PharmaSUG 2025 - Paper AP-039

How Not to SAS: Avoiding Common Pitfalls and Bad Habits

Melodie Rush, SAS Institute Inc., Cary, NC

ABSTRACT

Using SAS software effectively isn't just about knowing the syntax—it's about developing habits and strategies that enhance your efficiency and accuracy. In this 20-minute session, we'll take a humorous look at some of the worst practices in SAS programming, curated with the help of ChatGPT, and explain why these missteps can lead to frustration, inefficiency, or even disastrous results. By highlighting these "bad tips," we'll demonstrate best practices to avoid these mistakes, organize your work, optimize performance, and debug like a pro. Whether you're a beginner or an experienced SAS user, this session will give you practical advice to sharpen your skills and streamline your SAS journey.

INTRODUCTION

SAS is a powerful tool for data analysis, but its flexibility can sometimes lead you into developing bad programming habits. Although these shortcuts might not break your programs immediately, they can lead to inefficient, error-prone, and hard-to-maintain code. This paper identifies common pitfalls and provides straightforward, practical solutions to avoid them.

CODE ORGANIZATION & READABILITY

Effective code organization is foundational to successful SAS programming. Yet, it's easy to overlook best practices and fall into poor habits. Here are three common pitfalls you should avoid:

BAD TIP: NEVER COMMENT YOUR CODE

Who needs to understand what you wrote three months ago? Let your future self figure it out. One of the worst tips for SAS programmers is to neglect commenting. Although it might seem faster initially, this habit creates confusion later. Proper comments improve readability, facilitate collaboration, and ensure that your logic remains clear—even months or years down the road. Good comments should explain complex logic, clarify the purpose of key steps, and help you and others quickly understand the code structure. Clear commenting reduces maintenance headaches and significantly enhances teamwork.

Good Practice: Use clear and concise SAS comments in SAS code

Use comment syntax options in your SAS Code (Program1):

- block comments like /* my comment */
- single-line comments like * my comment;

```
/* my program comment */

data one; set sashelp.cars;
*subset to certain car types - SUV;
      where type = "SUV";
run;
```

Program 1. Example of block and single-line comments.

BAD TIP: WRITE ALL YOUR CODE IN ONE LONG, CONTINUOUS BLOCK

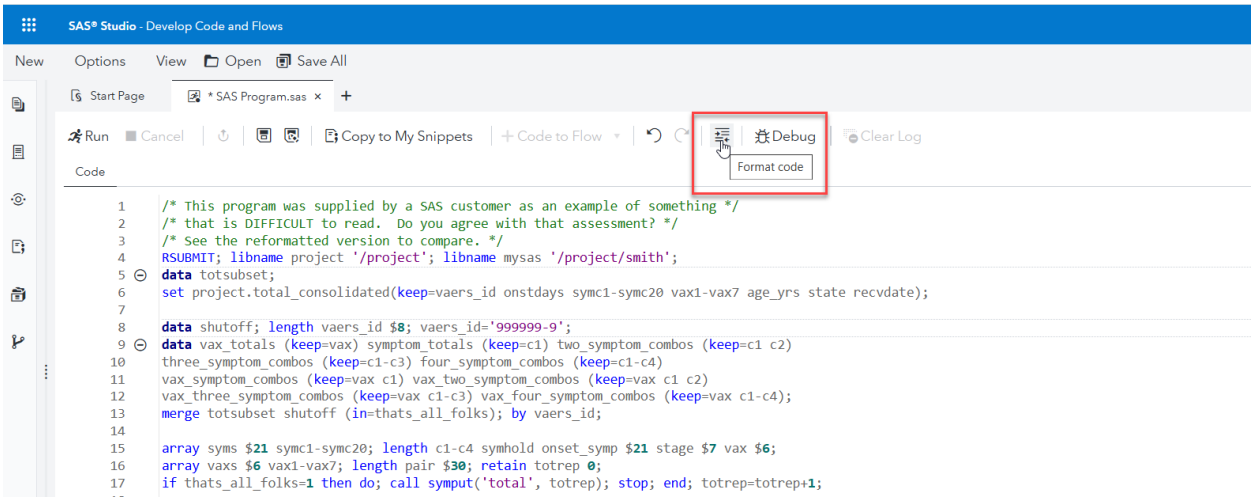
Because scrolling endlessly through code is an exciting way to test your patience and eyesight

