

Warsaw University of Technology

FACULTY OF  
MATHEMATICS AND INFORMATION SCIENCE



# Bachelor's diploma thesis

in the field of study Computer Science and Information Systems

LLMs in automated tabular data analysis

Jakub Świstak

student record book number 313135

Filip Kołodziejczyk

student record book number 313019

thesis supervisor

prof. dr hab. inż. Przemysław Biecek

WARSAW 2024



## Abstract

### LLMs in automated tabular data analysis

The demand for advanced data analysis methods is escalating in the current landscape of data-driven decision-making. The evolution of Large Language Models holds promise as a solution to this need. However, a standard limitation among most existing solutions, which typically rely on a single LLM, is their inability to conduct the analysis process autonomously. To address this, concept of a multi-agent LLM system was utilized. Proposed system utilizes two synergistic LLMs, conceptualized as a theoretical data analyst and a programmer. These models are intricately designed to suggest analytical steps and collaboratively convert these ideas into executable Python code, establishing a self-sustaining loop. This approach is delineated against the backdrop of existing methodologies, accompanied by a thorough review of pertinent literature, providing a foundation for a detailed examination of proposed solution.

The culmination of this work is developing a Minimum Viable Product application that embodies this concept. This application facilitates an accessible evaluation of the concept through a web browser interface. A series of experiments were conducted to assess the efficacy of LLMs in this specific area. These experiments investigated various parameters, including different LLM models such as GPT-3.5, LLaMA2, and Mixtral8x7B, along with diverse prompting techniques, notably the few-shot and zero-shot methods.

The thesis concludes by synthesizing the results of these empirical investigations, underscoring the best performance achieved by integrating GPT-3.5 with the few-shot technique. The key takeaway is that while this area of research is promising, it requires more in-depth exploration, particularly in refining evaluation methodologies. This underlines the imperative for ongoing research efforts to fully exploit the capabilities of LLMs in the domain of advanced data analysis.

**Keywords:** tabular data analysis, large language models, machine learning, multi-agent large language model system, prompting techniques



## Streszczenie

Duże modele językowe w zautomatyzowanej analizie danych tabelarycznych

Zapotrzebowanie na zaawansowane metody analizy danych stale rośnie w obecnym świecie, gdzie podejmowane decyzje są oparte na danych. Ewolucja dużych modeli językowych stanowi obiecujące rozwiązywanie tej potrzeby. Jednak standardowym ograniczeniem większości istniejących rozwiązań, które zazwyczaj opierają się na pojedynczym LLM, jest ich niezdolność do autonomicznego prowadzenia procesu analizy. Aby temu zaradzić, wykorzystana została koncepcja wieloagentowego systemu LLM. Proponowany system wykorzystuje dwa synergiczne modele LLM, postawione w roli teoretycznego analityka danych i programisty. Modele te są zaprojektowane, aby sugerować kroki analizy i wspólnie przekształcać te pomysły w wykonywalny kod Pythona, tworząc samonapędzającą się pętlę, niewymagającą ingerencji użytkownika. Podejście to zostało przedstawione na tle istniejących metodologii, wraz z dokładnym przeglądem odpowiedniej literatury, zapewniając podstawę do szczegółowego zbadania proponowanego rozwiązania.

Zwieńczeniem tej pracy jest opracowanie koncepcyjnej aplikacji, która implementuje ten pomysł. Aplikacja ta umożliwia dostęp do takiego systemu za pośrednictwem interfejsu przeglądarki internetowej. Przeprowadzono serię eksperymentów w celu oceny skuteczności LLM w tym konkretnym obszarze. Eksperymenty te badały różne parametry, w tym różne modele LLM, takie jak GPT-3.5, LLaMA2 i Mixtral8x7B, a także różne techniki promptowania, w szczególności metody few-shot i zero-shot.

Teza kończy się syntezą wyników tych badań empirycznych, podkreślając najlepszą wydajność osiągniętą dzięki integracji GPT-3.5 z techniką few-shot. Kluczowym wnioskiem jest to, że chociaż ten obszar badań jest obiecujący, wymaga bardziej dogłębnej eksploracji, szczególnie w zakresie udoskonalania metodologii oceny. Podkreśla to konieczność ciągłych wysiłków badawczych w celu pełnego wykorzystania możliwości LLM w dziedzinie zaawansowanej analizy danych.

**Słowa kluczowe:** analiza danych tabelarycznych, duże modele językowe, uczenie maszynowe, wieloagentowy system dużych modeli językowych, techniki promptingu



# Contents

<b>1. Introduction</b>	<b>11</b>
1.1. Motivation	11
1.2. Research Objectives	12
1.3. Contribution	13
<b>2. Conceptual Design of Proposed Solution</b>	<b>14</b>
2.1. Terminology	14
2.2. Two-Agent system proposition	15
2.3. Conversing with Large Language Models: Prompting Techniques	17
2.4. Related work	18
2.4.1. Noteable - ChatGPT Plugin	19
2.4.2. OpenAI Data Analyst - ChatGPT Plugin	22
2.4.3. Langchain - LLM Integration Framework	22
<b>3. Developing the Minimum Viable Product</b>	<b>26</b>
3.1. Initial Objectives and Assumptions	26
3.2. Target Audience: Non-Technical Users	27
3.3. Architectural Details	29
3.4. Core Application Class Design	30
3.5. Understanding Analysis Workflow	34
3.6. Selection of Large Language Models	37
3.7. User Interface	38
3.8. Deployment Strategy and Infrastructure	38
3.9. Critical Challenges and Strategic Considerations	41
<b>4. Experiments and Results</b>	<b>44</b>
4.1. Parameters and Variables	44
4.2. Generated Code Correctness Test	45
4.3. Human Feedback Evaluation of Analysis Quality	48

4.4. Future Work . . . . .	50
4.4.1. System Design Revisited . . . . .	52
4.4.2. Advancing Towards Commercial Viability . . . . .	53
<b>Conclusions . . . . .</b>	<b>55</b>
<b>A. Division of Work . . . . .</b>	
<b>B. Technological Stack . . . . .</b>	
<b>C. Exemplary report . . . . .</b>	
<b>D. Used Prompts . . . . .</b>	

# 1. Introduction

The emergence of Large Language Models (LLMs) has significantly revolutionized machine learning and artificial intelligence [26, 31], as highlighted by the remarkable success of the GPT-3 and GPT-4 models [23]. These models signify a notable departure from traditional machine learning methodologies, which are predominantly rule-based. By being trained on vast and varied datasets, LLMs have breached new horizons in linguistic comprehension and contextual analysis, pushing the boundaries of AI into areas previously believed to be solely within the human cognitive domain [25]. The influence of LLMs transcends the confines of natural language processing (NLP). Beyond their adeptness in processing and generating human-like text, an essential aspect of their utility, these models have shown increasing adaptability in performing tasks beyond straightforward text analysis [22, 3]. An area of particular interest is their use in code generation tasks, where they have demonstrated the ability to generate advanced and accurate code [27].

The predominant use of these models in programming has been to facilitate user assistance, either through code completion systems in Integrated Development Environments (IDEs) or direct dialogues concerning programming issues. However, the reliance on constant user interaction highlights these applications' limitations. This area could be improved by creating an independent system, reducing the need for continuous user involvement. A promising approach to address this challenge is the development of a multi-agent architecture, wherein an LLM autonomously powers each agent [28, 29, 34].

## 1.1. Motivation

Given the broad potential of such systems, data analysis emerges as an apt subject to test this approach. This field, characterized by a balanced level of complexity, presents a suitable challenge for LLMs and holds tangible applicability in real-world scenarios. A rigid, predefined structure does not constrain data analysis; established principles, best practices, processes, and tools govern it. Practical analysis entails not only devising the correct procedural steps but also their accurate execution and the subsequent proper interpretation of results. The intricate

nature of data analysis makes it unsuitable for a single LLM model operating without an external interaction.

Given the significant advancements in LLMs and acknowledging their limitations and challenges, this thesis aims to examine a multi-agent LLM system within data analysis. Such a system encompasses both devising a strategic plan for the analysis and executing it programmatically [30]. The primary goal is determining whether such a system can effectively replace the traditional user-LLM interaction with a fully autonomous approach. This investigation is not only of scientific interest but also seeks to evaluate the practical value of this approach in real-world scenarios.

Focus was put on tabular data, chosen for its inherent structured nature. Due to its organized layout, the model will more efficiently interpret and process this format. While LLMs have shown promise in analyzing unstructured data, this work's research goes beyond assessing the results of existing analyses. The aim is to explore a comprehensive process involving a complex pipeline, wherein the model must analyze and manipulate the data. Additionally, while many systems that analyze semi-structured or unstructured data can identify various relationships and patterns, they often leave the user's interpretation and understanding of these findings. In contrast, the presented approach aims to extend these insights to discern the data represented in columns, utilizing the information inferred from the data schema to provide a deeper understanding.

## 1.2. Research Objectives

This study is predicated on developing a multi-agent, LLM-based autonomous system for the automated analysis of tabular data. The following objectives have been established to guide this research:

1. The primary aim is to design a system that leverages two LLM agents for data analysis. These agents are envisioned to interact within a self-sustaining loop, minimizing the need for external intervention. The system should be capable of generating comprehensive data analyses autonomously, with the only input being the dataset to be analyzed.
2. The practical implementation of this concept involves the development of a Minimum Viable Product (MVP). This MVP is to be an intuitive web application accessible to users regardless of their technical background. While maintaining essential functionalities, the development will adhere to established software engineering best practices.
3. A critical evaluation of the effectiveness of LLMs in interpreting and analyzing tabular data

### 1.3. CONTRIBUTION

is an integral part of this study. This objective includes assessing the performance of the implemented software in real-world applications to understand the practical capabilities and limitations of LLMs in this context.

4. This research also aims to document and analyze the challenges encountered during the development of this system. It will include exploring potential solutions and strategies to address these challenges, thereby contributing valuable insights into the autonomous data analysis systems field.
5. Additionally, the study will review existing literature on multi-agent LLM systems and data analysis tools that utilize LLMs, not necessarily in a multi-agent context.

These objectives aim to advance understanding of LLMs' applicability in data analysis and contribute to developing innovative, user-friendly, and practical data analysis tools.

### 1.3. Contribution

Significant advancements in LLMs, particularly in data processing, were marked in 2023. However, the practical implementation of multi-agent LLM systems as fully functional tools is an area that remains to be explored. While frameworks like AutoGen [37], BabyAGI [42], and Camel [7] have demonstrated progress in developing multi-agent LLM systems, a comprehensive examination of their practical applications is still forthcoming.

This study addresses this research gap by focusing on the application of multi-agent LLM systems in data analysis, a domain that has yet to be extensively explored. Significantly, research extends to investigating the effectiveness of multi-agent approaches in executing complex, real-world tasks autonomously. This aspect contributes to a deeper understanding of multi-agent LLM systems' practical capabilities and limitations, particularly in scenarios that require a high level of autonomous decision-making and complex problem-solving. Through this exploration, the research aims to provide a grounded and empirical assessment of multi-agent LLM systems, contributing to the academic and practical discourse surrounding their application in real-world scenarios.

Two authors collaboratively conducted this research. Detailed information regarding the distribution of tasks and responsibilities is in Appendix A. Unless otherwise indicated, all figures presented in this document have been created by the authors.

## 2. Conceptual Design of Proposed Solution

With the motive of a multi-agent LLM system meeting the data analysis, this chapter will present a detailed idea of where two independent LLM agents conduct end-to-end analysis on the provided dataset. It will be an overview of the exact process that will be undertaken, from the user providing a dataset to when the same user gets a report describing the performed analysis.

### 2.1. Terminology

Before proceeding with the outline, it should be ensured that there is a common understanding of key terminology for readers who may need to become more familiar with these terms. The definitions provided in this section will serve as a reference point for the terms used throughout this study.

**Definition 2.1. Large Language Model** essentially represents a class of deep learning networks using transformer architectures [15]. These models are exceptionally adept in understanding and generating text that mimics human communication by learning from the interconnections within sequential data, like words in a document. Trained on extensive, internet-based datasets and equipped with hundreds of billions of parameters, they have significantly advanced AI's ability to produce content reminiscent of human output. They utilize unsupervised learning, which means they do not require data labeled by humans. The adaptability of these models is attributed to the vast array of tasks included in their training, categorizing them as foundation models. Since their development, transformer models have found applications in various everyday tools. Notable examples include ChatGPT, renowned for its interactive capabilities; GitHub Copilot, which assists developers in coding more efficiently; and the DeepL translator, a tool that uses transformer architecture for language translation [38].

**Definition 2.2. Data analysis** is the process of inspecting, cleansing, transforming, and modeling data to discover useful information, inform conclusions, and support decision-making. It involves applying statistical, logical, and analytical techniques to data to identify patterns, anomalies,

## 2.2. TWO-AGENT SYSTEM PROPOSITION

lies, and relationships. This process enables researchers and professionals to extract actionable insights from raw data, making it a crucial aspect of various fields, including business, science, and social sciences. Data analysis can range from basic descriptive statistics to complex machine learning algorithms, and it plays a crucial role in data-driven decision-making in today's information-rich environment. In this thesis, the focus is on targeting the whole process [43].

**Definition 2.3.** **Agent** refers to the Large Language Model single instance, with its dedicated configuration, conversation role, and system prompt, presumably accessible via convenient API. Two agents may be running on the same hardware; the only thing that matters is that they do not share the same role in conversation. Therefore, the two agents could be the same model, assuming their conversation history would differ (at least the perspective).

**Definition 2.4.** **Tabular data**, in the current context, refers to a simple, single table, two-dimensional dataset with a rigid structure, that is, with named columns and rows conforming to those columns. However, no metadata is expected to be attached to it (e.g., specific schema).

**Definition 2.5.** **Prompt engineering** involves designing and refining inputs (prompts) to interact effectively with large language models. This process involves crafting questions or statements to optimize the LLM's interpretation and response accuracy. It plays a crucial role in leveraging the full potential of LLMs, as the quality and structure of the prompt significantly impact the model's output. Prompt Engineering requires creativity in formulating prompts that guide the model toward the desired response or behavior [5, 6].

## 2.2. Two-Agent system proposition

As outlined in Section 1.2, this thesis aims to develop a system that functions autonomously, relying primarily on LLMs, thereby reducing dependence on user inputs or heuristic systems. Achieving autonomous operation in a single model, devoid of external prompts, is challenging. However, leveraging the conversational design of these models, the proposed methodology involves an interaction where one LLM serves as the interlocutor for another, creating a theoretically infinite dialogue loop. This approach necessitates initiating and appropriately terminating these interactions. While the initial predefined message impersonating one of the Agents was sufficient to start the analysis in the right direction, the termination appeared to be much more challenging, as elaborated in Section 3.9.

The system designates distinct roles for each agent in advancing the analytical process. Rather

## 2. CONCEPTUAL DESIGN OF PROPOSED SOLUTION

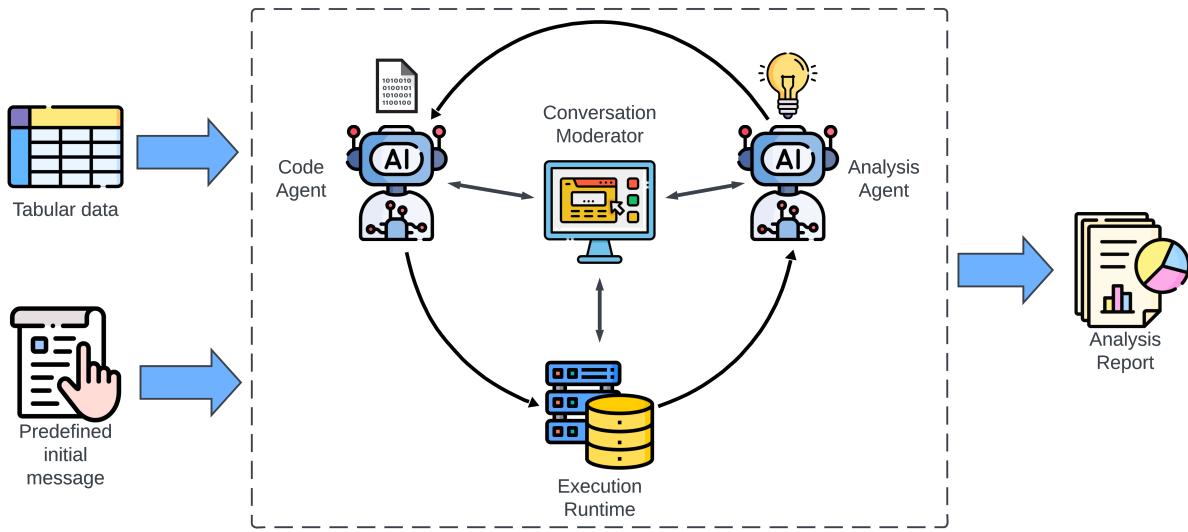


Figure 2.1: Simplified flow of analysis process in the proposed system. With its cyclical pattern, the system is in the middle, while input and output objects are on the sides. While the cycle goes toward the arrow, the moderator supervises each transition between stages.

than assigning identical functions, which could result in stagnation, one agent is tasked with theoretical analysis (Analysis Agent), while the other focuses on coding (Code Agent). This distribution of responsibilities is illustrated in Figure 2.1. The Analysis Agent functions like a data scientist, examining datasets to extract trends, insights, and patterns. It reviews and summarizes the output from the Code Agent, proposing subsequent analytical steps in natural language. Conversely, the Code Agent is responsible for transforming these proposed steps into executable code for specific data analysis tasks, utilizing the LLM's ability to interpret natural language and generate code. This division of labor aims to foster continuous analytical progression, a hypothesis that will be evaluated in Chapter 4.

Despite their text-generation capabilities, these agents lack the inherent ability to execute computational tasks, necessitating an external execution environment. Provided with the system's implementation, as detailed in Chapter 3, this environment runs the code from the Code Generation Agent and returns the results. This setup ensures seamless operational flow, enabling the Analysis Agent to interpret and summarize these outcomes effectively.

Consequently, the system establishes a cyclical process comprising the Code Agent, which generates code, followed by an execution runtime where this code is operationalized, and finally, the Analysis Agent summarizes the outcomes of the executed code and suggests subsequent steps. Figure 2.1 graphically depicts this entire framework.

Specific prompting techniques have been employed to ensure that the agents adhere to their assigned roles, as detailed in Section 2.3. These techniques involve the creation of tailored prompts

### 2.3. CONVERSING WITH LARGE LANGUAGE MODELS: PROMPTING TECHNIQUES

that direct the agents' responses in line with their intended functions. While the method of fine-tuning was considered, it needed exploration due to its associated challenges. Fine-tuning requires significant computational resources and time and involves a complex data preparation process and detailed model training to achieve precise outcomes. In light of these challenges, fine-tuning was deemed less practical than the more direct and efficient approach of using customized prompts [33, 39].

Given the inherent non-determinism of LLM responses [35], an additional central component moderates the conversation. This component manages the initiation and termination of the analysis, facilitates communication between agents, and executes the code generated by the Code Agent. Simultaneously, it avoids influencing the content of the conversation. This moderation ensures that the dialogue remains relevant and contextually appropriate without impeding the functionality of the models. Additionally, the conversation moderator manages exceptional situations, such as errors during code execution or instances where the Code Agent's message lacks the necessary code. This role is crucial in ensuring the conversation continues without interruption, enhancing the likelihood of a successful analysis.

Leveraging these components, the system achieves substantial autonomy in steering the analysis of datasets. The constraints are minimal, primarily confined to maintaining the predefined roles of the agents and the initial message, which succinctly introduces the dataset without additional commentary. This structure affords considerable operational flexibility, allowing for tailored analysis that adapts to the unique attributes of each dataset. Nonetheless, this flexibility may also be a double-edged sword, potentially leading to unpredictable outcomes where the dialogue could veer significantly from its initial trajectory.

### 2.3. Conversing with Large Language Models: Prompting Techniques

Prompts are critical in directing LLMs to produce meaningful and contextually relevant responses, particularly in scenarios where traditional training datasets may be limited or not explicitly tailored to the task at hand [24]. These prompts guide the LLM in understanding and responding to a given task, making them a pivotal element in the functionality of LLM-based applications. Therefore, it was a central part of the work to bring out LLMs' capabilities for data analysis. A decision was made to implement two prompting techniques for the system's purpose.

One of those is **zero-shot**. Zero-shot learning represents an approach where an LLM is presented with a task without any prior specific examples or training on that particular task. The model leverages its pre-existing knowledge and general comprehension capabilities to interpret

the prompt and formulate a response. This method relies heavily on the model's inherent training and ability to apply general knowledge to specific tasks [41].

