PharmaSUG 2025 - Paper SI-294

A Game Changer for Efficient SAS Programming using ChatGPT

Jyoti (Jo) Agarwal

ABSTRACT

This paper explores the potential of integrating ChatGPT, a natural language processing model developed by OpenAI, into SAS programming within the pharmaceutical industry. Leveraging ChatGPT's capabilities can enhance productivity by assisting with code generation, debugging, and workflow optimization. By demonstrating practical applications and techniques, this paper illustrates how ChatGPT can streamline statistical programming, reduce errors, and empower programmers with innovative tools to tackle common challenges.

DISCLAIMER

The views and opinions expressed in this paper are solely those of the author and do not reflect the views, of the author's employer. Furthermore, the data and content contained in this paper are sourced from OpenAI (external / publicly available sources) and do not incorporate data from Gilead programs.

INTRODUCTION

The pharmaceutical industry is investigating advanced technologies that could optimize operations and improve efficiency. One such technology is generative AI, which includes tools like ChatGPT, Claude AI, GitHub CoPilot, Google Gemini, etc. While this paper highlights the use of ChatGPT as a tool for SAS Programming, the concepts and techniques discussed apply broadly to most generative AI tools. As these technologies continue to evolve and transform programming workflows, they provide the innovative ways to enhance productivity, streamline processes, and tackle coding challenged more efficiently.

SAS programming is critical in data analysis across industries like pharmaceuticals, finance, and healthcare. However, programmers frequently encounter challenges such as complex syntax, debugging difficulties, and maintaining efficient documentation. By providing practical examples and addressing potential pitfalls, this paper serves as a guide for programmers looking to leverage ChatGPT to improve coding efficiency and workflow optimization.

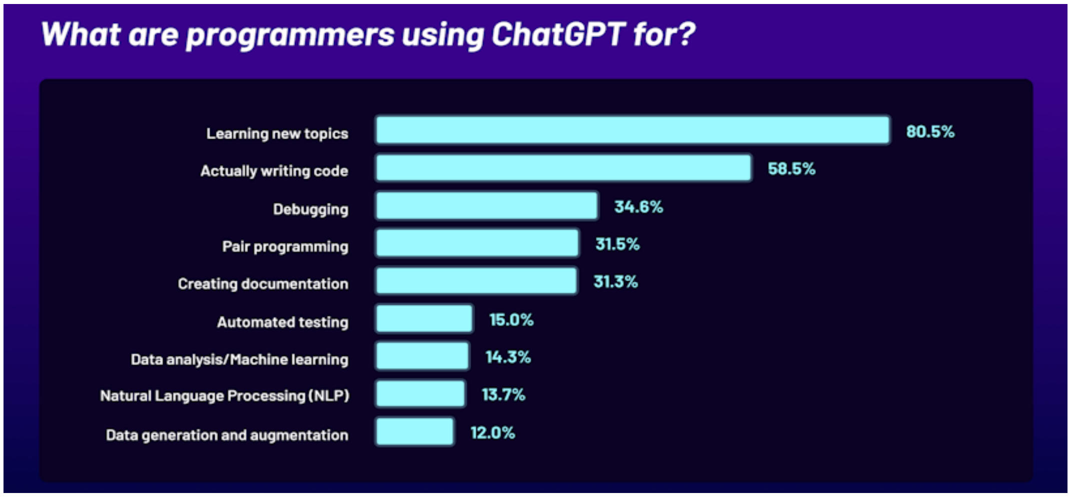
THE ROLE OF CHATGPT IN SAS PROGRAMMING

ChatGPT functions as a versatile assistant for SAS programmers in various ways:

- **Code Generation:** ChatGPT assists in writing SAS code for data manipulation, statistical analysis, and reporting. It generates scripts to merge datasets, compute summary statistics, and create tables efficiently.
- **Debugging:** AI-powered troubleshooting helps identify SAS errors quickly, reducing downtime and improving productivity.
- **Code Optimization and Documentation:** ChatGPT suggests performance improvements and best practices, enhancing code readability and maintainability. It also generates well-structured documentation for future reference.
- **Skill Development:** ChatGPT serves as an educational tool, helping programmers understand SAS concepts and explore other languages like R and Python.
- **Workflow Optimization:** Help automize repetitive tasks such as data validation, formatting outputs, report generation and reducing manual efforts.
- **Data Visualization Support:** Assists in creating SAS graphs and reports, offering suggestions for improving the aesthetics and interpretability of visualization.
- **Multi-Language Adaptation**: Helps SAS programmers translate code into other programming languages and facilitates cross-platform interoperability.

How programmers utilize ChatGPT

ChatGPT plays a significant role in programming by assisting with a wide range of tasks, including:



Source: ZerotoMastery (https://rb.gy/gql9tt)

While ChatGPT is a powerful tool, **programmers must validate its output and apply critical thinking to ensure accuracy**. Combining AI assistance with human expertise yields the best results.

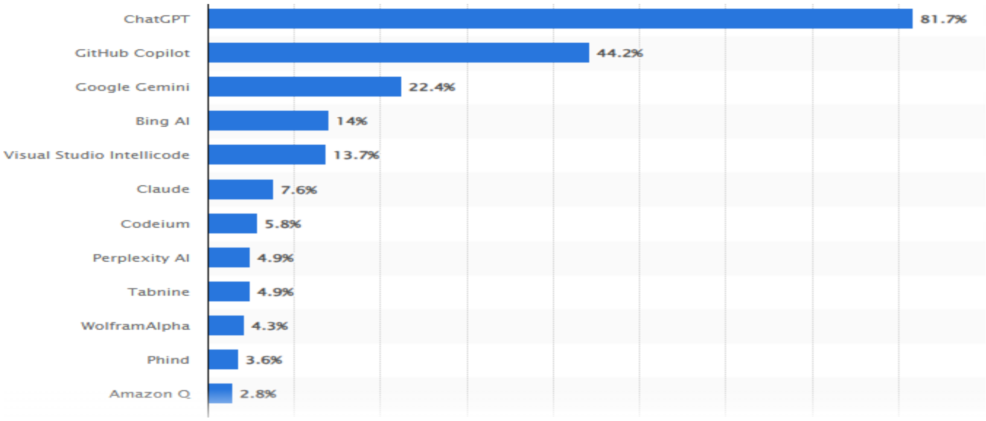
THE GROWING ROLE OF AI IN PROGRAMMING

Programmers increasingly use AI tools to streamline workflows. Popular AI tools include:

- ChatGPT-4.0
- Claude models
- GitHub Copilot
- Codeium
- Amazon Q Developer
- Google Gemini

These tools assist with code generation, debugging, and automation, enabling programmers to enhance productivity through smart code completion and iterative refinements. AI-driven pair programming further improves collaboration and efficiency.

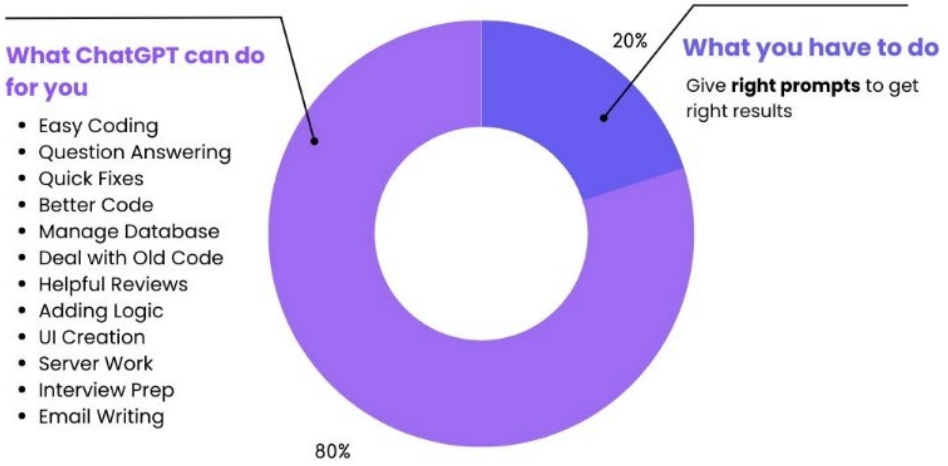
POPULAR AI TOOLS AMONG DEVELOPERS, HIGHLIGHTING THE MOST DESIRED AI ASSISTANTS



Source: © Statista 2025 (https://shorturl.at/qaGB3)

Prompt Engineering: Maximizing ChatGPT’s Potential

The key to effectively utilizing AI tools lies in crafting precise and well-structured prompts. Prompt engineering is the process of designing inputs that yield the most useful and relevant AI responses. The quality of a prompt significantly influences the accuracy and relevance of ChatGPT’s output.