Writing your code as one massive block is another common but harmful habit. Such a practice makes debugging and code revisions cumbersome and increases the likelihood of errors going unnoticed. Instead, write your code modularly. Break your code into smaller, logical steps, using clear headers or

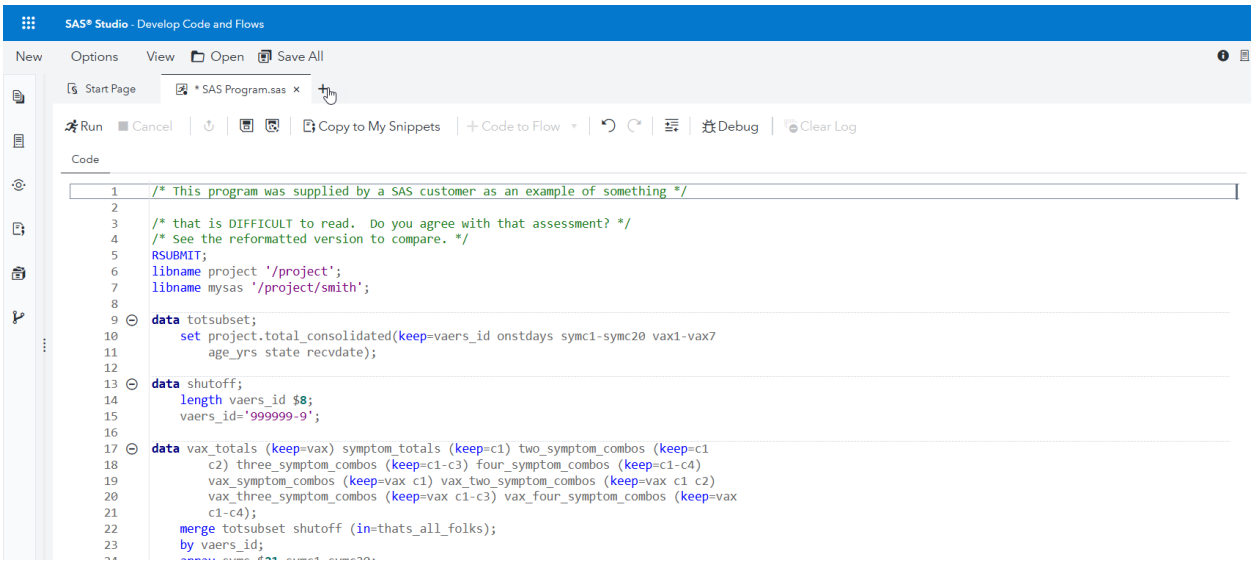
modular sections to define each block's purpose. Modular code not only improves readability but also allows you to easily reuse sections in other projects or analyses, saving valuable time.

