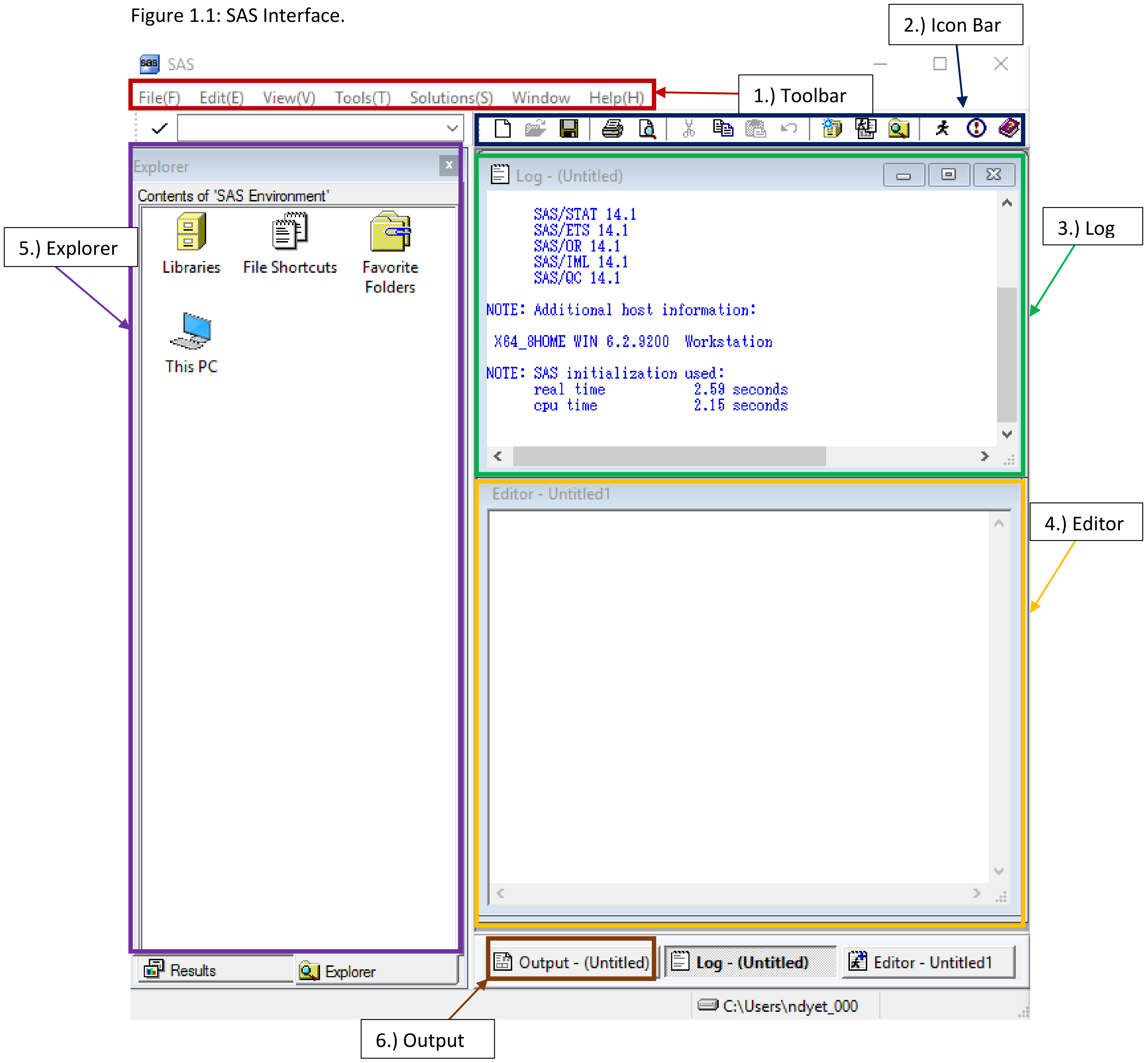
SAS MANUAL

NEIL YETZ

Table of Contents

THE SAS INTERFACE

Figure 1.1: SAS Interface.



```
      SAS/STAT 14.1
      SAS/ETS 14.1
      SAS/OR 14.1
      SAS/IML 14.1
      SAS/QC 14.1

NOTE: Additional host information:

 X64_8HOME WIN 6.2.9200  Workstation

NOTE: SAS initialization used:
      real time           2.59 seconds
      cpu time            2.15 seconds
```

1.) Toolbar

The toolbar contains several different functions. Within this included are the ability to *save, open and import files.*

2.) Icon Bar

The Icon Bar have quick functions such as saving, running the code and creating a new program. Hover over each image to determine the name of the image to get more insight of its function.

3.) Log Window

The log window reports errors and successful functions ran. If there is an error, normally some red text will be reported in this window to inform you that the function was not properly run. You will need to look into your code to assess for the error. Sometimes the text in the log will inform you of the type of error that was created within your code.

4.) Editor Window

The Editor Window is where all SAS code is written. For all code functions mentioned throughout the rest of this manual, please enter them into the editor window.

5.) Explorer Window

The Explorer Window is where you look to find where SAS files have been saved. Later in this manual, when we set up permanent and temporary SAS datasets, this is the window where you can find these files manually.

6.) Output Window

After you run your SAS code, results will appear here. Refer to your Output Window for all statistical analyses.

BASIC SAS FUNCTIONS

Selected Icon Explanations

The Save icon (💾)

- Use this icon to save your SAS code.
- During your first time saving, you will be prompted to choose a name and select a filepath of where to save your SAS coding.

"The Running Guy" Icon (🏃)

- This icon is used to run all your code or selected code.
- The code will run up to where your curser is current is placed among the SAS editor.
- To run selected code, simply highlight the lines that you would like to run and it will ONLY run the highlighted functions.

Selected code explanation:

Semi-Colon (;)

- A semi-colon should be used at the end of all command ends. The semi-colon indicates that that line of code is completed and it is time to look towards the command function. In the event your log window shows errors, always check to make sure ALL semi-colons are included appropriately.

Run;

- Run; is to be placed at the end of all command steps. Run; indicates that the command is complete and if one correctly, will be reported as complete in your log window or output window, depending on the command function.

Data=

- Whenever "data=" is seen in code, it is referring to the dataset that is to be analyzed. Most data commands require the use of data= so that it knows which dataset to analyze.

Asterisk (*;)

- Asterisks within the SAS code indicate that the code afterwards will not be read when ran (i.e. when you press "the running guy"). Often times, notes and important information about the code you are running are placed within these asterisk statement, so be sure to look at what they have to say!
- Note: all code that will not be read will appear GREEN within your program code. In order to end the asterisk non-read statement, place a semi-colon at the end.
- **ALSO NOTE**: The asterisk will be used in other statements, but when it is put at the beginning of a code it serves a separate function that within code.

Working With the Data Set

Creating a dataset

Sometimes, rather than importing a .csv file (instructions to import a .csv file below) we need to create a SAS dataset. Using the “cards” function.

Below is some example code of how to create a dataset through SAS.

```
data test;
input lung_cancer smoking;
cards;
1 1
0 0
1 0
run;
```

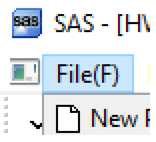
- `data test;` = creating a new data set, the name of the data set is “test” in this instance
- `input` = naming the variables in your data set. You can add as many as you want and they are separated using a “space”
- `lung_cancer smoking;` = the variables being inputted, in this case, the variables are “lung cancer” and “smoking”
- `cards;` = inititating the “cards” statement. After this is when you enter in the literal data cells. They need to match-up with your variables in the columns and rows.
- `run;` = run statement.

Importing a Data Set

1.) **Make sure data is saved as a .csv file**

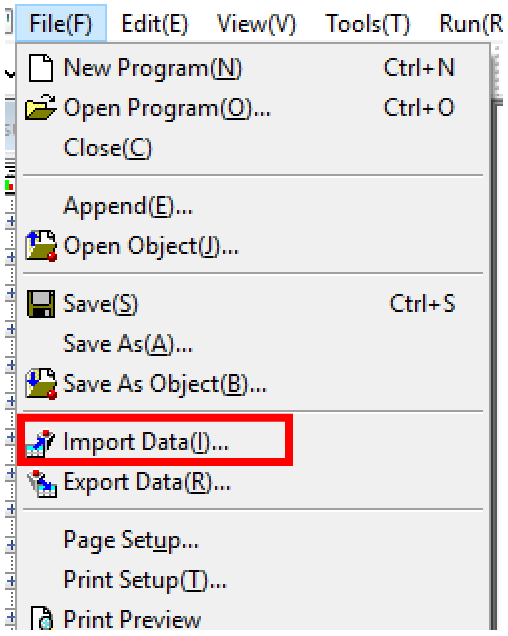
2.) **Click “File”**

Figure 2.1: screen shot of “file” on the SAS interface.

a. 

3.) Click "Import data"

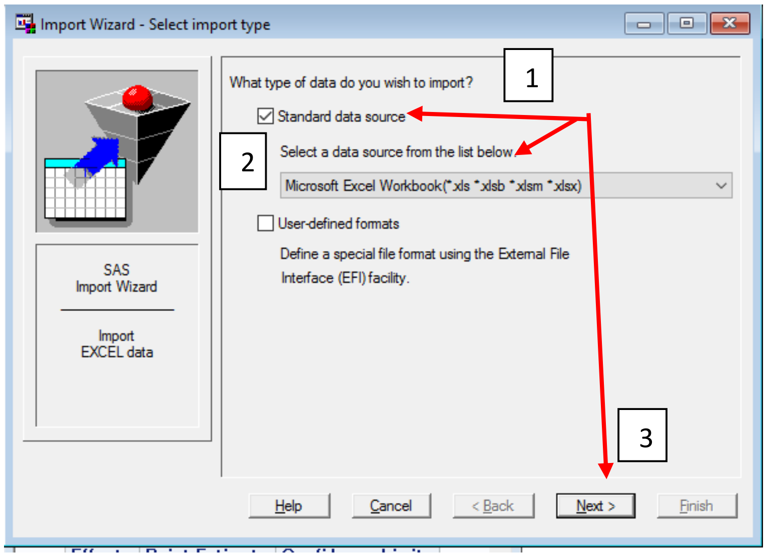
Figure 2.2: Screenshot of "File" expanded after clicking and "Imported data" highlighted.

a. 

4.) In the Import Wizard:

a. Check to make sure "Standard data source" is checked (1) and the data source is a "Microsoft Excel Workbook" (2) Then Click "Next" (3)

Figure 2.3: Screenshot of step 1 in the "Import Wizard".

i. 

Import Wizard - Select import type

What type of data do you wish to import?

☑ Standard data source

Select a data source from the list below

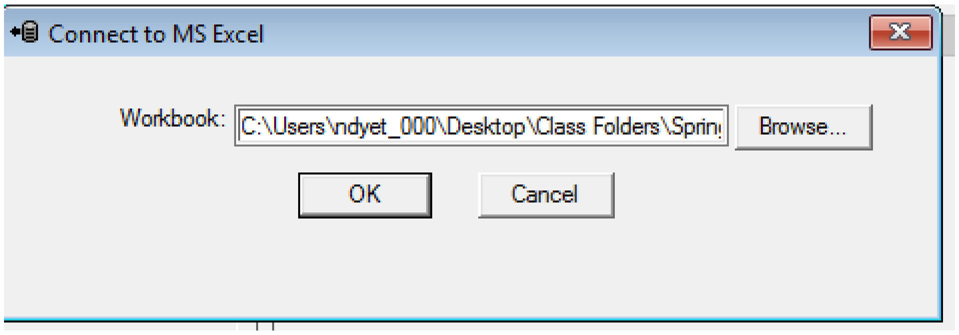
Microsoft Excel Workbook(*.xls *.xlsb *.xlsm *.xlsx)

☐ User-defined formats

Define a special file format using the External File Interface (EFI) facility.

b. On the next screen click "Browse" to find the file you wish to import. After finding the correct file, click "OK". (HINT: MAKE SURE "ALL FILES" is chosen to show all files in listed folders)

Figure 2.4: Screenshot of the Browse function of the import wizard.

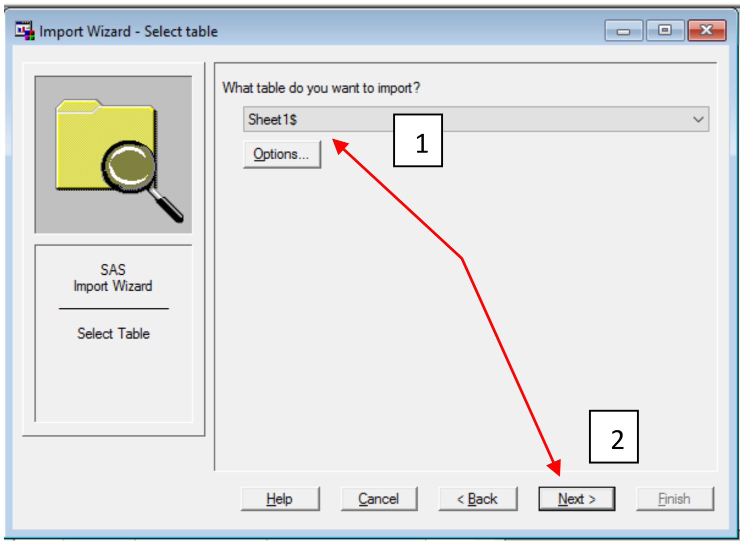
i. 

Connect to MS Excel

Workbook: C:\Users\ndyet_000\Desktop\Class Folders\Sprin

c. **Select Sheet you would like to import, Click "Next"**

Figure 2.5: Screenshot of the with numbers corresponding to instruction item (d).



What table do you want to import?

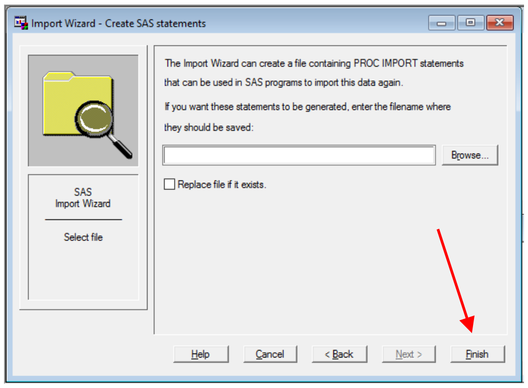
Sheet1$

i.

d. **Select where in SAS You would like the file to be saved (1), Click "Next"(2)**

e. **On the next screen, click "Finish" and your data will be imported**

Figure 2.6: Screenshot of the import wizard final step.



The Import Wizard can create a file containing PROC IMPORT statements that can be used in SAS programs to import this data again.

If you want these statements to be generated, enter the filename where they should be saved:

Replace file if it exists.

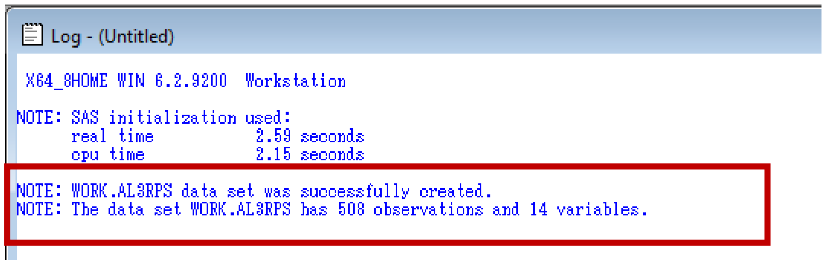
i.

f. **CHECK THE LOG!**

i. **In order to ensure that the data was properly imported you can check by going to the log.**

1. **If properly imported, the log will tell you how many observations and variables were properly imported**

Figure 2.7: Screenshot of log after data has successfully been imported.



```
 X64_8HOME WIN 6.2.9200  Workstation

NOTE: SAS initialization used:
      real time           2.59 seconds
      cpu time            2.15 seconds

NOTE: WORK.AL3RPS data set was successfully created.
NOTE: The data set WORK.AL3RPS has 508 observations and 14 variables.
```

2.

The “Datastep”

A “datastep” is you informing SAS what data you want to work with. Your data step is essential to create any command functions. Once a dataset has been imported, creating a dataset, allows you to access that data and properly utilize it for analyses. The datastep consists of the *libname statement*, permanently saving your dataset. Furthermore, later in this manual, when we begin to transform variables, these will be created *within* your datastep.

In SAS, be sure and use as few data steps as possible. This leads to less errors and better understanding of the code that is written. Optimally, you will have a total of ONE data step per SAS program script.

Creating a libname statement

A libname statement is an *ESSENTIAL* data step. The purpose of the libname statement is to tell you where the data that you will be working with should be pulled from. Essentially, it is a *filepath*, it is especially important you know where your data files are store. The libname statement is necessary for **ALL** SAS programming data steps. This will be at the top of all SAS codes.

To create a libname statement:

1.) Create a folder to store your files in.
   a. This folder should be used to store ALL data files associated with your current project.
2.) Within your windows file explorer, find the folder that your data is saved.
3.) Copy the file path associated with the files.
4.) Go back to the SAS editor and type *libname.*
5.) Press space
6.) Name what you would like the new libname reference to be called (for example, *sdat*)
7.) Press space
8.) Copy the filepath and put an apostrophe (‘) at each end of the filepath.
9.) end with a semicolon (;)

Statement **Reference** **File path**

EXAMPLE LIBNAME STATEMENT = `libname sdat 'C:\Users\ndyet_000\Desktop\Class Folders\Spring 2016\ERHS 642\Data';`

NOTE: Do NOT end statement with “run;” YOU ARE NOT DONE WITH YOUR DATA STEP! You will need still need to set up more statements to create permanent and temporary SAS data sets. These instructions are next.

Permanently save a data set

After you have imported the data and created a libname statement, it is time to permanently save the new SAS file into your computer. This prevents you from needing to re-import your data every time and enables you to open the data with ease.

To create a permanent SAS dataset include this statement AFTER your libname statement:

1.) Here is an example code to permanently save the new data set:
   a. `data sdat.AL3RPS; set AL3RPS;`
      i. `data sdat.AL3RPS` = informing SAS to permanently save a data set to the sdat library (that you set earlier in the libname statement instructions).

            ii. `Set AL3RPS` = Instructing SAS that this permanent dataset is the AL3RPS dataset.

2.) Once this is completed, you only need to do this ONCE per data importation.
3.) Assuming you have no errors in your log, the file should be permanently saved in the file path you set.
4.) This step only needs to be done ONCE. Once you have done this step, it will be stored in the library in your computer. It will be placed wherever your libname filepath statement led the file to be saved. If you are unsure if it was saved correctly, follow your libname filepath and it should be saved as a SAS data file.

NOTE: Do NOT end statement with "run;" YOU ARE NOT DONE WITH YOUR DATA STEP! You will need still need to set up more statements to Access the newly saved data set. Which is mentioned in the next instruction set labeled "Accessing Datasets".