Source: DeveloperUpdates (https://shorturl.at/O4hGI)

The Few Key Strategies for Crafting Effective Prompts:

- **Clarity and Specificity:** Clearly defining the request improves the likelihood of receiving an accurate response. Instead of asking, “How do I write SAS code?” a more specific prompt like “Generate a SAS program that calculates summary statistics using PROC MEANS” will yield more targeted results.
- **Providing Context:** Including relevant details enhances ChatGPT’s ability to generate precise and useful responses. For example, specifying dataset structures or expected outputs allows the AI to tailor its response more effectively.
- **Using Examples:** Providing examples of the desired outcome helps guide ChatGPT’s response. For instance, if requesting optimized code, supplying a sample of the current code ensures the AI refines it appropriately.
- **Iterative Refinement:** ChatGPT-generated responses can be further improved by refining the prompt based on initial outputs. Adjusting phrasing or adding constraints can enhance response quality.
- **Negative Prompting:** Clearly stating what to avoid in a response is useful. For instance, when asking ChatGPT to generate SAS code, specifying “without using PROC SQL” ensures alternative approaches are suggested.
- **Leveraging Known Patterns:** Using common structures or patterns, such as requesting step-by-step explanations, helps improve the clarity and structure of responses.

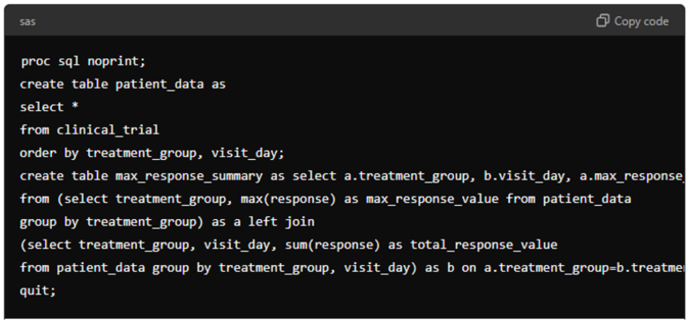
Applying these strategies enables programmers to maximize the benefits of ChatGPT, making AI assistance more accurate, relevant, and efficient.

PRACTICAL APPLICATIONS IN REAL STUDIES

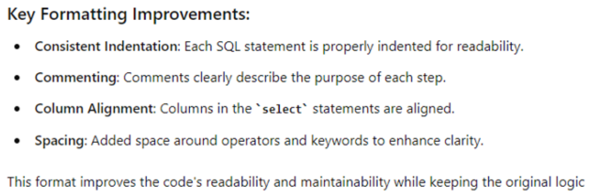
PROMPT 1: REFORMATTING CODE FOR BEST PRACTICES

***Prompt**: "Could you reformat the code to follow good programming practices without adding anything to the code?"*

ChatGPT restructures unformatted code to improve readability, maintainability, and adherence to best practices without altering its functionality. This ensures code consistency and facilitates collaboration among programmers.



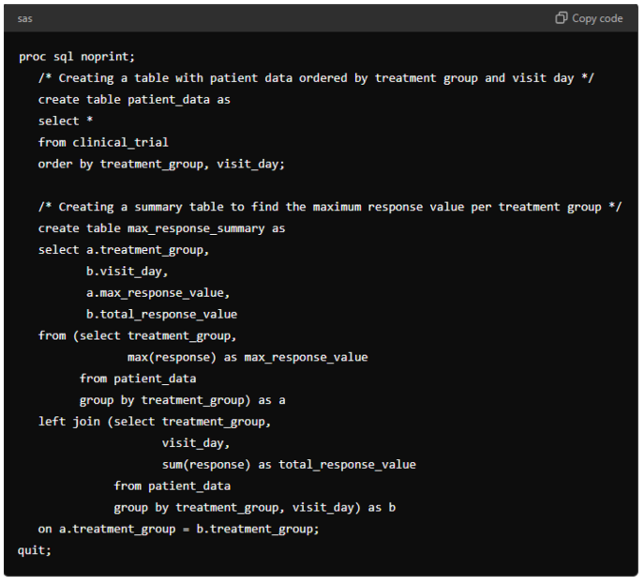
```sas
proc sql noprint;
create table patient_data as
select *
from clinical_trial
order by treatment_group, visit_day;
create table max_response_summary as select a.treatment_group, b.visit_day, a.max_response
from (select treatment_group, max(response) as max_response_value from patient_data
group by treatment_group) as a left join
(select treatment_group, visit_day, sum(response) as total_response_value
from patient_data group by treatment_group, visit_day) as b on a.treatment_group=b.treatme
quit;
```



Key Formatting Improvements:

- **Consistent Indentation:** Each SQL statement is properly indented for readability.
- **Commenting:** Comments clearly describe the purpose of each step.
- **Column Alignment:** Columns in the `` `select` `` statements are aligned.
- **Spacing:** Added space around operators and keywords to enhance clarity.

This format improves the code's readability and maintainability while keeping the original logic

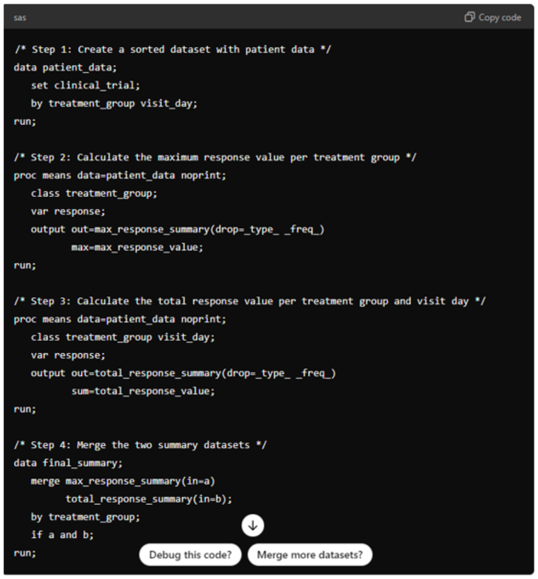


```sas
proc sql noprint;
   /* Creating a table with patient data ordered by treatment group and visit day */
   create table patient_data as
   select *
   from clinical_trial
   order by treatment_group, visit_day;

   /* Creating a summary table to find the maximum response value per treatment group */
   create table max_response_summary as
   select a.treatment_group,
          b.visit_day,
          a.max_response_value,
          b.total_response_value
   from (select treatment_group,
                max(response) as max_response_value
         from patient_data
         group by treatment_group) as a
   left join (select treatment_group,
                     visit_day,
                     sum(response) as total_response_value
              from patient_data
              group by treatment_group, visit_day) as b
   on a.treatment_group = b.treatment_group;
quit;
```