Good Practice: Use modular, structured code

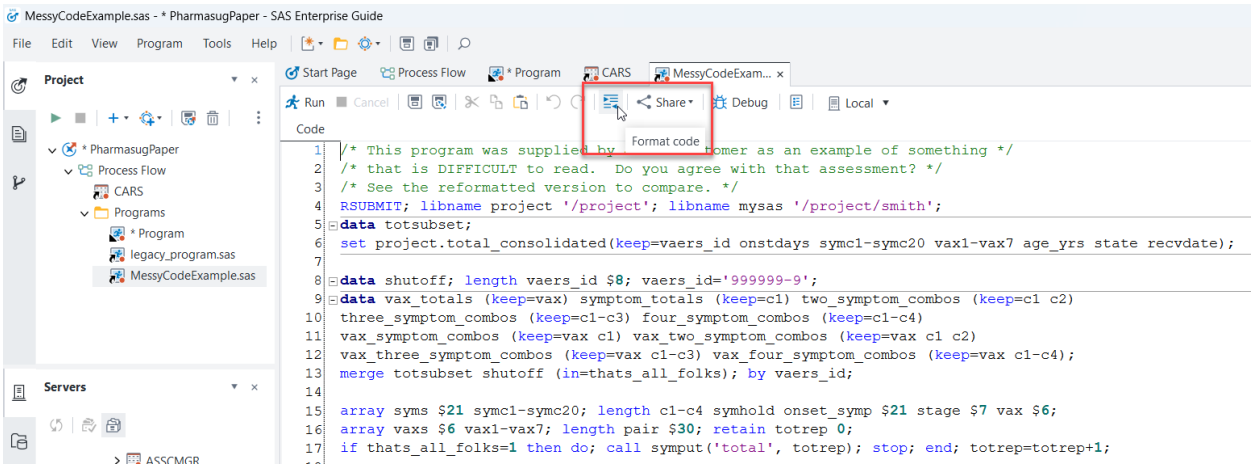
- Clearly separate different steps with comments.
- Take advantage of built-in formatting tools provided by SAS Studio (Display 1 and Display 2) and Enterprise Guide (Display 3 and Display 4) to enhance readability and simplify navigation.



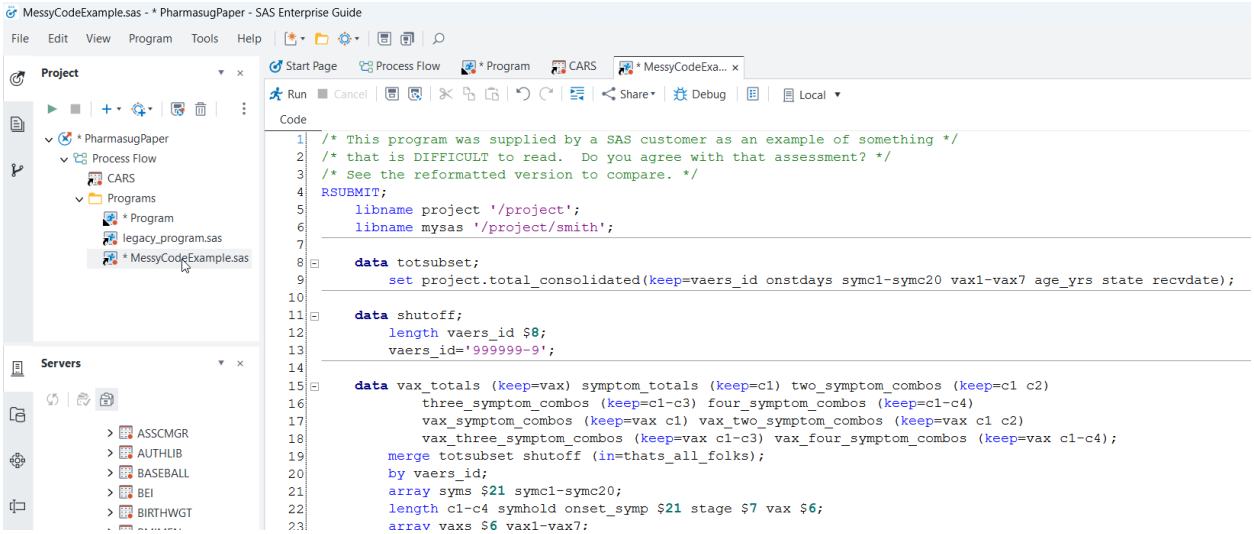
Display 1. SAS Studio Format Code Tool – Code Before Using Tool



Display 2. SAS Studio Format Code Tool – Code After Using Tool



Display 3. SAS Enterprise Guide Format Code Tool – Code Before Using Tool



Display 4. SAS Enterprise Guide Format Code Tool – Code After Using Tool

BAD TIP: USE UNCLEAR OR ARBITRARY VARIABLE NAMES

Using cryptic variable names like VAR1 and X turns your code into an unsolvable puzzle game for you or those who inherit your code.