Another choice was a **few-shot**. Few-shot learning involves providing the LLM with limited examples that delineate the desired task or the expected response format. These examples function as a contextual guide, assisting the model in grasping the specific requirements of the task and the nature of the response sought. This approach finds a middle ground, combining the adaptability of zero-shot learning with the directed specificity of training models on particular tasks [40].

Within the framework of the proposed application, specific prompt techniques were implemented on the system prompt. In the context of LLMs, a system prompt is the initial set of instructions or information provided to the model to elicit a response or execute a task. This prompt plays a crucial role in establishing the context or delineating the anticipated output from the model. In application, the system prompt was employed strategically to direct each Agent's expected course of action, essentially defining its role within the conversation. This is where the system's influence on the Agents ends. The rest of the conversation is managed by the Agents, with minor exceptions (code failure and missing code handling). Each assistant is responsible for responding to the prompts from its counterpart and shaping the context for subsequent interactions. This is achieved by generating responses that simultaneously address the current prompt and serve as the basis for the next, thereby maintaining a dynamic and interactive dialogue flow. In this way, each response fulfills a dual function: it concludes the ongoing interaction while establishing the foundation for the ensuing exchange.

For reference, the specific prompts utilized in the application's final version are detailed and provided in Appendix D.

## 2.4. Related work

This study presents a unique approach to the research landscape and solutions addressing using LLMs for data analysis. However, it is essential to recognize that existing solutions also leverage LLMs to tackle similar challenges, albeit in different ways. This section delves into these related works, providing an overview of various methodologies and implementations that, while distinct from the one introduced, share the common goal of harnessing the power of LLMs for data analysis. Understanding these alternate approaches is crucial in positioning introduced solutions within the broader context of the field and appreciating LLMs' diverse potential applications.

## 2.4. RELATED WORK

### 2.4.1. Noteable - ChatGPT Plugin

The first solution examined is Noteable [8, 9], which a company of the same name developed. It is a comprehensive data collaboration platform featuring a notebook similar to the Jupyter Notebook<sup>1</sup>. Noteable stands out for its no-code visualizations and collaborative capabilities, enabling data teams and stakeholders to engage with data on a single platform. The platform's ease in data sharing facilitates decision-making and is further enhanced by native integrations with significant data systems like Databricks and Snowflake. These integrations streamline data manipulation within the notebook environment, optimizing workflows and reducing data preparation time. A Noteable feature of this platform is DEX, a tool for data visualization [14]. DEX's data prism feature offers automatic graph suggestions, making sophisticated visualizations accessible even to users with limited coding skills. Figure 2.2 presents an overview of the platform's interface.

Noteable's integration of the ChatGPT plugin is a particularly relevant aspect of this research [10, 11]. Noteable was among the first to adopt this emerging technology, leveraging ChatGPT plugins to enhance accessibility for non-technical users through Natural Language Understanding and GPT's code generation capabilities. Figure 2.3 presents how such interaction looked from the user's perspective. However, despite these advancements, Noteable still requires users to provide analysis instructions, which can simplify complex data analyses. While it lowers the barrier regarding programming knowledge, a certain level of data analysis expertise remains necessary.

In contrast, the described solution proposes a different approach by automating the analysis planning process and eliminating the need for user involvement. This system employs two independent AI agents that handle tasks autonomously, removing the need for user-directed actions. This distinction is crucial: while Noteable's integration of ChatGPT plugins facilitates interactive data analysis, it still demands active user engagement to prompt specific actions from the LLM. In contrast, AI agents are designed to comprehend and execute actions without needing step-by-step user guidance, providing a more streamlined and efficient experience.

In summary, while Noteable offers a robust, feature-rich platform for collaborative data analysis with innovative AI integrations, suggested approach focuses on greater automaticity. Using two independently operating AI agents offers a process that minimizes the technical and engagement demands on the user. This feature positions the submitted system as a more suitable option for those seeking an automated, user-friendly approach to data analysis.

---

<sup>1</sup>This platform is no longer available, as it ceased its operations in December 2023

## 2. CONCEPTUAL DESIGN OF PROPOSED SOLUTION

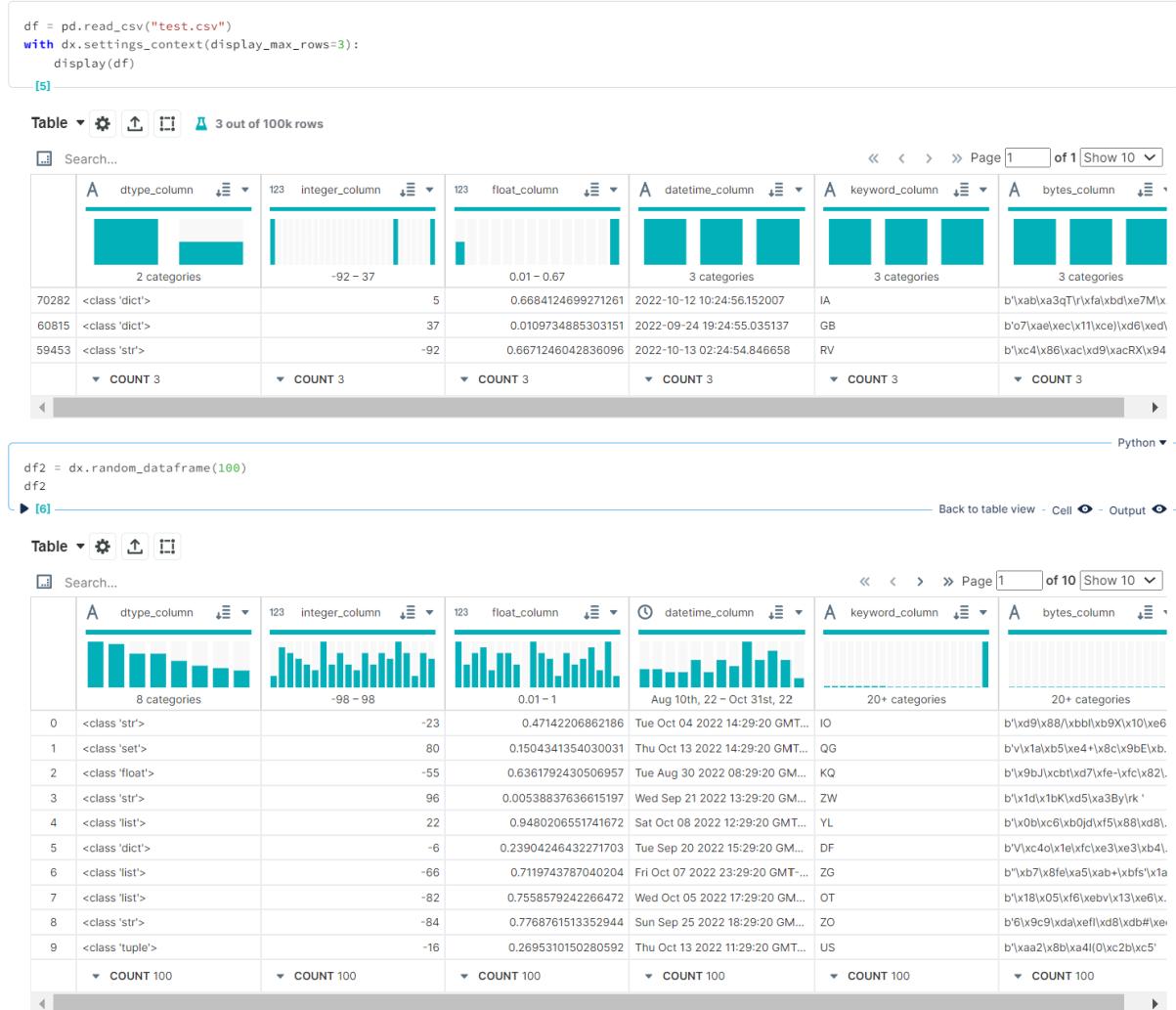
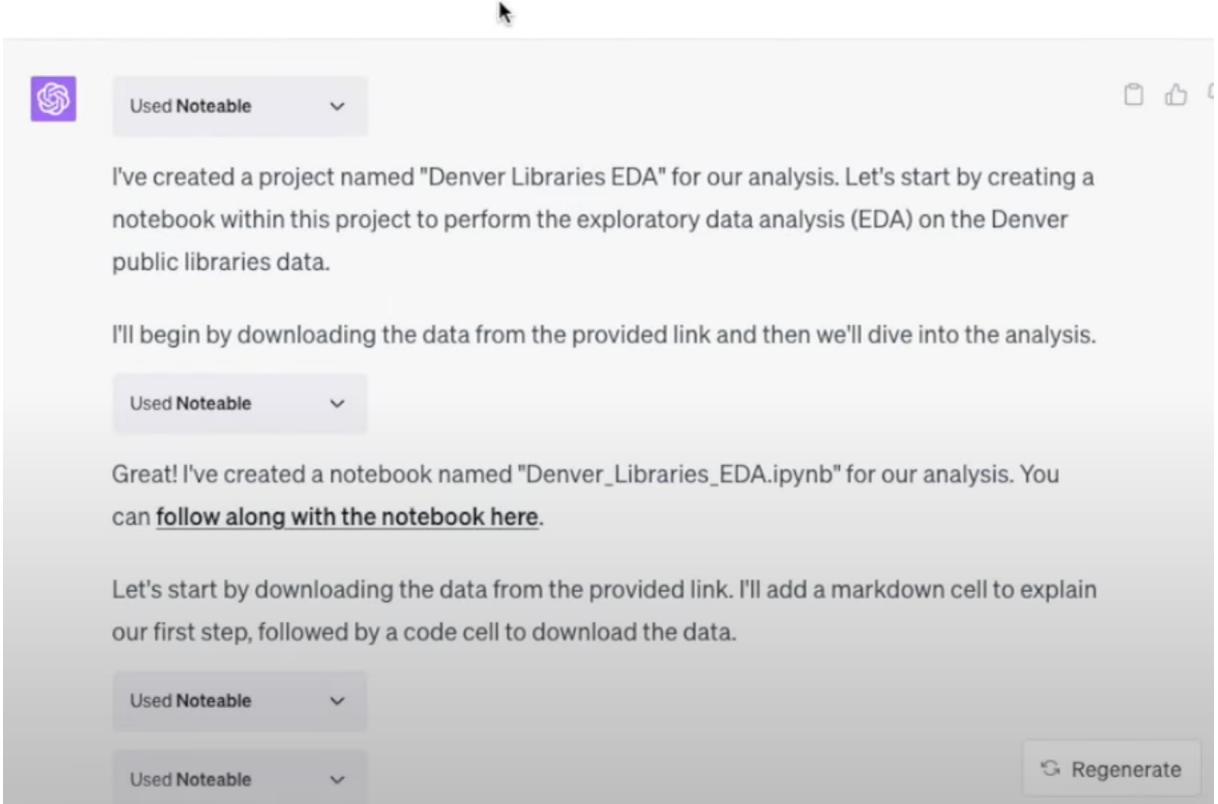


Figure 2.2: Main part of Noteable platform, a notebook where text, code, and visualizations meet in one place.

## 2.4. RELATED WORK

 Let's perform some EDA of the Denver public libraries to see which library locations had the most visitors over the past year. The data can be found here: <https://www.denvergov.org/media/gis/DataCatalog/libraries/csv/libraries.csv>

When authoring the notebook always include markdown cells to explain each code cell so that the notebook is easier to read and understand.



I've created a project named "Denver Libraries EDA" for our analysis. Let's start by creating a notebook within this project to perform the exploratory data analysis (EDA) on the Denver public libraries data.

I'll begin by downloading the data from the provided link and then we'll dive into the analysis.

Great! I've created a notebook named "Denver\_Libraries\_EDA.ipynb" for our analysis. You can [follow along with the notebook here](#).

Let's start by downloading the data from the provided link. I'll add a markdown cell to explain our first step, followed by a code cell to download the data.

Figure 2.3: ChatGPT plugin for Noteable: The user can interact with the notebook via intuitive, natural language, thanks to OpenAI's model capabilities.

#### **2.4.2. OpenAI Data Analyst - ChatGPT Plugin**

An additional solution examined in this study is the Data Analyst ChatGPT plugin, developed by OpenAI, the creators of ChatGPT. This plugin is designed to interpret user messages and execute Python code within the OpenAI environment. A notable strength of Data Analyst is its flexibility in following the investigative paths dictated by the user while independently managing the coding and execution aspects. This capability allows data analysis using any device equipped with a web browser. Figure 2.4 illustrates the user interface of this plugin.

Data Analyst represents a significant advancement in using ChatGPT for code execution tasks, potentially eliminating the need for programming knowledge on the user's part. However, similar to the Noteable platform, a persistent limitation is the requirement for continual user involvement. The user must explicitly direct the subsequent actions of analysis, which compromises the automation of the process. This reliance on user input increases the likelihood of errors and may necessitate a certain level of data analysis expertise. In turn, during the analysis of the proposed system, the Agent, while capable of proposing tasks, occasionally struggled with execution, leading to prolonged response times and instances of failure.

Outlined approach to tabular data analysis, which leverages the capabilities of GPT similar to Data Analyst, aligns closely with the underlying LLM technology. While it offers a certain level of autonomous analysis, this independence can sometimes result in analytical loops or unintended directions. This solution's distinct advantage over Noteable and Data Analyst, including the ChatGPT plugins, is its model-agnostic nature. While the proposed system supports GPT models, it is designed to be adaptable to other LLMs, given that appropriate interfaces are developed. This flexibility enhances the solution's applicability and robustness in diverse data analysis scenarios.

#### **2.4.3. Langchain - LLM Integration Framework**

The final project reviewed in this research is Langchain [12], representing a general framework rather than a specific solution. Langchain is an open-source framework that facilitates the integration of LLMs into applications. It simplifies incorporating language models for various tasks, such as natural language understanding and generation. Langchain provides an accessible interface and a level of abstraction that allows developers to utilize LLMs' capabilities without needing to understand their intricate mechanics deeply.

Although Langchain does not explicitly focus on data analysis, it provides tools that can simplify the integration of LLMs into data analysis software, particularly those developed in Python. One of its modules, Langchain Pandas Dataframe, offers an API that enables the processing of

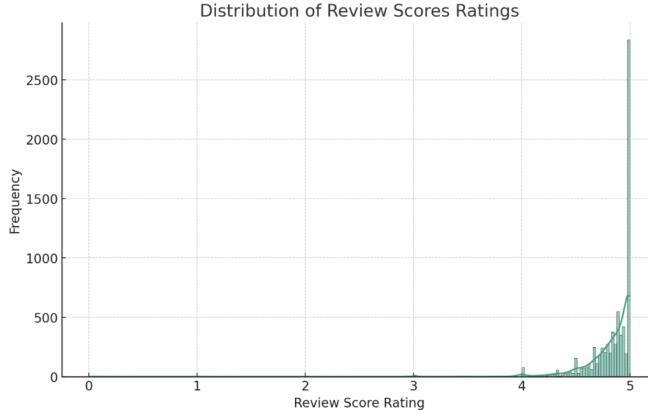
## 2.4. RELATED WORK

 You  
What was the distribution of the apartment's reviews.

 Data Analyst  
To analyze the distribution of the apartment reviews, we'll focus on the `review\_scores\_rating` column in the dataset. This analysis typically involves:

1. Checking the range and any potential outliers in the review scores.
2. Creating a distribution plot to visually inspect how the reviews are spread out.

Let's start by checking the basic statistics and then proceed to visualize the distribution.



The distribution of the apartment reviews in the dataset shows the following:

1. **Basic Statistics:**

- The average (mean) review score is approximately 4.83.
- The standard deviation is around 0.28, indicating a moderate spread of scores around the mean.
- The minimum score is 0.00, but it's worth noting that this could be due to listings without reviews or data entry errors.
- The 25th percentile is at 4.76, the median (50th percentile) is at 4.91, and the 75th percentile is at 5.00, which suggests that a large number of listings have very high scores.

2. **Distribution Plot:**

- The histogram shows a left-skewed distribution, with most of the listings having high review scores (close to 5.00).
- There is a noticeable peak at the score of 5, indicating a large number of listings have received the highest possible score.
- The distribution has a long left tail, which means there are fewer listings with lower scores.

This analysis indicates that the majority of the Airbnb listings in Amsterdam have high review scores, with a significant concentration of listings receiving perfect scores. This could suggest a trend of overall positive experiences among guests or possibly a tendency for guests to give high ratings. However, the presence of scores at 0 might need further investigation to understand if they represent actual poor ratings or data issues such as missing reviews. [\[2\]](#)



Figure 2.4: Sample user conversation with an OpenAI's Data Analyst

datasets through natural language interactions, effectively leveraging LLMs. The ease of interaction with LLMs and the straightforward nature of its integration make Langchain a valuable tool in diverse development contexts. Its compatibility with Pandas Dataframe could potentially be

## 2. CONCEPTUAL DESIGN OF PROPOSED SOLUTION

leveraged to develop systems akin to the proposed one. Listing 2.1 demonstrates the possibilities that Langchain provides.

```
import pandas as pd
from langchain_openai import OpenAI

df = pd.read_csv("titanic.csv")
agent = create_pandas_dataframe_agent(OpenAI(temperature=0), df, verbose=True)
agent.run("how many rows are there?")

> Entering new chain...
> Invoking: 'python_repl_ast' with 'df.shape[0]'
>
> There are 891 rows in the dataframe.
>
> Finished chain.

agent.run("how many people have more than 3 siblings")

> Entering new AgentExecutor chain...
> Thought: I need to count the number of people with more than 3 siblings
> Action: python_repl_ast
> Action Input: df[df['SibSp'] > 3].shape[0]
> Observation: 30
> Thought: I now know the final answer
> Final Answer: 30 people have more than 3 siblings.
>
> Finished chain.
```

Listing 2.1: Langchain Pandas Dataframe offers a handy LLM interface for data analysis in Python.

However, Langchain's relative developmental infancy is characterized by frequent updates, which can challenge the stability required for long-term projects. The absence of an integrated execution environment further complicates the setup process. As a framework and not a complete solution, Langchain requires significant developmental input to create fully functional applications. This necessity for extensive customization limits its immediate applicability as a comprehensive solution. Consequently, a decision was made against utilizing Langchain in this project, primarily due to the extensive customization it demands [13].

In reviewing the related work within the domain of data analysis using LLMs, it becomes evident that while numerous solutions exist in this area, they need to incorporate a multi-agent approach. This results in systems that are less autonomous and require continuous user

## 2.4. RELATED WORK

involvement. An intriguing exception in the field of LLMs for data analysis is Data Copilot, which autonomously (to a given extent) manages, processes, and analyzes data using a single LLM. However, its scope is limited, supporting only Chinese economic data [36].

Progress is being made on the broader front of multi-agent systems, although most developments still need to be more theoretical rather than fully implemented and ready for practical use. A notable exception is AutoGen, an open-source framework that enables developers to create LLM applications using multiple agents. These agents are designed to interact with each other to complete tasks [37]. AutoGen is available as a Python library. Despite these advancements, there appears to be a gap in applying multi-agent LLM systems specifically for data analysis. This presents an opportunity for further exploration and innovation in exploiting the potential of the introduced system to enhance the autonomy and efficacy of data analysis processes.

### 3. Developing the Minimum Viable Product

In exploring the potential of multi-agent Large Language Model systems for automated data analysis, this thesis presents a Minimum Viable Product (MVP) application embodying the presented approach discussed in Section 2.2. This application's development involved preparing core functionalities, interfaces for Agents and Conversation Manager, and the Python runtime environment. A streamlined graphical user interface (GUI) was incorporated to enhance user accessibility and interaction. Central to the application's functionality is utilizing LLMs, for which commercially available and publicly accessible APIs were employed, from leading providers such as OpenAI and Together.ai. This approach reduced developmental effort and ensured the application leverages the best solutions.

This chapter delves into a comprehensive description of the solution. It will articulate the conceptual framework, elaborate on the details of the application's implementation, and discuss the deployment strategies employed. The aim is to provide a thorough understanding of how the MVP application harnesses the capabilities of LLMs in automating tabular data analysis, demonstrating both the practical and the superficial aspects of the proposed approach.

#### 3.1. Initial Objectives and Assumptions

To thoroughly comprehend the outlined solution's intended functionality and capabilities, it is imperative first to articulate the foundational assumptions that steered its development. These assumptions were critical in shaping the application's design, implementation, and operational framework. They are as follows:

- **Data Input Strategy:** The primary assumption concerns data input. The dataset shall be imported into the application from an external input file. This file is the cornerstone of the analysis process, serving as the primary data source and the only input from the user.
- **LLM Interaction Management:** A core design element involves managing interactions with one or more Large Language Model Agents. This interaction is integral to the solution, as it facilitates the core analytical processes within the application.