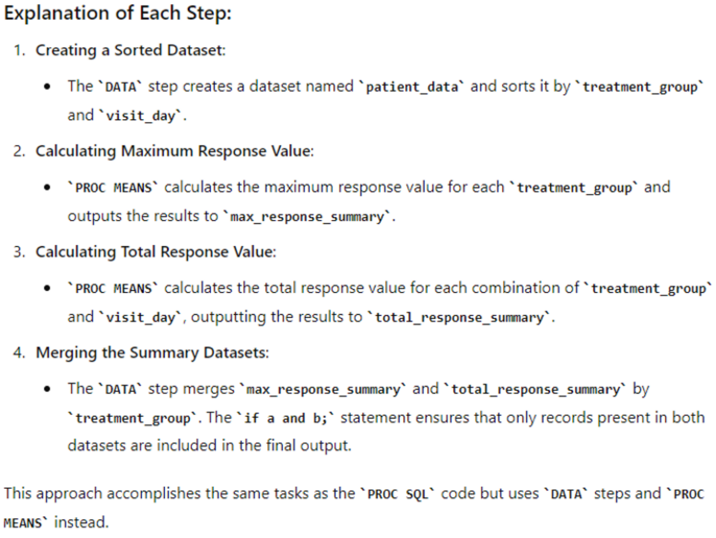
PROMPT 2: ALTERNATIVE APPROACHES WITHOUT PROC SQL

***Prompt:** "Could you do the same task the above code is doing without using PROC SQL?"*

ChatGPT suggests alternative solutions for tasks traditionally performed using PROC SQL, such as leveraging SAS procedures like PROC MEANS or PROC TRANSPOSE. This flexibility allows programmers to explore different coding methodologies based on specific project constraints.



```sas
/* Step 1: Create a sorted dataset with patient data */
data patient_data;
   set clinical_trial;
   by treatment_group visit_day;
run;

/* Step 2: Calculate the maximum response value per treatment group */
proc means data=patient_data noprint;
   class treatment_group;
   var response;
   output out=max_response_summary(drop=_type_ _freq_)
          max=max_response_value;
run;

/* Step 3: Calculate the total response value per treatment group and visit day */
proc means data=patient_data noprint;
   class treatment_group visit_day;
   var response;
   output out=total_response_summary(drop=_type_ _freq_)
          sum=total_response_value;
run;

/* Step 4: Merge the two summary datasets */
data final_summary;
   merge max_response_summary(in=a)
         total_response_summary(in=b);
   by treatment_group;
   if a and b;
run;
```



Explanation of Each Step:

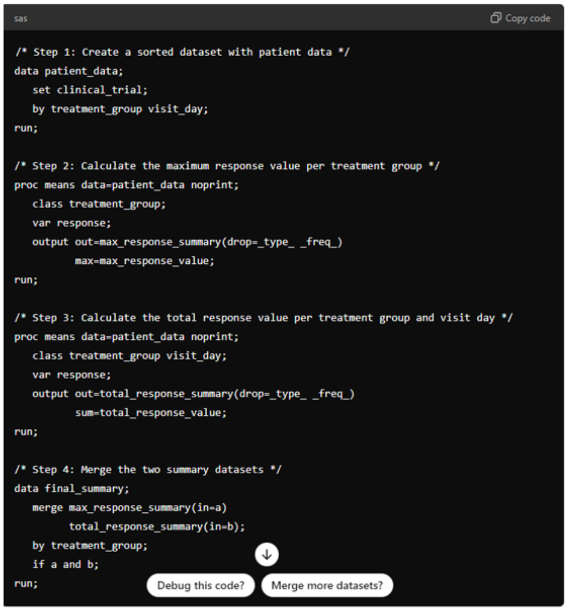
1. Creating a Sorted Dataset:
   - The `` `DATA` `` step creates a dataset named `` `patient_data` `` and sorts it by `` `treatment_group` `` and `` `visit_day` ``.
2. Calculating Maximum Response Value:
   - `` `PROC MEANS` `` calculates the maximum response value for each `` `treatment_group` `` and outputs the results to `` `max_response_summary` ``.
3. Calculating Total Response Value:
   - `` `PROC MEANS` `` calculates the total response value for each combination of `` `treatment_group` `` and `` `visit_day` ``, outputting the results to `` `total_response_summary` ``.
4. Merging the Summary Datasets:
   - The `` `DATA` `` step merges `` `max_response_summary` `` and `` `total_response_summary` `` by `` `treatment_group` ``. The `` `if a and b;` `` statement ensures that only records present in both datasets are included in the final output.

This approach accomplishes the same tasks as the `` `PROC SQL` `` code but uses `` `DATA` `` steps and `` `PROC MEANS` `` instead.

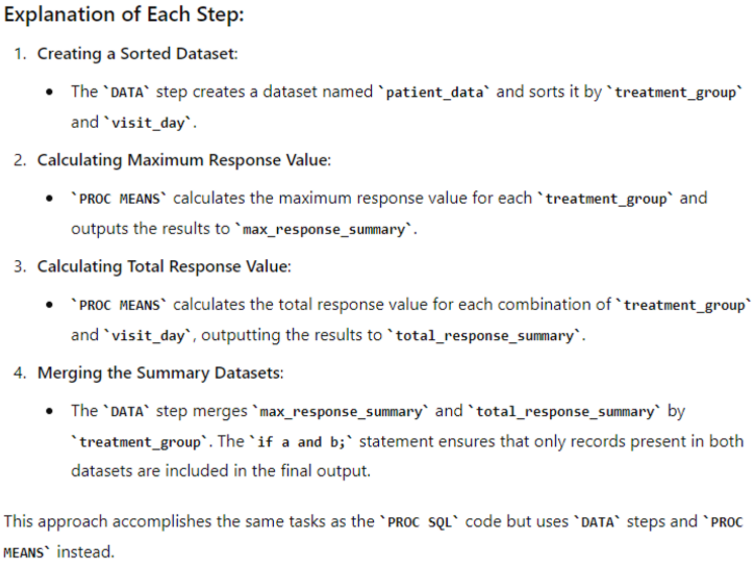
PROMPT 3: ENHANCING CODE WITH COMMENTS

Prompt: *"Could you add some comments to the code to help me understand?"*

AI-generated comments clarify code logic, enhance documentation and ease future modifications. This helps both the original programmer and future team members understand the code's purpose and functionality.



```sas
/* Step 1: Create a sorted dataset with patient data */
data patient_data;
   set clinical_trial;
   by treatment_group visit_day;
run;

/* Step 2: Calculate the maximum response value per treatment group */
proc means data=patient_data noprint;
   class treatment_group;
   var response;
   output out=max_response_summary(drop=_type_ _freq_)
          max=max_response_value;
run;

/* Step 3: Calculate the total response value per treatment group and visit day */
proc means data=patient_data noprint;
   class treatment_group visit_day;
   var response;
   output out=total_response_summary(drop=_type_ _freq_)
          sum=total_response_value;
run;

/* Step 4: Merge the two summary datasets */
data final_summary;
   merge max_response_summary(in=a)
         total_response_summary(in=b);
   by treatment_group;
   if a and b;
run;
```



Explanation of Each Step:

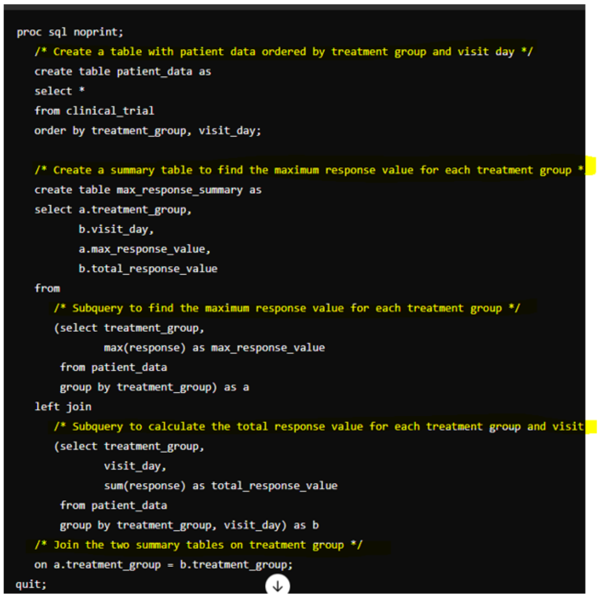
1. **Creating a Sorted Dataset:**
    - The `DATA` step creates a dataset named `patient_data` and sorts it by `treatment_group` and `visit_day`.
2. **Calculating Maximum Response Value:**
    - `PROC MEANS` calculates the maximum response value for each `treatment_group` and outputs the results to `max_response_summary`.
3. **Calculating Total Response Value:**
    - `PROC MEANS` calculates the total response value for each combination of `treatment_group` and `visit_day`, outputting the results to `total_response_summary`.
4. **Merging the Summary Datasets:**
    - The `DATA` step merges `max_response_summary` and `total_response_summary` by `treatment_group`. The `if a and b;` statement ensures that only records present in both datasets are included in the final output.