Accessing Datasets

After you have permanently saved the SAS dataset, accessing the dataset is necessary because it allows you to manipulate, transform and do anything to a temporary dataset without tampering the original data set.

To access a SAS dataset include this statement AFTER your libname statement and it is necessary that you have permanently saved your data step (refer to last step if you have not done this yet):

1.) Here is an example code to access a previously saved dataset:
    a. `data AL3RPS; set sdat.AL3RPS; run;`
        i. `data AL3RPS` = Instructing SAS to pull the correct data set. In this case, the data set is named "AL3RPS"
        ii. `Set sdat.AL3RPS` = Informs SAS to set the AL3RPS file within the sdat folder into a temporary data set.
        iii. `run; = informing SAS to run the code. This is the ending of the data step.`
    b. Once this step is completed: All analyses, manipulations, and transformations will only be done on the temporary dataset. Preventing and unwanted changes to the original dataset.

NOTE: The "run;" statement IS at the end of this command statement. This is necessary in order for complete the entire datastep. Please also note that later in this manual, whe we discuss data transformation, we will move the "run;" statement to end after the manipulation and transforming of variables.

Getting to Know Your Data & Descriptive Functions

Always get to know your data before you begin analyzing it. You'll be a better researcher for it!

Viewing your data (Proc Print)

There are many reasons to view the data. For one, after you run your datastep it can reassure that everything was imported and is going to be analyzed correctly. Also, it may be a good way to look for extreme errors in the entry of the data.

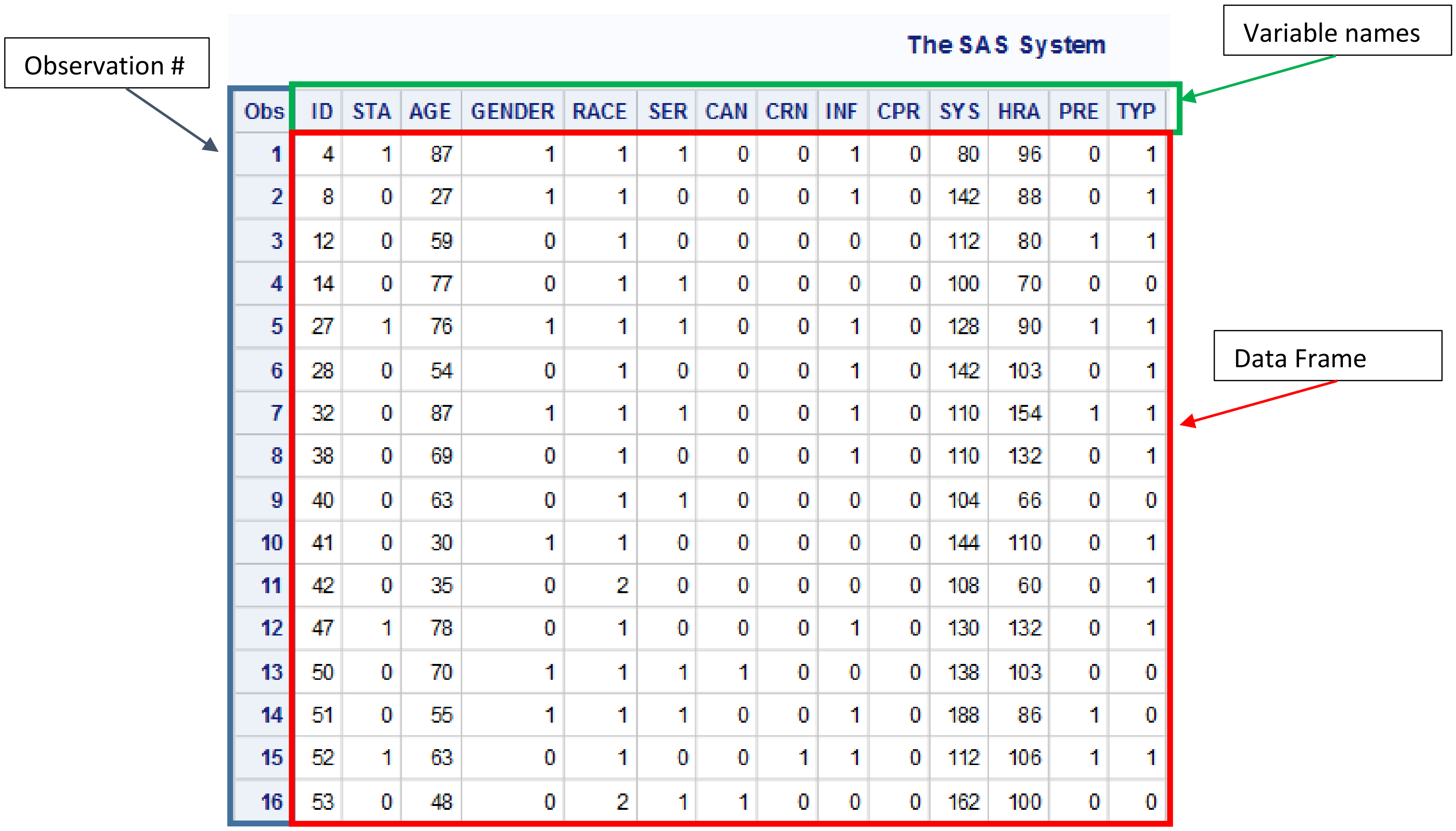
The proc print function allows you to view data sets that you are currently working with within your SAS program.

Example code: `proc print data = ICU; run;`

- `proc print` = Command function to print your data
- `data = ICU; = which dataset you would like to be printed (In this instance it is the ICU dataset being printed, change per necessary)`
- `run;` = run command

Once you run the code, it will appear in your output window if done correctly. As shown in the below figure:

Figure 3.1: Screenshot of proc print output of the ICU dataset with selected explanations.



The SAS System

Obs	ID	STA	AGE	GENDER	RACE	SER	CAN	CRN	INF	CPR	SYS	HRA	PRE	TYP
1	4	1	87	1	1	1	0	0	1	0	80	96	0	1
2	8	0	27	1	1	0	0	0	1	0	142	88	0	1
3	12	0	59	0	1	0	0	0	0	0	112	80	1	1
4	14	0	77	0	1	1	0	0	0	0	100	70	0	0
5	27	1	76	1	1	1	0	0	1	0	128	90	1	1
6	28	0	54	0	1	0	0	0	1	0	142	103	0	1
7	32	0	87	1	1	1	0	0	1	0	110	154	1	1
8	38	0	69	0	1	0	0	0	1	0	110	132	0	1
9	40	0	63	0	1	1	0	0	0	0	104	66	0	0
10	41	0	30	1	1	0	0	0	0	0	144	110	0	1
11	42	0	35	0	2	0	0	0	0	0	108	60	0	1
12	47	1	78	0	1	0	0	0	1	0	130	132	0	1
13	50	0	70	1	1	1	1	0	0	0	138	103	0	0
14	51	0	55	1	1	1	0	0	1	0	188	86	1	0
15	52	1	63	0	1	0	0	1	1	0	112	106	1	1
16	53	0	48	0	2	1	1	0	0	0	162	100	0	0

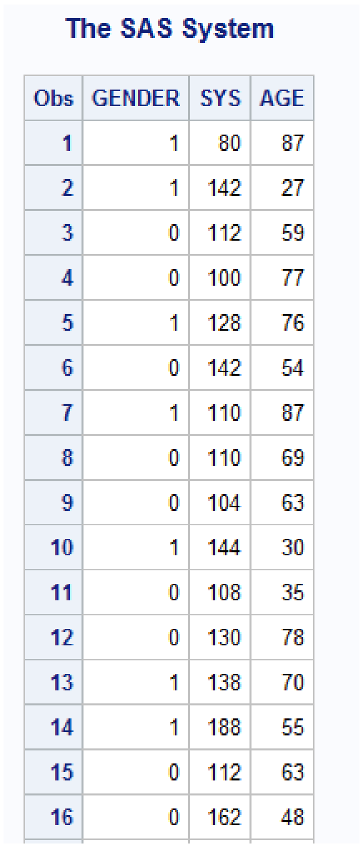
RESTRICTING PROC PRINT TO ONLY LIST SPECIFIC VARIABLES:

It is possible to restrict what SAS prints out in its proc print statement by using the “var” statement. The var statement is to be placed after you have specified which dataset you would like to be printed (i.e. data =ICU;) but before your run statement. After you enter the var statement you can enter the specific variable you would like to be printed out. Separated by spaces. Be sure to put a semi-colon at the end of the Var statement

Example Proc Print with Var statement:
```
proc print data = ICU;
var GENDER SYS AGE;
run;
```

- `var GENDER SYS AGE;` = Telling the proc print function to only display the variables of “Gender” “SYS” & “AGE”

Figure 3.2: Screenshot of output restricted to Gender, SYS, & Age from proc print function with var command added.



The SAS System

Obs	GENDER	SYS	AGE
1	1	80	87
2	1	142	27
3	0	112	59
4	0	100	77
5	1	128	76
6	0	142	54
7	1	110	87
8	0	110	69
9	0	104	63
10	1	144	30
11	0	108	35
12	0	130	78
13	1	138	70
14	1	188	55
15	0	112	63
16	0	162	48

Sorting Your Data (Proc Sort)

Sorting your data can be useful for many reasons. It may make the data easier to view when you run the proc freq command if it is sorted in a certain way.

The Proc sort function sorts you data by levels of a cerain variable (i.e by descending or ascending age).

Example Proc Sort command:
```
proc sort data=ICU;
by decending AGE;
run;
```

- `proc sort` = Command function to sort your data
- `data = ICU;` = which dataset you would like to be printed (In this instance it is the ICU dataset being printed, change per necessary)
- `by descending AGE;` = Telling the proc sort to sort the data by Age and specifically put the observations with a higher age at the top, and the lower ages at the bottom.
- `run;` = run command

Descriptive Functions

Descriptive functions are used to understand and explain your data. The next set of functions within this section are meant to output descriptive statistics including, but not limited to: Means, standard deviations, quartiles and ranges.

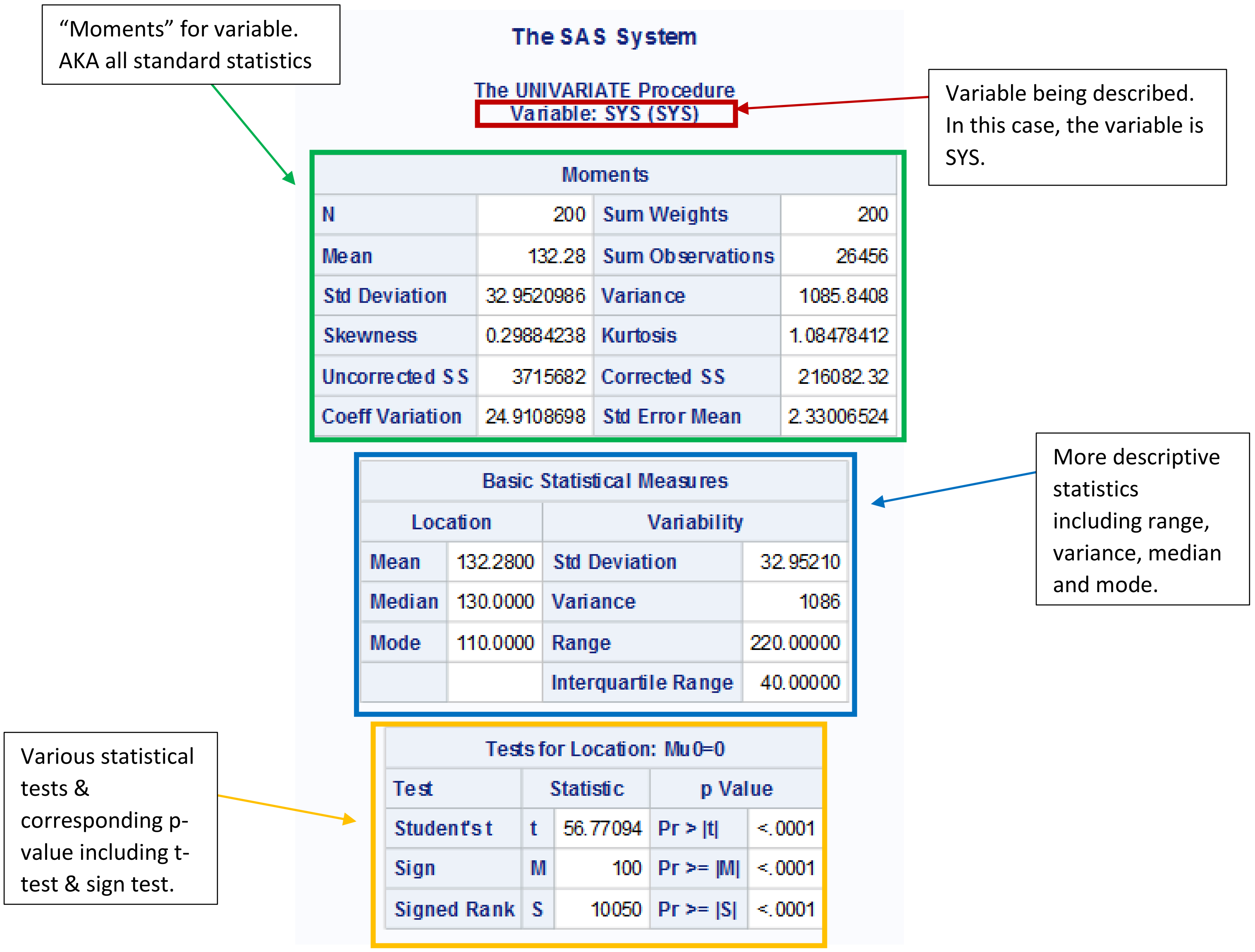
ALL Descriptives (Proc Univariate)

The Proc Univariate function prints ALL descriptive statistics and is useful for getting a large understanding of your data. It can be used to receive information on every variable in the dataset or it can be used with the combination of the "var" function to receive information of specific variables.

Example Proc Univariate Function: `proc univariate data=ICU; run; var SYS; run;`

- `proc univariate` = Command function to give you descriptive statistics.
- `data = ICU; = which dataset you would like to be printed (In this instance it is the ICU dataset being printed, change per necessary)`
- `var SYS;` = Telling the proc univariate function to only display the descriptive statistics of the variable "SYS".
- `run;` = run command

Figure 4.1: Example of first half of proc univariate output with selected explanations.


"Moments" for variable. AKA all standard statistics
Variable being described. In this case, the variable is SYS.
More descriptive statistics including range, variance, median and mode.
Various statistical tests & corresponding p-value including t-test & sign test.

The SAS System

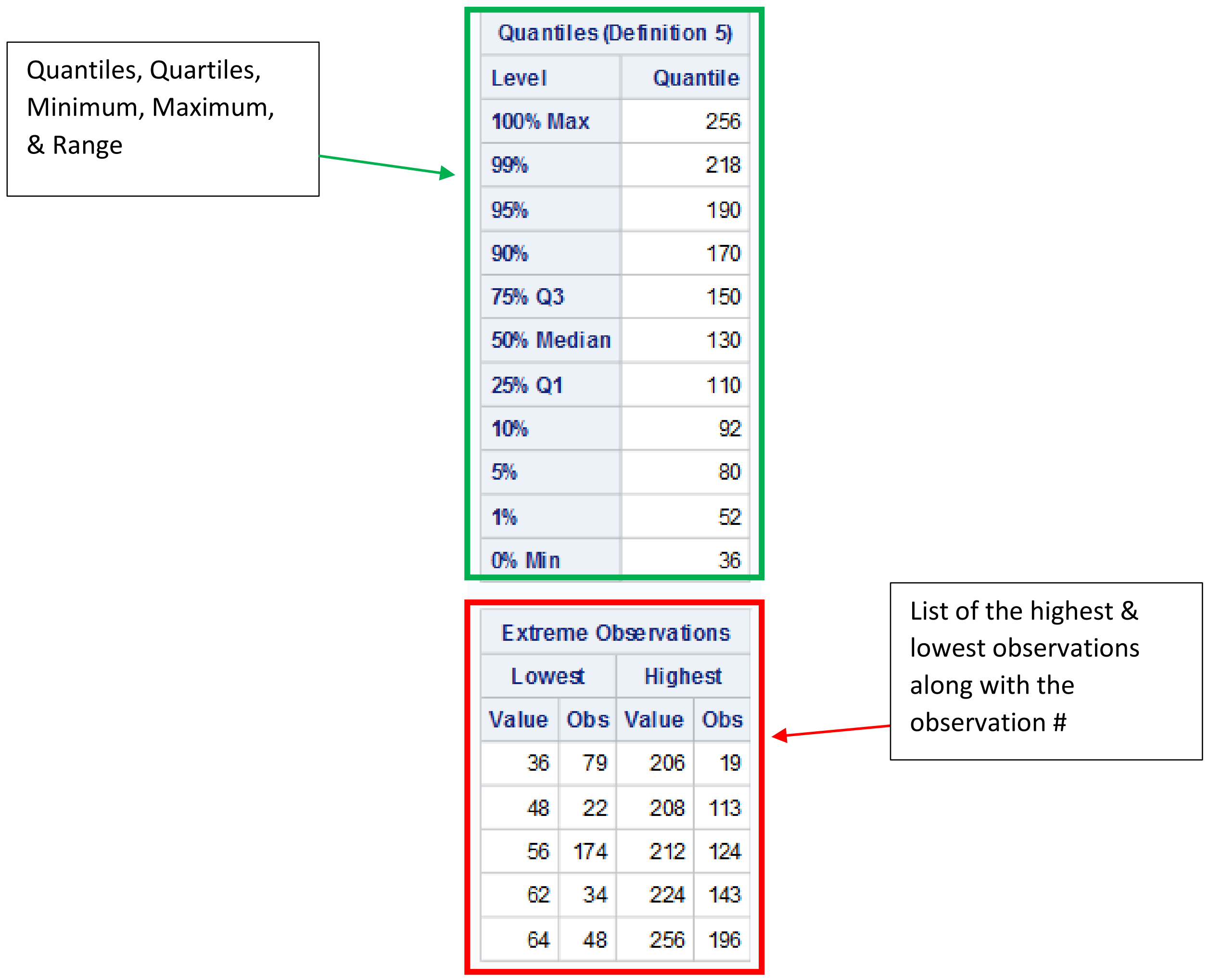
The UNIVARIATE Procedure
Variable: SYS (SYS)

Moments			
N	200	Sum Weights	200
Mean	132.28	Sum Observations	26456
Std Deviation	32.9520986	Variance	1085.8408
Skewness	0.29884238	Kurtosis	1.08478412
Uncorrected SS	3715682	Corrected SS	216082.32
Coeff Variation	24.9108698	Std Error Mean	2.33006524

Basic Statistical Measures			
Location		Variability	
Mean	132.2800	Std Deviation	32.95210
Median	130.0000	Variance	1086
Mode	110.0000	Range	220.00000
		Interquartile Range	40.00000

Tests for Location: Mu0=0				
Test	Statistic		p Value	
Student's t	t	56.77094	Pr > |t|	<.0001
Sign	M	100	Pr >= |M|	<.0001
Signed Rank	S	10050	Pr >= |S|	<.0001

Figure 4.2: Continuation of output from Proc Univariate command with selected explanations



Quantiles, Quartiles, Minimum, Maximum, & Range

Quantiles (Definition 5)	
Level	Quantile
100% Max	256
99%	218
95%	190
90%	170
75% Q3	150
50% Median	130
25% Q1	110
10%	92
5%	80
1%	52
0% Min	36

List of the highest & lowest observations along with the observation #

Extreme Observations			
Lowest		Highest	
Value	Obs	Value	Obs
36	79	206	19
48	22	208	113
56	174	212	124
62	34	224	143
64	48	256	196

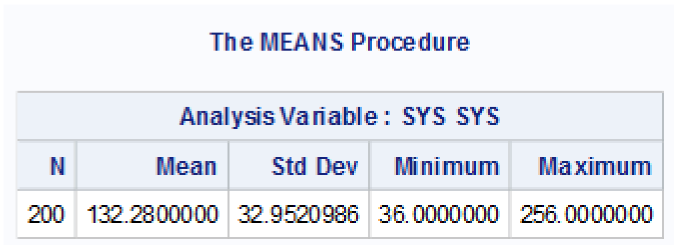
LIMITED Descriptives (Proc Means)

The Proc Means command is similar to the Proc Univariate function, but it only supplies: The number of observations, *the mean, standard deviation, and the Minimum/Maximum* of selected variables. Proc Univariate contains all the information you can get from proc means, but proc means is a more precise command that may be easier to read.