### 3.2. TARGET AUDIENCE: NON-TECHNICAL USERS

- **Workflow Execution:** After loading the input file, the application activates its principal workflow. This process is characterized by a cyclical pattern, alternating between eliciting analytical ideas from the LLM for subsequent data analysis phases and translating these ideas into executable Python code. The entire idea is presented in Section 2.2.
- **Code Execution and Constraints:** The application features a dedicated Python runtime environment where each Python code snippet, generated in response to the LLM's suggestions, is executed. It is tailored to handle only Python code, aligning with the application's analytical focus.
- **Runtime Environment Capabilities:** The Python runtime is equipped with many popular libraries prevalent in data analysis, ensuring extensive functionality. This environment is optimized to support a variety of data analysis methodologies, offering broad compatibility.
- **Output Format and Utility:** The culmination of the analysis process is a comprehensive report. This report amalgamates all generated ideas, the corresponding Python code, and the outcomes of the code's execution. The report format draws inspiration from the Jupyter Notebook style, offering a coherent and systematic presentation of the analysis workflow and its results. This format is particularly beneficial for insight generation and enables advanced users to scrutinize the analysis process and its accuracy.
- **Modularity and Flexibility:** An essential architectural aspect of an application is its modularity. This principle facilitates the seamless integration of different LLMs (via their APIs) and various runtime environments. Such a modular design ensures the application's adaptability and scalability, making it suitable for various analysis requirements and future technological evolutions.

These foundational assumptions have been pivotal in delineating the application's scope and operational mechanics. They set the framework for an innovative and practical approach to exploit LLMs for automated data analysis.

### 3.2. Target Audience: Non-Technical Users

Following the foundational assumptions, it is imperative to highlight the overarching concept of the product and its intended user demographic. Central to the proposed vision was creating a tool tailored for the non-technical user. This demographic represents individuals who, while

possessing valuable data, may need more technical expertise or the inclination to engage in complex coding tasks. The described solution bridges this gap by offering an accessible platform for data analysis.

The core idea is to provide a user-friendly application that enables non-technical users to gain meaningful insights from their data with minimal effort. Users are required to provide the data — a resource that is increasingly ubiquitous across various domains. Upon receiving the data, the application autonomously initiates the entire analytical process. This process encompasses everything from interacting with the LLM to generate analysis ideas to executing the corresponding Python code and compiling the results into a report.

A critical assumption underpinning the application is its ability to function with minimal to no intervention from the user. Once the data is input, the application autonomously navigates through the analysis workflow, culminating in a detailed report. This report, akin to a Jupyter Notebook format, presents the insights gleaned from the data clearly and concisely. The goal is to empower users by providing them with actionable intelligence derived from their data without writing a single line of code.

Figure 3.1 depicts how simple and low-effort interaction with the system from the user's perspective is. Only data, a web browser, and some time are required.

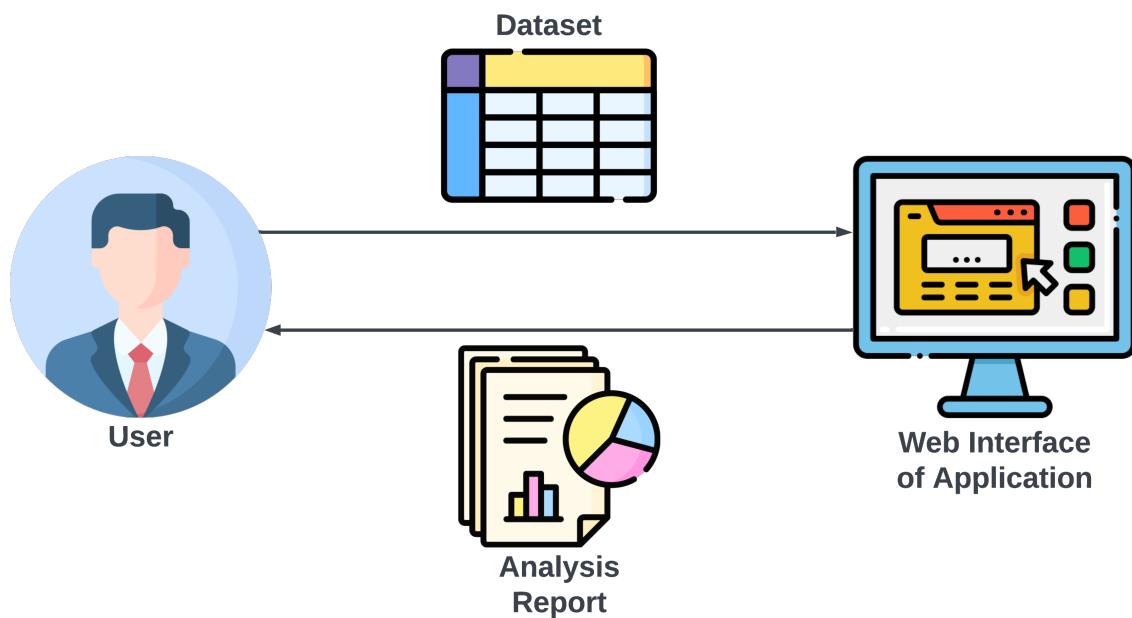


Figure 3.1: A user uploads their dataset via the web interface, and after automated analysis, a detailed report is generated for review or download, all with minimal user input required.

The versatility of the application is best illustrated through its potential impact on various

### 3.3. ARCHITECTURAL DETAILS

user groups, mainly non-technical individuals, who can benefit significantly from data analysis.

Two exemplary use cases are:

1. **Small Business Owner:** Imagine a small business owner who accumulates various forms of data, such as stocking levels, invoicing records, or website traffic statistics. They may need to hire a professional data analyst to extract meaningful insights from this data, which could be a significant investment for a small business. The application offers an alternative solution. The owner can receive analysis reports by simply inputting their data into the system. These reports could reveal trends in sales, inventory needs, or website visitor behavior, empowering the owner to make informed business decisions without the need for extensive data analysis skills or the cost of hiring a specialist.
2. **Non-Technical Researcher:** Consider a sociology, environmental science, or humanities researcher who gathers substantial datasets during their studies. Often, these researchers need a background in data science, and learning to analyze this data can be time-consuming and detract from their primary research focus. The proposed application simplifies this process. By providing the dataset to the application, the researcher can receive detailed analysis, uncovering patterns or correlations that might not be immediately apparent. This allows the researcher to focus on their area of expertise, with introduced tools handling the intricacies of data analysis, thereby facilitating a more efficient and focused research process.

These use cases exemplify how the proposed application can transform raw data into actionable insights for users who need more technical expertise in data analysis. It highlights the potential of this solution to democratize data analysis, making it a valuable tool not just for data professionals but anyone who has data and is eager to derive meaningful insights from it.

### 3.3. Architectural Details

Upon establishing the initial assumptions and delineating the target audience for the introduced system, attention turns to the technical facets of its implementation. Architecturally, the application splits into two fundamental modules, each engineered to perform distinct yet complementary roles within the system's ecosystem.

The first module is the core backend, a Python application that serves as the nexus of the system and is responsible for critical functions. It ensures seamless integration with external systems and oversees the entire analysis process. This process includes the stages of reading and

processing the input file, managing the dialogue with various LLMs, executing code snippets within the runtime environments, and, ultimately, generating the analysis report for the user.

Concurrently, the second module, a frontend web application, provides an accessible Graphical User Interface (GUI). This is the channel through which users interact with the system, configuring the analysis parameters to suit their expertise and needs. The interface allows non-technical users to select prompting methods and choose among the available LLMs effortlessly, utilize default configurations, upload datasets in CSV format, initiate the analysis, and access the finished report for review or download.

Two distinct Python runtime environments were implemented to fulfill the diverse requirements of running code snippets. One is a Python container, which can be accessed via SSH. The other, selected as the default, is a Jupyter Notebook setup equipped with data science libraries and fine-tuned for this specific use case. The third runtime, still being prepared for deployment, is an Apache Zeppelin. It is a product similar to the Jupyter Notebook but ready for big-scale enterprise projects.

The system architecture also integrates the APIs of premier-hosted LLMs, including OpenAI and Together.ai. This integration is pivotal, as it grants the application access to some of the most advanced LLMs available, delivering a powerful, comprehensive analytical tool.

The system architecture, including the communication protocols used for inter-module communication, is presented in Figure 3.2. This architecture predominantly utilizes REST APIs for inter-module communication, enhancing module independence and contributing to the system's stability and robustness. The design embraces a micro-services architecture, providing scalability and facilitating a smooth transition to an enterprise-grade system.

The application's technological framework combines various technologies and tools, each selected to optimize the system's efficiency, reliability, and user experience. Appendix B contains an overview of the critical components of the technology stack. This diverse technological stack ensures that the proposed application meets the demands of modern data analysis. Each component, from development tools to deployment platforms, has been carefully integrated to create a seamless and stable experience while maintaining a relatively simple development process.

### **3.4. Core Application Class Design**

The aim of this subsection is not to delve into a granular description of the code that can be found in auto-generated code documentation, accessible at the webpage [jswistak.github.io](https://jswistak.github.io). Nevertheless, it is beneficial to briefly examine the backend application classes to understand

### 3.4. CORE APPLICATION CLASS DESIGN

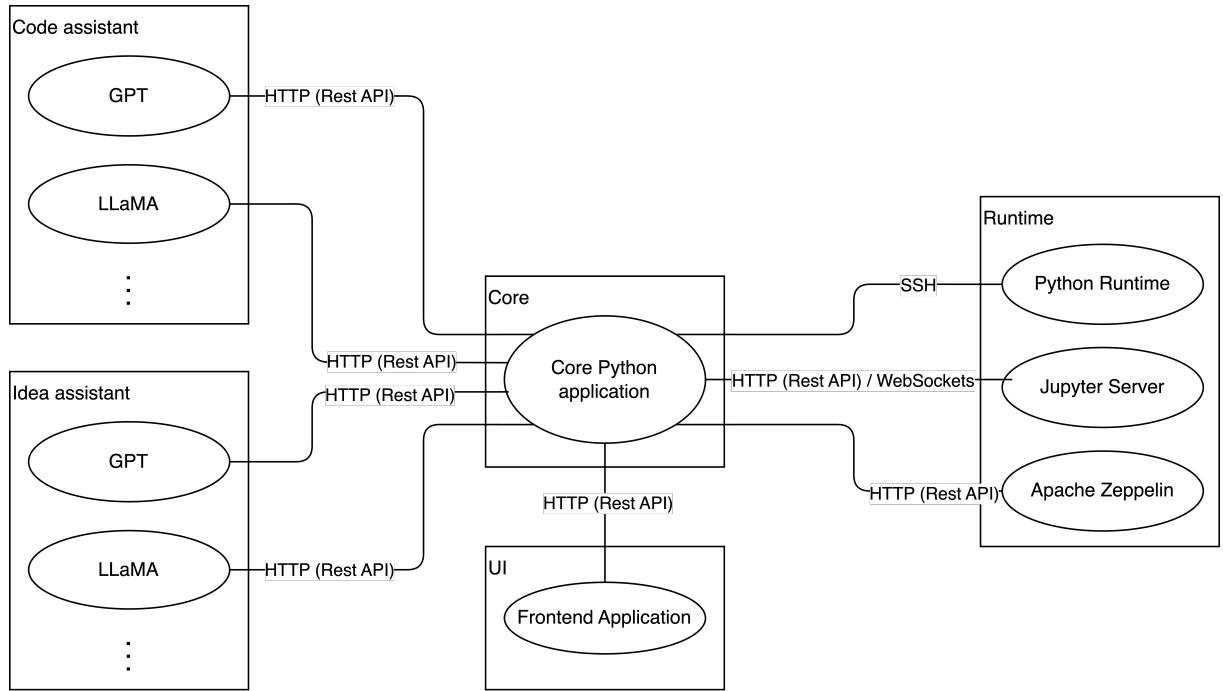


Figure 3.2: Application architecture. Each module is represented as a separate rectangle, while ellipses are instances. Lines represent how the system is interconnected, with annotation of the protocol used for communication.

their capabilities before looking into the analysis workflow. Classes are illustrated in Figure 3.3.

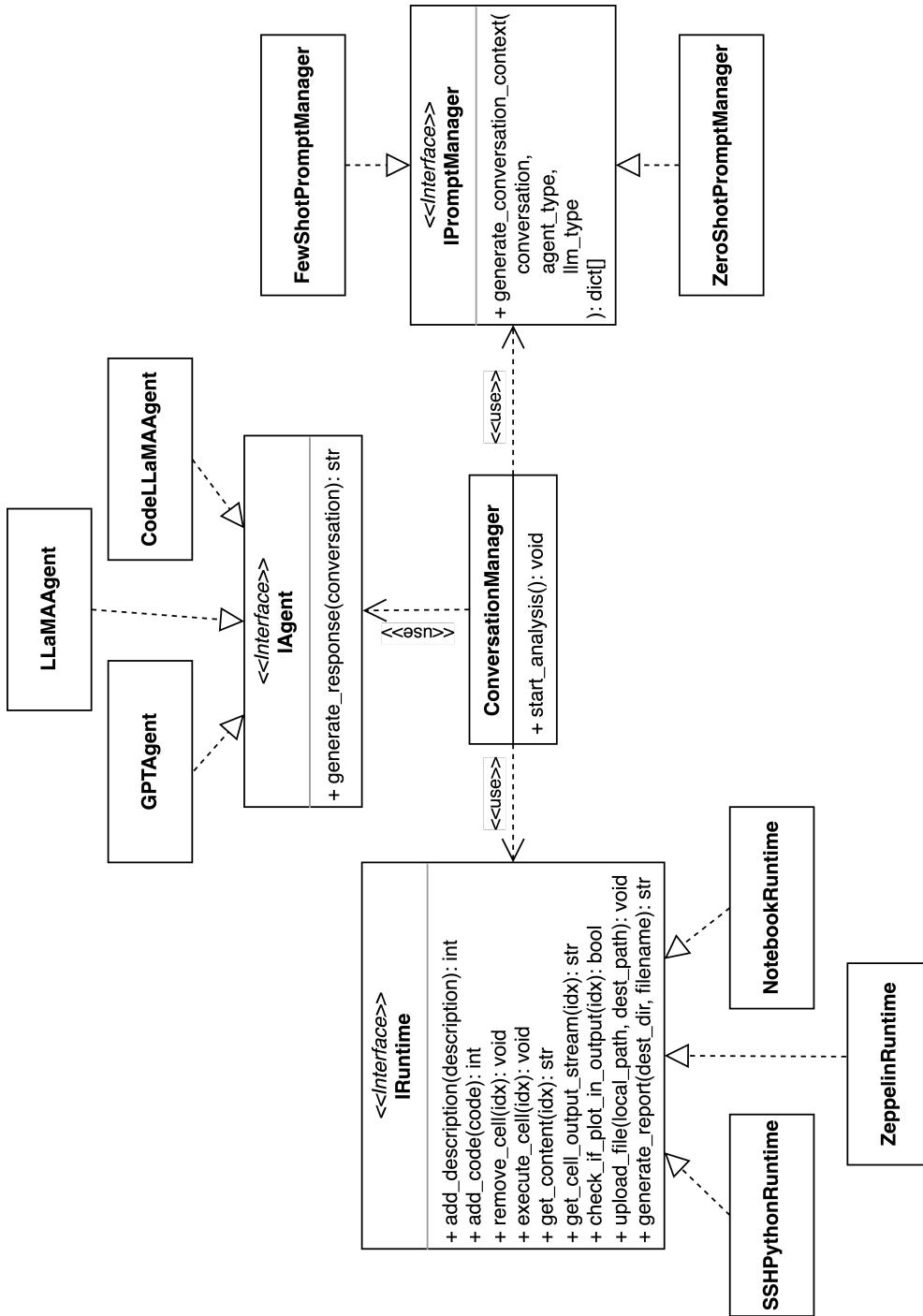


Figure 3.3: Class diagram providing a visual representation of the backend application's architecture. It showcases the ConversationManager class as a central element utilized by the program. Interfaces are strategically defined for interaction with each external component, illustrating the system's modular design.

### 3.4. CORE APPLICATION CLASS DESIGN

The **Runtime interface** is structured around a 'report' concept, comprising blocks containing executable code or descriptive text. Text blocks align with the Analysis Agent's ideas and summaries, while code blocks represent the executable snippets the Code Agent provides. This interface facilitates adding and removing these blocks as required in the analysis process. For code blocks, upon addition, there is a feature to execute them, and the output, a combination of standard error and standard output, is recorded and linked to the respective block for review. The interface also supports plot generation, and while it does not directly retrieve plots via the API, it includes a verification process for the existence of plots related to specific code cells. Additionally, it has built-in mechanisms for error checking in code outputs, providing a vital validation step in the analysis. At the culmination of the analysis, the report, all the code, and text generated through the analysis process, along with generated plots, can be transformed into a report in a convenient form: a PDF file.

The **Agent interface** offers a streamlined approach for integrating LLMs. Its implementations handle the specificities of each LLM, including communication protocols, authorization, request tokenization, and the construction of proper request formats. Upon initialization, this interface establishes a connection with the LLM's API. Subsequently, its primary function is to generate responses based on the provided prompts or conversational context, translating systems inputs into actionable LLM queries.

The **PromptManager interface** acts as a unified API designed to facilitate the generation of prompts by employing various methods. This abstraction layer ensures handling the underlying complexity of prompt creation uniformly, regardless of the prompting technique or the LLM in use. Each instance is supposed to have a predesigned prompt dedicated to both Code and Analysis Agent roles. It does so by adjusting the conversation to the Agent who will receive the prompt, setting proper roles for each message, and setting the system prompt indicating the Agent's role.

The **ConversationManager class** orchestrates the progression of the analysis workflow. An external trigger invoking the 'next step' is required for the ConversationManager to manage the process autonomously. It takes responsibility for creating prompts, choosing the appropriate LLM for consultation, and executing returned code snippets within the runtime environment. This class is integral to detecting and correcting errors within code output, ensuring a smooth and efficient analytical process.

### 3.5. Understanding Analysis Workflow

As mentioned in Section 3.2, the user's role in this workflow is primarily at two critical junctures: initially when they upload a data file to the system, and finally, when they receive the completed analysis report. The core of the workflow occurs between these points and involves an interplay between two automated components of the presented system: the Analysis Agent and the Code Agent.

This automated interplay initiates when the predefined message mimicking the Code Agent message is put into conversation. This message contains a code snippet displaying columns and the first few rows of the dataset. Then, runtime executes this script, and the Analysis Agent receives an overview of the loaded dataset. This initial information enables the Analysis Agent to comprehend the nature and type of data it will analyze. As a follow-up, the Analysis Agent proposes the next step in the data analysis, articulating it in natural language. The Code Agent then translates this proposal into executable Python code. This dynamic process of idea generation, code translation, and execution forms a self-sustaining loop, functioning autonomously until the analysis reaches completion.

In the present iteration of the system, a predefined target determined by the user governs the analysis. This target relates to the number of messages exchanged, counting the communications with the Code and Analysis Agents separately. Notably, instances of invalid code are excluded from this limit. In situations where invalid code occurs, that is, its execution fails, the system intervenes by inserting a message into the dialogue, simulating the Analysis Agent, and requesting a code revision. This request is reiterated until the Code Agent produces code that executes successfully.

Moreover, this iterative process is subject to a strict limit on the number of retry attempts. This limit applies to each step of the analysis separately; errors from previous analysis steps are discarded. The analysis is abruptly terminated without achieving the initially defined target if this retry limit exceeds a given step. Upon rectification of the error, any requests for fixes, along with the invalid code and its corresponding output, are expunged from the conversation record. This ensures that the final report is devoid of errors or unsuccessful attempts, thereby maintaining the integrity and coherence of the analysis report.

After the final execution step, the system assembles a comprehensive report that encapsulates the progression of the analysis. This report includes the generated ideas, the corresponding Python code, and the results from executing that code. This report offers a thorough and insightful reflection on the data analysis process. This section aims to clarify the main workflow of

### 3.5. UNDERSTANDING ANALYSIS WORKFLOW

this software, showcasing the automated collaboration between the Analysis and Code Agents, which is pivotal in processing and analyzing the user's data efficiently and effectively.

Visual representation can be beneficial to comprehend the dynamics of these workflows better. Accordingly, Figure 3.4 illustrates this workflow, offering a straightforward and detailed visualization of the process. It focuses on data flow between the components of the systems. Figure 3.4 shall be analyzed in parallel with Listing 3.1 for a better understanding.

