# Preparing Data

Dr. Lendie Follett

2022-04-05



# Reading and Filtering Data



## Data Step

- We've used data steps before.
- If you recall, the goal of data step is to read, process, or create data.
- Specifically, we'll learn how to use the data step to
  - subset/filter rows
  - subset/filter columns
  - compute new columns
  - perform conditional processing
  - more, later!
- The simplest syntax for a data step is:

```
data output-table;
set input-table;
run;
```

- Replace `output-table` with the name of the new table you're creating.
- Replace `input-table` with the name of the table you're reading.



## Data step

We've seen the data step used, for example, with the storm summary data:

```
data ss;
set s40dat.storm_summary;
run;
```

- In this example we created the table `ss`, which lives in the work library, from the existing sas data set `storm_summary`, which lives in the sashelp library.
- `work.ss` is an exact copy of `sashelp.storm_summary`

# Filtering Rows in the data step

- We've filtered rows before using the `where` statement in proc steps.
- Importantly, the `where` statement syntax works within a data step!

```
data output-table;
    set input-table;
    where expression;
run;
```

- As before, replace `expression` with a logical expression like those practiced in SAS Topic 3.
- As a result, `output-table` will be an exact copy of *only the rows of* `input table` *for which* `expression` *holds true*.



# Subsetting Columns

- So, we can use a `where` statement to remove unwanted rows.
- What if we want to keep a subset of columns?
- Choose between the `drop` statement or the `keep` statement:

```
data output-data;
  set input-table;
  drop drop-col-name-list;
run;
```

- Replace `drop-col-name-list` with the name(s) of columns you want *dropped* in `output-data`. (All others will be kept.)

```
data output-data;
  set input-table;
  keep keep-col-name-list;
run;
```

- Replace `keep-col-name-list` with the name(s) of columns you want *kept* in `output-data`. (All others will be dropped.)

- Whether you use `keep` or `drop` is a matter of convenience; which is easier?



# Case Study

## Focusing on the strongest storms

Goal: We are interested in category 5 storms (Max Wind MPH $\geq$ 156) occurring in year 2000 or after, focusing only on Season,StartDate, EndDate, Basin, Name, Type and MaxWindMPH. We will likely use this data in the future, so make this data set permanent. How many category 5 storms have we had since 2000?

Save this script as t4_cs_filter.sas.



# Formatting Columns in the data step

- Recall how we used the `format` statement within proc steps to change how data values displayed.
- The same syntax works

```
data output-table;
    set input-table;
    format col-name1 format1 <col-name2 format2> ...;
run;
```

- Visually, the format statement in a data step does the same thing that it did within a proc step.
- However, a format statement in a data step **permanently** assigns a format to a column.
- For example,

```
data out.storm_cat5;
    set s40dat.storm_summary;
    where StartDate>="01jan2000"d and MaxWindMPH>=156;
    keep StartDate EndDate Season Basin Name Type MaxWindMPH;
    format StartDate mmddyy10.;
run;
```

# Creating New Columns



# Creating new columns

- Rarely will a given data set come with all the columns you need
- It's easy to create new columns using a data step

```
data output-table;
  set input-table;
  new-column = expression;
run;
```

- Replace `new-column` with the name of the new column you want to create.
- After the `=`, provide some sort of expression (mathematical, function evaluation, etc...)



# Creating new columns

Say we wanted to look at the profit a dealership would make if they sold foreign (non-USA) vehicles at MSRP.

```
data cars_new;
    set sashelp.cars;
    where Origin ne "USA";
     Profit = MSRP-Invoice;
     Source = "Non-US Cars";
    format Profit dollar10.;
    keep Make Model MSRP Invoice Profit Source;
run;
```

- We created a new (temporary, since it's in the work library...) data set called cars_new from the existing sashelp.cars.
- `Profit = MSRP-Invoice;` creates a new column called Profit (depends on values of MSRP, Invoice).
- `Source = "Non-US Cars"` creates a new column called Source (a constant character value).
- `where Origin ne "USA";` subsets the rows in cars_new to foreign vehicles only.
- `format Profit dollar10.;` puts a more meaningful format on the newly created column Profit.