Choosing poor variable names like VAR1, X, or TEMP reduces readability and introduces confusion into your code (Program 2). Also, generic names increases the likelihood of errors and misinterpretation of the results.

```
data new;
  set old;
  z=x+y;
run;
```

Program 2. Example of not using clear and descriptive variable names

Good Practice: Use meaningful and descriptive variable names

Meaningful and descriptive variable names (such as CustomerAge instead of X) allow you and your colleagues to quickly grasp the purpose and content of each variable (Program 3). Adopting consistent naming conventions, such as using underscores (customer_age) or camel case (CustomerAge), further improves clarity. Clearly named variables also simplify debugging and significantly reduce the likelihood of errors during analysis.

```
data salaryinfo2021;
  set salaryinfo2020;
  newsalary=oldsalary+increase;
run;
```

Program 3. Example of using clear and descriptive variable names

DEBUGGING & ERROR HANDLING

Proper debugging and error handling are critical for creating reliable SAS programs. Here are common pitfalls and the best practices you should follow:

BAD TIP: IGNORING THE SAS LOG WINDOW

Red text is just a suggestion. Who needs to debug when you can keep running the code? Ignoring the log window means missing critical information about errors and warnings, causing unnoticed mistakes and incorrect results.

Good Practice: Check all the messages in your log

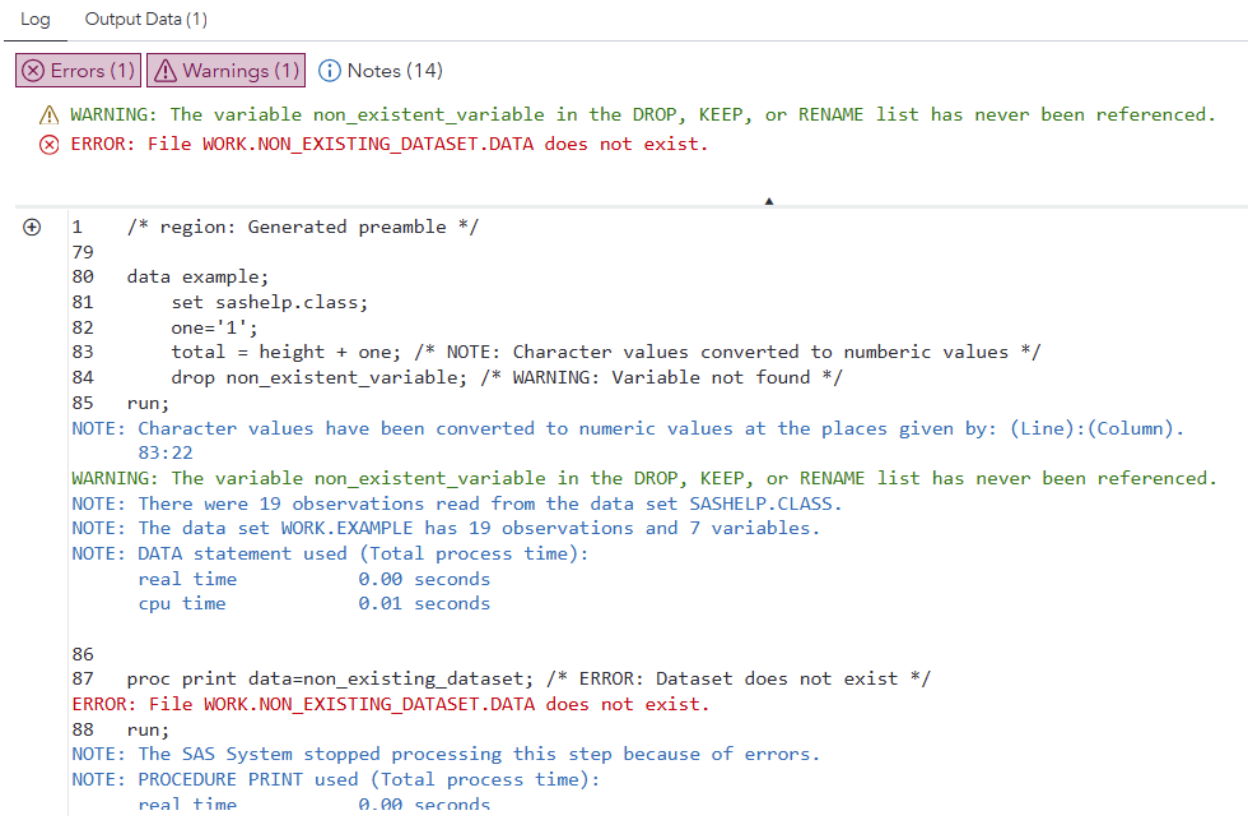
Always check for `ERROR`, `WARNING`, and `NOTE` messages in your log. Each of these messages can indicate fatal failures in your code (Display 5).