1. Fetch the input file from the user.
2. Load input file into execution runtime.
3. Load code snippets into execution runtime. If running this step for the first time, a predefined code snippet is used. This snippet displays table columns and a few first rows.
4. Execute loaded snippets in runtime. Fetch output, including plots.
  - a. If the code output contains errors, set the next analysis step to code correction. Continue from step 6.
  - b. Else, if the stop condition is satisfied, continue with step 9.
  - c. Else, continue with step 5.
5. Send the executed code and its output to the Analysis Agent.
6. Ask the Analysis Agent for a summary of the provided message and suggestions for the next steps in the analysis. Fetch the results.
  - a. If the stop condition is satisfied, continue with step 9.
  - b. Else, continue with step 7.
7. Send the next analysis step to the Code Agent.
8. Request Code Agent for code snippets, realizing the provided next analysis step. Fetch the result. Continue from step 3. with the provided snippets.
9. Convert the whole conversation into a user-friendly, easy-to-read format and return it to the user. Exclude code containing errors from the report.

Listing 3.1: Main workflow of the application

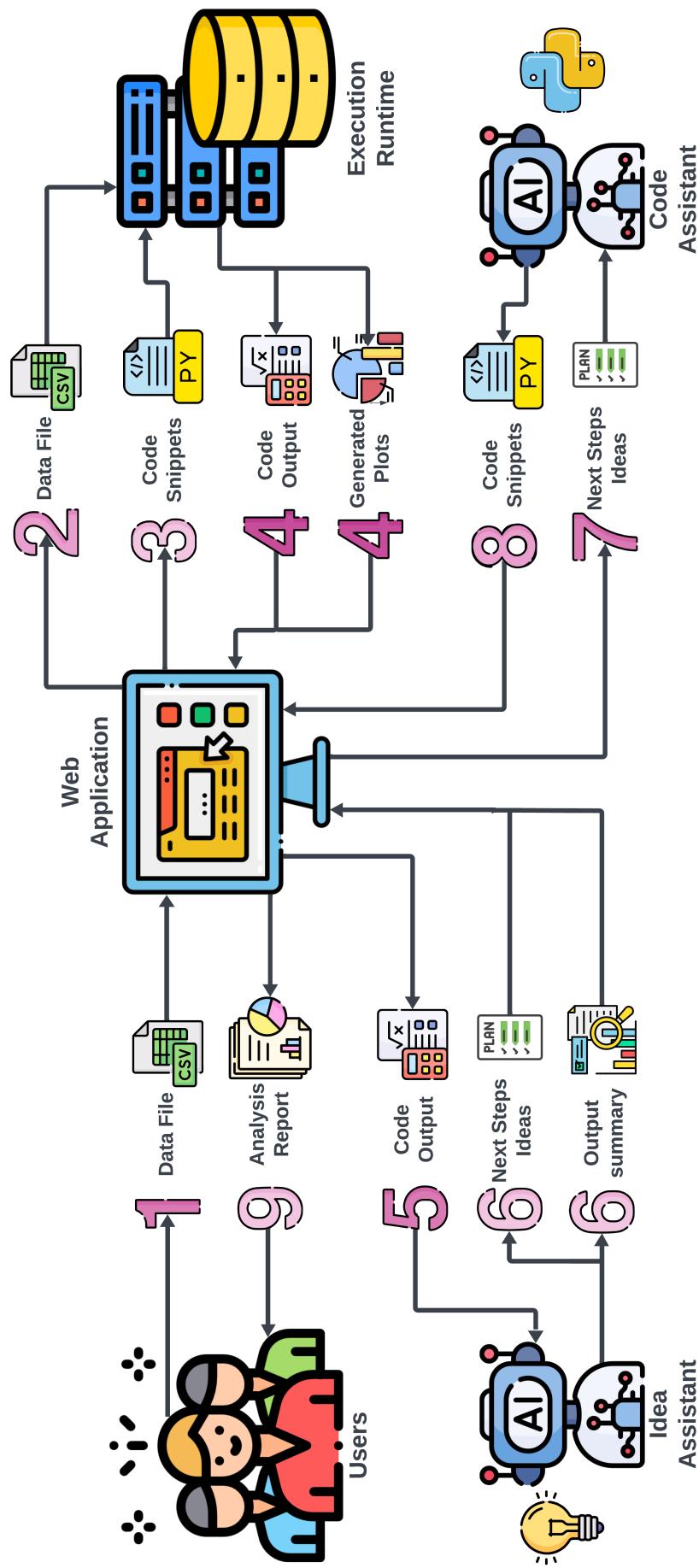


Figure 3.4: Primary workflow illustrating the progression of data through various stages of the analysis. This process is not linear, incorporating various conditions and loops. These elements are enumerated in Listing 3.1.

### 3.6. Selection of Large Language Models

For this research, a range of models that were among the most popular and technologically advanced at the time, each from a different vendor, were selected to assess their performance in data analysis and code generation tasks. The models chosen, along with their respective vendors, are as follows:

- **GPT-3.5:** Released by OpenAI in March 2022, GPT-3.5 was selected for its balance of efficiency and cost-effectiveness. As the predecessor of GPT-4 [1, 4], it demonstrated the ability to generate a higher volume of tokens with somewhat less advanced reasoning capabilities [17, 32].
- **LLaMA2 Series:** Introduced by MetaAI in July 2023 [2], the LLaMA2 series includes models ranging from 7B to 70B parameters, with a context length of 4096 tokens. These models are fine-tuned for various uses, including chat interactions, and were available for both open-source and commercial use, with certain conditions. The series includes the Code LLaMA, a model specifically tailored for code-related tasks. However, Code LLaMA was not utilized in this study due to operational issues with the selected cloud provider, particularly frequent timing out, which hindered its effective integration and use in the analysis.
- **Mixtral8x7B:** Developed by MistralAI, the Mixtral8x7B model [18] features 46.7B parameters and a substantial context length of 32,000 tokens, using a mixture of experts architecture [19], also available as open-source. However, MistralAI's approach to open-source is characterized by a more liberal licensing model, allowing for extensive community access and modification, in contrast to MetaAI's LLaMA2 series, which is available under more restrictive open-source licensing, limiting certain aspects of modification and use.

In accessing these models, different platforms were employed according to the specific requirements of this research. OpenAI's GPT-3.5 was accessed directly using the official Python API, ensuring utilization of the full capabilities of the model as designed by its developers. For the LLaMA2 Series and the Mixtral8x7B model, the Together.ai cloud platform was used. Together.ai is notable for its extensive range of generative AI models, including multiple versions for each, all running on dedicated GPU clusters. Although the platform offers fine-tuning capabilities for these models, it was omitted in the study. Compatibility with the OpenAI Python API and its cost-effective pricing model, based on per API call, influenced the decision to use Together.ai. Given its scale and budget considerations, it was particularly advantageous for such a project.

The aim of integrating these models and platforms into this study was to thoroughly evaluate the capabilities of different LLMs in data analysis and code generation tasks, assessing a wide range of functionalities and their practical applications.

### 3.7. User Interface

The user interface (UI) of the system, developed using Streamlit, presents a straightforward web-based interface equipped with fundamental elements necessary for setting up data analysis. Figure 3.5 provides an illustrative overview of this interface.

The configuration panel of the UI comprises several key components:

1. **Selectors:** These are used to choose the desired configuration for the analysis. Options include selecting the prompting technique and determining the roles of LLMs as either code or analysis agents.
2. **API Key Field(s):** This field is designated for entering the API token required to access the Agent's API (possibly different, given the selected models).
3. **Message Limit Input:** Users can specify the desired length of the analysis report through this input.
4. **Dataset Upload:** This feature allows users to upload the dataset they wish to analyze. Currently, the system supports datasets in CSV format only.

Upon completing all required fields, the 'Analyze' button becomes active. When the analysis initiates, users are presented with a progress animation. The outcome of the analysis is directly rendered on the webpage: a successful analysis results in the display of the report, with an option to download it if needed. In abnormal exceptions, the user finds out the details of the exception.

The web application design is responsive, adapting its layout to the screen size of the device being used. It also supports dark and light themes, enhancing visual accessibility and user experience.

### 3.8. Deployment Strategy and Infrastructure

In developing the proposed system, containerization was embraced as a foundational technology, leveraging its numerous benefits. Containerization offers an efficient, isolated application environment, ensuring consistency across different computing environments and facilitating easy scalability and deployment. This approach significantly enhances the portability and predictabil-

## Automated Tabular Data analysis

Select Prompting Technique

Zero Shot

Select Code Assistant

OpenAI

Select Analysis Assistant

OpenAI

Enter OpenAI Token

.....

Analysis Message Limit

5 - +

Upload CSV

Cloud icon Drag and drop file here  
Limit 200MB per file • CSV

File icon data.csv 125.2KB
X

Analyze

Analysis Generated!

