第3章

数据集的创建与整理

SAS数据集是SAS系统实现对数据的管理、分析和呈现的基础，而我们日常生活中需要处理的数据都是以各种不同的形式存在的，既可以是某种计算机文件的形式，也可以是记录在纸张上的一些文字信息；只有将各种不同形式的数据转化为SAS数据集，才能使用SAS系统的强大功能对数据进行分析处理。因此，创建与整理SAS数据集是SAS分析的第一步。

原始数据主要以4种形式存在。

① 流数据(Instream Data)，Data步中Cards语句后面的数据行。

② 平面文件(Flat file)，它是一种记录间没有结构关系的文件，如CSV、TXT文件。

③ 数据库管理系统(DBMS)数据文件，如DB2、Sybase、MySQL、Oracle、Teradata及Hadoop。

④ 单机文件(PC file)：是相对于DBMS文件而言的，如Excel、Lotus、DBF、MS Access以及JMP、SPSS、Stata、Paradox等文件。

本章首先介绍如何利用这4种原始数据创建SAS数据集；然后介绍整理SAS数据集的基本方法。

3.1 创建SAS数据集

首先介绍如何利用4种原始数据创建SAS数据集。

3.1.1 流数据和平面文件的读入

例2.3就是流数据读入最简单的例子，而实际工作中碰到的流数据，其书写格式非常丰富，针对不同格式的数据，需要Infile语句、Input语句和Cards语句配合使用来读入数据流。Input语句和Infile语句配合也可以读入平面文件数据。

1. 列表输入法(list)

如果我们的数据文件中(不管是书写在纸上还是以txt格式存在)的所有字段之间都是以空格(或其他的字符)分隔开的，最合适的输入方式就是如例2.3的输入方法，这种方式称为列表输入法。

列表输入法就是简单地按照数据字段在流数据块或平面文件中出现的顺序在Input语句中列出所要读入的变量名，除了需要在字符型变量名后加上$符号外，所有变量不加任何

修饰。

列表输入法下数据字段之间默认的分隔符是空格，也可以采用其他分隔符号，如果分隔符不是空格，需要在 Input 语句之前使用 Infile 语句来指定分隔符；在列表输入法下，连续的分隔符当作一个分隔符处理。列表输入法的 Input 语句语法格式如下：

```
Input  variable 1 <$> < variable 2 <$> …><@|@@>;
```

说明如下。

(1) variable 1，variable 2，…是数据集的变量名，用空格分开。

(2) 如果变量是字符型变量，除非字符型变量在 Input 语句之前已经声明过，否则必须在变量名后面加符号“$”。

列表法适用对象：①原始数据各字段之间以空格(或其他分隔符)分开；②数据值中不包含分隔符。

【例 3.1】 Data 步创建数据集。

```
Data score;
   input id name $ Chinese Math English Physics Chemical;
   cards;
101 欧阳全光河  0   69  92  62  48
102 张丽萍      0   69  89  72  54
103 檀巧萍      0  111  88  71  50
104 沈雅琦      0   96  74  62  53
105 赵婧        0   80  88  75  47
;
Proc print data = score noobs;
Title "高一学生成绩";run;
```

提交程序后，输出窗口显示结果如图 3.1 所示。

高一学生成绩

Id	name	Chinese	Math	English	Physics	Chemical
101	欧阳全光	0	69	92	62	48
102	张丽萍	0	69	89	72	54
103	檀巧萍	0	111	88	71	50
104	沈雅琦	0	96	74	62	53
105	赵婧	0	80	88	75	47
110	包海燕	0	80	85	68	51

图 3.1　输出的 score 数据集的内容

提交程序后根据第 2 章的 Data 步程序的编译与执行原理，已经知道数据是如何读入的。对于这种最简单的数据集创建方法，用指针再来回顾一下数据集的创建过程，以强化我们对列表输入法的理解和掌握。

开始读入数据时，指针从输入缓冲区第一列位置开始扫描，从第一个非空列开始读数据，直到读入一个空列就停止读数，指针停靠在紧邻空格列的下一列(这一列可以是空列也可以是非空列)，将读得的值赋给 PDV 中的变量(如果读取值的字节数超过该变量存储长度，在给 PDV 变量赋值时会发生截断)。接着指针从上次读数结束停靠的列开始扫描，从非空列开始读数只读到下一个空列为止，将读入数据赋给 PDV 中下一个变量，

指针停靠在紧邻空列的下一列；重复以上操作，为 PDV 中其他变量赋值。如果到输入缓冲区结尾，还没有为全部 PDV 变量赋值，则将数据块下一行数据读入输入缓冲区，指针自动移到输入缓冲区的第一列开始为剩余变量赋值，直到为全部变量赋完值为止，一次循环结束。这时不管指针所在输入缓冲区的数据是否读完，输入缓冲区的内容都会更新成数据块的下一行数据，指针也要重新移到输入缓冲区的第一列，准备进行第二次 Data 步循环。

例 3.1 中 Input 语句的变量 name 是字符型变量，使用默认格式（字符型变量的默认存储格式是 8 个字节），导致第一条记录的“姓名”，在读入到 PDV 中时发生了截断，只有“欧阳全光”这 4 个字读入到变量 name 中。

第一条记录读取时，指针停靠位置如图 3.2 所示。

```
----+----1----+----2----+----3----
101 欧 阳 全 光 河    0  69  92    62  48
    ↑                 ↑   ↑   ↑     ↑   ↑
```

图 3.2　输入缓冲区指针停靠位置示意图

如果流数据的字段子值之间的分隔符不是空格，那么需要在 Input 语句之前加上 Infile 语句来指定分隔符。语法格式如下：

```
Infile cards dlm = '单个字符分隔符';
```

或者

```
Infile cards dlmstr = '字符串分隔符';
```

例如，用“#”作为分隔符，例 3.1 的 SAS 代码应该写为：

```
Data score;
  Infile cards dlm = '#';
  input id name $ Chinese Math English Physics Chemical;
  cards;
101#欧阳全光河#0# 69#92#62#48
102#张丽萍    #0# 69#89#72#54
103#檀巧萍    #0#111#88#71#50
104#沈雅琦    #0# 96#74#62#53
105#赵婧      #0# 80#88#75#47
;
```

问题 1：上面提到当一次循环结束时，不管指针所在行的数据是否读完，指针都要移到下一行的第一列，准备进行第二次 Data 步循环。那么，如果数据块不是一行写一条记录，而是一行写两条记录，甚至一行写全部记录，例如：

```
Data score;
   input Id name $ Chinese Math English Physics Chemical;
   cards;
101 欧阳全光河  0   69  92  62  48  102 张丽萍  0  69  89  72  54
103 檀巧萍      0  111  88  71  50  104 沈雅琦  0  96  74  62  53
105 赵婧        0   80  88  75  47
;
Proc print data = score noobs;
Title "高一学生成绩";Run;
```

那么程序提交后，数据集只能得到3条记录，另两条记录就被忽略了。如何解决这个问题呢？可以通过在Input语句末尾加上选项@@来解决：

```
Input variable 1 <$><variable 2 <$> …> @@;
```

其中，@@指示SAS系统在读数据时，当一行数据包含多条观察记录时，用该符号把指针保持在同一行，为下一次Data步循环读入数据。

说明：为了方便叙述指针运行，除非严格按照输入缓冲区、PDV来叙述外，在不致引起混淆的情况下，都简单地用指针从一行的第一列开始、指针移到下一行等方式来叙述Input语句读取数据的机制。但我们要知道，所谓指针移到下一行，指的是输入缓冲区被数据块的下一行数据更新后，指针又回到输入缓冲区的第一列开始重复运行。

```
Data score;
  input Id name $ Chinese Math English Physics Chemical@@;
  cards;
101 欧阳全光河  0   69  92  62  48  102 张丽萍  0  69  89  72  54
103 檀巧萍      0  111  88  71  50  104 沈雅琦  0  96  74  62  53
105 赵婧        0   80  88  75  47
;
Proc print data = score noobs;
Title "高一学生成绩";Run;
```

问题2：SAS语言的一大特点是格式灵活，Data步中的Input语句可以不止一个，如果把Input语句拆成两条，例3.1还能正确创建数据集吗？

```
Data score;
   input Id name $ Chinese Math;
   input English Physics Chemical;
  cards;
101 欧阳全光河  0  69  92  62  48
102 张丽萍      0  69  89  72  54
103 檀巧萍      0  111  88  71  50
104 沈雅琦      0   96  74  62  53
105 赵婧        0   80  88  75  47
;
Proc print data = score noobs;
Title "高一学生成绩";Run;
```

程序提交以后，输出结果如图3.3所示。

高一学生成绩

Id	name	Chinese	Math	English	Physics	Chemical
101	欧阳全光	0	69	102	.	0
103	檀巧萍	0	111	104	.	

