

Data Cleanup



This Data Could Use Some Cleaning Up

Name	Division	Years	Sales	Expense	State	Birth_ Ind	Est_ Age	Birth_Dt	Net
CHRIS	H	2	233.11	94.12	WI	0	.	03FEB1962	138.99
MARK	H	5	298.12	52.65	WI	E	35	.	245.47
SARAH	S	6	301.21	65.17	MN	1	.	03MAY1948	236.04
PAT	S	8	486.42	.	MN	1	.	15JUN1956	.
JOHN	H	11	.	123.96	MN	0	.	14NOV1954	.
MARY	S	14	5691.78	2452.11	WI	E	27	.	3239.67
SARAH	S	6	301.21	65.17	MN	1	.	03MAY1948	236.04



Issues to consider:

- Date Checks
- Removal of Duplicate Records
- Data Look ups
- Check For Invalid Data in Calculations
- Type Conversions For Data Manipulation
- Justification and Embedded Spaces



Why is data clean-up necessary?

- Remove invalid data.
- Remove duplicate records.
- Avoid data errors due to inconsistent data types.



What should I look for during the data clean-up process?

- Variables with multiple meanings depending on the value received.
- Invalid data values.
- Missing data values.
- Values outside of a predetermined range.
- Duplicate data.

Date Checks



- Are dates within expected range?
- Are valid dates received?

The birth date field contains either the customer's birth date or an estimated age if the exact date is unknown.

```
CHRIS 03FEB1962
MARK  22OCT1975
SARAH E -35
PAT   15JUN1956
JOHN  E -27
```

Date Checks



Reading the birth date field as a date results in a data error.

```
Data Birthdt_Check;
  Infile Custdata;
  Input @01 NAME      $5.
         @07 BIRTH_DT Date9.;
Run;
```

Partial Log:

```
NOTE: Invalid data for BIRTH_DT in line 7 7-15.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+
7          SARAH E-35
NAME=SARAH BIRTH_DT=. _ERROR_=1 _N_=3
NOTE: Invalid data for BIRTH_DT in line 9 7-15.
9          JOHN  E-27
NAME=JOHN  BIRTH_DT=. _ERROR_=1 _N_=5
```

Date Checks



To cleanly read the birth date field the variable must be read twice and the value stored in 1 of 2 possible variables.

```
Data Birthdt_Check;
  Infile Custdata;
  Input @01 NAME      $5.
         @07 BIRTHIND $1. @;
  If Birthind = 'E' then
    Input @09 EST_AGE 2.;
  Else
    Input @07 BIRTH_DT Date9.;
Run;
```

Obs	NAME	BIRTHIND	EST_AGE	BIRTH_DT
1	CHRIS	0	.	03FEB1962
2	MARK	2	.	22OCT1975
3	SARAH	E	35	.
4	PAT	1	.	15JUN1956
5	JOHN	E	27	.

Duplicate Records



How should duplicate records be handled?

- Remove all completely duplicated observations if desired.
- Keep one record per key value.
- Keep the record with the maximum or minimum value.
- Compare to historical records and keep oldest or most recent.

Duplicate Records



Methods to remove duplicate records.

- SORT procedure
- FREQ procedure
- SQL procedure
- DATA step FIRST. and LAST. logic

Remove Duplicate Records – SORT Procedure



- Use the SORT procedure to remove identical records or records with duplicate keys.

How can we remove identical duplicated records?

Name	Division	Years	Sales	Expense	State
CHRIS	H	2	233.11	94.12	WI
MARK	H	5	298.12	52.65	WI
SARAH	S	6	301.21	65.17	MN
MARY	S	14	5691.78	2452.11	WI
SARAH	S	6	301.21	65.17	MN

Remove Duplicate Records – SORT Procedure



```
Proc Sort Data=Softsale NODUP;  
  BY name;  
Run;
```

```
Proc Print Data=Softsale;  
Run;
```

Resulting Output:

Obs	Name	Division	Years	Sales	Expense	State
1	CHRIS	H	2	233.11	94.12	WI
2	MARK	H	5	298.12	52.65	WI
3	MARY	S	14	5691.78	2452.11	WI
4	SARAH	S	6	301.21	65.17	MN

Note:

Only consecutive duplicates are deleted.