Example Proc Means command: `proc means data=ICU; var SYS; run;`

- `proc means` = Command function to give you limited descriptive statistics.
- `data = ICU;` = which dataset you would like to be printed (In this instance it is the ICU dataset being printed, change per necessary)
- `var SYS;` = Telling the proc univariate function to only display the descriptive statistics of the variable "SYS".
- `run;` = run command

Figure 5.1: Output created from the Proc Means command.



The MEANS Procedure

Analysis Variable : SYS SYS				
N	Mean	Std Dev	Minimum	Maximum
200	132.2800000	32.9520986	36.0000000	256.0000000

ADDING and arranging with Proc Means

Within the Proc means command you can detail also: display added information, choose not to display certain information, and order the display how you would like.

An example of this is:

```
proc means data=ICU
n min median max mean std;
var SYS;
run;
```

- `proc means` = Command function to give you limited descriptive statistics.
- `data = ICU;` = which dataset you would like to be printed (In this instance it is the ICU dataset being printed, change per necessary)
- `n` = display number of observations first
- `min`= display the minimum 2nd
- `median` = display the median 3rd
- `max` = display the maximum 4th
- `mean` = display the mean 5th
- `std;` = display the standard deviation last
- `var SYS;` = Telling the proc means function to only display the descriptive statistics of the variable “SYS”.
- `run;` = run command

Frequency Tests (Proc Freq)

The Proc Freq command simply creates a frequency table for a certain variable. You can get the frequency of 1 variable. Create a table accounting for 2 variables or even a table controlling for 1 variable.

Proc Freq for 1 variable:

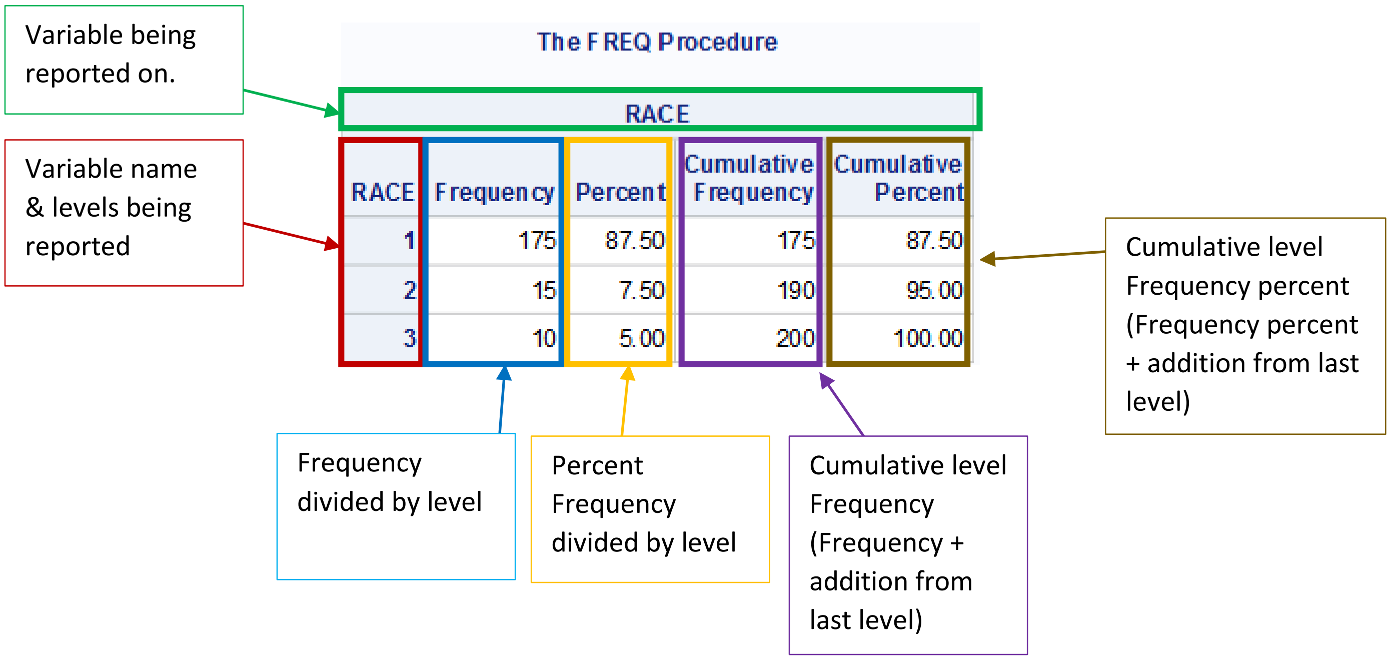
Example Proc Freq Command for 1 variable:

```
proc freq data=ICU order=data;
tables RACE;
run;
```

- `proc freq` = frequency command
- `data=ICU` = which dataset you would like to be printed (In this instance it is the ICU dataset being printed, change per necessary)

- `order=data;` = order the data as it appears.
- `tables RACE;` = The specific variable you would like to get frequency report on. In this case, the variable was “RACE”.
- `run;` = run command

Figure 6.1: Example output of Frequency report with selected explanations.



```
                 The FREQ Procedure

                        RACE
                                    Cumulative  Cumulative
RACE  Frequency  Percent             Frequency     Percent
   1        175    87.50                   175       87.50
   2         15     7.50                   190       95.00
   3         10     5.00                   200      100.00
```

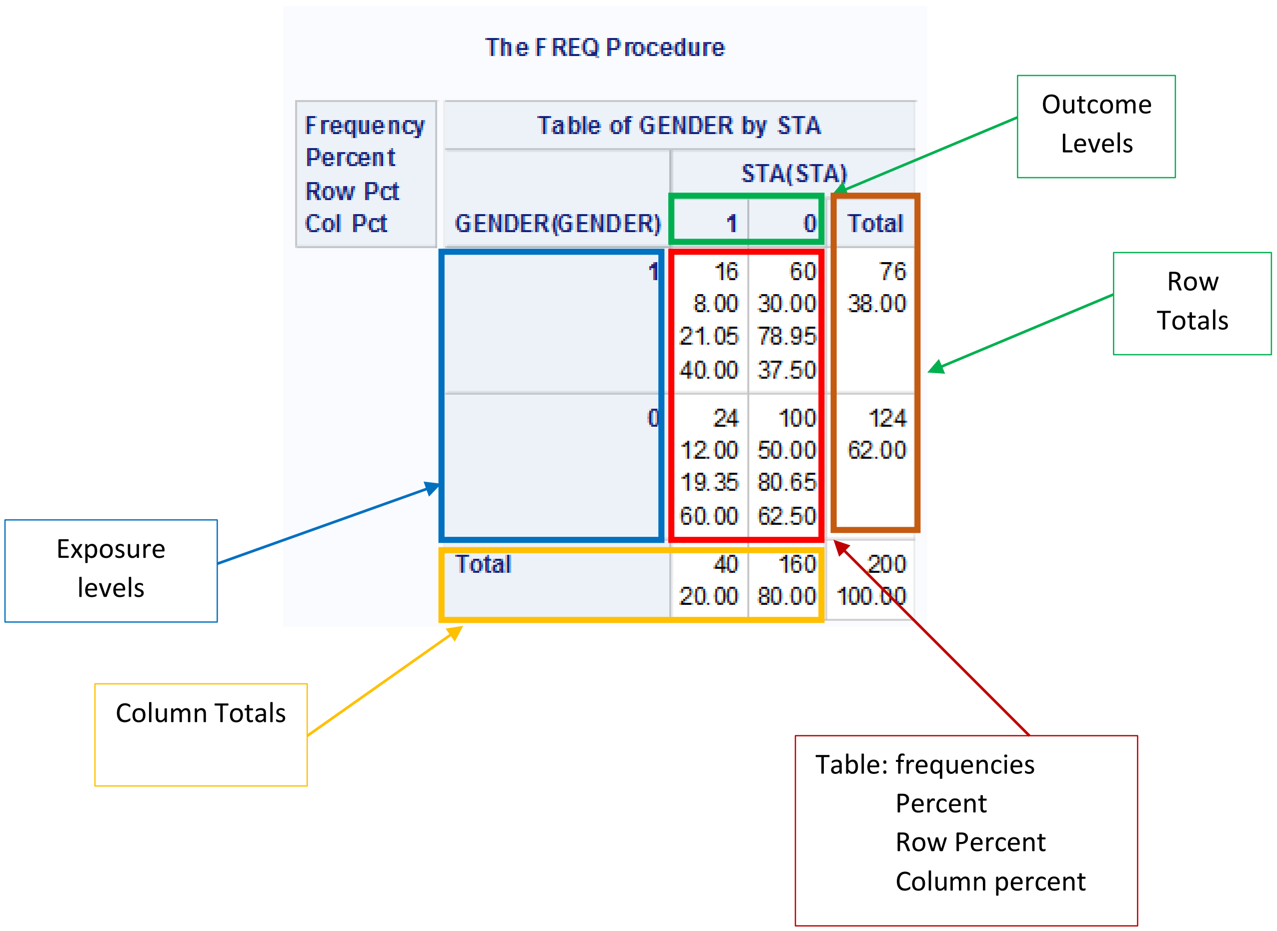
Variable being reported on.
Variable name & levels being reported
Cumulative level Frequency percent (Frequency percent + addition from last level)
Frequency divided by level
Percent Frequency divided by level
Cumulative level Frequency (Frequency + addition from last level)

Proc Freq for 2 variables (use this to create 2x2 tables):

Example Proc Freq Command for 2 variables:
```
proc freq data=ICU order=data;
tables GENDER*STA;
run;
```

- `proc freq` = frequency command
- `data=ICU` = which dataset you would like to be printed (In this instance it is the ICU dataset being printed, change per necessary)
- `order=data;` = order the data as it appears.
- `tables GENDER*STA;` = The specific variables you would like to get frequency table report on. In this Case, it is our exposure (race) by outcome (STA). **NOTE: when assessing outcomes & exposures/predictors, put the out come to the RIGHT of the asterisk. This will allow SAS to create the table to standard epidemiologic practice of the outcome on top & the exposure on the side.**
- `run;` = run command

Figure 6.2: Proc Freq output with selected explanations.



The F REQ Procedure

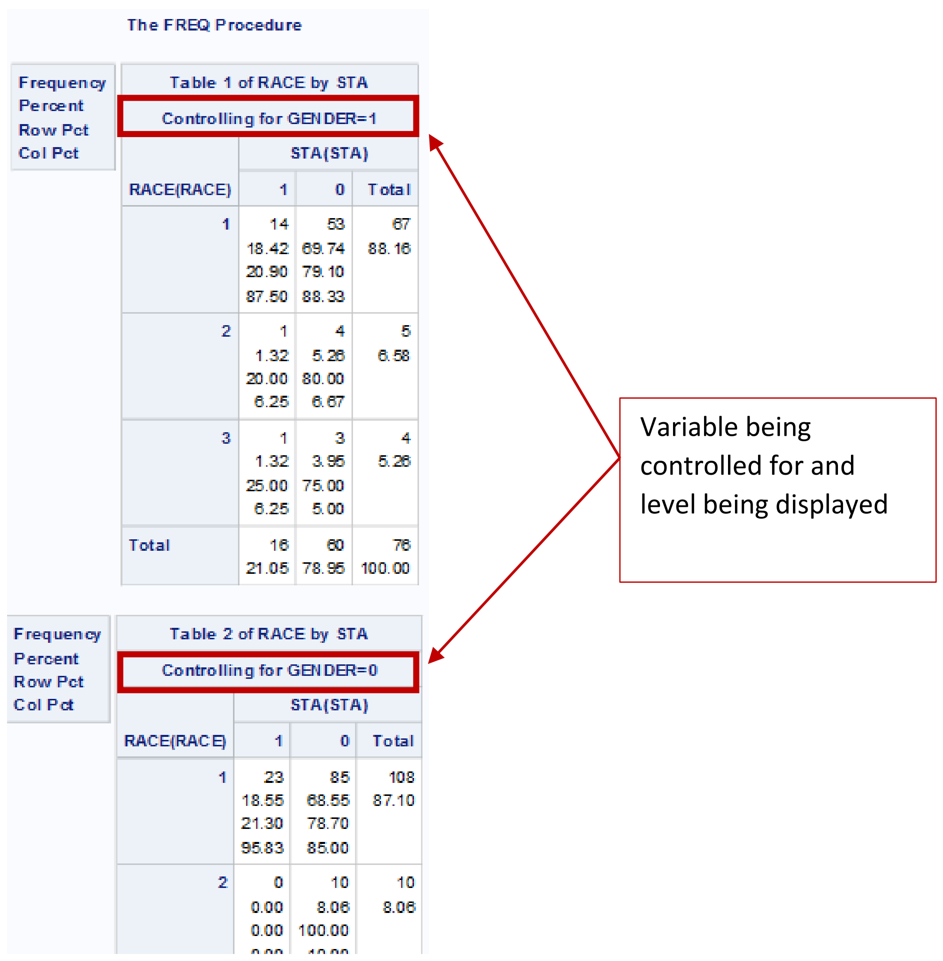
Frequency / Percent / Row Pct / Col Pct	Table of GENDER by STA		
	STA(STA)		
GENDER(GENDER)	1	0	Total
1	16	60	76
	8.00	30.00	38.00
	21.05	78.95	
	40.00	37.50	
0	24	100	124
	12.00	50.00	62.00
	19.35	80.65	
	60.00	62.50	
Total	40	160	200
	20.00	80.00	100.00

Proc Freq for 2 variables controlling for another variable

Example Proc Freq Command for 2 variables:
```
proc freq data=ICU order=data;
tables GENDER*RACE*STA;
run;
```

- `proc freq` = frequency command
- `data=ICU` = which dataset you would like to be printed (In this instance it is the ICU dataset being printed, change per necessary)
- `order=data;` = order the data as it appears.
- `tables GENDER*RACE*STA;` = The specific variables you would like to get frequency table report on. In this Case, it is our exposure (race) by outcome (STA) being separated by gender. **NOTE: multiple tabes will appear. # of tables that are output are dependent on the variable being controlled for (In this instance, GENDER)**
- `run;` = run command

Figure 6.3: Example output of Proc freq command separated by a control variable



The FREQ Procedure

Frequency
Percent
Row Pct
Col Pct

Table 1 of RACE by STA

Controlling for GENDER=1

	STA(STA)		
RACE(RACE)	1	0	Total
1	14	53	67
	18.42	69.74	88.16
	20.90	79.10	
	87.50	88.33	
2	1	4	5
	1.32	5.26	6.58
	20.00	80.00	
	6.25	6.67	
3	1	3	4
	1.32	3.95	5.26
	25.00	75.00	
	6.25	5.00	
Total	16	60	76
	21.05	78.95	100.00

Frequency
Percent
Row Pct
Col Pct

Table 2 of RACE by STA

Controlling for GENDER=0

	STA(STA)		
RACE(RACE)	1	0	Total
1	23	85	108
	18.55	68.55	87.10
	21.30	78.70	
	95.83	85.00	
2	0	10	10
	0.00	8.06	8.06
	0.00	100.00	

Log Window warnings

Zero/Sparse Cells

If a table created through variables has 0 or sparse cells, then a warning will appear in your log window when reunning a proc logistic function.

The warning appears in your log window as so:

```
WARNING: There is possibly a quasi-complete
separation of data points. The maximum likelihood
estimate may not exist.

WARNING: The LOGISTIC procedure continues in spite
of the above warning. Results shown are based on the
last maximum likelihood iteration. Validity of the model
fit is questionable.
```

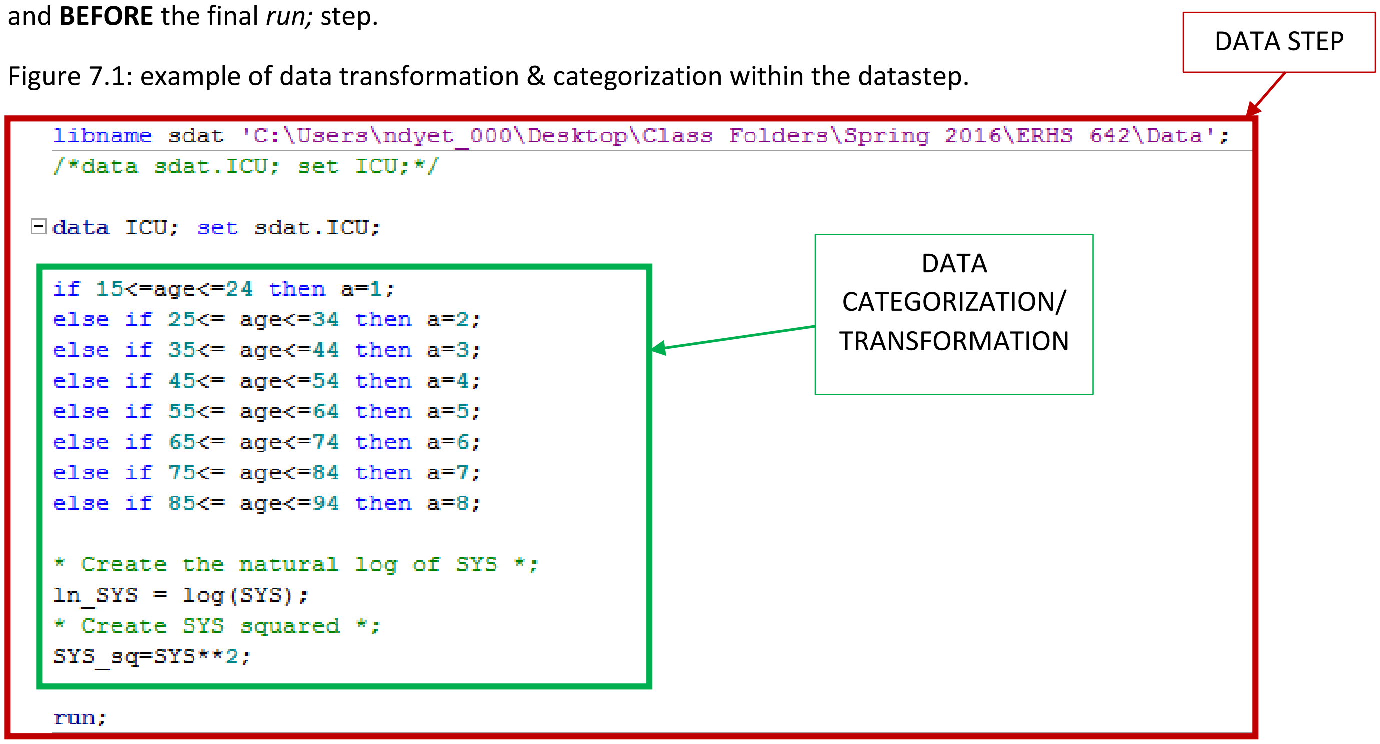
Please be aware.

