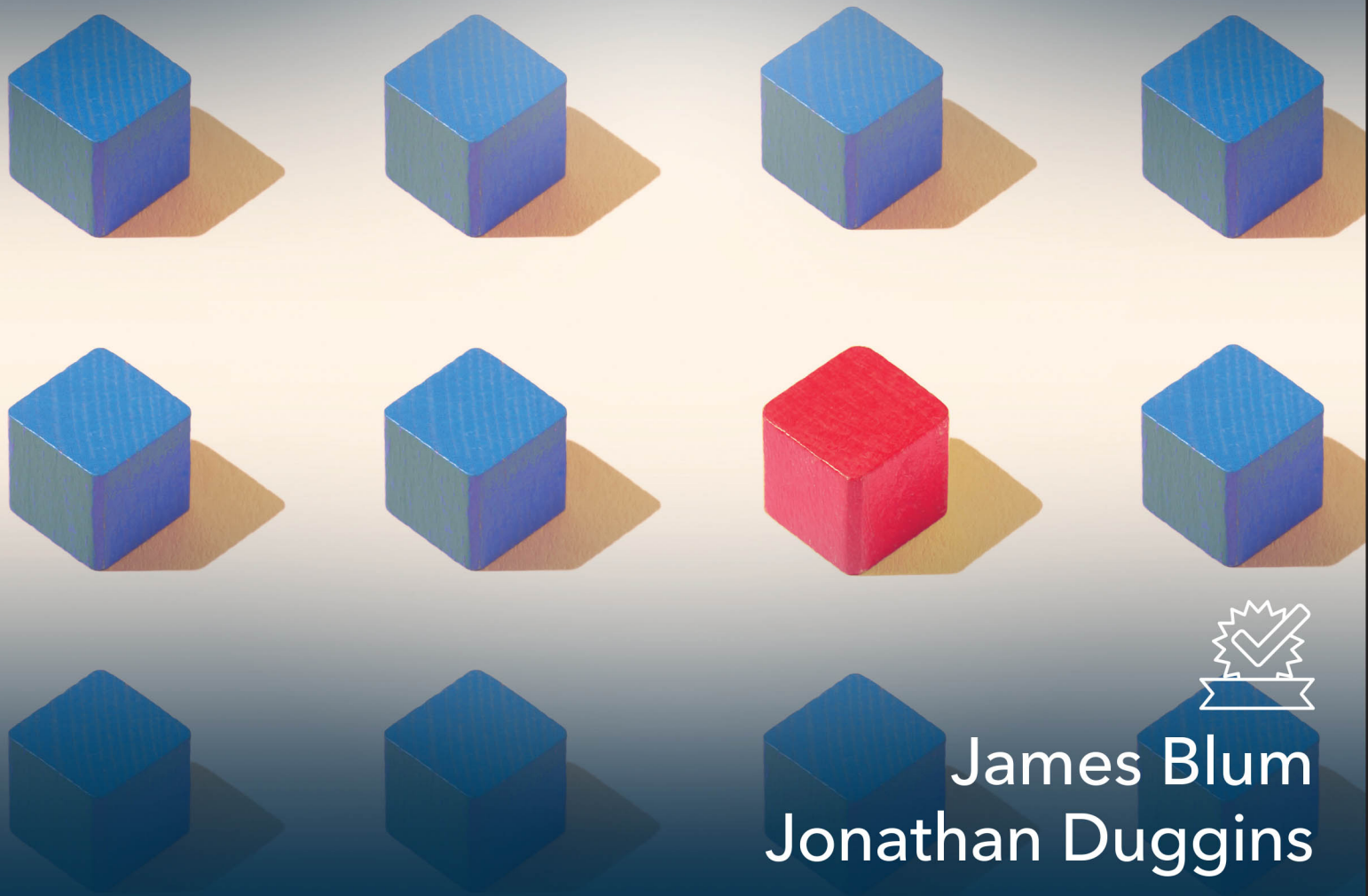
Fundamentals of Programming in SAS®

A Case Studies Approach



James Blum
Jonathan Duggins

The correct bibliographic citation for this manual is as follows: Blum, James and Jonathan Duggins. 2019. *Fundamentals of Programming in SAS®: A Case Studies Approach*. Cary, NC: SAS Institute Inc.

Fundamentals of Programming in SAS®: A Case Studies Approach

978-1-64295-228-5 (Hardcover)
978-1-63526-672-6 (Paperback)
978-1-63526-671-9 (Web PDF)
978-1-63526-669-6 (epub)
978-1-63526-670-2 (kindle)

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

July 2019

Contents

Foreword

To Readers

This book is designed to help you develop an understanding of the SAS programming language and to help you develop good programming practices. It is intended as a learning guide and a skill builder, not as a reference book. To that end, it introduces sets of topics within each chapter that are connected through a single case study. Concepts are introduced on an as-needed basis to complete required tasks, so you are immediately exposed to writing complete programs. As further concepts are introduced they might be new topics, or they might revisit previously introduced topics at a more complex level. This reflects how many of the best SAS programmers have built their talents—by continually adding layers of knowledge onto a base set of skills. The book mimics this type of experience by increasing the complexity of the case study, requiring the addition of newer skills, or more complex versions of earlier skills, as they are needed.

Because of this circling back to content from previous chapters, a pedagogical concept known as a *spiral curriculum*, you will not learn everything this book covers on a topic in any single chapter. Of course, no one book could serve the purpose of giving a complete treatment of all concepts included; therefore, you will often be referred to outside resources, such as SAS Documentation, for a full description of syntax or for more detailed commentary. SAS Documentation is the standardized name used in this text for the collection of help files and examples provided by SAS. This documentation is available via the Help menu in SAS or online. Reading such references is a strategy commonly used by the best SAS programmers to refine their abilities and is an important habit for you to develop to build your skills as a SAS programmer and to expand on those skills in the future.

Due to the introduction of concepts in spiral fashion, it is important to begin with the setup material in Chapter 1 and then to proceed through the book sequentially. For easy reference, the numbering on all output in Chapters 1 through 7 directly corresponds to the number of the program that generates it. However, not all programs generate output. In all chapters, the programs, output, tables, and figures are numbered sequentially within each section. The same case study is used to provide continuity through the narrative when building on earlier concepts. Other case studies are also available for use to build continuity for additional programming activities and exercises. This includes a case study located in Chapter 8 for which the sections are aligned with the learning objectives of each chapter. Additional case studies are available online by visiting the author page for either author.

To Classroom Instructors

As stated in the previous section, this book is designed to tap into some of the best practices in educational theory as it spirals back onto topics throughout your course. It is designed by instructors with over 25 years of combined experience in teaching SAS either in the classroom or in industry. The more technical details are isolated in their own sections so that you can easily include or exclude them to fit the needs of your course. Multiple case studies are also provided so that the case study assignments can be customized for your students' interests and to the content presented in your course. The case study provided in Chapter 8 ensures students have immediate access to a case study for reference while reading the text. Additional case studies are made available through the author pages for either author to ensure you get the benefit of updated materials on a regular basis. Any instructional materials will also be available either via the author pages (for public resources) or by contacting SAS to verify your status as an instructor (for instructor-only materials). These resources will be regularly updated.

About the IPUMS CPS Data

The IPUMS CPS data includes the Integrated Public Use Microdata Series (IPUMS) and Current Population Survey (CPS) beginning in 1962. These data sets provide person- and household-level information about a variety of demographic variables. A cross-section of recent data (2001, 2005, 2010, and 2015) was released for this publication and is included here as the main case study in the narrative. Visit https://cps.ipums.org to learn more about the IPUMS CPS or to extract newer data to continue honing your SAS programming skills.

About This Book

What Does This Book Cover?

This text covers a wide set of topics available in the Base SAS software including:

- DATA step programming including:
  - Reading data sets from non-SAS sources
  - Combining and restructuring SAS data sets
  - Functions and conditional logic
  - DO loops and arrays
- Basic analysis procedures: MEANS, FREQ, CORR, and UNIVARIATE
- Reporting procedures: CONTENTS, PRINT, and REPORT
- Restructuring data with PROC TRANSPOSE
- Visualization with the SGPLOT and SGPANEL procedures
- SAS formats and the FORMAT procedure
- Output Delivery System

While this book covers the foundations of the topics listed above, additional details are often beyond the scope of this text. References are provided for those interested in further study.

Is This Book for You?

Are you trying to learn SAS for the first time? Are you hoping to eventually earn your Base SAS certification and become a SAS Certified Professional? Are you already comfortable with some SAS programming but are looking to hone your skills? If the answer to any of those questions is "yes," then this is the book for you! This book takes a novel approach to learning SAS programming by helping you develop an understanding of the language and establish good programming practices. By following a single case study throughout the text and circling back to previous concepts, this book aids in the learning of new topics through explicit connections to previous material. Just as the best SAS programmers expand their capabilities by continually adding to their already impressive skill sets, as you read this text you will gain the skills and confidence to take on larger challenges with the power of SAS.

This book does not assume any prior knowledge of the SAS programming language. However, an understanding of how file paths function in your operating system is necessary to facilitate the storage and retrieval of data sets, raw data files, and other files such as documents and graphics.

What Should You Know About the Examples?

This book includes tutorials for you to follow to gain hands-on experience with SAS. The majority of the examples are based on a case study using real data. Some examples use subsets of the case study data or introduce smaller data sets to help illustrate a topic. Chapters 2 through 7 contain a wrap-up activity that uses the case study to tie together concepts from the current chapter, and every chapter references another case study contained in Chapter 8 for further practice. You need access to the software listed in the next section to complete the exercises.

For easy reference, the numbering on all output in Chapters 1 through 7 directly corresponds to the number of the program that generated it. However, not all programs generate output. In all chapters, the programs, output, tables, and figures are numbered sequentially within each section.

Software Used to Develop the Book's Content

SAS 9.4TS1M3 and higher were used to develop the examples and exercises. To follow along with the examples simultaneously or to complete the exercises, you only need the Base SAS software except for the portions of Chapters 7 and 8 that use SAS/ACCESS to connect to Microsoft Excel workbooks and Microsoft Access databases.

Example Code and Data

You can access the example code and data for this book by linking to its author page at https://support.sas.com/authors.

SAS University Edition

 This book is compatible with SAS University Edition. If you are using SAS University Edition, then begin here: https://support.sas.com/ue-data.

Where Are the Exercise Solutions?

Readers: Exercise solutions to selected exercises are posted on the author page at https://support.sas.com/authors.

Classroom Instructors: To obtain the full solutions, contact saspress@sas.com.

We Want to Hear from You

Do you have questions about a SAS Press book that you are reading? Contact us at saspress@sas.com.

SAS Press books are written *by* SAS Users *for* SAS Users. Please visit sas.com/books to sign up to request information on how to become a SAS Press author.

We welcome your participation in the development of new books and your feedback on SAS Press books that you are using. Please visit sas.com/books to sign up to review a book

Learn about new books and exclusive discounts. Sign up for our new books mailing list today at https://support.sas.com/en/books/subscribe-books.html.

Learn more about these authors by visiting their author pages, where you can download free book excerpts, access example code and data, read the latest reviews, get updates, and more:
http://support.sas.com/blum
http://support.sas.com/duggins

About These Authors

James Blum is a Professor of Statistics at the University of North Carolina Wilmington where he has developed and taught original courses in SAS programming for the university for nearly 20 years. These courses cover topics in Base SAS, SAS/SQL, SAS/STAT, and SAS macros. He also regularly teaches courses in regression, experimental design, categorical data analysis, and mathematical statistics; and he is a primary instructor in the Master of Data Science program at UNC Wilmington, which debuted in the fall of 2017. He has experience as a consultant on data analysis projects in clinical trials, finance, public policy and government, and marine science and ecology. He earned his MS in Applied Mathematics and PhD in Statistics from Oklahoma State University.

Jonathan Duggins is an award-winning Teaching Professor at North Carolina State University, where his teaching includes multiple undergraduate and graduate programming courses. His experience as a practicing biostatistician influences his classroom instruction, where he incorporates case studies, utilizes large data sets, and holds students accountable for the best practices used in industry. Jonathan is a member of the American Statistical Association and is active with the North Carolina chapter. He has been a SAS user since 1999 and has presented at both regional and national statistical and SAS user group conferences. Jonathan holds a BS and MS in mathematics from the University of North Carolina Wilmington and an MS and PhD in statistics from Virginia Tech.

Learn more about these authors by visiting their author pages, where you can download free book excerpts, access example code and data, read the latest reviews, get updates, and more:
https://support.sas.com/blum
http://support.sas.com/duggins

Chapter 1: Introduction to SAS

1.1 Introduction

This chapter introduces basic concepts about SAS that are necessary to use it effectively. This chapter begins with an introduction to some of the available SAS environments and describes the basic functionality of each. Essentials of coding in SAS are also introduced through some pre-constructed sample programs. These programs rely on several data sets, some provided with SAS, others are provided separately with the textbook, including those that form the basis for the case study used throughout Chapters 2 through 7. Therefore, this chapter also introduces SAS data sets and libraries. In addition, an introduction to debugging code is included, which includes a discussion of the SAS log where notes, warnings, and error messages are provided for any code submitted.

1.2 Learning Objectives

This chapter provides a basis for working in SAS, which is a necessary first step for successful mastery of the material contained in the remainder of this book. In detail, it is expected upon completion of this chapter that the following concepts are understood within the chosen SAS environment:

- Demonstrate the ability to open, edit, save, and submit a SAS program
- Apply the LIBNAME statement to create a user-defined library—including the BookData library that contains all files for this text, downloadable from the Author Page
- Demonstrate the ability to navigate through libraries and view data sets
- Think critically about all messages SAS places in the log to determine their cause and severity
- Apply ODS statements to manage output and output destinations
- Explain the basic rules and structure of the SAS language
- Demonstrate the ability to apply a template to customize output

Use the concepts of this chapter to solve the problems in the wrap-up activity. Additional exercises and case-studies are also available to test these concepts.