Understanding SAS Log Messages:

ERROR: Critical issues that prevent SAS from executing your code. Your results are incomplete or incorrect until these are resolved.

WARNING: Potential issues that SAS identifies but doesn't stop execution. These should be reviewed and addressed to ensure accuracy.

NOTE: Informational messages about code execution. These offer insights into dataset creation, memory usage, and other operational details.



Display 5. SAS Log Window showing ERROR, WARNING and NOTE

BAD TIP: NEVER USE DEBUGGING OPTIONS

Skipping debugging options is a great way to keep your coding life exciting—who doesn't love spending hours chasing hidden bugs? Avoiding the use of debugging options can significantly hinder your ability to troubleshoot and resolve standard SAS code and macro-related issues efficiently.

Good Practice: Use system options to help you debug SAS Code

Use system options to enhance log detail and improve debugging capabilities:

- **MSGLEVEL=I:** Provides additional informational messages in the log, especially useful when merging datasets to identify issues such as mismatches or data alignment problems.
- **SOURCE:** Displays the original SAS statements in the log.
- **SOURCE2:** Shows included SAS code from `%INCLUDE` statements.
- **FMTERR:** Issues an error if a specified format cannot be found.
- **DSNFERR:** Issues an error when a referenced dataset does not exist.
- **OBS=0:** Compiles the program without executing it, useful for syntax checking.
- **NOREPLACE:** Prevents accidental overwriting of existing datasets.

```
/* Turn on options */
options source2 source msglevel=i;
```

Program 4. Example of SAS Program options commonly used

Use debugging options specifically designed for macros: `OPTIONS MPRINT SYMBOLGEN MLOGIC;`

- **MPRINT:** Displays the actual SAS statements generated by macro execution, helping you identify issues within macros.
- **SYMBOLGEN:** Shows the resolution of macro variables, assisting you in confirming that macro variables resolve correctly.
- **MLOGIC:** Provides detailed information about macro execution, including macro parameter values and logical branching, useful for troubleshooting complex macro logic.

```
/* Turn on options */
options mprint symbolgen mlogic mautosource mcompilenote=ALL;
```

Program 5. Example of SAS Macro options commonly used

BAD TIP: RUNNING CODE WITHOUT VERIFYING INPUT DATA

Just assume your dataset is perfect—because real-world data is always flawless, right? Trusting imported data without verification can lead to incorrect analyses, wasted time, and unreliable results.

Good Practice: Assume all data is “guilty until proven innocent”

- Inspect dataset properties using PROC CONTENTS, PROC MEANS, and PROC FREQ before analysis.
- Validate key uniqueness, check for missing values, and confirm data quality before merging datasets.
- Check for numeric-to-character conversions and unexpected results to avoid unintended data type changes and associated analytical errors.

