SP205-207 Doing More with SAS Programming

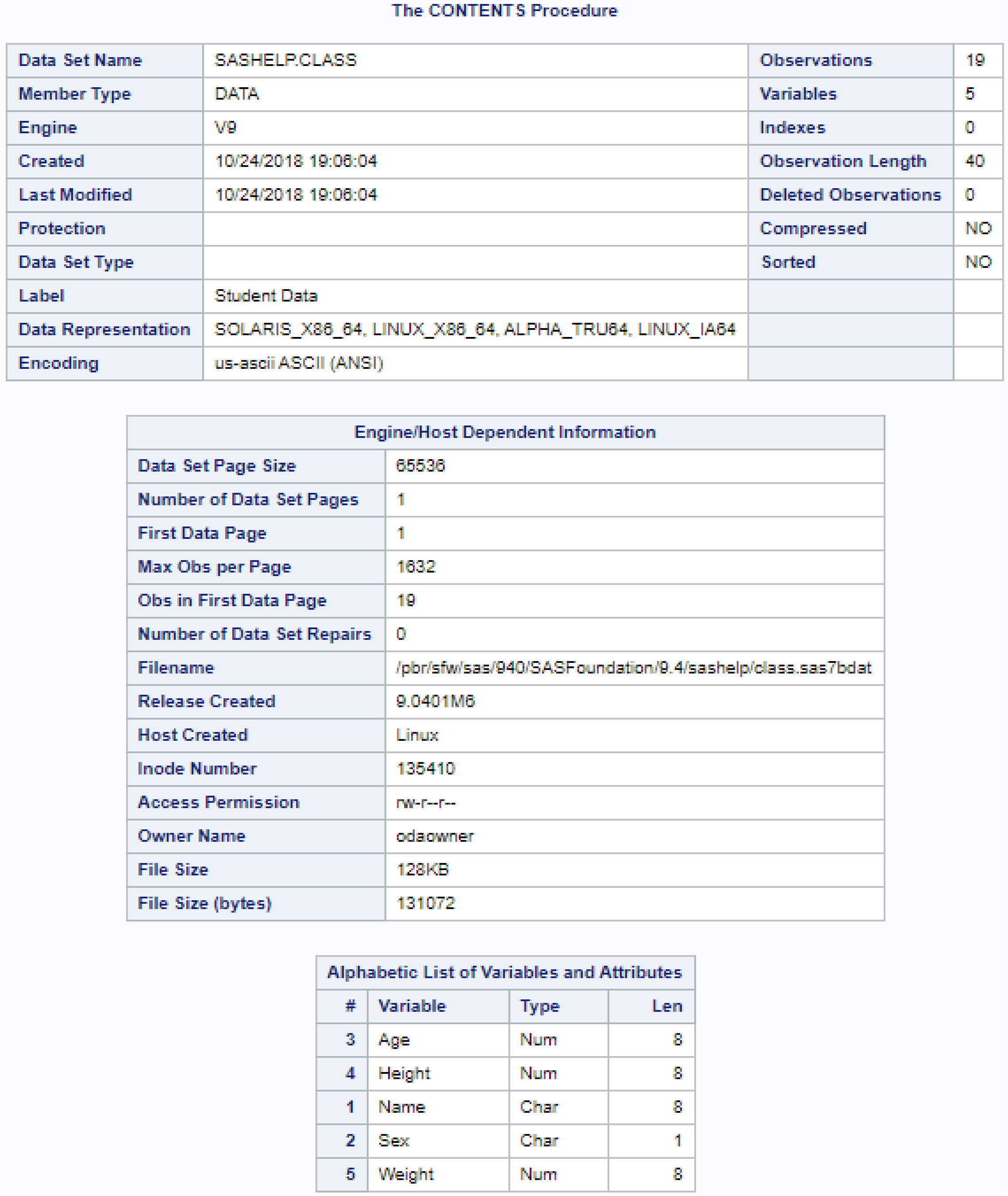
205 Concatenating Tables, Merging Tables, Identifying Matching and Non-matching Rows

206 Using Iterative DO Loops, Using Conditional DO Loops

207 Restructuring data with DATA Steps, Restructuring data with the TRANSPOSE Procedure

```
********************************************************;
*  p205a01.sas Activity 5.01                          *;
*  1) Notice the SET statement concatenates the        *;
*     SASHELP.CLASS and PG2.CLASS_NEW2 tables. Highlight  *;
*     the DATA step and run the selected code. What      *;
*     differences do you observe between the first 19    *;
*     rows and the last 3 rows?                      *;
*  2) Use the RENAME= data set option to change Student   *;
*     to Name in the PG2.CLASS_NEW2 table. Highlight the  *;
*     DATA step and run the selected code. What warning   *;
*     is issued in the log?                          *;
*  3) Highlight the two PROC CONTENTS steps and run the   *;
*     selected code. What is the length of Name in       *;
*     SASHELP.CLASS and Student in PG2.CLASS_NEW2?       *;
********************************************************;

data class_current;
   set sashelp.class pg2.class_new2(rename=(Student=Name));
run;

proc contents data=sashelp.class;
run;
```

```
proc contents data=pg2.class_new2;

run;
```



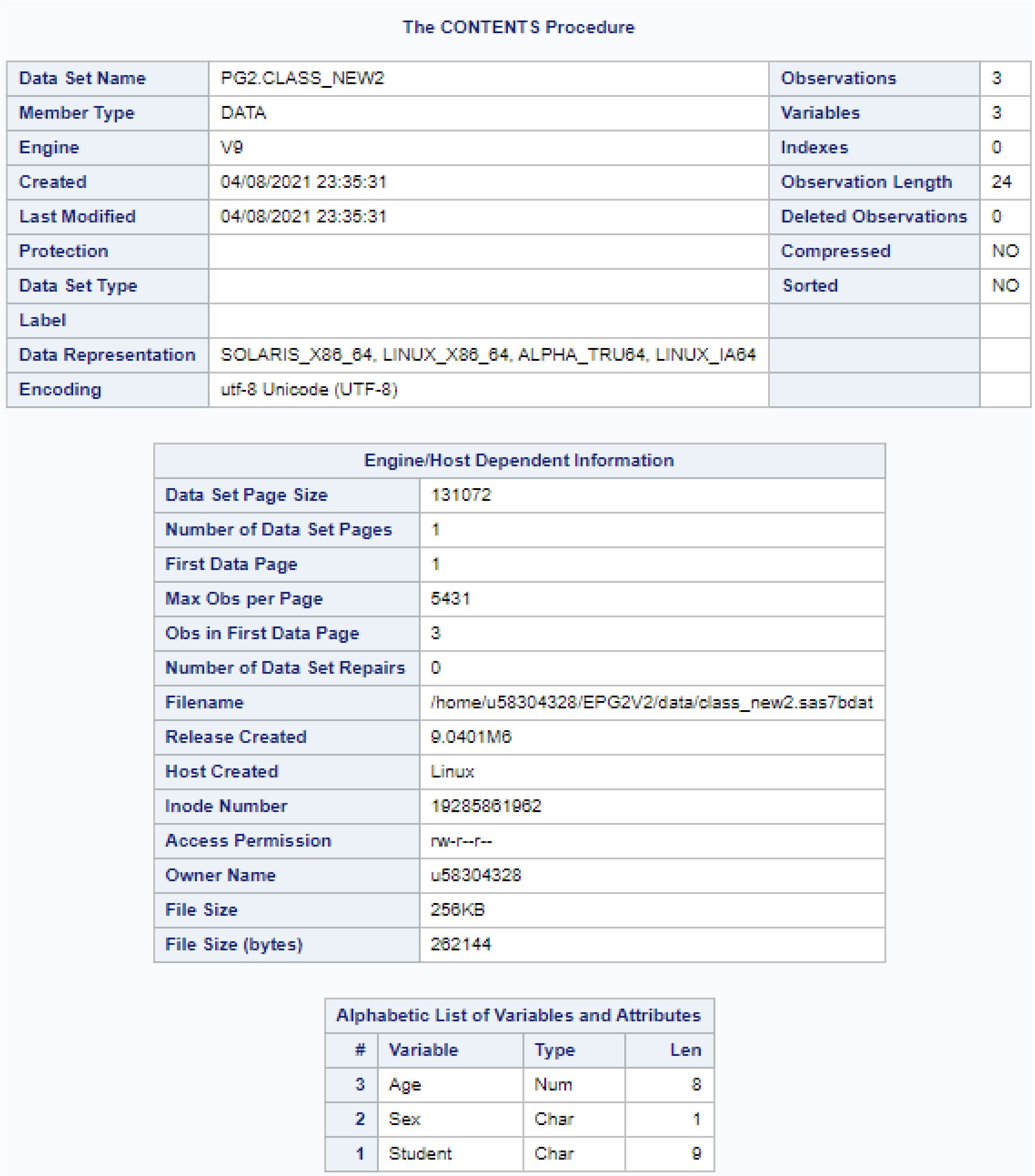
The CONTENTS Procedure

Data Set Name	SASHELP.CLASS	Observations	19
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Created	10/24/2018 19:06:04	Observation Length	40
Last Modified	10/24/2018 19:06:04	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label	Student Data		
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	us-ascii ASCII (ANSI)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	1632
Obs in First Data Page	19
Number of Data Set Repairs	0
Filename	/pbr/sfw/sas/940/SASFoundation/9.4/sashelp/class.sas7bdat
Release Created	9.0401M6
Host Created	Linux
Inode Number	135410
Access Permission	rw-r--r--
Owner Name	odaowner
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	8
2	Sex	Char	1
5	Weight	Num	8



The CONTENTS Procedure

Data Set Name	PG2.CLASS_NEW2	Observations	3
Member Type	DATA	Variables	3
Engine	V9	Indexes	0
Created	04/08/2021 23:35:31	Observation Length	24
Last Modified	04/08/2021 23:35:31	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Set Page Size	131072
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	5431
Obs in First Data Page	3
Number of Data Set Repairs	0
Filename	/home/u58304328/EPG2V2/data/class_new2.sas7bdat
Release Created	9.0401M6
Host Created	Linux
Inode Number	19285861962
Access Permission	rw-r--r--
Owner Name	u58304328
File Size	256KB
File Size (bytes)	262144

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	Age	Num	8
2	Sex	Char	1
1	Student	Char	9

```
***********************************************************;
*  p205a02.sas Activity 5.02                              *;
*  1) Highlight the two PROC SORT steps and run the       *;
*     selected code. How many rows per Name are in the    *;
*     and TEACHERS_SORT and TEST2_SORT tables?            *;
*  2) Complete the DATA step to merge the sorted tables   *;
*     by Name. Run the DATA step and examine the log and  *;
*     results. How many rows are in the output table?     *;
**********************************************************;


proc sort data=pg2.class_teachers out=teachers_sort;
          by Name;
run;


proc sort data=pg2.class_test2 out=test2_sort;
          by Name;
run;


data class2;
          *Complete the MERGE and BY statements;
          merge teachers_sort test2_sort;
          by Name;
run;
```

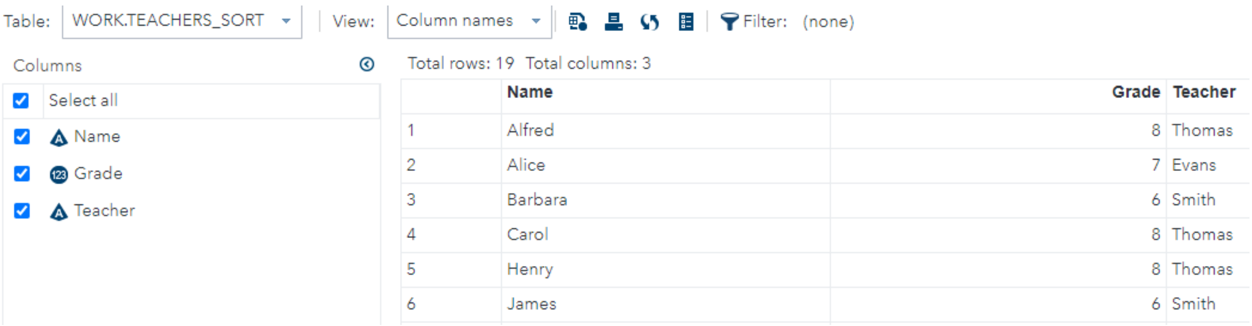


Table: WORK.TEACHERS_SORT View: Column names Filter: (none)

Columns

- ☑ Select all
- ☑ Name
- ☑ Grade
- ☑ Teacher

Total rows: 19 Total columns: 3

	Name	Grade	Teacher
1	Alfred	8	Thomas
2	Alice	7	Evans
3	Barbara	6	Smith
4	Carol	8	Thomas
5	Henry	8	Thomas
6	James	6	Smith

```
**********************************************************;
*  p205a04.sas Activity 5.04                            *;
*  1) Modify the final DATA step to create an additional    *;
*     table named STORM_OTHER that includes all          *;
*     nonmatching rows.                          *;
*  2) Drop the Cost column from the STORM_OTHER table only. *;                    *;
*  3) How many rows are in the STORM_OTHER table?        *;
```

```
*****************************************************************;

proc sort data=pg2.storm_final out=storm_final_sort;
	by Season Name;
run;

data storm_damage;
	set pg2.storm_damage;
	Season=Year(date);
	Name=upcase(scan(Event, -1));
	format Date date9. Cost dollar16.;
	drop event;
run;

proc sort data=storm_damage;
	by Season Name;
run;

data damage_detail storm_other(drop=Cost);
	merge storm_final_sort(in=inFinal) storm_damage(in=inDamage);
	keep Season Name BasinName MaxWindMPH MinPressure Cost;
	by Season Name;
	if inDamage=1 and inFinal=1 then output damage_detail;
	*Add ELSE statement;
	else output storm_other;
run;
```