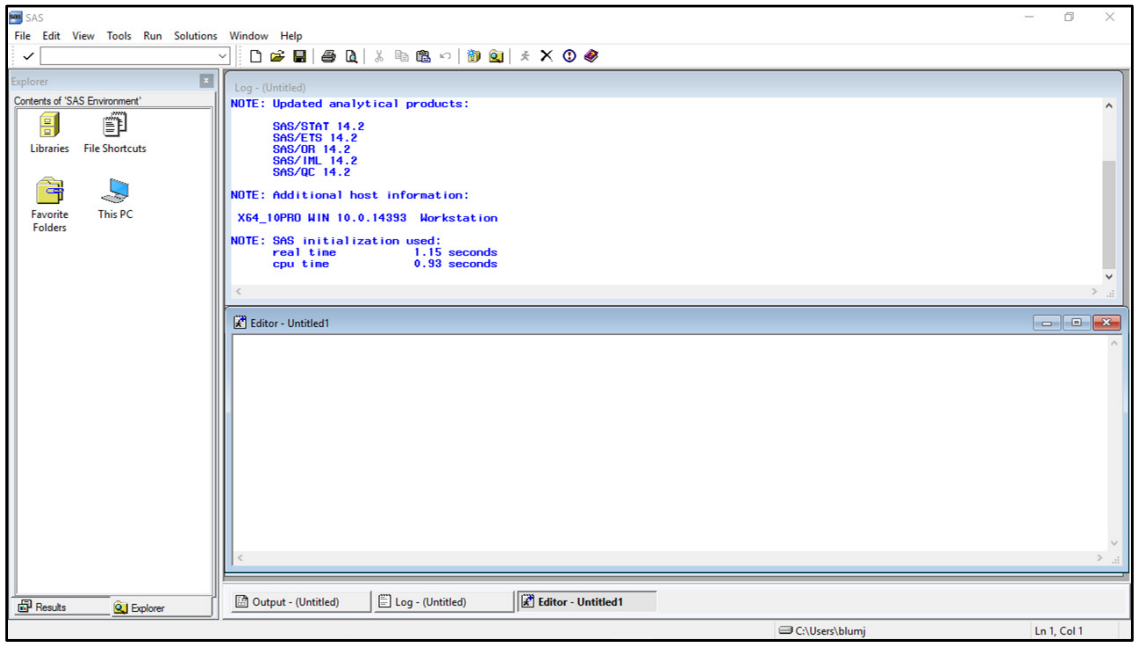
1.3 SAS Environments

Interacting with SAS is possible in a variety of environments, including SAS from the command line, the SAS windowing environment, SAS Enterprise Guide, SAS Studio, and SAS University Edition; with most of these being available on multiple operating systems. This chapter introduces the SAS windowing environment, SAS Studio, and SAS University Edition on the Microsoft Windows operating system and points out key differences between those SAS environments. For further specifics on differences across SAS environments and operating systems, consult the appropriate SAS Documentation. In nearly all examples in this book, code is given outside of any specific environment and output is shown in generic RTF-style tables or standard image formats. Output may vary somewhat from the default styles across SAS environments on various operating systems, and examples later in this chapter demonstrate some of these differences. Later chapters give information about how to duplicate the table styles.

1.3.1 The SAS Windowing Environment

The SAS windowing environment is shown in Figure 1.3.1 with three windows visible: Log, Explorer, and Editor (commonly referred to as the Enhanced Program Editor). The Results and Output windows are two other windows commonly available by default, but are typically obfuscated by other windows at launch. When code that generates output is executed, these windows (and possibly others) become relevant.

Figure 1.3.1: SAS Windowing Environment on Microsoft Windows



In the Microsoft Windows operating system, the menu and toolbars in the SAS windowing environment have a similar look and feel compared to other programs running on Windows. Exploring the menus reveals standard options under the **File**, **Edit**, and **Help** menus (such as **Open, Save, Clear, Find**). The **View, Tools, Solutions,** and **Window** menus have specialized options related to windows and utilities that are specific to SAS. The **Run** menu is dedicated to submissions of SAS code, including submissions to a remote session. As is typical in most applications, toolbar buttons in SAS provide quick access to common menu functions and vary depending on which window is active in the session. Some menu and toolbar options are reviewed below during the execution of the supplied sample code given in Program 1.3.1. This sample code is available from the author web pages for this book.

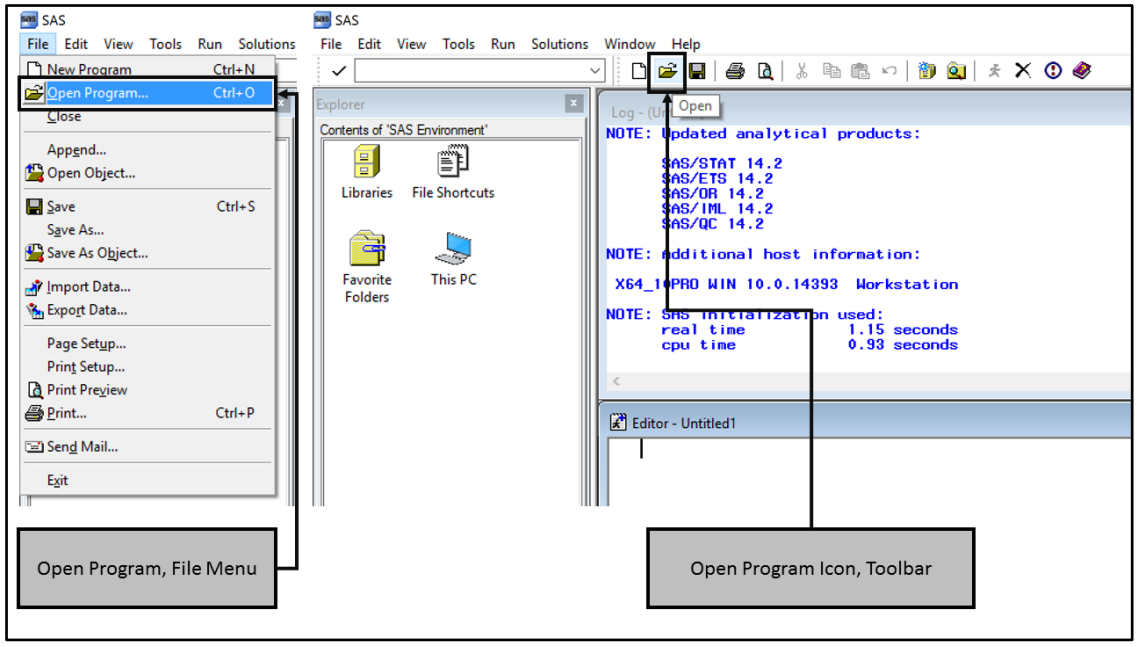
Program 1.3.1: Demonstration Code

```
options ps=100 ls=90 number pageno=1 nodate;

data work.cars;
  set sashelp.cars;

  mpg_combo=0.6*mpg_city+0.4*mpg_highway;
```

```
  select(type);
    when('Sedan','Wagon') typeB='Sedan/Wagon';
    when('SUV','Truck') typeB='SUV/Truck';
    otherwise typeB=type;
  end;

  label mpg_combo='Combined MPG' typeB='Simplified Type';
run;

title 'Combined MPG Means';
proc sgplot data=work.cars;
  hbar typeB / response=mpg_combo stat=mean limits=upper;
  where typeB ne 'Hybrid';
run;

title 'MPG Five-Number Summary';
title2 'Across Types';
proc means data=cars min q1 median q3 max maxdec=1;
  class typeB;
  var mpg:;
run;
```

After downloading the code to a known directory, there are multiple ways to navigate to and open this code. Figure 1.3.2 shows two methods to open the file, each requiring the Editor window to be active.

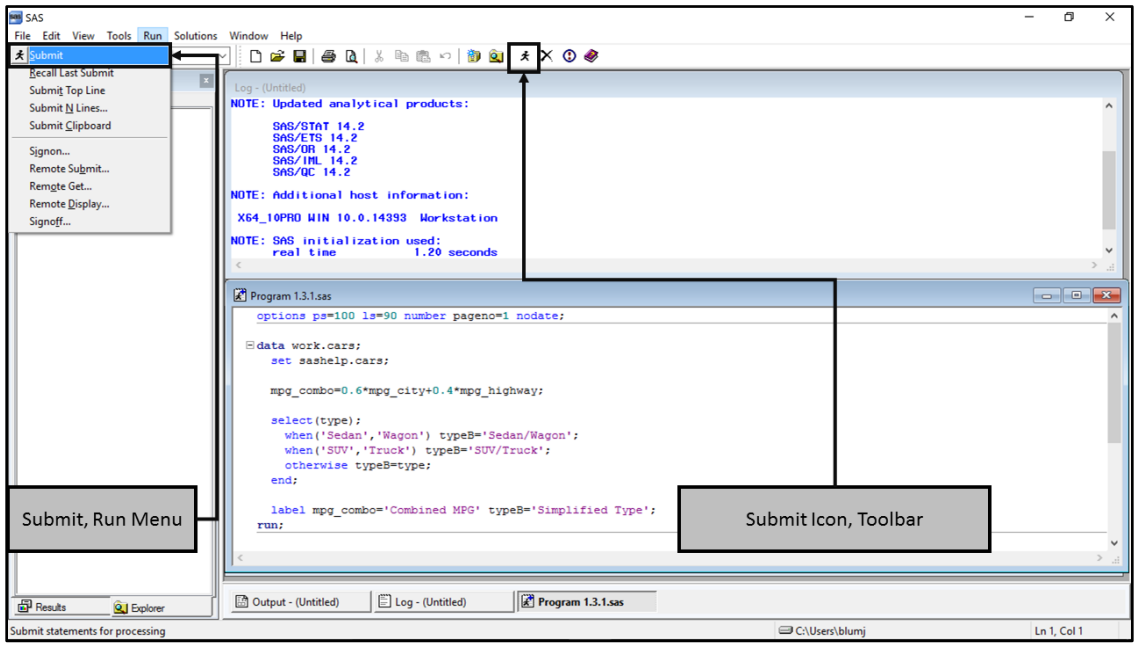
Figure 1.3.2: Methods for Opening SAS Code Files in the SAS Windowing Environment



Either of these choices launches a standard Microsoft Windows file selection window, which is used to navigate to and select the file of interest. Upon successful selection of the code, it appears in the Editor window, and is displayed with some color coding as shown in Figure 1.3.3 (assuming the Enhanced Program Editor is in use, the Program Editor window provides different color coding). It is not important to understand the specific syntax or how the code works at this point, for now it is used simply to provide an executable program to introduce some SAS fundamentals.

Code submission can also occur in multiple ways, two of which are shown in Figure 1.3.3, again each method requires the Editor window to be the active window in the session. If multiple Editor windows are open, only code from the active window is submitted.

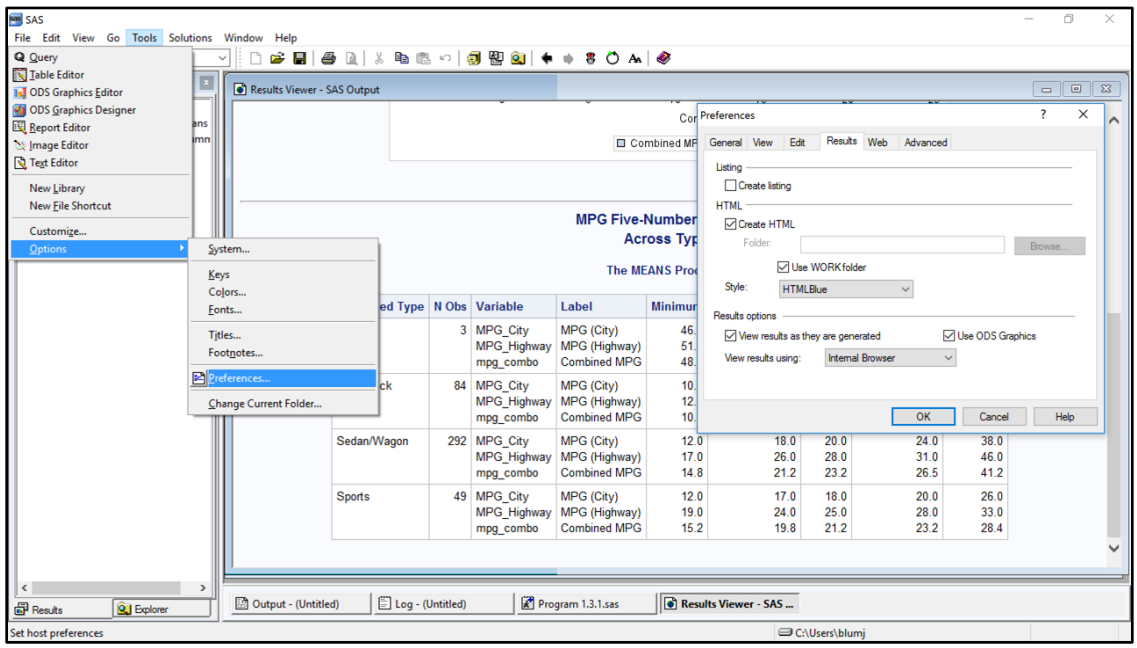
Figure 1.3.3: Submitting SAS Code in the SAS Windowing Environment



Typically, after any code submission the Results window activates and displays an index of links to various entities produced by the program, including output tables. While not all SAS code generates output, Program 1.3.1 does, and it may be routed to different destinations (and possibly more than one destination simultaneously) depending on the version of SAS in use and current option settings.

In SAS 9.4, the default settings route output to an HTML file which is displayed in the Results Viewer, a viewing window internal to the SAS session. Previous versions of SAS rely on the Output window for tables, an option which remains available for use in the SAS 9.4 windowing environment, and other specialized destinations for graphics. Default output options can be set by navigating to the **Tools** menu, selecting **Options**, followed by **Preferences** from that sub-menu, and choosing the **Results** tab in the window that appears, as shown in Figure 1.3.4.