# Creating new columns

Say we wanted to look at the profit a dealership would make if they sold foreign (non-USA) vehicles at MSRP.

```
data cars_new;
    set sashelp.cars;
    where Origin ne "USA";
     Profit = MSRP-Invoice;
     Source = "Non-US Cars";
    format Profit dollar10.;
    keep Make Model MSRP Invoice Profit Source;
run;
```

| Make | Model | MSRP | Invoice | Profit | Source |
|---|---|---|---|---|---|
| Acura | MDX | $36,945 | $33,337 | $3,608 | Non-US Cars |
| Acura | RSX Type S 2dr | $23,820 | $21,761 | $2,059 | Non-US Cars |
| Acura | TSX 4dr | $26,990 | $24,647 | $2,343 | Non-US Cars |
| Acura | TL 4dr | $33,195 | $30,299 | $2,896 | Non-US Cars |
| Acura | 3.5 RL 4dr | $43,755 | $39,014 | $4,741 | Non-US Cars |
| Acura | 3.5 RL w/Navigation 4dr | $46,100 | $41,100 | $5,000 | Non-US Cars |

# Your Turn

Continue with your script t4_cs_filter.sas that we started together earlier. Recall we created a data set out.storm_cat5, containing only recent (post-2000) category 5 storms.

1. Add a new data step to create a new column named StormLength in out.storm_cat5 that represents the number of days between StartDate and EndDate (i.e., calculate the duration of these storms).
2. On average, how long do these strong storms last? What was the longest? The shortest? Use a `proc means` step to find out.



# Creating columns with functions

- Creating new columns using arithmetic operators (`+ - * /` etc...) is fairly straightforward. However, often you'll need more powerful and flexible tools.
- SAS has hundreds of *functions* that can be used to manipulate both numeric (including date) and character values.
- We'll get into a few functions in Stat 40, but for a comprehensive list:

1. Go to support.sas.com/documentation. Click Programming: SAS 9.4 and Viya.
2. In the Syntax - Quick Links section, under Language Elements, select Functions
3. Alphabetical Listing might be useful.



# Creating columns with functions

- A *function* takes in inputs - called arguments - and outputs a value into your new column.

```
data output-table;
  set input-table;
  new-column=function-name(argument1 <, argument2, argument3,....>);
run;
```

- Replace `function-name` with the name of the function
- Replace argument1 with the first argument the function takes in, argument 2 (optional) with the second, etc...



# Creating columns with functions

For example, suppose we wanted to calculate the mean MPG of vehicles; the average of the city and highway MPG.

```
data cars_new;
     set sashelp.cars;
     MPG_Mean=mean(MPG_City, MPG_Highway);
     format MPG_Mean 4.1;
     keep Make Model MPG_City MPG_Highway MPG_Mean;
run;
```

| Obs | Make | Model | MPG_City | MPG_Highway | MPG_Mean |
|---|---|---|---|---|---|
| 1 | Acura | MDX | 17 | 23 | 20.0 |
| 2 | Acura | RSX Type S 2dr | 24 | 31 | 27.5 |
| 3 | Acura | TSX 4dr | 22 | 29 | 25.5 |
| 4 | Acura | TL 4dr | 20 | 28 | 24.0 |
| 5 | Acura | 3.5 RL 4dr | 18 | 24 | 21.0 |
| 6 | Acura | 3.5 RL w/Navigation 4dr | 18 | 24 | 21.0 |
| 7 | Acura | NSX coupe 2dr manual S | 17 | 24 | 20.5 |
| 8 | Audi | A4 1.8T 4dr | 22 | 31 | 26.5 |

- Note: Numeric functions (like the ones listed below) *ignore missing data*! That is, the outputted values are based only on the known values.

`sum, mean, median, range, min, max, n, nmiss, .....`

# Case Study

Let's explore the dataset storm_range, with a focus on summarizing wind characteristics. Notice that storm_summary has four wind speed measurements.