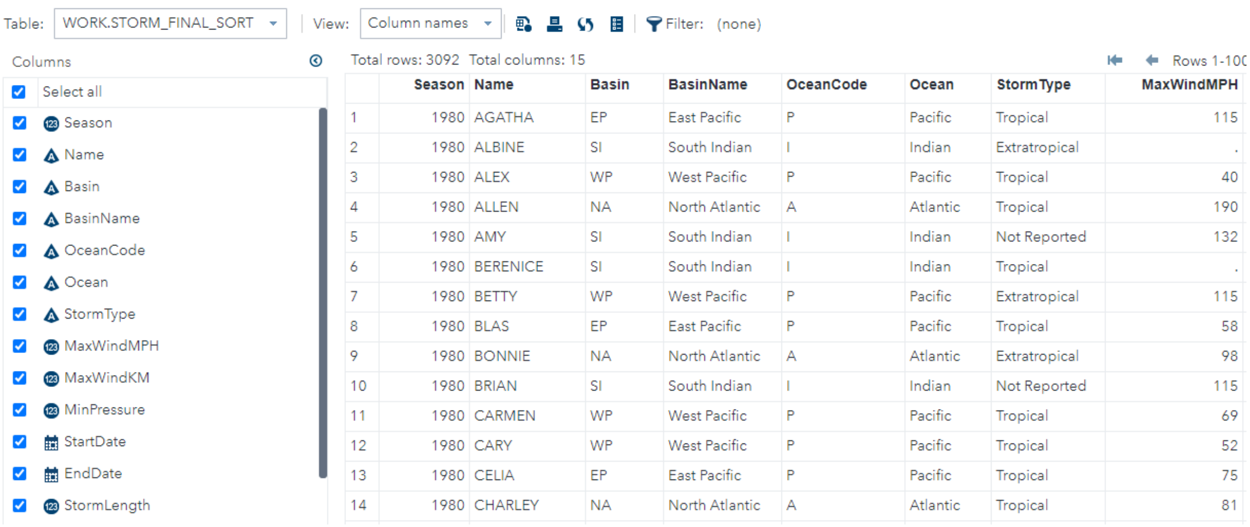


Table: WORK.STORM_FINAL_SORT View: Column names Filter: (none)

Columns

☑ Select all
☑ Season
☑ Name
☑ Basin
☑ BasinName
☑ OceanCode
☑ Ocean
☑ StormType
☑ MaxWindMPH
☑ MaxWindKM
☑ MinPressure
☑ StartDate
☑ EndDate
☑ StormLength

Total rows: 3092 Total columns: 15 Rows 1-100

	Season	Name	Basin	BasinName	OceanCode	Ocean	StormType	MaxWindMPH
1	1980	AGATHA	EP	East Pacific	P	Pacific	Tropical	115
2	1980	ALBINE	SI	South Indian	I	Indian	Extratropical	.
3	1980	ALEX	WP	West Pacific	P	Pacific	Tropical	40
4	1980	ALLEN	NA	North Atlantic	A	Atlantic	Tropical	190
5	1980	AMY	SI	South Indian	I	Indian	Not Reported	132
6	1980	BERENICE	SI	South Indian	I	Indian	Tropical	.
7	1980	BETTY	WP	West Pacific	P	Pacific	Extratropical	115
8	1980	BLAS	EP	East Pacific	P	Pacific	Tropical	58
9	1980	BONNIE	NA	North Atlantic	A	Atlantic	Extratropical	98
10	1980	BRIAN	SI	South Indian	I	Indian	Not Reported	115
11	1980	CARMEN	WP	West Pacific	P	Pacific	Tropical	69
12	1980	CARY	WP	West Pacific	P	Pacific	Tropical	52
13	1980	CELIA	EP	East Pacific	P	Pacific	Tropical	75
14	1980	CHARLEY	NA	North Atlantic	A	Atlantic	Tropical	81

```
***************************************************************;
*  p205d01.sas Concatenating Tables                            *;
***************************************************************;
*  Syntax and Example                              *;
*                                           *;
*   DATA output-table;                             *;
*      SET input-table1(rename=(current-colname=new-colname))*;
*         input-table2 ...;                        *;
*   RUN;                                       *;
***************************************************************;

*Example with all matching columns;
data class_current;
          set sashelp.class pg2.class_new;
run;

*Example with columns having different names;
data class_current;
          set sashelp.class pg2.class_new2(rename=(Student=Name));
run;
```

```
************************************************************;
*  Demo                                   *;
*  1) Modify the SET statement to concatenate           *;
*    PG2.STORM_SUMMARY and PG2.STORM_2017. Highlight the *;
*    DATA and PROC SORT steps and run the selected code. *;
*  2) Notice that for the 2017 storms Year is populated   *;
*    with 2017, Location has values, and Season is       *;
*    missing. Rows from the storm_summary table          *;
*    (starting with row 55) have Season populated and    *;
*    Year and Location are missing.                   *;
*  3) After PG2.STORM_2017, use the RENAME= data set      *;
*    option to rename Year as Season. Use the DROP= data *;
*    set option to drop Location. Highlight the demo     *;
*    program and run the selected code.                *;
************************************************************;


data storm_complete;
	*Complete the SET statement;
	set pg2.storm_summary pg2.storm_2017(rename=(Year=Season) drop=Location);
	Basin=upcase(Basin);
run;


proc sort data=storm_complete;
	by descending StartDate;
run;
```

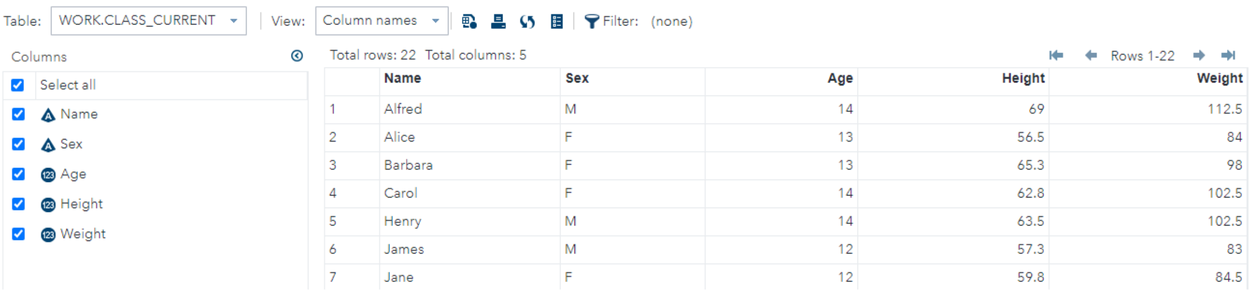


Table: WORK.CLASS_CURRENT View: Column names Filter: (none)

Columns

☑ Select all
☑ Name
☑ Sex
☑ Age
☑ Height
☑ Weight

Total rows: 22 Total columns: 5

Rows 1-22

	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5

```
***********************************************************;
*  p205d02.sas Merging Tables                             *;
***********************************************************;
*  Syntax and Example                                     *;
*                                                         *;
*    PROC SORT DATA=input-table OUT=output-table;         *;
*       BY BY-variable;                                   *;
*    RUN;                                                 *;
*    DATA output-table;                                   *;
*       MERGE input-table1 input-table2 ...;              *;
*       BY BY-variable;                                   *;
*    RUN;                                                 *;
***********************************************************;


/*One to one Merge*/
/*Both tables are already sorted*/
data class2;
  merge sashelp.class pg2.class_teachers;
  by name;
run;


***********************************************************;
*  Demo                                                   *;
*  1) Highlight the two PROC SORT steps and run the       *;
*    selected code. Examine the sorted tables, including *;
*    the number of rows in each. Notice that both tables *;
*    include a column representing basin codes. However, *;
*    the column is named Basin in the STORM_SORT table   *;
*    and BasinCode in the BASINCODES_SORT table.         *;
*  2) To combine the BasinName column with the columns in *;
*    the storm_summary table, the tables need to be      *;
*    merged. Complete the MERGE statement. Use the       *;
*    RENAME= data set option to rename the BasinCode     *;
*    column as Basin in the BASINCODES_SORT table. Add a *;
*    BY statement to combine the sorted tables by Basin. *;
*  3) Run the program and examine the STORM_SUMMARY2      *;
*    table. Notice that the BasinName values have been   *;
*    matched with each of the Basin code values.         *;
*  4) Scroll to the end of the STORM_SUMMARY2 table.      *;
*    Notice that when the value of Basin is lowercase    *;
*    na, the values for BasinName are missing. This is   *;
*    because lowercase na occurs only in the STORM_SORT  *;
*    table and not in BASINCODES_SORT.                   *;
***********************************************************;

proc sort data=pg2.storm_summary out=storm_sort;
	by Basin;
run;

proc sort data=pg2.storm_basincodes out=basincodes_sort;
	by BasinCode;
run;
```

```
data storm_summary2;
        merge storm_sort basincodes_sort(rename=(BasinCode=Basin));
        by Basin;
run;
```

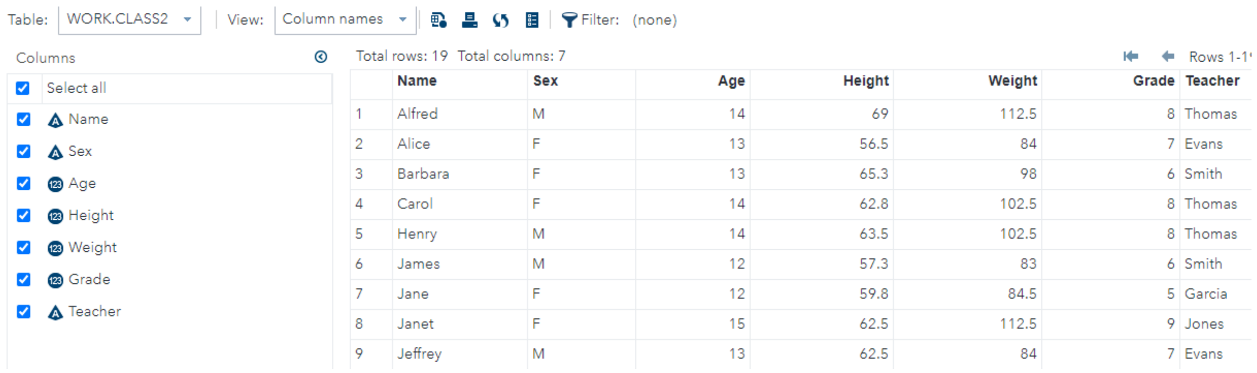


Table: WORK.CLASS2 View: Column names Filter: (none)

Columns

☑ Select all
☑ Name
☑ Sex
☑ Age
☑ Height
☑ Weight
☑ Grade
☑ Teacher

Total rows: 19 Total columns: 7 Rows 1-1'

	Name	Sex	Age	Height	Weight	Grade	Teacher
1	Alfred	M	14	69	112.5	8	Thomas
2	Alice	F	13	56.5	84	7	Evans
3	Barbara	F	13	65.3	98	6	Smith
4	Carol	F	14	62.8	102.5	8	Thomas
5	Henry	M	14	63.5	102.5	8	Thomas
6	James	M	12	57.3	83	6	Smith
7	Jane	F	12	59.8	84.5	5	Garcia
8	Janet	F	15	62.5	112.5	9	Jones
9	Jeffrey	M	13	62.5	84	7	Evans

```
*********************************************************;
*  p205d03.sas Merging Tables with Non-matching Rows        *;
*********************************************************;
*  Syntax and Examples                          *;
*                                          *;
*   DATA output-table;                          *;
*      MERGE input-table1(IN=variable1)             *;
*          input-table2(IN=variable2) ...;           *;
*      BY by-variable;                          *;
*      IF expression;                           *;
*   RUN;                                     *;
*********************************************************;


/*Include matching rows only*/
data class2;
   merge pg2.class_update(in=inUpdate)
       pg2.class_teachers(in=inTeachers);
   by name;
   if inUpdate=1 and inTeachers=1;
run;

***********************************************************;
*  Demo                                     *;
*  1) Highlight the first PROC SORT step and run the      *;
*     selected code. A table named STORM_FINAL_SORT is    *;
*     created, arranged by Season and Name. Because some  *;
*     storm names have been used more than once, unique   *;
*     storms are identified by both Season and Name.      *;
*  2) Open PG2.STORM_DAMAGE. Notice that it does not      *;
*     include the columns Season and Name, which are in   *;
*     STORM_FINAL_SORT. Season and Name must be derived   *;
*     from the Date and Event columns.                  *;
*  3) Examine the DATA step that creates a temporary      *;
*     table named STORM_DAMAGE. SAS functions are used to *;
*     create Season and Name with values that match the   *;
*     values in the STORM_FINAL_SORT table. Highlight the *;
*     DATA step and the PROC SORT step that follows it,   *;
*     and run the selection.                          *;
*  4) Complete the final DATA step to merge the sorted    *;
*     tables by Season and Name. Highlight the DATA step  *;
*     and run the selection. Notice in the output table   *;
*     that row 4 is storm Allen, which is included in the *;
*     STORM_DAMAGE table. Therefore, each of the columns  *;
*     has values read from both input tables. Most of the *;
*     values in the Cost column are missing because those *;
*     storms are not found in the STORM_DAMAGE table.     *;
*  5) Use the IN= data set option after the STORM_DAMAGE  *;
*    table to create a temporary variable named inDamage *;
*    that flags rows where Season and Name were read     *;
*    from the STORM_DAMAGE table. Add a subsetting IF    *;
*    statement to write the 38 rows from STORM_DAMAGE    *;
*    and the corresponding data from STORM_FINAL_SORT to *;
*    the output table. Highlight the DATA step and run   *;
*    the selection.                                       *;
***********************************************************;

proc sort data=pg2.storm_final out=storm_final_sort;
	by Season Name;
run;

data storm_damage;
	set pg2.storm_damage;
	Season=Year(date);
	Name=upcase(scan(Event, -1));
	format Date date9. Cost dollar16.;
	drop event;
run;

proc sort data=storm_damage;
	by Season Name;
run;

data damage_detail;
	merge storm_final_sort storm_damage(in=inDamage);
	keep Season Name BasinName MaxWindMPH MinPressure Cost;
	if inDamage=1;
```

```
run;
```

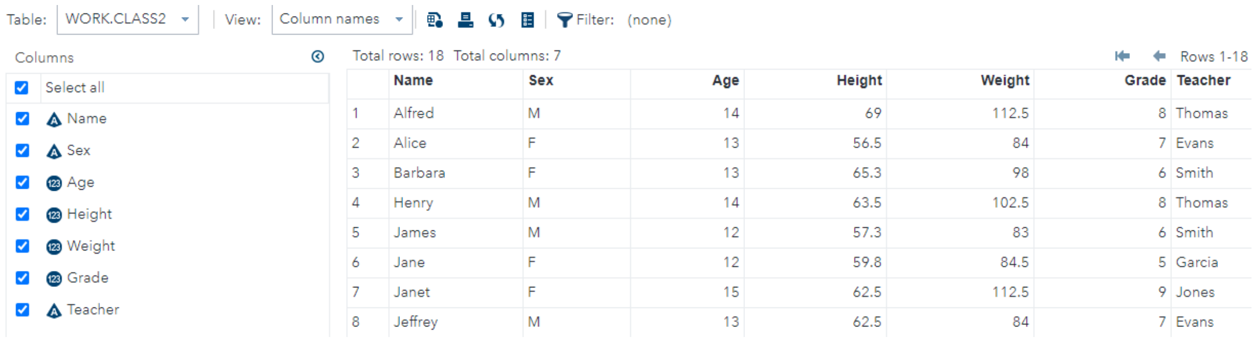


Table: WORK.CLASS2 View: Column names Filter: (none)

