An Exploratory Attempt: Evaluating AI Assistants for Data Analysis and Chart and Cross-Table Generation

Introduction:

This document outlines the procedure for using a Python script to test the capabilities of AI assistants (like Google Gemini 1.5 Flash, but the script is model-agnostic) for data analysis, interactive questioning, and chart/cross-table generation. The script reads data from a CSV file, allows you to ask open-ended questions to the AI assistant, and sends prompts to the AI assistant to generate JSON specifications for charts or cross-tables. It then utilizes these JSON objects to create visualizations using Matplotlib (for charts) or generate Excel files (for cross-tables). This "paper boat" approach helps understand how to approach such solutions, assess the current accuracy of AI assistants for such use cases, and determine how to better integrate them into existing workflows and optimize them further. This is an exploratory attempt using a small dataset; therefore, results may not be comprehensive.

Prerequisites:

Python Environment: Ensure you have Python 3.6 or higher installed.

Required Libraries: Install the necessary Python libraries using pip:

```
pip install pandas matplotlib requests openpyxl jsonschema
```

CSV Data File: Prepare a CSV file named "TestData.csv" containing your data. Place this file in the same directory as the Python script. Refer to the script's 'read_csv' function for expected column names. You will need to change the script if the column name does not match.

AI Assistant API Key: Obtain an API key for the AI assistant you intend to use. **Important: The current script has a major security vulnerability. You'll need to replace the "KEY" in the interact_with_ai function with your actual API key. However, storing API keys directly in the code is HIGHLY DISCOURAGED. Use environment variables or a more secure configuration management system for production environments.**

Procedure:

1. **Script Setup:**
   - Copy the following Python code into a file named AI_chitchat_chart_crosstable_generator.py:
2. import pandas as pd
3. import matplotlib.pyplot as plt
4. import requests
5. import json
6. import re
7. import jsonschema
8. from openpyxl import Workbook
9. from openpyxl.utils.dataframe import dataframe_to_rows

```python

# --- Utility Functions ---
def set_chart_title_and_labels(title, subtitle, x_title, y_title, ax):
    """Sets the chart title and axis labels."""
    ax.set_title(f"{title}\n{subtitle}", fontsize=16)
    ax.set_xlabel(x_title, fontsize=12)
    ax.set_ylabel(y_title, fontsize=12)

def apply_common_chart_formatting(ax):
    """Applies common formatting to charts."""
    ax.legend(loc="best", fontsize='medium')
    ax.grid(axis='y', linestyle='--', alpha=0.5)
    ax.tick_params(axis='x', rotation=45)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['bottom'].set_visible(False)
    ax.spines['left'].set_visible(False)

# --- Data Reading Function ---
def read_csv(file_path):
    """Reads a CSV file, handles errors, converts numeric columns, and cleans whitespace."""
    try:
        data_frame = pd.read_csv(file_path, encoding='utf-8-sig', skipinitialspace=True)
    except FileNotFoundError:
        raise FileNotFoundError(f"Error: File not found at path: {file_path}")
    except pd.errors.EmptyDataError:
        raise ValueError("CSV file is empty.")
    except Exception as e:
        raise Exception(f"Error while reading the file: {e}")

    numeric_columns = ['Rating', 'CustomerSatisfactionScore', 'MeanRating', 'StdDevRating',
'VarianceRating', 'tStatistic', 'pValue']
    for col in numeric_columns:
        if col in data_frame.columns:
            data_frame[col] = pd.to_numeric(data_frame[col], errors='coerce')

    if data_frame.isnull().values.any():
        print("Warning: Missing or invalid data detected.")
        print(data_frame.isnull().sum())

    for col in data_frame.select_dtypes(include=['object']).columns:
        data_frame[col] = data_frame[col].str.strip()

    return data_frame

# --- API Interaction Function ---
def interact_with_ai(prompt, data):
    """Interacts with the Google Gemini API. WARNING: API key is hardcoded. Use
environment variables instead."""