CATEGORIZING & MANIPULATING VARIABLES

A Note regarding the Data Step

Recall from the previous section labeled "The data step". It is important whenever producing any categorization or manipulation of variables that this section contains are produced *within* the first data step. In other words, code within this section should be put **AFTER** the *ACCESS YOUR DATA* code and **BEFORE** the final *run;* step.

Figure 7.1: example of data transformation & categorization within the datastep.



```
libname sdat 'C:\Users\ndyet_000\Desktop\Class Folders\Spring 2016\ERHS 642\Data';
/*data sdat.ICU; set ICU;*/

data ICU; set sdat.ICU;

if 15<=age<=24 then a=1;
else if 25<= age<=34 then a=2;
else if 35<= age<=44 then a=3;
else if 45<= age<=54 then a=4;
else if 55<= age<=64 then a=5;
else if 65<= age<=74 then a=6;
else if 75<= age<=84 then a=7;
else if 85<= age<=94 then a=8;

* Create the natural log of SYS *;
ln_SYS = log(SYS);
* Create SYS squared *;
SYS_sq=SYS**2;

run;
```

Categorizing Variables

Categorizing variables is an essential part to analyze data. Often times, we need to create categories for continuous variables. In this example, we will be categorizing a range of age (Variable = AGE) values into 8 separate categories.

The 8 categories we will convert categorize the AGE variable to are as follows:

1. between 15 and 24
2. between 25 and 34
3. between 35 and 44
4. between 45 and 54
5. between 55 and 64
6. between 65 and 74
7. between 75 and 84
8. between 85 and 94

To do this the SAS code looks like this, it is produced through a series of "IF THEN" statements:

```
if 15<=age<=24 then a=1;
else if 25<= age<=34 then a=2;
else if 35<= age<=44 then a=3;
else if 45<= age<=54 then a=4;
else if 55<= age<=64 then a=5;
else if 65<= age<=74 then a=6;
else if 75<= age<=84 then a=7;
else if 85<= age<=94 then a=8;
```

- `if` = Telling SAS that IF the falling values fall between certain categories then they need to be categorized
- `15<=` = if the values fall below(or equal to) **15** then categorize
- `age` = the variable that is being categorized (in this instance the variable beig categorized is AGE)
- `<=24` = if the value falls below (or equal to) **24** the categorize
- `then a=1;` = telling SAS to categorize the new range of variables into a new variable (the new variable being created in this example is "a")
- `else if` = Telling SAS that the categorization procedure is continuing and that you are creating another level for the newly categorized variable.
- Repeat these steps until all value ranges are completed and categorized.

Once you run (or re-run) the data step with this procedure implemented it will create a new variable with the variable "a" and the age categories will range from 1-8 corresponding with the observations range of values under in the variable AGE.

NOTE: To check if your function worked correctly, run the data step and use the proc print command with your newly created variable inputted in the "var" section to see if the data is truly in the temporary data set now.

Categorization Functions

Below is a list and explanation of different ways to create categorization ranges. This is the portion in between the "IF" and "THEN" statement.

- #<= variable = number less than or equal to in variable
- #<variable = number less than in variable
- #=>variable = number Greater than or equal to in variable
- #>variable = number is greater than in variable
- #=variable = Number is equal to in variable
- Variable<=# =in variable is less than or equal to #
- Variable=># = in variable is greater than or equal to #
- Variable=# = in variable is equal to number

Design Variables (Dummy Variables)

Design variables are necessary to be created for categorical variables that have multiple levels (i.e. 1,2,3) but are not continuous in nature (For examples 1 is not necessarily worse than3 and 3 is not any better than 2). Race commonly needs to be converted into a design variable (i.e 1 = white, 2 = Asian, 3 =Black… etc.)

Creating design variables is similar to categorizing variables except that we are splitting one variable into multiple variables and only using 0's and 1's. We are not using a range. **Remember:** design variables are used to stop SAS from interpreting a variable as continuous.

For our example, we will be creating a design variable for Race. The levels for race are as follows:

1 = White (Reference category)
2 = Black
3 = Other

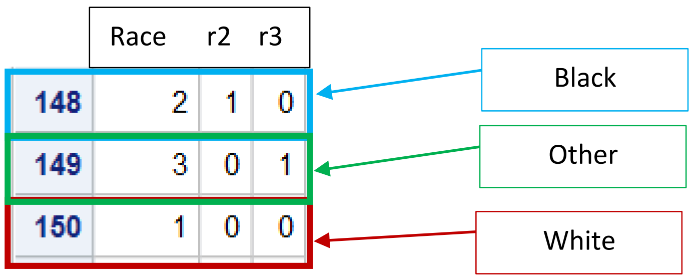
The design variable coding for this is:

```
if RACE=1 then do; r2=0; r3=0; end;
else if RACE=2 then do; r2=1; r3=0; end;
else if RACE=3 then do; r2=0; r3=1; end;
run;
```

- `If RACE=1` = Telling SAS to find all the 1's in the Variable Race
- `then do; r2=0; r3=0;` = Telling SAS that for all of the values equivalent to 1, put a 0 in the new design variables lableled r2 amd r3. The first line (where your design variables are = 0 should always be your reference category).
- `end;` = Unlike the categorizing section, this "end;" step is essential to every design variable created. Be sure to include it or the code will not work properly.
- `else if`= Telling SAS that the design variable procedure is continuing and that you are creating another level for the newly formed dummy variable.
- Repeat these steps until all dummy variables are created. **NOTE: Number of design variables needed to be created is equivalent to the # of categories minus 1 (c-1).**

Use Proc Print command to make sure they were properly created.

Figure 7.2: Example of newly created design Variable in data set



	Race	r2	r3
148	2	1	0
149	3	0	1
150	1	0	0

Manipulating Variables

Sometimes data manipulation is necessary in Logistic regression. Below we show how to manipulate variables so it converts each cell in the dataset to the natural log, square it or get the square root.

The Natural Log (ln(variable))

The code create the natural log of data within a variable is the “log();” statement.

Here is an example code: `ln_SYS = log(SYS);`

- `ln_SYS` = The name of the new variable you are creating that has the natural log function applied to it. This can be named whatever you want. It is recommended that the format for the new variable name created from this function simply be “ln_variable” to keep things clear in your analyses.
- `= log(SYS);` = The application of the log function to the variable of interest, this case the Variable is “SYS”, plug your variable of interest into the function.

Squaring a Variable (variable^2)

To square a variable use “**2;” after the variable name.

Here is an example code: `SYS_sq = SYS**2;`

- `SYS_sq` = The name of the new variable you are creating that has the squared function applied to it. This can be named whatever you want. It is recommended that the format for the new variable name created from this function simply be “variable_sq” to keep things clear in your analyses.
- `=SYS**2;` = Is telling SAS to square the variable of interest. In this case the Variable is “SYS”, plug your variable of interest into the function.

Square Root of a Variable (sqrt(variable))

To square root a variable use the “sqrt();” statement

Here is an example code: sqrt`_SYS = sqrt(SYS);`

- `sqrt_SYS` = The name of the new variable you are creating that has the square rootfunction applied to it. This can be named whatever you want. It is recommended that the format for the new variable name created from this function simply be “sqrt_variable” to keep things clear in your analyses.
- `=sqrt(SYS);` = Is telling SAS to square root the variable of interest. In this case the Variable is “SYS”, plug your variable of interest into the function.

LOGISTIC REGRESSION PROCEDURES (PROC LOGISTIC)

Proc Logistic

The proc logistic function is the essential function used to run all logistic regression functions. The outcome will always be dichotomous values and there are several outputs. Not all output numbers will be discussed in this manual but selected ones will and those are the most important numbers to look at.

Single Regression Model

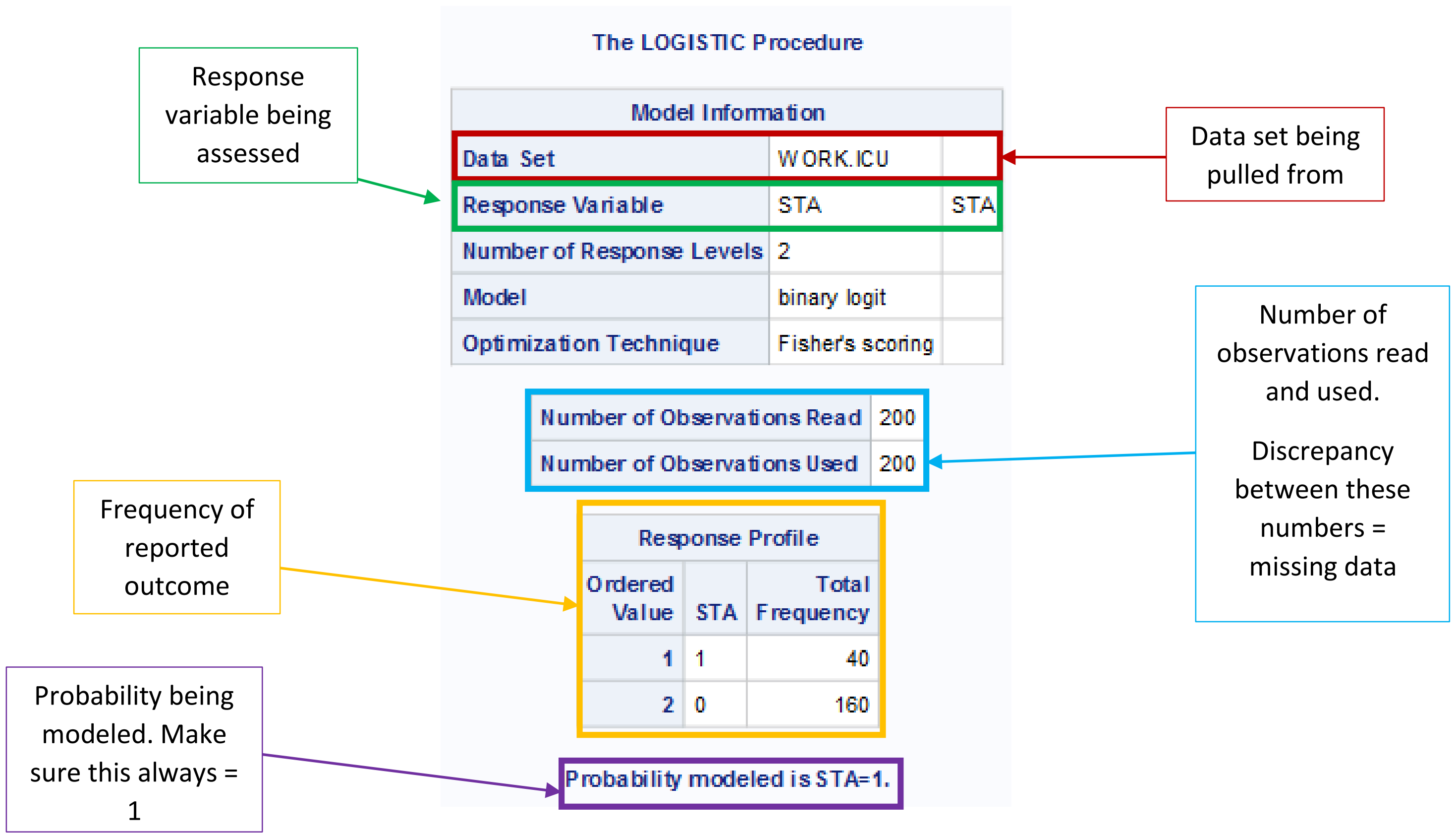
The single regression Model assumes there is one dichotomous outcome and 1 single exposure/predictor variable.

Example Single Regression model code:

```
proc logistic descending data=icu;
model sta=age;
run;
```

- **`proc logistic`** = Telling SAS to run a logistic regression model.
- `descending` = Telling SAS to order the data with the outcome of yes (1) on top. This makes the data easier to read, and along with standard epidemiologic standards of data interpretation. Include this statement in ALL proc logistic commands in which 1 is equivalent to having the outcome.
- `data=icu;` = Which data set you would like the proc logistic function to report on, and which dataset the variables are in.
- `model sta=age;` = Indicates what the logistic regression model. In this instance the outcome is “sta”. The outcome ALWAYS goes to the left of the equal sign. In this example, “Age” is the exposure/predictor of sta, therefore it is to the right of the equal sign.
- **`run;`** = run command

Figure 8.1: Output example of Single Logistic Regression with selected explanations



Response variable being assessed
Data set being pulled from
Number of observations read and used.
Discrepancy between these numbers = missing data
Frequency of reported outcome
Probability being modeled. Make sure this always = 1

The LOGISTIC Procedure

Model Information		
Data Set	WORK.ICU	
Response Variable	STA	STA
Number of Response Levels	2	
Model	binary logit	
Optimization Technique	Fisher's scoring	

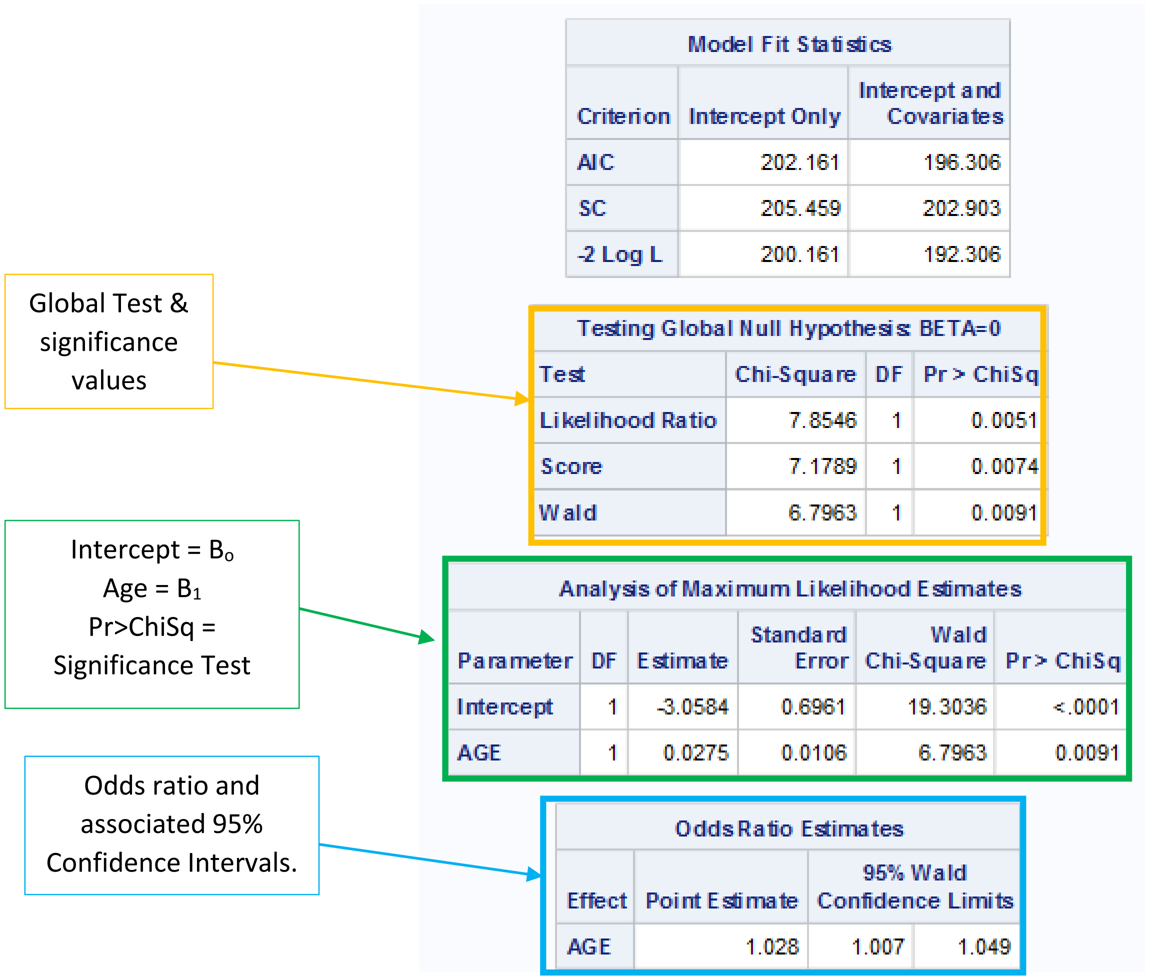
Number of Observations Read	200
Number of Observations Used	200

Response Profile

Ordered Value	STA	Total Frequency
1	1	40
2	0	160

Probability modeled is STA=1.

Figure 8.1: Output example of Single Logistic Regression with selected explanations Continued


Global Test & significance values
Intercept = B_o
Age = B_1
Pr>ChiSq = Significance Test
Odds ratio and associated 95% Confidence Intervals.

Model Fit Statistics

Criterion	Intercept Only	Intercept and Covariates
AIC	202.161	196.306
SC	205.459	202.903
-2 Log L	200.161	192.306

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	7.8546	1	0.0051
Score	7.1789	1	0.0074
Wald	6.7963	1	0.0091

Analysis of Maximum Likelihood Estimates

Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	-3.0584	0.6961	19.3036	<.0001
AGE	1	0.0275	0.0106	6.7963	0.0091

Odds Ratio Estimates

Effect	Point Estimate	95% Wald Confidence Limits	
AGE	1.028	1.007	1.049

Multiple Regression Model

All output are similar to that of the single regression model except that you get more coefficients and Odds ratio estimates. The only difference in the multiple regression model is the addition of multiple exposure/predictor variables.