Columns

☑ Select all

☑ Name

☑ Sex

☑ Age

☑ Height

☑ Weight

☑ Grade

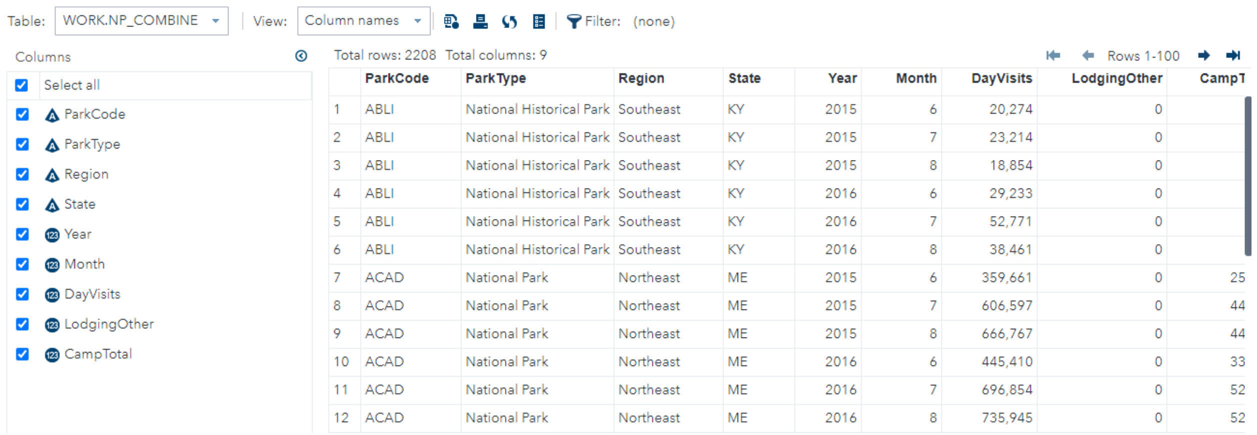
☑ Teacher

Total rows: 18 Total columns: 7 Rows 1-18

	Name	Sex	Age	Height	Weight	Grade	Teacher
1	Alfred	M	14	69	112.5	8	Thomas
2	Alice	F	13	56.5	84	7	Evans
3	Barbara	F	13	65.3	98	6	Smith
4	Henry	M	14	63.5	102.5	8	Thomas
5	James	M	12	57.3	83	6	Smith
6	Jane	F	12	59.8	84.5	5	Garcia
7	Janet	F	15	62.5	112.5	9	Jones
8	Jeffrey	M	13	62.5	84	7	Evans

```
***********************************************************;
*  p205p01.sas LESSON 5, PRACTICE 1                        *;
*  a) Complete the SET statement to concatenate the        *;
*     PG2.NP_2015 and PG2.NP_2016 tables to create a new  *;
*     table, NP_COMBINE.                                   *;
*  b) Use a WHERE statement to include only rows where     *;
*     Month is 6, 7, or 8.                                 *;
*  c) Create a new column named CampTotal that is the sum *;
*     of CampingOther, CampingTent, CampingRV, and         *;
*     CampingBackcountry. Format the new column with       *;
*     commas.                                              *;
***********************************************************;


*Practice2;
data work.np_combine;
   set pg2.np_2015 pg2.np_2016 pg2.np_2014(rename=(Park=ParkCode Type=ParkType));
   CampTotal=sum(of Camping:);
   where Month in (6, 7, 8) and ParkType = 'National Park';
   format CampTotal comma15.;
   drop Camping:;
run;
```

```
proc sort data=work.np_combine;
          by ParkType ParkCode Year Month;
run;


*Practice1;
data work.np_combine;
   set pg2.np_2015 pg2.np_2016;
   drop Camping:;
   where Month IN (6,7,8);
   CampTotal=sum(CampingOther, CampingTent, CampingRV, CampingBackcountry);
   format CampTotal commas16.;
run;


proc sort data=work.np_combine;
          by ParkCode;
run;
```



Total rows: 2208 Total columns: 9

	ParkCode	ParkType	Region	State	Year	Month	DayVisits	LodgingOther	CampT
1	ABLI	National Historical Park	Southeast	KY	2015	6	20,274	0	
2	ABLI	National Historical Park	Southeast	KY	2015	7	23,214	0	
3	ABLI	National Historical Park	Southeast	KY	2015	8	18,854	0	
4	ABLI	National Historical Park	Southeast	KY	2016	6	29,233	0	
5	ABLI	National Historical Park	Southeast	KY	2016	7	52,771	0	
6	ABLI	National Historical Park	Southeast	KY	2016	8	38,461	0	
7	ACAD	National Park	Northeast	ME	2015	6	359,661	0	25
8	ACAD	National Park	Northeast	ME	2015	7	606,597	0	44
9	ACAD	National Park	Northeast	ME	2015	8	666,767	0	44
10	ACAD	National Park	Northeast	ME	2016	6	445,410	0	33
11	ACAD	National Park	Northeast	ME	2016	7	696,854	0	52
12	ACAD	National Park	Northeast	ME	2016	8	735,945	0	52

```
********************************************************;
*  p205p03.sas LESSON 5, PRACTICE 3                   *;
*  a) Submit the two PROC SORT steps. Determine the name  *;
*     of the common column in the sorted tables.        *;
*  b) Modify the second PROC SORT step to use the RENAME= *;
*    option after the PG2.NP_2016TRAFFIC table to rename *;
*    Code to ParkCode. Modify the BY statement to sort   *;
*    by the new column name.                             *;
*  c) Write a DATA step to merge the sorted tables by the *;
*    common column to create a new table,                *;
*    WORK.TRAFFICSTATS. Drop the Name_Code column from   *;
*    the output table.                                   *;
*********************************************************;

proc sort data=pg2.np_codelookup out=work.codesort;
        by ParkCode;
run;

proc sort data=pg2.np_2016traffic(rename=(Code=ParkCode)) out=work.traf2016Sort;
        by ParkCode month;
run;

data work.trafficstats(drop=Name_Code);
        merge codesort traf2016Sort;
        by ParkCode;
run;
```

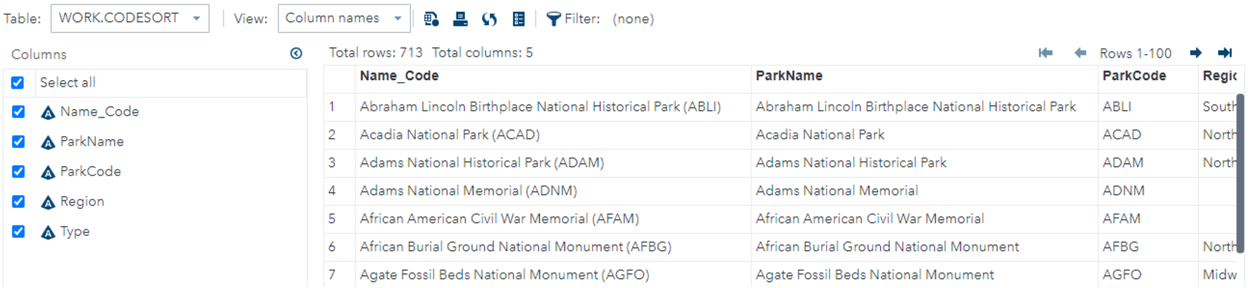


Table: WORK.CODESORT | View: Column names | Filter: (none)

Columns

☑ Select all
☑ Name_Code
☑ ParkName
☑ ParkCode
☑ Region
☑ Type

Total rows: 713 Total columns: 5   Rows 1-100

	Name_Code	ParkName	ParkCode	Regic
1	Abraham Lincoln Birthplace National Historical Park (ABLI)	Abraham Lincoln Birthplace National Historical Park	ABLI	South
2	Acadia National Park (ACAD)	Acadia National Park	ACAD	North
3	Adams National Historical Park (ADAM)	Adams National Historical Park	ADAM	North
4	Adams National Memorial (ADNM)	Adams National Memorial	ADNM	
5	African American Civil War Memorial (AFAM)	African American Civil War Memorial	AFAM	
6	African Burial Ground National Monument (AFBG)	African Burial Ground National Monument	AFBG	North
7	Agate Fossil Beds National Monument (AGFO)	Agate Fossil Beds National Monument	AGFO	Midw

/*p205p04.sas Level 2 Practice: Writing Matches and Nonmatches to Separate Tables

TOTAL POINTS 2

1.

Question 1

The pg2.np_2016 table contains monthly public use statistics from the National Park Service for parks by ParkCode.

The pg2.np_codelookup table contains the full name for each park code value.

Create a table, parkStats, that contains all park codes found in the np_2016 table.

Create a second table, parkOther, that contains ParkCode values in the np_codelookup table, but not in the np_2016 table.

If necessary, start SAS Studio before you begin.

Reminder: If you restarted your SAS session, submit your libname.sas program to access the practice data.

Determine the name of the common column in the pg2.np_codelookup and pg2.np_2016 tables.

Write a new program to sort the data in both tables by the matching column.

Using a DATA step, merge the pg2.np_codelookup and pg2.np_2016 tables to create two new tables named work.parkStats and work.parkOther.

The work.parkStats table should contain only ParkCode values that are in the np_2016 table, and

it should only the ParkCode, ParkName, Year, Month, and DayVisits columns.

The work.parkOther table should contain all other rows, and it should include only the ParkCode and ParkName columns.

Submit the program and examine the output data.

How many rows are in the parkStats table?

Question 2

How many rows are in the parkOther table?

1 point

*/

```
proc sort data=pg2.np_codelookup out=work.np_code_sort;
        by ParkCode;
run;

proc sort data=pg2.np_2016 out=work.np_2016_sort;
        by ParkCode;
run;

data parkStats parkOther(keep=ParkCode ParkName);
        merge np_code_sort np_2016_sort(in=inStats);
        by ParkCode;
        keep ParkCode ParkName Year Month DayVisits;
        if inStats=1 then output parkStats;
        else output parkOther;
run;
```

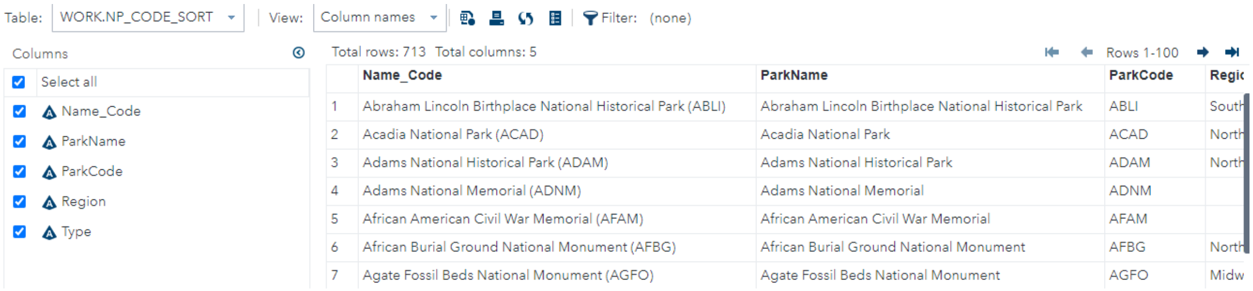


Table: WORK.NP_CODE_SORT | View: Column names | Filter: (none)

Columns

☑ Select all
☑ Name_Code
☑ ParkName
☑ ParkCode
☑ Region
☑ Type

Total rows: 713 Total columns: 5 | Rows 1-100

	Name_Code	ParkName	ParkCode	Regic
1	Abraham Lincoln Birthplace National Historical Park (ABLI)	Abraham Lincoln Birthplace National Historical Park	ABLI	South
2	Acadia National Park (ACAD)	Acadia National Park	ACAD	North
3	Adams National Historical Park (ADAM)	Adams National Historical Park	ADAM	North
4	Adams National Memorial (ADNM)	Adams National Memorial	ADNM	
5	African American Civil War Memorial (AFAM)	African American Civil War Memorial	AFAM	
6	African Burial Ground National Monument (AFBG)	African Burial Ground National Monument	AFBG	North
7	Agate Fossil Beds National Monument (AGFO)	Agate Fossil Beds National Monument	AGFO	Midw

```
**********************************************************;
*  p206a01.sas Activity 6.01                            *;
*  1) In the DATA step, add the following sum statement  *;
*     after the Savings sum statement to add 2% interest  *;
*     compounded monthly:                               *;
*       Savings+(Savings*0.02/12);                      *;
*  2) Run the program. How much is in savings at month 12?*;
*  3) Delete the OUTPUT statement and run the program     *;
*     again.                                            *;
*  4) How many rows are created?                    *;
*  5) What is the value of Month?                   *;
*  6) What is the value of Savings?                 *;
****************************************************;


data YearlySavings;
  Amount=200;
  do Month=1 to 12;
    Savings+Amount;
           *add a SUM Statement;
           Savings+(Savings*0.02/12);
           *output;
  end;
  format Savings 12.2;
run;
```

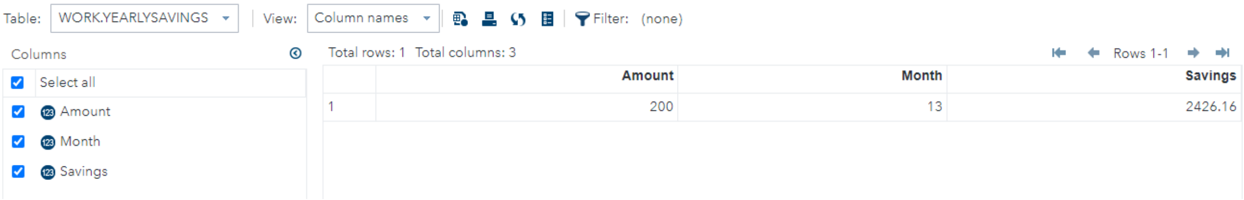


Table: WORK.YEARLYSAVINGS View: Column names Filter: (none)