    api_key = "KEY"  # Replace with your actual API key
    api_url = "https://generativelanguage.googleapis.com/v1beta/models/gemini-1.5-
flash:generateContent"
    headers = {"Content-Type": "application/json"}
    data_str = data.to_string()
    full_prompt = f"Here's some information about the data:\n{data_str}\n\n{prompt}"
    payload = {"contents": [{"parts": [{"text": full_prompt}]}]}

    try:
        response = requests.post(f"{api_url}?key={api_key}", headers=headers, json=payload)
        response.raise_for_status()
        result = response.json()
        ai_response_text = result['candidates'][0]['content']['parts'][0].get('text', "Error: Unexpected API response format.")
        ai_response_text = ai_response_text.replace("*", "")
    except requests.exceptions.RequestException as e:
        raise Exception(f"Error communicating with the API: {e}") from e
    except json.JSONDecodeError as e:
        raise Exception(f"Error decoding JSON response: {e}") from e
    except (KeyError, IndexError) as e:
         raise ValueError("Error: The API response structure is not as expected.") from e


    return ai_response_text, full_prompt

# --- JSON Validation Function ---
def validate_json(data):
    """Validates JSON against a schema."""
    schema = {
        "type": "object",
        "properties": {
            "chart": {
                "type": "object",
                "properties": {
                    "type": {"type": "string"},
                    "title": {"type": "object"},
                    "subtitle": {"type": "object"},
                    "xAxis": {"type": "object"},
                    "yAxis": {"type": "object"},
                    "series": {"type": "array"}
                },
                "required": ["type", "title", "subtitle", "xAxis", "yAxis", "series"]
            }
        },
        "required": ["chart"]
    }
    try:
        jsonschema.validate(instance=data, schema=schema)
        return True, data
    except jsonschema.exceptions.ValidationError as e:
        print(f"JSON validation error: {e}")
        return False, None

# --- Chart Generation Functions ---
def generate_column_chart(chart_data):
    """Generates a column chart."""
    title = chart_data['title']['text']
    subtitle = chart_data['subtitle']['text']
    x_categories = chart_data['xAxis']['categories']
    x_title = chart_data['xAxis']['title']['text']
    y_title = chart_data['yAxis']['title']['text']
    series = chart_data['series']

    fig, ax = plt.subplots(figsize=(10, 6))
    df = pd.DataFrame({serie['name']: serie['data'] for serie in series}, index=x_categories)
    df.plot(kind='bar', stacked=False, rot=45, ax=ax)

    for p in ax.patches:
        ax.annotate(str(p.get_height()), (p.get_x() * 1.005, p.get_height() * 1.005), fontsize=8)

    set_chart_title_and_labels(title, subtitle, x_title, y_title, ax)
    apply_common_chart_formatting(ax)
    plt.tight_layout(pad=2)

    return fig

def generate_pie_chart(chart_data):
    """Generates a pie chart."""
    title = chart_data['title']['text']
    subtitle = chart_data['subtitle']['text']
    x_categories = chart_data['xAxis']['categories']
    x_title = chart_data['xAxis']['title']['text']
    y_title = chart_data['yAxis']['title']['text']
    series = chart_data['series']

    fig, ax = plt.subplots(figsize=(8, 8))
    for i, serie in enumerate(series):
        colors = [plt.cm.tab20(j / len(serie['data'])) for j in range(len(serie['data']))]
        wedges, texts, autotexts = ax.pie(
            serie['data'], labels=x_categories, autopct='%1.1f%%',
            startangle=90, colors=colors, explode=[0.01] * len(serie['data']],
            textprops=dict(color='w')
        )
        plt.set_p(autotexts, size=10, weight="bold")
        ax.set_title(serie['name'])
        ax.axis('equal')

    set_chart_title_and_labels(title, subtitle, x_title, y_title, ax)
    apply_common_chart_formatting(ax)
    plt.tight_layout(pad=2)

    return fig

def generate_line_chart(chart_data):
    """Generates a line chart."""