Example Multiple Regression model code:
```
proc logistic descending data=icu;
model sta=age SYS;
run;
```

- `proc logistic` = Telling SAS to run a logistic regression model.
- `descending` = Telling SAS to order the data with the outcome of yes (1) on top. This makes the data easier to read, and along with standard epidemiologic standards of data interpretation. Include this statement in ALL proc logistic commands in which 1 is equivalent to having the outcome.

- `data=icu;` = Which data set you would like the proc logistic function to report on, and which dataset the variables are in.
- `model sta=age RACE SYS;` = Indicates what the logistic regression model. In this instance the outcome is "sta". The outcome ALWAYS goes to the left of the equal sign. In this example, "Age" & "SYS" are all the exposure/predictor of sta, therefore it is to the right of the equal sign separated only by a space.
- `run;` = run command

Multiplicative Interaction

Assessing multiplicative interaction does not add to much to our proc logistic function. Essentially it is just telling SAS to run the regression procedure to measure an outcome for a variable GIVEN the level of another variable. We do this by simply incorporating an asterisk (*) in between the 2 variables we would like to see the interaction in.

Here is an example of a code incorporating multiplicative interaction:

```
proc logistic descending data=myopia;
model myopic=gender spheq gender*spheq;
run;
```

- `proc logistic` = Telling SAS to run a logistic regression model.
- `descending` = Telling SAS to order the data with the outcome of yes (1) on top. This makes the data easier to read, and along with standard epidemiologic standards of data interpretation. Include this statement in ALL proc logistic commands in which 1 is equivalent to having the outcome.
- `data=myopia;` = Which data set you would like the proc logistic function to report on, and which dataset the variables are in.
- `model myopic=gender spheq` = Indicates what the logistic regression model. In this instance the outcome is "myopic". The outcome ALWAYS goes to the left of the equal sign. In this example, "gender" & "spheq" are all the exposure/predictor of sta, therefore it is to the right of the equal sign separated only by a space.
- `gender*spheq;` = Indicates that you would also like SAS to run the interaction. NOTICE the "*" in between the 2 variables. This means you want to see the interaction between 2 variables. Here we are assessing the variable gender, given the level of the variable spheq on the outcome myopic.
- `run;` = run command

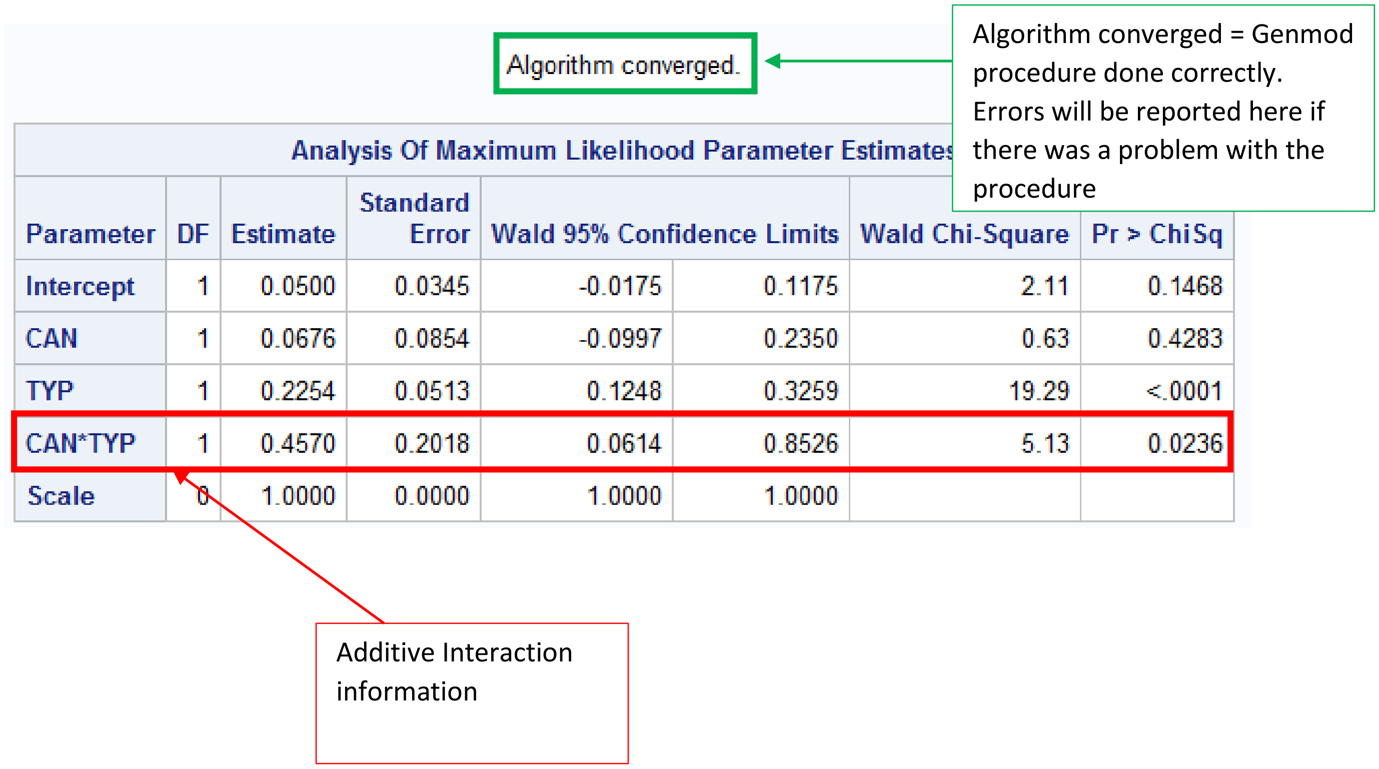
Additive Interaction (Proc Genmod)

SAS automatically assesses for multiplicative interaction. Therefore, we need to create a different statement apart from "Proc Logistic" so that it creates a different algorithm than is used by default. Therefore, instead of Proc logistic, we use a statement called "Proc Genmod" to create the correct algorithm.

```
proc genmod descending data=ICU_altered;
model STA = CAN TYP CAN*TYP/dist=bin link = identity;
output out=pdat p=phat;
run;
```

- `proc genmod` = Telling SAS to run a logistic regression model with an additive interaction.
- `descending` = Telling SAS to order the data with the outcome of yes (1) on top. This makes the data easier to read, and along with standard epidemiologic standards of data interpretation. Include this statement in ALL proc genmod commands in which 1 is equivalent to having the outcome.
- `data=ICU_altered;` = Which data set you would like the proc genmod function to report on, and which dataset the variables are in.
- `model STA = CAN TYP` = Indicates what the logistic regression model. In this instance the outcome is "STA". The outcome ALWAYS goes to the left of the equal sign. In this example, "CAN" & "sTYP" are all the exposure/predictor of sta, therefore it is to the right of the equal sign separated only by a space.
- `CAN*TYP;` = Indicates that you would also like SAS to run the interaction. NOTICE the "*" in between the 2 variables. This means you want to see the interaction between 2 variables. Here we are assessing the variable CAN, given the level of the variable TYP on the outcome STA.
- `/dist=bin link = identity;` = an essential piece to the genmod statement. Include this in all genmod statements.
- `output out=pdat p=phat;` = Here you are outputting the pihats from the procedure. This is necessary when running the genmod statement because if you have any pihats <0 or >1 then you can NOT trust the results. More explanation is of this is below in the "view your pihats!" subsection.
- `run;` = run command

Figure 9.1: Proc Genmod output with selected explanations



Algorithm converged.

Algorithm converged = Genmod procedure done correctly. Errors will be reported here if there was a problem with the procedure

Analysis Of Maximum Likelihood Parameter Estimates

Parameter	DF	Estimate	Standard Error	Wald 95% Confidence Limits		Wald Chi-Square	Pr > ChiSq
Intercept	1	0.0500	0.0345	-0.0175	0.1175	2.11	0.1468
CAN	1	0.0676	0.0854	-0.0997	0.2350	0.63	0.4283
TYP	1	0.2254	0.0513	0.1248	0.3259	19.29	<.0001
CAN*TYP	1	0.4570	0.2018	0.0614	0.8526	5.13	0.0236
Scale	0	1.0000	0.0000	1.0000	1.0000		

Additive Interaction information

View your pihats!

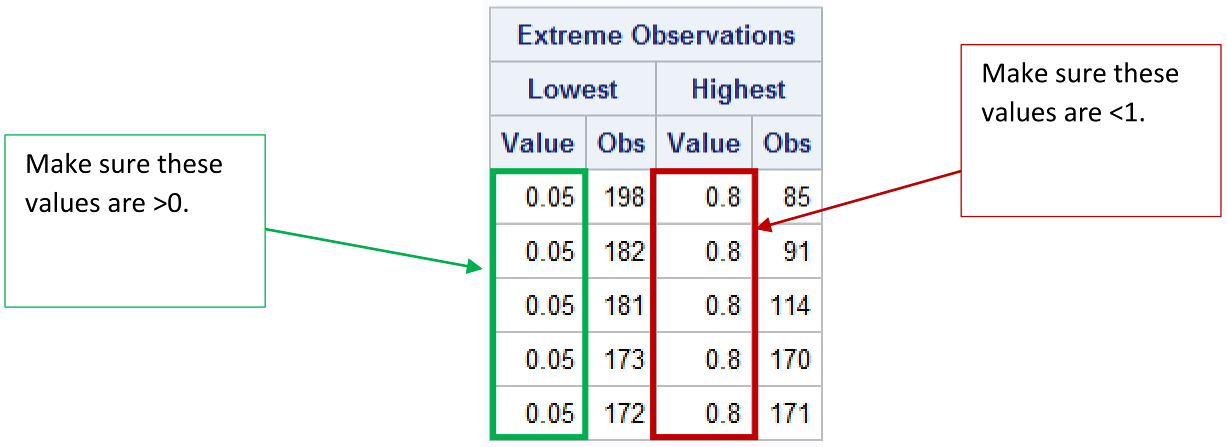
Recall that we told SAS to create a new file and output our phats into that file using the “output” function. This is essential for proc genmod because you need to view extreme pihat values to determine if your test can be trusted. If you have unrealistic pihats, then you can not trust the output. To test this, simply do a proc univariate of the outputted data you created from earlier.

```
proc univariate data=pdat;
var phat;
run;
```

- `proc univariate` = Command function to give you descriptive statistics.
- `data=pdat;` = Telling SAS that the data we want is named pdat (NOTE: we created this with out output statement in the proc genmod procedure)
- `var phat;` = Telling SAS to output the information only for the variable “phat”. Again, we told the proc genmod statement to output the variables in the previous section. Therefore, these will be your phats.
- `run;` = run command

Once you have run the proc univariate procedure, the ONLY output you need to look at is your extreme observations. You need to make sure that the extreme values fall between 0 and 1.

Figure 9.2: pihat Extreme observations with selected explanations.



Extreme Observations			
Lowest		Highest	
Value	Obs	Value	Obs
0.05	198	0.8	85
0.05	182	0.8	91
0.05	181	0.8	114
0.05	173	0.8	170
0.05	172	0.8	171

Make sure these values are >0.
Make sure these values are <1.

Design Variables Through Proc Logistic

Recall that we learned to create earlier in the **CATEGORIZING & MANIPULATING VARIABLES** section of this manual. SAS Proc logistic also has the ability to create design variables itself.

Here is some an example code:

```
proc logistic descending data=glow500;
class raterisk/param=ref ref=first;
model fracture=raterisk;
run;
```

- `proc logistic` = Telling SAS to run a logistic regression model.
- `descending` = Telling SAS to order the data with the outcome of yes (1) on top. This makes the data easier to read, and along with standard epidemiologic standards of data

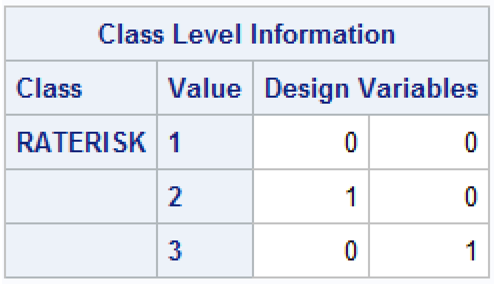
interpretation. Include this statement in ALL proc logistic commands in which 1 is equivalent to having the outcome.

- `data=glow500;` = Which data set you would like the proc logistic function to report on, and which dataset the variables are in.
- `class raterisk/` = Indicated that you would like SAS to classify the variable numbers in the Variable “raterisk”
- `param=ref` = Telling SAS that one of the categories is a reference category.
- `ref=first;` = Telling SAS that the first category is the reference category
- `model fracture=raterisk;` = Indicating the outcome of “fracture” and predictor/exposure of “raterisk”.
- `run;` = run command
- **NOTE: SAS will automatically create all the design variables needed based on how many levels there are (i.e. if you have three levels in the variable, then as will create 2 design variables and use the first one as a reference category)**

Once the You run the statement successfully there are a couple of things to check to make sure it was done correctly as shown in the figures below.

First, you need to look at the “Class level information. This is SAS telling you how it created the design variables, you will notice it looks exactly like the design variable table you were taight to create earlier in the “Creating design variables” section

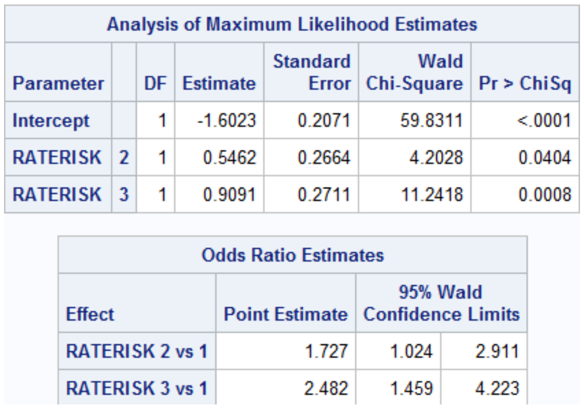
Figure 9.3: SAS created design variables.



Class Level Information			
Class	Value	Design Variables	
RATERISK	1	0	0
	2	1	0
	3	0	1

Next, it is important to look at the Estimates and OR’s of the SAS created design variable.

Figure 9.4: Likelihood Estimates & OR’s of SAS created design Variables.



Analysis of Maximum Likelihood Estimates

Parameter		DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept		1	-1.6023	0.2071	59.8311	<.0001
RATERISK	2	1	0.5462	0.2664	4.2028	0.0404
RATERISK	3	1	0.9091	0.2711	11.2418	0.0008

Odds Ratio Estimates

Effect	Point Estimate	95% Wald Confidence Limits	
RATERISK 2 vs 1	1.727	1.024	2.911
RATERISK 3 vs 1	2.482	1.459	4.223

SAS created design variables in multivariate models

Incorporating SAS created design variables into a multivariate model is exactly the same as the normal procedure. Just make sure you have the “class” statement included in the line above the model statement.

Here is some an example code:

```
proc logistic descending data=glow500;
class raterisk/param=ref ref=first;
model fracture= age weight raterisk priorfrac;
run;
```

REFER TO “DESIGN VARIABLES THROUGH PROC LOGISTIC” FOR EXPLANATIONS OF OUTPUT AND CODE. THE ONLY DIFFERENCE IN THIS MODEL IS THERE ARE MULTIPLE RISK FACTORS INVOLVED. INTERPRET IS AS A MULTIVARIATE MODEL.

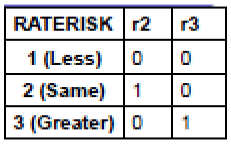
Contrast Statements

To Create a Contrast statement in SAS, you need to incorporate it AFTER your proc logistic statement and BEFORE Your run statement. Contrast statements are used for multiple things such as polychotomous variables (categorical variables with more than 2 levels)

For Polychotomous Variables

When variables have more than two possible outcomes, we need to use contrast stamemtns to compare the multiple levels. In tis example, we use RATERISK which has 3 potential choices and have already been design variable coded.

Figure 10.1: Design variable coding for RATERISK variable



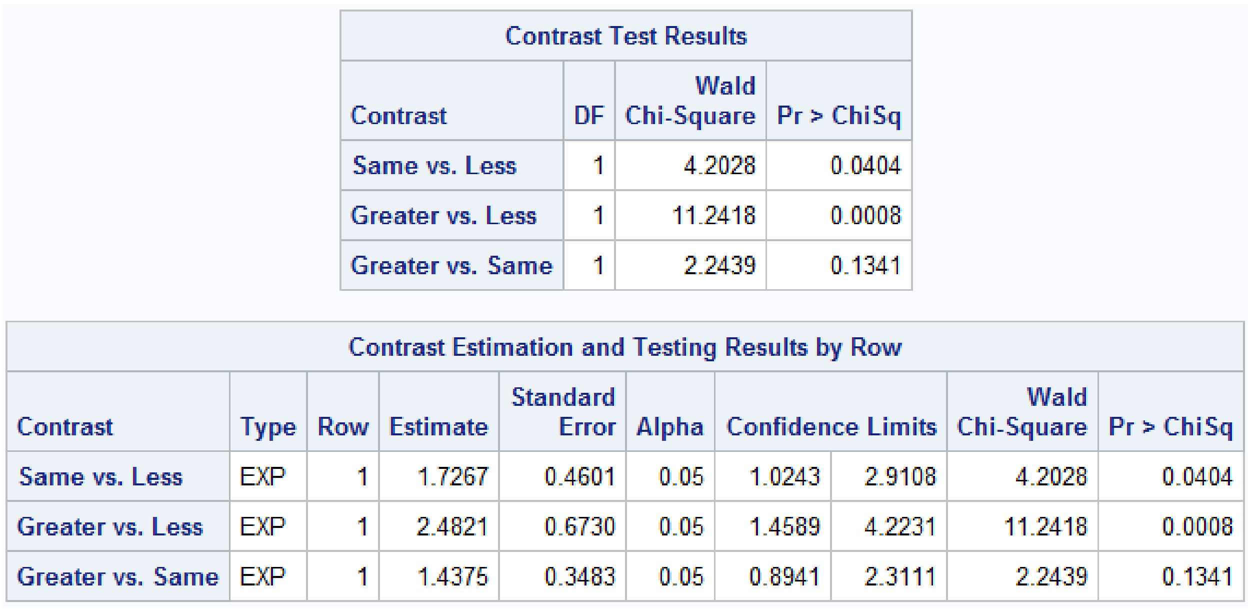
RATERISK	r2	r3
1 (Less)	0	0
2 (Same)	1	0
3 (Greater)	0	1

Here is an example of a proc logistic statement with a contrast statement incorporated:

```
proc logistic descending data=glow500;
class raterisk/param=ref ref=first;
model fracture=raterisk;

contrast 'Same vs. Less' raterisk 1 0/estimate=exp;
contrast 'Greater vs. Less' raterisk 0 1/estimate=exp;
contrast 'Greater vs. Same' raterisk -1 1/estimate=exp;

run;
```

- `Contrast` = Telling SAS to begin the contrast statement.
- `'Same vs. Less'`= What you are naming this portion of the contrast.
- `Raterisk` = Name of the variable you are contrasting
- `1 0` = How you would like the 2 to be compared. i.e the distance between the 2.
- `/estimate=exp;` = an essential piece to the contrast statement. Include this in all contrast statements.