This approach accomplishes the same tasks as the `PROC SQL` code but uses `DATA` steps and `PROC MEANS` instead.

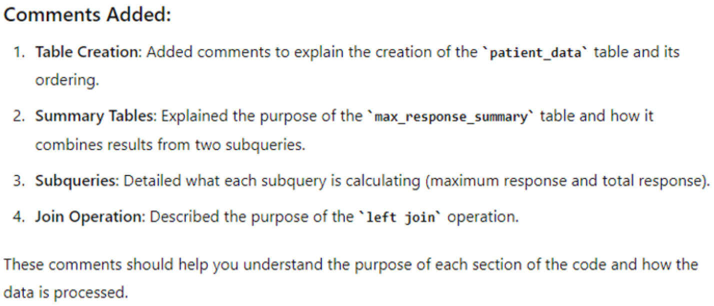
PROMPT 4: IMPLEMENTING LOCF (LAST OBSERVATION CARRIED FORWARD)

Prompt: *"Could you write a program on how to implement LOCF?"*

ChatGPT provides a SAS program to implement LOCF, a widely used method for handling missing data in clinical trials. The AI-generated code ensures robust analysis while maintaining data integrity.



```sas
proc sql noprint;
   /* Create a table with patient data ordered by treatment group and visit day */
   create table patient_data as
   select *
   from clinical_trial
   order by treatment_group, visit_day;

   /* Create a summary table to find the maximum response value for each treatment group *
   create table max_response_summary as
   select a.treatment_group,
          b.visit_day,
          a.max_response_value,
          b.total_response_value
   from
      /* Subquery to find the maximum response value for each treatment group */
      (select treatment_group,
              max(response) as max_response_value
       from patient_data
       group by treatment_group) as a
   left join
      /* Subquery to calculate the total response value for each treatment group and visit
      (select treatment_group,
              visit_day,
              sum(response) as total_response_value
       from patient_data
       group by treatment_group, visit_day) as b
   /* Join the two summary tables on treatment group */
   on a.treatment_group = b.treatment_group;
quit;
```



Comments Added:

1. **Table Creation:** Added comments to explain the creation of the `patient_data` table and its ordering.
2. **Summary Tables:** Explained the purpose of the `max_response_summary` table and how it combines results from two subqueries.
3. **Subqueries:** Detailed what each subquery is calculating (maximum response and total response).
4. **Join Operation:** Described the purpose of the `left join` operation.

These comments should help you understand the purpose of each section of the code and how the data is processed.

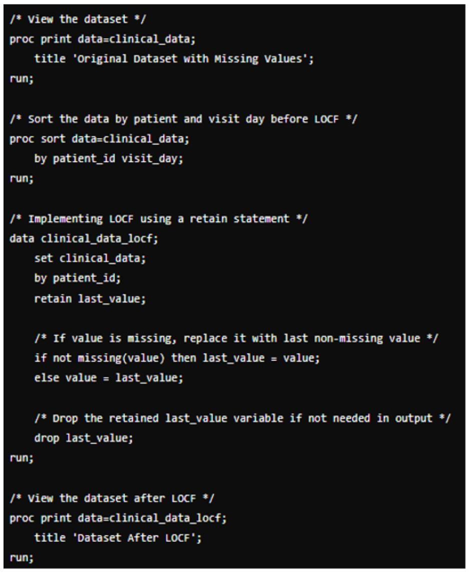
PROMPT 5: IMPLEMENTING LOCF IN MULTIPLE LANGUAGES

***Prompt:** "Write a SAS program that demonstrates how to implement LOCF in SAS, Python, R, and SQL."*

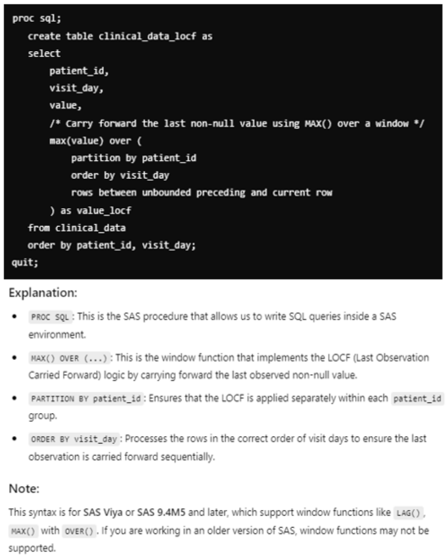
ChatGPT translates LOCF logic across SAS, Python, R, and SQL, allowing programmers to apply consistent data imputation techniques across different platforms.

Implementation in Different Languages:

- **SAS:** Uses retain and by statements to carry forward the last non-missing value.
- **Python:** Utilizes the fillna(method='ffill') function with groupby() in pandas to propagate the last valid observation forward.
- **R:** Uses the na.locf() function from the zoo package to handle missing values efficiently.
- **SQL:** Can be implemented using window functions such as LAST_VALUE() or custom queries to propagate non-missing values within a partition.



```
/* View the dataset */
proc print data=clinical_data;
    title 'Original Dataset with Missing Values';
run;

/* Sort the data by patient and visit day before LOCF */
proc sort data=clinical_data;
    by patient_id visit_day;
run;

/* Implementing LOCF using a retain statement */
data clinical_data_locf;
    set clinical_data;
    by patient_id;
    retain last_value;

    /* If value is missing, replace it with last non-missing value */
    if not missing(value) then last_value = value;
    else value = last_value;

    /* Drop the retained last_value variable if not needed in output */
    drop last_value;
run;

/* View the dataset after LOCF */
proc print data=clinical_data_locf;
    title 'Dataset After LOCF';
run;
```



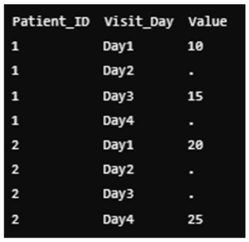
```
proc sql;
   create table clinical_data_locf as
   select
       patient_id,
       visit_day,
       value,
       /* Carry forward the last non-null value using MAX() over a window */
       max(value) over (
           partition by patient_id
           order by visit_day
           rows between unbounded preceding and current row
       ) as value_locf
   from clinical_data
   order by patient_id, visit_day;
quit;
```

Explanation:

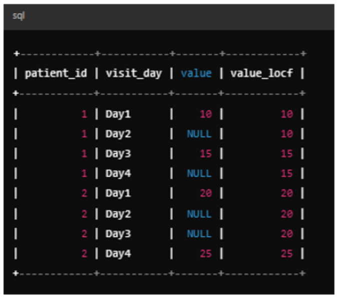
- `PROC SQL`: This is the SAS procedure that allows us to write SQL queries inside a SAS environment.
- `MAX() OVER (...)`: This is the window function that implements the LOCF (Last Observation Carried Forward) logic by carrying forward the last observed non-null value.
- `PARTITION BY patient_id`: Ensures that the LOCF is applied separately within each `patient_id` group.
- `ORDER BY visit_day`: Processes the rows in the correct order of visit days to ensure the last observation is carried forward sequentially.

Note:

This syntax is for **SAS Viya** or **SAS 9.4M5** and later, which support window functions like `LAG()`, `MAX()` with `OVER()`. If you are working in an older version of SAS, window functions may not be supported.