图3.3　score数据集的打印结果

出现这样的结果是由于在读数据时，第一行的4个数据分别读给了第一条Input语句的变量Id、name、Chinese和Math，然后指针转到第二行开始为第二个Input语句的变量English、Physics和Chemical读数据，然后指针跳到第三行开始第二次循环。如何解决这个问题呢？

可以通过在最后一个Input之前的Input语句末尾加上选项@来解决。

```
Input variable <$> <variable <$> …> @;
Input variable <$> <variable <$> …> ;
```

其中，选项@指示SAS系统在同一次Data步循环中，保持指针在同一记录行为下一个Input语句的变量读入数据，这个@叫做Trailing@。

```
Data score;
input Id name $ Chinese Math@;                    input English Physics Chemical;
101 欧阳全光河  0   69  92  62  48                   cards;
102 张丽萍      0   69  89  72  54                ;
103 檀巧萍      0  111  88  71  50                Proc print data = score noobs;
104 沈雅琦      0   96  74  62  53                Title "高一学生成绩";Run;
105 赵婧        0   80  88  75  47
```

从例 3.1 可以看到，如果字符型变量的值超过 8 个字符，超出部分会被截断，这是因为字符型变量默认的存储长度是 8 个字节。可以通过指定变量的存储长度来解决这个问题。指定长度存储长度有两种方法。

方法 1：Length 变量 1 <$> length <变量> 2 <$> length …>；

方法 2：Input 变量 1 <$> informatw. d <变量> 2 <$> informatw. d. >…>；

特别提醒：如果用 Length 语句声明变量的长度，Length 语句必须出现在 Input 语句之前。Length 语句声明的变量长度后不需加“.”，Input 语句声明的长度后要加“.”。

【例 3.2】 指定变量的长度。

```
Data score;
    length name $10;
    input Id name Chinese Math@;
    input English Physics Chemical;
    cards;
101 欧阳全光河  0  69  92  62  48
102 张丽萍  0  69  89  72  54
103 檀巧萍  0  111  88  71  50
104 沈雅琦  0  96  74  62  53
105 赵婧  0  80  88  75  47
;
```

这样提交后就可以得到预期的结果。

注意：利用 Length 语句修改变量的默认存储长度，并不影响列表输入法读取数据时指针的移动方式。

而在 Input 语句给变量指定输入格式 informatw. d 时不但修改了变量的默认存储长度为 w 个字节，而且读取数据时指针的移动方式和列表输入法也略有差异。这一点将在**格式输入法**中详细讲解。

列表输入法虽然简单，但也存在一些局限性。

① 一次必须读取一条观测的全部数据，不能跳过任何变量；

② 不允许用空格来代替任何变量的缺失值，缺失值必须用“.”(“.”称为占位符)来代替才能得到预期的结果；

③ 如果是字符型变量，默认的存储长度为 8 字节，这时数据值中间不允许有内置空格；

④ 如果变量值是日期或其他需要进行特殊处理的数据(如会计上习惯书写格式的金额)，列表输入就显得无法处理；

⑤ Input 语句声明的变量顺序必须和数据记录中变量值的顺序一致。

正如在例 2.3 中所列，如果第三条记录中的第三个值用空格代替，而不是用“.”，同时把第二、三条记录的次序交换一下，看看运行结果。

```
Data Salary.Jan;
    Input Name $ Depart $ duty $ salary;
  Provident_fund = salary * 0.07;
    Cards;
宋江 总经理办  CEO  12 000
林冲 待业处
李逵 行政处  处长 9000
;
Run;
```

运行结果显示如下：

```
2  Data Salary.Jan;
3  Input Name $ Depart $ duty $ salary;
4  Cards;
NOTE: 在第 7 行、第 1 - 4 列中有对"salary"无效的数据.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6
----+----7----+----8----+----9----+----0
7          李逵 行政处        处长 9000
Name = 林冲 Depart = 待业处 duty = 李逵 salary = . Provident_fund = . _ERROR_ = 1 _N_ = 2
NOTE: INPUT 语句到达一行的末尾,SAS 已转到新的一行.
NOTE: 数据集 SALARY.JAN 有 2 个观测和 5 个变量.
NOTE: "DATA 语句"所用时间(总处理时间):
      实际时间          0.03 秒
      CPU 时间          0.03 秒
```

再打开数据集看一下，其记录如图 3.4 所示。

图 3.4　例 3.2 运行结果显示

2. 按列输入

当原始数据记录的每个字段值占据相同的列宽时，可以使用按列输入的方式，语法格式如下：

```
Input  variable <$> start - col <- end - col ><.decimals ><@|@@>
```

说明如下。

(1) start-col 表示开始列，end-col 表示结束列，如果字段值仅占 1 列，结束列可以省去。

(2) .decimals 指定小数点右边的数字位数，输入值如果包含小数点，则.decimals 不起作用。

(3) @|@@的用法同列表输入法。

优点：①字段值之间无须用分隔符分隔；②缺失值可以用空格替代，而不再使用占位符“.”；③字符数据允许内嵌空格；④可以跳过不需要的字段值。

【例 3.3】 按列输入。

```
Data scores;
   input name  $  1 - 18 @;
   input score1 25 - 27@;
   input score2 30 - 32@;
   input score3 35 - 37;
   cards;
Joseph       11    32    76
Mitchel      13    29    82
Sue Ellen    14    27    74
Donald Trump
;
Run;
```

按列输入的指针说明：如果为每个变量指定了列宽，那么依次从指定开始列开始把相应的列数据读给变量，相应列有空格的，空格也作为变量值的一部分，只是在把读取的数据赋给变量时，会去掉字段值的前部或尾部的空格；然后将指针移到下一列位置。

3. 格式输入法

有时数值型变量的数据会以不同的形式显示。例如，财务数据往往习惯写成 $568，900.01 的形式，日期型数据的表示形式更是多样化，如 12Feb2013、12-02-2013 等。按照前面的办法读入数据时，SAS 会把它识别成字符型数据。对于这些非标准型的数值型数据的读取，数据的输入格式就变得非常重要。需要注意的是，不管以什么样的格式读入数字，它在 SAS 数据集中都是以数学上的数值本身存储的，如 03Jan1960 存储的数值是 2，因为它和 01Jan1960 之差是 2 天。

格式输入法就是在 Input 语句的变量后面跟随一个**输入格式**指令，格式指令由**格式名**＋**宽度**＋.＋**小数位数**四部分组成，小数位数根据实际需要可选择。

格式有 3 种基本类型，即字符格式、数值格式和日期格式。SAS 提供了丰富的输入格式以便读入各种形式的数据，常用格式见表 3.1 和表 3.2。

表 3.1　常用数字和字符输入(出)格式

格 式 名 称	意　　义	示　　例
$ BinaryW.	将二进制数据转换为字符数据	$ binary16.
$ CharW.	读取带空格的字符数据	$ char20.
$ W.	读入标准字符数据	$ 20.
BinaryW. D	将正二进制数转换为整数	binary8.
CommaW. D/Dollarw. D	读取数字并将逗号和 $ 去掉	Comma10.
W. D	读取标准的数字数据	8. 2

注：$ 表示字符型变量的格式；W 是总长度；D 是小数位数(仅限数值型输入格式)；“.”是输入格式名称的重要部分，不能省略。

表 3.2　常用的 SAS 日期、时间、日期时间输入(出)格式

格 式 名 称	对象	W 范围	W 缺失值	读入数据的形式	备　　注
YYMMDDW.	日期	6～32	6	YYMMDD/YYYYMMDD	
MMDDYYW.	日期	6～32	6	MMDDYY/MMDDYYYY	
DDMMYYW.	日期	6～32	6	DDMMYY/DDMMYYYY	
DATEW.	日期	7～32	7	ddmmmyy/ddmmmyyyy	mmm 为月份的英文缩写
TIMEW.	时间	5～32	8	hh:mm:ss. sspm(am)	分隔符':空格 －'
MONYYw.	时间	5～7	5	MMMYY/MMMYYYY	
DATETIMEW.	时间	13～40	18	ddmmmyyyy hh:mm:ss. ss ddmmmyy hh:mm:ss. ss	

格式化输入常见的有以下几种输入格式。

格式一：Input <指针控制> variable1 <$> informat < variable2 <$> informat…@|@@>

说明：读入数据时，指针要从某一列开始读数据，读完一个变量值后指针就处于下一列的位置，指针控制就是移动指针。指针包括列指针和行指针。指针控制的常用命令见表 3.3。

表 3.3　Input 语句指针控制命令

指针控制	相 对 位 置	绝 对 位 置	说　　明
列指针	＋n；＋numeric-variable；	@n；@ numeric-variable；@(expression)；	
	＋(expression)	@'character-string'；@character-variable；	指针位于紧接字符值的下一列
	向后移动＋(－n)	@(character-expression)	
行指针		#n；# numeric-variable；#(expression)	

注：＋n 表示指定在所停留的位置向前移动 n 列；@n 表示指定要从第 n 列开始读数据。

【例 3.4】 下面是某公司分厂的财务数据，每一行数据包含分厂名字、报表日期和利润。

```
    Factory-1 01Jan2018 $598,467.02
    Factory-2 02Jan2018 $698,345.09
    Factory-3 02Feb2018 $789,342.03
Data profit;
    Input name $9. Date date9. Profit comma11.2;
    Cards;
Factory-101Jan2018$598,467.02
Factory-202Jan2018$698,345.09
Factory-302Feb2018$789,342.03
;
```

VIEWTABLE: Work.Profit

	name	Date	Profit
1	Factory-1	21185	598467.02
2	Factory-2	21186	698345.09
3	Factory-3	21217	789342.03

图 3.5 数据集 profit 存储的内容

提交运行后，直接打开数据集，看到的结果如图 3.5 所示。

如果这个数据集的服务对象是企业财务人员，那么当他打开数据集时对数据集的第二列一定一头雾水，对最后一列的 Profit 的数值看起来也非常不习惯。这说明不管“数据”以何种格式输入到数据集，SAS 在把“数据”写入数据集时只有两种格式，即字符和普通的整数或浮点数。如果这时利用 Print 过程打印数据集报表，得到的结果也和图 3.5 一样。

如何才能使得我们看到的数据集内容符合我们的习惯或者以我们想要的格式展现出来？即便直接打开数据集看到的不是我们习惯上的格式，但是利用 Print 过程打印时，如何使打印出来的报表是我们想要的格式？这就需要使用 format 语句为数据集变量指定数据的输出格式，或者在 Print 过程中添加 format 语句指定要打印的每个变量的打印格式。

把例 3.4 的 Data 步改写如下：

```
Data profit_1;
    Input name $9. Date date9. Profit comma11.2;
    Format date mmddyy8. Profit dollar11.2;
    Cards;
Factory-101Jan2018$598,467.02
Factory-202Jan2018$698,345.09
Factory-302Feb2018$789,342.03
;
Run;
```

或者对 Data 步不做任何修改，而在 Print 过程中增加 format 语句，例如

```
Proc print data=profit;
   Title "带输出格式的报表";
   Format date mmddyy10. Profilt dollar11.2;
Run;
```

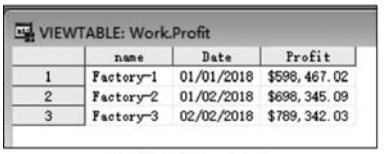
带输出格式的数据集的内容和 Print 过程带输出格式语句的打印结果见图 3.6(a)和图 3.6(b)。