Figure 1.3.4: Managing Output for Program Submissions



Among other options, Figure 1.3.4 shows the option for **Create HTML** checked and **Create Listing** unchecked. For tables, the listing destination is the Output window, so when **Create Listing** is checked, tables also appear

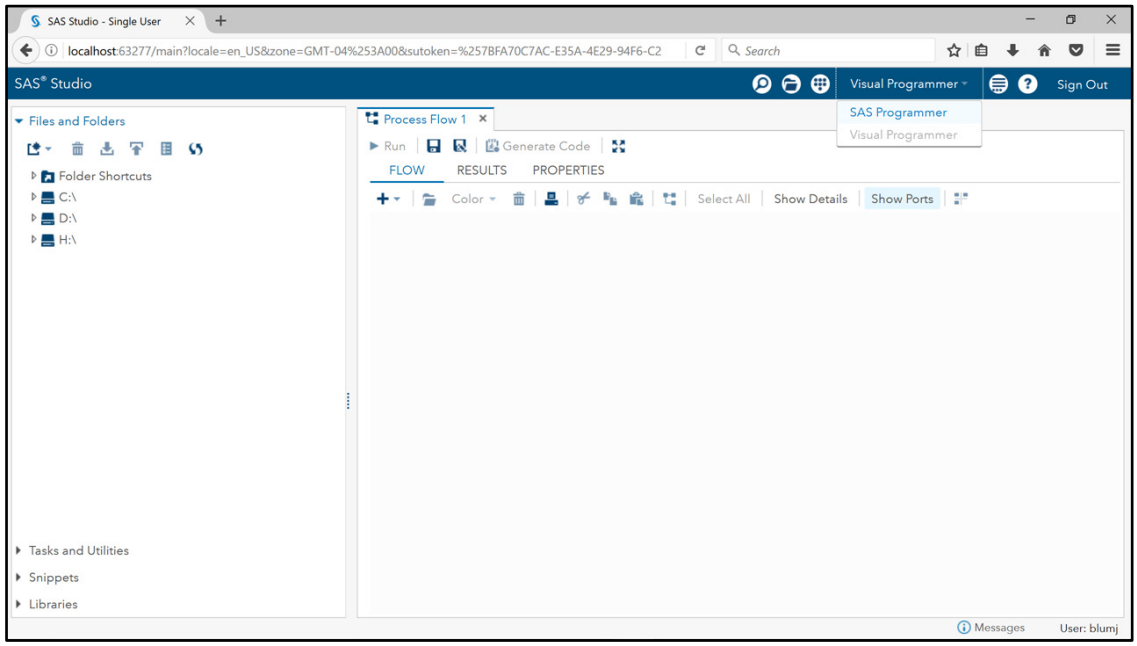
in the Output window in what appears as a plain text form. It is possible to check both boxes, and it is also possible to check neither, whichever is preferred.

In the remainder of this book, output tables are shown in an RTF form embedded inside the book text, outside of any SAS Results window. Appearance of output tables and graphs in the book is similar to what is produced by a SAS session, but is not necessarily identical when default session options are in place. Later in this chapter, the ability to use SAS code to control delivery of output to each of these destinations is demonstrated, along with use of the listing destination as an output destination for graphics files.

1.3.2 SAS Studio and SAS University Edition

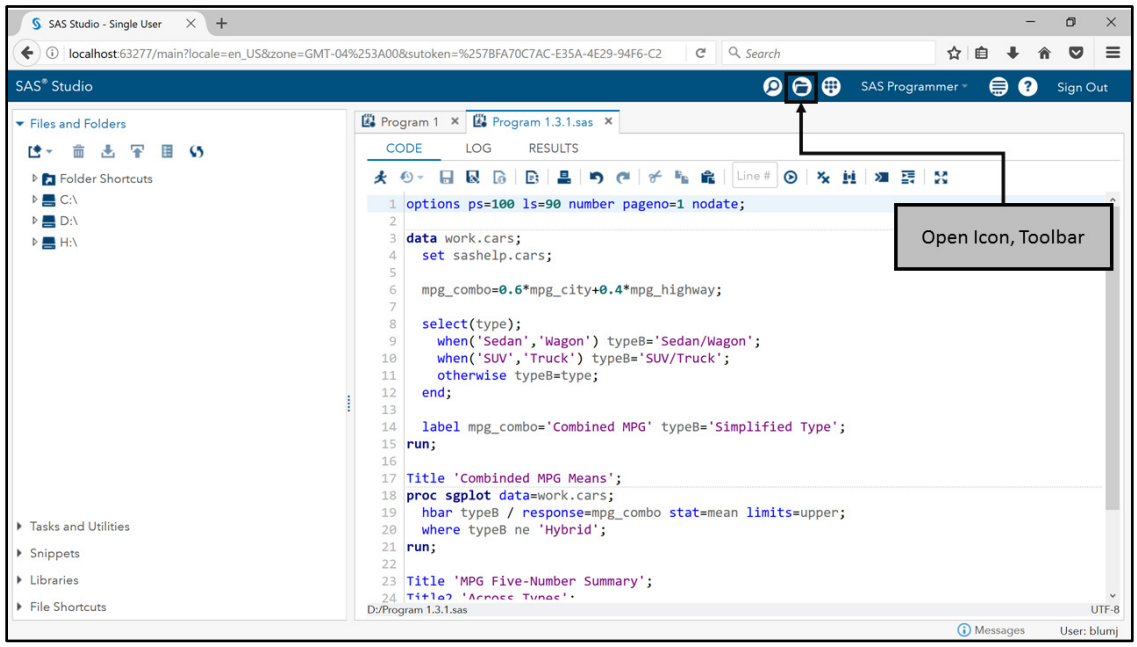
SAS Studio and SAS University Edition (which are, for the remainder of this text, singularly referred to as SAS University Edition) interface with SAS through a web browser. Typically, the browser used is the default browser for the machine hosting the SAS University Edition session, but this is not a requirement. Figure 1.3.5 shows a typical result of launching SAS University Edition (in this case using the Firefox browser on Microsoft Windows), launching in visual programmer mode by default. A closer match to the structure of the SAS windowing environment is provided by selecting SAS Programmer from the toolbar as shown.

Figure 1.3.5: SAS University Edition



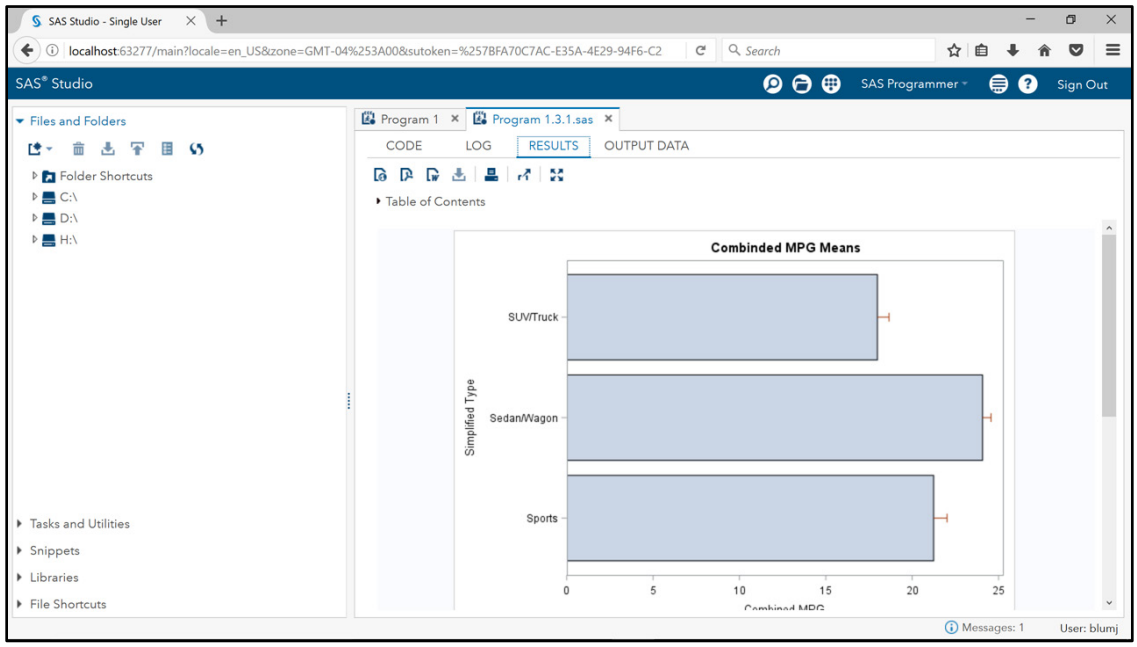
Opening a program is accomplished via the **Open** icon on the toolbar, as illustrated in Figure 1.3.6, and the opened code is displayed in a manner very similar to the that of the Enhanced Program Editor display shown in Section 1.3.1.

Figure 1.3.6: Opening a Program in SAS University Edition



Though in a different position, the toolbar icon for submission is the same as in the SAS windowing environment, and selecting it produces output in the Results tab as shown in Figure 1.3.7.

Figure 1.3.7: Execution of a Program in SAS University Edition



A few important differences to note in SAS University Edition: first, the output is displayed starting at the top, rather than at the bottom of the page as in the SAS windowing environment. Second, there is an additional tab for Output Data in this session. In Section 1.4.3, libraries, data sets, and navigation to each are discussed; however, SAS University Edition also includes a special tab whenever a program generates new data sets, which aids in directly viewing those results. Finally, note that the Code, Log, Results, and Output Data tabs are contained within the Program 1.3.1 tab, and each program opened is given its own set of tabs. In contrast, the SAS windowing environment supports multiple Editor windows in a single session, but they all share a common Log window, Output window, and (under default conditions) output HTML file. As discussed in other examples and in Chapter Notes 1 and 2 in Section 1.7, submissions from any and all Editor windows in the SAS

windowing environment are cumulative in the Log and Output windows; therefore, managing results in each environment is quite different.

1.4 SAS Fundamentals

To build an initial understanding of how to work with programs in SAS, Program 1.3.1 is used repeatedly in this section to introduce various SAS language elements and concepts. For both SAS windowing environment and SAS University Edition, the features of each environment and navigation within them are discussed in conjunction with the language elements that relate to them.

1.4.1 SAS Language Basics

Program 1.4.1 is a duplicate of Program 1.3.1 with certain elements noted numerically throughout the code, followed by notes on the specific code in the indicated position. Throughout this book, this style is used to detail important features found in sample code.

Program 1.4.1: Program 1.3.1, Revisited

```
options ps=100 ls=90 number pageno=1 nodate;❶

data work.cars;❷
  set sashelp.cars;

  MPG_Combo=0.6*mpg_city+0.4*mpg_highway;

  select(type);
    when('Sedan','Wagon') TypeB='Sedan/Wagon';
    when('SUV','Truck') TypeB='SUV/Truck';
    otherwise TypeB=type;
  end;

  label mpg_combo='Combined MPG' typeB='Simplified Type';
 run;

title 'Combined MPG Means';❶
proc sgplot data=work.cars;❸
  hbar typeB / response=mpg_combo stat=mean limits=upper;
  where typeB ne 'Hybrid';
run;

title 'MPG Five-Number Summary';❶
title2 'Across Types';❶
proc means data=work.cars min q1 median q3 max maxdec=1;❸
  class typeB;
  var mpg:;❹
run;❺
```

❶ SAS code is written in statements, each of which ends in a semicolon. The statements indicated here (OPTIONS and TITLE) are examples of global statements. Global statements are statements that take effect as soon as SAS compiles those statements. Typically, the effects remain in place during the SAS session until another statement is submitted that alters those effects.

❷ The SAS DATA step has a variety of uses; however, it is primarily a tool for creation or manipulation of data sets. A DATA step is generally comprised of several statements forming a block of code, ending with the RUN statement, the role of which is described in ❹.

❸ Procedures in SAS are used for a variety of tasks and, like the DATA step, are generally comprised of several statements. These are generically referred to as PROC steps.

❹ The PROC MEANS result includes the variables MPG_City, MPG_Highway, and MPG_Combo even though none of these are explicitly written in the procedure code. The colon (:) at the end of a variable name acts as a wildcard indicating that any variable name starting with the prefix given is part of the designated set, this shortcut is known in SAS as a name prefix list. For other types of variable lists, see Chapter Note 3 in Section 1.7.

- With DATA and PROC steps defined as blocks of code, each of these blocks is terminated with a step-boundary. The RUN statement is a commonly used as a step boundary, though it is not required for each DATA or PROC step. See Section 1.4.2 for details.

1.4.2 SAS DATA and PROC Steps

SAS processing of code submissions includes two major components: compilation and execution. In some cases, individual statements are compiled and take effect immediately, while at other times, a series of statements is compiled as a set and then executed after the complete set is processed by the compiler. In general, statements that compile and take effect individually and immediately are global statements. Statements that compile and execute as a set are generally referred to as steps, with the SAS language including both DATA steps and procedure (or PROC) steps.

The DATA step starts with a DATA statement, and a PROC step starts with a PROC statement that includes the name of the procedure, and all steps end with some form of a step boundary. As noted in Program 1.4.1, a commonly used step boundary in the SAS language is the RUN statement, but it is technically not required for each step. Any invocation of any DATA or PROC step is also defined as a step boundary due to the fact that DATA and PROC steps cannot be directly nested together in the SAS language. In general, it is considered a good programming practice to explicitly provide a statement for the step boundary, rather than implicitly through invocation of a DATA or PROC step. The code submissions in Figure 1.4.1 and Program 1.4.2 provide illustrations of the advantages of explicitly defining the end of a step.

In either the SAS windowing environment or SAS University Edition, portions of code can be compiled and executed by highlighting that section and then submitting. Having clear definitions from beginning to end for any DATA or PROC step aids in the ability to submit portions of code, which can be accomplished by using the RUN statement as an explicit step boundary. Figures 1.4.1A and 1.4.1B show submissions of the two PROC steps from Program 1.4.1 along with their associated TITLE statements.