    title = chart_data['title']['text']
    subtitle = chart_data['subtitle']['text']
    x_categories = chart_data['xAxis']['categories']
    x_title = chart_data['xAxis']['title']['text']
    y_title = chart_data['yAxis']['title']['text']
    series = chart_data['series']

    fig, ax = plt.subplots(figsize=(10, 6))
    color_palette = plt.cm.viridis(range(len(series)))
    for i, serie in enumerate(series):
        color = color_palette[i]
        ax.plot(x_categories, serie['data'], label=serie['name'], marker='o', linestyle='-', color=color, linewidth=2, markersize=5)

    set_chart_title_and_labels(title, subtitle, x_title, y_title, ax)
    apply_common_chart_formatting(ax)
    plt.tight_layout(pad=2)

    return fig

def generate_scatter_chart(chart_data):
    """Generates a scatter chart."""
    title = chart_data['title']['text']
    subtitle = chart_data['subtitle']['text']
    x_title = chart_data['xAxis']['title']['text']
    y_title = chart_data['yAxis']['title']['text']
    series = chart_data['series']

    fig, ax = plt.subplots(figsize=(10, 6))
    color_palette = plt.cm.viridis(range(len(series)))
    for i, serie in enumerate(series):
        points = serie.get('data', [])
        x_vals, y_vals = zip(*points) if points else ([], [])
        color = color_palette[i]
        ax.scatter(x_vals, y_vals, label=serie['name'], color=color, s=50, alpha=0.8)

    set_chart_title_and_labels(title, subtitle, x_title, y_title, ax)
    apply_common_chart_formatting(ax)
    plt.tight_layout(pad=2)

    return fig

def generate_histogram_chart(chart_data):
    """Generates a histogram chart."""
    title = chart_data['title']['text']
    subtitle = chart_data['subtitle']['text']
    x_title = chart_data['xAxis']['title']['text']
    y_title = chart_data['yAxis']['title']['text']
    series = chart_data['series']

    fig, ax = plt.subplots(figsize=(10, 6))

    for i, serie in enumerate(series):
        ax.hist(serie['data'], bins=10, alpha=0.7, label=serie['name'], edgecolor='black')

    set_chart_title_and_labels(title, subtitle, x_title, y_title, ax)
    apply_common_chart_formatting(ax)
    plt.tight_layout(pad=2)

    return fig

```

```python
def generate_cross_table(chart_data):
    """Generates a cross-table in Excel."""
    title = chart_data['title']['text']
    subtitle = chart_data['subtitle']['text']
    x_categories = chart_data['xAxis']['categories']
    series = chart_data['series']
    try:
        # Create a Pandas DataFrame
        data = {}
        for serie in series:
            data[serie['name']] = serie['data']

        df = pd.DataFrame(data, index=x_categories)


        new_index = []
        for cat in x_categories:
            parts = cat.split('-')

            new_index.append(tuple(parts))
        num_levels = len(new_index[0])
        index_names = [f'Level {i+1}' for i in range(num_levels)]

        df.index = pd.MultiIndex.from_tuples(new_index, names=index_names) # index names will now be the columns names


        column_names = []
        for serie in series:
            column_names.append(serie['name'])

        new_columns = []
        for name in column_names:
            if '-' in name:
                new_columns.append(tuple(name.split('-'))) #splits the data to create columns names
            else:
                new_columns.append(name)

        df.columns = pd.MultiIndex.from_tuples([new_columns], names=['ColumnNames']) if all(isinstance(col, tuple) for col in new_columns) else new_columns


        # Create an Excel workbook and sheet
        wb = Workbook()
        ws = wb.active

        # Add the DataFrame to the worksheet
        for r in dataframe_to_rows(df, index=True, header=True):
            ws.append(r)

        # Save the Excel file
        filename = "cross_table.xlsx"
        wb.save(filename)
        print(f"Cross-table generated and saved to {filename}")

    except (TypeError, KeyError, AttributeError) as e:
        print(f"Error generating cross-table: {e}")
    return

def generate_chart_from_json(json_str):
    """Generates charts based on JSON specification and returns the figure."""
    try:
        match = re.search(r'\{.*\}', json_str, re.DOTALL)
        if match:
            json_data = json.loads(match.group(0))
        else:
            print("Error: Could not find valid JSON in the response.")