Figure 10.2: Example output from contrast statement.



Contrast Test Results			
Contrast	DF	Wald Chi-Square	Pr > ChiSq
Same vs. Less	1	4.2028	0.0404
Greater vs. Less	1	11.2418	0.0008
Greater vs. Same	1	2.2439	0.1341

Contrast Estimation and Testing Results by Row									
Contrast	Type	Row	Estimate	Standard Error	Alpha	Confidence Limits		Wald Chi-Square	Pr > ChiSq
Same vs. Less	EXP	1	1.7267	0.4601	0.05	1.0243	2.9108	4.2028	0.0404
Greater vs. Less	EXP	1	2.4821	0.6730	0.05	1.4589	4.2231	11.2418	0.0008
Greater vs. Same	EXP	1	1.4375	0.3483	0.05	0.8941	2.3111	2.2439	0.1341

For Continuous Variables

If you have a continuous variable, you can incorporate a simple contrast statement within your logistic procedure to tell it to give you estimates based on a certain increase.

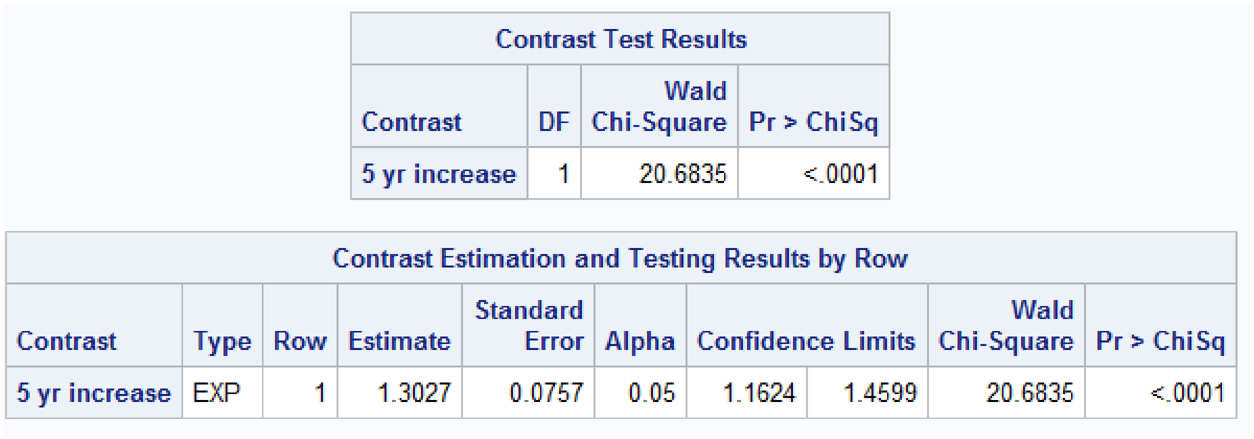
For this example, we will see how much a 5 year increase in age effects the outcome of fracture.

Here is the example code:

```
proc logistic descending data=glow500;
model fracture=age;
contrast '5 yr increase' age 5/estimate=exp;
run;
```

- `Contrast` = Telling SAS to begin the contrast statement.
- `'5 yr increase'` = Naming the contrast
- `age 5` = Telling SAS what you think a meaningful change in the continuous variable is.
- `/estimate=exp;` = an essential piece to the contrast statement. Include this in all contrast statements.

Figure 11.1: Example of output incorporating the contrast statement for a continuous variable.



Contrast Test Results			
Contrast	DF	Wald Chi-Square	Pr > ChiSq
5 yr increase	1	20.6835	<.0001

Contrast Estimation and Testing Results by Row									
Contrast	Type	Row	Estimate	Standard Error	Alpha	Confidence Limits		Wald Chi-Square	Pr > ChiSq
5 yr increase	EXP	1	1.3027	0.0757	0.05	1.1624	1.4599	20.6835	<.0001

For Interaction

Often times, when we see interaction, it is necessary to report multiple Odds Ratios, because, well think about it, a variable has a different effect on an outcome on different levels, so we need to get differing OR's depending on what level is being affected because the effect is clearly not linear.

Example contrast statement with Interaction:

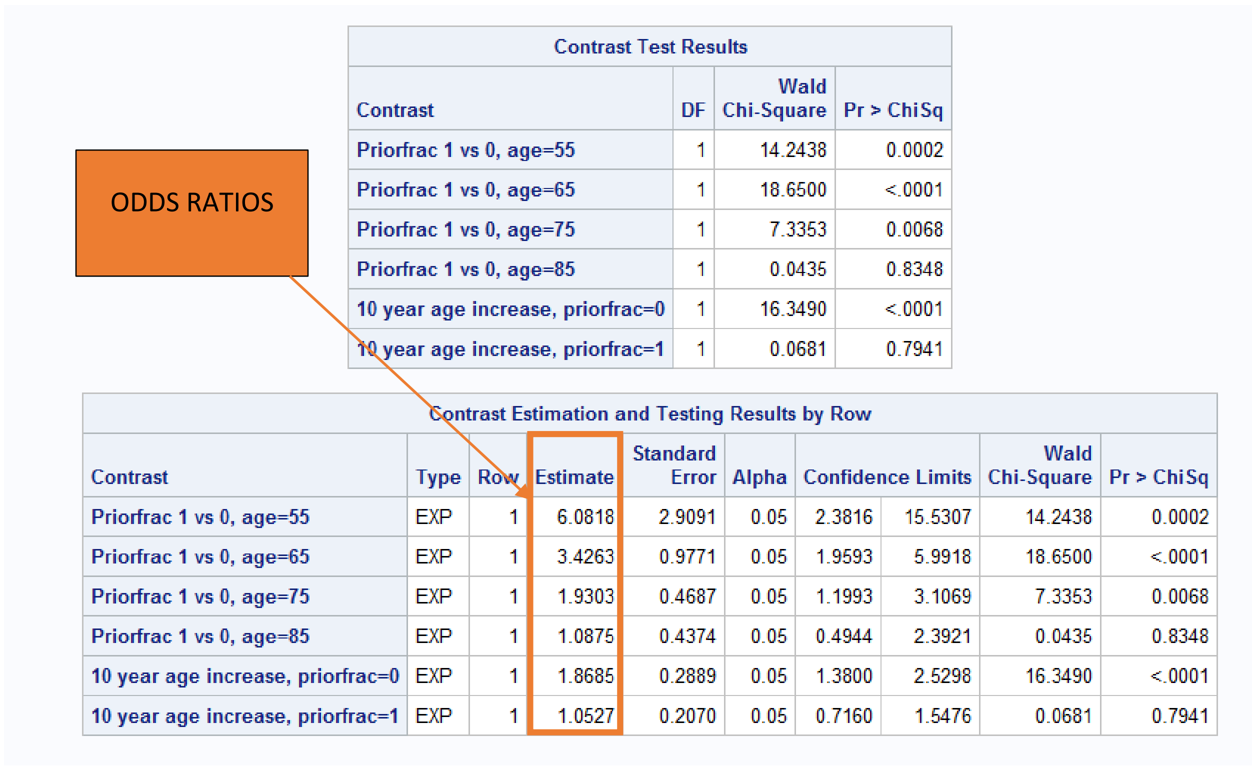
```
proc logistic descending data=glow500;
model fracture=priorfrac age priorfrac*age;

contrast 'Priorfrac 1 vs 0, age=55'
priorfrac 1 age 0 priorfrac*age 55 /estimate=exp;
contrast 'Priorfrac 1 vs 0, age=65'
priorfrac 1 age 0 priorfrac*age 65/estimate=exp;
contrast 'Priorfrac 1 vs 0, age=75'
priorfrac 1 age 0 priorfrac*age 75/estimate=exp;
contrast 'Priorfrac 1 vs 0, age=85'
priorfrac 1 age 0 priorfrac*age 85/estimate=exp;

contrast '10 year age increase, priorfrac=0'
priorfrac 0 age 10 priorfrac*age 0 /estimate=exp;
contrast '10 year age increase, priorfrac=1'
priorfrac 0 age 10 priorfrac*age 10 /estimate=exp;
run;
```

- `Contrast` = Telling SAS to begin the contrast statement.
- `'Priorfrac 1 vs 0, age=55'` = Telling SAS what you want to name the contrast statement
- `Priorfrac 1` = Telling SAS to Set Prior Frac to 1. i.e "IN the presence of Prior Frac"
- `age 0`= Telling SAS to set age at 0. i.e "and with age set to zero"
- `priorfrac*age 55` = Looking at the age of interaction of priorfrac and age, when age is set at 55.

- `/estimate=exp;` = an essential piece to the contrast statement. Include this in all contrast statements.

-
  ```
  contrast '10 year age increase, priorfrac=0'
  priorfrac 0 age 10 priorfrac*age 0 /estimate=exp;
  ```
  - Depending how you code it, you can also just look at how much the age incorporates a constant age increase, as noted in the statement above.

Figure 12.1: Example Output incorporating Contrast statements for at different levels of age and a 10 year increase.



Contrast Test Results

Contrast	DF	Wald Chi-Square	Pr > ChiSq
Priorfrac 1 vs 0, age=55	1	14.2438	0.0002
Priorfrac 1 vs 0, age=65	1	18.6500	<.0001
Priorfrac 1 vs 0, age=75	1	7.3353	0.0068
Priorfrac 1 vs 0, age=85	1	0.0435	0.8348
10 year age increase, priorfrac=0	1	16.3490	<.0001
10 year age increase, priorfrac=1	1	0.0681	0.7941

Contrast Estimation and Testing Results by Row

Contrast	Type	Row	Estimate	Standard Error	Alpha	Confidence Limits		Wald Chi-Square	Pr > ChiSq
Priorfrac 1 vs 0, age=55	EXP	1	6.0818	2.9091	0.05	2.3816	15.5307	14.2438	0.0002
Priorfrac 1 vs 0, age=65	EXP	1	3.4263	0.9771	0.05	1.9593	5.9918	18.6500	<.0001
Priorfrac 1 vs 0, age=75	EXP	1	1.9303	0.4687	0.05	1.1993	3.1069	7.3353	0.0068
Priorfrac 1 vs 0, age=85	EXP	1	1.0875	0.4374	0.05	0.4944	2.3921	0.0435	0.8348
10 year age increase, priorfrac=0	EXP	1	1.8685	0.2889	0.05	1.3800	2.5298	16.3490	<.0001
10 year age increase, priorfrac=1	EXP	1	1.0527	0.2070	0.05	0.7160	1.5476	0.0681	0.7941

4-Row Table Creation

Assuming you have dichotomous variables, you can create a 4-row table using contrast statements (EXACTLY the same as those in your epidemiology class) that incorporate the variables acting independently as compared to the interaction of the variables. This is important in assessing both multiplicative and additive interaction.

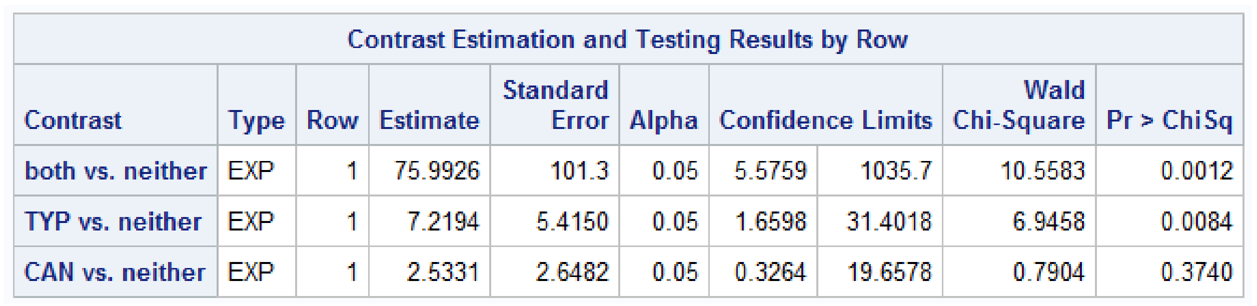
```
proc logistic descending data=ICU_altered;
model STA= CAN TYP CAN*TYP;

contrast 'both vs. neither' CAN 1 TYP 1 CAN*TYP 1/estimate=exp;
contrast 'TYP vs. neither' CAN 0 TYP 1 CAN*TYP 0/estimate=exp;
contrast 'CAN vs. neither' CAN 1 TYP 0 CAN*TYP 0/estimate=exp;
run;
```

- `Contrast` = Telling SAS to begin the contrast statement.
- `'both vs. neither'`= Telling SAS what you want to name the contrast statement

- `CAN 1 TYP 1 CAN*TYP 1` = Comparing The Presence of CAN, TYP and The Interaction of the 2 to your reference category (i.e both v neither). This is the top of your 4-row table.
- `CAN 0 TYP 1 CAN*TYP 0` = Comparing The Presence of ONLY TYP (i.e TYP v neither) to your reference category. This is the 2nd row of your 4-row table.
- `CAN 1 TYP 0 CAN*TYP 0` = Comparing The Presence of ONLY CAN (i.e CAN v neither) to your reference category. This is the 3rd row of your 4-row table.
- `/estimate=exp;` = an essential piece to the contrast statement. Include this in all contrast statements.

Once run, your 4-row table will appear in the output, **NOTE: You will NOT see the reference category, BUT you know the OR is equal to 1.00 so there is no point of showing it!**

Figure 13.1: Creation of a 4-row table through contrast statements.



Contrast Estimation and Testing Results by Row

Contrast	Type	Row	Estimate	Standard Error	Alpha	Confidence Limits		Wald Chi-Square	Pr > ChiSq
both vs. neither	EXP	1	75.9926	101.3	0.05	5.5759	1035.7	10.5583	0.0012
TYP vs. neither	EXP	1	7.2194	5.4150	0.05	1.6598	31.4018	6.9458	0.0084
CAN vs. neither	EXP	1	2.5331	2.6482	0.05	0.3264	19.6578	0.7904	0.3740

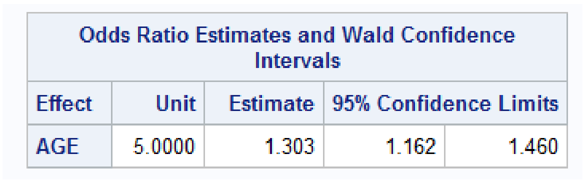
Units Statements

The Units statement is similar to the contrast statement, but can ONLY be used with continuous variables. Therefore, when using a units statement, make sure you are truly aware of your data.

```
proc logistic descending data=glow500;
model fracture=age/clodds=Wald;
units age=5;
run;
```

- `/clodds=Wald;` = Telling SAS togive you the odds Ratio and corresponding Confidence intervals of the incorporated Units statement. (It also spits out a nice graph)
- `Units` = Telling SAS To begin the Units function within this regression model
- `age=5;` = Telling SAS to incorporate a 5 year increase of age into the model.
    - **NOTE: You can add multiple UNITS statements in the same function by just adding what units you would like to see the increase by. The output will give you multiple Odds Ratio Estimates.**

Figure 14.1: Output of Odds ratio with Unit Statement.



Odds Ratio Estimates and Wald Confidence Intervals

Effect	Unit	Estimate	95% Confidence Limits	
AGE	5.0000	1.303	1.162	1.460

OUTPUTTING RESULTS & MERGING DATASETS

Output Results into a New Dataset (Output Out=)

Outputting data is used to create a variable based on the analyses you ran, it can be used with any of the proc functions (i.e proc univariate, proc means, proc logistic etc.). Essentially it is taking the statistical procedure and listing it into its own variable within a dataset into a NEW dataset you create.

```
proc logistic descending data=chdage;
model chd=age;
output out=pdat p=pihat;
run;
```

Regular proc function of your choosing.

- `output out=pdat` = Telling SAS what the new dataset the output will be put into.
- `p=` = Telling SAS what part of the output you want (in this case the pihats (p=pihat))
- `pihat;` = Telling SAS what we want the new variable to be named within our new data set. We are naming it our new variable “pihat” in this scenario.

Next, you will want to proc print your new dataset with to ensure that it was properly created. Simply proc print the new dataset to check. IF done successfully, you will see the new variable created within the new dataset.

Merging Datasets (merge)

Merging datasets is important; especially for later when we begin overlaying graphs in the “Creating Graphs” section. Generally, we merge outputted values (see above in “Output Results into a new dataset”) into one dataset so we can plot them on the same graph. Everytime you merge 2 datasets, you are creating another dataset with both of them in it.

```
data plotdat;
merge pdat sta_means;
by a;
run;
```

- `data plotdat;` = Formation of a new dataset. This new dataset is being called “plotdat”
- `merge` = Beginning the merge function. Telling SAS you will be merging 2 or more datasets together into the new dataset (In this case the new dataset will be named “plotdat”)
- `pdat sta_means;` = The datasets being merged together. The two datasets are pdat and sta_means that are being merged.
- `by a;` = you want the datasets to be organized by the variable “a” (they will be be organized in that fashion)

To check to see if this function was ran properly, use proc print on the new dataset that was created and make sure the datasets were merged appropriately.

Creating Graphs

Universal Graph Statement

The Universal Graph statement is a special code that helps to create better looking Graphs through SAS (Using the default SAS graph does not look so nice and may be hard to read). This Universal Graph statement is relatively standard and can be used in most practices, slight adjustments can be made, but for the most part this is a relatively good graphing statement for Logistic regression purposes.

Just an FYI: The SAS support website has a wealth of information about great Universal graph statements that you can refer to for any and they look pretty nice!
https://support.sas.com/sassamples/graphgallery/PROC_GPLOT.html

Below is an example of the code for the Universal graph statement. This code will always go BEFORE The actual plotting of your graphs (proc gplot) and will NOT have a run; statement at the end of it. The run; statement will be used when you actually run your gplot (next section). Once you have ran this code ONCE, it is good for the rest of your SAS session… SAS automatically assumes that this is the format you want.

```
axis1 minor=none label=(f=swiss h=2.5 'AGE');
axis2 minor=none label=(f=swiss h=2.5 a=90 'STA');
goptions FTEXT=swissb HTEXT=2.0 HSIZE=6 in
VSIZE=6 in;
symbol1 c=black v=dot;
symbol2 c=black v=circle;
symbol3 c=black v=star h=2;
```

- `axis1` = Defining the formatting of Axis1
- `minor=none` = Telling SAS NOT to include minor tick marks on axis 1 of the graph.
- `label=` = Telling SAS the formatting of the label for Axis 1
- `(f=swiss` = Defining the font style for the graph (swiss is a standard font style)
- `h=2.5` = Defining the font size (2.5 is a standard font size)
- `'AGE');` = Defining what the Axis 1 variable is called (It will display Axis 1 with the label "AGE" in this example).
- `axis2` = Defining the formatting of Axis2
- `a=90` = Telling SAS to rotate this Axis label 90 degrees. Axis 2 should always be the definer for you Y-axis. Thereofore, we want to rotate in 90 degrees so it fits nicely on the graph we are creating.
- `goptions FTEXT=swissb` = Telling SAS that the numbers on the Axes need to be swissb font (This is a standard font that looks nice).
- `HTEXT=2.0` = Essential function, keep it in your Universal Graph procedure
- `HSIZE=6 in` = The size of the Horizontal text size should be 6 (6 is a standard size)
- `VSIZE=6 in;` = The size of the Vertical text size should be 6 (6 is a standard size)
- `symbol1` = Creating a symbol for when you plot points. This symbol will be named "symbol1" and will be necessary for the gplot statement
- `c=black` = This is a color statement. Telling SAS That the color you want symbol1 to be is black.
- `v=dot;` = Defining what kind of symbol you want to be associated with Symbol1. Whenever Symbol1 is plotted, it will appear as a dot.