Figure 1.4.1A: Submitting Portions of Code in SAS University Edition

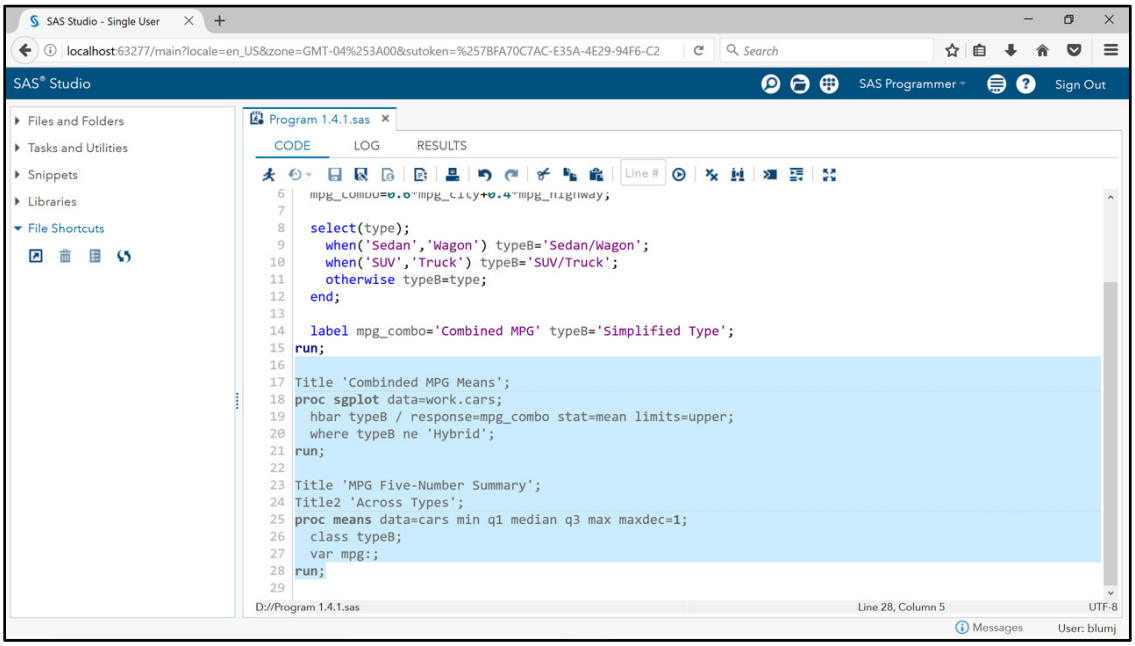
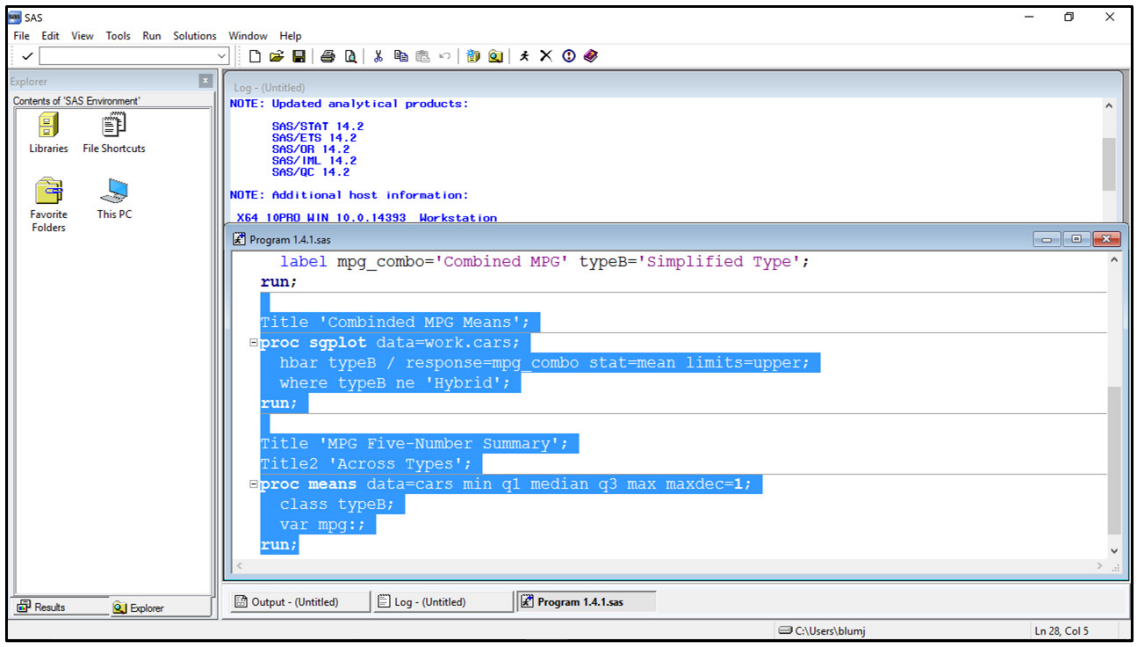


Figure 1.4.1B: Submitting Portions of Code in the SAS Windowing Environment



This submission reproduces the bar chart and the table of statistics produced previously in Figure 1.3.7. However, notice that the result is somewhat different in the SAS windowing environments and SAS University Edition. In the SAS windowing environment, the output is added to the output from the previous submission (and the log from this submission is also added to the previous log information). In SAS University Edition, the output is replaced, and the sub-tab for Output Data is not present because the DATA step did not run. With default settings in place, submissions are cumulative for both log and output in SAS windowing environment; conversely, replacement is the default in SAS University Edition. For more information about managing results in either environment, see Chapter Note 1 in Section 1.7.

Program 1.4.2 shows the code portion submitted in Figure 1.4.1 with the first RUN statement removed. Delete the RUN statement and re-submit the selection, review the output (Figure 1.4.2) and details below for another example of why explicitly ending steps in SAS is a good programming practice.

Program 1.4.2: Multiple Steps Without Explicit Step Boundaries

```
title 'Combined MPG Means';❶
proc sgplot data=work.cars;❷
  hbar typeB / response=mpg_combo stat=mean limits=upper;
  where typeB ne 'Hybrid';
❸

title 'MPG Five-Number Summary';
title2 'Across Types';❹
proc means data=work.cars min q1 median q3 max maxdec=1;❺
  class typeB;
  var mpg:;
run;
```

❶ The first statement compiled and executed is this TITLE statement, which assigns the quoted/literal value as the primary title line.

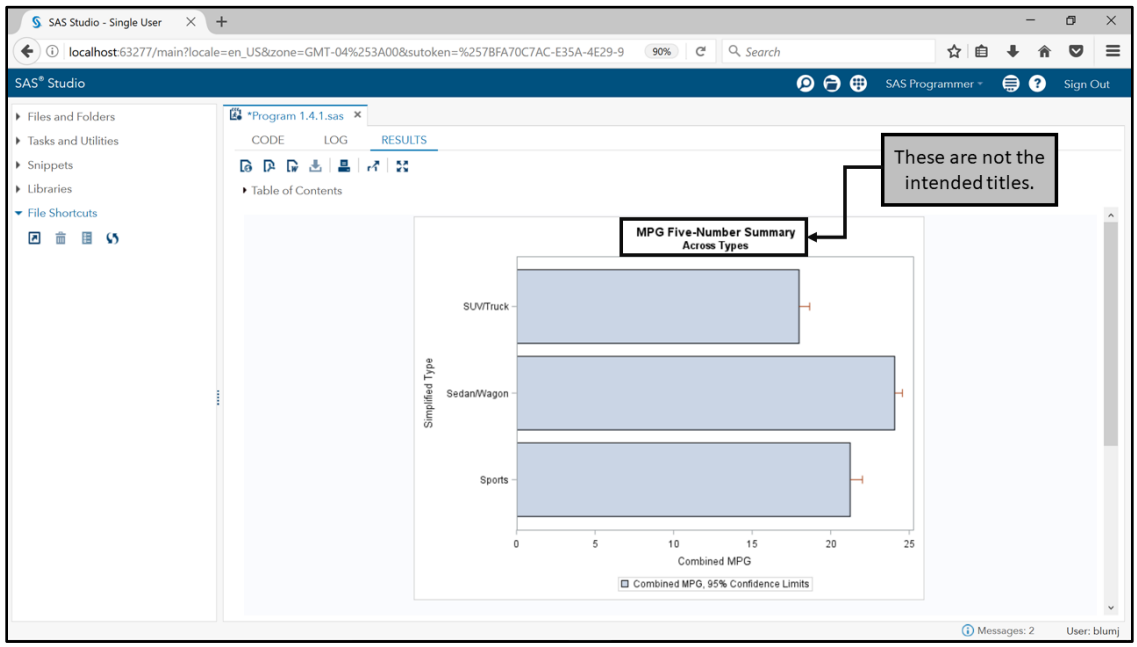
❷ The SGPLOT procedure is invoked for compilation and execution by this statement. Subsequent statements are compiled as part of the SGPLOT step until a step boundary is reached.

❸ This is the position of the RUN statement in Program 1.4.1 and, when it is compiled in that program, it signals the end of the SGPLOT step. Assuming no errors, PROC SGPLOT executes at that point; however, with no RUN statement present in this code, compilation of the SGPLOT step is not complete and execution does not begin.

❹ These two TITLE statements, which are global, now compile and take effect. Since the SGPLOT procedure still has not completed compilation, nor started execution, this TITLE statement replaces the first title line assigned in ❶.

❺ This statement starts the MEANS procedure which, due to the fact that steps cannot be nested, indicates that the SGPLOT statements are complete. Compilation of the SGPLOT step ends and it is executed, with the titles in ❹ now placed erroneously on the graph.

Figure 1.4.2: Failing to Define the End of a Step



In any interactive session, the final step boundary must be explicitly stated. For a discussion of the differences between interactive and non-interactive sessions in the SAS windowing environment and SAS University Edition, see Chapter Note 2 in Section 1.7.

The remainder of Program 1.4.1 is a DATA step, which is shown as Program 1.4.3 with a few details about its operation highlighted. The DATA step is a powerful tool for data manipulation, offering a variety of functions, statements, and other programming elements. The DATA step is of such importance that it is featured in every chapter of this book.

Program 1.4.3: DATA Step from Program 1.4.1

```
data work.cars❶;
  set sashelp.cars❶;

  MPG_Combo=0.6*mpg_city+0.4*mpg_highway;❷

  select(type);
    when('Sedan','Wagon') TypeB='Sedan/Wagon';
    when('SUV','Truck') TypeB='SUV/Truck';
    otherwise TypeB=type;
  end;❸

  label mpg_combo='Combined MPG' typeB='Simplified Type';❹
run;
```

❶ The DATA statement that opens this DATA step names a SAS data set Cars in the Work library. Work.Cars is populated using the SAS data set referenced in the SET statement, also named Cars and located in the Sashelp library. Data set references are generally two-level references of the form `library.dataset`. The exception to this is the Work library, which is taken as the default library if only a data set name is provided. Details on navigating through libraries and data sets are given in Section 1.4.3.

❷ MPG_Combo is a variable defined via an arithmetic expression on two of the existing variables from the Cars data set in the Sashelp library. Assignments of the form `variable = expression;` do not require any explicit declaration of variable type to precede them, the compilation process determines the appropriate variable type from the expression itself. SAS data set variables are limited to two types: character or numeric.

❸ The variable TypeB is defined via assignment statements chosen conditionally based on the value of the Type variable. The casing of the literal values is an exact match for the casing in the data set as shown subsequently in Figures 1.4.4 and 1.4.5—matching of character values includes all casing and spacing. Various forms of conditional logic are available in the DATA step.

❹ Naming conventions in SAS generally follow three rules, with some exceptions noted later. Names are permitted to include only letters, numbers, and underscores; must begin with a letter or underscore; and are limited to 32 characters. Given these naming limitations, labels are available to provide more flexible descriptions for the variable. (Labels are also available for data sets and other entities.) Also note that references to the variables MPG_Combo and TypeB use different casing here than in their assignment expressions; in general, the SAS language is not case-sensitive.

1.4.3 SAS Libraries and Data Sets

Program 1.4.1 involves data sets in each of its programming steps. The DATA step uses one data set as the foundation for creating another, and the data set it creates is used in each of the PROC steps that follow. Again, data set references are generally in a two-level form of `library.dataset`, other than the exception for the Work library noted in the discussion of Program 1.4.3. The PROC steps in Program 1.4.1 each use one of the possible forms to reference the Cars data located in the Work library.

Navigation to data sets in various libraries is possible in either the SAS windowing environment or SAS University Edition. In the SAS windowing environment, the Explorer window permits navigation to any assigned library, while in SAS University Edition, the left panel contains a section for libraries. In either setting, opening a library potentially reveals a series of table icons representing various SAS data sets, which can be opened to view the contents of the data. As an example, navigation to and opening of the Cars data set in the Sashelp library is shown below for each of the SAS windowing environment and SAS University Edition. Figures 1.4.3 and 1.4.4 demonstrate one way to open the Cars data in the SAS windowing environment.

Figure 1.4.3: Starting Points for Library Navigation, SAS Windowing Environment

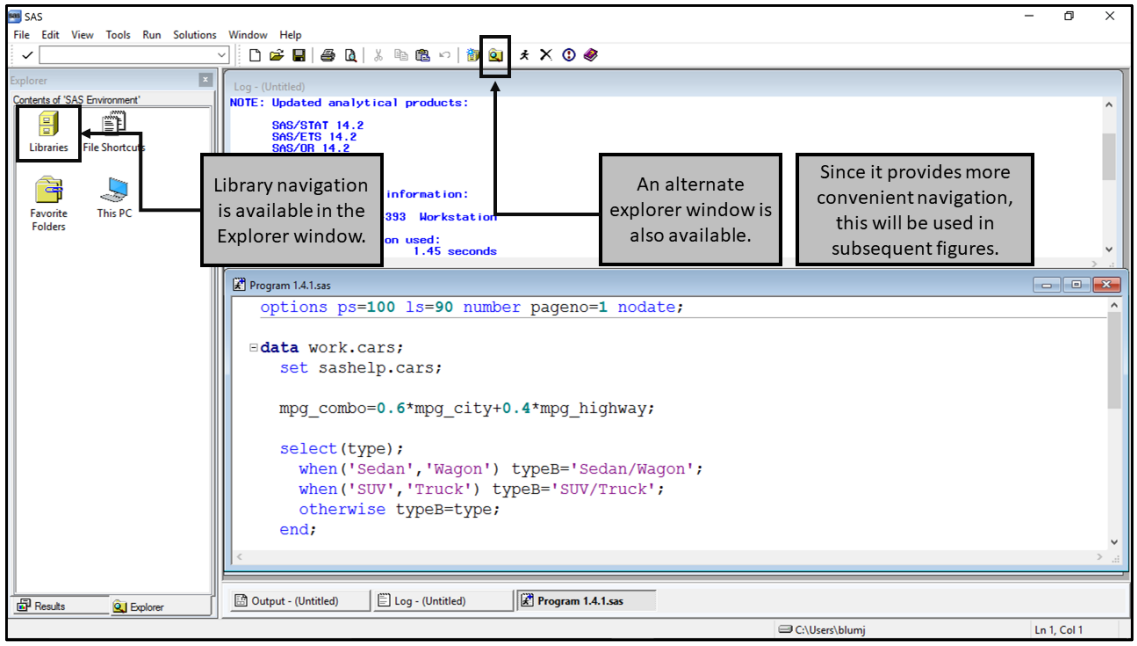


Figure 1.4.4: Accessing the Cars Data Set in the Sashelp Library in the Windowing Environment