常见的输出格式见表 3.1 和表 3.2。

格式二：Input <指针控制> (variable-list) (informat-list) <@|@@>

说明如下。

(1) (variable-list)必须有(informat-list)跟随。这种情况下数据读入时指针严格按照列



VIEWTABLE: Work.Profit

	name	Date	Profit
1	Factory-1	01/01/2018	$598,467.02
2	Factory-2	01/02/2018	$698,345.09
3	Factory-3	02/02/2018	$789,342.03

(a) 数据集profit_1

带输出格式的报表

Obs	name	Date	Profit
1	Factory-1	01/01/2018	598467.02
2	Factory-2	01/02/2018	698345.09
3	Factory-3	02/02/2018	789342.03

(b) 数据集profit

图 3.6　打印结果

宽读入数据。数据值列宽不足时，对于数值型数据只能在前或后用空格补足列宽，不能在数字中间插入空格；字符型值可在任何位置插入空格补足列宽。

(2) 利用(variable-list)组织变量，读入数据时数据之间不再用空格分开，指针是连续移动按照(informat-list)把相应列的数据直接读给变量，除非用指针命令来移动指针位置。

(3) (informat-list)中的格式可以多于(variable-list)中变量的个数，多余的部分不起作用。

(4) (informat-list)中也可以使用指针，如(3.+1)，表示(variable-list)中每个变量占 3 列，指针向前移动一列再为下一变量读入 3 列的数据。例如，input name $6.(score1-score2)(2.+1);，表示先把 1～6 列读给变量 name，然后把 7、8 列读给 score1，指针停在第 9 列；为 score2 读数据时，指针向前移动一列到第 10 列，然后把第 10、11 列读给 score2。

(5) (informat-list)中相邻的相同格式可以使用 n * informat，表示格式 informat 重复 n 次。

【例 3.5】 格式输入。

(1) 格式列表。

```
Data test;
    input (x y z) (2., +1);
    datalines;
2  24  36
0  20  30
;
Run;
```

(2) 指针移动。

```
Data sales;
    input item $10. +5 jan comma5. +5 feb comma5. +5 mar comma5.;
    Cards;
trucks  1,382  2,789  3,556
vans    1,265  2,543  3,987
sedans  2,391  3,011  3,658
Run;
```

小结：读入数据时指针的位置

理解 Input 语句如何从输入缓冲区中读取数据以及读取数据前后的指针位置，是掌握 Input 语句工作机制的基础。

1. 列表输入法下的指针位置

在列表输入法下，指针扫描数据，从非空列开始读入数据直到读入一个空格，数据读取结束(对于字符型变量来说，如果数据长度大于 8 位，只要不遇到空格读入就不停止，虽然读入超

过了 8 位数据，但存入的只能是 8 位），指针停在紧邻空格的下一列（不管它是空格列还是非空格列）。为下一个变量读入数据时，指针继续扫描，从非空列开始读数据直到读入一个空列停止，指针停在紧邻空列的下一列。继续上述操作，直到为所有变量读入数据完成为止。

例如，为 input region $ jansales；读入数据

----+----1----+----2----+----3

REGION1　　49670

REGION2　　97540

从第 1 列开始，当数据读到第 8 列空格停止读数，指针位于第 9 列，将读入数据 REGION1 赋给变量 region；然后指针继续扫描，首先掠过第 10、11 列两个空格，从第 12 列开始读数据，读到第 17 列（空格）停止读数，指针停在第 18 列，将读到的数据 49670 赋给变量 jansales。

2. 列输入和格式输入

指定变量的起始和终止列时，SAS 读取 Input 语句中指定的列；对于格式输入，SAS 按照指定的格式读入相应的列数；然后指针停留在下一列，准备为下一变量读值。

例如：

input region $ 1-7 sales 12-16；或者 input region $7. +4 sales 5.；

----+----1----+----2----+----3

REGION1　　49670

为变量 Region 读入 7 列值后，指针停留在第 8 列。对于列输入，指针从第 8 列移动到第 12 列开始为 sales 读入数据；对于格式输入，指针从第 8 列，向前移动 4 列，到达第 12 列，开始为 sales 读 5 列数据，读到第 16 列。

上述两个格式不同，但效果相同。如果上面的格式输入没有指针移动控制——“+4”，那么为 sales 读入的就是第 8 列到第 12 列，结果是 4，而不是预想的 49670。

3. 混合格式

【例 3.6】 混合格式。

```
Data club1;
   input Id Name $18. Team $ 25-30 StartWeight EndWeight;
   datalines;
1023 David Shaw          red    189  165
1049 Amelia Serrano      yellow 145  124
1219 Alan Nance          red    210  192
1246 Ravi Sinha          yellow 194  177
1078 Ashley McKnight     red    127  118
1221 Jim Brown           yellow 220  .
;
Proc print data=club1;
   title 'Weight Club Members';
Run;
```

当为 Id 读入数据后，指针停在第 6 列开始为 name 读入数据，因为变量 name 是格式输入，因此读到第 23 列就读够了 18 列，指针停在第 24 列位置，Team 是列输入，指针移动到第 25 列开始读数据，读到第 30 列，指针停在第 31 列；StartWeight 为列表输入格式，指针从第 31 列开始扫描从非空列开始读数据直到读入一个空格，指针停在紧邻空格的后一列准

备为 EndWeight 读数据。

4. 命名输入法

命名输入法读取包含变量名、等号(“=”)和变量值的输入数据。语法格式如下。

格式一：Input <指针控制> variable1= <$>　<variable 2= <$>　…@|@@>;

格式二：Input <指针控制> variable1=informat <variable 2= informat　…@|@@>;

【例 3.7】 命名输入法。

```
Data list;
    length name $ 20 gender $ 1;
    informat dob ddmmyy8.;
    input id name = gender = age = dob = ;
    datalines;
4798 name = COLIN gender = m age = 23 dob = 16/02/75
2653 name = MICHELE gender = f age = 46 dob = 17/02/73
;
Proc print data = list; Run;
```

使用命名输入格式时，Input 语句在命名格式之前的变量可以使用任意输入格式，一旦 Input 语句开始使用命名输入格式，其后的变量必须也是命名输入格式。与之相对应地需要读入的数据，每行对应非命名输入格式的变量值必须按照 Input 语句中的非命名格式的变量顺序排列，其后才是命名输入格式变量的值，这些值由于带有命名输入格式，其顺序可以与命名输入格式的变量顺序不一致。

5. 带修饰的列表输入

大家知道，列表输入有一些局限性。例如，字符型变量长度不能超过 8 字节；默认分隔符为空格，不能读入内嵌空格的数据值；不能处理带特殊符号的数值。SAS 系统添加了“格式修饰符”来消除这些限制。

(1) &. 修饰符。

作用：读入内嵌一个空格的数据值。

不足：数据值之间空格分隔符必须是至少两个空格。

注意：&. 修饰符仅对空格为分隔符的情况有效。

【例 3.8】 带修饰符 &. 示例。

```
Data test1
   Input name &$ duty $;
   Cards;
B.Obama  President
D.Trump  Presiden
   ;
Proc print;Run;
```

(2) 冒号“:”修饰符。

位置：冒号“:”必须放置于格式的前面。

作用：从非空列读数据直到下一个非空列或者已经读入了变量的规定长度，或者到了数据行尾为止。

【例 3.9】 冒号修饰符示例。

```
Data test2;
```

```
   input item : $ 10. Amount :4.;
   datalines;
trucks      1382    /* 数据值之间插入 4 个空格 */
vans        1235    /* 数据值之间插入 6 个空格 */
sedans 2391
;
Proc print; Title "Data of &syslast";Run;
```

由此可见，在格式中加入“:”后，即便指定了列宽，读入数据时也不再像有列宽时直接从指针所在位置开始读，而是先找到非空格列开始，而且即使不够宽度，只要遇到空格，依然会终止数据读入。

（3）～修饰符。

作用：读入数据值时，如果分隔符在单引号或双引号内，该分隔符作为字符处理，同时单引号和双引号也作为字符的一部分赋给变量。

注意：～修饰符要起作用必须在 Infile 语句中使用 DSD 选项；否则 Input 语句会忽视～的作用。

【例 3.10】 带修饰符～示例。

```
Data scores;
   infile datalines dsd dlm = ',';
   input Name : $ 9. Score1 - Score3 Team ~ $ 25. Div $;
   datalines;
Joseph,11,32,76,"Red Racers, Washington",AAA
Mitchel,13,29,82,"Blue Bunnies, Richmond",AAA
Sue Ellen,14,27,74,"Green Gazelles, Atlanta",AA
   ;
Proc print; Run;
```

6. 数组的使用

1）数组的定义

数组是一组按顺序排列的相同类型的 SAS 变量，利用数组可以简化复杂数据的处理。引用数组时，相当于引用构成数组的每一个变量。定义数组的语法格式如下：

```
ARRAY array - name{subscript}<$><length><array - elements><(initial - value - list)>;
```

说明如下。

（1）数组里的元素必须是相同的类型，因此数组分为数值型数组和字符型数组，字符型数组定义时需要在数组名称后面加上“$”。

（2）subscript(下标)描述数组元素的个数和元素的排列；下标有 3 种表示法，即{维数}、{<lower:> upper <,… ,<lower:> upper >}、{*}。subscript 用 * 时，需要列出数组的每一个变量，系统根据列出变量的个数来决定数组的下标。