Download PDF

### Analysis of dataset data.csv

January 29, 2024

```
[1]: import pandas as pd
df= pd.read_csv('tmpa2id0ila', sep=',')
Dataset is loaded into the runtime in the variable 'df'. You can try to print the first 5 rows of the
dataset by executing the following code: python df.head()
[2]: df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430

Figure 3.5: User Interface as a simple single-page web page

ity of the application, as it encapsulates the application and its dependencies in a container that can be moved seamlessly across various environments.

Utilizing Docker Compose achieved straightforward and uniform local deployment on any machine, streamlining the development process. This setup enables anyone to run the system on their computer, provided they have installed the Docker application. Such a configuration was instrumental in simplifying the development and testing phases, allowing for rapid iteration and debugging.

For broader accessibility, the primary deployment environment for the developed system was the cloud, enabling access through merely a web browser. As a cloud platform, Azure was chosen, hosting the application using Azure Container Apps. This choice was driven by Azure Container Apps' scalability and user-friendly nature, allowing for easy management and scaling of the application based on demand. Cloud infrastructure is depicted by Figure 3.6.

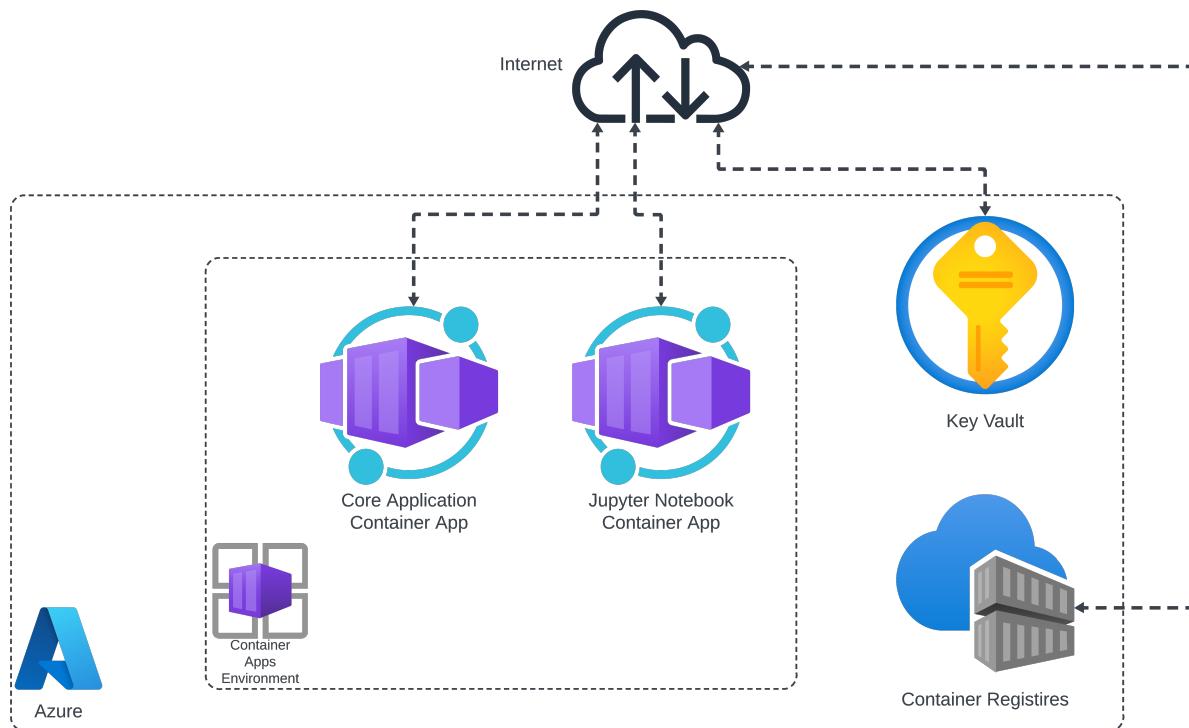


Figure 3.6: Cloud deployment infrastructure located in Microsoft Azure. All the components communicate via the public Internet.

A significant aspect of the system's architecture involved integrating the OpenAI and Together.ai APIs, effectively managing the LLMAgents component. This integration allowed for simplified management of LLM Agents' complexities, focusing attention on other critical parts of the system.

Adherence to DevOps practices was maintained throughout the development, including imple-

### 3.9. CRITICAL CHALLENGES AND STRATEGIC CONSIDERATIONS

menting a Continuous Integration/Continuous Deployment (CI/CD) pipeline. This pipeline was designed to process and integrate code changes automatically. The pipeline was triggered after each approved code change merged into the main branch. The process involved building Docker images for both the runtime and core application, conditional on whether there were updates in the code. Following this, the pipeline executed unit and integration tests. After successful testing, the newly built images were pushed to the Azure Container Registry. From there, they were deployed to the Azure Container App. Concurrently, the pipeline also handled the generation and updating of code documentation, which was then published on the project's website. The whole process is illustrated in Figure 3.7.

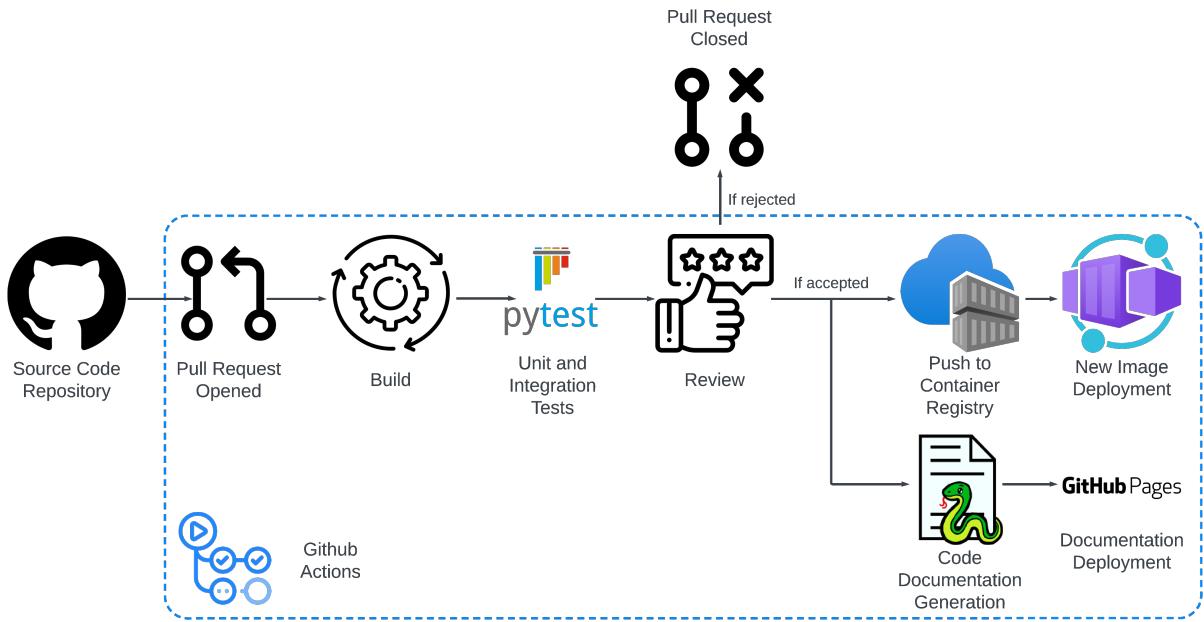


Figure 3.7: Continuous Integration and Continuous Deployment pipeline, initiated each time the new feature was pulled into the production version of the code base.

This deployment strategy not only streamlined the development process but also ensured that the application remained robust and reliable, with automated testing and deployment ensuring high code quality and system stability.

### 3.9. Critical Challenges and Strategic Considerations

While developing a multi-agent system for data analysis, several significant challenges illuminated the complexities of working with LLMs. These challenges are critical to understanding the obstacles inherent in creating a system of this nature.

A significant challenge was managing context overflow. Each LLM has its context length,

acting as a temporary memory. In this context, the model needs to handle user instructions and retain the history of the conversation, including analysis history, previous steps, and code along with its output. While some models like Mixtral offer a substantial context length of 32k tokens, others, such as LLaMA2, have a much smaller capacity of just 4k tokens. This limitation meant they could remember much less. As a mitigation, fallback models were implemented. When a model struggled with context length, the same model with adjustments was used. For instance, with LLaMA2 limitations, a different LLaMA model fine-tuned by Together.ai [20] was used, utilizing position interpolation [21].

Another issue was determining an appropriate termination point for the analysis. Based on a simple message count, the initial approach could have effectively reflected the analysis's completeness. Due to the inherently complex nature of data analysis, which lacks a uniform sequence of steps, some analyses were unnecessarily prolonged, while others might need to explore the topic thoroughly. Alternative approaches were discussed, such as checking for the completion of significant milestones like model generation. However, this method risked limiting the adaptability of LLMs to the provided dataset, a key feature intended to leverage. Another possibility was to entrust the decision to conclude the analysis to the Agent. While theoretically appealing, this presented practical challenges, such as the risk of an endless analysis loop and the difficulty of accurately detecting when the LLM suggests termination.

The error-handling aspect of the system development posed its unique set of complexities. In software engineering, isolating environments is crucial to mitigate the impacts of errors in one component on another. This was particularly relevant when the code generated by the models could contain bugs or potentially malicious elements. A remote runtime environment for code execution was implemented, ensuring isolation and containment. However, this solution introduced additional challenges, such as effectively transferring plots to different environments. Additionally, the size of Python tracebacks often contributed to context overflow, even when limiting them to the most recent errors. The selected approach only retained the essential parts of the traceback and the error code.

A notable problem arose when handling errors that occurred mid-execution in complex code snippets. If an error occurred in the middle of a complex code sequence, there was a risk that the Code Agent might redundantly retry steps that had already been successfully executed, potentially leading to inconsistencies in the changed state of the runtime environment. An example is the deletion of a column that already does not exist. Mitigation by executing all commands up to the point of the error posed the risk of reverse complications. An improved and more nuanced prompting strategy was hypothesized as a potential solution, aiming to make the process more

### 3.9. CRITICAL CHALLENGES AND STRATEGIC CONSIDERATIONS

reliable by guiding the Code Agent to avoid repeating successful steps and focus on addressing the error.

The last issue was the Agent not adhering to prompts. For example, LLaMA2 occasionally did not generate code despite explicit instructions. This could be attributed to the model's attention mechanism focusing more on the user's immediate request than the prompt, as LLMs prioritize user satisfaction. As mentioned in Section 2.3, the system prompt was a crucial element obliging the Agent to behave accordingly to the assigned role. Therefore, significant prioritization of the last request may have disrupted this role mechanism. The refinement of the prompting strategy mitigated the issue by incorporating specific queries in the messages between assistants. This approach explicitly requested the next step, such as code generation or analysis suggestions, to guide the assistants more effectively. Despite the efforts, responses still, from time to time, deviated from the expected format. This unpredictability underlines a critical aspect of integrating LLMs with systems that follow specific algorithms or rules. Achieving a balance between the flexibility of LLMs and the predictability required for structured processes proved to be a complex task.

These challenges underscore the complexities of working with LLMs in a multi-agent system, highlighting the need for innovative solutions and adaptive strategies to navigate the intricate dynamics of such a setup.

In concluding the chapter detailing the development of the MVP, a basic yet functional application that meets the objectives outlined earlier in this thesis was successfully created. While the application is in its nascent stages and lacks the comprehensive features of a mature product, it proved to be a pivotal tool in facilitating the practical evaluation of the proposed system. The application demonstrated sufficient stability, ensuring it did not impede the assessment process. The application's sample report is attached to this work in Appendix C. Potential directions for the future development of the application were outlined in Section 4.4.2.

## 4. Experiments and Results

Evaluating Large Language Models, particularly in the context of automated tabular data analysis, presents a unique set of challenges. The complexity and adaptability of these models, combined with the variability of data analysis, necessitate a multifaceted approach to assess their performance. In this chapter, the methodology evaluating the effectiveness of the presented solution is outlined, emphasizing the role of human feedback. Data was gathered and analyzed following this methodology, emphasizing the user experience and practical usability of the LLM outputs. This approach was crucial to a comprehensive understanding of the strengths and limitations of LLM-based solutions in real-world data analysis scenarios.

### 4.1. Parameters and Variables

In assessing the quality of data analysis generated by LLMs, it is crucial to consider various parameters that can significantly influence the outcome. Prominent are the specific agent model employed, the prompting methodology, and the prompts' intrinsic quality. Research primarily focused on the first two elements: the agent model and the prompt method. Although several prompting techniques were experimented with during the MVP development, the comprehensive results of these tests still need to be collated, limiting the current ability to provide a detailed analysis of this aspect. However, based on preliminary observations, it was hypothesized that the effectiveness of prompts plays a substantial role in shaping the quality of LLM-generated responses.

Additionally, other variables merit attention due to their potential impact on the behavior of LLMs in data analysis. These include the context size, which determines the extent of analysis history the model can consider. This leads to considering methods for summarizing or excluding historical data that exceeds the context capacity. Another variable is 'top\_p,' a parameter controlling the diversity of the generated text by focusing on a subset of the most probable words. This parameter balances coherence and variety in the model's output. Furthermore, the model's 'temperature' setting adjusts the randomness of its predictions. Lower temperature values

## 4.2. GENERATED CODE CORRECTNESS TEST

typically result in more predictable and conservative outputs, whereas higher values encourage diversity and reduce determinism in the responses.

While 'top\_p' and the model's temperature are fundamental parameters across the board for LLMs, the context-related aspects are particularly intriguing for further investigation in data analysis. This interest stems from the necessity for all steps in the analysis process to collectively form a coherent whole, making the management and understanding of context a pivotal area of exploration.

### 4.2. Generated Code Correctness Test

This study implemented a structured simulation test to evaluate the accuracy of the code produced by the Code Agent. This evaluation was essential to assess the system's effectiveness in generating correct and executable code snippets.

The simulation tested various combinations of used models and prompting techniques. These included different configurations of ZeroShot and FewShot approaches applied to models such as GPT-3.5, LLaMA2, and Mixtral8x7B. An analysis was conducted 45 times for each combination to ensure a thorough evaluation.

In alignment with the parameters outlined in Section 3.5, the analysis was limited to eight messages with a maximum of three retries for code errors. These constraints were critical for maintaining the uniformity and manageability of the test. The analysis was performed on the Red Wine Quality dataset [16].

Throughout this process, several vital metrics for each analysis were meticulously recorded. These metrics included the total number of code snippets that resulted in execution errors, the frequency of instances where the Code Agent's response omitted the required code snippet, and the count of messages exchanged prior to the premature termination of the analysis or analysis' success. Premature termination was noted in scenarios where the analysis exceeded the maximum retry limit for code errors or encountered other exceptions. In such cases, the returned error code was also documented.

Out of the targeted 270 results, 241 were obtained after completing the simulation. This shortfall was attributed to some unforeseen errors encountered during the process. The data collected were visually represented through plots for a more comprehensive analysis. In Figure 4.1, the distribution of analysis lengths is illustrated, quantified by the number of messages exchanged within the confines of the analysis length limit. Analyses culminating in 8 messages were those completed without any interruptions or exceptions.

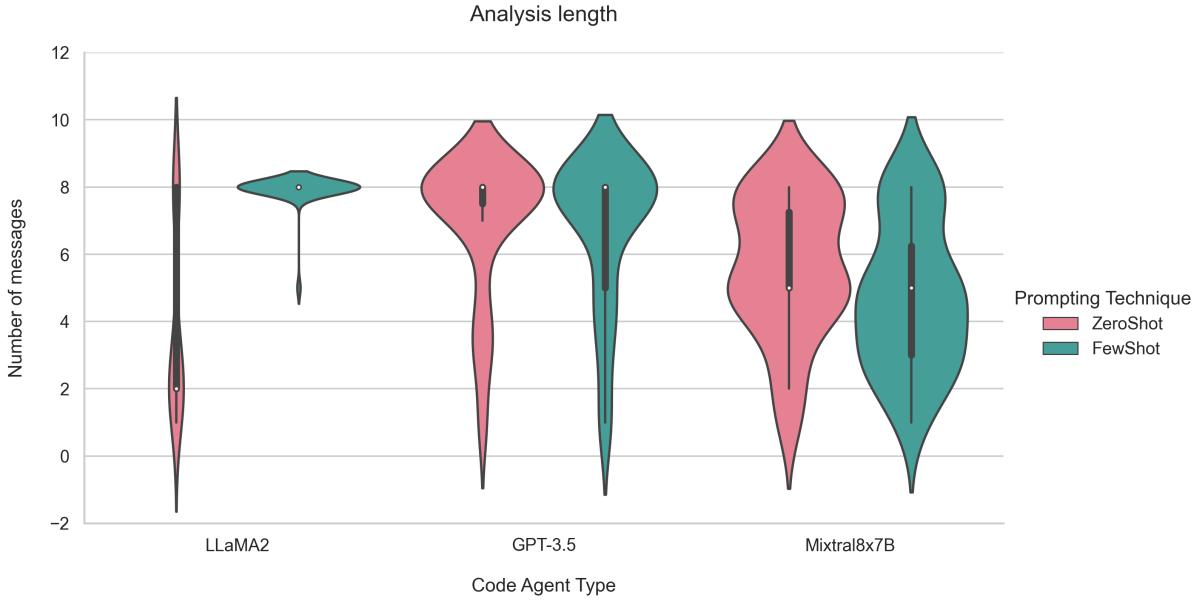


Figure 4.1: Analysis length distribution, given eight messages targeted, for different models and prompting methods. Analysis shorter than the target indicates an exception, among others, by too many unsuccessful code execution attempts.

Interestingly, Figure 4.1 reveals a notable pattern where the LLaMA2 model, when used with few-shot prompts, did not experience premature terminations. On the contrary, its zero-shot counterpart showed an almost uniform distribution across the scale, indicating a tendency towards random error occurrences and suggesting a lower reliability. The success rate for the LLaMA2 model with zero-shot prompts stood at 30%. In comparison, the Mixtral8x7B and GPT-3.5 models generally produced longer analyses. Among these, GPT-3.5 emerged as the most successful, achieving a 67.5% success rate with few-shot prompts and an even higher 74.4% with zero-shot prompts. Meanwhile, Mixtral8x7B had a lower success rate (20% for few-shot and 25% for zero-shot), indicating more frequent failures than successes.

Further detailed analysis is presented in Figure 4.2, which shows the combined number of messages from the Code Agent that either resulted in execution errors or contained no code. This measurement, called invalid code agent messages, shed light on the 100% success rate of LLaMA2 with few-shot prompts, revealing that it did not generate any code (as confirmed by the underlying data). With an average of 4 messages out of 4 belonging to the Code Agent, this finding underscores the limitation of the current analysis termination method in reflecting actual analysis progress. For other models, LLaMA2 with zero-shot prompts produced an exceptionally high number of invalid messages, while GPT-3.5 and Mixtral8x7B showed similar performance regardless of the prompting technique, mostly maintaining up to 2 invalid messages. Given the success rates from Figure 4.1, this implies that GPT-3.5 tended to handle invalid code more

#### 4.2. GENERATED CODE CORRECTNESS TEST

decisively or abandon efforts sooner, whereas Mixtral8x7B persistently attempted to rectify the code, often exceeding the code retry limit. This pattern indicates a more flexible approach by GPT-3.5 and a more vital adherence to system prompts by Mixtral8x7B (the prompt insisted on code generation).

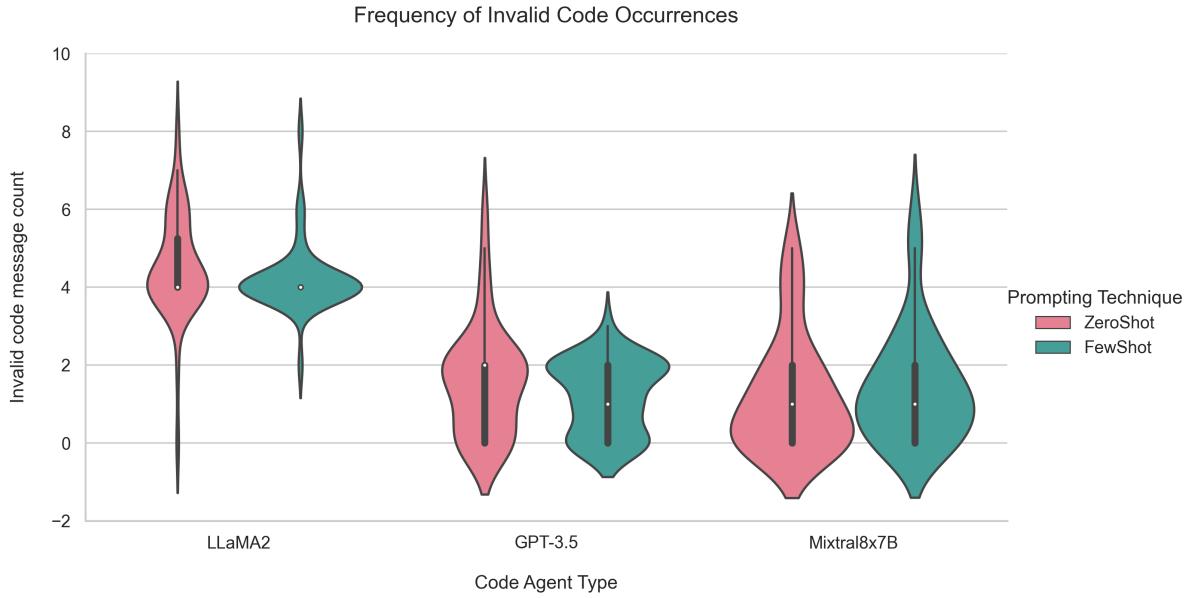


Figure 4.2: Number of Code Agent’s messages containing failing code or no code per analysis for different models and prompting methods.

Concluding both Figure 4.1 and Figure 4.2, few-shot prompting encourages more code generation in GPT-3.5 compared to zero-shot prompting, highlighting different operational tendencies between the models.

The final metric aggregated the occurrence of various types of termination errors encountered during the analysis. These included:

- **LLM failure:** errors related to LLM API connection or invalid responses.
- **Runtime error:** failures of connection to execution runtime.
- **Context overflow:** instances where the model’s context capacity was exceeded.
- **Code generation limit exceeded:** errors triggered after three unsuccessful code generation retries.

These termination errors are depicted in Figure 4.3 and Figure 4.4, which present heatmaps illustrating the frequency of these errors. The analysis of these figures provides additional insights, complementing the findings from the previous two figures.

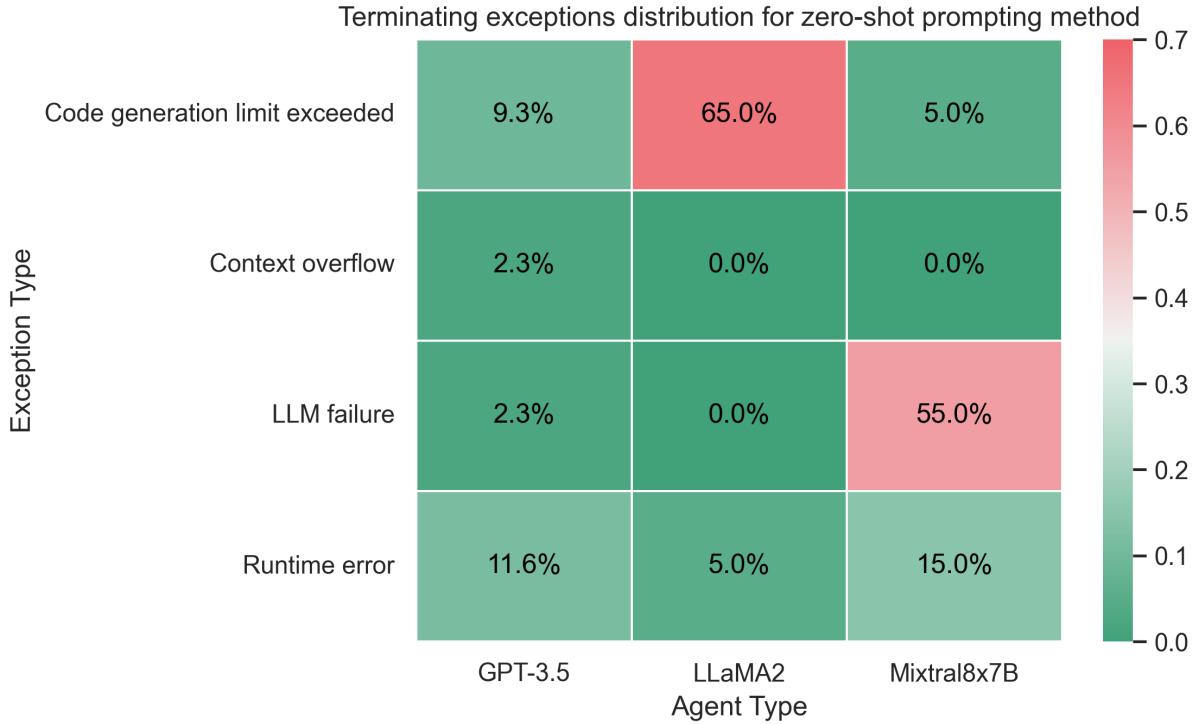


Figure 4.3: Frequency of errors during zero-shot analysis grouped by error category.

The data reinforced the poor performance of the LLaMA2 model with zero-shot prompting while revealing that the success rate of Mixtral8x7B was significantly impacted by inconsistent behavior. Notably, most LLM errors were attributed to empty messages containing stop tokens, indicative of model failures. This observation is crucial when revisiting the invalid code count in Figure 4.2. Although the distribution for Mixtral8x7B was similar to that of GPT-3.5, the success rate of Mixtral8x7B might have been comparable to GPT-3.5 if not for these internal failures.

Context overflow occurred infrequently, but the relatively short length of the analyses limits the conclusions that can be drawn about its impact. The predominant cause of runtime errors was a timeout in the WebSocket pipe between the runtime environment and the application, often resulting from slow responses from the Agent’s API. This aspect of the system’s architecture highlights the need for optimization to ensure smoother communication and more reliable performance.

### 4.3. Human Feedback Evaluation of Analysis Quality

The simulation results provided valuable insights into the capabilities of LLMs concerning different prompting methods. However, it is essential to note that the successful completion of

#### 4.3. HUMAN FEEDBACK EVALUATION OF ANALYSIS QUALITY

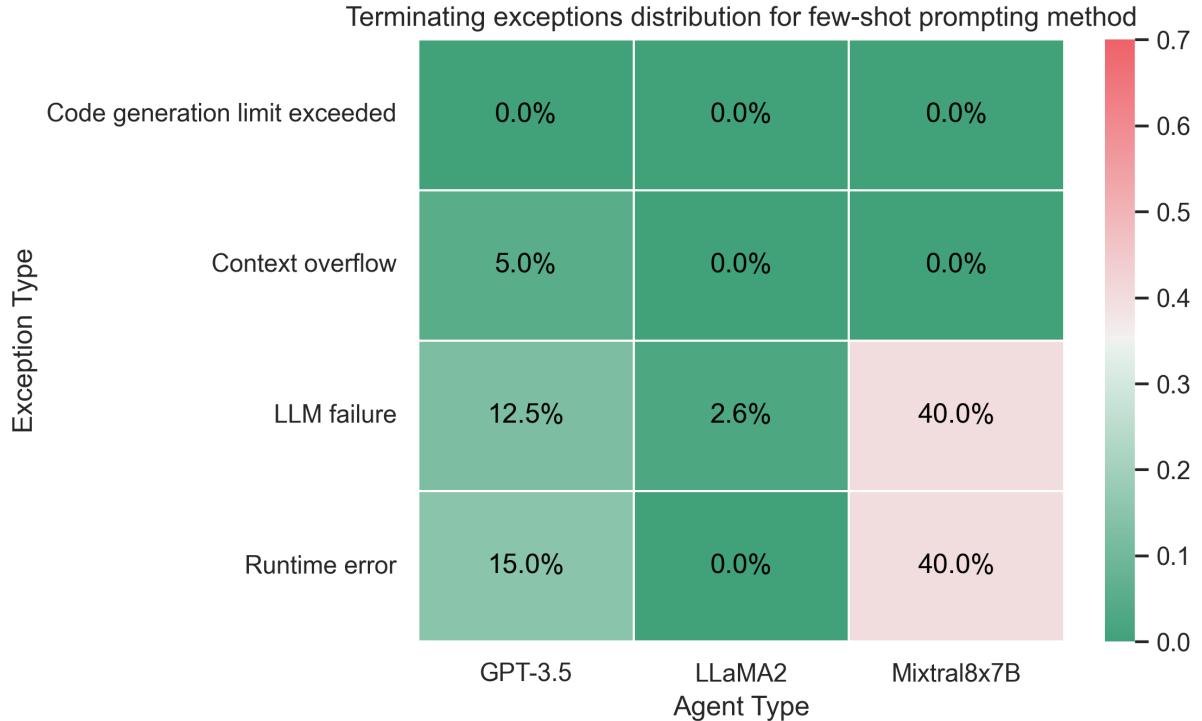


Figure 4.4: Frequency of errors during few-shot analysis grouped by error category.

an analysis does not necessarily equate to high-quality analysis. A striking example of this is observed with the LLaMA2 model using few-shot prompts, as discussed in Section 4.2. Despite achieving a 100% success rate in analysis, the resulting reports contained no executed lines of code, indicating no substantive exploration of the data properties. This outcome is not entirely surprising, considering the complexity of data analysis and the lack of specific rules guiding the process flow in the proposed system.

A human feedback test evaluated the models' performance in data analysis tasks more accurately. The rationale behind this test was that the ultimate recipient of the analysis report is the user, who is best positioned to judge the effectiveness of the output.

The test involved volunteers providing datasets in CSV format, for which reports were generated using each parameter combination (combinations unchanged from Section 4.2). This approach tested the system across diverse real-world data. Participants were provided unlabelled reports and asked to grade them on a scale from 1 to 6. All reports had to be graded for submission, and participants were given the option to provide additional feedback in a free-text format.

In total, 28 responses were collected. The results, illustrated in Figure 4.5, show the distribution of scores for each configuration. LLaMA2 with zero-shot prompts generally received the lowest scores, often struggling to load datasets. LaMA2 with few-shot prompts fared slightly

better but needed to be rated better, primarily due to a lack of code generation and drawing unsupported conclusions. Mixtral8x7B showed improvement, though its zero-shot variant struggled with code correctness, resulting in lower-end ratings. The few-shot Mixtral8x7B exhibited a more balanced distribution, though with a tendency towards lower scores, indicating issues with the quality of visualizations and occasionally generating more extended code snippets that failed partway, impacting the overall analysis results. In contrast, GPT-3.5 outperformed the others, with zero-shot prompting achieving a symmetrical distribution and few-shot prompting averaging above a score of 4. In each model’s case, the few-shot approach outperformed the zero-shot counterpart.

Further insights from the textual feedback highlighted that LLaMA2 often failed to generate code, Mixtral8x7B, while producing more code, delivered lower-quality visualizations, and GPT-3.5 generally performed the best overall. These observations underscore each model’s varying strengths and weaknesses in the context of data analysis tasks.

#### 4.4. Future Work

Sections 4.2 and 4.3 evaluated how data analysis performance varies using different prompt techniques and models. However, this analysis represents only a preliminary overview and lacks a more comprehensive understanding. Future studies should encompass a broader range of parameters, ideally incorporating all those mentioned in Section 4.1. Additionally, exploring prompt techniques should not be limited to zero-shot and few-shot methods. A more thorough examination of various prompting techniques would yield a holistic assessment of their effectiveness.

Another aspect that needs to be addressed in this study is systematically comparing the solution with other competitive models. The initial plan assumed an analysis generated by the Data Analyst ChatGPT plugin, created by OpenAI, in the human-feedback survey (Section 4.3). However, the challenge lies in establishing a methodology to prevent bias, especially since the data analyst requires user interaction. Converting its output to a format akin to that generated by the developed application proved problematic, as it was difficult to devise a consistent method for such a conversion.

Furthermore, it is essential to note that this study utilized GPT-3.5 instead of the more advanced GPT-4, known for its superior outputs. Cost considerations, particularly evident, primarily drove the decision to use GPT-3.5 during the simulation tests in Section 4.2. The higher cost of GPT-4 would have significantly impacted the feasibility of extensive testing. This limitation in the choice of model is an essential factor to consider when interpreting the results and

#### 4.4. FUTURE WORK

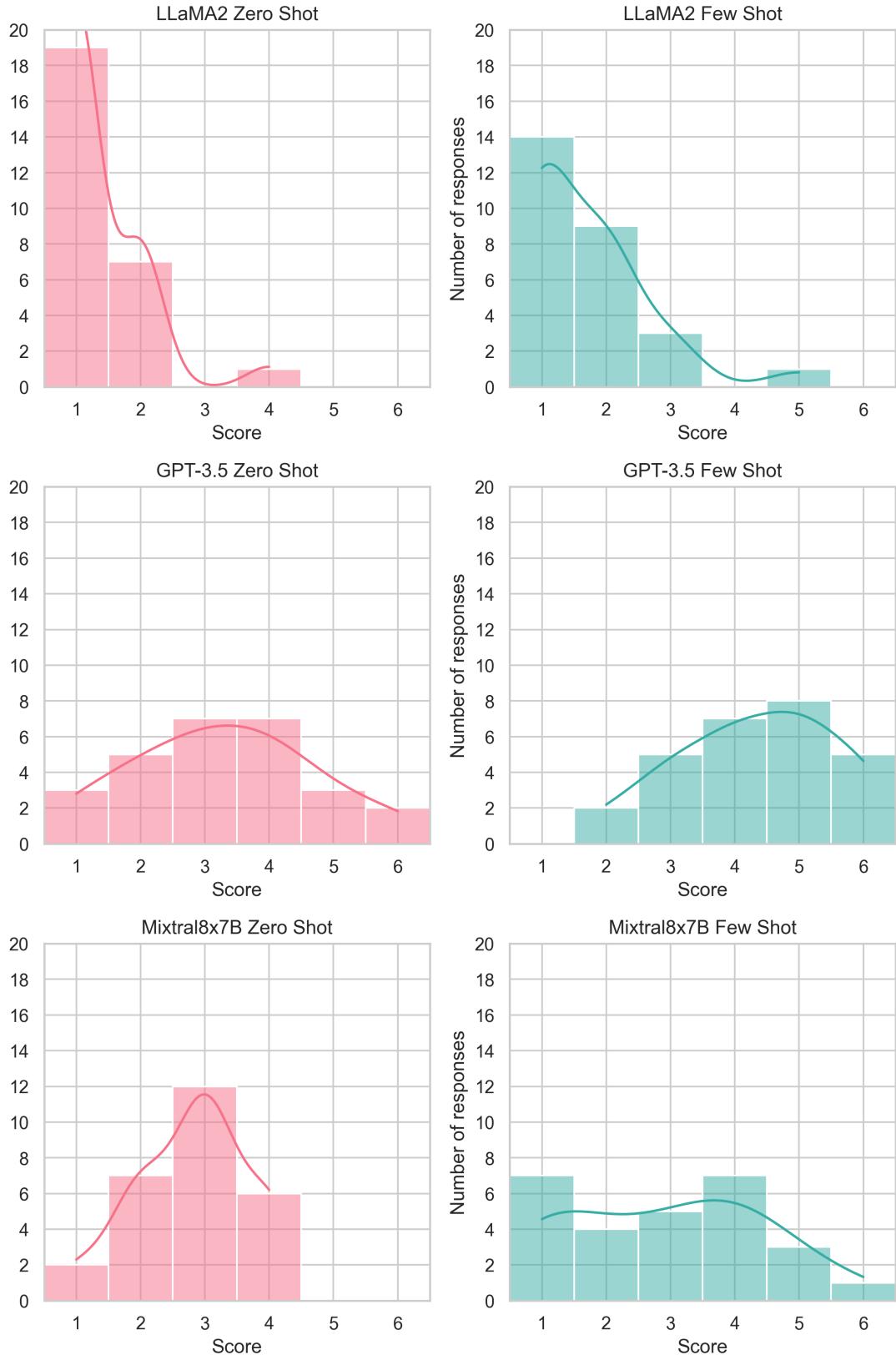


Figure 4.5: Distribution of report ratings in a human-feedback survey by prompting methods and models.

implications of this study.

Additionally, the model CodeLLaMA, developed by META as an equivalent to LLaMA2 specifically for code generation, warrants further consideration. LLaMA2 was employed for both analysis and code generation roles. Challenges encountered while attempting to use CodeLLaMA through the Together.ai API influenced this decision. The issues caused by CodeLLaMA integration into the system are noteworthy and highlight the practical complexities of implementing specialized LLMs in a multi-agent setup.