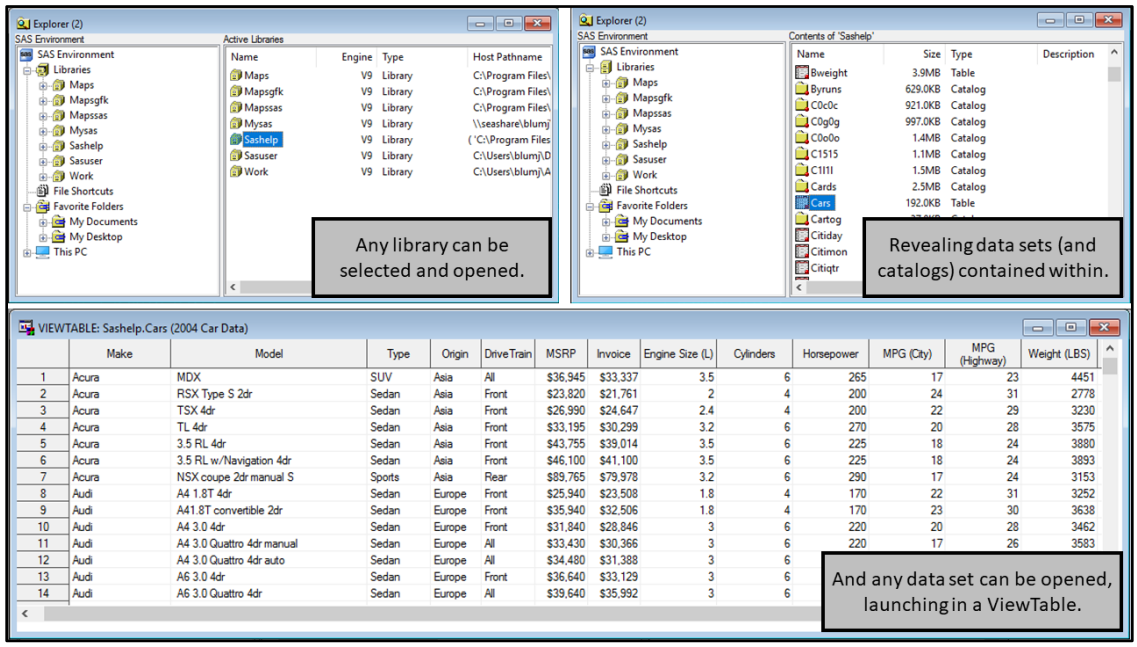
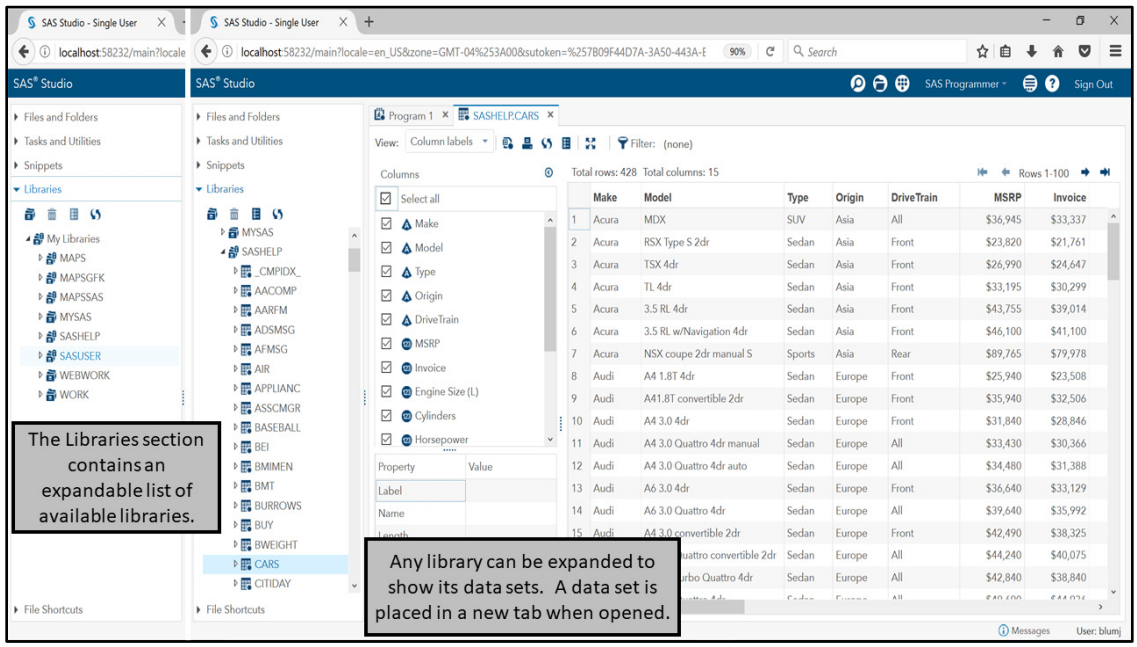


Figure 1.4.5 shows how to open the Cars data set in a SAS University Edition session, revealing several differences in the library navigation and the data view, which opens in a separate tab in the University Edition session.

Figure 1.4.5: Accessing the Cars Data Set in the Sashelp Library in University Edition



In the SAS windowing environment, options for the data view are driven by menus and toolbar buttons for the active window, while in SAS University Edition, each data set tab contains a set of buttons and menus in its toolbar. As part of this tab, SAS University Edition also offers boxes to select a subset of variables and gives properties for each variable as it is selected. Such changes are possible in the SAS windowing environment as well, but are menu-driven. For more information, see the SAS Documentation for the chosen environment.

Another major difference between the two data views is that the ViewTable in the SAS windowing environment has active control over the data set selected. The view in SAS University Edition is generated when the data set is opened, or re-generated if new options are selected, and control of the data set is released. For further detail on the implications of these differences, see Chapter Note 4 in Section 1.7.

Though there are ultimately several different forms of SAS libraries, the most basic simply assigns a library reference (or *libref*) to a folder which the SAS session can access. A library can be assigned in a program via the LIBNAME statement or through other tools available in the SAS windowing environment or SAS University Edition. In order to use this book, it is essential to assign library references to the data sets downloaded from the author web pages. Figures 1.4.6 and 1.4.7 show an assignment of a library named BookData to an assumed location. The path must be set to the actual location of the downloaded files, and the choice of library name must follow the naming conventions given previously in Program 1.4.3, with the additional restriction that the library reference is limited to 8 characters.

Figure 1.4.6: Assigning a Library in the SAS Windowing Environment

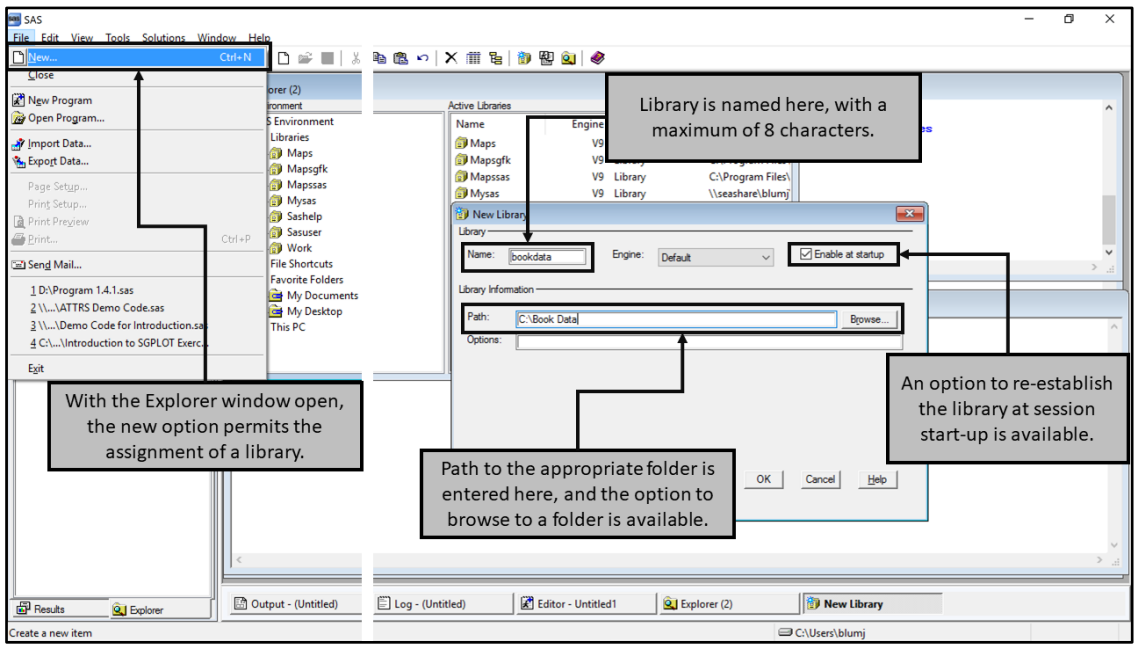
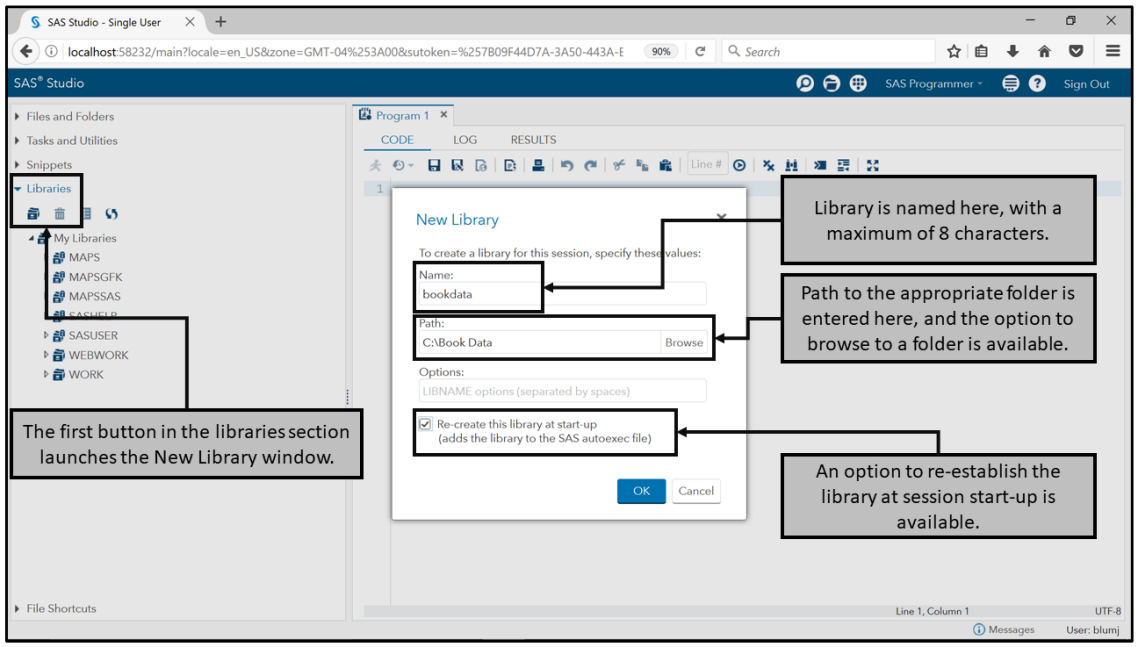


Figure 1.4.7: Assigning a Library in SAS University Edition



Submitting the following LIBNAME statement is equivalent to the assignments shown in the Figures 1.4.6 and 1.4.7, except for the fact that the assigned library is not re-created at start-up of the next session.

```
libname bookdata 'C:\Book Data';
```

Any of these assignments creates what is known as a permanent library, meaning that data sets and other files stored there remain in place until an explicit modification is made to them. Temporary libraries are expunged when the SAS session ends—in the SAS Windowing environment, Work is a temporary library; in SAS University Edition, Work and Webwork are temporary.

The PRINT and CONTENTS procedures provide information about data and metadata as program output. Program and Output 1.4.4 provide a demonstration of their use.

Program 1.4.4: Using the CONTENTS and PRINT Procedures to Display Metadata and Data

```
proc contents data=sashelp.cars;❶
run;

proc print data=sashelp.cars(obs=10) label;❷
  var make model msrp mpg_city mpg_highway;❸
run;
```

❶ The CONTENTS procedure output shows a variety of metadata, including the number of variables, number of observations, and the full set of variables and their attributes. Adding the option VARNUM to the PROC CONTENTS statement reorders the variable attribute table in column order—default display is alphabetical by variable name. The keyword _ALL_ can be used in place of the data set name, in this instance, the output contains a full list of all library members followed by metadata for each data set in the library.

❷ The PRINT procedure directs the data portion of the selected data set to all active output destinations. By default, PROC PRINT displays variable names as column headers, the LABEL option changes these to the variable labels (when present). Display of labels is also controlled by the LABEL/NOLABEL system options, see Chapter Note 5 in Section 1.7 for additional details.

❸ Default behavior of the PRINT procedure is to output all rows and columns in the current data order. The VAR statement selects the set of columns and their order for display.

Output 1.4.4A: Output from PROC CONTENTS for Sashelp.Cars

Data Set Name	SASHELP.CARS	**Observations**	428
Member Type	DATA	**Variables**	15
Engine	V9	**Indexes**	0
Created	***Local Information Differs***	**Observation Length**	152
Last Modified	***Local Information Differs***	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	YES
Label	2004 Car Data		
Data Representation	WINDOWS_64		
Encoding	us-ascii ASCII (ANSI)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	2
First Data Page	1
Max Obs per Page	430
Obs in First Data Page	413

Engine/Host Dependent Information	
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	***Local Information Differs***
Release Created	9.0401M4
Host Created	X64_SR12R2
Owner Name	BUILTIN\Administrators
File Size	192KB
File Size (bytes)	196608

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
9	Cylinders	Num	8		
5	DriveTrain	Char	5		
8	EngineSize	Num	8		Engine Size (L)
10	Horsepower	Num	8		
7	Invoice	Num	8	DOLLAR8.	
15	Length	Num	8		Length (IN)
11	MPG_City	Num	8		MPG (City)
12	MPG_Highway	Num	8		MPG (Highway)
6	MSRP	Num	8	DOLLAR8.	
1	Make	Char	13		
2	Model	Char	40		
4	Origin	Char	6		
3	Type	Char	8		
13	Weight	Num	8		Weight (LBS)
14	Wheelbase	Num	8		Wheelbase (IN)

Sort Information	
Sortedby	Make Type
Validated	YES
Character Set	ANSI

Output 1.4.4B: Output from PROC PRINT (First 10 Rows) for Sashelp.Cars

Obs	Make	Model	MSRP	MPG (City)	MPG (Highway)
1	Acura	MDX	$36,945	17	23
2	Acura	RSX Type S 2dr	$23,820	24	31

Obs	Make	Model	MSRP	MPG (City)	MPG (Highway)
3	Acura	TSX 4dr	$26,990	22	29
4	Acura	TL 4dr	$33,195	20	28
5	Acura	3.5 RL 4dr	$43,755	18	24
6	Acura	3.5 RL w/Navigation 4dr	$46,100	18	24
7	Acura	NSX coupe 2dr manual S	$89,765	17	24
8	Audi	A4 1.8T 4dr	$25,940	22	31
9	Audi	A41.8T convertible 2dr	$35,940	23	30
10	Audi	A4 3.0 4dr	$31,840	20	28

1.4.4 The SAS Log

The SAS log tracks program submissions and generates information during compilation and execution to aid in the debugging process. Most of the information SAS displays in the log (besides repeating the code submission) falls into one of five categories:

- *Errors*: An error in the SAS log is an indication of a problem that has stopped the compilation or execution process. These may be generated by either syntax or logic errors, see the example in this section for a discussion of the differences in these two error types.
- *Warnings*: A warning in the SAS log is an indication of something unexpected during compilation or execution that was not sufficient to stop either from occurring. Most warnings are an indication of a logic error, but they can also reflect other events, such as an attempt by the compiler to correct a syntax error.
- *Notes*: Notes give various information about the submission process, including: process time, records and data set used, locations for file delivery, and other status information. However, some notes actually indicate potential problems during execution. Therefore, reviewing notes is important, and they should not be presumed to be benign. Program 1.4.5, along with others in later chapters, illustrates such an instance.
- *Additional Diagnostic Information*: Depending on the nature of the note, error, or warning, SAS may transmit additional information to the log to aid in diagnosing the problem.
- *Requested Information*: Based on various system options and other statements, a SAS program can request additional information be transmitted to the SAS log. The ODS TRACE statement is one such statement covered in Section 1.5. Other statements and options are included in later chapters.