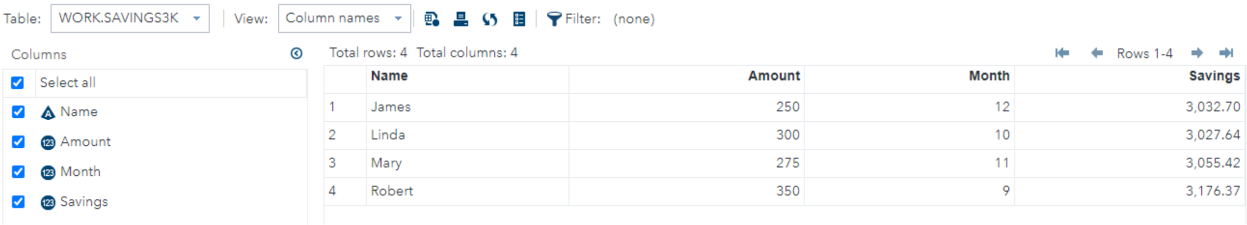
Columns

☑ Select all
☑ Amount
☑ Month
☑ Savings

Total rows: 1 Total columns: 3 Rows 1-1

	Amount	Month	Savings
1	200	13	2426.16

```
****************************************************;
*  p206a02.sas Activity 6.02                        *;
*  1) Run the program and view the Savings3K table.      *;
*  2) How many months until James exceeds 3000 in       *;
*     savings?                               *;
*  3) How much savings does James have at that month?    *;
*  4) Change the DO UNTIL statement to a DO WHILE        *;
*     statement and modify the expression to produce the  *;
*     same results.                          *;
*  5) Run the program and view the Savings3K table.      *;
*  6) Are the results for James identical with the DO     *;
*     WHILE as compared to the DO UNTIL?                  *;
**********************************************************;


data Savings3K;
  set pg2.savings;
  Month=0;
  Savings=0;
  do while (Savings<=3000);
    Month+1;
    Savings+Amount;
    Savings+(Savings*0.02/12);
  end;
  format Savings comma12.2;
run;
```



Total rows: 4 Total columns: 4

	Name	Amount	Month	Savings
1	James	250	12	3,032.70
2	Linda	300	10	3,027.64
3	Mary	275	11	3,055.42
4	Robert	350	9	3,176.37

```
**********************************************************;
*  p206d01a.sas Executing an Iterative DO Loop            *;
**********************************************************;
*  Syntax                                 *;
*                                         *;
*    DATA output-table;                   *;
*      . . .                              *;
*      DO index-column = start TO stop <BY increment>;  *;
*        . . . repetitive code . . .      *;
*      END;                               *;
*     . . .                       *;
*   RUN;                          *;
*******************************************************;

*******************************************************;
*  Demo                                   *;
*  1) Run the program and view the Forecast output table. *;
*     Notice that there are three rows (Year 1, 2, and 3) *;
*     for each combination of Region, Product, and       *;
*     Subsidiary.                         *;
*  2) Return to the Program tab and click the DATA step   *;
*     markers for debugging button to enable debugging in *;
*     the program if it is not already enabled. Click the *;
*     Debugger icon next to the DATA statement. The DATA  *;
*     Step Debugger window appears.                 *;
*  3) Click the Step execution to next line button to     *;
*     execute the highlighted SET statement.           *;
*  4) Click the button again to execute the highlighted   *;
*     DO statement. Notice that the Year value has been   *;
*     set to 1.                           *;
*  5) Click the button three times to execute the         *;
*     statements inside the DO loop and the END          *;
*     statement. Notice that the Year value has been      *;
*     incremented to 2 and that processing returns to the *;
*     inside of the DO loop.                     *;
*  6) Continue to click the button to execute the         *;
*     highlighted statements inside the DO loop. Observe  *;
*     the changing of values in the PDV.               *;
*  7) At the end of third iteration of the DO loop,       *;
*    notice that the Year value is incremented to 4 and  *;
*    that processing does not return to the inside of    *;
*    the DO loop.                                        *;
*  8) Close the DATA Step Debugger.                      *;
***********************************************************;


data forecast;
   set sashelp.shoes(rename=(Sales=ProjectedSales));
   do Year = 1 to 3;
      ProjectedSales=ProjectedSales*1.05;
      output;
   end;
   keep Region Product Subsidiary Year ProjectedSales;
   format ProjectedSales dollar10.;
run;
```

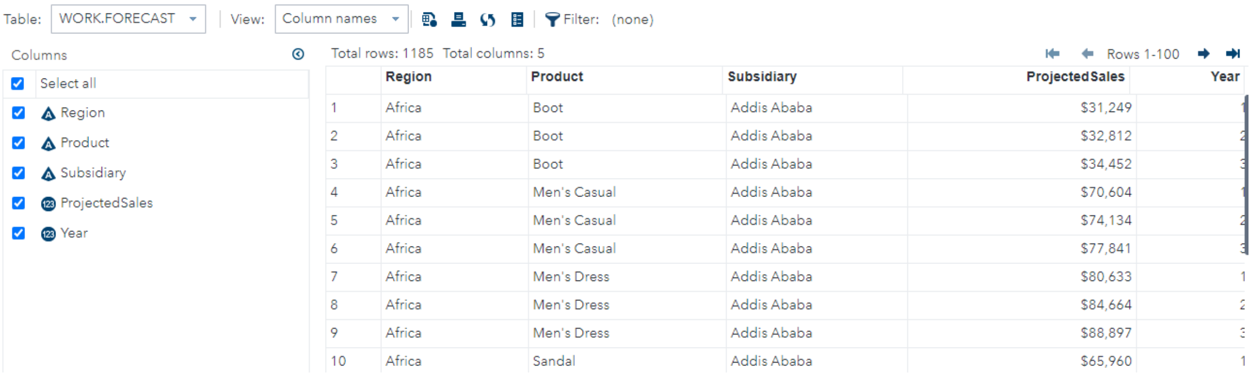


Table: WORK.FORECAST | View: Column names | Filter: (none)

Columns

☑ Select all
☑ Region
☑ Product
☑ Subsidiary
☑ ProjectedSales
☑ Year

Total rows: 1185 Total columns: 5   Rows 1-100

	Region	Product	Subsidiary	ProjectedSales	Year
1	Africa	Boot	Addis Ababa	$31,249	1
2	Africa	Boot	Addis Ababa	$32,812	2
3	Africa	Boot	Addis Ababa	$34,452	3
4	Africa	Men's Casual	Addis Ababa	$70,604	1
5	Africa	Men's Casual	Addis Ababa	$74,134	2
6	Africa	Men's Casual	Addis Ababa	$77,841	3
7	Africa	Men's Dress	Addis Ababa	$80,633	1
8	Africa	Men's Dress	Addis Ababa	$84,664	2
9	Africa	Men's Dress	Addis Ababa	$88,897	3
10	Africa	Sandal	Addis Ababa	$65,960	1

```
***********************************************************;
*  p206d01b.sas Executing an Iterative DO Loop           *;
***********************************************************;
*  Syntax                              *;
*                                   *;
*    DATA output-table;                   *;
*       . . .                        *;
*       DO index-column = start TO stop <BY increment>;  *;
*          . . . repetitive code . . .                   *;
*       END;                                        *;
*       . . .                                    *;
*    RUN;                                           *;
***********************************************************;

***********************************************************;
*  Demo                                             *;
*  1) Notice the three PUTLOG statements in the DATA step.*;
*  2) Run the program and view the Forecast output table. *;
*     Notice that there are three rows (Year 1, 2, and 3) *;
*     for the first two input rows.                      *;
*  3) View the PUTLOG text in the SAS log.               *;
***********************************************************;

data forecast;
   putlog 'Top of DATA Step ' Year= _N_=;
   set sashelp.shoes(obs=2 rename=(Sales=ProjectedSales));
   do Year = 1 to 3;
      ProjectedSales=ProjectedSales*1.05;
      output;
      putlog 'Value of Year written to table: ' Year=;
   end;
   putlog 'Outside of DO Loop: ' Year=;
   keep Region Product Subsidiary Year ProjectedSales;
   format ProjectedSales dollar10.;
run;
```

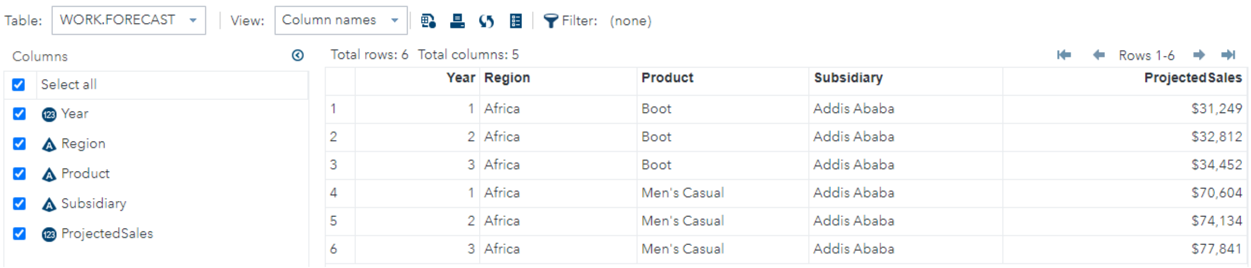


Table: WORK.FORECAST View: Column names Filter: (none)

Columns

☑ Select all
☑ Year
☑ Region
☑ Product
☑ Subsidiary
☑ ProjectedSales

Total rows: 6 Total columns: 5 Rows 1-6

	Year	Region	Product	Subsidiary	ProjectedSales
1	1	Africa	Boot	Addis Ababa	$31,249
2	2	Africa	Boot	Addis Ababa	$32,812
3	3	Africa	Boot	Addis Ababa	$34,452
4	1	Africa	Men's Casual	Addis Ababa	$70,604
5	2	Africa	Men's Casual	Addis Ababa	$74,134
6	3	Africa	Men's Casual	Addis Ababa	$77,841

```
***********************************************************;
*  p206d02.sas Using Iterative DO Loops                 *;
***********************************************************;
*  Syntax                            *;
*                                  *;
*    DATA output-table;                       *;
*       SET input-table;                     *;
*       . . .                             *;
*       DO index-column = start TO stop <BY increment>;  *;
*          . . . repetitive code . . .            *;
*          OUTPUT;                         *;
*       END;                            *;
*       . . .                             *;
*       OUTPUT;                          *;
*    RUN;                               *;
***********************************************************;
*  Demo                               *;
*  1) Open the PG2.SAVINGS table. Notice there are four  *;
*     rows representing different people. The Amount     *;
*     value is a monthly savings value.            *;
*  2) Run the program and notice that four rows are     *;
*     created due to four rows being read from the input  *;
*     table. Also, notice how the Savings value keeps     *;
*     increasing for each row.                 *;
*  3) Fix the issue by adding an assignment statement     *;
*    before the DO loop to set the value of Savings to   *;
*    0. Run the program and notice the correct values    *;
*    for Savings.                                *;
*  4) Add an outer DO loop to iterate through five years  *;
*    per each of the 12 months. Run the program and      *;
*    notice that you have one row per each person. Each  *;
*    row represents the savings after five years,        *;
*    assuming that savings are added each month. The     *;
*    value of Year is 6 and the value of Month is 13, an *;
*    increment beyond each stop value.                 *;
*  5) Add an OUTPUT statement to the bottom of the outer  *;
*    DO loop. Run the program and notice that you now    *;
*    have 5 rows per each person (a total of 20 rows).   *;
*    Each row represents the savings at each of the five *;
*    years.                                      *;
*  6) Move the OUPUT statement to the bottom of the inner *;
*    DO loop. Run the program and notice that you now    *;
*    have 60 rows per each person (a total of 240 rows). *;
*    Each row represents the savings at each year and    *;
*    month combination.                          *;
************************************************************;

*Demo3: Move output inside the inner loop;
data YearSavings;
        set pg2.savings;
        *add an assignment statement;
        Savings=0;
        do Year=1 to 5;
        do Month=1 to 12;
            Savings+Amount;
            Savings+(Savings*0.02/12);
            output;
        end;
    end;
    format Savings comma12.2;
run;


*Demo2: add yearly savings as outer loop and output after inner loop finishes;
data YearSavings;
    set pg2.savings;
    *add an assignment statement;
    Savings=0;
    do Year=1 to 5;
        do Month=1 to 12;
            Savings+Amount;
            Savings+(Savings*0.02/12);
        end;
        output;
    end;
    format Savings comma12.2;
run;

*Demo1: reset the savings amounts, so it would not accumulate to next individual savings;
data YearSavings;
    set pg2.savings;
    *add an assignment statement;
```

```
        Savings=0;
                do Month=1 to 12;
                        Savings+Amount;
                        Savings+(Savings*0.02/12);
                end;
        format Savings comma12.2;
run;

*Original;
data YearSavings;
        set pg2.savings;
        *add an assignment statement;
                do Month=1 to 12;
                        Savings+Amount;
                        Savings+(Savings*0.02/12);
                end;
        format Savings comma12.2;
run;
```

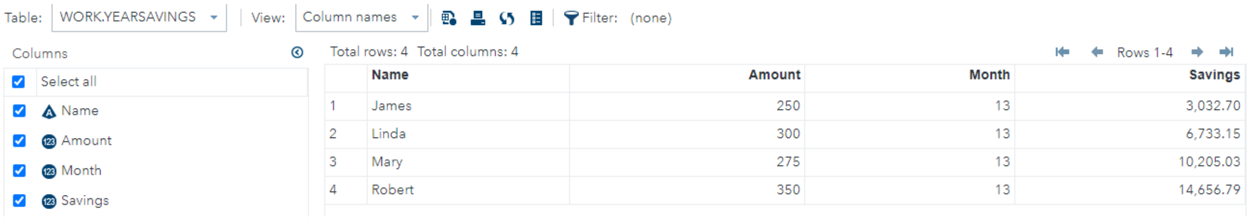


Table: WORK.YEARSAVINGS View: Column names Filter: (none)

