To solve the problem, I have implemented two methods. The first method is using Jupyter Notebook, and the second method is using a `.py` file. In the second method, I am trying to implement the code using Streamlit. In both methods, I use Langchain and Llama 3 on the Groq cloud platform. The large language models that can be used are "llama-3.1-70b-versatile," "mixtral-8x7b-32768," and "gemma-7b-it.".

Method-1

Step-by-Step Documentation

Step 1: Import Necessary Libraries

First, we need to import the required libraries. For this task, we need the `pandas` library to handle the DataFrame and the `os` library to set environment variables. Additionally, the `create_pandas_dataframe_agent` function from the `langchain_experimental.agents.agent_toolkits` module and the `ChatGroq` class from the `langchain_groq` module are used for interacting with the Groq API.

Step 2: Load the Data

Next, we load the data from a CSV file into a pandas DataFrame. Ensure that the path to the CSV file is correct.

Step 3: Set the Groq API Key

Set the Groq API key using the `os` library. This key is necessary to authenticate requests to the Groq API.

Step 4: Initialize the ChatGroq Model

Initialize the `ChatGroq` model. The `model` parameter specifies the model version, and the `temperature` parameter controls the randomness of the model's responses (set to 0 for deterministic output).

Step 5: Create the DataFrame Agent

Create a pandas DataFrame agent using the `create_pandas_dataframe_agent` function. The agent will use the previously initialized `ChatGroq` model and the DataFrame we loaded. The `verbose` parameter is set to `True` to enable detailed logging, and `allow_dangerous_code` is set to `True` to permit the execution of potentially risky code.

Step 6: Invoke the Agent to chat with the csv file

Invoke the agent with a specific task.

Method-2

This documentation describes the code for a Streamlit application designed to analyze CSV data using Groq and LangChain. The application allows users to upload a CSV file, ask questions about the data, and visualize the results.

Importing Required Libraries

The application starts by importing the necessary libraries:

python
Copy code
```
import streamlit as st
import pandas as pd
import os
import matplotlib.pyplot as plt
from langchain_experimental.agents.agent_toolkits import
create_csv_agent
from langchain_groq import ChatGroq
from io import StringIO
import sys
import re
from Eda_plot import *
```

- `streamlit` for creating the web application.
- `pandas` for data manipulation.
- `os` for handling file operations.
- `matplotlib.pyplot` for plotting data.
- `langchain_experimental.agents.agent_toolkits` for creating a CSV agent.
- `langchain_groq` for interacting with the Groq API.
- `StringIO` and `sys` for capturing and handling output.

- `re` for regular expression operations.
- `Eda_plot` for performing exploratory data analysis (EDA).

Streamlit Configuration and Styling

The application sets the page configuration and adds custom styling:

python
Copy code
```
st.set_page_config(page_title="CSV Data Analyzer",
layout="wide")

st.markdown("""
    <style>
    .stApp {
        max-width: 1200px;
        margin: 0 auto;
    }
    </style>
    """, unsafe_allow_html=True)
```

- `set_page_config` sets the title and layout of the Streamlit app.
- `markdown` adds custom CSS to style the app.

Output Capture Class

A custom class is created to capture and display output within the Streamlit app:

python
Copy code
```
class StreamlitOutputCapture:
    def __init__(self):
        self.string_io = StringIO()
        self.output_container = st.empty()

    def write(self, data):
        self.string_io.write(data)
        try:
```

```
self.output_container.text(self.string_io.getvalue())
        except:
            pass

    def flush(self):
        pass
```

- `StreamlitOutputCapture` captures and displays output using `StringIO` and Streamlit's `empty` container.

Executing Python Code

A function is defined to extract and execute Python code from responses:

python
Copy code

```
def execute_python_code(response):
    code_blocks = re.finditer(r'Action Input: (.*?)(?:\n|$)',
response, re.DOTALL)

    for match in code_blocks:
        code = match.group(1).strip()
        st.code(code, language='python')
        st.write("Executing code:")

        old_stdout = sys.stdout
        sys.stdout = StringIO()

        try:
            if 'df.plot' in code:
                fig, ax = plt.subplots()
                eval(code)
                st.pyplot(fig)
            elif 'plt.' in code:
                exec(code)
                if plt.get_fignums():
                    st.pyplot(plt.gcf())
            plt.close()
            output = sys.stdout.getvalue()
            if output:
                st.write(output)
        except Exception as e:
            st.error(f"An error occurred: {str(e)}")
        finally:
            sys.stdout = old_stdout
        break
```

- `execute_python_code` finds code blocks, executes them, and handles any plotting commands.

Sidebar Configuration

The sidebar collects user input for configuration:

python
Copy code

```
st.sidebar.title("Configuration")

api_key = st.sidebar.text_input("Enter your Groq API Key:",
type="password")
model_options = ["llama-3.1-70b-versatile",
"mixtral-8x7b-32768", "gemma-7b-it"]
selected_model = st.sidebar.selectbox("Select a model:",
model_options)
```

- `text_input` collects the Groq API key.
- `selectbox` allows the user to select a model.

Main Content

The main section of the app handles file upload, question input, and analysis:

python
Copy code
```
st.title("CSV Data Analyzer")

uploaded_file = st.sidebar.file_uploader("Choose a CSV file",
type="csv")

@st.cache_data
def load_csv(file):
    return pd.read_csv(file)

if uploaded_file:
    df = load_csv(uploaded_file)

    user_question = st.text_area("Enter your question about the
data:")
    analyze_button = st.button("Analyze")

    if api_key and user_question and analyze_button:
        with st.spinner("Processing..."):
            try:
                os.environ["GROQ_API_KEY"] = api_key
                df.to_csv("temp.csv", index=False)

                llm = ChatGroq(model=selected_model,
temperature=0)
                output_capture = StreamlitOutputCapture()
                agent_executer = create_csv_agent(
                    llm,
                    "temp.csv",
                    verbose=True,
                    allow_dangerous_code=True,
                    handle_parsing_errors=True
                )
                original_stdout = sys.stdout
                sys.stdout = output_capture
                response = agent_executer.invoke(user_question)
                sys.stdout = original_stdout

                st.success("Generated Response:")
                st.write(response)

execute_python_code(output_capture.string_io.getvalue())

            except Exception as e:
                st.error(f"An error occurred: {str(e)}")
            finally:
                if os.path.exists("temp.csv"):
                    os.remove("temp.csv")

    st.write("Data Preview and Basic EDA:")
    basic_eda(df)

else:
    st.info("Please upload a CSV file to begin.")
```

- `file_uploader` allows the user to upload a CSV file.
- `load_csv` is a cached function to load the CSV data.
- If a file is uploaded, the user can enter a question and click the analyze button to process the data using the Groq API.
- The response is captured, displayed, and any included Python code is executed.

Footer

A simple footer for branding:

python
Copy code
```
st.markdown("---")
```