Program 1.4.5 introduces errors into the code given in Program 1.4.1, with a review of the nature of the mistakes and the log entries corresponding to them shown in Figure 1.4.8. Errors can generally be split into two types: syntax and non-syntax errors. A syntax error occurs when the compiler is unable to recognize a portion of the code as a legal statement, option, or other language element; thus, it is a situation where programming statements do not conform to the rules of the SAS language. A non-syntax error occurs when correct syntax rules are used, but in a manner that leads to an incorrect result (including no result at all). In this book, non-syntax errors are also referred to as logic errors (an abbreviated phrase referring to errors in programming logic). Chapter Note 6 in Section 1.7 provides a further refinement of such error types.

Program 1.4.5: Program 1.4.1 Revised to Include Errors

```
options pagesize=100 linesize=90 number pageno=1 nodate;

data work.cars;
  set sashelp.cars;

  mpg_combo=0.6*mpg_city+0.4*mpg_highway;
```

```
  select(type);
    when('Sedan','Wagon') typeB='Sedan/Wagon';
    when('SUV','Truck') typeB='SUV/Truck';
    otherwise typeB=type;
  end;

  label mpg_combo='Combined MPG' type2❶='Simplified Type';
run;

Title 'Combined MPG Means';
proc sgplot daat❷=work.cars;
  hbar typeB / response=mpg_combo stat=mean limits=upper;
  where typeB ne 'Hybrid';
run;

Title 'MPG Five-Number Summary';
Titletwo❸ 'Across Types';
proc means data=car❹ min q1 median q3 max maxdec=1;
  class typeB;
  var mpg:;
run;
```

❶ This is a non-syntax error; the variable name Type2 is legal and is used correctly in the LABEL statement. However, no variable named Type2 has been defined in the data set.

❷ This is a syntax error, `daat` is not a legal option in this PROC statement.

❸ This is a syntax error, `titletwo` is not a legal statement name.

❹ This is a non-syntax error; the syntax is legal and directs the procedure to use a data set named Car in the Work library; however, no such data set exists.

Figure 1.4.8A: Checking the SAS Log for Program 1.4.4, First Page

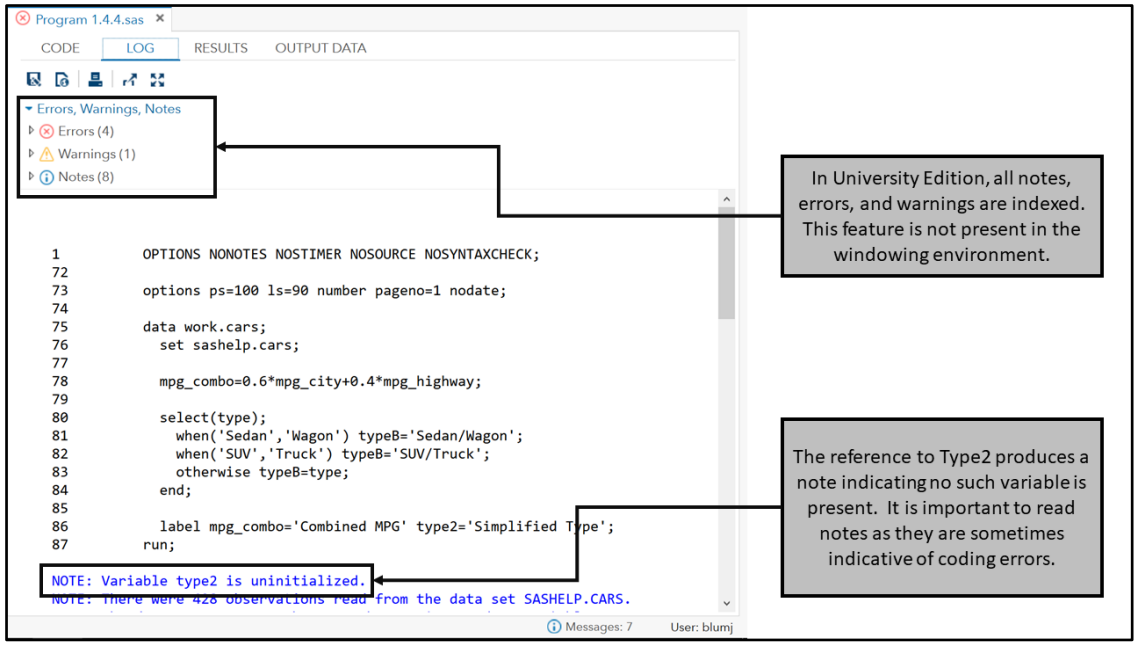


Figure 1.4.8B: Checking the SAS Log for Program 1.4.4, Second Page

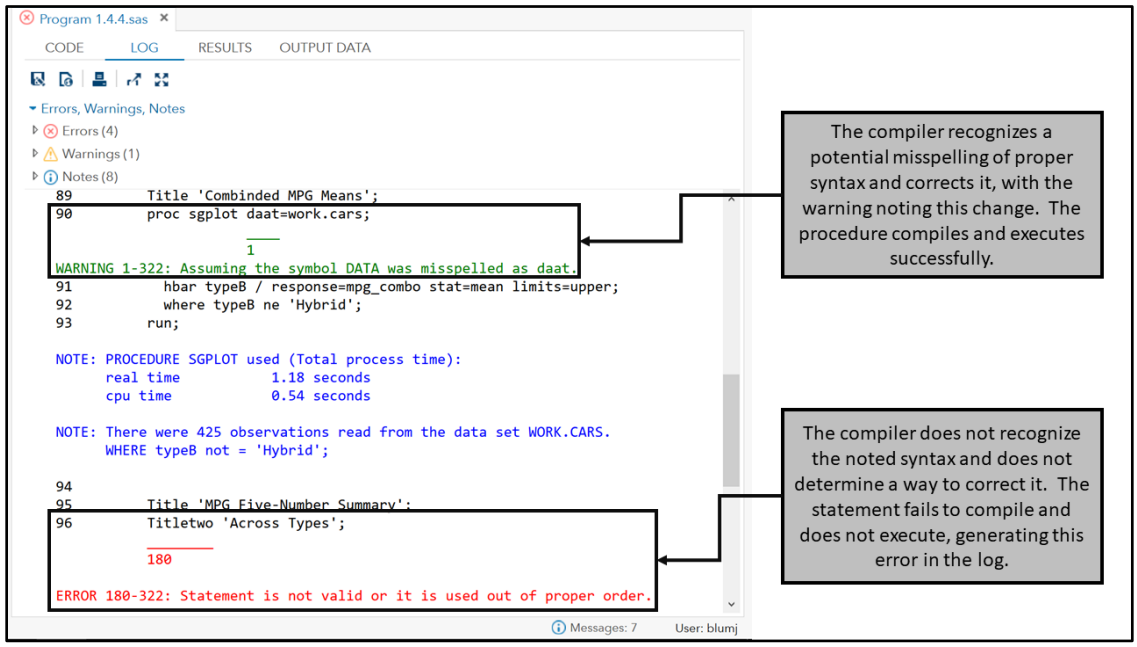
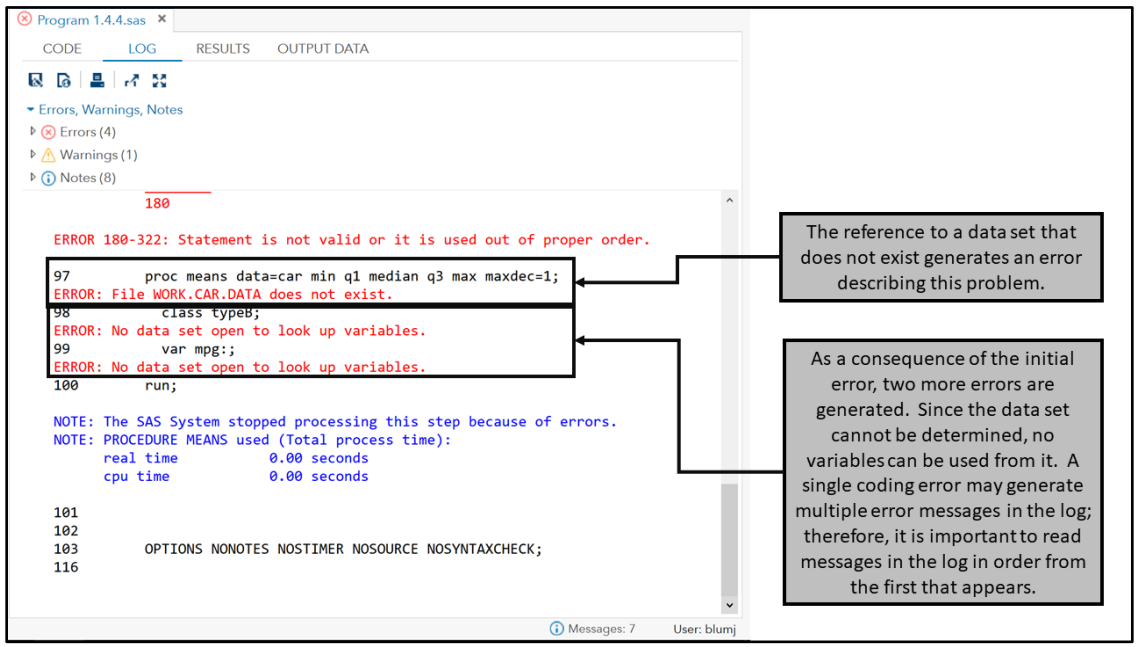


Figure 1.4.8C: Checking the SAS Log for Program 1.4.4, Third Page



The value of a complete review of the SAS log cannot be overstated. Programmers often believe the code is correct if it produces output or if the log does not contain errors or warnings, a practice that can leave undetected problems in the code and the results.

Upon invocation of the SAS session, the log also displays notes, warnings, and errors as appropriate relating to the establishment of the SAS session. See the SAS Documentation for information about these messages.

1.5 Output Delivery System

The sample code presented in this section introduces SAS programming concepts that are important for working effectively in a SAS session and for re-creating samples shown in subsequent sections of this book. Delivery of output to various destinations, naming output files, and choosing the location where they are stored are included. Some differences in appearance that may arise between destinations are also discussed.

Program 1.5.1 revisits the CONTENTS procedure shown in Program 1.4.4, which generates output that is arranged and displayed in four tables. An Output Delivery System (ODS) statement, ODS TRACE ON, is supplied to deliver information to the log about all output objects generated.

Program 1.5.1: Using ODS TRACE to Track Output

```
ods trace on;❶
proc contents data=sashelp.cars;❷
run;

proc contents data=sashelp.cars varnum;❷
run;
```

❶ There are many ODS statements available in SAS, some act globally—they remain in effect until another statement alters that effect—while others act locally—for the execution of the current or next procedure. The TRACE is a recording of all output objects generated by the code execution. ON delivers this information to the SAS log; OFF suppresses it. The effect of ODS TRACE is global, the ON or OFF condition only changes with a submission of a new ODS TRACE statement that makes the change. The typical default at the invocation of a SAS session is OFF.

❷ The VARNUM option in PROC CONTENTS rearranges the table showing the variable information from alphabetical order to position order. This also represents a change in the name of the table as indicated in the TRACE information shown in the log.

Log 1.5.1A: Using ODS TRACE to Track Output

```
74         ods trace on;
75           proc contents data=sashelp.cars;
76         run;


Output Added:
-------------
Name:        Attributes
Label:       Attributes
Template:    Base.Contents.Attributes
Path:        Contents.DataSet.Attributes
-------------

Output Added:
-------------
Name:        EngineHost
Label:       Engine/Host Information
Template:    Base.Contents.EngineHost
Path:        Contents.DataSet.EngineHost
-------------

Output Added:
-------------
Name:        Variables
Label:       Variables
Template:    Base.Contents.Variables
Path:        Contents.DataSet.Variables
-------------

Output Added:
-------------
Name:        Sortedby
Label:       Sortedby
Template:    Base.Contents.Sortedby
Path:        Contents.DataSet.Sortedby
-------------
NOTE: PROCEDURE CONTENTS used (Total process time):
      real time           0.29 seconds
      cpu time            0.20 seconds
```

Log 1.5.1B: Using ODS TRACE to Track Output

```
78         proc contents data=sashelp.cars varnum;
79         run;


Output Added:
-------------
Name:        Attributes
Label:       Attributes
Template:    Base.Contents.Attributes
Path:        Contents.DataSet.Attributes
-------------

Output Added:
-------------
Name:        EngineHost
Label:       Engine/Host Information
Template:    Base.Contents.EngineHost
Path:        Contents.DataSet.EngineHost
-------------

Output Added:
-------------
Name:        Position
Label:       Varnum
Template:    Base.Contents.Position
Path:        Contents.DataSet.Position
-------------

Output Added:
-------------
Name:        Sortedby
Label:       Sortedby
Template:    Base.Contents.Sortedby
Path:        Contents.DataSet.Sortedby
-------------
NOTE: PROCEDURE CONTENTS used (Total process time):
      real time           0.13 seconds
      cpu time            0.07 seconds
```

Each table generated by PROC CONTENTS has a name and a label; sometimes these are the same. Labels are free-form, while names follow the SAS naming conventions described earlier which are revisited in Section 1.6.2. The SAS Documentation also includes lists of ODS table names for each procedure, along with information about which are generated as default procedure output and which tables are generated as the result of including specific options. From the traces shown in Logs 1.5.1A and 1.5.1B, the rearrangement of the variable information when using the VARNUM option is actually a replacement of the Variables table with the Position table.

If ODS table names (and other output object names, such as graphs) are known, other forms of ODS statements are available to choose which output to include or not. Program 1.5.2 shows how to modify each of the CONTENTS procedures in Program 1.5.1 to only display the variable information.