- `Symbol2` = Creating a symbol for when you plot points. This symbol will be named "symbol2" and will be necessary for the gplot statement
- `v=circle;` = Defining what kind of symbol you want to be associated with Symbol2. Whenever Symbol1 is plotted, it will appear as a circle.
- `Symbol3` = Creating a symbol for when you plot points. This symbol will be named "symbol3" and will be necessary for the gplot statement
- `v=star h=2;` = Defining what kind of symbol you want to be associated with Symbol3. Whenever Symbol1 is plotted, it will appear as a star. For stars, they are by default to small, so we add a height statement of "h=2" So they can be seen easier (This is standard practice.

Additions to Universal Graph Statement

Please Note: You are NOT limited to the above Universal Graph statement, you are free to add and remove, and change certain code associated with the universal Graph statement.

For example, if you add, `i=join;` after the `v=circle` of your symbol creation statements `(so the code would be symbol1 c=black v=dot i=join;)` statement, it will join together all of your points when you plot it on the gplot function in the next section.

Here is a link with more explanations of the symbol functions: http://support.sas.com/documentation/cdl/en/graphref/63022/HTML/default/viewer.htm#annodict-symbol.htm

Plotting the Graph (proc gplot)

The Proc gplot function is your main way to plot graphs. It should ALWAYS be inputted AFTER your Universal Graph Statement (refer to above section labeled "The Universal Graph Statement). If you don't, SAS is likely to output a graph that will be harder to read.

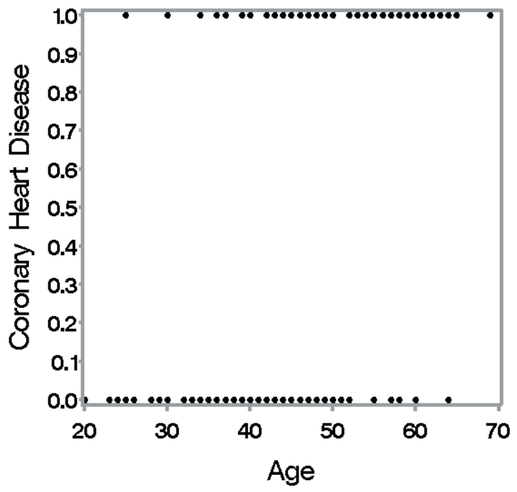
UNIVERSAL GRAPH STATEMENT

```
axis1 minor=none label=(f=swiss h=2.5 'AGE');
axis2 minor=none label=(f=swiss h=2.5 a=90 'STA');
goptions FTEXT=swissb HTEXT=2.0 HSIZE=6 in
VSIZE=6 in;
symbol1 c=black v=dot;
symbol2 c=black v=circle;
symbol3 c=black v=star h=2;

proc gplot data=chdage;
plot chd*age/haxis=axis1 vaxis=axis2;
run; quit;
```

- **`proc gplot`** = run the graph plotting function
- `data=chdage;` = Telling SAS that what dataset we are working with (in this case the dataset is chdage)
- `plot chd*age` = Telling SAS which variables from the dataset we want to plot. In this case we are plotting chd by age.
    - **NOTE: the variable to the left of the asterisk is always the y-axis (This should be your OUTCOME variable).**

- `/haxis=axis1 vaxis=axis2;` = Referring to your Universal Graph statement. It is telling Sas that the Horizontal axis should take the attributes of AXIS1 and the vertical axis should take the attributes of AXIS2.
- `run;` = Run Command
- `quit;` = Vital code to put at the end of the proc gplot procedure, Without quit; function SAS will keep re-looping the code and may shut down.
- **NOTE:** gplot will attach the symbol statements and other attributes from your Universal Graph Statement on its own.

Figure 15.1: Graph created from Proc Gplot procedure



Overlaying Multiple Procedures

In addition just plotting scatterplot of the exposure and outcome, you can also overlay values in between such as pihats and so forth. To overlay a function, simply add it into the proc gplot statement right after your "plot" function. **NOTE:** All functions are the same as the above proc gplot section, except for the added pihat variable, which is noted below.

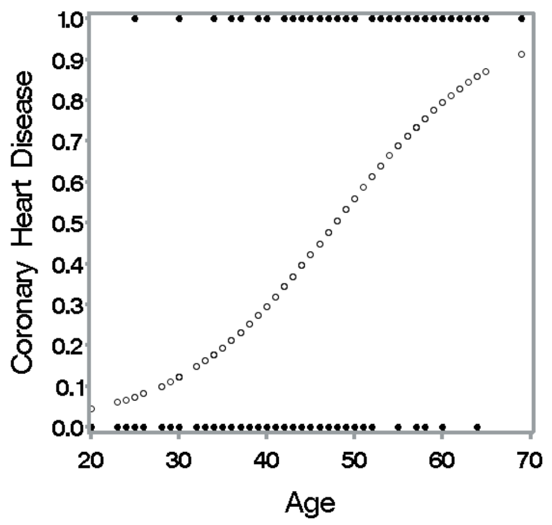
```
axis1 minor=none label=(f=swiss h=2.5 'AGE');
axis2 minor=none label=(f=swiss h=2.5 a=90 'STA');
goptions FTEXT=swissb HTEXT=2.0 HSIZE=6 in
VSIZE=6 in;
symbol1 c=black v=dot;
symbol2 c=black v=circle;
symbol3 c=black v=star h=2;

proc gplot data=pdat;
plot (chd pihat)*age/overlay haxis=axis1 vaxis=axis2;
run; quit;
```

UNIVERSAL GRAPH STATEMENT

Added pihat variable from a datset containg it (often times created using the merge and output functions)

NOTE: Multiple Variables can be added and overlaid into the graph such as midpoints, etc. Assuming they are in the dataset you are pulling from. Just note, you need to have as many symbols as variables being plotted. (Notice how we haven't even used the stars symbol yet, therefore we have enough symbol statements).

Figure 15.2: Graph created proc gplot procedure with pihat data points overlaid

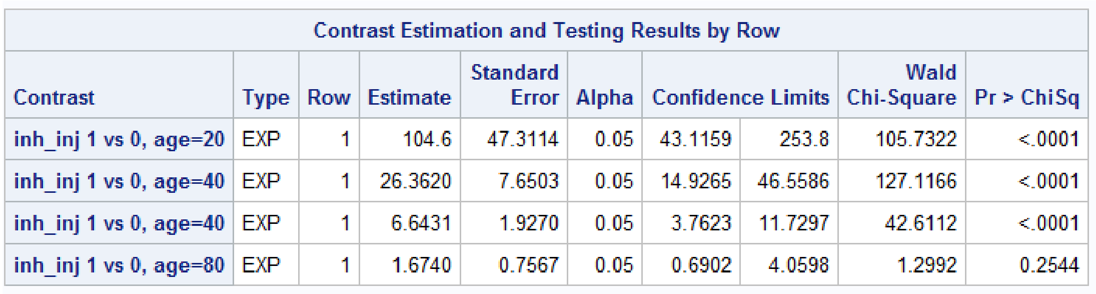


The "Cards" Statement

The Cards statement is to be used BEFORE the Universal Graph statement & proc gplot procedure. This statement essentially creates a new dataset specifically meant for graphin variables. You use this statement when you want to manually plot specific points on a graph using the gplot statement. Often times you need it to plot multiple Odds Ratios for specific groups when you see an interaction. Furthermore, it allows you to lot Confidence intervals along with their corresponding OR's in the Graph.

For example, if we get an interaction and use our contrast statements, we may get a table such as this:

Figure 15.3: Example output of Contrast statements we want to plot on a graph.



Contrast Estimation and Testing Results by Row

Contrast	Type	Row	Estimate	Standard Error	Alpha	Confidence Limits		Wald Chi-Square	Pr > ChiSq
inh_inj 1 vs 0, age=20	EXP	1	104.6	47.3114	0.05	43.1159	253.8	105.7322	<.0001
inh_inj 1 vs 0, age=40	EXP	1	26.3620	7.6503	0.05	14.9265	46.5586	127.1166	<.0001
inh_inj 1 vs 0, age=40	EXP	1	6.6431	1.9270	0.05	3.7623	11.7297	42.6112	<.0001
inh_inj 1 vs 0, age=80	EXP	1	1.6740	0.7567	0.05	0.6902	4.0598	1.2992	0.2544

To do this, we create a cards statement which looks like so:

```
data OR_CL; input age OR CIL CIU;
cards;
20 104.6 43.1159 253.8
40 26.320 14.9265 46.5586
60 6.6431 3.7623 11.7297
80 1.6740 0.6902 4.0598
run;
```

- `data OR_CL;` = The new dataset we are creating to put the numbers we want into.

- `input age OR CIL CIU;` = The Variables we want to be added to the newdata set. Age, OR, CIL, & CIU are the new variables we are adding into the dataset.
    - **NOTE:** The order you set these in is VERY important.
- `cards;` = setting up to manually enter the data in the appropriate cells. All numbers after this statement are entered very specifically into places until the run; statement is read.
- `20` = corresponds to the 1st input variable (age) and will be the first cell of that variable.
- `104.6` = corresponds to the 2nd input variable (OR) and will be the first cell of that variable.
- `43.1159` = corresponds to the 3rd input variable (CIL) and will be the first cell of that variable.
- `253.8`= corresponds to the 4th input variable (CIU) and will be the first cell of that variable.
- `40` = corresponds to the 1st input variable (age) and will be the 2nd cell of that variable.
- *Repeat this until all of the variables and cells are filled to your liking.*
- `run;` = Run Command

Be sure to Proc Print the newly created datset to ensure it was performed correctly.

In combination with the Universal Graph Statement and the proc gplot, the entire coding for this statement will look as so. Take note to the order of the code:

```
data OR_CL; input age OR CIL CIU;
cards;
20 104.6 43.1159 253.8
40 26.320 14.9265 46.5586
60 6.6431 3.7623 11.7297
80 1.6740 0.6902 4.0598
run;
```

Cards statement First.

```
proc print data = OR_CL; run;
```

Proc Print to check the data 2nd

```
axis1 minor=none label=(f=swiss h=2.5 'AGE');
axis2 minor=none label=(f=swiss h=2.5 a=90 'Odds Ratio');
goptions FTEXT=swissb HTEXT=2.0 HSIZE=6 in
VSIZE=6 in;
symbol1 c=black v=dot i=join;
symbol2 c=black v=circle i=join;
symbol3 c=black v=star h=2 i=join;
```

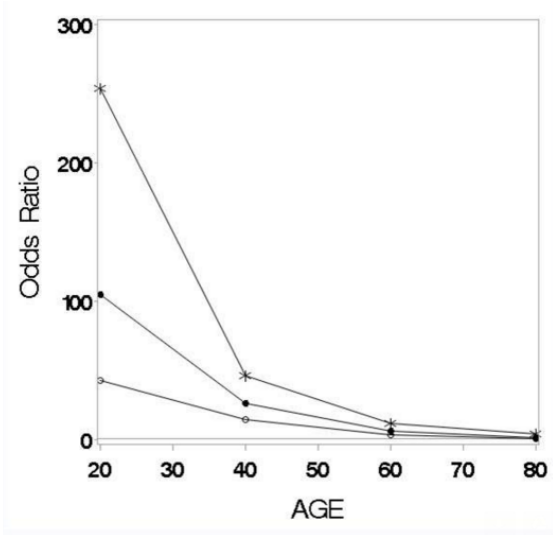
Universal Graph Statement 3rd

```
proc gplot data=OR_CL;
plot (OR CIL CIU)*age/overlay haxis=axis1 vaxis=axis2 vref=1;
run; quit;
```

Proc gplot statement last

If done correctly, the graph will appear as so:

Figure 15.4: Proc gplot using Cards statement



SCALE ASSESSMENT

Splines

Splines is an important feature in logistic regression that helps to assess which line best fits with your outcome. It is extremely important in displaying interactions and can help to display your data effectively. The splines feature gives you a few ways of graphing out your data. This is separated into 3 distinct splines features. The first is Constant Connection, the next is the cubic connection. Using these methods, you can view how your data looks under each condition and potentially find out how the data looks under different mathematical procedures. You should always compare the three with eachother to see which best assesses your logistic procedure.

A FEW NOTES REGARDING THE SPLINES PROCEDURE:

1.) **THE SPLINES PROCEDURE IS ALWAYS RAN WITHIN THE PROC LOGISTIC FUNCTION**
2.) **THE GRAPHS CREATED HERE ARE INDEPENDENT FROM THE PROC GPLOT FUNCTION**
3.) **All 3 codes are exactly the same EXCEPT for the changing of the "degree" statement**
    a. **Constant Connection = "**`degree=0`**"**
    b. **Linear Connection = "**`degree=1`**"**
    c. **Cubic Connection = "**`naturalcubic`**"**

Constant Connection

The constant connection does NOT attempt to connect your data in any smooth fashion. It literally connects the points with no smoothing at all. Often times (Unless your data is a straight line, or a lot of data points) the constant connection method will be very jumpy.

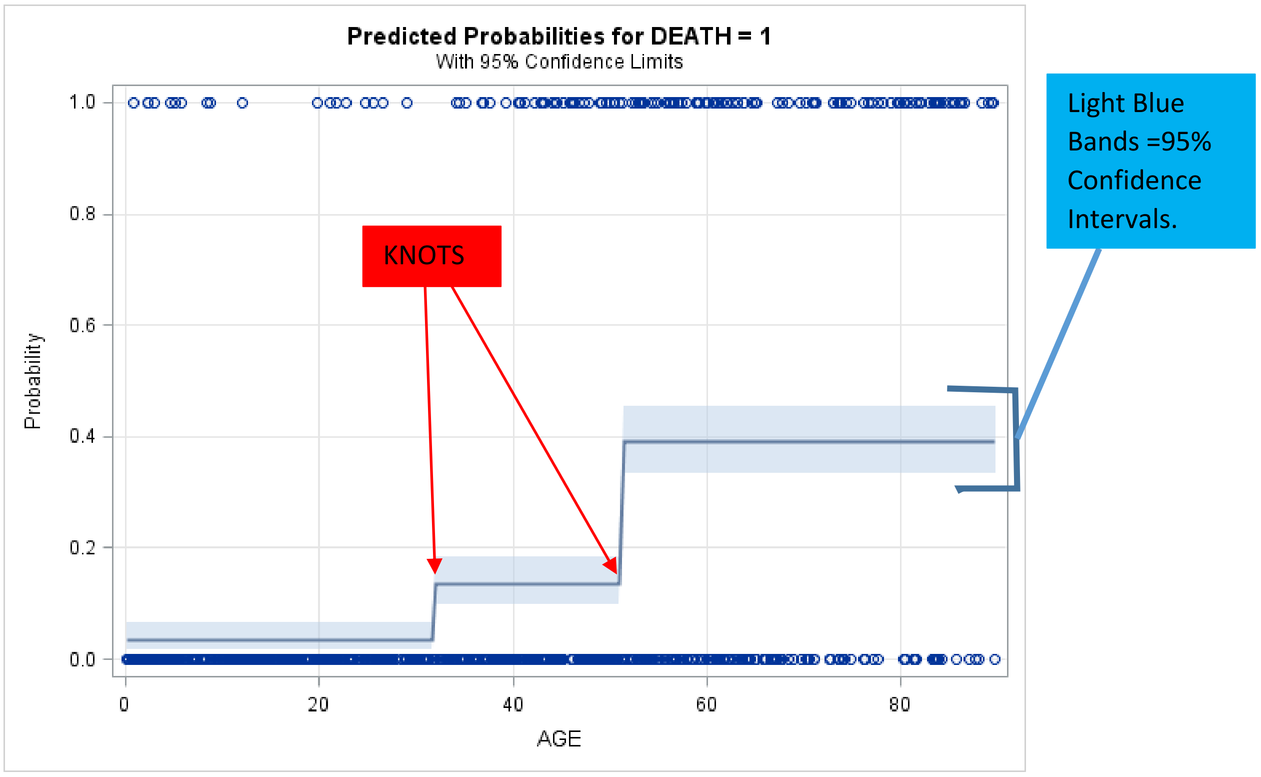
```
proc logistic descending data=burn1000;
effect AGEs=spline(AGE/knotmethod=list(10.8 31.95 51.25)
basis=tpf(noint) degree=0);
model DEATH=AGEs;
effectplot;
run;
```

- `effect` = Telling SAS that you be effecting a certain variable. The text that comes after this is the effect statement and the effect is the `spline` statement to be seen later.
- `AGEs` = Here you are creating a new variable. "AGEs" indicates that AGE is the variable and the s at the end indicates that it has been splined. To help yourself out, always name this variable "VARIABLEs". In other words, the variable you are working with, with a lowercase s at the end.
- `Spline` = Indicating you are beginning the spline function on a variable of interest.
- `AGE`= Indicating the variable that is being splined. This the variable of interest. Put any risk factor variable here.
- `/knotmethod=list` = Means that you want to input knots into your statement. **KNOTS** are connection points. The graph that is outputted will make sure to connect at all the points you indicate. MORE knots =more flexibility in your graph.
- `(10.8 31.95 51.25)` = Where you want your knots to be placed. In this instance, we used the 1st quartile, median, and 3rd quartile as knot points. But these are whatever are

important to you. Common methods for determining knots = quartiles, percentiles, meaningful units and even just even spacing. This is completely arbitrary and where you want them placed. Normally there are 3-5 knot statements.

- `basis=tpf(noint)` = An essential function for the splines statement. ALWAYS leave this in.
- `degree=0);` = a degree=0 statement will always indicate that you are telling SAS that you want this to be a **CONSTANT CONNECTION.** Always put degree=0 here if you want to see the constant connection of your data.
- `model DEATH=AGEs;` = Here we are indicating our model to be the predictor of AGEs (Our newly created splined variable) on the outcome of death. Be sure and make sure this is the newly created variable or else the function will not work appropriately.
- `effectplot;` = This statement tells SAS to plot the function we just created above. It will give you the logistic model as well as give you what the data looks like graphed out along with the knots that you created.
- `run;` = Run Command.