（3）length 规定数组中元素的长度。

（4）array-elements 是数组的元素列表，元素之间用空格分隔。

（5）initial-value-list 为数组变量赋初值。

（6）数组语句只能出现在 Data 步中，数组中的变量自动加入 Data 步创建的数据集中。

【例 3.11】 定义数组。

```
Array simple{3} red green yellow;   /* 定义一维数组变量名 red green yellow */
```

```
Array x{5,3} score1 - score15;        /* 定义二维数组,变量名为 score1 - score15 */
Array x{1:5,1:3} score1 - score15;  /* 定义二维数组变量名 score1 - score15 */
Array d{101:105} $ 4 vd1 - vd5 (2 * ('nine' 'nine'));
                    /* 定义数组,下标为 101 - 105,元素数据长度为 4,4 个变量赋初值'nine' */
Array e{ * } _numeric_;   /* 包含当前定义过的所有数值型变量,个数系统自动统计并自动标号 */
Array f( * ) _character_;  /* 包含当前定义过的所有字符型变量,个数系统自动统计并自动标号 */
Array g[ * ] _all_;    /* 包含当前定义过的所有变量,类型必须一致,个数系统自动统计并自动标号 */
Array h{5} _temporary_;
    /* 定义数组元素没有变量名的临时数组,包含 5 个元素,临时数组不能用 * 号自动定义个数 */
```

说明：①在数组定义时,用大括号、中括号或小括号均可,不过为规范起见,建议全部用大括号；②临时数组的元素没有变量名,也不能输出到数据集中。

2）数组的引用

通过数组名和下标来引用数组的元素。由于常常会对数组的多个元素进行处理,因此使用数组一般要用到 Do 循环语句。

【例 3.12】 数组的引用示例。

```
Data text;
    array names{ * } $ n1 - n5;
    array capitals{ * } $ c1 - c5;
    input names{ * };
    do i = 1 to 5;
       capitals{i} = upcase(names{i});
    end;
    datalines;
smithers michaels gonzalez hurth frank
    ;
Proc Print data = text;
    title '小写名字转换成大写名字';
Run;
```

3.1.2　读取外部数据文件

读取外部数据分为用 Data 步读入外部数据和用 Proc 步读入外部数据。首先介绍用 Data 步读入分隔符文件(平面文件)；其次介绍用 Import 过程读入外部数据。前面在列表输入法中已经提到如何用 Data 步读入外部文件。下面根据数据文件的不同类型分别介绍导入的方法。

1. 用 Data＋Infile＋Input 语句读取分隔符文件

正如 Data 步读取流数据一样,如果把流数据存储成文本文件,数据出现的位置和存储的物理地址不同。流数据和文本数据本质上是记录间没有结构关系的数据,通常把它看作一类。这样的文件叫做平面文件(Flat file),同时各个字段数据值之间是用特殊字符分隔的,也称为分隔符文件。

一个平面文件既可以是纯文本文件(Plain text file),也可以是二进制文件(Binary file),常见平面文件包括 **TXT 文件**和 **CSV 文件**。

用 Data 步读取分隔符文件,在 Input 语句之前加上 Infile 语句指定数据的来源,即可实现 Input 语句对文本文件的读取。需要注意的是,大多数情况下,外部数据的格式远比我们的流数据要复杂,如 Input 语句要读入的变量长度超出了输入缓冲区的长度、记录中有缺失

值但没有用占位符等。这时 SAS 读入一行记录时，如果 Input 语句在当前输入行找不到所有数据，就会把下一行记录的数据读入，导致出现意想不到的情况。这就需要在 Infile 语句中通过设置特定的选项来处理。

Infile 语句的语法格式如下：

```
Infile fileref < options >;
```

Options 选项众多，比较常见的选项有以下几个。

① DLM＝：指定非空格字符为分隔符；如果分隔符为制表符，在使用 ASCII 码的系统下，取值为'09'X，不可使用 Tab 键(在 Proc Import 过程中可使用 Tab 键)。

② DLMstr＝：指定字符串分隔符。

③ DSD 选项的默认分隔符是逗号，而且连续的逗号表示缺失值。

④ End＝variable：指定一个变量，当读入数据是最后记录时，该变量置为 1；否则为 0。

⑤ Firstobs＝：规定 SAS 从文件中的第几条记录行开始读取数据，默认为 1。

⑥ OBS＝：规定按顺序读入的最后记录数。

⑦ Flowover：当前记录中没有所有变量的值时，从下一记录继续读值，Flowover 是默认选项。

⑧ Missover：阻止到下一条记录为上一条记录的没读到值的变量读值，没读到值的变量设置为缺失值。

⑨ Truncover：当使用列输入或格式输入读取数据时，一些数据字段小于 Input 语句设定的长度时，默认情况下，Input 语句自动读取下一个输入数据记录，导致错误发生。Truncover 使您能够读取可变长度的记录。没有任何赋值的变量被设置为缺失值。

Missover 和 Truncover 的区别在于，如果当前变量没有读到规定的长度，Missover 会将当前变量的值置为缺失值。

⑩ Stopover：如果 Input 语句到了当前记录的尾部，仍有变量没有被赋值，Data 步停止执行。

如果平面文件后缀是 CSV(分隔符为逗号的文件)时，Infile 语句最好用 DSD 选项，这时引号括起来的逗号作为变量值的一部分。

【例 3.13】 读入 CSV 文件。文件 Score.csv 记录了一个班级同学的成绩，该文件的前两行如表 3.4 所示。

表 3.4 班级成绩登记表

班级	学号	姓名	数学	英语	物理	化学
一(3)	333	沈浩	0	0	0	0

第一行实际上相当于数据集的变量标签，真正的记录应该从第二行开始，程序如下：

```
Data score;
     infile 'C:\score.csv' dsd firstobs = 2;
  input class $  id name $  Math English Physics Chemical;
  label class = 班级 id = 学号 name = 姓名 Math = 数学 English = 英语 Physics = 物理 Chemical = 化学;
Run;
```

说明：DSD 选项针对分隔符敏感数据，它有 3 个作用：①使用 DSD 选项时，分隔符默认为逗号“,”，如果分隔符不是逗号“,”，需要使用选项 DLM＝指定分隔符；②它忽略引号

括起来的数据值中的分隔符，而将其看作数据值的一部分；存储数据时能够自动把引号去掉；如果需要把引号作为数据值的一部分，则需要使用特殊修饰符“～”；③把连续的两个分隔符视作一个变量的缺失值。

对于 txt 文件的导入和 CSV 文件的导入没有本质的区别，但是不同的 txt 文件可能会采用不同的分隔符，这时需要正确选用分隔符，如果是逗号分隔符，选 DSD 或 dlm＝“,”都可以；如果 txt 文件是以制表符分隔的文件，由于使用的 Windows 操作系统采用 ASCII 编码，对应的制表符十六进制表示为'09'x，这时分隔符就需要写成 dlm＝'09'x。

注意：在使用 Data 步导入分隔符为制表符时，需要使用 dlm＝'09'x。在 Import 过程导入相同文件，要使用选项 DBMS＝tab，这一点请务必注意。

例如，score.txt(由 Excel 表另存得到，分隔符为制表符)，导入程序为：

```
Data score;
    infile 'C:\score.txt' dlm = '09'x firstobs = 2;
    input class $ id name $ Math English Physics Chemical;
label  class = 班级  id = 学号  name = 姓名  Math = 数学  English = 英语  Physics = 物理
Chemical = 化学;
Run;
```

为了便于看清每条记录的情况，把平面文件中的数据改为流数据，来帮助理解 Infile 语句选项的作用。

【例 3.14】 Missover 应用示例。

```
Data missover;
    input Subject_id $ Subject_name &$18. number_student Teacher &$;
cards;
M01  Advanced algebra    30  Gao San
M02  Calculus
M03  probability theory  34  Xu Jia
    ;
```

其中的第二条记录只有两个值，而 number_student 和 Teacher 这两个变量没有数值，也没有占位符，我们希望得到 3 条记录，其中第二条记录的后两个变量为缺失值。结果读入数据集的记录不是所期望的。

为了得到期望结果，可以运行以下代码：

```
Data missover;
   infile cards missover;
   input Subject_id $ Subject_name &$18. number_student Teacher &$;
cards;
M01  Advanced algebra    30  Gao San
M02  Calculus
M03  probability theory  34  Xu Jia
    ;
Proc print data = missover;Run;
```

输出结果如图 3.6 所示。

SAS 系统

Obs	Subject_id	Subject_name	number_student	Teacher
1	M01	Advanced algebra	30	Gao San
2	M02	Calculus	.	
3	M03	probability theory	34	Xu Jia

图 3.6　打印 Missover 的记录

如果把以上的流数据存放到 txt 文件，如 c:\missover.txt，那么创建数据集的代码可以写为：

```
Data missover;
   infile 'c:\missover.txt' missover;
   input Subject_id $ Subject_name $ number_student Teacher $ ;
Run;
```

请注意,变量的长度这里不需要了,这是因为 SAS 系统会通过扫描文件的记录自动确定每个变量的长度。

【例 3.15】 Truncover 示例。以下数据存储在 Address.dat 中:

```
John Garcia       114  Maple Ave.
Sylvia Chung     1302  Washington Drive
Martha Newton      45  S.E. 14th St.
```

下面程序使用列输入读取 Address.dat 文件。因为有些地址在变量 street 的长度不够输入列 22～37 的宽度,如果不使用 Trunover 选项,SAS 将在第一条和第三条记录尝试转到下一行末 street 读入数据。

```
Data homeaddress;
   infile "C:\address.txt";
  input n $ 1 - 15 nu 16 - 19 st $ 22 - 37;
    run;
Proc print data = homeaddress; Run;
```

输出结果如图 3.7(a)所示。

```
Data homeaddress;
   infile "C:\address.txt" truncover;
  input n $ 1 - 15 nu 16 - 19 st $ 22 - 37;
    run;
Proc print data = homeaddress; Run;
```

输出结果如图 3.7(b)所示。

Obs	Name	Number	Street
1	John Garcia	114	Sylvia Chung 1
2	Martha Newton	45	

(a) 无选项Truncover的记录

Obs	Name	Number	Street
1	John Garcia	114	Maple Ave.
2	Sylvia Chung	1302	Washington Drive
3	Martha Newton	45	S.E. 14th St.