Program 1.5.2: Using ODS SELECT to Subset Output

```
proc contents data=sashelp.cars;
  ods select Variables;❶
run;

proc contents data=sashelp.cars varnum;
  ods select Position;❷
run;
```

❶ ODS SELECT and ODS EXCLUDE each support a space-separated list of output object names. SELECT chooses output objects to be delivered; EXCLUDE chooses those that are not delivered. Only one should be used in any procedure, typically corresponding to whichever list of tables is shorter—those to be included or excluded. In place of the list of object names, one of the keywords of ALL or NONE can be used. ODS SELECT or ODS EXCLUDE can be placed directly before or within a procedure, and its effect is

local if a list of objects is given, only applying to the execution of that procedure. If the ALL or NONE keywords are used, the effect is global, remaining in place until another statement alters it.

❷ The VARNUM option produces a table (Output 1.5.2B) with the same variable information as the first PROC CONTENTS but, as shown in the trace, it is a different table with a different name.

Output 1.5.2A: Using ODS SELECT to Subset Output

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
9	Cylinders	Num	8		
5	DriveTrain	Char	5		
8	EngineSize	Num	8		Engine Size (L)
10	Horsepower	Num	8		
7	Invoice	Num	8	DOLLAR8.	
15	Length	Num	8		Length (IN)
11	MPG_City	Num	8		MPG (City)
12	MPG_Highway	Num	8		MPG (Highway)
6	MSRP	Num	8	DOLLAR8.	
1	Make	Char	13		
2	Model	Char	40		
4	Origin	Char	6		
3	Type	Char	8		
13	Weight	Num	8		Weight (LBS)
14	Wheelbase	Num	8		Wheelbase (IN)

Output 1.5.2B: Using ODS SELECT to Subset Output

Variables in Creation Order					
#	Variable	Type	Len	Format	Label
1	Make	Char	13		
2	Model	Char	40		
3	Type	Char	8		
4	Origin	Char	6		
5	DriveTrain	Char	5		
6	MSRP	Num	8	DOLLAR8.	
7	Invoice	Num	8	DOLLAR8.	
8	EngineSize	Num	8		Engine Size (L)
9	Cylinders	Num	8		
10	Horsepower	Num	8		
11	MPG_City	Num	8		MPG (City)
12	MPG_Highway	Num	8		MPG (Highway)
13	Weight	Num	8		Weight (LBS)
14	Wheelbase	Num	8		Wheelbase (IN)
15	Length	Num	8		Length (IN)

ODS statements can be used to direct output to various destinations, including multiple destinations at any one time. Output styles can vary across destinations, as Program 1.5.3 demonstrates by delivering the same graph to a PDF and PNG file.

Program 1.5.3: Setting Output Destinations Using ODS Statements

```
x 'cd C:\Output';❶
ods _ALL_ CLOSE;❷
ods listing;❸
ods pdf file='Output 1-5-3.pdf';❹
proc sgplot data=sashelp.cars;❺
       styleattrs datasymbols=(square circle triangle);
       scatter y=mpg_city x=horsepower/group=type;
       where type in ('Sedan','Wagon','Sports');
run;
ods pdf close;❻
```

❶ The X command allows for submission of command line statements. CD is the change directory command in both Windows and Linux, here its effect is to change the SAS working directory. The SAS working directory is the default destination for any file reference that does not include a full path—one that starts with a drive letter or name. This directory must exist to successfully submit this code; therefore, either create the directory C:\Output or substitute another that the SAS session has write access to.

❷ The ODS _ALL_ CLOSE statement closes all output destinations.

❸ The ODS LISTING statement activates the listing destination, which is the destination for all graphics files created by the SGPLOT procedure. In the SAS windowing environment the ODS LISTING statement also activates the Output window, but graphics generated by PROC SGPLOT are not displayed there.

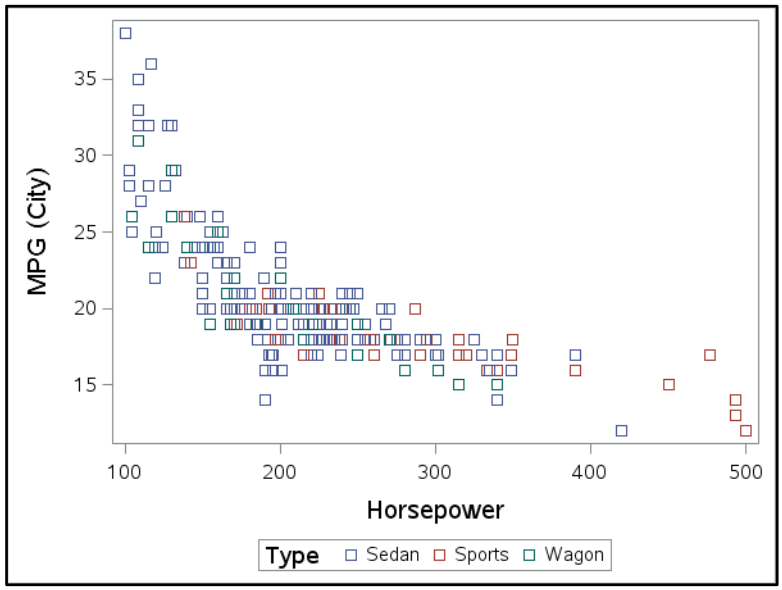
❹ The ODS PDF statement opens the PDF destination specified in the FILE= option (if this option is omitted the file is automatically named). Since the file name does not reference any path, it is placed in the

location specified in ❶. A full-path reference, starting with a drive letter or name, can be given here. Commonly used destinations include PDF, RTF, HTML, and LISTING, but several others are available.

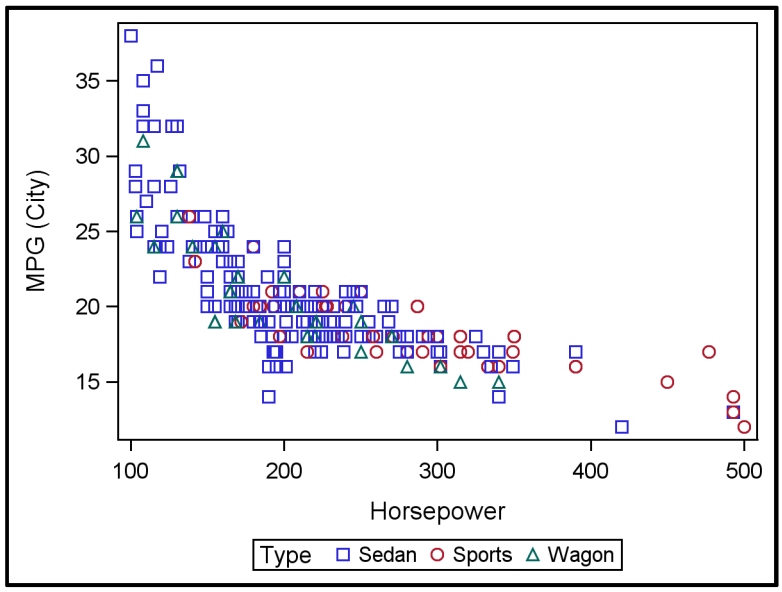
❺ Output 1.5.3A shows the graph generated by PROC SGPLOT and placed in the PDF file, while Output 1.5.3B shows the graphics file (a PNG file by default) generated. Note the difference in appearance between the two (and check the log)—different output destinations can have different options or styles in effect.

❻ The ODS PDF CLOSE statement closes the PDF destination opened in ❹ and completes writing of the file, which includes all output generated between the opening and closing ODS statements. In general, any ODS statement that opens a destination should have a complementary CLOSE statement.

Output 1.5.3A: Graph Delivered to PDF File



Output 1.5.3B: Graph Delivered to PNG File (Listing Destination)



While the graph in the PDF file uses the same plotting shape and cycles the colors, the one delivered as an image file cycles through both different shapes and colors. The takeaway from this example, which applies in several instances, is that not all output destinations use the same styles. In this book, graphs are shown in the form generated by direct delivery to TIF files, see Chapter Note 7 in Section 1.7 for options used to generate these graphs.

Program 1.5.3 shows that ODS statements permit delivery of output to more than one destination at a time, and they also allow for different subsets of output to be delivered to each. While Program 1.5.3 shows that different destinations may have certain style elements that are different, it is also possible to specifically prescribe different styles to different output destinations. Program 1.5.4 opens a PDF and an RTF destination, sending different subsets of the output to each, and with different styles assigned to each.

Program 1.5.4: Setting Multiple Output Destinations and Styles Using ODS Statements

```
ods rtf file='RTF Output 1-5-4.rtf' style=journal;❶
ods pdf file='PDF Output 1-5-4.pdf';❷

ods trace on;❸
proc corr data=sashelp.cars;
  var mpg_city;
  with mpg_highway;
  ods select pearsoncorr;
run;

ods rtf exclude onewayfreqs;❹
proc freq data=sashelp.cars;
  table type;
run;

ods pdf exclude summary;❺
proc means data=sashelp.cars;
  class origin;
  var mpg_city;
run;
ods rtf close;
ods pdf close;❻
```

❶ This opens an RTF destination and applies a style named Journal to it. The style templates available in a given SAS session can be viewed via PROC TEMPLATE, see Chapter Note 8 in Section 1.7 for details. Most tables in this book are the result of delivery to RTF with a template called CustomSapphire—the code required to create the CustomSapphire template is provided in the code that comes with the book, and details about how to set it up and use it are given in Chapter Note 9 in Section 1.7.

❷ This opens a PDF destination, since no STYLE= option is provided, the default style template is used.

❸ The ODS TRACE ON statement is included to ensure that information about output objects is transmitted to the SAS log. To understand the role of ❹ and ❺, review this information in the log.

❹ This PROC FREQ generates a single table named OneWayFreqs. Rather than using ODS EXCLUDE, which would leave it out of both destinations, ODS RTF EXCLUDE keeps it out of the RTF file, but it is included in the PDF file—see Output 1.5.4A and 1.5.4B.

❺ PROC MEANS generates only one table named SUMMARY, and this statement stops it from being put into the PDF file, but it does get delivered to the RTF file.

❻ The ODS RTF CLOSE and ODS PDF CLOSE statements close the destinations opened in ❶ and ❷, completing the writing of those files. In this case, as both destinations are effectively closed at the same time, a single ODS _ALL_ CLOSE statement can replace these two statements.

Output 1.5.4A: Multiple Output Destinations—RTF Results

Pearson Correlation Coefficients, N = 428
Prob > |r| under H0: Rho=0

	MPG_City
MPG_Highway	0.94102
MPG (Highway)	<.0001

Analysis Variable : MPG_City MPG (City)						
Origin	N Obs	N	Mean	Std Dev	Minimum	Maximum
Asia	158	158	22.0126582	6.7333066	13.0000000	60.0000000
Europe	123	123	18.7317073	3.2895093	12.0000000	38.0000000
USA	147	147	19.0748299	3.9829920	10.0000000	29.0000000

Output 1.5.4B: Multiple Output Destinations—PDF Results

Pearson Correlation Coefficients, N = 428 Prob > |r| under H0: Rho=0	
	MPG_City
MPG_Highway MPG (Highway)	0.94102 <.0001

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Hybrid	3	0.70	3	0.70
SUV	60	14.02	63	14.72
Sedan	262	61.21	325	75.93
Sports	49	11.45	374	87.38
Truck	24	5.61	398	92.99
Wagon	30	7.01	428	100.00

1.6 SAS Language Basics

This chapter concludes with a review of the sample programs presented previously, highlighting language rules and variations in style and structure that are permitted.

1.6.1 SAS Language Structure

The following rules govern the structure of SAS programs:

- The SAS language is not case-sensitive. For example, `PROC SGPLOT`, `proc SGPLOT`, `Proc SGplot`, and other casing variations are all equivalent. This applies to all statements, names, functions, keywords, and other SAS language elements.
- In general, SAS statements end with a semicolon; otherwise, the SAS language is relatively free-form. The line breaks and indentations in Program 1.4.1, for example, are chosen to improve readability. In fact, as seen in later chapters, there are times when it is helpful to write one SAS statement across many lines with several levels of indentation. Good programming practice relating to code structure includes two fundamental rules:
  - Develop standards for easy readability of code
  - Follow these standards consistently

- Comments are available in SAS code, and good programming practice requires that code is commented to a level that makes its method and purpose clear. Two ways to write comments are shown in the following samples:
  - /*this is a comment*/
  - *this is also a comment;

1.6.2 SAS Naming Conventions

SAS language elements such as statements, names, functions, and keywords follow a standard set of naming conventions, as follows:

- Permitted characters include letters, numbers, and underscores.
- Names must begin with a letter or underscore
- Maximum length is 32 characters

Some methods are available to avoid these rules when it is deemed necessary, such as connections to other data sources that follow different rules, see Chapter Note 10 in Section 1.7 for more information. There are also some exceptions to these rules for certain SAS language elements. For example, the limitation to 8 characters for a library reference is discussed in Section 1.4.3. SAS formats, introduced in Chapter 2, are another example of a language element having some exceptions to these rules; however, most language elements, including data set and variable names, follow all three exactly.

Text literal values, such as title text or file paths, are encased in single or double quotation marks, as long as the opening quotation mark type matches the closing quotation mark. Be careful when copying text from other applications into SAS, various characters that fill the role of a quotation mark in other software are not interpreted in that manner by SAS—the color coding in the editors is often helpful in diagnosing this problem. In the role of a path, the first example of a text literal given below is generally interpreted as distinct from the second and third due to spacing. However, whether the second and third are taken as distinct due to casing is dependent on the operating system—for example, Microsoft Windows is not case-sensitive.