Columns

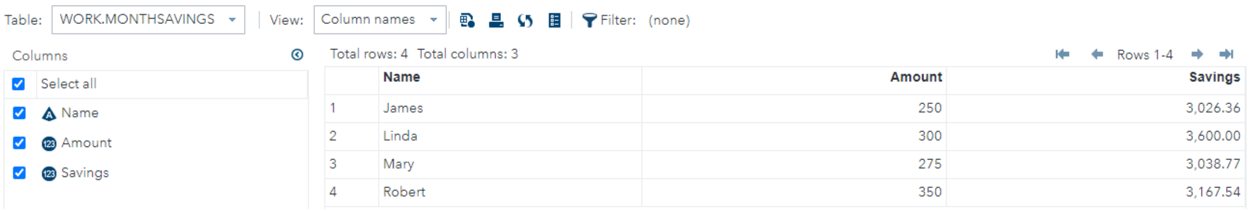
☑ Select all
☑ Name
☑ Amount
☑ Month
☑ Savings

Total rows: 4 Total columns: 4 Rows 1-4

	Name	Amount	Month	Savings
1	James	250	13	3,032.70
2	Linda	300	13	6,733.15
3	Mary	275	13	10,205.03
4	Robert	350	13	14,656.79

```
***********************************************************;
*  p206d03.sas Using Conditional DO Loops              *;
***********************************************************;
*  Syntax                          *;
*                                *;
*    DATA output-table;                   *;
*      SET input-table;                 *;
*      . . .                                  *;
*      DO UNTIL | WHILE (expression);                  *;
*        . . . repetitive code . . .                  *;
*        OUTPUT;                                  *;
*      END;                                  *;
*      DO index-column = start TO stop <BY increment>    *;
*        UNTIL | WHILE (expression);                  *;
*        . . . repetitive code . . .                  *;
*        OUTPUT;                                  *;
*      END;                                  *;
*      . . .                                  *;
*      OUTPUT;                                  *;
*    RUN;                                  *;
***********************************************************;

***********************************************************;
*  Demo                                        *;
*  1) Open the PG2.SAVINGS2 table. This table contains a  *;
*     column named Savings that is the current value of   *;
*     each person's savings account. Notice that Linda's  *;
*     value is already greater than 3000.                *;
*  2) Notice the DO UNTIL expression is Savings equal to  *;
*     3000. Run the program. Because Savings is never     *;
*     equal to 3000, the program is in an infinite loop.  *;
*     Stop the infinite DO loop from running.            *;
*     * In SAS Enterprise Guide, click the Stop toolbar   *;
*       button on the Program tab.                      *;
*     * In SAS Studio, click Cancel in the Running pop-up *;
*       window.                                  *;
*  3) Make the following modifications to the DATA step.  *;
*     a) Replace the equal sign with a greater than       *;
*        symbol.                                  *;
*     b) Add a sum statement inside the DO loop to create *;
*        a column named Month that will increment by 1    *;
*        for each loop.                           *;
*     c) Before the DO loop add an assignment statement   *;
*        to reset Month to 0 each time a new row is read  *;
*        from the input table.                    *;
*  4) Run the program. Notice that even though Linda      *;
*     began with 3600 for Savings, the DO LOOP executed   *;
*     once.                                       *;
*  5) Change the DO UNTIL expression to DO WHILE so that  *;
*     the condition will be checked at the top of the     *;
*     loop. Run the program and verify Linda's Savings    *;
*     amount is 3600.                             *;
**************************************************************;

data MonthSavings;
   set pg2.savings2;
   do until (Savings>3000);
      Savings+Amount;
      Savings+(Savings*0.02/12);
   end;
   format Savings comma12.2;
run;


data MonthSavings;
```

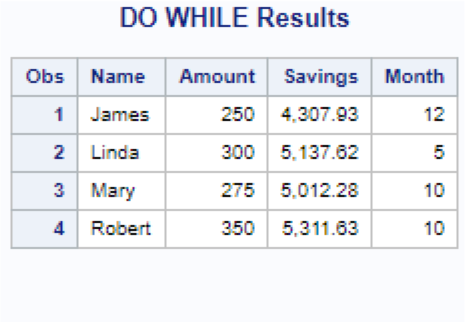
```
  set pg2.savings2;
  do while (Savings<3000);
    Savings+Amount;
    Savings+(Savings*0.02/12);
  end;
  format Savings comma12.2;
run;
```



Total rows: 4 Total columns: 3

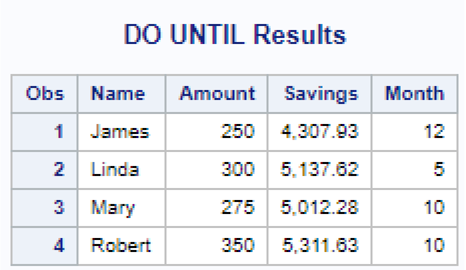
	Name	Amount	Savings
1	James	250	3,026.36
2	Linda	300	3,600.00
3	Mary	275	3,038.77
4	Robert	350	3,167.54

```
**********************************************************;
*  p206d04.sas Combining Iterative and Conditional DO Loops          *;
**********************************************************;
*  Syntax                                      *;
*                                              *;
*   DATA output-table;                              *;
*     SET input-table;                            *;
*     . . .                                    *;
*     DO UNTIL | WHILE (expression);                  *;
*       . . . repetitive code . . .                  *;
*       OUTPUT;                                *;
*     END;                                     *;
*     DO index-column = start TO stop <BY increment>    *;
*       UNTIL | WHILE (expression);                  *;
*       . . . repetitive code . . .                  *;
*       OUTPUT;                                *;
*     END;                                     *;
*     . . .                                    *;
*     OUTPUT;                                  *;
*   RUN;                                       *;
**********************************************************;

**********************************************************;
*  Demo                                        *;
*  1) The intent of both DATA steps is process the DO     *;
*    loop for each row in the PG2.SAVINGS2 table. One    *;
*    DATA step uses DO WHILE and the other uses DO       *;
*    UNTIL. Each loop represents one month of savings.   *;
*    The loop should stop iterating when Savings exceeds *;
*    3000 or 12 months pass, whichever comes first.      *;
*  2) Run the demo program and view the 2 reports that    *;
*    are created. Notice that the values of Savings in   *;
*    the DO WHILE and DO UNTIL reports match, indicating *;
*    that the DO loops executed the same number of times *;
*    for each person.                                *;
*  3) Observe that for the first row in both the DO WHILE *;
*    and DO UNTIL reports has Month equal to 13. Savings *;
*    did not exceed $5,000 after 12 iterations of the DO *;
*    loop. The Month index variable was incremented to   *;
*    13 at the end of the twelfth iteration of the loop, *;
*    which triggered the end of the loop in both DATA    *;
*    steps and an implicit output action to the output   *;
*    table.                                        *;
*  4) Observe that in rows 2, 3 and 4, the value of Month *;
*    in the DO WHILE results is one greater compared to  *;
*    the DO UNTIL results. This is because in the DO     *;
*    WHILE loop, the index variable Month increments     *;
*    before the condition is checked. Therefore, the     *;
*    Month column in the output data does not accurately *;
*    represent the number of times the DO loop iterated  *;
*    in either DATA step.                            *;
*  5) To create an accurate counter for the number of     *;
*    iterations of a DO loop, make the following         *;
*    modifications to both DATA steps:                  *;
*    a) Add a sum statement inside the loop to create a  *;
*       column named Month and add 1 for each iteration. *;
*    b) Before the DO loop add an assignment statement   *;
*       to reset Month to 0 each time a new row is read  *;
*       from the input table.                        *;
*     c) Change the name of the index variable to an      *;
*       arbitrary name, such as i.                    *;
*     d) Add a DROP statement to drop i from the output   *;
*       table.                                   *;
*  6) Run the program and examine the results. Notice the *;
*     values of Savings and Month match for the DO WHILE  *;
*     and DO UNTIL reports. Month represents the number   *;
*     of times the DO loop executed for each row.        *;
********************************************************************;

data MonthSavingsW;
   set pg2.savings2;
   Month=0;
   do i=1 to 12 while (savings<=5000);
      Month+1;
      Savings+Amount;
      Savings+(Savings*0.02/12);
   end;
   format Savings comma12.2;
   drop i;
run;

data MonthSavingsU;
   set pg2.savings2;
   Month=0;
   do i=1 to 12 until (savings>5000);
      Month+1;
      Savings+Amount;
      Savings+(Savings*0.02/12);
   end;
   format Savings comma12.2;
   drop i;
run;

title "DO WHILE Results";
proc print data=MonthSavingsW;
run;

title "DO UNTIL Results";
proc print data=MonthSavingsU;
run;

*Original;
data MonthSavingsW;
   set pg2.savings2;
   do Month=1 to 12 while (savings<=5000);
      Savings+Amount;
      Savings+(Savings*0.02/12);
   end;
   format Savings comma12.2;
run;

data MonthSavingsU;
   set pg2.savings2;
   do Month=1 to 12 until (savings>5000);
      Savings+Amount;
      Savings+(Savings*0.02/12);
   end;
   format Savings comma12.2;
run;
```



DO WHILE Results

Obs	Name	Amount	Savings	Month
1	James	250	4,307.93	12
2	Linda	300	5,137.62	5
3	Mary	275	5,012.28	10
4	Robert	350	5,311.63	10

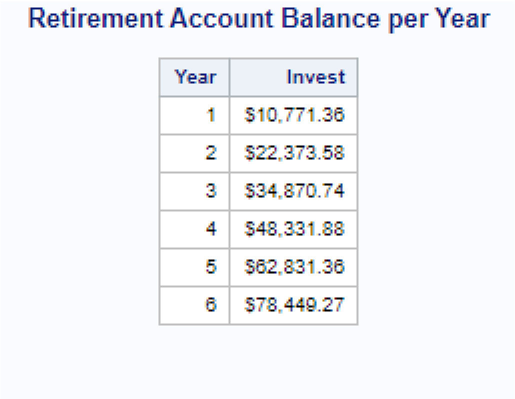


DO UNTIL Results

Obs	Name	Amount	Savings	Month
1	James	250	4,307.93	12
2	Linda	300	5,137.62	5
3	Mary	275	5,012.28	10
4	Robert	350	5,311.63	10

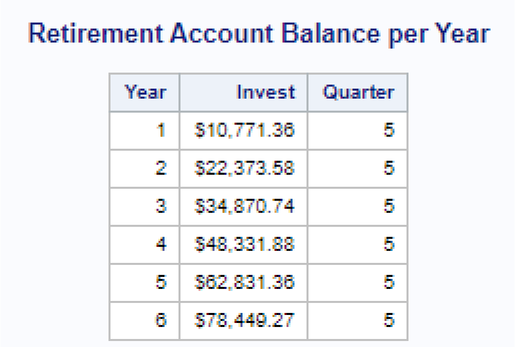
```
***********************************************************;
*  p206p01.sas LESSON 6, PRACTICE 1                        *;
*  a) Add an iterative DO loop around the sum statement   *;
*     for Invest.                                          *;
*     1) Add a DO statement that creates the column Year  *;
*        with values ranging from 1 to 6.                  *;
*     2) Add an OUTPUT statement to show the value of the *;
*        retirement account for each year.                 *;
*     3) Add an END statement.                             *;
*  b) Run the program and review the results.              *;
*  c) Add an inner iterative DO loop between the sum       *;
*     statement and the OUTPUT statement to include the   *;
*     accrued quarterly compounded interest based on an   *;
*     annual interest rate of 7.5%.                        *;
*     1) Add a DO statement that creates the column        *;
*       Quarter with values ranging from 1 to 4.        *;
*     2) Add a sum statement to add the accrued interest  *;
*       to the Invest value.                          *;
*          Invest+(Invest*(.075/4));                  *;
*     3) Add an END statement.                         *;
*  d) Run the program and review the results.            *;
*  e) Drop the Quarter column. Run the program and review *;
*     the results.                                   *;
***********************************************************;


*Practice3: Drop Quarter column;
data retirement;
            do Year=1 to 6;
      Invest+10000;
              do Quarter=1 to 4;
             Invest+(Invest*(0.075/4));
      end;
      output;
            end;
            Drop Quarter;
run;

title1 'Retirement Account Balance per Year';
proc print data=retirement noobs;
   format Invest dollar12.2;
run;
title;

*Practice2: add quarterly compound interest rate of 7.5% as inner loop;
```

```
data retirement;
            do Year=1 to 6;
      Invest+10000;
               do Quarter=1 to 4;
              Invest+(Invest*(0.075/4));
      end;
      output;
            end;
run;

title1 'Retirement Account Balance per Year';
proc print data=retirement noobs;
   format Invest dollar12.2;
run;
title;

*Practice1: add year 1 to 6 do loop;
data retirement;
            do Year=1 to 6;
      Invest+10000;
      output;
            end;
run;
```



Retirement Account Balance per Year

Year	Invest
1	$10,771.36
2	$22,373.58
3	$34,870.74
4	$48,331.88
5	$62,831.36
6	$78,449.27