DATA MANAGEMENT MISTAKES

Efficient data management is crucial in SAS programming to avoid data loss, facilitate easy retrieval, and ensure accurate analyses. Here are some common mistakes to avoid and best practices to adopt:

BAD TIP: STORE ALL YOUR DATA IN WORK

Because who doesn’t enjoy the adrenaline rush of potentially losing hours of work? Storing all data in the temporary WORK library is risky because data stored there is deleted once your SAS session ends. This practice can lead to significant data loss, especially if you encounter unexpected session closures or interruptions.

Good Practice: Store important data in permanent libraries

- Store important datasets in permanent libraries to ensure data persistence beyond the current session.
- Permanent libraries help secure your data, enabling long-term storage, sharing across sessions, and preventing accidental data loss.

BAD TIP: AVOID USING LIBRARIES

Why make things easy when you can spend extra hours hunting for files? Avoiding the use of libraries can lead to disorganized file management, making it challenging to locate datasets and maintain clean project structures.

Good Practice: Create and use SAS libraries

- Use SAS libraries to streamline data management by logically grouping related datasets.
- Clearly named and structured libraries improve data accessibility, simplify data sharing, and enhance project collaboration.

BAD TIP: RUNNING CODE ON PRODUCTION DATABASE WITHOUT TESTING IT FIRST

Nothing spices up the workday quite like taking unnecessary risks with live data! Running untested code directly on a production database risks data integrity, can cause significant disruptions, and might lead to costly errors or downtime.

Good Practice: Use a development or test environment for creating code

- Always test your code thoroughly in a safe, isolated environment before deploying it to production.
- Comprehensive testing helps identify potential issues early, ensuring that your code operates reliably and safeguards the production environment.

WORKING WITH DATES & TIMES IN SAS

Accurate handling of dates and times is critical for reliable analyses in SAS. Mistakes in this area can lead to serious analytical errors and confusion. Here are common pitfalls and best practices to adopt:

UNDERSTANDING SAS DATES

SAS dates are numeric values representing the number of days since January 1, 1960. This numeric representation simplifies calculations involving dates, such as finding differences between two dates or shifting dates by specific intervals.

Example:

January 1, 1960, is represented as `0`.

January 2, 1960, is represented as `1`.

December 31, 1959, is represented as `-1`.

When printed or displayed, SAS applies date formats to convert these numeric values into readable dates.

```
data _null_;
    today_date = today();
    put today_date= date9.;
run;
```

Program 5. SAS Code demonstrating SAS dates

```
today_date=17MAR2025
```

Output 1. Output from SAS Code demonstrating SAS dates

BAD TIP: TREAT DATE VARIABLES AS CHARACTER STRINGS

Because nothing makes your day more interesting than trying to calculate differences between words! Treating date variables as character strings complicates calculations and can easily introduce errors.

Good Practice: Efficiently handle date values

- SAS dates, times, and datetime values are stored as numbers, making them ideal for calculations and comparisons.
- Use the DATEPART(datetime_variable) function to easily extract date values from datetime variables. (Program 6 and Output2)
- Utilize the INTNX function for precise date shifting, such as adjusting to the first day of the next month. (Program 6 and Output2)

```
data one;
      dtvalue=2064365417;
      StartDate=put(datepart(dtvalue), date9.);
      EndDate=put(intnx('DAY', datepart(dtvalue), 3), date9.);
run;
```

Program 6. SAS Code demonstrating SAS dates with the DATEPART and INTNX function

dtvalue	StartDate	EndDate
2064365417	01JUN2025	04JUN2025

Output 2. Output from SAS Code demonstrating SAS dates created with DATEPART and INTNX function

BAD TIP: FAIL TO CHECK FOR MISSING OR INVALID DATE VALUES

Why waste time checking dates when you can be surprised by data anomalies at the most inconvenient moments? Ignoring checks for missing or invalid date values often results in inaccurate analyses and unexpected results.