(b) 有选项Truncover的记录

图 3.7 例 3.15 图

图 3.7(a)不是期望的结果。图 3.7(b)是在 Infile 语句加上选项 Truncover 后所期望的结果。

2. Proc Import 过程导入外部数据文件

在互联网时代,数据的获取方式呈现多元化,有平面文件、PC 文件以及种类繁多的统计分析软件生成的数据等,如 SPSS 数据集(文件后缀 sav)、Stata 数据集(文件后缀 dta),这些数据都可以通过 Import 过程导入 SAS 系统,形成 SAS 数据集。

该过程语法结构如下:

```
Proc import Datafile = 'filename' out = < libref.> dataset <(option(s))>< DBMS = identifier >
< Replace >;
    Datarow = n;     /* 从指定行开始读数据,默认从第一行开始读,该语句仅对分隔符文件有效 */
    Delimiter = char / 'nn'x;        /* 当 DBMS = DLM, 分隔符不是空格时加该语句 */
    Getnames = Yes / No;             /* 第一行各列的值是否作为数据集的变量名,默认是 yes */
    Guessingrows = n / Max;          /* 确定变量类型需扫描的文件行数,默认为 20 */
```

```
    Sheet = "sheet - name";          /* 指定读入 Excel 文件的表单名,默认读入第一个表单 */
    Range = "sheet - name $ UL:LR";  /* 指定 Excel 文件某个表单导入的范围,UL 为左上单元格,
                                        LR 为右下单元格;默认为整个表单 */
Run;
```

Import 语句的各选项的含义如下。

① Datafile=：指定需要导入的外部文件；可以是“文件引用名”或“用括号括起来的完整外部文件路径”。

② Out=：指定要生成的 SAS 数据集。

③ Replace：如果生成的数据集存在，那么数据集被重写。数据集选项很多，需要时可以访问帮助文件。

④ DBMS=：指定输入数据类型，常见类型的外部文件及扩展名见表 3.5。

表 3.5　文件类型及扩展名

标识	外部文件类型	扩展名	备　　注
CSV	逗号分隔符	csv	
TAB	制表符分隔符	txt	
DLM	其他分隔符		用 Delimiter=语句指定除逗号和制表符外的其他分隔符
SPSS	SPSS 数据集	sav	
Stata	Stata 数据集	dta	
XLSL	Excel 文件	xlsx,xls	

【例 3.16】 用 Import 读入 csv 数据。

```
Proc import datafile = "C:\score.csv" out = score DBMS = csv replace;
   datarow = 2;
   getnames = no;
Run;
```

【例 3.17】 用 Import 导入 excel 文件。

```
Proc import datafile = "C:\score.xlsx" out = score DBMS = xlsx replace;
  datarow = 2;
  getnames = no;
Run;
```

说明：当导入文件是 Excel 文件时，建议使用 range=语句读取指定工作表的一个矩形区域的数据；该语句缺省时，读入整个表单的数据，但可能出现意想不到的情况。因此，强烈推荐使用 Range 语句，如果 Range 语句规定的数据范围包括标题行，可以使用 Getnames=yes|no 决定第一行是否作为变量名，当选择 Yes 而第一行又不适合作变量名时，自动作为变量的标签。

```
Proc import datafile = "C:\score.xlsx" out = score dbms = xlsx replace;
   sheet = 'class1';
   range = 'class1 $ A1:H10';
   getnames = yes;
Run;
```

3. 读取 DBMS 数据文件

SAS 提供了一组访问关系型数据库的 SAS/Access 接口，借助这些接口，SAS 可以访问其他数据库的数据。SAS 支持访问的关系型数据库有 DB2、MySQL、Hadoop、ODBC、

Oracle、Sybase、Teradata 等 17 个。

SAS/Access 接口引擎提供以下方式访问关系型 DBMS 对象。

① Libname 语句。

② Proc SQL 过程。

③ Access 过程。

详细讲解可以参考 SAS 帮助文档。

3.2 数据集的整理

3.1 节介绍了创建数据集的常用方法，在进行数据分析之前，还需要对已经创建的数据集进行加工和整理。本节介绍使用 Data 步对 SAS 数据进行加工处理。

3.2.1 在 Input 语句之外增加数据集变量

1. 用赋值语句增加变量

在创建数据集时，一般通过 Input 语句设定数据集变量。实际上，赋值语句既可以实现对已有变量的重新赋值，也可以定义新的数据集变量。赋值语句格式如下：

```
New - variable = SAS - expression;
```

【例 3.18】 赋值语句创建数据集变量。

在例 3.13 创建数据集 score 时，数据集的每一条记录是一名同学的数学、英语、物理和化学 4 门课程的成绩。如果在创建数据集的同时，把每一名同学的总成绩和平均成绩计算出来并写入数据集，可以通过增加一个赋值语句来实现。

```
Data score;
   infile 'C\score.csv' dsd firstobs = 2;
   input class $ id name $ Math English Physics Chemical;
   label  class = 班级  id = 学号  name = 姓名  Math = 数学  English = 英语  Physics = 物理
Chemical = 化学 total = 总成绩 means = 平均成绩;
   total = Math + English + Physics + Chemical;
   means = mean(Math,  English,  Physics,  Chemical);
Run;
```

在例 3.18 中，赋值语句 total＝Math＋English＋Physics＋Chemical 和 means＝mean(Math,English,Physics,Chemical)是对每一条记录进行的运算。这样创建的数据集中就会有变量 Total 和 Means。

如果还想得到按照总成绩由高到低的排序，就要借助 sort 过程：

```
Proc sort data = work.Score;
   by descending total; /* 如果是升序用 ascending 选项 */
Run;
```

这时你再看数据集就是按照总成绩的降序排列观测了。

这里用到了排序(sort)过程，它的语法格式为：

```
Proc sort  data = SAS - dataset < out = dataset >;
    By  < Descending > variable;
Run;
```

其中 Out＝dataset 是排序后得到的数据集，原数据集不做改变。如果该选项缺失，那么排序后数据将替代原有的数据存放在原来的位置。

如果要处理的数据集是一个零售企业的每日销售记录，看下面的示例。

【例 3.19】 日销售记录。

```
Data sale;
    Input day date9. Sale;
datalines;
01Jan2018 180.0
02Jan2018 175.0
03jan2018 186.0
;
Run;
```

管理人员不仅需要了解当天的销售情况，更希望了解截至当天的当月累计销售额，因此，需要增加一个变量 Tota_sale 来表示累计销售额，初始值为零；这时需要用到 Retain 语句，一般形式如下：

```
Retain Varibale  Initial－Value … Varibale Initial－Value;
```

该语句的作用是：在编译阶段，如果 Retain 语句指明的变量在 PDV 存储区域不存在，就创建它并为其赋初值；如果没有指定初始值，则初值为缺失值；在执行 Data 步的每次循环中，Retain 语句指定的变量不会被重新初始化，而是保留它上一次循环结束时存储的值。

【例 3.20】 接例 3.19。

```
Data sale;
   Input day date9. Sale;
   retain Total_sale 0;
   Total_sale = Total_sale + sale;
datalines;
01Jan2018 180.0
02Jan2018 175.0
03jan2018 186.0
;
Run;
Proc print; title '日销售记录'; run;
```

运行结果显示如图 3.8 所示。

日销售记录

Obs	day	Sale	Total_sale
1	21185	180	180
2	21186	175	355
3	21187	186	541

图 3.8　打印 Sale 数据集

2. SAS 函数

在例 3.18 中使用了 mean()函数，SAS 系统为用户提供了丰富的函数。这里给出一些经常用到的函数，欲知更多的函数及其用法可以通过“SAS help→SAS 产品→SAS base→SAS 9.4 Functions and CALL Routines：Reference，Fifth Edition”进一步了解。

SAS 函数的调用形式如下：

```
Function－name(variable1 <,variable2, …>)
```

这里简单介绍一些常用的 SAS 函数。

(1) 数学函数,见表 3.6。

表 3.6 常用的数学函数

函 数	功 能	函 数	功 能
ABS(x)	返回 x 的绝对值	Mod(x,y)	返回 x/y 的余数
max(x_1,x_2,…,x_n)	返回 x_1,…,x_n 的最大值	min(x_1,x_2,…,x_n)	返回 x_1,…,x_n 的最小值
Sign(x)	符号函数	Sqrt(x)	返回 x 的平方根
Ceil(x)	大于等于 x 的最小整数	Floor(x)	小于等于 x 的最大整数
int(x)	取 x 的整数部分	Round(x,n)	按指定精度 n,取舍入值
Trunc(x,n)	按规定长度 n 截取数值 x	Exp(x)	指数函数
log(x)	自然对数	$\log_n(x)$	以 n 为底的对数
Sin(x)	正弦函数	Cos(x)	余弦函数
Tan(x)	正切函数	Cot(x)	余切函数
Arcsin(x)	反正弦函数	Arccos(x)	反余弦函数
Atan(x)	反正切函数	Sinh(x)	双曲正弦函数
Cosh(x)	双曲余弦函数	Tanh(x)	双曲正切函数
Lagn(x)	x 的滞后 n 期值	Difn(x)	计算差分,等于 $x-\text{lag}_n(x)$

(2) 样本统计函数:对于一组抽样数据:x_1,…,x_n,常用的统计量函数见表 3.7。

表 3.7 常用的统计量函数

函 数	功 能	函 数	功 能
Mean(x_1,x_2,…,x_n)	返回样本均值	N(x_1,x_2,…,x_n)	返回非缺失样本数
Sum(x_1,x_2,…,x_n)	返回样本和	Nmiss(x_1,x_2,…,x_n)	返回缺失样本值个数
max(x_1,x_2,…,x_n)	样本的最大值	Range(x_1,x_2,…,x_n)	样本极差
min(x_1,x_2…,x_n)	样本的最小值	CV(x_1,x_2,…,x_n)	样本的变异系数
var(x_1,x_2,…,x_n)	样本方差	Std(x_1,x_2,…,x_n)	样本标准差
Stderr(x_1,x_2,…,x_n)	样本标准误差	CSS(x_1,x_2,…,x_n)	样本偏差平方和
Skewness(x_1,x_2,…,x_n)	样本偏度	Kurtosis(x_1,x_2,…,x_n)	样本峰度
Uss(x_1,x_2,…,x_n)	样本未校正平方和		