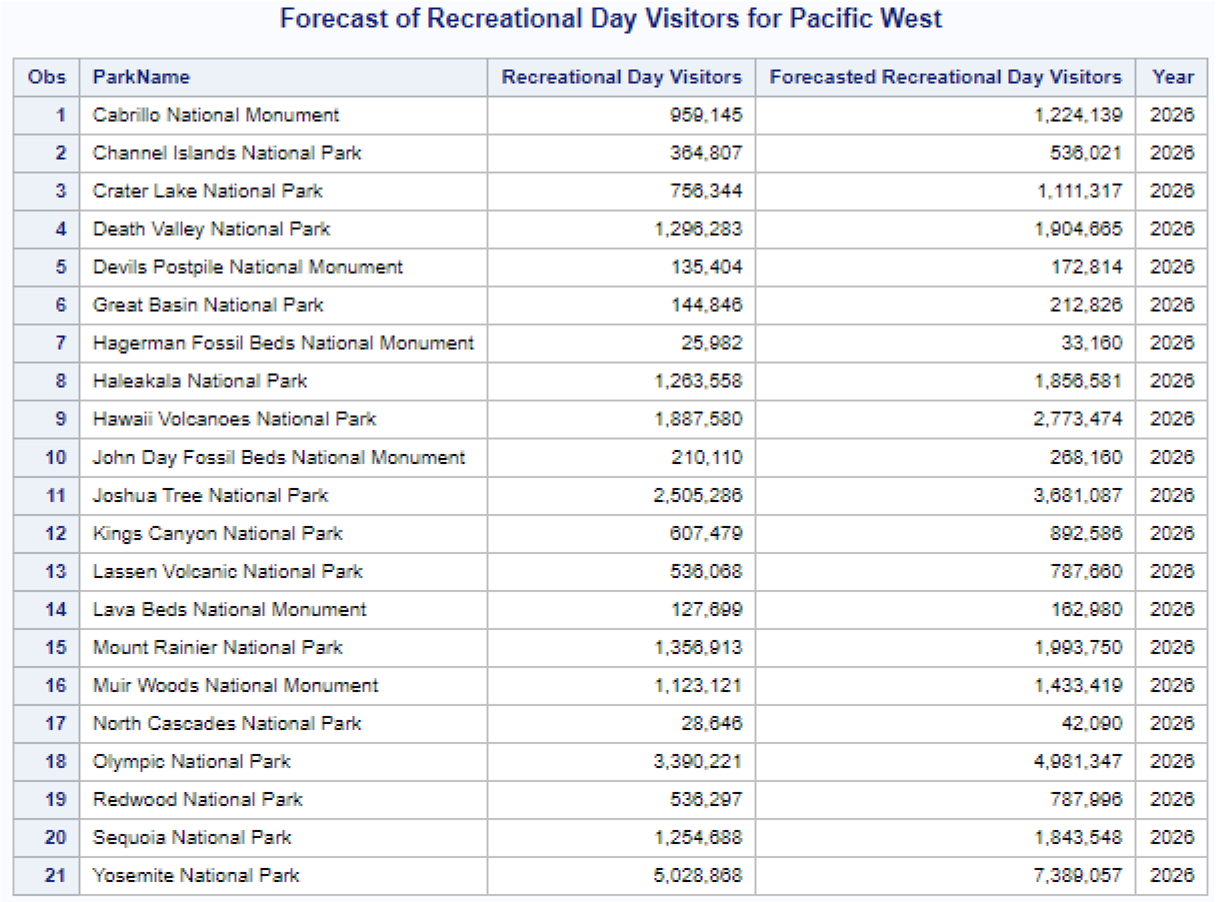


Retirement Account Balance per Year

Year	Invest	Quarter
1	$10,771.36	5
2	$22,373.58	5
3	$34,870.74	5
4	$48,331.88	5
5	$62,831.36	5
6	$78,449.27	5

```
***********************************************************;
*  p206p02.sas LESSON 6, PRACTICE 2                       *;
*  a) Run the program and review the results. Notice that *;
*     the initial program is showing the forecasted value *;
*     for the next year. The next year is based on adding *;
*     one year to the year value of today's date.         *;
*     Depending on the current date, your NextYear value  *;
*     might be bigger than the NextYear value in the      *;
*     following results.                                  *;
*  b) Add an iterative DO loop around the conditional     *;
*     IF-THEN statements.                                 *;
*     1) The DO loop needs to iterate five times.         *;
*     2) In the DO statement, a new column named Year     *;
*        needs to be created that starts at the value of  *;
*        NextYear and stops at the value of NextYear plus *;
*        4.                                               *;
*    3) A row needs to be created for each year.        *;
*  c) Modify the KEEP statement to keep the column Year   *;
*    instead of NextYear.                              *;
*  d) Run the program and review the results.           *;
*  e) (Optional) Modify the OUTPUT statement to be a      *;
*    conditional statement that outputs only on the      *;
*    fifth iteration. Run the program and review the     *;
*    results.                                  *;
************************************************************;

*Practice2: Modify the OUTPUT statement to conditionally output a row only on the fifth iteration;
data ForecastDayVisits;
   set pg2.np_summary;
   where Reg='PW' and Type in ('NM','NP');
   ForecastDV=DayVisits;
   NextYear=year(today())+1;

          do Year=NextYear to NextYear+4;
          if Type='NM' then ForecastDV=ForecastDV*1.05;
          if Type='NP' then ForecastDV=ForecastDV*1.08;
                    if Year=NextYear+4 then output;
          end;

   format ForecastDV comma12.;
   label ForecastDV='Forecasted Recreational Day Visitors';
   keep ParkName DayVisits ForecastDV Year;
run;

proc sort data=ForecastDayVisits;
```

```
   by ParkName;
run;

title 'Forecast of Recreational Day Visitors for Pacific West';
proc print data=ForecastDayVisits label;
run;
title;

*Practice1: add do loops from NextYear to NextYear+4 and add output the data each year;
data ForecastDayVisits;
   set pg2.np_summary;
   where Reg='PW' and Type in ('NM','NP');
   ForecastDV=DayVisits;
   NextYear=year(today())+1;

          do Year=NextYear to NextYear+4;
          if Type='NM' then ForecastDV=ForecastDV*1.05;
          if Type='NP' then ForecastDV=ForecastDV*1.08;
          output;
          end;

   format ForecastDV comma12.;
   label ForecastDV='Forecasted Recreational Day Visitors';
   keep ParkName DayVisits ForecastDV Year;
run;

*Original;
data ForecastDayVisits;
   set pg2.np_summary;
   where Reg='PW' and Type in ('NM','NP');
   ForecastDV=DayVisits;
   NextYear=year(today())+1;

         if Type='NM' then ForecastDV=ForecastDV*1.05;
         if Type='NP' then ForecastDV=ForecastDV*1.08;

   format ForecastDV comma12.;
   label ForecastDV='Forecasted Recreational Day Visitors';
   keep ParkName DayVisits ForecastDV NextYear;
run;
```



Forecast of Recreational Day Visitors for Pacific West

Obs	ParkName	Recreational Day Visitors	Forecasted Recreational Day Visitors	Year
1	Cabrillo National Monument	959,145	1,224,139	2026
2	Channel Islands National Park	364,807	536,021	2026
3	Crater Lake National Park	756,344	1,111,317	2026
4	Death Valley National Park	1,296,283	1,904,665	2026
5	Devils Postpile National Monument	135,404	172,814	2026
6	Great Basin National Park	144,846	212,826	2026
7	Hagerman Fossil Beds National Monument	25,982	33,160	2026
8	Haleakala National Park	1,263,558	1,856,581	2026
9	Hawaii Volcanoes National Park	1,887,580	2,773,474	2026
10	John Day Fossil Beds National Monument	210,110	268,160	2026
11	Joshua Tree National Park	2,505,286	3,681,087	2026
12	Kings Canyon National Park	607,479	892,586	2026
13	Lassen Volcanic National Park	536,068	787,660	2026
14	Lava Beds National Monument	127,699	162,980	2026
15	Mount Rainier National Park	1,356,913	1,993,750	2026
16	Muir Woods National Monument	1,123,121	1,433,419	2026
17	North Cascades National Park	28,646	42,090	2026
18	Olympic National Park	3,390,221	4,981,347	2026
19	Redwood National Park	536,297	787,996	2026
20	Sequoia National Park	1,254,688	1,843,548	2026
21	Yosemite National Park	5,028,868	7,389,057	2026

/* p206p04.sas

Level 1 Practice: Using a Conditional DO Loop

TOTAL POINTS 4

1.

Question 1

The pg2.np_summary table contains public use statistics from the National Park Service.

The Northeast region has seen an increase in visitors at its national monuments that previously experienced low visitation.

Determine the number of years it will take for the number of visitors to exceed 100,000, assuming an annual 6% increase.

If necessary, start SAS Studio before you begin.

Reminder: If you restarted your SAS session, submit your libname.sas program to access the practice data.

Open p206p04.sas from the practices folder.

Submit the program and examine the results.

Notice that the first two monuments are not near 100,000 visitors, but the third monument is near 100,000 after one year with a 6% increase.

Add a conditional DO loop around the assignment statement where IncrDayVisits is being increased by 6%.

The DO UNTIL statement should execute until the value of IncrDayVisits exceeds 100,000.

Write a row to the output table for each iteration of the DO loop.

Don't forget to add an END statement.

Submit the program and examine the results.

How many rows are in the IncreaseDayVisits table?

Suppose you want to add a Year column so you know how many years it takes to reach the visitor goal.

Before the DO loop, add an assignment statement to set the Year to 0.

Within the DO loop, add a sum statement to add 1 to the value of Year.

Add Year to the KEEP statement.

Submit the program and examine the results.

How many years does it take African Burial Ground National Monument to exceed 100,000 visitors?

Question 3

How would you modify the code if you want the output table to only include the row for each monument where the number of visitors exceeded 100,000?

1 point

Modify the DO UNTIL statement to be a DO WHILE statement that produces the same results.

Submit the program and verify the results.

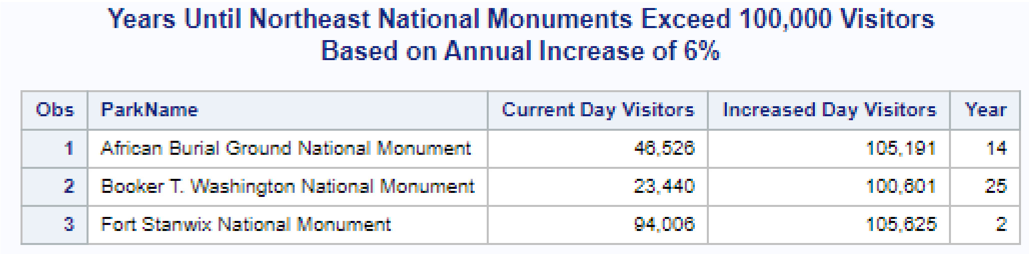
What DO WHILE loop statement did you use?

1 point

```
*/

**************************************************;

*  LESSON 6, PRACTICE 4                          *;

**************************************************;


*Practice4: Using Do While Modify to output to only include the row for each monument where the number of visitors exceeded 100,000;

data IncreaseDayVisits;

  set pg2.np_summary;

  where Reg='NE' and DayVisits<100000;

  IncrDayVisits=DayVisits;

  Year=0;

        do while(IncrDayVisits<=100000);

               Year+1;

        IncrDayVisits=IncrDayVisits*1.06;

        end;
```

```
          output;
   format IncrDayVisits comma12.;
   keep ParkName DayVisits IncrDayVisits Year;
run;


*Practice3: Modify to output to only include the row for each monument where the number of visitors exceeded 100,000;
data IncreaseDayVisits;
   set pg2.np_summary;
   where Reg='NE' and DayVisits<100000;
   IncrDayVisits=DayVisits;
   Year=0;
          do until(IncrDayVisits>100000);
                    Year+1;
          IncrDayVisits=IncrDayVisits*1.06;
                    if IncrDayVisits>100000 then output;
          end;
   format IncrDayVisits comma12.;
   keep ParkName DayVisits IncrDayVisits Year;
run;



proc sort data=IncreaseDayVisits;
   by ParkName;
run;


title1 'Years Until Northeast National Monuments Exceed 100,000 Visitors';
title2 'Based on Annual Increase of 6%';
proc print data=IncreaseDayVisits label;
```

```
   label DayVisits='Current Day Visitors'
         IncrDayVisits='Increased Day Visitors';
run;
title;

*Practice2: Add Year;
data IncreaseDayVisits;
   set pg2.np_summary;
   where Reg='NE' and DayVisits<100000;
   IncrDayVisits=DayVisits;
   Year=0;
          do i=1 to 100 until(IncrDayVisits>100000);
                    Year+1;
          IncrDayVisits=IncrDayVisits*1.06;
                    output;
          end;
   format IncrDayVisits comma12.;
   keep ParkName DayVisits IncrDayVisits Year;
run;

*Practice1: add do until IncrdayVisits>100000 and output each iteration;
data IncreaseDayVisits;
   set pg2.np_summary;
   where Reg='NE' and DayVisits<100000;
   IncrDayVisits=DayVisits;
          do until(IncrDayVisits>100000);
          IncrDayVisits=IncrDayVisits*1.06;
                    output;
          end;
   format IncrDayVisits comma12.;
   keep ParkName DayVisits IncrDayVisits;
run;


*Original;
data IncreaseDayVisits;
   set pg2.np_summary;
   where Reg='NE' and DayVisits<100000;
   IncrDayVisits=DayVisits;


   IncrDayVisits=IncrDayVisits*1.06;


   format IncrDayVisits comma12.;
   keep ParkName DayVisits IncrDayVisits;
run;
```



Years Until Northeast National Monuments Exceed 100,000 Visitors
Based on Annual Increase of 6%

Obs	ParkName	Current Day Visitors	Increased Day Visitors	Year
1	African Burial Ground National Monument	46,526	105,191	14
2	Booker T. Washington National Monument	23,440	100,601	25
3	Fort Stanwix National Monument	94,006	105,625	2

/*p206p05.sas Level 2 Practice: Using an Iterative and Conditional DO Loop

TOTAL POINTS 3

Question 1

The pg2.eu_sports table contains European Union trade amounts for sport products.

Belgium wants to see their exports exceed their imports for golf and racket products.

They expect exports to increase annually by 7% and want to achieve their goal within 10 years.

If necessary, start SAS Studio before you begin.

Reminder: If you restarted your SAS session, submit your libname.sas program to access the practice data.

Open p206p05.sas from the practices folder.

Submit the program and examine the results.

Notice that the golf export number is farther from the golf import number as compared to the racket export and import numbers.

Add a conditional DO loop around the assignment statement for Amt_Export.

Use a DO WHILE statement that executes while the export value is less than or equal to the import value.

Create a Year column that increments by a value of 1.

Create a row of output for each year.

Submit the program and examine the results.

How many years does take until the exports exceed imports for Racket products?

Question 2

Suppose you only want to forecast for 10 years.

Modify the DO statement to include an iterative portion before the conditional portion.

The iterative portion needs to be based on Year values of 2016 to 2025 (10 years).

Within the DO loop, delete any statements that increment Year.

Submit the program and review the results.

How many rows are in the output table?

Question 3

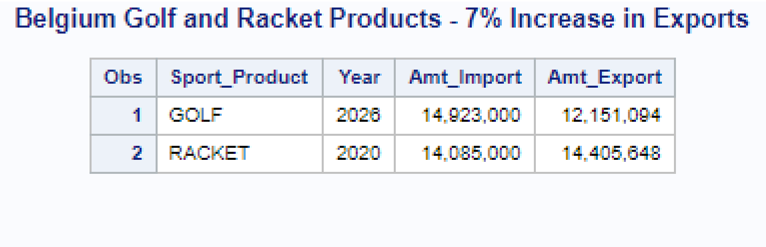
In the last output table, the final year for Golf products was 2025 and the final year for Racket products was 2019.

Delete the OUTPUT statement.

Submit the program and examine the results.