Remove Duplicate Records – SORT Procedure



What states are serviced by our sales people?

- Keep 1 record per state.
- Remove duplicate *keyed* records.

Employee Data:

ANDREW	MN
BENJAMIN	IL
BETH	WI
CHRIS	WI
JANET	IA
JENNIFER	IL
JOHN	IA
JOY	MI
MARK	WI

Remove Duplicate Records – SORT Procedure



```
Proc Sort Data=Employee
      Out=Sales_St NODUPKEY ;
      BY State;
Run;
```

```
Proc Print Data=Sales_St;
      Title 'STATES REPRESENTED';
Run;
```

Resulting Output:

STATES REPRESENTED		
Obs	Name	State
1	JANET	IA
2	BENJAMIN	IL
3	JOY	MI
4	ANDREW	MN
5	BETH	WI

Remove Duplicate Records – SORT Procedure



DUPOUT= Option

This option allows you to specify a data set which will capture duplicate observations detected by the Proc Sort procedure.

```
Proc Sort Data      = Softsale Nodup  
          Out       = Unique_Softsale  
          DUPOUT = Dup_Softsale;  
          BY Name;  
Run;
```

Note: New option in SAS v9

Remove Duplicate Records – Frequency Procedure



The Frequency procedure counts observations. We can use it to find all observations with multiple occurrences.

Remove all records if account appears more than once.

CHRIS	233.11	94.12	345234545
MARK	298.12	52.65	565634514
SARAH	301.21	65.17	455651144
PAT	4009.21	322.12	754332244
JOHN	678.43	150.11	664913568
WILLIAM	3231.75	644.55	658695545
ANDREW	1762.11	476.13	886868444
BENJAMIN	201.11	25.21	273644455
JANET	98.11	125.32	966588444
STEVE	6153.32	1507.12	904545234
JENNIFER	542.11	134.24	965842354
PAT	4009.21	322.12	754332244
JOY	2442.22	761.98	234543545
MARY	5691.78	2452.11	523452345
TOM	5669.12	798.15	544445234
BETH	4822.12	982.10	995586666

Remove Duplicate Records – Frequency Procedure



```
Proc Freq Data=Cust_Acct Noprint;  
    Table Acct_Num / Out=Acct_Freq;  
Run;
```

```
Proc Sort Data=Cust_Acct;  
    By Acct_Num;  
Run;
```

```
Data Customers;  
    Merge Cust_Acct  
          Acct_Freq;  
    By Acct_Num;  
    If Count = 1;  
Run;
```

```
Proc Print Data=Customers;  
    Title 'CUSTOMERS WITH ONLY A SINGLE ACCOUNT RECORD';  
Run;
```

Remove Duplicate Records – Frequency Procedure



```
Proc Freq Data=Cust_Acct Noprint;  
  Table Acct_Num / Out=Acct_Freq (Where=(Count=1)) ;  
Run;
```

```
Proc Sort Data=Cust_Acct;  
  By Acct_Num;  
Run;
```

```
Data Customers;  
  Merge Cust_Acct  
        Acct_Freq (In=Onaf) ;  
  By Acct_Num;  
  If Onaf;  
Run;
```

```
Proc Print Data=Customers;  
  Title 'CUSTOMERS WITH ONLY A SINGLE ACCOUNT RECORD';  
Run;
```

Remove Duplicate Records – Frequency Procedure



Resulting Output:

CUSTOMERS WITH ONLY A SINGLE ACCOUNT RECORD						
Obs	NAME	SALES	EXPENSE	ACCT_NUM	COUNT	PERCENT
1	JOY	2442.22	761.98	234543545	1	6.25
2	BENJAMIN	201.11	25.21	273644455	1	6.25
3	CHRIS	233.11	94.12	345234545	1	6.25
4	SARAH	301.21	65.17	455651144	1	6.25
5	MARY	5691.78	2452.11	523452345	1	6.25
6	TOM	5669.12	798.15	544445234	1	6.25
7	MARK	298.12	52.65	565634514	1	6.25
8	WILLIAM	3231.75	644.55	658695545	1	6.25
9	JOHN	678.43	150.11	664913568	1	6.25
10	ANDREW	1762.11	476.13	886868444	1	6.25
11	STEVE	6153.32	1507.12	904545234	1	6.25
12	JENNIFER	542.11	134.24	965842354	1	6.25
13	JANET	98.11	125.32	966588444	1	6.25
14	BETH	4822.12	982.10	995586666	1	6.25

Remove Duplicate Records – SQL Procedure



Keep one record per key value:

- Similar to SORT procedure with NODUP option.
- Duplicate records are not required to be consecutive duplicates.