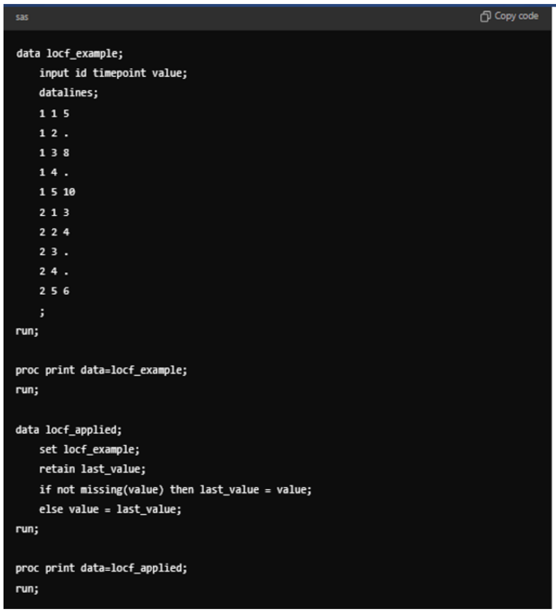


```
Patient_ID  Visit_Day  Value
1           Day1       10
1           Day2       .
1           Day3       15
1           Day4       .
2           Day1       20
2           Day2       .
2           Day3       .
2           Day4       25
```

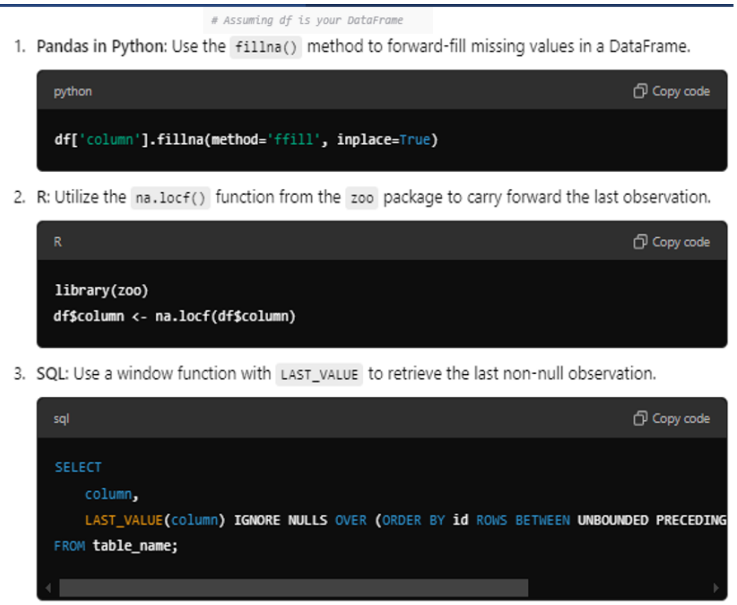


sql

```
+------------+-----------+-------+------------+
| patient_id | visit_day | value | value_locf |
+------------+-----------+-------+------------+
|          1 | Day1      |    10 |         10 |
|          1 | Day2      |  NULL |         10 |
|          1 | Day3      |    15 |         15 |
|          1 | Day4      |  NULL |         15 |
|          2 | Day1      |    20 |         20 |
|          2 | Day2      |  NULL |         20 |
|          2 | Day3      |  NULL |         20 |
|          2 | Day4      |    25 |         25 |
+------------+-----------+-------+------------+
```



```
data locf_example;
    input id timepoint value;
    datalines;
    1 1 5
    1 2 .
    1 3 8
    1 4 .
    1 5 10
    2 1 3
    2 2 4
    2 3 .
    2 4 .
    2 5 6
    ;
run;

proc print data=locf_example;
run;

data locf_applied;
    set locf_example;
    retain last_value;
    if not missing(value) then last_value = value;
    else value = last_value;
run;

proc print data=locf_applied;
run;
```



```
# Assuming df is your DataFrame
```

1. Pandas in Python: Use the `fillna()` method to forward-fill missing values in a DataFrame.

```python
df['column'].fillna(method='ffill', inplace=True)
```

2. R: Utilize the `na.locf()` function from the `zoo` package to carry forward the last observation.

```r
library(zoo)
df$column <- na.locf(df$column)
```

3. SQL: Use a window function with `LAST_VALUE` to retrieve the last non-null observation.

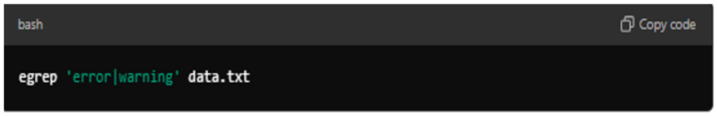
```sql
SELECT
    column,
    LAST_VALUE(column) IGNORE NULLS OVER (ORDER BY id ROWS BETWEEN UNBOUNDED PRECEDING
FROM table_name;
```

This demonstrates how ChatGPT can provide cross-platform solutions, making it easier for programmers to adapt their methods across different analytical tools.

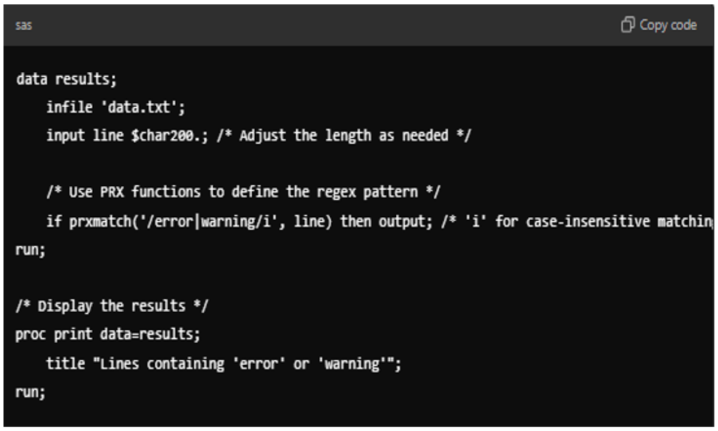
PROMPT 6: CONVERTING LINUX EGREP FUNCTION TO SAS

Prompt: *"Write a SAS program to convert the egrep Linux function."*

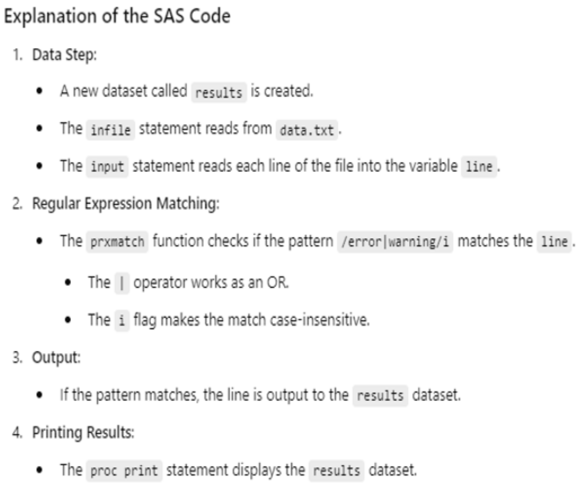
ChatGPT translates Linux egrep functionality into SAS using PRX functions, demonstrating how AI can assist in adapting regular expression searches to different programming environments.



```bash
egrep 'error|warning' data.txt
```



```sas
data results;
    infile 'data.txt';
    input line $char200.; /* Adjust the length as needed */

    /* Use PRX functions to define the regex pattern */
    if prxmatch('/error|warning/i', line) then output; /* 'i' for case-insensitive matchin
run;

/* Display the results */
proc print data=results;
    title "Lines containing 'error' or 'warning'";
run;
```