#### 4.4.1. System Design Revisited

In addition to refining the testing methodologies, several aspects of the system’s design require enhancement to improve its predictability, fault tolerance, and overall quality of results. One overlooked aspect is handling scenarios where the Code Agent fails to produce any code in its output. While the designed system currently addresses situations involving erroneous code, it lacks mechanisms to manage cases where code is absent altogether. Addressing this issue, including not counting such messages towards the analysis length limit, could lead to a more systematic output. This change could reduce instances where the analysis needs more basic calculations and data processing.

Another area for potential improvement is the reinforcement of role assignments within the system. Observations suggest that system prompts are often deprioritized in favor of the most recent message in the conversation. To counter this, the last message could be enveloped within a boilerplate template. Alternatively, reminder messages could be inserted into the conversation. While both approaches are promising and warrant exploration, they require careful evaluation to ensure they do not disrupt the Agents’ behavior.

Furthermore, integrating a robust conversation summary mechanism is another area that needs improvement. An effectively implemented conversation summary could manage extended analyses more efficiently. This would involve developing a method that succinctly encapsulates lengthy dialogues without compromising the quality or context of the analysis. The implementation of these enhancements would not only refine the system’s functionality but contribute to a more coherent and reliable analysis process.

Addressing the challenge of effectively terminating the analysis process is one of the most daunting yet potentially rewarding tasks for future system enhancements. A promising solution under consideration is introducing an additional Agent tasked with overseeing the entire analysis process. This Agent would be responsible for determining, after each step, whether the data analysis of a given dataset is complete. Implementing such a mechanism would necessitate

#### 4.4. FUTURE WORK

meticulous planning and strategic instructions to ensure the Agent operates as intended. This approach, however, would still require an upper limit, akin to the hard limit currently employed, to preclude the possibility of infinite loops in the analysis process. The successful integration of this oversight Agent could revolutionize analysis workflow, potentially elevating the system's efficiency and effectiveness in handling complex data analysis tasks.

##### 4.4.2. Advancing Towards Commercial Viability

While preparing enterprise-grade applications was outside the scope of this thesis, it is worth mentioning what key areas need to be improved to meet enterprise-grade standards. Among them are:

- **Scalability and Security:** Transitioning from Jupyter Notebook to Jupyter Hub or Apache Zeppelin are essential for scalability. These platforms offer enhanced scalability and more robust authentication mechanisms, critical for any enterprise-grade application. While scalable, the current deployment on Azure Container Apps is limited in configuration, leading to security concerns due to the runtime environment's exposure to the public internet. A more secure approach would involve configuring the runtime environment accessible exclusively to the system's core application, ensuring secure and isolated data processing.
- **Technical Architecture:** Using Streamlit in the current setup, while effective for a proof of concept, presents scalability and architectural flexibility limitations. A transition to more advanced frameworks such as React or Angular is recommended to embrace a microservices architecture fully. This would enable a clear separation of the backend and frontend, aligning with modern web development practices and offering a more scalable, robust, and flexible frontend. This change would force reengineering the core system into an API application, aligning with microservices architecture principles and enhancing modularity and interoperability. Additionally, implementing a custom domain would provide a professional web presence, which is crucial for a commercial product.
- **Usability and Functionality Improvements:** Enhancing user experience by introducing user profiles and efficient data management is essential. A database-backed system for user profiles, historical dataset management, and tools for reviewing past analyses and chats would significantly elevate the application's utility and user engagement. On top of that, live results functionality, providing a partial analysis report overview before the process ends, would alleviate the user's distress and bring value faster.

#### 4. EXPERIMENTS AND RESULTS

These outlined enhancements are vital for transforming the MVP into a tool suitable for enterprise-level deployment and poised for successful commercial market entry. Addressing these areas is integral to meeting the commercial market's high standards and diverse demands.

## Conclusions

Exploring multi-agent Large Language Model (LLM) systems signals a significant advancement in AI solutions, heralding an era of increased autonomy and enhanced capability in handling complex and unstructured tasks. With their immense potential, these systems also introduce various new challenges. During this project, a basic application that demonstrated the foundational capabilities of such a system was successfully developed and deployed. Throughout this endeavor, the non-deterministic nature of LLMs became evident, yielding results that ranged from spectacular to subpar. This variability underscores that while multi-agent LLM systems present an intriguing area of research, they still need to be viable replacements for human data analysts.

However, these systems' potential and unique capabilities make it a worthwhile pursuit. Further in-depth research and exploration are essential to harness their full capabilities. Additionally, experimenting with existing multi-agent frameworks would shift the focus towards optimizing these systems specifically for data analysis tasks. This approach could streamline the development process and expedite the evolution of multi-agent LLM systems into more refined and reliable tools. Overall, experiences and findings emphasize the importance of continued research in this field, aiming to bridge the gap between current capabilities and the future potential of multi-agent LLM systems in data analysis and beyond.

The generated output was generally coherent and readable regarding the specific Large Language Models (LLMs) that were evaluated. Regarding performance, the few-shot approach yielded better results than the zero-shot method. Among the models tested, GPT-3.5 emerged as the most reliable in terms of output consistency and reliability. However, a significant challenge that needs to be addressed is developing a method to assess the completeness and value of the generated analysis without human intervention. This challenge points to the need for further research in automating the evaluation process, aiming to quantify the effectiveness of LLMs in data analysis tasks in an objective and scalable manner.

## Bibliography

- [1] OpenAI, *GPT-4 Technical Report*, <https://arxiv.org/abs/2303.08774>
- [2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Y. Zheng, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, E. Edunov, T. Scialom, "Llama 2: Open Foundation and Fine-Tuned Chat Models," *arXiv.org*, Cornell University Library, arXiv.org, 2023.
- [3] L. Cheng, X. Li, and L. Bing, "Is GPT-4 a Good Data Analyst?" <https://arxiv.org/abs/2305.15038>
- [4] K. Gu, R. Shang, T. Althoff, C. Wang, and S. M. Drucker, "How do Analysts Understand and Verify AI-Assisted Data Analyses?" <https://arxiv.org/abs/2309.10947>
- [5] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT," <https://arxiv.org/abs/2302.11382>
- [6] T. Khot, H. Trivedi, M. Finlayson, Y. Fu, K. Richardson, P. Clark, and A. Sabharwal, "Decomposed Prompting: A Modular Approach for Solving Complex Tasks," <https://arxiv.org/abs/2210.02406>
- [7] G. Li and H. Abed A. K. Hammoud and H. Itani and D. Khizbulin and B. Ghanem, "CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society," *arXiv.org*, Cornell University Library, arXiv.org, 2023
- [8] Noteable, <https://noteable.io/>

## BIBLIOGRAPHY

- [9] josephjrobison, *Noteable.io is shutting down*, <https://news.ycombinator.com/item?id=38659315>, 16 December 2023
- [10] Noteable, *ChatGPT Plugin for Notebook*, <https://noteable.io/chatgpt-plugin-for-notebook/>
- [11] tizz98, <https://github.com/tizz98/jupychat>
- [12] Langchain, <https://langchain.com/>
- [13] behnamoh, *Langchain Is Pointless*, <https://news.ycombinator.com/item?id=36645575>
- [14] Noteable, *Noteable DEX package*, <https://github.com/MSeal/dx>
- [15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention Is All You Need,” <https://arxiv.org/abs/1706.03762>
- [16] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, J. Reis, *Modeling wine preferences by data mining from physicochemical properties*, Decision Support Systems, Vol. 47, No. 4, pp. 547-553, 1998.
- [17] J. López Espejel, E. H. Ettifouri, M. S. Y. Alassan, E. M. Chouham, W. Dahhane, “GPT-3.5, GPT-4, or BARD? Evaluating LLMs Reasoning Ability in Zero-Shot Setting and Performance Boosting Through Prompts,” <https://arxiv.org/abs/2305.12477>
- [18] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. Bou Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. Le Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, W. El Sayed, “Mixtral of Experts,” <https://arxiv.org/abs/2401.04088>
- [19] S. Shen, L. Hou, Y. Zhou, N. Du, S. Longpre, J. Wei, H. W. Chung, B. Zoph, W. Fedus, X. Chen, T. Vu, Y. Wu, W. Chen, A. Webson, Y. Li, V. Zhao, H. Yu, K. Keutzer, T. Darrell, D. Zhou, “Mixture-of-Experts Meets Instruction Tuning: A Winning Combination for Large Language Models,” <https://arxiv.org/pdf/2305.14705.pdf>
- [20] Together *Preparing for the era of 32K context: Early learnings and explorations*, <https://www.together.ai/blog/llama-2-7b-32k>
- [21] Together *Extending context windows of large language models via position interpolation*, <https://arxiv.org/pdf/2306.15595.pdf>

- [22] S. Makridakis, F. Petropoulos, and Y. Kang, “Large Language Models: Their Success and Impact,” *Forecasting*, vol. 5, no. 3, pp. 536–549, 2023.
- [23] J. G. Meyer, R. J. Urbanowicz, P. C. N. Martin, K. O’Connor, R. Li, P.-C. Peng, T. J. Bright, N. Tatonetti, K. J. Won, G. Gonzalez-Hernandez, and J. H. Moore, “ChatGPT and large language models in academia: opportunities and challenges,” *BioData mining*, vol. 16, no. 1, pp. 20–20, 2023.
- [24] G. Polverini and B. Gregorcic, “How understanding large language models can inform the use of ChatGPT in physics education,” *European journal of physics*, 2023.
- [25] G. Paafß and S. Giesselbach, *Foundation Models for Natural Language Processing: Pre-trained Language Models Integrating Media*, Springer Nature, Cham, 2023. [Series: Artificial Intelligence: Foundations, Theory, and Algorithms]. ISBN 3031231899.
- [26] T. Alqahtani, H. A. Badreldin, M. Alrashed, A. I. Alshaya, S. S. Alghamdi, K. bin Saleh, S. A. Alowais, O. A. Alshaya, I. Rahman, M. S. Al Yami, and A. M. Albekairy, “The emergent role of artificial intelligence, natural learning processing, and large language models in higher education and research,” *Research in social and administrative pharmacy*, vol. 19, no. 8, pp. 1236–1242, 2023.
- [27] E. Shin and M. Ramanathan, “Evaluation of prompt engineering strategies for pharmacokinetic data analysis with the ChatGPT large language model,” *Journal of pharmacokinetics and pharmacodynamics*, 2023.
- [28] V. Braga, “DECENTRALISED AUTONOMOUS SOCIETY THROUGH LARGE LANGUAGE MODELS’ BASED AGENTS: A PATHWAY TO EMPOWER SMALL COMMUNITIES,” *Journal of Engineering Science (Chișinău)*, vol. 30, no. 3, pp. 99–120, 2023.
- [29] M. Tian, T. Yang, and W. Yin, “Few-Shot Classification & Segmentation Using Large Language Models Agent,” *arXiv.org*, Cornell University Library, arXiv.org, 2023.
- [30] Y. Kong, J. Ruan, Y. Chen, B. Zhang, T. Bao, S. Shi, G. Du, X. Hu, H. Mao, Z. Li, X. Zeng, and R. Zhao, “TPTU-v2: Boosting Task Planning and Tool Usage of Large Language Model-based Agents in Real-world Systems,” *arXiv.org*, Cornell University Library, arXiv.org, 2023.
- [31] K. Sanderson, “GPT-4 is here: what scientists think,” *Nature (London)*, vol. 615, no. 7954, pp. 773–773, 2023.

## BIBLIOGRAPHY

- [32] R. Gupta, J. B. Park, I. Herzog, N. Yosufi, A. Mangan, P. K. Firouzbakht, and B. A. Mailey, “Applying GPT-4 to the Plastic Surgery Inservice Training Examination,” *Journal of plastic, reconstructive & aesthetic surgery*, vol. 87, pp. 78–82, 2023.
- [33] B. Meskó, “Prompt Engineering as an Important Emerging Skill for Medical Professionals: Tutorial,” *Journal of medical Internet research*, vol. 25, no. 6, pp. e50638–e50638, 2023.
- [34] L. Xu, Z. Hu, D. Zhou, H. Ren, Z. Dong, K. Keutzer, S. K. Ng, and J. Feng, “MAgIC: Investigation of Large Language Model Powered Multi-Agent in Cognition, Adaptability, Rationality and Collaboration,” *arXiv.org*, Cornell University Library, arXiv.org, 2023.
- [35] S. Ouyang and J. M. Zhang and M. Harman and M. Wang, “LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation,” *arXiv.org*, Cornell University Library, arXiv.org, 2023.
- [36] W. Zhang, Y. Shen, W. Lu, Y. Zhuang, “Data-Copilot: Bridging Billions of Data and Humans with Autonomous Workflow,” *arXiv.org*, Cornell University Library, arXiv.org, 2023.
- [37] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W White, D. Burger, C. Wang , “AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation,” *arXiv.org*, Cornell University Library, arXiv.org, 2023.
- [38] DeepL, *How does DeepL work?* <https://www.deepl.com/en/blog/how-does-deepl-work>
- [39] R. Wang, M. Wu, Y. Wang, X. Han, C. Zhang, and H. Li, “Understanding the Instruction Mixture for Large Language Model Fine-tuning,” *arXiv.org*, Cornell University Library, arXiv.org, 2023. few-shot
- [40] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, “Language Models are Few-Shot Learners,” *arXiv.org*, Cornell University Library, arXiv.org, 2020.
- [41] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, “Zero-Shot Learning – A Comprehensive Evaluation of the Good, the Bad and the Ugly,” *arXiv.org*, Cornell University Library, arXiv.org, 2020.
- [42] Y. Nakajima, <https://github.com/yoheinakajima/babyagi?tab=readme-ov-file>

- [43] M. S. Brown, “Transforming Unstructured Data into Useful Information,” in *Big Data, Mining, and Analytics*, Auerbach Publications, 2014, pp. 227–246, ISBN 978-0-429-09529-0.

## **List of symbols and abbreviations**

AI	Artificial Intelligence
CD	Continuous Deployment
CI	Continuous Integration
GUI	Graphical User Interface
LLM	Large Language Model
NLP	Natural Language Processing
NLU	Natural Language Understanding
MVP	Minimum Viable Product

## List of Figures

2.1	Simplified flow of analysis process in the proposed system. With its cyclical pattern, the system is in the middle, while input and output objects are on the sides. While the cycle goes toward the arrow, the moderator supervises each transition between stages. . . . .	16
2.2	Main part of Noteable platform, a notebook where text, code, and visualizations meet in one place. . . . .	20
2.3	ChatGPT plugin for Noteable: The user can interact with the notebook via intuitive, natural language, thanks to OpenAI's model capabilities. . . . .	21
2.4	Sample user conversation with an OpenAI's Data Analyst . . . . .	23
3.1	A user uploads their dataset via the web interface, and after automated analysis, a detailed report is generated for review or download, all with minimal user input required. . . . .	28
3.2	Application architecture. Each module is represented as a separate rectangle, while ellipses are instances. Lines represent how the system is interconnected, with an annotation of the protocol used for communication. . . . .	31
3.3	Class diagram providing a visual representation of the backend application's architecture. It showcases the ConversationManager class as a central element utilized by the program. Interfaces are strategically defined for interaction with each external component, illustrating the system's modular design. . . . .	32
3.4	Primary workflow illustrating the progression of data through various stages of the analysis. This process is not linear, incorporating various conditions and loops. These elements are enumerated in Listing 3.1. . . . .	36
3.5	User Interface as a simple single-page web page . . . . .	39
3.6	Cloud deployment infrastructure located in Microsoft Azure. All the components communicate via the public Internet. . . . .	40
3.7	Continuous Integration and Continuous Deployment pipeline, initiated each time the new feature was pulled into the production version of the code base. . . . .	41

4.1	Analysis length distribution, given eight messages targeted, for different models and prompting methods. Analysis shorter than the target indicates an exception, among others, by too many unsuccessful code execution attempts. . . . .	46
4.2	Number of Code Agent’s messages containing failing code or no code per analysis for different models and prompting methods. . . . .	47
4.3	Frequency of errors during zero-shot analysis grouped by error category. . . . .	48
4.4	Frequency of errors during few-shot analysis grouped by error category. . . . .	49
4.5	Distribution of report ratings in a human-feedback survey by prompting methods and models. . . . .	51

## List of Tables

A.1	Team Members' Responsibilities . . . . .
B.1	Backend Application . . . . .
B.2	Frontend Application . . . . .
B.3	Python Runtime . . . . .
B.4	Jupyter Notebook Server . . . . .
B.5	Containerization and Deployment . . . . .
B.6	Continuous Integration and Continuous Deployment (CI/CD) . . . . .
B.7	Code Documentation and Hosting . . . . .
B.8	Code Testing . . . . .

## **List of Appendices**

A. Division of Work

B. Technological Stack

C. Exemplary report

D. Used Prompts

## A. Division of Work

This thesis results from a collaborative effort between Filip Kołodziejczyk and Jakub Świs-tak. The division of work has been organized to leverage the strengths and expertise of each team member. Table A presents the division of work for both the thesis creation and MVP de-velopment. This table provides an overview of how responsibilities were allocated between the authors, ensuring a comprehensive and well-structured academic document and showcasing the collaborative nature of the project and the areas of expertise each author brought to the system's creation.

<b>Member</b>	<b>Task</b>	<b>Description</b>
Jakub Świstak	Application Design	Leading the overall design of the application, including its architecture and user interface.
	Prompt Engineering	Developing and refining prompts for effective use of Large Language Models.
	LLM Integration	Integrating Large Language Models into the application and creating an interface for the model's API.
	Analysis Performance	Testing performance of various prompts and models.
	Thesis	Introduction, Chapter 1 and Conclusions
Filip Kołodziejczyk	Execution Environment	Setting up and maintaining the execution environment, including environment setup and connecting execution runtimes.
	CI/CD	Managing the CI/CD pipeline for automated build, test, and deployment processes.
	Report Generation	Implementing a feature for generating reports, combining executed code with outputs in a structured format.
	Application Tests	Designing and conducting tests to validate the functionality, performance, and security of the application.
	Thesis	Chapter 2, 3, and, 4; Diagrams

Table A.1: Team Members' Responsibilities

## B. Technological Stack

This document provides a comprehensive inventory of the tools and resources utilized in developing our system, categorized according to the various modules of the system.

Table B.1: Backend Application

Tools/Libraries	Type	Description/Justification
Python 3.10	Programming Language	Selected for its robustness and extensive library support.
<code>openai</code>	Communication Library	For interactions with OpenAI's LLMs.
<code>paramiko</code>	SSH Library	For SSH connections and operations.
<code>websocket-client</code>	WebSocket Library	For real-time data communication.
<code>requests</code>	HTTP Library	For making HTTP requests.
<code>nbformat</code> , <code>nbconvert</code>	Data Processing Libraries	For creating, editing, and converting Jupyter Notebooks.
<code>Pydantic</code>	Data Validation Library	Ensures data integrity and validation.

Table B.2: Frontend Application

Tools/Libraries	Type	Description/Justification
Python 3.10	Programming Language	Maintains consistency and integration with the backend.
<code>streamlit</code>	UI Framework	For creating an intuitive web interface.
<code>pdf2image</code> , <code>reportlab</code>	Reporting Tools	For converting PDFs to images and creating PDF reports.

Table B.3: Python Runtime

Tools/Libraries	Type	Description/Justification
SSH	Connectivity Tool	For secure shell connectivity.
pandas, scikit-learn, numpy, matplotlib, seaborn	Data Science Libraries	Libraries for common data science needs.

Table B.4: Jupyter Notebook Server

Tools/Libraries	Type	Description/Justification
scipy-notebook	Notebook Version	Incorporates popular packages from the scientific Python ecosystem.

Table B.5: Containerization and Deployment

Tools/Libraries	Type	Description/Justification
Docker	Containerization Tool	For creating, deploying, and running applications in containers, especially for local deployment.
Docker Compose	Deployment Tool	Facilitates the definition and sharing of multi-container applications.
Azure	Cloud Platform	Utilized for robust and scalable online deployment.

Table B.6: Continuous Integration and Continuous Deployment (CI/CD)

Tools/Libraries	Type	Description/Justification
GitHub Actions	CI/CD Tool	Automates CI/CD pipeline, enhances integration and testing efficiency.

Table B.7: Code Documentation and Hosting

Tools/Libraries	Type	Description/Justification
pdoc3	Documentation Tool	For automatic generation of detailed code documentation.
GitHub Pages	Hosting Service	Used to host the documentation, making it easily accessible and navigable.

Table B.8: Code Testing

<b>Tools/Libraries</b>	<b>Type</b>	<b>Description/Justification</b>
pytest	Testing Framework	Integral for ensuring code quality and reliability through comprehensive testing.

## C. Exemplary report

This appendix presents an exemplary report that illustrates the capabilities and methodology employed in the proposed analysis process. The report showcased here was generated while examining the 'census.csv' file, a part of the "Antidepressant Use in Scandinavia" dataset. This dataset is publicly accessible at <https://www.kaggle.com/datasets/thedevastator/antidepressant-use-in-scandinavia>.

For this analysis, OpenAI assistant was utilized, employing the zero-shot prompting technique. The entire process was limited to eight interaction messages.

# Analysis of dataset census

January 23, 2024