Input Data:

Name	Division	Years	Sales	Expense	State
CHRIS	H	2	233.11	94.12	WI
MARK	H	5	298.12	52.65	WI
SARAH	S	6	301.21	65.17	MN
MARY	S	14	5691.78	2452.11	WI
SARAH	S	6	301.21	65.17	MN

Remove Duplicate Records – SQL Procedure



```
Proc Sql;  
  Create Table Softsale_Dup As  
  Select DISTINCT *  
  From Softsale;  
Quit;  
  
Proc Print Data=Softsale_Dup;  
Run;
```

Resulting output:

Obs	Name	Division	Years	Sales	Expense	State
1	CHRIS	H	2	233.11	94.12	WI
2	MARK	H	5	298.12	52.65	WI
3	MARY	S	14	5691.78	2452.11	WI
4	SARAH	S	6	301.21	65.17	MN

Remove Duplicate Records – SQL Procedure



Keep record with maximum or minimum value:

HAVING clause is used to determine the maximum or minimum value.

Remove duplicate name records and keep record with greatest sales amount.

Name	Division	Years	Sales	Expense	State
CHRIS	H	2	233.11	94.12	WI
MARK	H	5	298.12	52.65	WI
SARAH	S	6	301.21	65.17	MN
MARY	S	14	5691.78	2452.11	WI
SARAH	S	6	458.62	98.54	MN

Remove Duplicate Records – SQL Procedure



```
Proc Sql;  
  Create Table Softsale_Max As  
  Select DISTINCT *  
  From Softsale  
  GROUP BY Name  
  HAVING Max(Sales) =Sales;  
Quit;  
Proc Print Data=Softsale_Max;  
Run;
```

Resulting Output:

Obs	Name	Division	Years	Sales	Expense	State
1	CHRIS	H	2	233.11	94.12	WI
2	MARK	H	5	298.12	52.65	WI
3	MARY	S	14	5691.78	2452.11	WI
4	SARAH	S	6	458.62	98.54	MN

Remove Duplicate Records – SQL Procedure



Remove duplicate name records and keep the record with the lowest expense amount.

```
Proc Sql;  
  Create Table Softsale_Min As  
  Select DISTINCT *  
  From Softsale  
  GROUP BY Name  
  HAVING Min(Expense) =Expense;  
Quit;  
Proc Print Data=Softsale_Min;  
Run;
```

Resulting Output:

Obs	Name	Division	Years	Sales	Expense	State
1	CHRIS	H	2	233.11	94.12	WI
2	MARK	H	5	298.12	52.65	WI
3	MARY	S	14	5691.78	2452.11	WI
4	SARAH	S	6	301.21	65.17	MN

Remove Duplicate Records – FIRST. and LAST.



Scenarios where FIRST. and LAST. logic may be used.

- Remove all duplicate records
- Keep first occurrence of the key value
- Keep the last occurrence of the key value
- Keep record with the maximum of a non-key value
- Keep record with the minimum of a non-key value
- Keep the most recent
- Keep the oldest

Input Data:

Name	Division	Years	Sales	Expense	State
CHRIS	H	2	233.11	94.12	WI
MARK	H	5	298.12	52.65	WI
SARAH	S	6	301.21	65.17	MN
MARY	S	14	5691.78	2452.11	WI
SARAH	S	6	458.62	98.54	MN

Remove Duplicate Records – FIRST. and LAST.



Keep ONLY those records with a SINGLE observation:

```
Proc Sort Data=Softsale;  
  BY Name;  
Run;  
  
Data Softsale_Nodups;  
  Set Softsale;  
  BY Name;  
  If FIRST.Name and LAST.Name;  
Run;
```

Resulting Output:

Obs	Name	Division	Years	Sales	Expense	State
1	CHRIS	H	2	233.11	94.12	WI
2	MARK	H	5	298.12	52.65	WI
3	MARY	S	14	5691.78	2452.11	WI

Remove Duplicate Records – FIRST. and LAST.