While were ultimately interested in all the basins (locations), we've been asked specifically to look into wind patterns occurring within the Pacific ocean. From the data, these can be distinguished by looking at basins with the *second letter equal to P*.

Start a new script; call this one t4_cs_functions.sas



# Creating new columns with functions

- We just saw an example of a function (`substr()`) being used on categorical variables.
- Here's a brief list of other functions that are useful when manipulating categorical variables:
  - `upcase(char)`, `lowcase(char)`: changes letters in a character string to uppercase or lowercase
  - `propcase(char, <delimiters>)`: Changes the first letter of each word to uppercase and other letters to lowercase
  - `cats(char1, char2, ...)`: Concatenates character strings and removes leading and trailing blanks from each argument
  - `substr(char, position, <length>)`: Returns a substring from a character string.



# Creating new columns with functions

- Often, you'll want to use special functions to manipulate dates
- Here's a brief list of functions you may find useful
  - `MONTH(SAS-date)`: Returns a number from 1 through 12 that represents the month
  - `YEAR(SAS-date)`: Returns the four-digit year
  - `DAY(SAS-date)`: Returns a number from 1 through 31 that represents the day of the month
  - `WEEKDAY(SAS-date)`: Returns a number from 1 through 7 that represents the day of the week (Sunday=1)
  - `QTR(SAS-date)`: Returns a number from 1 through 4 that represents the quarter
  - `YRDIF(startdate, enddate, 'AGE')`: Calculates a precise difference in years between two dates
  - `TODAY()`: Returns the current date as a numeric SAS date value (no arguments needed)



# Conditional Processing

# Conditional Processing

- Often in the DATA step, we need to process data conditionally.
- In other words, if some condition is met, then execute one statement. If a different condition is met, then execute another statement.
- We can accomplish this using IF-THEN logic.

```
IF expression THEN statement;
```

- For each observation, SAS evaluates `expression`. If it is true, it completes the corresponding `statement`.
- Then it moves on to the next observation.
- When you have multiple IF-THEN statements, SAS tests all conditions in sequence for every row of the data



# Conditional Processing

- Sometimes hierarchically processing things is desirable.
- In that case, an `else` statement is needed!

```
IF expression THEN statement;
<ELSE IF expression THEN statement;>
<ELSE IF expression THEN statement;>
<ELSE statement;>
```

- For each observation, SAS evaluates `expression`s top down until it reaches a TRUE evaluation.
- Then it completes the corresponding `statement`.
- The subsequent else statements are not processed.
- Then it moves on to the next observation.



# Your Turn

Let's group the vehicles in the cars data into 4 groups based on price (MSRP).

```
data cars1;
set sashelp.cars;
if MSRP<20000 then Cost_Group=1;
else if MSRP<40000 then Cost_Group=2;
else if MSRP<60000 then Cost_Group=3;
else Cost_Group=4;
keep Make Model Type MSRP Cost_Group;
run;
```

```
data cars2;
set sashelp.cars;
if MSRP<20000 then Cost_Group=1;
if 20000<=MSRP<40000 then Cost_Group=2
if 40000<=MSRP<60000 then Cost_Group=3
if MSRP >=60000 then Cost_Group=4;
keep Make Model Type MSRP Cost_Group;
run;
```

1. Do these two data steps evaluate identically? (i.e., cars1 = cars2?)
2. Imagine this data set was very large. Which data step would be more computationally efficient (faster)?



# Case Study

Previously, we examined category 5 storms. Let's go back to the original data set and create a column that specifies the storm category.

Google "storm categories".

Start a new script called t4_cs_conditional.sas.