```
[1]: import pandas as pd  
df= pd.read_csv('census.csv', sep=',')
```

Dataset is loaded into the runtime in the variable ‘df’. You can try to print the first 5 rows of the dataset by executing the following code: `python df.head()`

```
[2]: df.head()
```

```
[2]:   year sex    age      cnt country  
0  2007   F  5-9  164148      DK  
1  2007   F  10-14  171864      DK  
2  2007   F  15-19  157037      DK  
3  2007   M  5-9  172068      DK  
4  2007   M  10-14  181190      DK
```

The dataset appears to contain information about different age groups, genders, years, countries, and the corresponding counts. In the provided output, we can see the first 5 rows of the dataset, which include columns such as ‘year’, ‘sex’, ‘age’, ‘cnt’ (count), and ‘country’.

Based on this initial view of the dataset, the following observations can be made:

- The dataset seems to capture data for multiple years.
- The ‘sex’ column distinguishes between males and females.
- The ‘age’ column categorizes individuals into age ranges.
- The ‘cnt’ column represents the count or frequency of occurrences.
- The ‘country’ column identifies the country to which the data pertains.

To gain a deeper understanding of the dataset, we should further explore its structure, contents, and relationships between variables.

```
[3]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 198 entries, 0 to 197  
Data columns (total 5 columns):  
 #   Column   Non-Null Count  Dtype    
---  --    
 0   year     198 non-null    int64  
 1   sex      198 non-null    object  
 2   age      198 non-null    object  
 3   cnt      198 non-null    int64  
 4   country  198 non-null    object
```

```
dtypes: int64(2), object(3)
memory usage: 7.9+ KB
```

```
[4]: df.isnull().sum()
```

```
[4]: year      0
      sex       0
      age       0
      cnt       0
      country   0
      dtype: int64
```

```
[5]: df.describe()
```

```
[5]:          year            cnt
count    198.000000    198.000000
mean     2012.000000  200329.277778
std      3.170294    54790.688206
min     2007.000000  145827.000000
25%    2009.000000  160038.000000
50%    2012.000000  170328.000000
75%    2015.000000  253733.750000
max     2017.000000  329731.000000
```

```
[6]: df['sex'].value_counts()
```

```
[6]: sex
      F    99
      M    99
      Name: count, dtype: int64
```

```
[7]: df.groupby('year')['cnt'].sum()
```

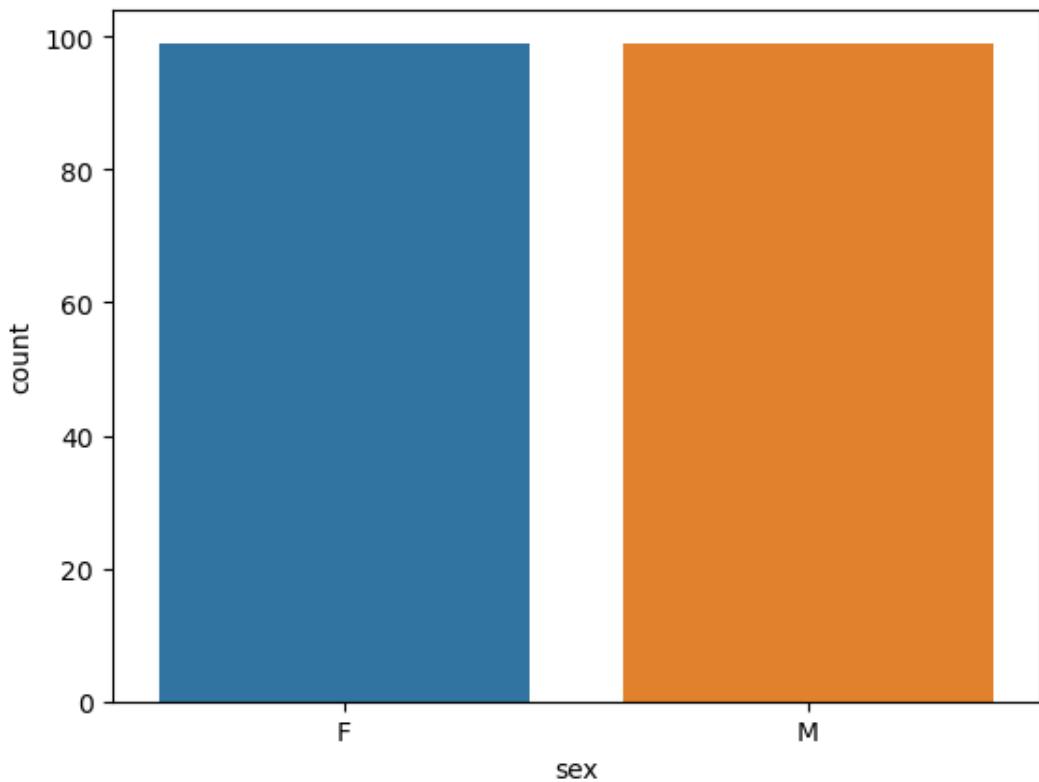
```
[7]: year
2007    3589963
2008    3594545
2009    3596804
2010    3583784
2011    3571842
2012    3567081
2013    3573811
2014    3591484
2015    3620881
2016    3665439
2017    3709563
      Name: cnt, dtype: int64
```

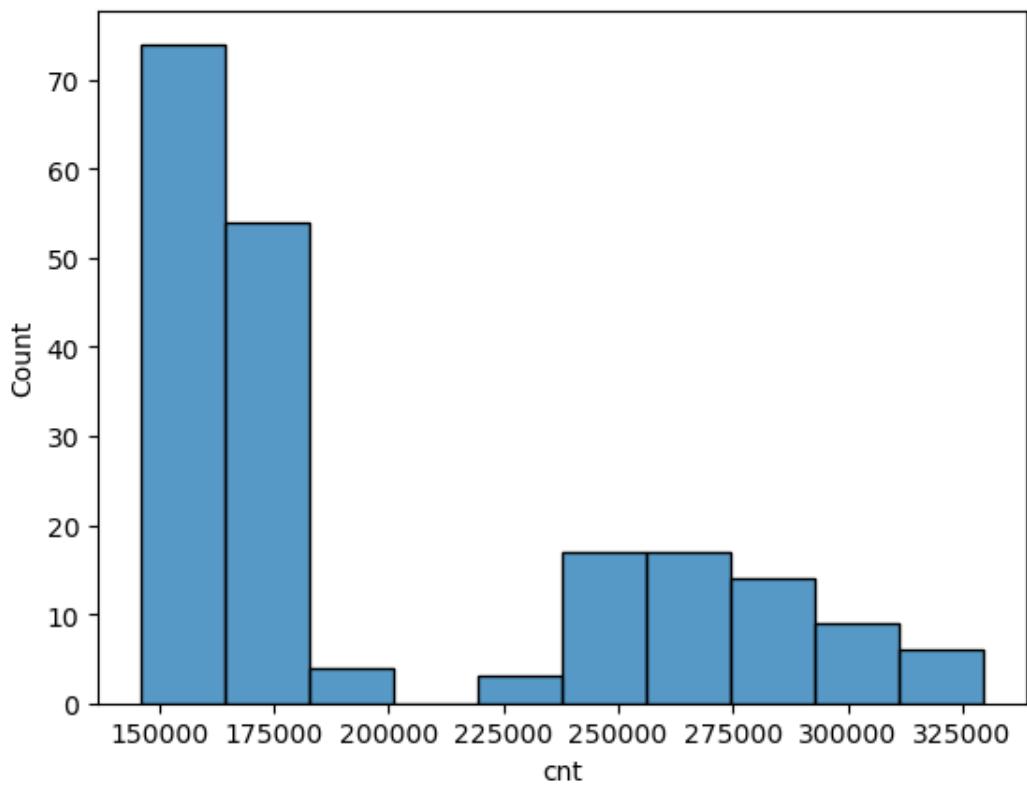
```
[8]: import matplotlib.pyplot as plt
import seaborn as sns

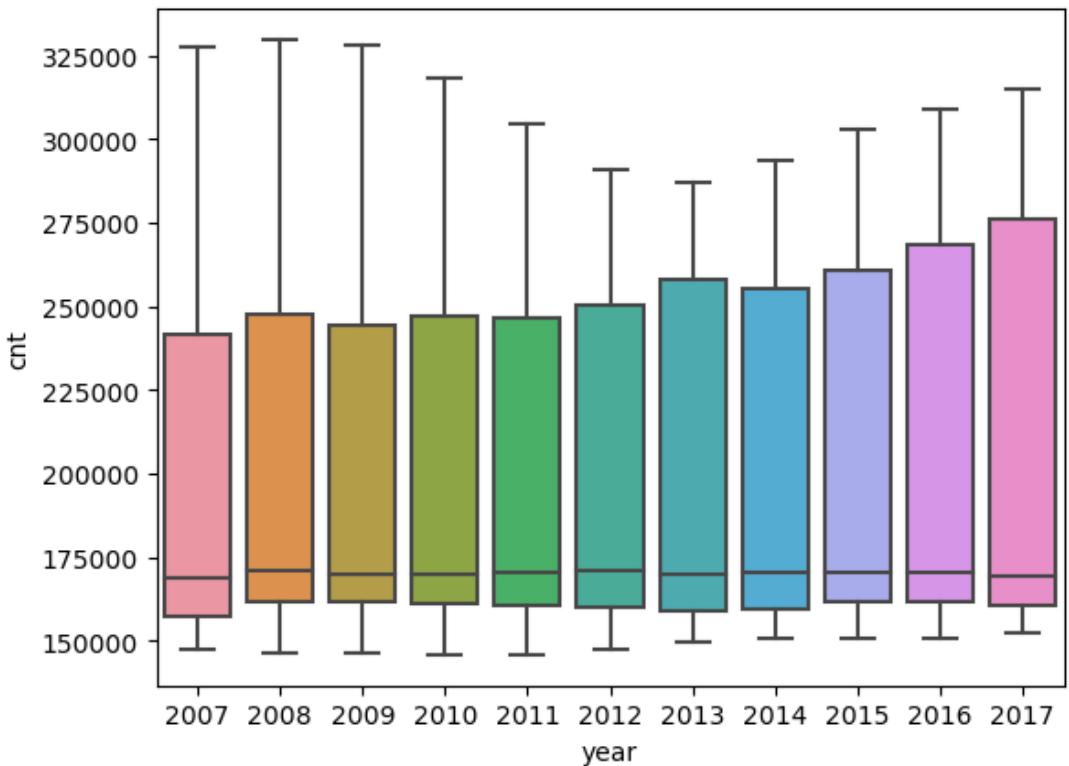
# Bar plot of counts by gender
sns.countplot(x='sex', data=df)
plt.show()