(3) 累积概率分布函数,见表 3.8。

表 3.8 常用的累积概率分布函数

函 数	功 能
ProbNorm(x)	标准正态分布的分布函数
ProbChi(x,df,nc)	自由度 df,非中心 nc 的卡方分布函数
ProbGam(x,a)	Gamma 分布函数
ProbBeta(x,a,b)	Beta 分布函数
ProbF(x,ndf,ddf,nc)	分子自由度 ndf,分母自由度 ddf,nc 非中心参数的 F 分布函数函数
ProbT(x,df,nc)	自由度 df,非中心参数 nc 的 T 分布函数
ProbBnml(p,n,m)	参数为 p 和 n 的 $x\leqslant m$ 的二项分布概率
ProbNegb(p,n,m)	参数为 p 和 n 的 $x\leqslant m$ 的负二项分布概率
Poisson(lambda,n)	Poisson 分布概率

（4）分位数函数，见表 3.9。

表 3.9　常用的分位数函数

函数（$0 \leqslant p \leqslant 1$）	功　能
ProbIt(p)	标准正态分布的分位数
Cinv(p,df,nc)	卡方分布分位数
Finv(p,ndf,ddf,nc)	F 分布分位数
Tinv(p,df,nc)	T 分布分位数
Betainv(p,a,b)	Beta 分布分位数

（5）字符函数，见表 3.10。

表 3.10　常用的字符处理函数

函　数	功　能	函　数	功　能
Left(s)	左对齐字符串	Right(s)	右对齐字符串
Repeat(s,n)	将 s 重复 n 次	Length(s)	字符串的长度
Reverse(s)	反转字符串表达式	Scan(s,n)	搜寻 s 中第 n 个词
Trim(s)	去掉尾部空格	Substr(s,p,n)	从第 p 个字符开始抽 n 个字符
Upcase(s)	转换 s 为大写	Lowcase(s)	转换 s 为小写
Quote(s)	给字符串加引号	Dequote(s)	移去 s 中的引号

（6）日期时间函数，见表 3.11。

表 3.11　常用的日期时间函数

函　数	功　能	函　数	功　能
Date()	今天的 SAS 日期	Datatime()	当前的日期时间
Time()	取当前时间	Minute(time)	得到分钟数
Hour(time)	计算小时数	Second(time)	计算秒数
Datepart(date)	取 SAS 日期时间的日期	DHMS(d,h,m,s)	生成 SAS 日期时间值
Mdy(m,d,y)	生成 SAS 日期	Hms(h,m,s)	生成 SAS 时间
Month(date)	计算 SAS 日期的月份	Weekday(date)	计算星期几
Day(date)	某月的哪一天	Year(date)	计算哪一年

（7）数组函数，见表 3.12。

表 3.12　常用的数组函数

函　数	功　能	函　数	功　能
Dim(x)	数组 x 的第一维的元素个数	Hbound(x)	数组 x 的第一维上界
Dimk(x)	数组 x 的第 k 维的元素个数	Hboundk(x)	数组 x 的第 k 维上界
Lbound(x)	数组 x 的第一维下界	Lboundk(x)	数组 x 的第 k 维下界

3. 随机函数

随机函数用来生成各种随机数，是随机模拟的基础。

如果一个随机变量 x 的分布函数为 $F(x)$，称随机变量的抽样序列 $x_1, x_2, \cdots, x_n$ 为分布 $F(x)$ 的随机数。

任何一个随机数生成函数都需要一个种子（seed），种子是产生随机数的初始开始点，可以由用户体提供或者由计算机时钟提供。seed 必须是小于 $2^{31}-1$（即 2 147 483 647）的非负整数。如果 seed 是一个正整数，那么随机函数生成的随机数可以再次重复生成；如果 seed

取 0,计算机的时钟初始化随机数,产生的随机数不能重复生成。

(1) 正态分布随机数:Normal(seed)(别名:Rannor())函数产生均值为 0、方差为 1 的标准正态分布随机数。

(2) 均匀分布随机数 Uniform(seed)(别名:Ranuni())产生(0,1)区间上的均匀分布随机数。

(3) 指数分布随机数 RanExp(seed)产生 lambda=1 的指数分布随机数。

(4) 二项分布随机数 RanBin(seed,n,p)产生参数为 n、p 的二项分布随机数。

(5) 泊松分布随机数 RanPoi(seed,lambda)产生参数为 lambda 的泊松分布随机数。

除了上述常用的几个随机数生成函数外,还有一个可以产生所有分布随机数的函数 Rand,语法格式如下:

```
Rand('dist ',parameter-1,…,parameter-k)
```

说明如下。

(1) dist 是随机数服从的分布,如可取 Bernoulli、Beta、Binomial、Poisson、Normal、Chisquare、Exponential、Lognormal 等。

(2) parameter-1,…,parameter-k 是特定分布的形状、位置和规模参数。

Rand()生成的随机数具有很好的统计性质,相关性极小。

【例 3.21】 二项分布和正态随机数。

```
Data test3;
   Do i = 1 to 1000;
      X = Rand('BINOMIAL',0.5,100);
      Y = Rand('Normal',0,16);
      Output;
   End;
Run;
```

这里用到了 Data 步中的 Output 语句,Output 语句将当前记录写到数据集。其语法格式如下:

```
Output <data-set-name(s)>;
```

其中:没有数据集参数的 Output 语句将当前记录写到 Data 语句中命名的所有数据集中;如果 Output 语句后面有数据集名字,那么该数据集必须是 Data 语句中指定的数据集。

为何这里要使用 Output 语句呢?这是因为在 Data 步中,当数据步最后一条语句执行后才会将一条记录写入数据集。而 Output 语句的作用是不等最后一条语句执行,碰到 Output 就将当前记录或变量写入数据集。如果这里不用 Output 语句,Data 步的最后一条语句是 End 语句,当 End 语句执行时,当前的记录是 $i=1000$,x 和 y 分别是第 1000 个二项分布和正态分布随机数,输出到数据集的就只有这一条记录。

3.2.2 变量和观测的选择

1. Set 语句——复制数据集记录

对于一个已经存在的数据集的加工可以使用 Set 语句。Set 语句的作用就是从一个或

多个数据集读取观测。语法格式如下：

```
Set  SAS-dataset(s) <(dataset-option(s))>< options >;
```

说明如下。

(1) dataset-options(s)选项主要有以下几个。

① Keep=variable1 … variablek；在生成的数据集中保留这些变量。

② Drop=variable1 … variablek；在生成的数据集中不包含这些变量。

③ Rename=(old-var-name=new-var-name)；为生成数据集的变量更换名称。

④ Firstobs=设定从某条观测开始读数据。

⑤ Obs=设定读到第几条观测为止。

(2) Options 选项主要有以下几个。

① Curobs=variable 创建一个变量，该变量的值为 Set 从数据集中刚刚读入的那一条观测的 obs 数，该变量不加入数据集，如果需要把该变量加入数据集，需要另外增加一条语句，将该变量赋值给数据集的变量。

② End=variable 创建一个变量，它的值为文件的末尾指示，该变量初始化为零，当 Set 读到数据集的最后一条记录时，该变量值置为 1，该变量不加入任何数据集。

使用 Set 语句的 Data 步的基本格式如下：

```
Data  生成数据集名;
   Set  读入数据集名;
   数据加工语句;
Run;
```

【例 3.22】 Set 语句示例。

```
Data score_1;
    set score;
Run;
```

运行后发现，数据集 score_1 和 score 完全一致，其等同于实现了 SAS 数据集的复制。不过 Set 语句的复制是首先打开数据集，然后一次复制(读取)一条记录。因此，Set 语句中的数据集有多少条记录，Data 步就要循环多少次。效率没有 Copy 过程高。

```
Data score_1;
   set score curobs = cobs;
   if class ne "一(4)" then delete;
   No = cobs;
   total = Math + English + Physics + Chemical;
Run;
```

可以看到，cobs 变量的值是每一次从原始数据集 score 中读入记录的 obs 值。

带 Set 语句的 Data 步的编译与执行步骤如下。

(1) 编译阶段。

① SAS 按照 Set 语句中数据集的排列顺序，读入输入数据集中变量的描述部分。

② 创建包含所有数据集变量和 Data 步创建的其他变量的 PDV。

(2) 执行阶段。

① 读入第一个数据集的第一条观测到 PDV，没有读入值的变量置为缺失值。

② 执行 Data 步的其他语句，可能对 PDV 的内容进行修改。

③ 在 Data 步的最后将 PDV 的内容写入新生成的 SAS 数据集。

④ 每次迭代开始时 PDV 中数据集的变量值保持不变，Data 步创建的变量置为缺失值。

⑤ 程序执行返回 Data 步的开始，继续下一次循环，读入的观测覆盖 PDV 中数据集变量的值；当第一个 set 数据集记录读入结束后，PDV 中数据集变量置为缺失值，准备开始读入第二个 set 数据集记录。

⑥ SAS 按照以上逻辑处理完所有数据集中的所有观测后，Data 步循环结束。

2. 对变量的选择与更名

为了对生成数据集的变量进行删减，可以使用 Drop 或 Keep 语句，这两条语句是相互排斥的，只能使用其中的一个，但效果是相同的。对变量的更名可以用 Rename 语句，它们的一般语法格式如下：

```
Keep variable1 … variablek;
Drop variable1 … variablek;
Rename old - variable - name = new - varibale - name;
```

这 3 条语句的功能也可以通过 3 个数据集选项来实现：

```
(keep = variable1 …variablek)
(drop = variable1 …variablek)
(rename = ( old - variable - name = new - varibale - name))
```

例如，想把例 3.22 生成的一(4)班的成绩中总成绩去掉，例 3.23 所示程序是等效的。

【例 3.23】 保留或删除变量、为变量更名。

(1) 保留或删除变量。

```
Data score_2;
    Set score_1 (drop = total) ;
Run;
```

```
Data score_2;
   Set score_1;
   Drop total;
Run;
```

或：

```
Data score_2;
    Set score_1 (keep = class id name Math English Physics Chemical);
Run;
```

或：

```
Data score_2;
    Set score_1;
    Keep class id name Math English Physics Chemical;
Run;
```