Keep the first occurrence of the key value:

```
Proc Sort Data=Softsale;  
  BY Name;  
Run;
```

```
Data Softsale_Nodups;  
  Set Softsale;  
  BY Name;  
  If FIRST.Name;  
Run;
```

Resulting Output:

Obs	Name	Division	Years	Sales	Expense	State
1	CHRIS	H	2	233.11	94.12	WI
2	MARK	H	5	298.12	52.65	WI
3	MARY	S	14	5691.78	2452.11	WI
4	SARAH	S	6	301.21	65.17	MN

Remove Duplicate Records – FIRST. and LAST.



Keep the record with the maximum of a non-key value:

```
Proc Sort Data=Softsale;  
  BY Name DESCENDING Sales;  
Run;  
Data Softsale_Nodups;  
  Set Softsale;  
  BY Name;  
  If FIRST.Name;  
Run;
```

Resulting Output:

Obs	Name	Division	Years	Sales	Expense	State
1	CHRIS	H	2	233.11	94.12	WI
2	MARK	H	5	298.12	52.65	WI
3	MARY	S	14	5691.78	2452.11	WI
4	SARAH	S	6	458.62	98.54	MN



Why use data look-ups?

- Create value groups
- Verify that the value received is within valid list of values

How can a data look-up be created?

- User defined informat
- User defined format

User-Defined Informats



Proc FORMAT's INVALUE statement edits and changes values as they are read.

Example: translate letter grades to numerics and validate departments.

```
Proc Format;  
  Invalue Gradfmt 'A'=4 'B'=3 'C'=2 'D'=1 'F'=0;  
  Invalue Deptfmt 101-888 = _same_  
                  other   = _error_;
```

```
Run;  
Data Graddata;  
  Infile Datalines;  
  Input @1 Grade Gradfmt1.  
        @3 Dept  EDeptfmt3. ;  
  Datalines;
```

```
A 707
```

```
B 999
```

```
;
```

```
Run;
```

```
Proc Print Data=Graddata;  
  Title 'Graddata';  
  Format Grade 3.1;
```

```
Run;
```

```
Data Clean-up
```

User-Defined Informats



The Resulting Output

Graddata

Obs	grade	dept
1	4.0	707
2	3.0	.

User-defined Format



User-defined format may be used as a look-up table to:

- Select records
- Create new variables

Create user-defined format

```
Proc Format;
```

```
  Value $Itemcat
```

```
    'PRINTERS', 'MONITORS', 'PCS'           = 'COMPUTER'
```

```
    'PAPER', 'PENS', 'CALC', 'PENCILS', 'STAPLES', 'CLIPS1',  
    'CLIPS2', 'CLIPS3', 'LPADS'           = 'SUPPLIES'
```

```
    'DESKS', 'LAMPS', 'CHAIR', 'CREDZA'    = 'FURNITURE';
```

```
Run;
```

Apply User-Defined Format to Select Records



Select only the items that are in the supplies category:

```
Data Supply_Data;  
  Set Item_Data;  
  
  If PUT(Item_Cd, $ITEMCAT.) = 'SUPPLIES';  
  
  ... additional SAS code  
Run;
```

Apply User-Defined Format to Create New Variables



Create a new variable that contains the item category value:

```
Data Supply_Data;  
  Set Item_Data;  
  
  Item_Cat = PUT(Item_Cd, $ITEMCAT.);  
  
  ... additional SAS code  
Run;
```

Check for Invalid Data in Calculations



Why should you check for invalid calculation data?

- Invalid data may cause unexpected results.
- Invalid data may cause data errors.
- Missing values will cause the result to be a missing value.

Invalid Data - Division by 0



Problems with dividing by 0:

- Division by 0 is an invalid calculation.
- Division by 0 will cause a data error.
- Division by 0 will result in a missing value.

```
Data Sales_Results;  
  Input Name $ Sales Expense;  
  Ratio = Sales / Expense;  
Datalines;  
BETH 1526.76 0  
;  
Run;
```

Invalid Data - Division by 0



Partial Log:

```
155 Data Sales_Results;
156     Input Name $ Sales Expense;
157     Ratio = Sales / Expense;
158 Datalines;
```

NOTE: Division by zero detected at line 157 column 17.