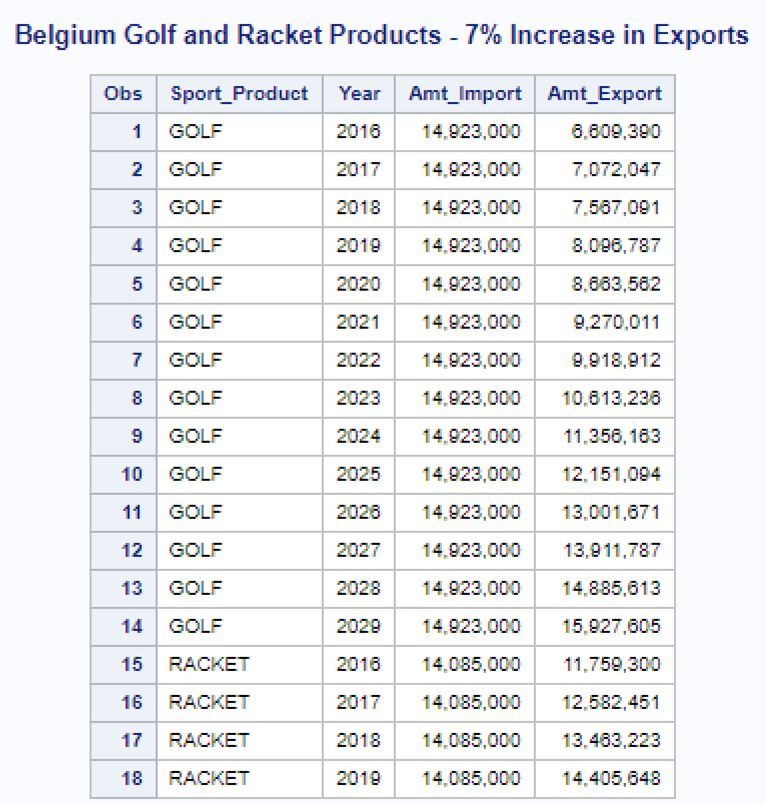
```
Are the new values of  Year the same as the final Year values before deleting this statement? Why or
why not?
*/
***************************************************;
*  LESSON 6, PRACTICE 5                           *;
***************************************************;

*Practice2;
data IncrExports;
   set pg2.eu_sports;
   where Year=2015 and Country='Belgium'
        and Sport_Product in ('GOLF','RACKET');

           do Year=2016 to 2025 while(Amt_Export<=Amt_Import);
              Amt_Export=Amt_Export*1.07;
              *output;
           end;
   format Amt_Import Amt_Export comma12.;
run;

title 'Belgium Golf and Racket Products - 7% Increase in Exports';
proc print data=IncrExports;
   var Sport_Product Year Amt_Import Amt_Export;
run;
title;

*Answer to practice1;
data IncrExports;
   set pg2.eu_sports;
   where Year=2015 and Country='Belgium'
       and Sport_Product in ('GOLF','RACKET');
   do while (Amt_Export<=Amt_Import);
     Year+1;
     Amt_Export=Amt_Export*1.07;
     output;
   end;
   format Amt_Import Amt_Export comma12.;
run;

title 'Belgium Golf and Racket Products - 7% Increase in Exports';
proc print data=IncrExports;
   var Sport_Product Year Amt_Import Amt_Export;
run;
title;

*Practice1;
data IncrExports;
   set pg2.eu_sports;
   where Year=2015 and Country='Belgium'
       and Sport_Product in ('GOLF','RACKET');
   Year=0;
          do while(Amt_Export<=Amt_Import);
             Amt_Export=Amt_Export*1.07;
             Year+1;
             output;
          end;
   format Amt_Import Amt_Export comma12.;
run;
```

```
*Original;

data IncrExports;

   set pg2.eu_sports;

   where Year=2015 and Country='Belgium'

        and Sport_Product in ('GOLF','RACKET');


   Amt_Export=Amt_Export*1.07;


   format Amt_Import Amt_Export comma12.;

run;
```



Belgium Golf and Racket Products - 7% Increase in Exports

Obs	Sport_Product	Year	Amt_Import	Amt_Export
1	GOLF	2026	14,923,000	12,151,094
2	RACKET	2020	14,085,000	14,405,648



Belgium Golf and Racket Products - 7% Increase in Exports

Obs	Sport_Product	Year	Amt_Import	Amt_Export
1	GOLF	2016	14,923,000	6,609,390
2	GOLF	2017	14,923,000	7,072,047
3	GOLF	2018	14,923,000	7,567,091
4	GOLF	2019	14,923,000	8,096,787
5	GOLF	2020	14,923,000	8,663,562
6	GOLF	2021	14,923,000	9,270,011
7	GOLF	2022	14,923,000	9,918,912
8	GOLF	2023	14,923,000	10,613,236
9	GOLF	2024	14,923,000	11,356,163
10	GOLF	2025	14,923,000	12,151,094
11	GOLF	2026	14,923,000	13,001,671
12	GOLF	2027	14,923,000	13,911,787
13	GOLF	2028	14,923,000	14,885,613
14	GOLF	2029	14,923,000	15,927,605
15	RACKET	2016	14,085,000	11,759,300
16	RACKET	2017	14,085,000	12,582,451
17	RACKET	2018	14,085,000	13,463,223
18	RACKET	2019	14,085,000	14,405,648

```
**********************************************************;
*  p207a02.sas Activity 7.02                              *;
*  1) Examine the DATA step code and run the program.     *;
*     Uncomment the RETAIN statement and run the program  *;
*     again. Why is the RETAIN statement necessary?       *;
*  2) Add a subsetting IF statement to include only the   *;
*     last row per student in the output table. Run the   *;
*     program.                                            *;
*  3) What must be true of the input table for the DATA   *;
*     step to work?                                       *;
**********************************************************;

*Practice2;
data class_wide;
	set pg2.class_test_narrow;
	by Name;
	retain Name Math Reading;
	keep Name Math Reading;
	if TestSubject="Math" then Math=TestScore;
	else if TestSubject="Reading" then Reading=TestScore;
		if last.name=1 then output;
run;

*Practice1;
data class_wide;
	set pg2.class_test_narrow;
	by Name;
	retain Name Math Reading;
	keep Name Math Reading;
```

```
        if TestSubject="Math" then Math=TestScore;
        else if TestSubject="Reading" then Reading=TestScore;
run;
```

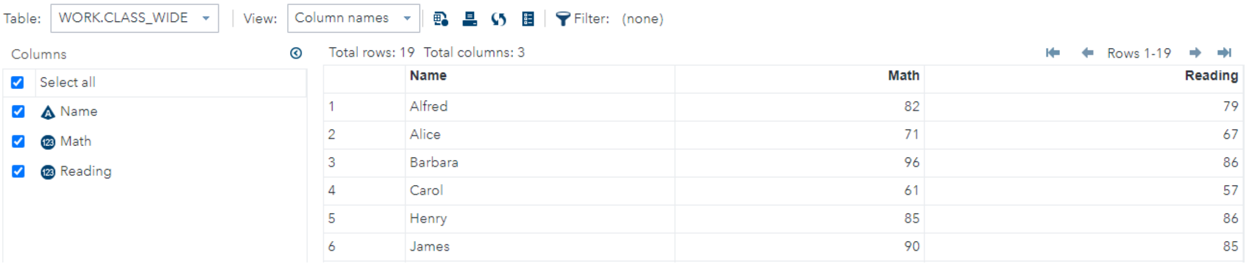


Table: WORK.CLASS_WIDE View: Column names Filter: (none)

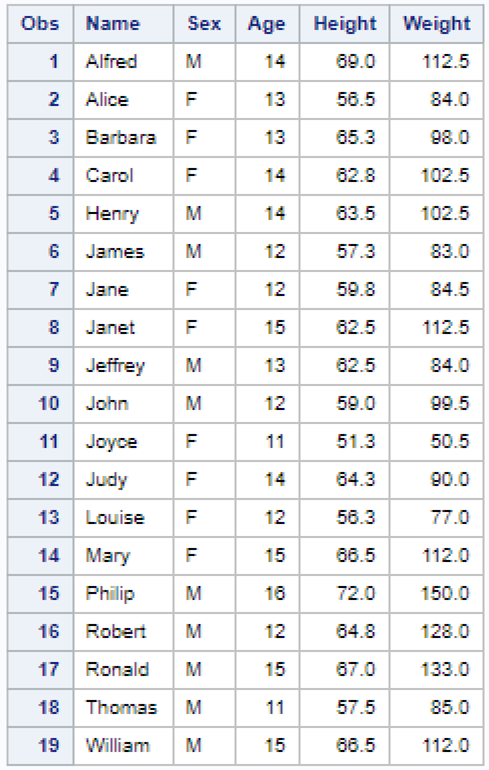
Columns

☑ Select all
☑ Name
☑ Math
☑ Reading

Total rows: 19 Total columns: 3 Rows 1-19

	Name	Math	Reading
1	Alfred	82	79
2	Alice	71	67
3	Barbara	96	86
4	Carol	61	57
5	Henry	85	86
6	James	90	85

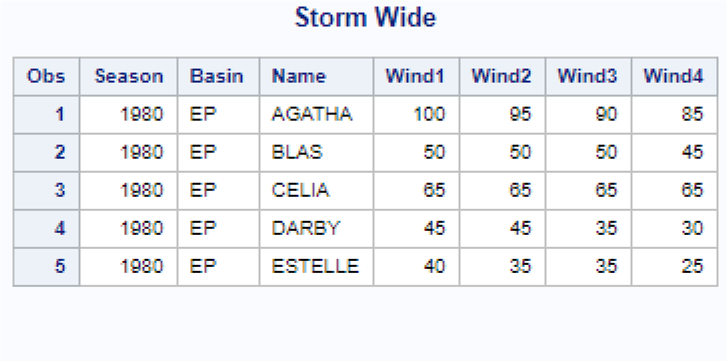
```
*****************************************************************;
*  p207a03.sas Activity 7.03                                    *;
*  1) Highlight the PROC PRINT step and run the selection.*;
*     Note how many rows are in the SASHELP.CLASS table.  *;
*  2) Highlight the PROC TRANSPOSE step and run the       *;
*     selection. Answer the following questions:          *;
*     Which columns from the input table are transposed   *;
*     into rows?                                          *;
*     What does each column in the output table represent?*;
*     What is the name of the output table?               *;
*****************************************************************;

proc print data=sashelp.class;
run;

proc transpose data=sashelp.class;
run;

*****************************************************************;
*  Activity 7.04                                          *;
*  1) Add the OUT= option on the PROC TRANSPOSE statement *;
*    (in the program above) to create an output table    *;
*    named CLASS_T.                                      *;
*  2) Add the following ID statement and run the step.    *;
*    What changes in the results?                        *;
*      id Name;                                          *;
*  3) Add the following VAR statement and run the step.   *;
*    What changes in the results?                        *;
*      var Height Weight;                                *;
****************************************************************;

proc transpose data=sashelp.class out=class_t;
        id Name;
        *var Height Weight;
run;

proc transpose data=sashelp.class out=class_t;
        id Name;
        var Height Weight;
run;
```



Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84.0
10	John	M	12	59.0	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90.0
13	Louise	F	12	56.3	77.0
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0
18	Thomas	M	11	57.5	85.0
19	William	M	15	66.5	112.0

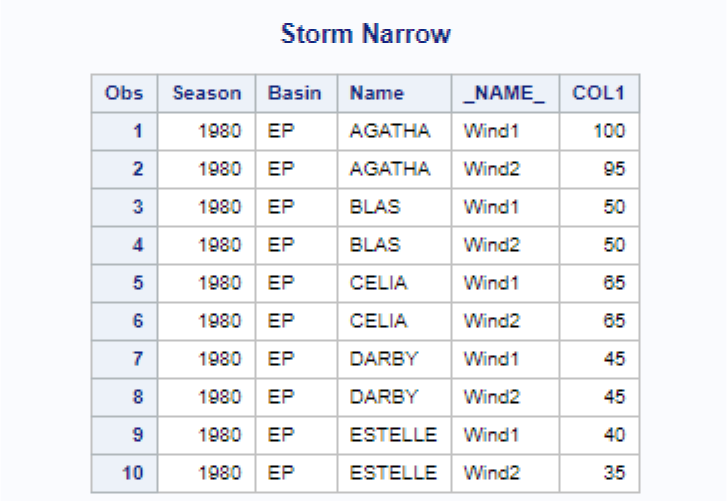
```
*****************************************************;
*  p207a05.sas Activity 7.05                          *;
*  1) Run the program. Notice that, by default, PROC     *;
*     TRANSPOSE transposes all the numeric columns,      *;
*     Wind1-Wind4.                                  *;
*  2) Add a VAR statement in PROC TRANSPOSE to transpose  *;
*     only the Wind1 and Wind2 columns. Run the program.  *;
*  3) What are the names of the columns that contain the  *;
*     column names and values that have been transposed?  *;
*****************************************************;


title "Storm Wide";

proc print data=pg2.storm_top4_wide(obs=5);

run;
```

```
proc transpose data=pg2.storm_top4_wide out=storm_top4_narrow;
        by Season Basin Name;
        *Add a VAR statement;
        var Wind1 Wind2;
run;


title "Storm Narrow";
proc print data=storm_top4_narrow(obs=10);
run;
title;
```



Storm Wide

Obs	Season	Basin	Name	Wind1	Wind2	Wind3	Wind4
1	1980	EP	AGATHA	100	95	90	85
2	1980	EP	BLAS	50	50	50	45
3	1980	EP	CELIA	65	65	65	65
4	1980	EP	DARBY	45	45	35	30
5	1980	EP	ESTELLE	40	35	35	25



Storm Narrow

Obs	Season	Basin	Name	_NAME_	COL1
1	1980	EP	AGATHA	Wind1	100
2	1980	EP	AGATHA	Wind2	95
3	1980	EP	BLAS	Wind1	50
4	1980	EP	BLAS	Wind2	50
5	1980	EP	CELIA	Wind1	65
6	1980	EP	CELIA	Wind2	65
7	1980	EP	DARBY	Wind1	45
8	1980	EP	DARBY	Wind2	45
9	1980	EP	ESTELLE	Wind1	40
10	1980	EP	ESTELLE	Wind2	35

```
*****************************************************************;
*  p207d01.sas Creating a Stacked Table with the DATA Step          *;
*****************************************************************;
```

```
*********************************************************;
*  Demo                                  *;
*     View steps in the course notes to use the DATA step    *;
*     debugger.                                *;
*********************************************************;


data class_test_narrow;
        set pg2.class_test_wide;
        keep Name Subject Score;
        length Subject $ 7;
        Subject="Math";
        Score=Math;
        output;
        Subject="Reading";
        Score=Reading;
        output;
run;
```