(2) 为变量更名。

```
Data score_2;
   Set score_1 (rename = (Math = m English = e Physics = p Chemical = c));
Run;
```

或：

```
Data score_2;
   Set score_1;
```

```
   Rename Math = m English = e Physics = p Chemical = c;
Run;
```

说明：在使用 drop＝数据集选项时，既可以指定在 Data 语句，也可以指定在 Set 语句，它们的区别在于以下两点

① 如果 Data 步需要处理 drop＝选项指定的变量，那么 drop＝选项只能放在 Data 语句；否则会引起程序执行时找不到所需要的变量而产生错误。

② 如果 Data 步不需要处理 drop＝选项选定的变量，那么 drop＝选项放在 Set 语句可以使得 PDV 中变量减少，从而提高了运行效率。

3. 提取数据集的子集

在对数据集进行加工时，可以对读入数据集的观测进行选择，以生成新的数据集。

1）IF 语句

在 Data 步加工过程中，使用 If 语句可选择要保留（或剔除）的观测，语法格式如下：

```
IF expression  Then statement;
               <Else statement;>
```

例 3.18 创建的数据集是整个年级 4 个班的数据，现在把这个数据集拆分成按班级构成的数据集，数据集名字分别为 class1、class2、class3 和 class4。

【例 3.24】 抽取 4 个班级成绩的数据集。

```
Data class1 class2 class3 class4;
   set score;
   if  class = "一(1)" then output class1;
   if  class = "一(2)" then output class2;
   if  class = "一(3)" then output class3;
   if  class = "一(4)" then output class4;
Run;
```

需要说明的是，不管是 Then 后面的语句，还是 Else 后面的语句，都只能是一条语句，如果语句多于一条时，要用 Do-End 语句来封装成语句组，格式为：

```
Do; 语句组;  End;
```

【例 3.25】 （接例 3.24 续）抽取某个班级（如一班的成绩）成绩的数据集，并计算学生的平均成绩，代码如下：

```
Data class1;
  set score;
  if class = "一(1)" then
     do;
     class = "一班";
     Mean = (Math + English + Physics + Chemical)/4;
      end;
   else delete;
Run;
```

按照数学成绩是否大于 70，将数据集拆分为两个数据集，即 low 和 high，代码如下：

```
Data low high;
   Set class1_m;
   If math < 70 then output low;
                Else output high;
```

```
Run;
```

说明：语句“Data low high；”创建两个数据集 low 和 high；output low 和 output high 将满足条件的记录分别输出到两个数据集。

2）取子集 IF 语句

取子集的 IF 语句，其基本语法格式如下：

```
If expression;
```

它执行这样的操作：当 expression 为真时，SAS 继续执行 Data 步；如果 expression 为假时，停止处理后续语句，当前观测也不会添加到数据集，然后 SAS 开始下一次循环。该语句只能用在 Data 步中。

【例 3.26】 接例 3.25。

```
Data class1;
   set score;
   If class = "一(1)";
Run;
```

运行该程序后就可以抽取出一(1)班学生的各科成绩数据集。

3）数据集选项 where＝与 where 语句

数据集选项 where＝也可以用来提取数据子集，其一般形式为：

```
Where = (expression)
```

【例 3.27】 接例 3.26。

```
Data class1(Where = ( class = "一(1)"));
   set score;
Run;
```

或者：

```
Data class1;
   set score(Where = ( class = "一(1)"));
Run;
```

也可以使用 where 语句，where 语句的格式为：

```
Where expression;
```

Where 语句既可以在 Data 步使用，也可以在 Proc 步使用的，它的作用类似于取子集 If 语句，用来选择满足条件的记录。但取子集 If 语句只能用在 Data 步，而不能用在 Proc 步。

【例 3.28】 接例 3.27。

```
Data class1;
   set score;
   Where class = "一(1)";
Run;
```

4）数据集选项 firstobs＝与 obs＝。

【例 3.29】 读取第 5～9 条共 5 条记录。

```
Data score_5;
    set score(firstobs = 5 obs = 9);
Run;
```

3.2.3　数据集的合并

使用 Data 步可以对两个或两个以上的数据集进行合并，数据集的合并分为串接(concatenation、interleave)和并接(merge) 两种方式。

① 串接。多个数据集的记录按照一定的顺序加在一起形成新的数据集，新数据集的记录个数等于每个数据集记录个数之和。

② 并接。每条记录的字段增加，形成新数据集。

数据集合并的机制见图 3.9。

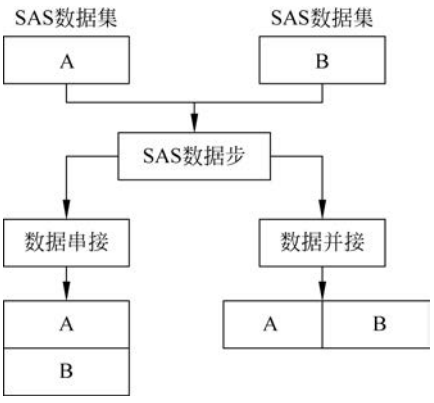


图 3.9　数据集合并框图

1. 数据集的串接

Data 步串接数据集的语法格式如下：

```
Data new - dataset;
    Set dataset - 1 < dataset - option > … dataset - n
< dataset - option >;
    < By < descending > variable - 1 variable - 2 … ;>
Run;
```

串接生成的新数据集的变量个数等于 Set 语句后所有数据集变量的全体，记录个数等于 Set 语句后所有数据集记录的总和，如果某个数据集的变量不包含在其他数据集中，那么来自其他数据集的观测对应的该变量取值为缺失值。

By 语句的作用是让观测按某些变量的升序或降序穿插排列。默认是升序排列。使用 By 语句的前提是每个被串接的数据集必须已经按照 By 语句指定的变量事先进行了排序。

【例 3.30】 数据集 Class1～Class4 里分别存放的是 4 个班的语文、数学、英语、物理成绩，将这 4 个数据集串接成全年级成绩的数据集 score，并按数学成绩由高到低排序：

```
Data score;
  Set class1 class2 class3 class4;
  By math;
Run;
```

注意：数据集串接时，如果相同的信息在不同的数据集中对应的变量名字不同，则需要在使用 Set 语句时对变量名进行重新命名，以确保相同的信息有相同的变量名称。

假如这个年级的 4 个班级在考试时，由于教学进度不一致，比如四班的数学教学进度晚了一节课，而恰好有一道填空题分值是 2 分，四班学生都没填，为便于教学评比，在形成全年级成绩时，对四班的数学成绩每人加 2 分。这就需要用到数据集选项“IN＝”来追踪或选择观测，其一般形式如下：

```
Set dataset(IN = variable);
```

其中：variable 是一个数值型临时变量，它出现在 PDV 中，但不输出到数据集中。当某条记录是从该数据集读入时，该变量置 1；否则置 0。

【例 3.31】 接例 3.30。

```
Data score;
    Set class1 class2 class3 class4(In = in4);
    By math;
```

```
    If in4 = 1 then math = math + 2;
Run;
```

2. 数据集的并接

1）匹配并接(Match-Merging)

数据集合并的另一种情况是描述同一组对象的不同属性信息被分别记录在不同的数据集中，这时如果需要记录同一组对象的完整信息的数据集，就需要进行数据集的并接。

例如，学校每次的考试成绩都是各任课教师把自己所任课程的成绩单独创建数据集进行记录(数据集的变量为**班级、学号、姓名、单科成绩**)，那么要得到全班一次考试的综合成绩就需要将这些数据集合并到一起，合并时学生学号相同的合并成一条记录，合并后的数据集变量为班级、学号、姓名、各科成绩。生成的数据集的变量增多了，但记录数不增加。

这种情况就是SAS数据集的匹配并接，它可以将描述同一组对象的信息合并在同一条记录中。用Data步匹配并接的一般格式如下：

```
Data new - dataset;
     Merge dataset1 dataset2 … ;
     By variable;
Run;
```

需注意以下几点。

(1) 进行匹配并接前，所有相关数据集要按照by变量排序或有适当的索引。

(2) 如果除了by变量外，合并的数据集有相同名称的变量，那么最后一个数据集的变量将会覆盖前面数据集的同名变量。为了避免这种情况发生，可以通过数据集选项重新命名这些变量。

(3) 匹配并接时，一般情况下是一对一并接；如果一个数据集中的by分组记录都读入，而另一个数据集中同样的by分组记录还有没被读入的记录，SAS执行一对多并接，就是把较少by组的那个数据集的最后一条记录和较多by组的其余记录分别并接，形成一对多记录。

【例3.32】 csv文本文件class1-P.csv、class1-M.csv、class1-E.csv、class1-chin.csv和class1-chem.csv分别为一班的物理、数学、英语、语文和化学成绩，将它们转换成SAS数据集，并合并成为一个数据集。代码如下：

```
Data class1_p;
    Infile  'c:\class1 - P.csv' dsd firstobs = 2;
    Input class $  id name $  physics;
    ;
Data class1_M;
    Infile  'c:\class1 - M.csv' dsd firstobs = 2;
    Input class $  id name $  math;
    ;
Data class1_E;
    Infile  'c:\class1 - e.csv' dsd firstobs = 2;
    Input class $  id name $  english;
    ;
Data class1_chin;
    Infile  'c:\class1 - chin.csv' dsd firstobs = 2;
    Input class $  id name $  chinese;
    ;
```

```
Data class1_chem;
     Infile  'c:\ class1 - chem.csv' dsd firstobs = 2;
     Input class $  id name $  chemical;
     ;
Data class1;
   merge class1_chin class1_m class1_e class1_p class1_chem;
   By id;
Run;
```

【例 3.33】 一对多匹配并接。

```
Data n1;
 input num vara $  var1 $  @@;
 cards;
 1 a1 a 2 a2 b 2 a3 c 3 a4 d
 ;
 Data n2;
 input num varb $  varc $ @@;
 cards;
 1 B1 A 2 B2 B 2 B3 C 2 B4 D 3 B5 E
 ;
 Data hb1;
 merge n1(rename = (var1 = varc)) n2;
    by num;
Run;
```