RULE: - - - - + - - - - 1 - - - - + - - - - 2 - - - - + - - - - 3 - - - - + - - - - 4 - - - - + - - - - 5

```
159     BETH 1526.76 0
```

Name=BETH Sales=1526.76 Expense=0 Ratio=. _ERROR_=1 _N_=1

NOTE: Mathematical operations could not be performed at the following places. The results of the operations have been set to missing values.

Each place is given by: (Number of times) at (Line):(Column).

1 at 157:17

Invalid Data - Division by 0



Modify program to bypass the calculation when expense is 0.

```
Data Sales_Results;  
    Input Name $ Sales Expense;  
    If Expense ne 0 then  
        Ratio = Sales / Expense;  
Datalines;  
BETH 1526.76 0  
;  
Run;
```

Invalid Data - Missing values



Missing values in a calculation will result in a missing value.

```
Data Sales_Results;  
    Input Name $ Sales Expense;  
    Ratio = Sales / Expense;  
Datalines;  
BETH . 260.57  
;  
run;
```

Invalid Data - Missing values



Partial Log:

```
162 Data Sales_results;  
163     Input Name $ Sales Expense;  
164     Ratio = Sales / Expense;  
165 Datalines;
```

NOTE: Missing values were generated as a result of performing an operation on missing values.

Each place is given by: (Number of times) at (Line):(Column).

1 at 164:17

NOTE: The data set WORK.SALES_RESULTS has 1 observations and 4 variables.

Partial Data Set:

Obs	Name	Sales	Expense	Ratio
1	BETH	.	260.57	.

Invalid Data - Missing values



Modify the code to bypass the calculation when the variables are missing.

```
Data sales_results;  
  Input Name $ Sales Expense;  
  If Sales = . or Expense = . Then  
    Ratio = 0;  
  Else  
    Ratio = Sales / Expense;  
Datalines;  
BETH . 260.57  
;  
Run;
```

Note: The result will still be missing since ratio will not be calculated but the messages will no longer appear in the log.

Type Conversions for Data Manipulation



Why should data type conversions occur?

- Prevents inefficient data conversion by SAS.
- Data defined as character is needed in a calculation.
- Data defined as numeric is to be combined with data defined as character.

Type Conversions for Data Manipulation



INPUT and PUT functions give you complete variable type conversion

- INPUT function passes a PDV value through a SAS informat.
- INPUT function is used primarily to convert character values to numeric.
- PUT function passes a PDV value through a SAS format.
- PUT function is used primarily to convert numeric to character.

Syntax

variable = **INPUT**(*argument*,*informat*);

variable = **PUT**(*argument*,*format*);

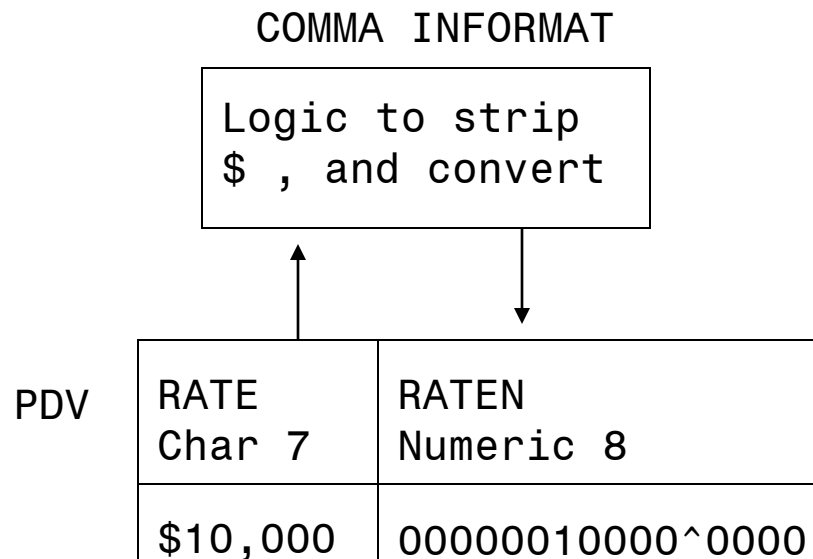
Character to Numeric Conversion



Convert a character field to a numeric field.