Good Practice: Ensure accuracy and data integrity

- Regularly use PROC FREQ or PROC UNIVARIATE to identify unexpected gaps, missing values, or out-of-range dates.
- Apply ANYDTDTE. informats to reliably convert messy or varied external date formats into SAS-readable dates.

```
data test;
      dateinfo='01JUN2025 9:00:08.5';
      sasdate=input(dateinfo, anydtdte21.);
      put sasdate;
      put sasdate date9.;
run;
```

Program 7. SAS Code demonstrating SAS dates using the ANYDTDTE. Informat.

```
80   data test;
81      dateinfo='01JUN2025 9:00:08.5';
82      sasdate=input(dateinfo,anydtdte21.);
83      put sasdate ;
84      put sasdate date9.;
85   run;
23893
01JUN2025
```

Output 3. Output from SAS Code demonstrating SAS dates using the ANYDTDTE. Informat.

AUTOMATION & REUSABILITY

Automation and reuse of code are essential for improving efficiency, accuracy, and maintainability in your SAS workflows. Here are two common pitfalls to avoid and best practices to adopt:

BAD TIP: AVOID UISNG MACRO VARIABLES

Because repeatedly updating the same hardcoded values in multiple places keeps your day excitingly error prone. Using macro variables prevents errors, simplifies updates, and enhances consistency throughout your code.

```
data report1;
    set sales;
    where year = 2025;
run;

data report2;
    set expenses;
    where year = 2025;
run;
```

Program 8. SAS Code demonstrating bad practice of hard coding values and not using macro variables.

Good Practice: Use macro variables for consistent values

Define macro variables to manage and update values consistently across your programs.

```
%let report_year = 2025;

data report1;
    set sales;
    where year = &report_year;
run;

data report2;
    set expenses;
    where year = &report_year;
run;
```

Program 9. SAS Code demonstrating good practice of using macro variables instead of hard coding values.

BAD TIP: NEVER USE MACROS, JUST COPY AND PASTE EVERYTHING

Copy-pasting your code every time ensures plenty of errors and hours of frustrating troubleshooting. Repetitive manual edits increase the chance of mistakes and decrease productivity.

```
proc means data=dataset1;
    var sales;
run;

proc means data=dataset2;
    var sales;
run;
```

Program 10. SAS Code demonstrating bad practice of copy-pasting code over and over instead of using macros to automate.

Good Practice: Automate repetitive tasks

Use SAS macros to parameterize your repetitive code. This approach significantly reduces manual errors, enhances productivity, and streamlines your workflow.

```
%macro summarize_sales(dataset);
proc means data=&dataset;
    var sales;
run;
%mend;

%summarize_sales(dataset1);
%summarize_sales(dataset2);
```

Program 11. SAS Code demonstrating good practice of using macros to automate code.

CONCLUSION

Avoiding these common pitfalls requires discipline and adherence to best practices. By improving code readability, proactively debugging, handling data efficiently, and automating repetitive tasks, you can enhance productivity, minimize errors, and produce more reliable results. Embracing quality control checks, validating data inputs, and leveraging SAS macros and macro variables will significantly improve code efficiency and maintainability. Remember, SAS coding isn't just about getting results—it's about getting them right, efficiently!

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my colleagues in the SAS Customer Success team for their invaluable support, insightful recommendations, and generous sharing of resources. Their collaborative spirit and expertise have significantly contributed to shaping the content and enhancing the clarity of this paper.

RECOMMENDED READING

SAS Conference Papers

- *Programming For Job Security Revisited: Tips and Techniques to Maximize Your Indispensability*
- *Programming For Job Security Revisited: Even More Tips and Techniques to Maximize Your Indispensability*
- *Merge with Caution: How to Avoid Common Problems when Combining SAS Datasets*
- *20 in 20: Quick Tips for SAS Enterprise Guide Users*

SAS Documentation

- *Date and Time Intervals*
- *DATEPART Function*
- *INTNX Function*
- *ANYDTDTEw. Informat*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Melodie Rush
SAS Institute Inc.
Melodie.rush@sas.com