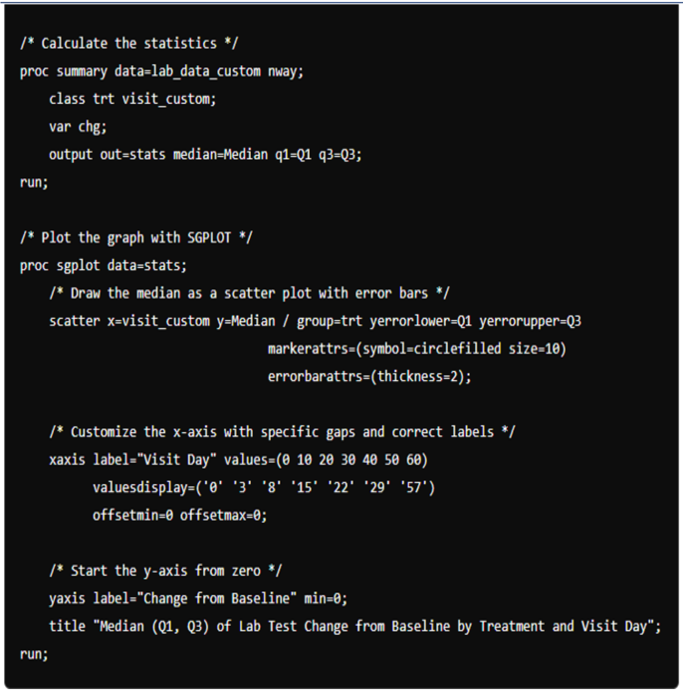
Explanation of the SAS Code

1. Data Step:
    - A new dataset called `results` is created.
    - The `infile` statement reads from `data.txt`.
    - The `input` statement reads each line of the file into the variable `line`.
2. Regular Expression Matching:
    - The `prxmatch` function checks if the pattern `/error|warning/i` matches the `line`.
        - The `|` operator works as an OR.
        - The `i` flag makes the match case-insensitive.
3. Output:
    - If the pattern matches, the line is output to the `results` dataset.
4. Printing Results:
    - The `proc print` statement displays the `results` dataset.

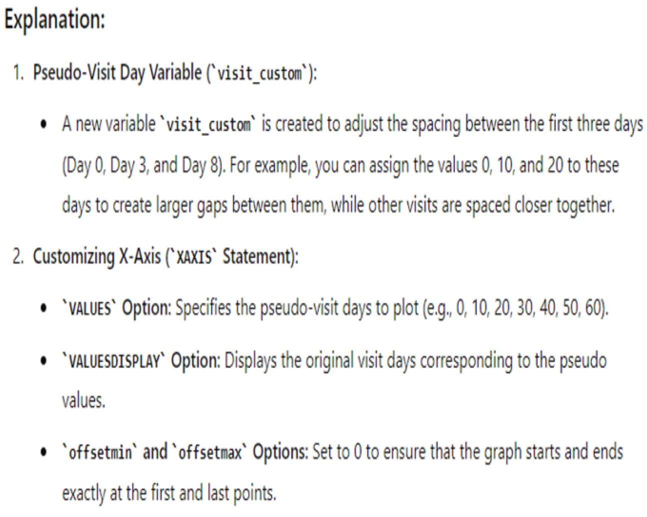
PROMPT 7: CREATING AN SGPLOT FOR MEDIAN (Q1, Q3) BY TREATMENT AND VISIT

Prompt*: "Can you generate a SAS code to create SGPLOT to display Median (Q1, Q3) for Lab test (PARAMCD) change from baseline (CHG) by treatment (TRT) and visit where a graph line starts from zero? Add a relevant title and aesthetically pleasing colors."*

ChatGPT generates SAS code to create clear and aesthetically appealing data visualizations, ensuring effective representation of statistical summaries.

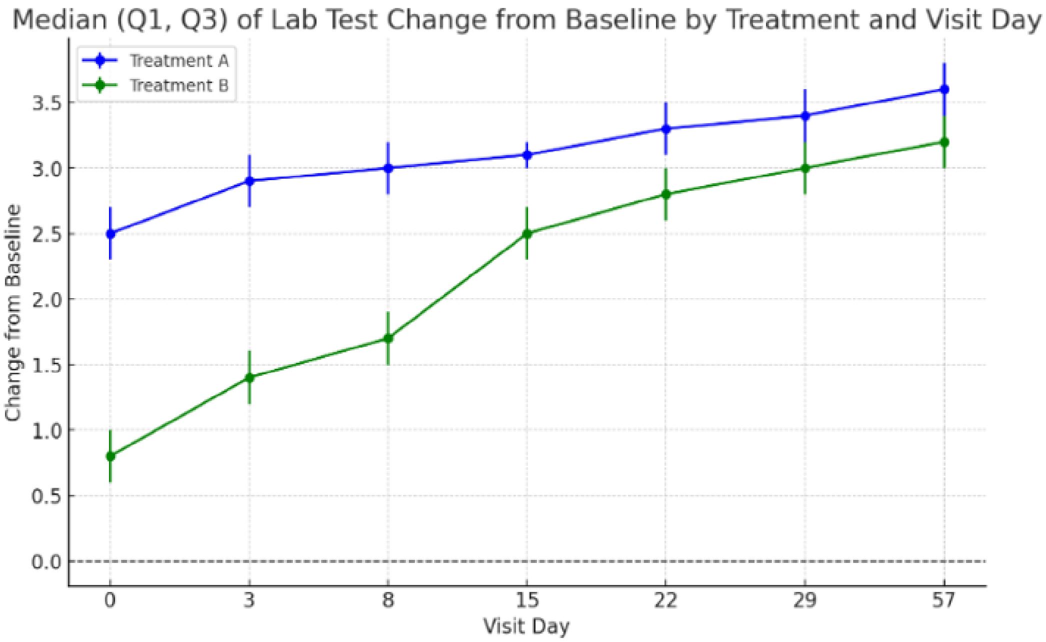


```sas
/* Calculate the statistics */
proc summary data=lab_data_custom nway;
    class trt visit_custom;
    var chg;
    output out=stats median=Median q1=Q1 q3=Q3;
run;

/* Plot the graph with SGPLOT */
proc sgplot data=stats;
    /* Draw the median as a scatter plot with error bars */
    scatter x=visit_custom y=Median / group=trt yerrorlower=Q1 yerrorupper=Q3
                              markerattrs=(symbol=circlefilled size=10)
                              errorbarattrs=(thickness=2);

    /* Customize the x-axis with specific gaps and correct labels */
    xaxis label="Visit Day" values=(0 10 20 30 40 50 60)
          valuesdisplay=('0' '3' '8' '15' '22' '29' '57')
          offsetmin=0 offsetmax=0;

    /* Start the y-axis from zero */
    yaxis label="Change from Baseline" min=0;
    title "Median (Q1, Q3) of Lab Test Change from Baseline by Treatment and Visit Day";
run;
```



Explanation:

1. Pseudo-Visit Day Variable (`` `visit_custom` ``):
    - A new variable `` `visit_custom` `` is created to adjust the spacing between the first three days (Day 0, Day 3, and Day 8). For example, you can assign the values 0, 10, and 20 to these days to create larger gaps between them, while other visits are spaced closer together.
2. Customizing X-Axis (`` `XAXIS` `` Statement):
    - `` `VALUES` `` Option: Specifies the pseudo-visit days to plot (e.g., 0, 10, 20, 30, 40, 50, 60).
    - `` `VALUESDISPLAY` `` Option: Displays the original visit days corresponding to the pseudo values.
    - `` `offsetmin` `` and `` `offsetmax` `` Options: Set to 0 to ensure that the graph starts and ends exactly at the first and last points.

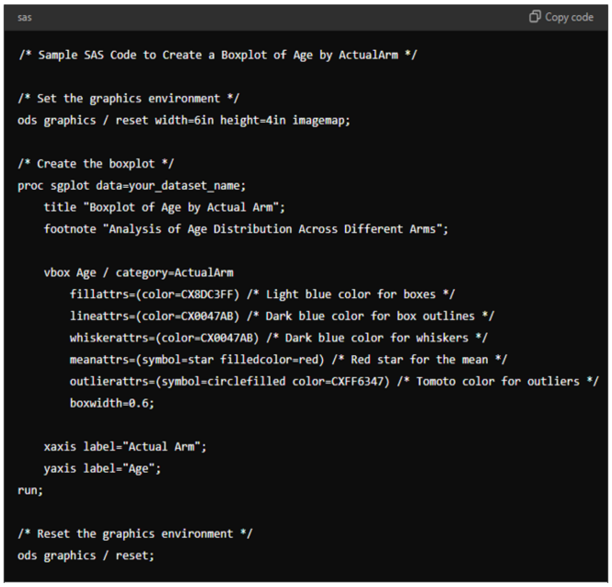


Here's a mockup figure that represents the concept from the SAS code. The figure shows the medians with interquartile ranges (Q1, Q3) for two treatment groups, with custom gaps on the x-axis between the first three visits (Day 0, Day 3, and Day 8). The subsequent visit days are spaced more closely together. The x-axis labels reflect the actual visit days, while the spacing between them highlights the early time points.