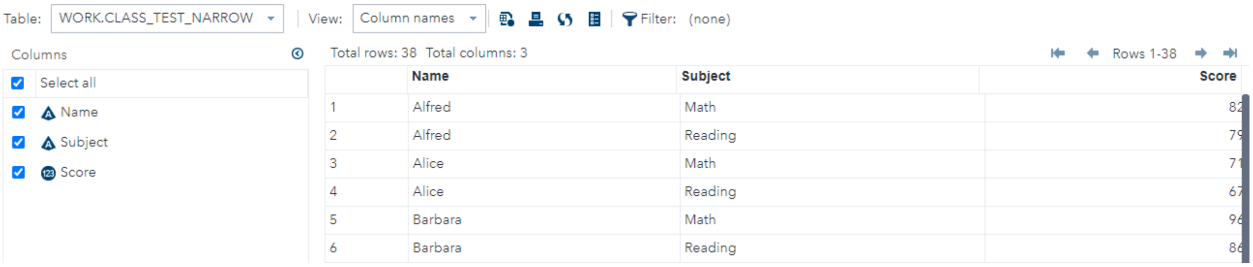


Table: WORK.CLASS_TEST_NARROW View: Column names Filter: (none)

Columns

☑ Select all

☑ Name

☑ Subject

☑ Score

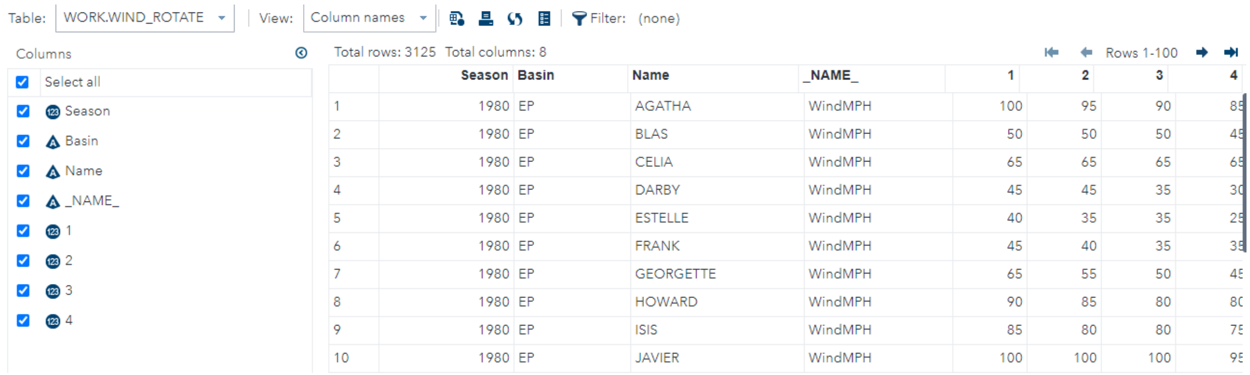
Total rows: 38 Total columns: 3

Rows 1-38

	Name	Subject	Score
1	Alfred	Math	82
2	Alfred	Reading	79
3	Alice	Math	71
4	Alice	Reading	67
5	Barbara	Math	96
6	Barbara	Reading	86

```
*****************************************************;
*  p207d02.sas Creating a Split Table with PROC TRANSPOSE           *;
*****************************************************;
*  Syntax                              *;
*                                   *;
*    PROC TRANSPOSE DATA=input-table OUT=output-table     *;
*                <PREFIX=column> <NAME=column>;       *;
*       <VAR columns(s)>;                          *;
*       <ID column>;                               *;
*       <BY column(s)>;                            *;
*    RUN;                                          *;
**************************************************************;


**************************************************************;
*  Demo                                            *;
*  1) Run the PROC TRANSPOSE step and examine the error   *;
*     in the log. The step fails because the values of ID *;
*     are not unique.                              *;
*  2) Add a BY statement to transpose the values within   *;
*     the groups of Season, Basin, and Name. Run the      *;
*     program.                                     *;
*  3) Notice that the unique values of WindRank (1, 2, 3, *;
*     and 4) are assigned as the column names for the     *;
*     transposed values of WindMPH.                *;
*  4) To give the transposed columns standard names, add  *;
*     the PREFIX=Wind option in the PROC TRANSPOSE        *;
*     statement. To rename the _name_ column that         *;
*     identifies the source column for the transposed     *;
*     values, add the NAME=WindSource option as well. Run *;
*     the step.                                    *;
*  5) Delete the NAME= option and add the DROP= data set  *;
*     option on the output table to drop the _name_       *;
*     column. Run the step.                        *;
**************************************************************;


*Practice3: Delete name= option and add drop=;
```

```
proc transpose data=pg2.storm_top4_narrow out=wind_rotate(drop=_Name_) prefix=Wind;

          var WindMPH;

          id WindRank;

          by Season Basin Name;

run;


*Practice2: add prefix and name;

proc transpose data=pg2.storm_top4_narrow out=wind_rotate prefix=Wind name=WindSource;

          var WindMPH;

          id WindRank;

          by Season Basin Name;

run;


*Practice1: adding by statement;

proc transpose data=pg2.storm_top4_narrow out=wind_rotate;

          var WindMPH;

          id WindRank;

          by Season Basin Name;

run;
```

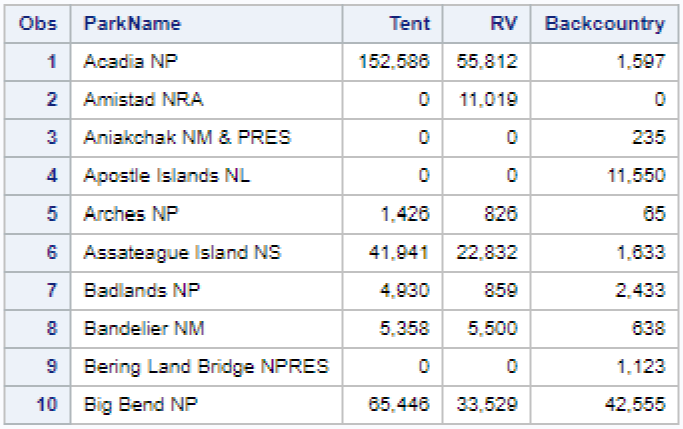


Total rows: 3125 Total columns: 8

	Season	Basin	Name	_NAME_	1	2	3	4
1	1980	EP	AGATHA	WindMPH	100	95	90	85
2	1980	EP	BLAS	WindMPH	50	50	50	45
3	1980	EP	CELIA	WindMPH	65	65	65	65
4	1980	EP	DARBY	WindMPH	45	45	35	30
5	1980	EP	ESTELLE	WindMPH	40	35	35	25
6	1980	EP	FRANK	WindMPH	45	40	35	35
7	1980	EP	GEORGETTE	WindMPH	65	55	50	45
8	1980	EP	HOWARD	WindMPH	90	85	80	80
9	1980	EP	ISIS	WindMPH	85	80	80	75
10	1980	EP	JAVIER	WindMPH	100	100	100	95

```
***********************************************************;

*  p207p01.sas LESSON 7, PRACTICE 1                       *;

*  a) Highlight the PROC PRINT step and run the selected  *;
*    code. Note that the Tent, RV, and Backcountry       *;
*    columns contain visitor counts.                   *;
*  b) To convert this wide table to a narrow table, the   *;
*    DATA step must create a new column named CampType   *;
*    with the values Tent, RV, and Backcountry, and      *;
*    another new column named CampCount with the numeric *;
*    counts. The DATA step includes statements to output *;
*    a row for CampType='Tent'. Modify the DATA step to  *;
*    output additional rows for RV and Backcountry.      *;
*  c) Add a LENGTH statement to ensure that the values of *;
*    the CampType column are not truncated.            *;
*  d) Run the DATA step. Confirm that each ParkName value *;
*    has three rows corresponding to the Tent, RV, and   *;
*    Backcountry visitor counts.                       *;
***********************************************************;

proc print data=pg2.np_2017camping(obs=10);
run;

data work.camping_narrow(drop=Tent RV Backcountry);
	set pg2.np_2017Camping;
	length CampType $ 11;
	format CampCount comma12.;
	CampType='Tent';
	CampCount=Tent;
	output;
	*Add statements to output rows for RV and Backcountry;
	CampType='RV';
	CampCount=RV;
```

```
        output;
        CampType='Backcountry';
        CampCount=Backcountry;
        output;
run;
```



Obs	ParkName	Tent	RV	Backcountry
1	Acadia NP	152,586	55,812	1,597
2	Amistad NRA	0	11,019	0
3	Aniakchak NM & PRES	0	0	235
4	Apostle Islands NL	0	0	11,550
5	Arches NP	1,426	826	65
6	Assateague Island NS	41,941	22,832	1,633
7	Badlands NP	4,930	859	2,433
8	Bandelier NM	5,358	5,500	638
9	Bering Land Bridge NPRES	0	0	1,123
10	Big Bend NP	65,446	33,529	42,555

/* p207p04.sas Level 1 Practice: Restructuring a Table Using PROC TRANSPOSE: Wide to Narrow

TOTAL POINTS 3

1.

Question 1

The pg2.np_2017camping table contains public use statistics for camping in 2017 from the National Park Service.

Convert the data from a wide table to a narrow table. If necessary, start SAS Studio before you begin.

Reminder: If you restarted your SAS session, submit your libname.sas program to access the practice data.

Open the p207p04.sas program in the practices folder.

Submit the PROC PRINT step to display the first five rows of pg2.np_2017camping.

Notice that the table contains three columns (Tent, RV, and Backcountry) with visitor counts for each value of ParkName.

In addition, notice that the table is sorted by ParkName.

In the PROC TRANSPOSE step, add the OUT= option to create a table named work.camping2017_t.

Add a BY statement to group the data by ParkName. This creates one row in the output table for each unique value of ParkName.

Add a VAR statement to transpose the Tent and RV columns.

Submit the PROC TRANSPOSE step and examine the output data.

How many rows are in the camping2017_t table?

What are the column names in the output table?

Modify the program and use the NAME= option to specify Location as the name for the column that

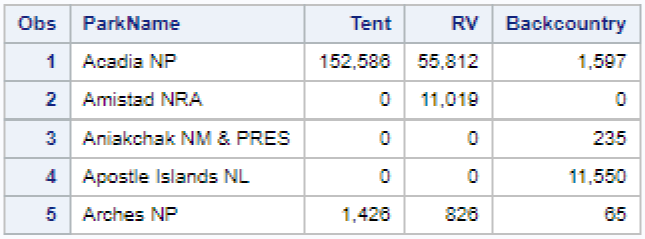
contains the names of the columns from the input table.

Use the RENAME= data set option after the output table to rename COL1 as Count.

Submit the PROC TRANSPOSE step and verify the results.

What are the column names in the camping2017_t table?

```
*/
***********************************************;
*  LESSON 7, PRACTICE 4                       *;
***********************************************;


proc print data=pg2.np_2017camping(obs=5);
run;


proc transpose data=pg2.np_2017camping out=work.camping2017_t(rename=(COL1=Count))
name=Location;
        by ParkName;
        var Tent RV;
run;
```



Obs	ParkName	Tent	RV	Backcountry
1	Acadia NP	152,586	55,812	1,597
2	Amistad NRA	0	11,019	0
3	Aniakchak NM & PRES	0	0	235
4	Apostle Islands NL	0	0	11,550
5	Arches NP	1,426	826	65

/*p207p05.sas Level 2 Practice: Restructuring a Table Using PROC TRANSPOSE: Narrow to Wide

TOTAL POINTS 2

1.

Question 1

The pg2.np_2016camping table contains public use statistics for camping in 2016 from the National Park Service.

Convert the data from a narrow to a wide table. If necessary, start SAS Studio before you begin.

Reminder: If you restarted your SAS session, submit your libname.sas program to access the practice data.

Examine the np_2016camping table. Notice that the table contains one row for each location type (Tent, RV, and Backcountry) by ParkName.

In addition, notice that the table is sorted alphabetically by ParkName.

Open a new program window and write a PROC TRANSPOSE step to create a wide table named work.camping2016_t.

Include only the ParkName column and individual columns for the values of CampType.

Submit the program and examine the output data.

How many rows are in the camping2016_t table?

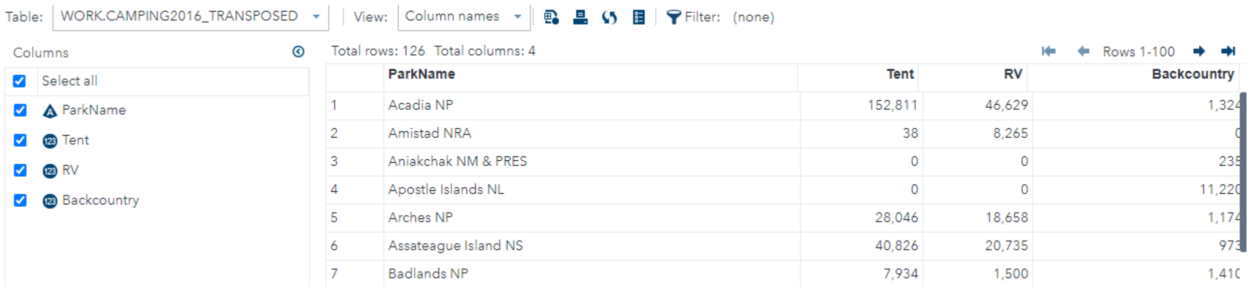
Question 2

How many columns are in the camping2016_t table?

*/

```
proc transpose data=pg2.np_2016camping

            out=work.camping2016_transposed(drop=_name_);

   by ParkName;

   id CampType;

   var CampCount;

run;


proc transpose data=pg2.np_2016camping out=work.camping2016_t;

            keep ParkName Tent RV Backcountry;

            by ParkName;
```

```
run;
```



Total rows: 126 Total columns: 4

	ParkName	Tent	RV	Backcountry
1	Acadia NP	152,811	46,629	1,324
2	Amistad NRA	38	8,265	0
3	Aniakchak NM & PRES	0	0	235
4	Apostle Islands NL	0	0	11,220
5	Arches NP	28,046	18,658	1,174
6	Assateague Island NS	40,826	20,735	973
7	Badlands NP	7,934	1,500	1,410