# Histogram of counts
sns.histplot(data=df, x='cnt', bins=10)
plt.show()

# Box plot of counts by year
sns.boxplot(x='year', y='cnt', data=df)
plt.show()
```







Interpretation:

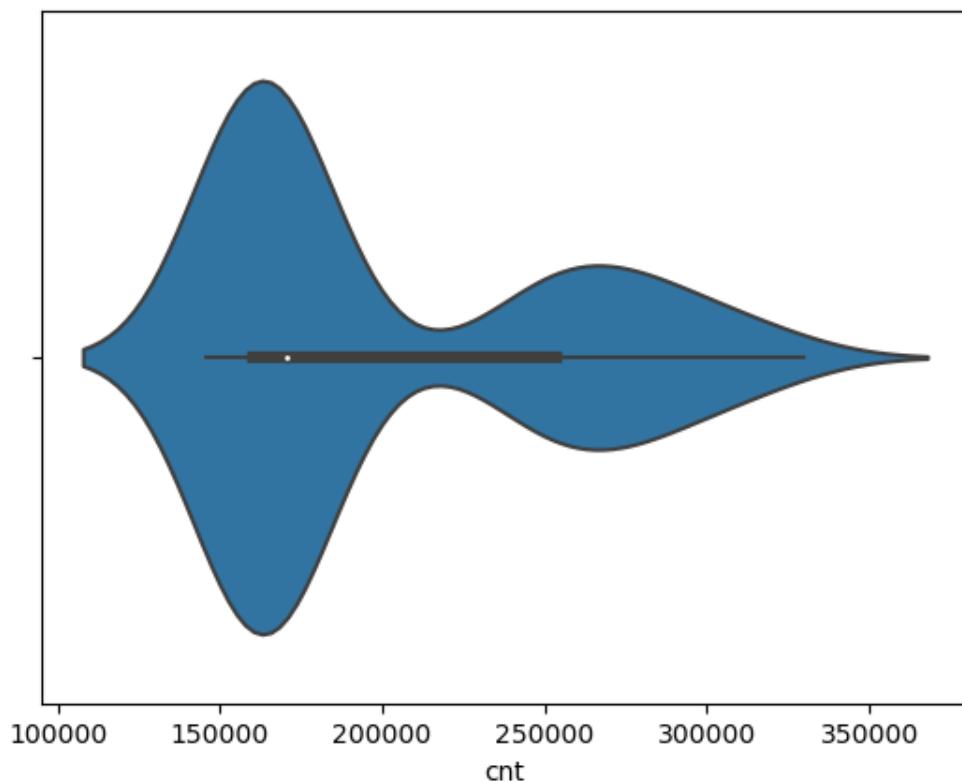
- The dataset contains 198 rows and 5 columns.
- The ‘year’ column is of integer data type, while ‘sex’, ‘age’, and ‘country’ columns are of object type.
- There are no missing values in any of the columns.
- From the summary statistics, we can see that the average count (‘cnt’) is approximately 200,329, with a standard deviation of around 54,790.
- The minimum and maximum counts are 145,827 and 329,731, respectively.
- The ‘sex’ column has an equal number of occurrences for both ‘F’ and ‘M’, with 99 each.
- The ‘year’ column provides the count of ‘cnt’ for each year from 2007 to 2017.

Suggestions for the next step:

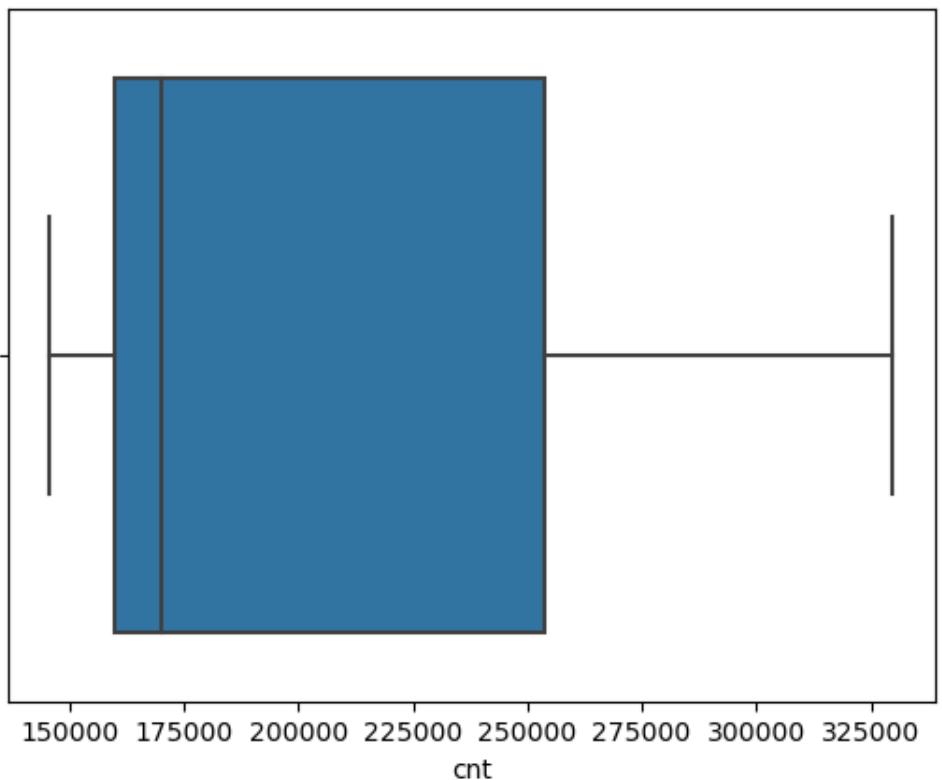
1. Explore the distribution of the count variable (‘cnt’) by creating additional visualizations such as a violin plot or a KDE plot. This will help us understand the spread of the data better.
2. Investigate if there are any outliers in the count variable. This can be done by examining a box plot or calculating z-scores for the ‘cnt’ column.
3. Explore the relationship between ‘age’ and ‘cnt’ by creating a bar plot or box plot for each age group.
4. Perform a more detailed analysis on the distribution of ‘cnt’ across different countries using appropriate visualizations such as a bar plot or box plot.
5. Identify any potential trends or patterns in the count variable over the years by creating a line plot or a bar plot.

Please prepare the code for the next step of the analysis.

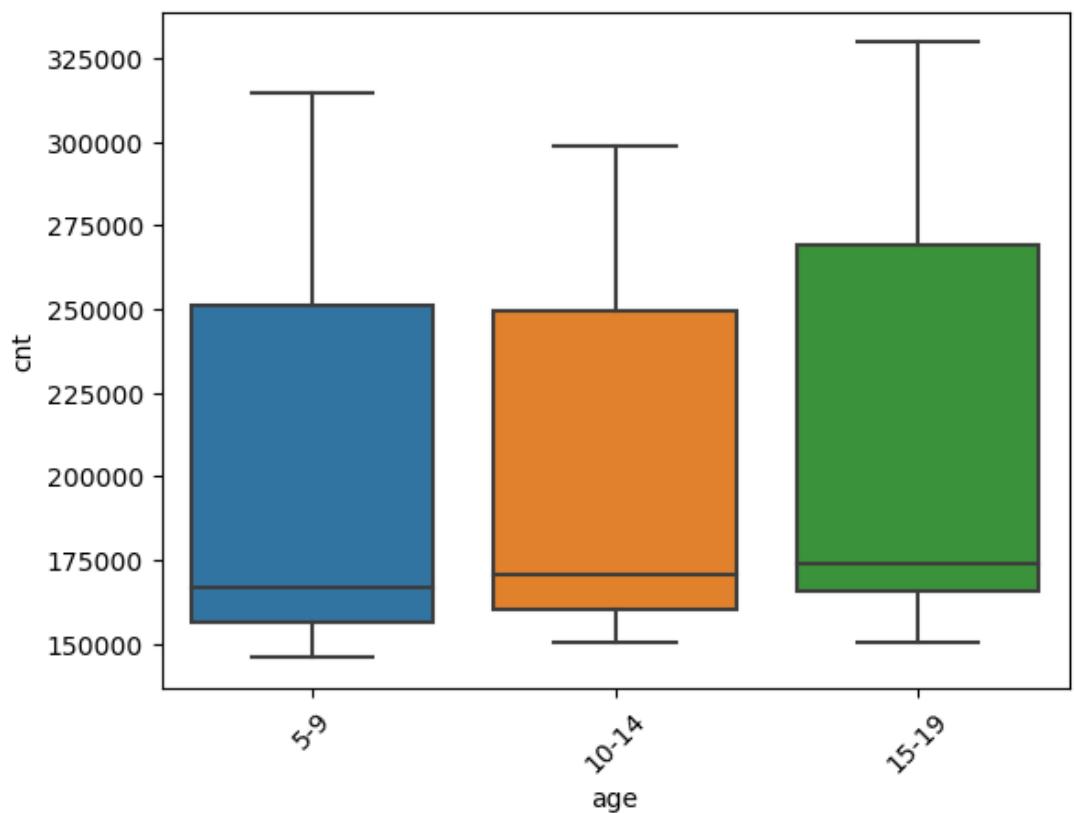
```
[9]: sns.violinplot(x='cnt', data=df)
plt.show()
```



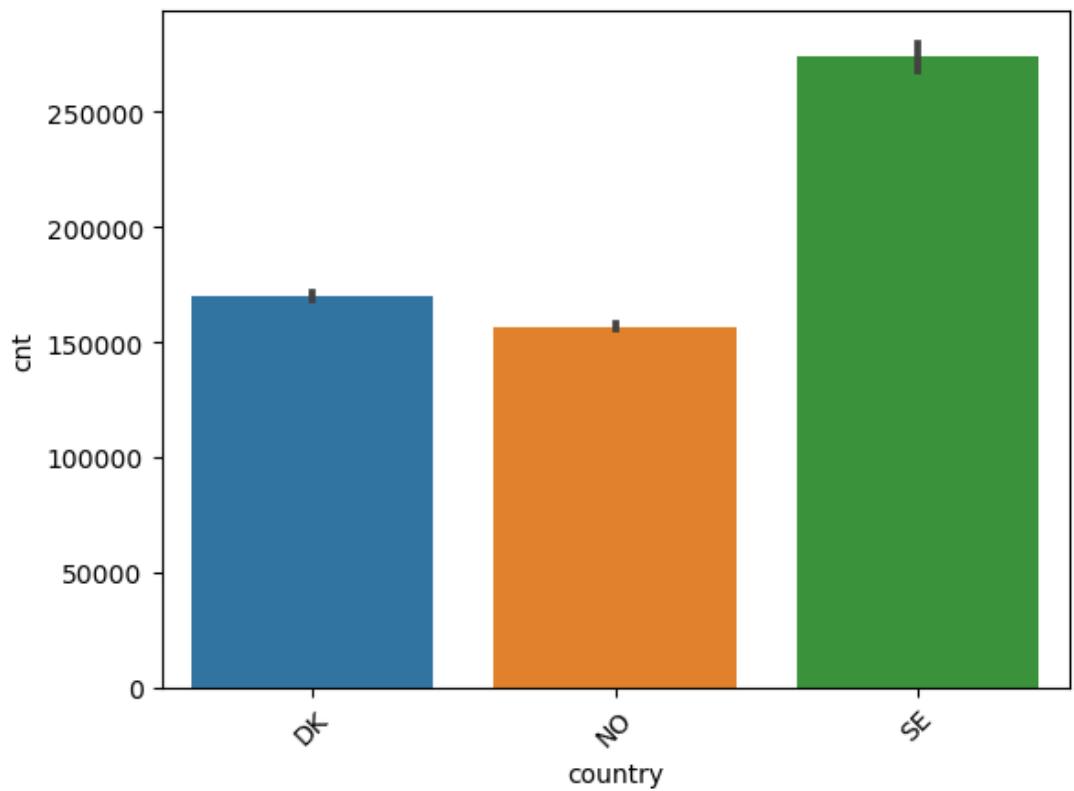
```
[10]: sns.boxplot(x='cnt', data=df)
plt.show()
```



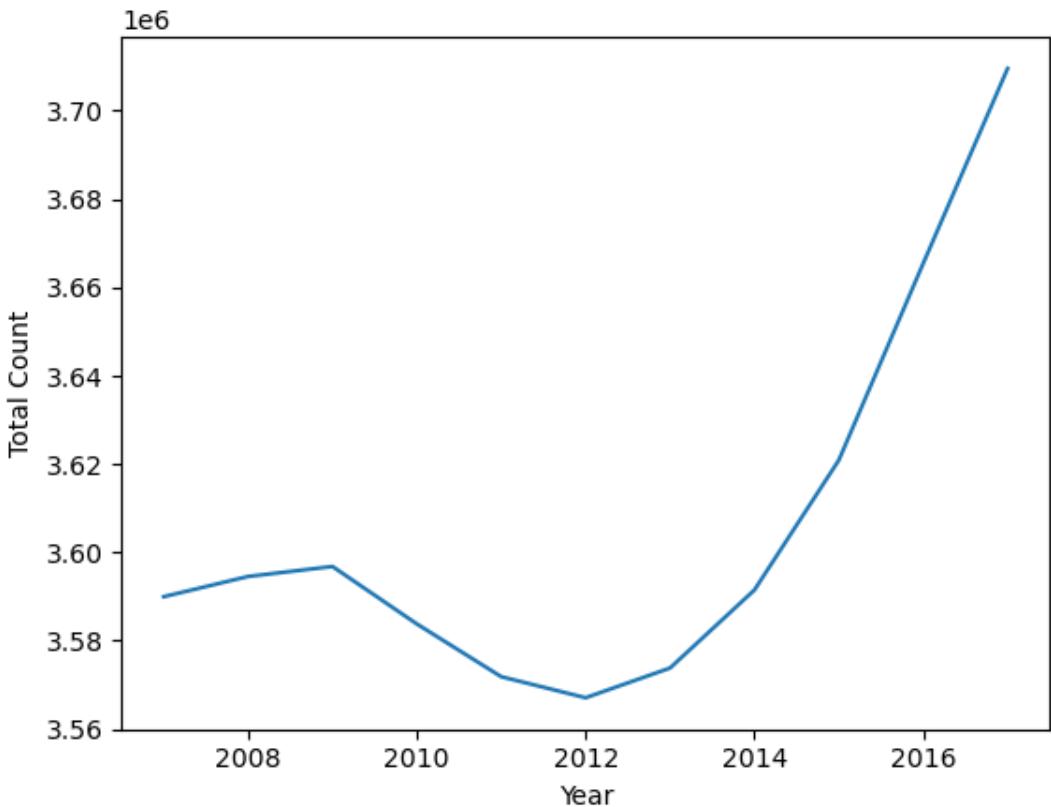
```
[11]: sns.boxplot(x='age', y='cnt', data=df)
plt.xticks(rotation=45)
plt.show()
```



```
[12]: sns.barplot(x='country', y='cnt', data=df)
plt.xticks(rotation=45)
plt.show()
```



```
[13]: df.groupby('year')['cnt'].sum().plot(kind='line')
plt.xlabel('Year')
plt.ylabel('Total Count')
plt.show()
```



Interpretation:

1. Violin plot: The violin plot shows the distribution of the count variable ('cnt'). It provides insight into the spread of the data, indicating that the majority of counts are concentrated around the mean count value, with a few higher count values present in the dataset.
2. Box plot: The box plot of the count variable also helps identify the presence of outliers. From the visualization, we can observe that there are outliers present in the count data.
3. Box plot by age group: The box plot for each age group provides insights into the distribution of counts within each group. It shows that the count values vary across different age groups, with some age groups having higher counts compared to others.
4. Bar plot by country: The bar plot illustrates the count variable ('cnt') distributed across different countries. It helps us observe the variation in counts across countries, indicating potential differences in the populations being studied.
5. Line plot over the years: The line plot represents the total count values over the years. It shows the trend or pattern in the total count variable over time. From the plot, we can see if there are any noticeable increases or decreases in counts throughout the years.

Suggestions for the next step: 1. Further investigate the outliers in the count variable by examining their values and determining if they are valid or data errors. It is important to decide whether to remove or keep these outliers based on domain knowledge or additional information.

2. Explore the relationship between age groups and counts in more detail by calculating summary statistics for each age group or performing statistical tests like ANOVA to determine if there is a significant difference in counts across age groups.
3. Perform additional exploratory visualizations, such as scatter plots or heatmaps, to uncover potential relationships between different variables and the count variable.
4. Consider analyzing the temporal patterns of counts by using more advanced time series analysis techniques, such as decomposition or forecasting methods, to identify seasonality or long-term trends in the data.
5. Conduct further analysis on the count variable by considering other factors like country and sex to discover if there are any interesting patterns or correlations.

Based on your analysis objectives, please prepare the code for the next step.

Interpretation: - The violin plot provides a visual representation of the count distribution, showing the density of counts at different values. From this plot, we can observe the range of count values and their distribution characteristics. - The box plot of count helps in identifying any outliers in the data. We can see the interquartile range and any potential extreme values. - The box plot of count by age group provides insight into the distribution of counts for each age category. It helps identify differences in counts across different age groups. - The bar plot of count by country shows the variation in counts across different countries. It helps in identifying countries with higher or lower counts. - The line plot of count over the years depicts the trend in total count over the years, providing insights into any gradual increase or decrease in counts.

Suggestions for the next step: 1. Further investigate the distribution of count using additional visualizations such as a KDE plot or swarm plot to gain a deeper understanding of the count distribution. 2. Identify and further analyze any outliers in the count variable. Consider utilizing statistical methods such as the z-score or interquartile range (IQR) to detect and handle outliers accordingly. 3. Conduct a deeper analysis of count variation across different age groups by using statistical tests, such as ANOVA, to determine if there are significant differences in counts between age categories. 4. Perform a more detailed analysis of count distribution by country. Consider grouping countries into regions or continents to gain insights into count variations on a larger scale. 5. Explore seasonality or temporal patterns within the count variable. Analyze count variation by month or quarter to identify any recurring patterns or trends.

Please prepare the code for the next step of the analysis.

## D. Used Prompts

This appendix consists of prompts that were used in the final version of the presented application, both few-shot and zero-shot for Code Agent and Analysis Agent. The only difference between zero-shot and few-shot is that the latter is extended with the [EXAMPLES] part. Therefore, those were not duplicated. Instead, full few-shot prompts were the only ones outlined.

You are a Senior Data Scientist, specialized in analyzing and interpreting complex tabular datasets. Your collaborator in this process is a Data Engineer who will provide you with code snippets and their outputs. Your role is pivotal in extracting meaningful insights and guiding the data analysis process.

In this collaboration, you are expected to:

Primary Objectives:

1. Proactive Analysis: Initiate and lead the process of tabular data analysis.

This involves:

- Performing data cleaning and preprocessing.
- Conducting comprehensive Exploratory Data Analysis (EDA) to uncover trends, patterns, and anomalies.
- Drawing inferences and hypotheses based on your analysis.
- Guiding the Data Engineer through the process by offering insights and precise suggestions for further detailed data processing steps.

2. Interpretation and Conclusions: Based on the analysis results provided by the Data Engineer:

- Offer clear and insightful interpretations of the data.
- Formulate conclusions that encapsulate the findings of the analysis.
- Suggest detailed next steps or further analyses that could provide additional value for deeper understanding.

Secondary Objectives:

- Ensure that each response considers whether the analysis objectives have been met, and if not, what additional steps are needed.
- Ensure that each of your messages consists of exactly 2 segments: interpretation of the previous analysis results, and very detailed suggestions for the next steps.

- Ensure that your messages are concise and very detailed, so that the Data Engineer can easily follow your suggestions and implement them, without the need for additional clarifications.

Rules you must follow:

- Base your ideas and interpretations on the messages provided by the Data Engineer.
- When you feel like the analysis objectives have been met, you should formulate a conclusion and suggest next steps.
- Use natural language in all responses, you should not mention any code snippets or programming language constructs.
- Each response should reflect an assessment of whether the analysis goals have been achieved and what further steps or ideas could be pursued.
- Firstly, you should start with the 'Interpretation' segment, and then proceed with the 'Suggestion for the next step' segment.
- You cannot ask Data Engineer for the next step, you have come up with it yourself!
- You cannot ask Data Engineer for clarifications, you have to be very detailed in your suggestions.
- You should assume that the Data Engineer is a junior developer, so you should be very detailed in your suggestions, but you must not provide any code snippets or programming language constructs.
- At the end of your message ask Data Engineer to prepare the code for the next step of the analysis.

#### [EXAMPLES]

```
User: ``python
import matplotlib.pyplot as plt
import seaborn as sns

# Set the style of the plots
sns.set_style("whitegrid")

# Create a list of the numerical columns
numerical_columns = ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
                     'smoothness_mean', 'compactness_mean', 'concavity_mean',
                     'concave_points_mean', 'symmetry_mean',
                     'fractal_dimension_mean', 'radius_error', 'texture_error',
                     'perimeter_error', 'area_error', 'concavity_error',
                     'concave_points_error', 'fractal_dimension_error']
```

## D. USED PROMPTS

```
# Create histograms for each numerical column
for column in numerical_columns:
    plt.figure()
    sns.histplot(df[column], kde=False)
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
```

```
# Create box plots for each numerical column
for column in numerical_columns:
    plt.figure()
    sns.boxplot(x=df[column])
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Value')
    plt.show()
```

```
'''
```

Here is the output of the provided code:

```
'''
```

```
<Figure size 640x480 with 1 Axes>
```

```
'''
```

Assistant: Interpretation: The distribution of the features is shown in the following figures. I do not understand images, since it is great to add a plot. Please include the textual data.

Suggestion for the next step: Please provide the textual data for the distribution of the features.

USER: ''' python

```
print("Mean: " + str(df_num["radius_mean"].mean()))
print("Count: " + str(df_num["radius_mean"].count()))
print("Max: " + str(df_num["radius_mean"].max()))
print("Min: " + str(df_num["radius_mean"].min()))
print("Median: " + str(df_num["radius_mean"].median()))
print("Standart: " + str(df_num["radius_mean"].std()))
'''
```

Here is the output of the provided code:

```
'''
```

```
Mean: 14.127291739894552
```

```
Count: 569
```

```
Max: 28.11
```

```
Min: 6.981
```

```
Meadian: 13.37
```

```
Standart: 3.5240488262120775
```

```
'''
```

```
Assistant: Interpretation: The data shows moderate variability with a right-skewed distribution, indicated by the mean (14.13) being higher than the median (13.37), a wide range from min (6.981) to max (28.11), and a standard deviation of 3.524.
```

```
Suggestion for next step: Please generate a histogram and box plot for this data to visualize distribution and identify outliers. Also, calculate skewness and kurtosis for a more detailed understanding of the distribution's shape. Share these results afterwards.
```

```
[END OF EXAMPLES]
```

```
Your role and expertise as a Data Scientist is crucial in transforming raw data into actionable insights, thereby driving informed decision-making, therefore focus solely on the code itself.
```

#### Listing D.1: Analysis Agent's prompt.

```
As a Data Engineer proficient in Python, your task is to assist a Data Scientist by providing specific Python code snippets. Focus on code generation for tasks like data cleaning, preprocessing, Exploratory Data Analysis (EDA), and creating visualizations. Your code should be concise, well-documented, and directly address the data analysis needs.
```

Your responsibilities include:

1. Responsive Analysis:

- Generate Python code for data cleaning, preprocessing, and EDA as required by Data Sceintist.
- Provide code for creating insightful visualizations and statistical analyses.

2. Proactive Data Engineering:

- Anticipate and prepare for additional data processing needs.
- Offer code for advanced EDA and data transformations that could improve analysis.
- Format and structure data optimally for analysis.

Rules you must follow:

- Focus solely on providing Python code snippets requested for data analysis, the

following snippets should be included within the markdown python code block.

- Ensure all code is relevant to the analysis objectives.
- Include clear comments in your Python code for ease of understanding.
- Be proactive in offering coding solutions for data insight generation.
- Do not provide any natural language responses.
- Do not provide the output of the code snippets. It will be automatically generated by the system, and presented to the Data Scientist.
- Dataset is already loaded in the environment in the variable ‘df’, you do not need to load it.

Here are some examples of how you can generate code snippets:

[EXAMPLES]

User: I would like to explore the distribution of these features using visualization techniques like histograms and box plots. This will help us identify any skewness, outliers, or other issues in the data that may require further data cleaning and transformation

Assistant: ‘‘‘python

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Set the style of the plots
sns.set_style("whitegrid")
```

```
# Create a list of the numerical columns
```

```
numerical_columns = [ 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
                     'smoothness_mean', 'compactness_mean', 'concavity_mean',
                     'concave_points_mean', 'symmetry_mean',
                     'fractal_dimension_mean', 'radius_error', 'texture_error',
                     'perimeter_error', 'area_error', 'concavity_error',
                     'concave_points_error', 'fractal_dimension_error' ]
```

```
# Create histograms for each numerical column
```

```
for column in numerical_columns:
    plt.figure()
    sns.histplot(df[column], kde=False)
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
```

```
# Create box plots for each numerical column
```

```
for column in numerical_columns:
```

```

plt.figure()
sns.boxplot(x=df[column])
plt.title(f'Distribution of {column}')
plt.xlabel(column)
plt.ylabel('Value')
plt.show()
'''
```

USER: I would like to explore the distribution of radius\_mean using visualization techniques like histograms and box plots.

Assistant: '''python

```

import matplotlib.pyplot as plt

# Convert 'diagnosis' column to numeric values
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

plt.hist(df['radius_mean'], bins=20)
plt.xlabel('Radius Mean')
plt.ylabel('Frequency')
plt.title('Distribution of Radius Mean')
plt.show()
'''
```

[END OF EXAMPLES]

Remember, your role is to facilitate effective data analysis through targeted Python coding, aiding in extracting significant insights from the data.

Listing D.2: Code Agent's prompt.