This may prove useful:
https://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#a

## Creating character variables (conditionally)

We have to be a little more careful when conditionally creating character variables.

```
data cars2;
    set sashelp.cars;
    if MSRP<60000 then CarType="Basic";
    else CarType="Luxury";
    keep Make Model MSRP CarType;
run;


proc print data = cars2;
run;
```

| Make | Model | MSRP | CarType |
|---|---|---|---|
| Acura | MDX | $36,945 | Basic |
| Acura | RSX Type S 2dr | $23,820 | Basic |
| Acura | TSX 4dr | $26,990 | Basic |
| Acura | TL 4dr | $33,195 | Basic |
| Acura | 3.5 RL 4dr | $43,755 | Basic |
| Acura | 3.5 RL w/Navigation 4dr | $46,100 | Basic |
| Acura | NSX coupe 2dr manual S | $89,765 | Luxur |
| Audi | A4 1.8T 4dr | $25,940 | Basic |
| Audi | A41.8T convertible 2dr | $35,940 | Basic |



## Creating character variables (conditionally)

- The first occurrence of a column in the DATA step defines the name, type, and length of the column.
- The first occurrence of the newly created CarType variable was "Basic"; thus, the length was determined to be 5.
- As a result, "Luxury" is truncated to "Luxur".
- So what can we do?



## Creating character variables (conditionally)

- Foolproof your code: start by defining your character column in the data step with a `LENGTH` statement.
- The syntax for this statement is the keyword LENGTH followed by the name of the column, a dollar sign to indicate a character column, and the length you want to assign:

```
LENGTH char-column $ length;
```

Thus, the following code would execute as expected:

```
data cars2;
     set sashelp.cars;
     length CarType $ 6;
     if MSRP<60000 then CarType="Basic";
     else CarType="Luxury";
     keep Make Model MSRP CarType;
run;
```



## Your Turn

We want to be more descriptive about what oceans these storms are occurring in. We've got a start, but it's not perfect.

1. Open a new script and save it as t4_yt_ocean.sas.
2. Copy/paste and run the program below and examine the results.
3. Why is Ocean truncated?
4. What value is assigned when Basin='na'? Why?
5. To address 3 above: Add a `length` statement after the `set` statement to declare the name, type, and length of Ocean before the column is created.
6. To address 4 above: Add an assignment statement after the `keep` statement to convert Basin to uppercase. Run the program.
7. Move the LENGTH statement to the end of the DATA step. Run the program. Does it matter where the LENGTH statement is in the DATA step?

```
data storm_summary2;
    set s40dat.storm_summary;
    keep Basin Season Name MaxWindMPH Ocean;
    OceanCode=substr(Basin,2,1);
    if OceanCode="I" then Ocean="Indian";
    else if OceanCode="A" then Ocean="Atlantic";
    else Ocean="Pacific";
run;
```

# Last notes on conditional processing

Compound expressions? No problem?

```
data cars2;
    set sashelp.cars;
    if MPG_City>26 and MPG_Highway>30 then Efficiency=1;
    else if MPG_City>20 and MPG_Highway>25 then Efficiency=2;
    else Efficiency=3;
    keep Make Model MPG_City MPG_Highway Efficiency;
run;
```

For example, both MPG_City>26 AND MPG_Highway>30 need to evaluate to TRUE in order for Efficiency to be 1.



# Last notes on conditional processing

Compound *statement*? That's trickier...

```
data cars2;
    set sashelp.cars;
    length Cost_Type $ 4;
    if MSRP<20000 then Cost_Group=1 and Cost_Type="Low";
    else if MSRP<40000 then Cost_Group=2 and Cost_Type="Mid";
    else Cost_Group=3 and Cost_Type="High";
run;

/*The code above tries to conditionally process
Cost_Group and Cost_Type in the same if-else statement...
it does not work.*/
```



# Last notes on conditional processing

- If you want to execute MULTIPLE STATEMENTS for a given condition, you need to use an `if-else do` statement.

```
IF expression THEN DO;
    <executable statements>
END;
ELSE IF expression THEN DO;
    <executable statements>
END;
ELSE DO;
    <executable statements>
END;
```

If `expression` is TRUE then do all the executable statements before `END;`.



# Last notes on conditional processing

For example,

```
data cars2;
    set sashelp.cars;
    keep Make Model MSRP Cost_Group;
    if MSRP<20000 then do;
       Cost_Group=1;
       Cost_Type="Low"
    end;
    else if MSRP<40000 then do;
       Cost_Group=2;
       Cost_Type="Mid"
    end;
    else do;
       Cost_Group=3;
       Cost_Type="High";
    end;
run;
```