RATE is a 7 byte character field containing '\$10,000' with which we would like to use in computations.

```
RATEN=INPUT (RATE,COMMA7.);
```



Notes:

- The *INPUT statement* converts values from an INPUT BUFFER.
- The *INPUT function* converts values from the PDV.

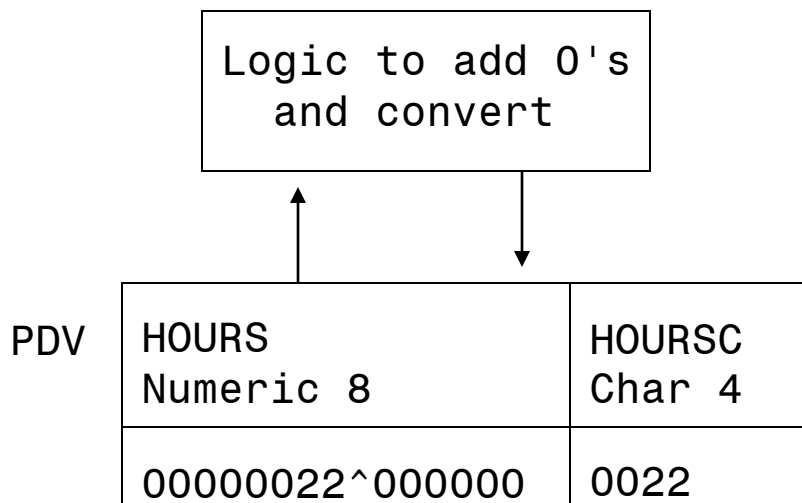
Numeric to Character Conversion



Convert a numeric field to a character field.

HOURS is a 8 byte numeric field containing 22. We need to convert to a four byte character field with leading zeros.

```
HOURSC=PUT (HOURS,Z4.);
```



Notes:

- The *PUT statement* converts values to an OUTPUT BUFFER.
- The *PUT function* converts values to the PDV.

Justification and Embedded Spaces



LEFT (<i>string</i>)	Left aligns value.
RIGHT (<i>string</i>)	Right aligns value.
TRIM (<i>string</i>)	Removes trailing spaces.
STRIP (<i>string</i>)	Removes leading and trailing spaces*.
COMPRESS (string, chars, optional modifiers)	Removes spaces or indicated characters.
COMPBL (<i>string</i>)	Consecutive spaces converted to single space.

Note:

*STRIP function is new in SAS v9.

Justification and Embedded Spaces Example



```
Data Test;
```

```
  Input Name $1-14 @16 Cityin $CHAR11. @26 State $CHAR6.  
    @29 Address $CHAR35. ;  
  Packname   = Compress(Name);  
  Leftc      = Left(Nityin);  
  Rightc     = Right(Nityin);  
  Cityst     = Trim(Nityin)!!!, '!!!State;  
  Stripname  = Strip(Name);  
  Address    = Compbl(Address);  
  Drop Name Cityin State;
```

```
Datalines;
```

```
  Chris Lee      Madison    WI    123 W    1st    St  
  Mark Jones    Milwaukee WI    449 N Ridge St  
  Mary Clark    Columbus  WI    331    Western St  
  ;  
Run;
```

```
Proc Print Data = Test;
```

```
  Var Packname Leftc Rightc Cityst Stripname Address;  
  Title 'Functions';
```

```
Run;
```

Justification and Embedded Spaces



Resulting Output:

Functions					
Packname	Leftc	Rightc	Cityst	Stripname	Address
ChrisLee	Madison	Madison	Madison, WI	Chris Lee	123 W 1st St
MarkJones	Milwaukee	Milwaukee	Milwaukee, WI	Mark Jones	449 N Ridge St
MaryClark	Columbus	Columbus	Columbus, WI	Mary Clark	331 Western St



SYSTEMS SEMINAR CONSULTANTS, INC.

SAS[®] Training, Consulting, & Help Desk Services
2997 Yarmouth Greenway Drive • Madison, WI 53711

(608) 278-9964 • Fax (608) 278-0065

www.sys-seminar.com



Steven First

President

sfirst@sys-seminar.com