            return None

        if not isinstance(json_data, dict) or "chart" not in json_data:
            print("Error: JSON structure is invalid.")
            return None

        chart = json_data['chart']
        chart_type = chart.get('type', 'column').lower()

        if chart_type == 'cross-table':
            generate_cross_table(chart)
            return None
        elif chart_type == 'column' or chart_type == 'bar':
            fig = generate_column_chart(chart)
        elif chart_type == 'pie':
            fig = generate_pie_chart(chart)
        elif chart_type == 'line':
            fig = generate_line_chart(chart)
        elif chart_type == 'scatter':
            fig = generate_scatter_chart(chart)
        elif chart_type == 'histogram':
            fig = generate_histogram_chart(chart)
        else:
            print(f"Unsupported chart type: {chart_type}")
            return None

        return fig

    except (json.JSONDecodeError, TypeError, KeyError) as e:
        print(f"Error processing the JSON input: {e}")
        print(json_str)
        return None


# --- Menu Display Function ---
def display_menu():
    menu_width = 40
    menu_title = " MAIN MENU "
    print("\n" + "=" * menu_width)
    print(f"{menu_title:^{menu_width}}")
    print("=" * menu_width)
    print("\n1.  Ask an open question about the data (no chart option)")
    print("2.  Select a chart Or cross-table option from predefined prompts")
    print("3.  Exit\n")
    print("=" * menu_width)

# --- Main Function ---
def main():
    """Main function to run the program."""
    file_path = "TestData.csv"
    try:
        data = read_csv(file_path)
    except FileNotFoundError as e:
        print(e)
        return
    except ValueError as e:
        print(e)
        return

    chart_prompts = [
    "Compare the customer satisfaction scores across different products using a column chart.",
    "Compare the purchase frequencies between premium and budget products using a column chart.",
    "Show the count of ratings (1 to 5) by gender using a grouped column chart.",
    "Show the distribution of purchase contexts (e.g., online purchase, in-store purchase) using a pie chart.",
    "Show the gender distribution among customers for premium product using a pie chart.",
    "Show the ratings provided by customers in different income levels using a column chart.",
    "Show the variation in purchase frequencies across regions using a column chart.",
    "Show the relationship between customer satisfaction scores and ratings for each product using a scatter plot.",
    "Show the distribution of income levels across different price categories using a grouped column chart.",
    "Show the mean ratings for products categorized by age groups using a line chart.",
    "Show the customer satisfaction scores for different purchase contexts using a column chart.",
    'Generate a cross-table with rows nested hierarchically by \'Product\', \'Region\', \'PurchaseFrequency\', and \'AgeGroup\'.\nDisplay \'CustomerSatisfactionScore\' in columns, ensuring clarity and highlighting trends explicitly present in the data.'
]


    while True:
        display_menu()
        try:
            choice = input("Please enter your choice (1-3): ").strip()
            if choice in ["1", "2", "3"]:
```

```
            print(f"\nYou selected option {choice}.")
            if choice == "3":
                print("Exiting program. Have a great day!\n")
                break
        else:
            print("\n Invalid input. Please select a valid option (1-3).\n")
    except Exception as e:
        print(f"\n An error occurred: {e}\n")

    if choice == '1':
        open_question = input("Ask your question about the data: ").strip()
        try:
            ai_response, _ = interact_with_ai(open_question, data)
        except ValueError as e:
            print(e)
            continue
        except Exception as e:
            print(f"Error during AI interaction: {e}")
            continue

        if ai_response is None:
            print("Error: Failed to get a response from the AI assistant.")
            continue
        print("AI Response:")
        wrapped_response = ""
        line_length = 80
        words = ai_response.split()
        current_line = ""
        for word in words:
            if len(current_line + word) + 1 <= line_length:
                current_line += word + " "
            else:
                wrapped_response += current_line.rstrip() + "\n"
                current_line = word + " "
        wrapped_response += current_line.rstrip()
        print(wrapped_response)



    elif choice == '2':
        print("\nSelect a chart prompt:")
        for i, prompt in enumerate(chart_prompts, start=1):
            print(f"{i}. {prompt}")

        try:
            chart_choice = int(input("Enter the prompt number: ").strip()) - 1
            if chart_choice < 0 or chart_choice >= len(chart_prompts):
                print("Invalid prompt selection. Please try again.")