2）非匹配并接(one-to-one Merging)

使用 Merge 语句时，如果不使用 By 语句，这时对被并接的数据也不要求按任何变量排序，系统处理这样的并接相当于按照观测序号进行匹配并接，即 SAS 连接每个数据集的第一条观测，然后连接每个数据集的第二条观测，以此类推。

【例 3.34】 非匹配并接。

```
Data n3;
 input num vara $  @@;
 cards;
 1 a1 2 a2 2 a3 3 a4
 ;
 Data n4;
 input num varb $  @@;
 cards;
 1 B1 2 B2 2 B3 2 B4 3 B5
 ;
 Data hb2;
 merge n3 n4;
 Run;
```

3. 数据集合并示例

下面通过一些例子来介绍数据集合并的编程技巧。

1）临时变量 First. var 和 Last. var

在 Data 步中如果使用了 By 语句，系统会在 PDV 中产生两个临时变量 First. var 和 Last. var，这里的 var 指的是 By 语句后的变量名。

① 读入 by 分组的每组第一条记录时，First. var 为 1；否则为 0。

② 读入 by 分组的每组最后一条记录时，Last.var 为 1；否则为 0。

比如，为了把一个班里学生按照数学成绩排序，相同数学成绩的挑一个做代表。

【例 3.35】 试比较以下两个 Data 步的结果。

```
Data class1;
  set class1;
  by math;
  if first.math = 1;
Run;
```

或：

```
Data score.class1;
  set score.class1;
  by math;
  if last.math = 1;
Run;
```

2）Set 语句的选项 Point＝和 Nobs＝

在 Data 步中使用 Set 语句读入数据时，可以使用选项 Point＝直接指明要读入的记录序号。语法格式如下：

```
Set dataset  Point = pointer - variable;
```

其中，pointer-variable(指针变量)是临时变量，且必须在 Set 语句之前执行对它的赋值，该变量不输出到数据集。

【例 3.36】 从一班每隔 5 个学生挑一个组成一个新班集。

```
Data class_1;
     do i = 5 to 35 by 5;
        set class1 point = i;
        output;
     end;
     stop;
Run;
```

如果不知道 class1 班的学生是 35 人，可以在 Set 语句中使用 Nobs 选项，格式为：

```
Nobs = variable;
```

它的作用是：创建一个临时变量来存放数据集中的观测个数，这一临时变量在 Data 步编译时被赋予数据集的记录个数。

【例 3.37】 （接例 3.36。）

```
Data class_1_1;
     do i = 5 to num by 5;
        set class1 point = I Nobs = num;
        output;
     end;
     stop;
Run;
```

运行结果是一样的。

注意：Stop 语句控制在 Do-End 循环语句结束后终止 Data 步的执行。因为在 Set 语句的 Data 步程序中，系统将反复执行 Data 步的语句，直到读入数据集的结束标志。由于使用

了 Point=，会直接读入指针指向的记录，使得 Data 步可能不能读到数据集的结束标志，这会引起 Data 步死循环。因而必须采用 Stop 语句，让系统执行完 Do-End 语句组后结束 Data 步循环。

【例 3.38】 随机从一班抽取 5 个学生。

```
Data class_1_2(drop = i);
    do i = 0 to 5;
      choice = ceil(rand('uniform') * num);
      set score.class1 point = choice nobs = num;
      output;
   end;
   stop;
Run;
```

小结：提供数据集信息的临时变量。

在数据集的加工过程中，正确使用数据集处理过程中产生的临时变量，可以对数据集进行更加灵活的操作。

（1）自动生成的临时变量：_n_和_error_分别记录 Data 步的循环次数和执行中的出错信息。

（2）First.var 和 Last.var 是带 By 语句的 Data 步中自动生成的，标志 by 分组的开始与结束。

（3）Set 语句选项产生的临时变量，如 Point=、in=、Nobs=、end=等。

以上这些都是临时变量，仅存在于 PDV 中，可以在 Data 步中使用，但不存入 SAS 数据集。如需存入数据集，需要将临时变量的值赋给数据集变量。

3.2.4　数据集的转置

将一个数据集转置就是将观测转为变量、变量转为观测。转置过程可以消除需要编写冗长的程序也可以达到相同的结果，便于实现对数据的分析处理。数据集转置是通过 Transpose 过程完成的，其一般格式如下：

```
Proc transpose data = input - dataset   Out = output - dataset   < options >;
     By variable - list;
     Id variable - list;
     Var variable - list;
Run;
```

说明如下。

① By 语句：如果有分组变量需要保留到新数据集中作为变量，可以使用 By 语句。By 变量本身不转置。新数据集的数值型变量的个数等于最大的 by 组的观测个数；如果一个 by 组较其他 by 组有更多的观测，那么在输出数据集中，对没有相应输入观测的变量分配缺失值。

② Id 语句：Id 变量的非缺失格式化值将作为转置后数据集的变量名，如果 Id 变量多于一个，那么 Id 语句中出现的 Id 变量的值将依次连接在一起形成新的变量名。Id 变量的值必须是唯一的；如果 Id 变量的值缺失，那么相应的观测就不被转置。

③ Var 语句：列出需要转置的变量，这些变量的变量名变成新数据集的变量_name_的值，如果希望改变变量 _name_ 的名字，可在 Proc Transpose 语句中加入选项 name=‘新变

量名'。如果 Var 语句缺失,那么所有在其他语句里没有出现的数值型变量都将被转置;如果 Var 语句中的某个变量是字符型变量,那么所有被转置的变量在新数据集中都被处理成字符变量。

【例 3.39】 简单的转置。

```
Data test;
     Input Tester1 Tester2 Tester3 Tester4@@;
     Cards;
22 25 21 21 15 19 18 17 17 19 19 19
20 19 16 19 14 15 13 13 15 17 18 19
;
Run;
proc print data = test;
     title "Test 数据集数据";
Run;
proc transpose data = test out = transposed_test;
Run;
proc print data = transposed_test;
     title "转置后的数据集数据";
Run;
```

运行结果如图 3.10 所示。

Test数据集数据

Obs	Tester1	Tester2	Tester3	Tester4
1	22	25	21	21
2	15	19	18	17
3	17	19	19	19
4	20	19	16	19
5	14	15	13	13
6	15	17	18	19

转置后的数据集数据

Obs	_NAME_	COL1	COL2	COL3	COL4	COL5	COL6
1	Tester1	22	15	17	20	14	15
2	Tester2	25	19	19	19	15	17
3	Tester3	21	18	19	16	13	18
4	Tester4	21	17	19	19	13	19

图 3.10 数据集转置前后的数据

【例 3.40】 复杂的转置。

```
Data auto_dealer;
     input Model $ month $ sold notsold repaired junked@@;
     cards;
BJ  Jan 23 17 1 0 DF  Jan 34 6  2 1 BYD Feb 45 21 3 3
BJ  Feb 25 15 1 2 DF  Feb 24 16 0 2 BYD Mar 45 25 3 5
BJ  Mar 33 7  0 0 DF  Mar 30 10 2 3
;
proc sort data = auto_dealer;
     by model;
proc print data = auto_dealer;
    Title "Auto_dealer 数据";
proc transpose data = auto_dealer out = transpoed_dealer;
     by model;
     id month;
proc print data = transpoed_dealer;
     Title "转置后数据";
Run;
```

运行结果如图 3.11 所示。

Auto_dealer数据

Obs	Model	month	sold	notsold	repaired	junked
1	BJ	Jan	23	17	1	0
2	BJ	Feb	25	15	1	2
3	BJ	Mar	33	7	0	0
4	BYD	Feb	45	21	3	3
5	BYD	Mar	45	25	3	5
6	DF	Jan	34	6	2	1
7	DF	Feb	24	16	0	2
8	DF	Mar	30	10	2	3

转置后数据

Obs	Model	kind	Jan	Feb	Mar
1	BJ	sold	23	25	33
2	BJ	notsold	17	15	7
3	BJ	repaired	1	1	0
4	BJ	junked	0	2	0
5	BYD	sold	.	45	45
6	BYD	notsold	.	21	25
7	BYD	repaired	.	3	3
8	BYD	junked	.	3	5
9	DF	sold	34	24	30
10	DF	notsold	6	16	10
11	DF	repaired	2	0	2
12	DF	junked	1	2	3

图 3.11　数据集转置前、后的数据

习　题　3

3.1　以下为某快递公司的部分员工信息。

姓名	性别	籍贯	入职日期	基本工资
张　雯	女	黑龙江	2016-01-15	3300.00
P. John	男	海南	2018-02-12	3200.00
童子铭	男	云南	2018-02-03	3260.00
刘爱华	女	陕西	2019-03-01	3000.00
王国栋	男	山西	2015-03-03	3600.00
关楚宇	男	湖北	2018-03-03	3100.00
赵　迪	女	湖南	2015-06-01	3500.00

(1) 用 Data 步把数据输入到 SAS 数据集。注意，姓名为两个字的中间要有一个空格，英文名字名和姓中间有一个空格。日期的输出格式要采用如 09Jan2008 的格式。

(2) 把生成的数据集按照入职时间进行排序，排序后生成新的数据集，原数据集保留。

(3) 编程挑选基本工资超过 3500 元的员工单独生成一个 SAS 数据集。

3.2　平面文件 szs_CINFO.xlsx 存放的是深圳证券交易所的上市公司基本信息，shs_CINFO.xlsx 存放的是上海证券交易所的上市公司基本信息。

(1) 利用 Import 过程将以上两个文件导入 SAS 形成两个数据集。

(2) 将两个数据集合并成沪深上市公司基本信息的数据集。

(3) 在合并数据集的基础上，按照省份将数据集拆分形成各省上市公司基本信息数据集。

3.3　shs_bs.xlsx 存放的是沪市上市公司 2019 年度的部分财务信息，shs_CINFO.xlsx 存放的是沪市上市公司的基本信息。

(1) 将两个文件导入 SAS 生成两个数据集。

(2) 将两个数据集按照股票代码并接成一个上市公司完整信息的数据集，并计算上市公司的资产负债比率和每股利润。