与应答: 系统应使用大模型基于执行结果生成自然语言形式的回答

参考:https://platform.openai.com/docs/assistants/tools/code-interpreter

二、 测试数据和数据分析问题

大模型实习项目测试.csv (在代码中为了防止中文名称可能导致的问题, 重命名为 LLM_test.csv)

测试环节应输入 3 个问题, 且 3 个问题间应存在关联性, 即后续问题需要基于前序问题的历史 (问题, 代码, 执行结果, 大模型解释等) 来解决. 下面是一组问题示例:

1. 分析 Clothing 随时间变化的总销售额趋势.

2. 请对 Bikes 进行同样的分析.

3. 哪些年份 Components 比 Accessories 的总销售额低?

三、 代码运行环境和步骤

运行环境是 vscode+python3.11，点击 jupyter 的运行按钮即可，退出在命令行输入“退出”或者 ESC 键退出即可。

四、 核心代码简述

该系统调用的是 DeepSeek 的 API，它支持与 OpenAI 兼容的 API 格式，具体使用的模型是上周最新发布的 deepseek-chat（V2.5-1210）

1. 在调用 LLM 后生成的代码往往会包含一些非代码的字符，例如中文解释，引号等。因此想要直接使用代码需要进行清理

清理非 code 字符

```
def clean_code(code):
    # 替换中文引号和全角字符为对应的英文字符
    code = code.replace(' “', '"').replace('" ', '"')
    code = code.replace(', ', ',')
    # 清理生成的代码，去掉 python 和
    code = code.strip("python") 去掉开头的 python 和换行符
    code = code.strip("") 去掉结尾的
    return code
```

2. 在完成指定任务时设计了三类 LLM 角色：

第一个是专注于生成解决具体问题的 python 代码的工程师，这里添加了思维链的引导式 prompt 帮助深入思考和分解问题。同时为了方便后续调用代码，还要求他尽可能精简输出，只专注输出代码本身

LLM1

```
1
2  # 使用DeepSeek-chat的API生成Python代码
3  def generate_code_from_query(query,context,content):
4      global conversation_history
5
6      response = client.chat.completions.create(
7          model="deepseek-chat",
8          messages=[
9              {"role": "system", "content": "你是一个非常擅长处理CSV类型文件数
                   据的Python代码工程师.在生成回答时，你只需要输出解决问题的
                   Python代码，不需要解释。请你根据user提出的要求后，Lets␣think␣
                   step␣by␣step，对任务逐步思考后，来输出能够完成对应分析任务和
                   绘制相应图像的python代码.用户会对一个文件进行一系列的分析，所
                   以你需要记住前面提问的细节"},
10             {"role": "user", "content": f"我希望你完成的任务是{query}，其中我
                   的数据保存在{context}这个路径中的.CSV文件里,具体的数据内容是{
                   content}"},*conversation_history,
11         ],
12         stream=False
13     )
14     conversation_history.append({"role": "user", "content": query})
15     # 获取生成的 Python 代码
16     generated_code = response.choices[0].message.content
17     print("生成的代码是: ")
18     print(generated_code)  # 打印生成的代码
19     conversation_history.append({"role": "assistant", "content":
           generated_code})
20     return clean_code(generated_code)
```

第二个是专注于异常处理的 python 检错专家，当角色 1 生成的代码出现报错时，根据报错信息和任务需求进行修改代码

LLM2

```
1
2  def fix_code_with_api(query,code,error_message):
3      global conversation_history
4      response = client.chat.completions.create(
5          model="deepseek-chat",
6          messages=[
7              {"role": "system", "content": "你是一个Python专家，专门帮助修正生
                   成的Python代码。"},
8              *conversation_history,
9              {"role": "user", "content": f"请你检查你之前所写的代码，并修正以
                   下错误: {error_message}后重新生成完整的新的分析代码继续完成给
                   定的任务以下是之前生成的代码: \n{code}。任务是: {query}。请重
                   新生成完整的新的分析代码继续完成给定的任务，在生成回答时，你
```

```
                只需要输出能够完成分析的Python代码，不需要额外计算平均值等操
                作以及文字解释。"},
        ],
        stream=False
    )
    # 获取修正后的代码
    generated_code = response.choices[0].message.content
    #print(clean_code(generated_code))
    conversation_history.append({"role": "assistant", "content":
        generated_code})
    return clean_code(generated_code)
```

第三个是专注于输出自然语言解释的数据分析师，它根据正确执行命令后的 python 输出结果对任务需求进行细致的文字解释

LLM3

```

def generate_explanation(execution_results, query):
    explanation_prompt = f"根据以下数据分析任务的结果，请提供一段自然语言解
        释：任务:␣{query}\n结果:␣{execution_results}"
    response = client.chat.completions.create(
        model="deepseek-chat",
        messages=[
            {"role": "system", "content": "你是一个数据分析师，负责根据Python
                代码执行结果生成自然语言解释。"},
            {"role": "user", "content": explanation_prompt},
        ],
        stream=False
    )
    explanation = response.choices[0].message.content
    return explanation
```

其实最重要的核心功能上述四个函数已经可以满足本次任务要求了。不过我认为在实际产品使用过程中还应该为用户提供更好的体验，为此我做了如下尝试：

提供模糊匹配：如果我们在用户上传文件后可以先检索确定文件中每个属性名的话，那么对于用户在输入时大小写等不规范的输入，我们应该提供模糊匹配的方法。

正如我在视频中演示的一样，在文件中每个属性名其实开头都是大写的，例如 Clothing，但我输入 “对 clothing 总销售额随时间变化

的分析”，同样可以完成对应的任务。

Expexted Sarsa

```
# 检查常见错误
def check_for_input_errors(query):
    common_errors = {
        "clothing": "Clothing",
        "bikes": "Bikes",
        "components": "Components",
        "accessories": "Accessories",
    }

    for error, correct in common_errors.items():
        if error in query.lower():
            query = query.replace(error, correct)

    return query
```

五、 实验结果

1. 能够完成指定的三个任务，即能够生成相应的 python 代码，并且对可能出现问题的代码进行修改。在保证调试成功输出结果后，能够根据具体需求附上直观的图表说明和自然语言的文字解释

2. 同时支持上下文理解，能够对后面较为模糊的指令跟前面历史对话记录中具体的指令联系起来，从而正确实现相应功能

3. 从用户角度出发，为优化体验做了一些尝试和实践。

上述工作将在近期开源在我的 GitHub 主页中

(https://github.com/githublgc)