1. ‘C:\MyFolder’
2. ‘C:\My Folder’
3. ‘C:\my folder’

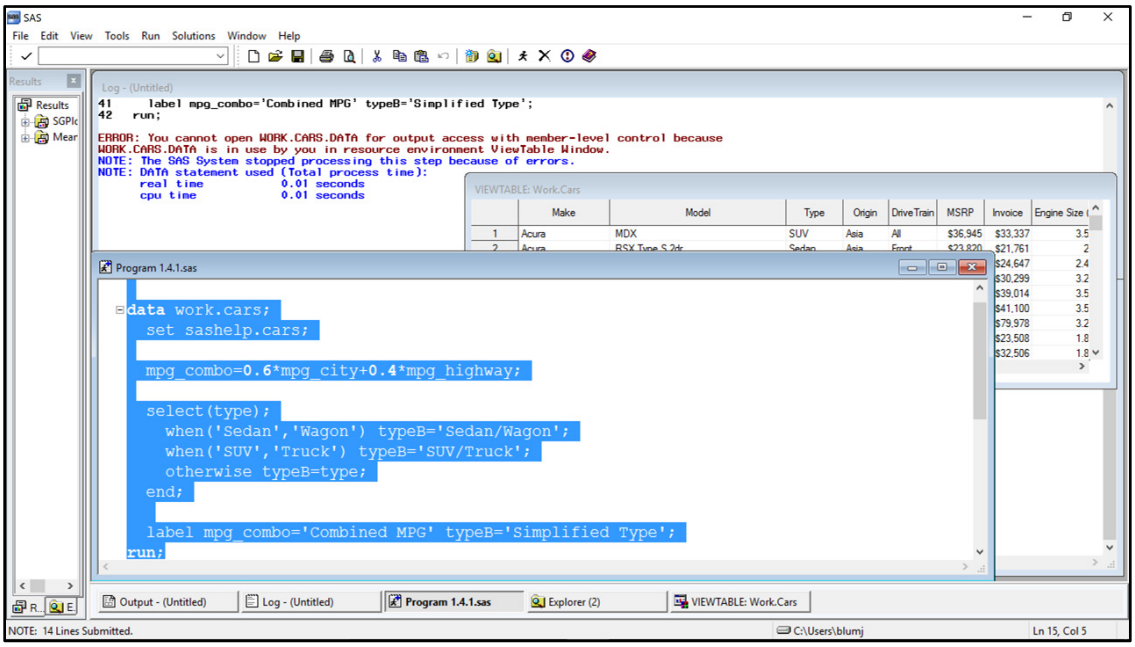
1.7 Chapter Notes

1. *Managing Results in the SAS Windowing Environment*. In the SAS windowing environment, under default conditions, results of code submissions are cumulative in the HTML destination and in the Log window. If the listing destination is active, results are also cumulative in the Output window. For the Log and Output windows, the command **Clear All** from the **Edit** menu is used to clear either of these provided that window is active (take care not to use this command when an Editor window is active). Since the SAS windowing environment only allows for a single Log window and a single Output window during the session, the **New** command (from the **File** menu or using the toolbar button) also clears either window when active. Managing the **HTML** window is a bit more difficult; it too can be controlled by the ODS statements shown in this chapter. See the SAS Documentation for additional details. By default, SAS University Edition replaces the log and results on any code submission, so these steps are not necessary. As stated in Chapter Note 2, SAS University Edition also supports interactive mode and, when active, the Results and Log tabs are cumulative as they are in the SAS windowing environment.
2. *Interactive Mode*. The SAS windowing environment runs in interactive mode by default, while SAS University Edition runs in non-interactive mode by default, but can be set to run in interactive mode. There are two major differences between the two modes. First, results and logs from multiple code submissions are cumulative in interactive mode, while each code submission results in replacements for the log and results in non-interactive mode. (See Chapter Note 1 above.) Next, the final statement in any

code submission made in non-interactive mode is taken as a step boundary, which is not the case in interactive mode, so any submission in interactive mode must end with a step boundary.

3. *SAS Variable Lists.* To aid in simplifying references to sets of variables, SAS provides four types of variable lists:
    a. *Numbered Range Lists.* Numbered range lists are of the form VarM-VarN, where M and N are positive, whole numbers. This syntax selects all variables with the prefix given and numerical suffixes from M and N; for example, Name3-Name5 is equivalent to the list Name3 Name4 Name5. There is no restriction on the order of M and N, so Value6-Value3 is legal and is equivalent to the list Value6 Value5 Value4 Value3. All variables names corresponding to such a reference must be legal and, unless the variables are being created, all in the list must exist.
    b. *Name Range Lists.* Named range lists are of the form StartVar-EndVar, with no special restrictions on the variable names beyond their being legal. The set selected is the complete set of columns between the two variables in column order in the data set—PROC CONTENTS with the VARNUM options provides a method for checking column order. Referring to Output 1.5.2B and the Sashelp.Cars data set, the list Make--Origin is equivalent to Make Model Type Origin. For this list, the order of the two variable names is important, the first variable listed must precede the second variable listed in column order; for example, Origin--Make generates an error when used with Sashelp.Cars. It is possible to insert either of the keywords CHARACTER or NUMERIC between the two dashes, limiting the list to the variables of the chosen type.
    c. *Name Prefix Lists.* As used in example code in this chapter, a name prefix list is of the form var:, referencing all variables, in their column order, that start with the given prefix. For example, MPG: references MPG_City MPG_Highway in Sashelp.cars.
    d. *Special SAS Name Lists.* SAS also provides special lists for selection of variables without actually naming any variables. These are:
        i. _NUMERIC_ : All numeric variables in the data set, in column order
        ii. _CHARACTER_ : All character variables in the data set, in column order
        iii. _ALL_ : All variables in the data set, in column order
4. *Data View in SAS University Edition and SAS Windowing Environments*. Section 1.4.3 shows how to open a data set for viewing in each of the environments and the difference in appearance; however, there is another important difference in how these viewing utilities operate. The ViewTable in the SAS windowing environment maintains an active control over the data set in use, while the tab displayed in SAS University Edition does not. To see one problem that this can cause, submit the code from Program 1.4.1 in the SAS windowing environment, open the Cars data set from the Work library, then re-submit Program 1.4.1 (or, at least, the DATA step at the top of that program). An error message appears in the log, as shown in Figure 1.7.1, indicating the data set being open in a ViewTable has locked out any modifications to it.

Figure 1.7.1: Error Message for Updating a Data Set Open in a View Table



With this active control, the resource overhead in having large tables open in a ViewTable can be substantial. In contrast, the data views in SAS University Edition are based on results of a query of 100 records of the data set (and potentially a limited number of variables when many are present). Once the query is made, control of the data set is released and no resources beyond the current display are in use.

5. *LABEL/NOLABEL System Option*. By default, most procedures in SAS use variable labels (when present) in their output; however, this is in conjunction with the default system option LABEL. It is possible to suppress the use of most labels with the NOLABEL option in an OPTIONS statement, but some labels are still displayed (for example, labels defined in axis statements on a graph or chart). This not only affects variable labels, but also procedure labels—data sets can also have labels, which are unaffected by the NOLABEL option.
6. *Error Types.* The SAS Documentation separates errors into several types. Syntax errors are cases where programming statements do not conform to the rules of the SAS language—for example, misspellings of keywords or function names, missing semi-colons, or unbalanced parentheses. Semantic errors are those where the language element is correct, but the element is not valid for that usage—examples include using a character variable where a numeric variable is required or referencing a library or data set that does not exist. Execution-time errors are errors that occur when proper syntax leads to problems in processing—for example, invalid mathematical operations (such as division by zero) or incorrect sort orders when working with grouped data. Data errors are cases where data values are invalid, such as trying to store character values in numeric variables. Other error types involve SAS language elements that are beyond the scope of this book.
7. *Graphics File Setup.* All graphs shown in the book in Chapters 2 through 8 are generated as TIF files with a specific size and resolution. While running code copied directly from the book often produces similar results, as Output 1.5.3A and B show, it is not guaranteed to be the same. To match the specifications for the graphs in the book exactly, the following statements should precede any graph code (mostly generated with the SGPLOT and SGPANEL procedures):

```
ods listing image_dpi=300;
ods graphics / reset imagename='—give file name here--' width=4in imagefmt=tif;
```

The ODS LISTING statement directs the graphics output to a file, IMAGE_DPI= sets the resolution in dots per inch. The ODS GRAPHICS statement includes options after the slash (/): RESET resets all options to their default, including the sequence of file names. (Image files are not replaced by default, new files are given the same name with counting numbers attached as a suffix.) IMAGENAME= allows for a filename to be specified (a default name is given if none is specified). This can be given as a full-path reference or be

built off the working directory. WIDTH= specifies the width with various units available, HEIGHT= can also be specified—when only one of height or width is specified, the image is produced in a 4:3 ratio. IMAGEFMT= allows for a file type to be chosen, most standard image types are available.

8. *Viewing Available Style Templates.* Lists of available style templates can be viewed using the TEMPLATE procedure with the LIST statement. The following code lists all style templates available in the default location—a template store named Tmplmst in the Sashelp library.

```
proc template;
  list styles;
run;
```

9. *Setting Up the CustomSapphire Template.* Nearly all output tables in the book are built as RTF tables using a custom template named CustomSapphire. The code to generate this template is provided as one of the files included with the text: CustomSapphire.sas. It is designed to store the CustomSapphire template in the BookData library in a template store named Template (which can be changed in the STORE= option in the DEFINE statement in the provided code). If the BookData library is assigned, submitting the CustomSapphire.sas code creates the template in that library. To use the template, two items are required. First, an ODS statement must be submitted to direct SAS to look for templates at this location, such as:

```
ods path (prepend) BookData.Template;
```

The ODS PATH includes a list of template stores that SAS searches whenever a request for a template has been made. Since multiple stores may have templates with the same name, the sequence matters, so the PREPEND option ensures the listed stores are at the start of the list (with the possible exception of the default template store in the WORK library). To see the current template stores listed in the path, submit the statement:

```
ods path show;
```

The template store(s) named in other ODS PATH statements appear in this list shown in the SAS log if they were correctly assigned. These two statements appear in the code given for every chapter of this book.

To see a list of templates available in any template store, a variation on the PROC TEMPLATE code given in Chapter Note 8 is given.

```
proc template;
  list / store=BookData.Template;
run;
```

Finally, to use the style template with any ODS file destination (assuming all previous steps are functional), use STYLE=CustomSapphire, similar to the use of STYLE=Journal in Program 1.5.4.

10. *Valid Variable and Other Names.* For most of the activities in this book, the naming conventions described in Section 1.6.2 apply. However, since SAS can connect to other data sources have different naming conventions, there are times when it is advantageous to alter these conventions. The VALIDVARNAME= system option allows for different rules to be enacted for variable names, while the VALIDMEMNAME= option allows for altering the naming conventions for data sets and data views. These are taken up in Section 7.6, which covers connections to Microsoft Excel workbooks and Access databases, which have different naming conventions than SAS.

1.8 Exercises

Concepts: Multiple Choice

1. Using the default rules for naming in SAS, which of the following is a valid library reference?
    a. ST445Data
    b. _LIB_
    c. My-Data
    d. 445Data
2. Which of the following is not a syntax error?
    a. Misspelling a keyword like PROC as PORC
    b. Misspelling a variable name like TypeB as TypB
    c. Omitting a semicolon at the end of a RUN statement
    d. Forgetting to close the quotation marks around the path in the LIBNAME statement
3. Which of the following is a temporary data set?
    a. Work.Employees
    b. Employees.Work
    c. Temp.Employees
    d. Employees.Temp
4. Using the default rules for naming data sets in SAS, which of the following cannot be used when naming a data set?
    a. Capital letters
    b. Digits
    c. Dashes
    d. Underscores
5. What statement is necessary to produce the following information?

```
Output Added:
-------------
Name:        Report
Label:
Data Name:   ProcReportTable
Path:        Report.Report.Report
```

    a. `ODS RTF;`
    b. `ODS PDF;`
    c. `ODS TRACE ON;`
    d. `ODS LISTING;`

Concepts: Short Answer

1. Under standard SAS naming conventions, decide whether each of the following names is legal syntax when used as a:
        i. Library reference
        ii. Data set name
        iii. Variable name

    Provide justification for each answer.
    a. mydata
    b. myvariable
    c. mylibrary
    d. left2right
    e. left-2-right

   f. house2
   g. 2nd_house
   h. _2nd

2. Classify each of the following statements as: always true, sometimes true, or never true. Provide justification for each answer.
   a. An error message in the log is an indication of a syntax error.
   b. A warning message in the log is an indication of a logic error.
   c. Notes in the log provide details about successful code execution.
   d. Checking the log only for errors and warnings is considered a good programming practice.
3. Classify each of the following statements as: always true, sometimes true, or never true. Provide justification for each answer.
   a. SAS data sets can contain only numeric and character variables.
   b. If the library is omitted from a SAS data set reference, the Work library is assumed.
   c. Once a library is assigned in a SAS session, it is available automatically in subsequent SAS sessions on that machine.
   d. A library and a data set can have the same name.
4. Classify each of the following statements as: always true, sometimes true, or never true. Provide justification for each answer.
   a. A PROC step can be nested with a DATA step.
   b. The RUN statement must be used as a step boundary at the end of a DATA or PROC step.
   c. It is a good programming practice to use an explicit step boundary, such as the RUN statement, at the end of any DATA or PROC step.
   d. Global statements can be included inside DATA or PROC steps.
5. Consider the program below.

```
options date label;
title 'Superhero Profile: Jennie Blockhus';
proc print data = superheroes label;
  where homeTown eq 'Redmond' and current = 'Themyscira';
  var Alias Powers FirstIssue Superfriend Nemesis;
run;

options nonumber nolabel;
proc freq data = superheroes;
  where homeTown eq "Redmond" and current = "Themyscira";
  table sightings*state;
run;
```

   a. Determine the number of global statements.
   b. Determine the number of steps.
   c. What distinguishes global statements from other statements in the above program?
   d. Is it a syntax error to enclose literals in both single and double quotation marks as done in the above program? Why or why not?

Programming Basics

1. Complete the following steps, in either the SAS windowing environment or SAS University Edition:
   a. Open Program 1.8.1 (shown below).
   b. Download the data for the textbook and assign a library to its location.
   c. Replace the comment with the library reference established in part (b).
   d. Submit the code.
   e. If the submission does not execute successfully, check the log and output for errors in the library assignment and/or reference.
   f. Repeat the above steps as necessary until the code executes properly.

Program 1.8.1: Sample Program for Submission

```
proc means data=/*put library reference here*/.IPUMS2005Basic;
  class MortgageStatus;
  var HHIncome;
run;
```

Output 1.8.1: Expected Result from Program 1.8.1 (Colors and Fonts May Differ)

Analysis Variable : HHINCOME Total household income						
MortgageStatus	N Obs	N	Mean	Std Dev	Minimum	Maximum
N/A	303342	303342	37180.59	39475.13	-19998.00	1070000.00
No, owned free and clear	300349	300349	53569.08	63690.40	-22298.00	1739770.00
Yes, contract to purchase	9756	9756	51068.50	46069.11	-7599.00	834000.00
Yes, mortgaged/ deed of trust or similar debt	545615	545615	84203.70	72997.92	-29997.00	1407000.00

Case Study

For additional practice, multiple case studies are available in addition to the IPUMS CPS case study used in subsequent chapters. See Section 8.1 to apply the skills from this chapter to the Clinical Trials Case Study. For additional case studies, including extensions to the IPUMS CPS case study, see the author pages.