PROMPT 8: CREATING A BOXPLOT IN SAS

***Prompt**: "Can you generate a SAS code to create a boxplot of age by actual arm? Add a relevant title and footnote and aesthetically pleasing colors."*

ChatGPT automates SAS boxplot generation, enhancing data visualization with professional and informative graphical representations.



```sas
/* Sample SAS Code to Create a Boxplot of Age by ActualArm */

/* Set the graphics environment */
ods graphics / reset width=6in height=4in imagemap;

/* Create the boxplot */
proc sgplot data=your_dataset_name;
    title "Boxplot of Age by Actual Arm";
    footnote "Analysis of Age Distribution Across Different Arms";

    vbox Age / category=ActualArm
        fillattrs=(color=CX8DC3FF) /* Light blue color for boxes */
        lineattrs=(color=CX0047AB) /* Dark blue color for box outlines */
        whiskerattrs=(color=CX0047AB) /* Dark blue color for whiskers */
        meanattrs=(symbol=star filledcolor=red) /* Red star for the mean */
        outlierattrs=(symbol=circlefilled color=CXFF6347) /* Tomoto color for outliers */
        boxwidth=0.6;

    xaxis label="Actual Arm";
    yaxis label="Age";
run;

/* Reset the graphics environment */
ods graphics / reset;
```

Explanation:

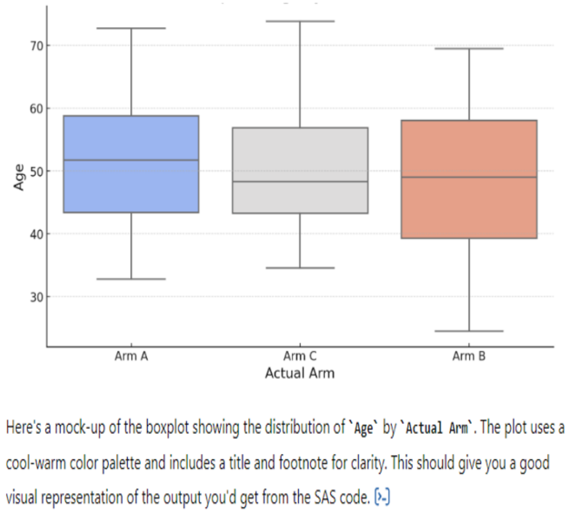
- **Title and Footnote**: A title and footnote are added to make the plot more informative.
- **Color Customization**: The boxplot colors are customized for a more aesthetically pleas appearance:
    - **Boxes**: Light blue color.
    - **Outlines and Whiskers**: Dark blue color.
    - **Mean**: Red star.
    - **Outliers**: Tomato color circles.
- **Box Width**: Adjusted for better visual appeal.

Replace `your_dataset_name` with the actual name of your dataset and adjust any labels o fit your needs.

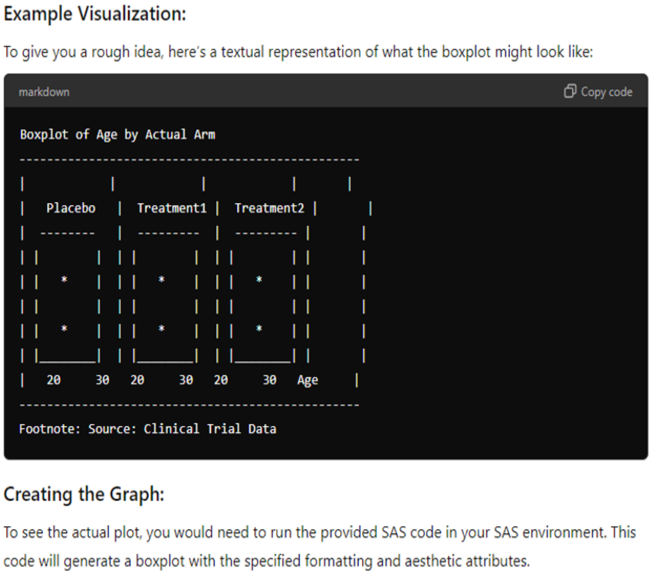
PROMPT 9: GENERATING A MOCK-UP GRAPH

Prompt: *"Can you generate a mock-up graph for the code above?"*

AI assists in creating preliminary visualizations before finalizing SAS-generated graphs, ensuring alignment with project expectations.



Here's a mock-up of the boxplot showing the distribution of `Age` by `Actual Arm`. The plot uses a cool-warm color palette and includes a title and footnote for clarity. This should give you a good visual representation of the output you'd get from the SAS code.



Example Visualization:

To give you a rough idea, here's a textual representation of what the boxplot might look like:

```
Boxplot of Age by Actual Arm

-------------------------------------------------

|            |            |            |       |
|   Placebo   |  Treatment1 |  Treatment2 |       |
|  --------   |  ---------  |  --------- |       |
| |        |  | |        |  | |        | |       |
| |   *    |  | |   *    |  | |   *    | |       |
| |        |  | |        |  | |        | |       |
| |   *    |  | |   *    |  | |   *    | |       |
| |________|  | |________|  | |________| |       |
|   20     30   20     30   20     30   Age     |

-------------------------------------------------

Footnote: Source: Clinical Trial Data
```

Creating the Graph:

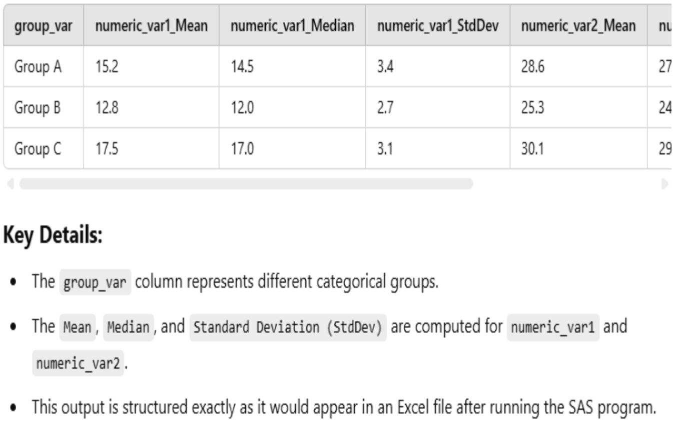
To see the actual plot, you would need to run the provided SAS code in your SAS environment. This code will generate a boxplot with the specified formatting and aesthetic attributes.

PROMPT 10: GENERATING SUMMARY STATISTICS AND EXPORTING TO EXCEL

Prompt: *"Can you generate a SAS program to compute summary statistics (mean, median, standard deviation) for key variables and export the results to an Excel file?"*

ChatGPT produces a SAS program to compute summary statistics and export results efficiently to Excel, streamlining data reporting workflows.