                continue

            chart_prompt = chart_prompts[chart_choice]
            print(f"Selected Chart Prompt: {chart_prompt}")

            clear_instruction = (
f"You are a JSON data provider for chart generation or cross-table data. "
f"Given the prompt: '{chart_prompt}', "
f"your task is to generate a JSON object that strictly adheres to the following requirements:\n\n"
f"1. Strict JSON Structure:\n"
f"   The JSON must follow one of these formats exactly, depending on the requested chart type:\n"
f"   {{\n"
f"       'chart': {{\n"
f"      'type': '[chart_type]',\n"
f"      'title': {{'text': '[chart_title]'}},\n"
f"      'subtitle': {{'text': '[chart_subtitle]'}},\n"
f"      'xAxis': {{\n"
f"        'categories': [categories_list],\n"
f"        'title': {{'text': '[x_axis_title]'}}\n"
f"      }},\n"
f"      'yAxis': {{\n"
f"        'title': {{'text': '[y_axis_title]'}}\n"
f"      }},\n"
f"      'series': [\n"
f"        {{'name': '[series_1_name]', 'data': [series_1_data_list]}},\n"
f"        {{'name': '[series_2_name]', 'data': [series_2_data_list]}},\n"
f"      ]\n"
f"    }}\n"
f"  }}\n\n"
f"  B. For Cross-Table Generation ('cross-table'):\n"
f"  {{\n"
f"    'chart': {{\n"
f"      'type': 'cross-table',\n"
f"      'title': {{'text': '[table_title]'}},\n"
f"      'subtitle': {{'text': '[table_subtitle]'}},\n"
f"      'xAxis': {{\n"
f"        'categories': [row_categories_list],\n"
f"        'title': {{'text': '[row_axis_title]'}}\n"
f"      }},\n"
f"      'series': [\n"
f"        {{'name': '[column_1_name]', 'data': [column_1_data_list]}},\n"
f"        {{'name': '[column_2_name]', 'data': [column_2_data_list]}},\n"
f"      ]\n"
f"    }}\n"
```

```
        f"   }}\n\n"
        f"2. Prompt Alignment:\n"
        f"   - Ensure the JSON is tailored to the context of the prompt.\n"
        f"   - For pie charts, ensure all 'data' values are valid numbers (e.g., integers or floats).\n"
        f"   - Use logical and complete placeholders where details are missing.\n\n"
        f"3. Validation and Accuracy:\n"
        f"   - The JSON must be syntactically correct.\n"
        f"   - Validate that 'data' fields in the 'series' array contain only a list of numeric values.\n"
        f"   - Ensure the number of 'categories' matches the number of 'data' points for pie charts and cross-tables.\n\n"
        f"4. Supported Chart/Table Types:\n"
        f"   - Supported chart types include: column, bar, pie, line, scatter, and histogram.\n"
        f"   - A cross-table can also be generated (type: 'cross-table').\n"
        f" - For cross-tables:\n"
        f" - The 'xAxis' exclusively defines the row categories. The 'categories' list within 'xAxis' contains the labels for each row.\n"
        f" - Each item in 'series' represents a column. Each item must contain 'name' (column header) and 'data' (column values).\n"
        f" - The length of each 'data' list in 'series' must exactly match the length of the 'categories' list in 'xAxis'. This consistency is mandatory.\n"
        f"5. Output Only JSON:\n"
        f"   - Do not include additional text, explanations, or comments. Return only the JSON object.\n"
    )
            try:
                chart_response, _ = interact_with_ai(clear_instruction, data)
            except ValueError as e:
                print(e)
                continue
            except Exception as e:
                print(f"Error during AI interaction: {e}")
```

```
                    continue

                if chart_response is None:
                    print("Error: Failed to get a chart response from the AI assistant.")
                    continue

                fig = generate_chart_from_json(chart_response)

                if fig:
                    plt.tight_layout(pad=2)
                    plt.show()

        except ValueError:
            print("Invalid input. Please enter a number corresponding to a prompt.")