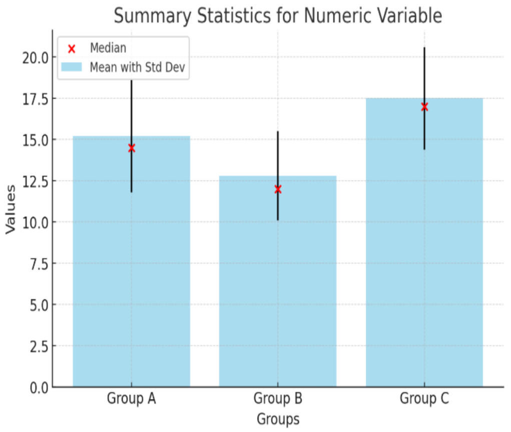
- Compute mean, median, and standard deviation for numeric variables using PROC MEANS.
- Categorizes data using CLASS variables for stratified analysis.
- Exports the results directly to an Excel file using PROC EXPORT.



group_var	numeric_var1_Mean	numeric_var1_Median	numeric_var1_StdDev	numeric_var2_Mean	nu
Group A	15.2	14.5	3.4	28.6	27
Group B	12.8	12.0	2.7	25.3	24
Group C	17.5	17.0	3.1	30.1	29

Key Details:

- The `group_var` column represents different categorical groups.
- The `Mean`, `Median`, and `Standard Deviation (StdDev)` are computed for `numeric_var1` and `numeric_var2`.
- This output is structured exactly as it would appear in an Excel file after running the SAS program.



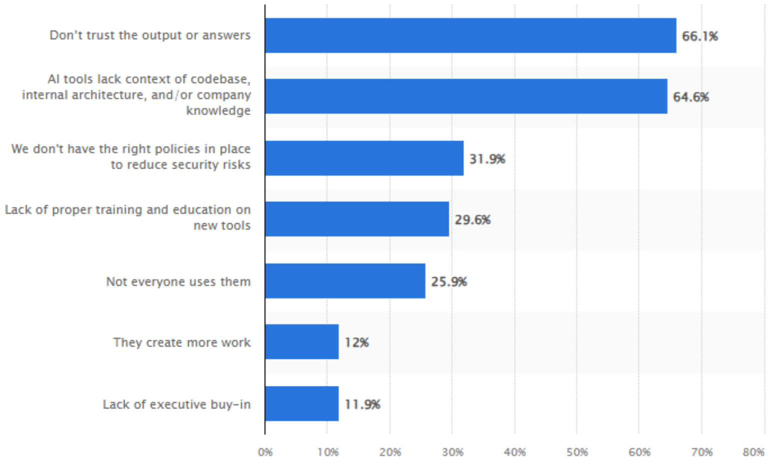
*Right figure is a **mock-up graph** visualizing the summary statistics:*

- Bars represent the mean values with standard deviation error bars.
- Red dots indicate the median values for each group.

CHALLENGES AND CONSIDERATIONS IN AI-DRIVEN SAS PROGRAMMING

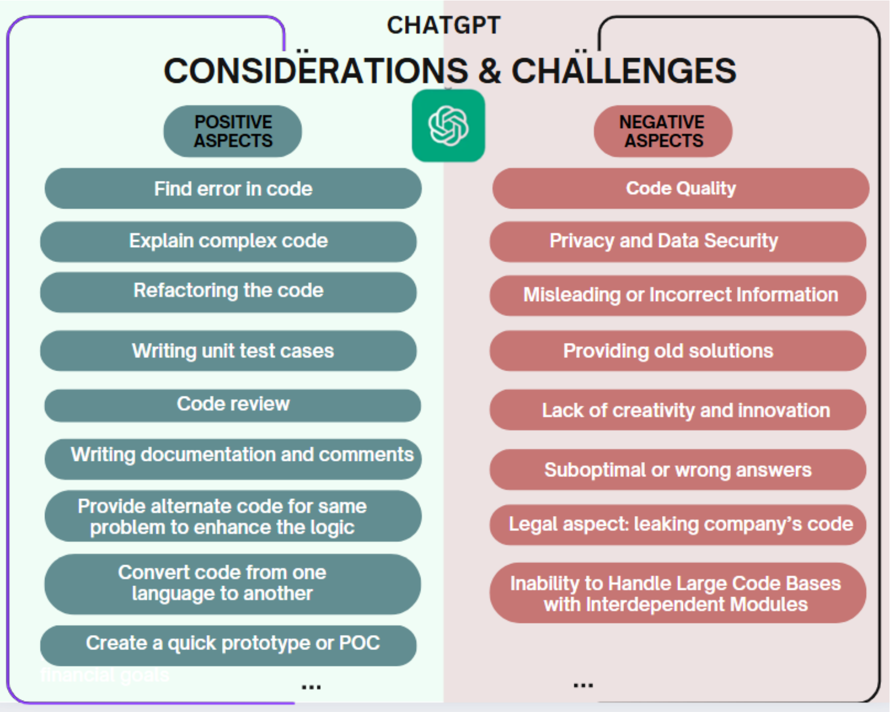
While AI tools like ChatGPT enhance SAS programming, they also pose challenges.

- **Data privacy** is a major concern, especially in regulated industries like healthcare, where AI models risk exposing sensitive data. Ensuring compliance with GDPR and HIPAA is crucial.
- **Bias in AI** can also impact results, as models trained on imbalanced data may produce skewed insights, requiring human oversight and well-curated datasets.
- **Regulatory compliance** must be maintained, aligning AI-generated outputs with FDA and EMA guidelines to ensure accuracy and transparency.
- **Ethical concerns** arise regarding AI accountability, decision-making, and the lack of transparency in how AI generates conclusions.
- **Resource investment and expertise** are essential, as AI adoption requires skilled professionals and cost-effective implementation.



Source: © Statista 2025 (https://shorturl.at/7hRDs)

Addressing these challenges proactively allows organizations to harness AI's benefits while ensuring data security, compliance, and ethical integrity.



CHATGPT

CONSIDERATIONS & CHALLENGES

POSITIVE ASPECTS

- Find error in code
- Explain complex code
- Refactoring the code
- Writing unit test cases
- Code review
- Writing documentation and comments
- Provide alternate code for same problem to enhance the logic
- Convert code from one language to another
- Create a quick prototype or POC
- ...

NEGATIVE ASPECTS

- Code Quality
- Privacy and Data Security
- Misleading or Incorrect Information
- Providing old solutions
- Lack of creativity and innovation
- Suboptimal or wrong answers
- Legal aspect: leaking company's code
- Inability to Handle Large Code Bases with Interdependent Modules
- ...

CONCLUSION

The integration of AI tools like ChatGPT into SAS programming is transforming the way programmers approach coding, debugging, and data analysis. By leveraging AI-driven prompts, users can efficiently reformat code, explore alternative solutions, generate visualizations, and optimize workflows. However, AI is not a replacement for human expertise. Programmers must remain vigilant in reviewing AI-generated code to ensure accuracy, efficiency, and compliance with industry standards.

As AI evolves, its capabilities in SAS programming will continue to advance. By adopting a balanced approach that combines AI-driven efficiency with human oversight, programmers can unlock new levels of productivity while ensuring data integrity, security, and compliance.

REFERENCES

- OpenAI. (2025). ChatGPT and AI in Programming. OpenAI. Available at https://openai.com.
- Wikipedia. (2025). Generative Artificial Intelligence. Available at https://en.wikipedia.org/wiki/Generative_artificial_intelligence.
- Statista Research. (2025). AI Adoption in Software Development. Statista. Available at https://www.statista.com.
- New York Post. (2025). How AI is Transforming Coding Practices. Available at https://nypost.com.
- Stack Overflow. (2025). 2025 Developer Survey Results. Available at https://survey.stackoverflow.co.
- Zero To Mastery. (2023). The State of AI Tools and Coding: 2023 Edition. Available at https://zerotomastery.io/blog/the-state-of-ai-tools-and-coding-2023-edition/.
- Statista. (2025). AI Tools Usage Among Developers Worldwide. Available at https://www.statista.com/statistics/1483838/ai-tools-usage-among-developers-use-worldwide/.
- Developer Updates. (2025). ChatGPT Guide for Software Developers. Available at https://www.developerupdates.com/shop/chatgpt-guide-for-software-developers.

ACKNOWLEDGMENTS

The author acknowledges the AI assistance and invaluable insights gained through interactions with ChatGPT and the support of the OpenAI community.

CONTACT INFORMATION

Comments and questions are valued and encouraged. Contact the author at:

Jyoti (Jo) Agarwal
Jyoti.agarwal1@gilead.com