        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

API Key Replacement:

- Open the AI_chitchat_chart_crosstable_generator.py file in a text editor.
- Locate the line api_key = "KEY" within the interact_with_ai function.
- **Replace "KEY" with your actual AI assistant's API key.** Remember the Security WARNING above!

Execution:

- **General Execution:**
    - Open a terminal or command prompt.
    - Navigate to the directory where you saved AI_chitchat_chart_crosstable_generator.py and placed TestData.csv.
    - Run the script using:

- python AI_chitchat_chart_crosstable_generator.py

Google Colab:

1. Upload both AI_chitchat_chart_crosstable_generator.py and TestData.csv to your Google Colab environment.
2. Open a new code cell in Colab.
3. Run the following commands in the cell to install dependencies and execute the script:
4. !pip install pandas matplotlib requests openpyxl jsonschema
5. !python AI_chitchat_chart_crosstable_generator.py

6. Follow the on-screen prompts. Any charts will be displayed within the Colab notebook. Excel files will be saved to the Colab environment; download them from the files pane.

Spyder IDE:

1. Open AI_chitchat_chart_crosstable_generator.py in Spyder.
2. Ensure that your working directory in Spyder is set to the directory where AI_chitchat_chart_crosstable_generator.py and TestData.csv are located. You can usually set this in the top right corner of the IDE or using the "os" module in python:
3. import os
4. os.chdir("your/path/to/the/directory")

5. Run the script within Spyder.
6. Charts will be displayed in Spyder's plot pane. Excel files will be saved to the working directory.

Analyzing the Results & Integration Strategy:

1. **Evaluate AI Responses:** For option 1, critically assess the accuracy, relevance, and completeness of the AI-generated answers to your questions. Note any misunderstandings or inaccuracies.
2. **Analyze Visualizations:** For option 2, examine the generated charts for visual insights into your data. Verify if the chart type, axes labels, and data representation align with your expectations and the prompt's intent.
3. **Inspect Cross-Tables:** Open the "cross_table.xlsx" file to view the generated cross-table and analyze the tabular data. Check if the table structure, row and column labels, and data values are correct and meaningful.
4. **Assess JSON Generation:** Examine the Python console output for any JSON validation errors. If errors occur, it indicates that the AI is not reliably generating correct JSON specifications.

5. **Integration Strategy:** Consider how the AI assistant could be integrated into existing data analysis workflows. Could it automate report generation, provide quick answers to ad-hoc questions, or suggest relevant visualizations? Identify areas where the AI excels and areas where human oversight is still necessary.

6. **Optimization Opportunities:** Document any patterns in the AI's mistakes or limitations. This feedback can be used to refine the prompts, improve the data preprocessing steps, or select a different AI model that is better suited for the task. Think about using a validation function. Consider using a more detailed and complete prompt with examples on how the JSON should look.

Important Considerations:

- API Usage Costs: Be aware of the costs associated with using the AI assistant's API. Monitor your API usage to avoid unexpected charges.
- Data Privacy: Ensure that your data does not contain sensitive or confidential information before sending it to the AI assistant.
- Prompt Engineering: The quality of the generated charts and responses depends heavily on the prompts provided to the AI. Experiment with different prompts to achieve the desired results. The clear instruction provided to the AI assistant is crucial for the program to function.
- Error Handling: This script includes basic error handling, but you may need to add more robust error handling for production environments.
- Security: Never hardcode API keys directly in your code. Use environment variables or secure configuration management systems.

This guide provides a structured approach to testing AI assistants for data analysis, interactive questioning, and visualization, using a small dataset for an initial exploration. By following these steps, you can gain valuable insights into the capabilities of these AI assistants, their limitations, and their potential for improving data-driven decision-making within your organization. Remember to document your findings and use them to inform your integration and optimization strategies.