Figure 16.1: Graph Outputted from splines function using constant connection with selected descriptions



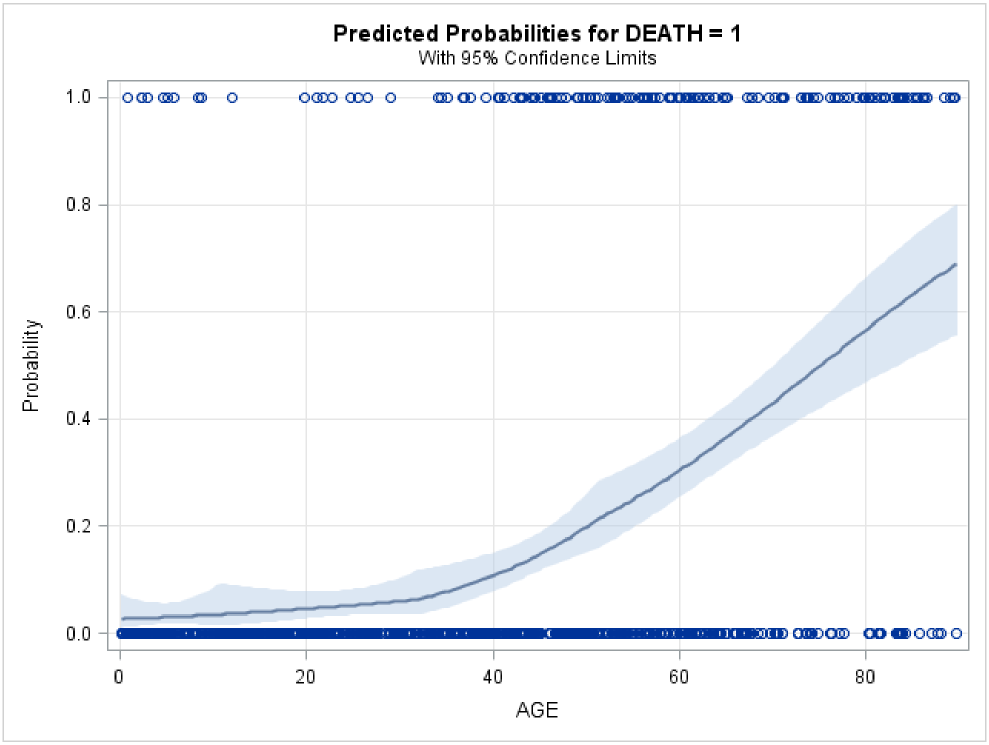
Linear Connection

The Linear Connection makes more of an attempt to connect the smoothness of the Graph. You will find that when you use this procedure, it will make an attempt to make the line as straight as possible and will try to show a trend in one direction as much as possible.

```
proc logistic descending data=burn1000;
effect AGEs=spline(AGE/knotmethod=list(10.8 31.95 51.25)
basis=tpf(noint) degree=1);
model DEATH=AGEs;
effectplot;
run;
```

- `effect` = Telling SAS that you be effecting a certain variable. The text that comes after this is the effect statement and the effect is the `spline` statement to be seen later.
- `AGEs` = Here you are creating a new variable. “AGEs” indicates that AGE is the variable and the s at the end indicates that it has been splined. To help yourself out, always name this variable “VARIABLEs”. In other words, the variable you are working with, with a lowercase s at the end.
- `Spline` = Indicating you are beginning the spline function on a variable of interest.
- `AGE`= Indicating the variable that is being splined. This the variable of interest. Put any risk factor variable here.
- `/knotmethod=list` = Means that you want to input knots into your statement. **KNOTS** are connection points. The graph that is outputted will make sure to connect at all the points you indicate. MORE knots =more flexibility in your graph.
- `(10.8 31.95 51.25)` = Where you want your knots to be placed. In this instance, we used the 1st quartile, median, and 3rd quartile as knot points. But these are whatever are important to you. Common methods for determining knots = quartiles, percentiles, meaningful units and even just even spacing. This is completely arbitrary and where you want them placed. Normally there are 3-5 knot statements.
- `basis=tpf(noint)` = An essential function for the splines statement. ALWAYS leave this in.
- `degree=1);` = a degree=1 statement will always indicate that you are telling SAS that you want this to be a **LINEAR CONNECTION.** Always put degree=1 here if you want to see the linear connection of your data.
- `model DEATH=AGEs;` = Here we are indicating our model to be the predictor of AGEs (Our newly created splined variable) on the outcome of death. Be sure and make sure this is the newly created variable or else the function will not work appropriately.
- `effectplot;` = This statement tells SAS to plot the function we just created above. It will give you the logistic model as well as give you what the data looks like graphed out along with the knots that you created.
- `run;` = Run Command.

Figure 16.2: Outputted Graph from Splines connection using Linear Connection



Cubic Connection

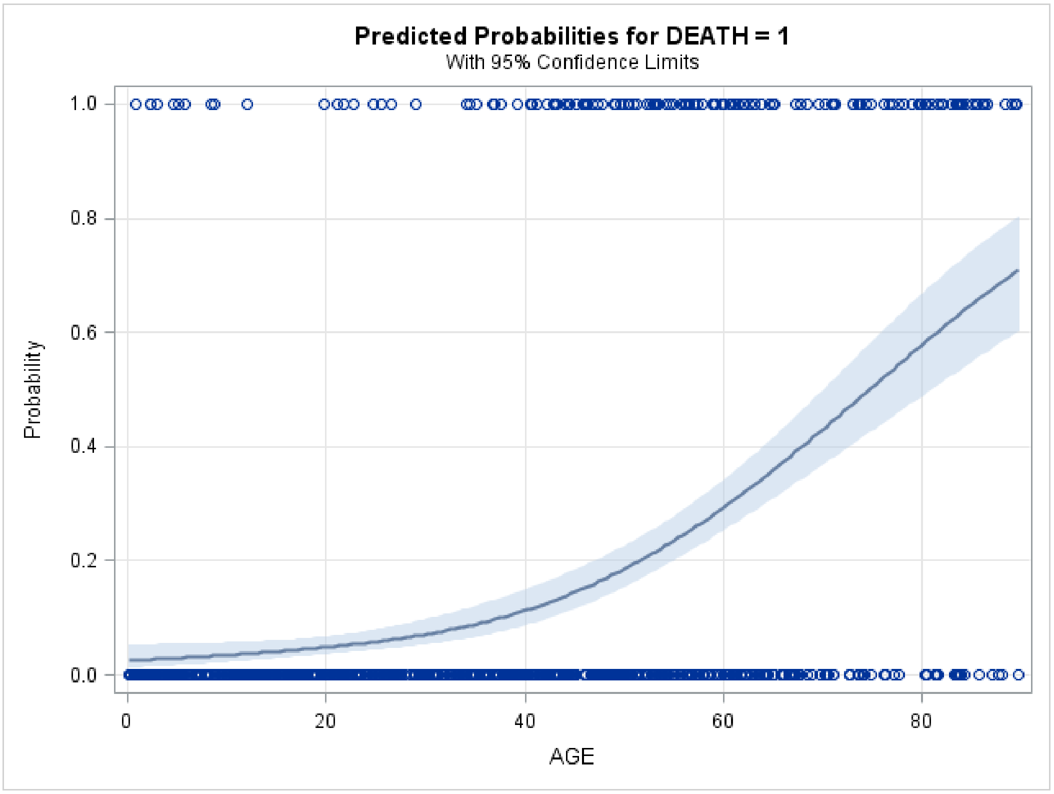
The Cubic connection allows for the most freedom out of the 3 spline methods. Essentially, it is has the most bend and is more likely to fit correctly plot correctly to your data if it is full of weird bends and turns.

```
proc logistic descending data=burn1000;
effect AGEs=spline(AGE/knotmethod=list(10.8 31.95 51.25)
basis=tpf(noint) naturalcubic);
model DEATH=AGEs;
effectplot;
run;
```

- `effect` = Telling SAS that you will be effecting a certain variable. The text that comes after this is the effect statement and the effect is the `spline` statement to be seen later.
- `AGEs` = Here you are creating a new variable. "AGEs" indicates that AGE is the variable and the s at the end indicates that it has been splined. To help yourself out, always name this variable "VARIABLEs". In other words, the variable you are working with, with a lowercase s at the end.
- `Spline` = Indicating you are beginning the spline function on a variable of interest.
- `AGE`= Indicating the variable that is being splined. This the variable of interest. Put any risk factor variable here.

- `/knotmethod=list` = Means that you want to input knots into your statement. **KNOTS** are connection points. The graph that is outputted will make sure to connect at all the points you indicate. MORE knots =more flexibility in your graph.
- `(10.8 31.95 51.25)` = Where you want your knots to be placed. In this instance, we used the 1st quartile, median, and 3rd quartile as knot points. But these are whatever are important to you. Common methods for determining knots = quartiles, percentiles, meaningful units and even just even spacing. This is completely arbitrary and where you want them placed. Normally there are 3-5 knot statements.
- `basis=tpf(noint)` = An essential function for the splines statement. ALWAYS leave this in.
- `naturalcubic);` = A `naturalcubic` statement will always indicate that you are telling SAS that you want this to be a **CUBIC CONNECTION.** Always put `naturalcubic` here if you want to see the cubic connection of your data.
- `model DEATH=AGEs;` = Here we are indicating our model to be the predictor of AGEs (Our newly created splined variable) on the outcome of death. Be sure and make sure this is the newly created variable or else the function will not work appropriately.
- `effectplot;` = This statement tells SAS to plot the function we just created above. It will give you the logistic model as well as give you what the data looks like graphed out along with the knots that you created.
- `run;` = Run Command.

Figure 16.2: Outputted Graph from Splines connection using Cubic Connection



Fractional Polynomial (fp) Procedure

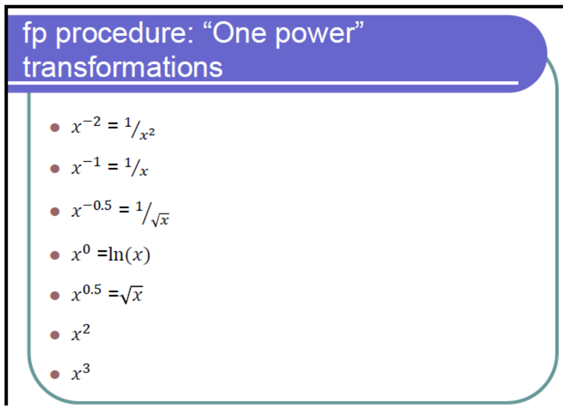
The Fractional Polynomial (fp) procedure is a method to model a continuous variable and can assess many types of different scales at once. This should be conducted after you have looked at all of the separate splines procedures. The code for this procedure is extremely long and will NOT be explained in detail. Instead, we will start off by: 1.) explaining some pertinent concepts associated with the code, 2.) then show the code (known as the fp macro) and 3.) last we will preview the small output associated with this code and explain how to interpret the output created from the fp macro.

Pertinent fp Procedure concepts (1 & 2 power transformations)

The main point of the fp procedure is to automatically assess if a transformed procedure is the best method for modeling your continuous covariate. For example, IF the outcome *of death to car crash* on predictor *age* is low in young ages, high in young adult hood, low in middle adulthood, and high again in older adult hood, then a linear function is not likely to map onto this correctly. Therefore, we use the fp method to automatically assess which transformation would work best with certain outcomes. There are 1 and 2 power transformations. The point of knowing these is to know which transformation to use once you learn how to use the fp macro.

ONE POWER TRANSFORMATIONS:

16.3: List of potential one power transformations.

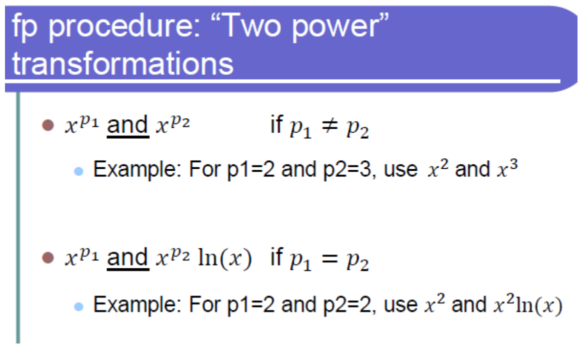


fp procedure: "One power" transformations

- $x^{-2} = {}^{1}/_{x^2}$
- $x^{-1} = {}^{1}/_{x}$
- $x^{-0.5} = {}^{1}/_{\sqrt{x}}$
- $x^{0} = \ln(x)$
- $x^{0.5} = \sqrt{x}$
- x^{2}
- x^{3}

TWO POWER TRANSFORMATIONS:

Figure 16.3: Example of how two Power transformations are used.



fp procedure: "Two power" transformations

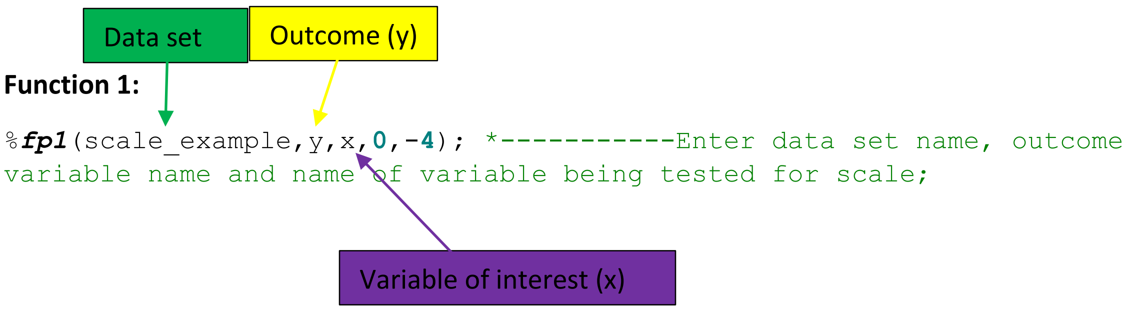
- x^{p_1} and x^{p_2} if $p_1 \neq p_2$
  - Example: For p1=2 and p2=3, use x^2 and x^3
- x^{p_1} and $x^{p_2}\ln(x)$ if $p_1 = p_2$
  - Example: For p1=2 and p2=2, use x^2 and $x^2\ln(x)$

In summary, 2 power transformations are the combination of 2 1 power transformations. (i.e x^2 & x^3)

fp Macro explanation

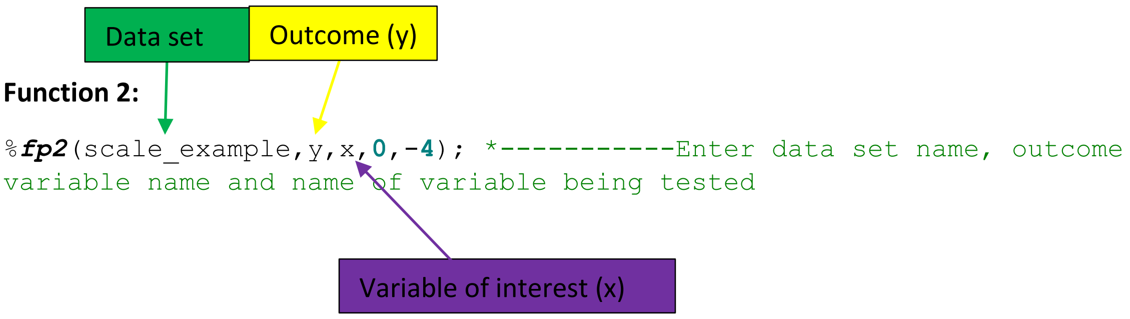
Below you will find an extremely long code. The creator of this Macro statement was Dr. Bachand, if you have any specific questions regarding its use of function, ask her directly.

The only important thing you need to know about this procedure is Where to place you **Dataset name, Outcome, & your continuous variable of interest**. There will be statements noted with asterisks telling you where to place these items. The main thing we will be concentrated on is the output statement it creates. There are only **2 places** that you need to enter this information in. I have sparsed out the 2 parts you need to worry about. Again, I emphasize do **NOT** get held up in this code there are only 2 important functions to worry about.



Function 1:

```
%fp1(scale_example,y,x,0,-4); *-----------Enter data set name, outcome
variable name and name of variable being tested for scale;
```



Function 2:

```
%fp2(scale_example,y,x,0,-4); *-----------Enter data set name, outcome
variable name and name of variable being tested
```

OKAY, below is the FULL fp macro. Simply copy it into your SAS code (yes, all 3 pages of it) BELOW your data step and fill on those variables to get the best transformed functions. NEXT, we will learn how to interpret the output.

PLEASE NOTE: The fp Macro may take a long time to complete (anytime between 1 minute – 20 minutes depending on your computer specs)

fp MACRO for Univariate Analyses

```
*** ADD A DATA STEP ***;


*___________________________________________________________________*;
*                                                                   *;
*                     FP METHOD                                     *;
*___________________________________________________________________*;

** Macro for fp assessment **;

%macro fp1(dset,y,var,lb,p1);
 %do %until(&p1=7);
 %put ***** &p1 *****;
 ODS output FitStatistics = mfs;
 data fpdat; set &dset; if &var>&lb; pc=&p1/2;
    if pc ne 0 then F1=&var**pc; else if pc = 0 then F1=log(&var);
 run;
 proc logistic descending data=fpdat;

   model &y=F1; *-------------------F1 represents the variable being tested
for scale;

 run;
 data mfs; set mfs; if criterion='-2 Log L'; drop Criterion InterceptOnly;
run;
 proc append data=mfs base=tres; run;
 proc datasets; delete fpdat mfs; run;
 quit;
 %let p1=%eval(&p1+1);
 %end;
%mend fp1;


%fp1(scale_example,y,x,0,-4); *-----------Enter data set name, outcome
variable name and name of variable being tested for scale;



data pvals; do p1=-4 to 6; output; end; run;
data pvals; set pvals; p1=p1/2; run;
data tres; merge pvals tres; if p1 in (-1.5, 1.5, 2.5) then delete; run;
proc sort data=tres; by InterceptAndCovariates; run;
data tres; set tres; if _N_=1 or p1=1; run;


%macro fp2(dset,y,var,lb,p1,p2);
 %do %until(&p1=7);
  %do %until(&p2=7);
 %put ***** &p1 &p2 *****;
 ODS output FitStatistics = mfs;
 data fpdat; set &dset; if &var>&lb; pc1=&p1/2; pc2=&p2/2;
    if pc1 ne 0 then F1=&var**pc1; else if pc1 = 0 then F1=log(&var);
    if pc1 ne pc2 then do; if pc2 ne 0 then F2=&var**pc2;
                       else if pc2 = 0 then F2=log(&var); end;
    if pc1=pc2 then F2=F1*log(&var);
 run;

 proc logistic descending data=fpdat;

   model &y=F1 F2; *------------F1 and F2 represent the variable being tested
for scale;

 run;
 data mfs; set mfs; if criterion='-2 Log L'; drop Criterion InterceptOnly;
run;
 proc append data=mfs base=tres2; run;
 proc datasets; delete fpdat mfs; run;
 quit;
  %let p2=%eval(&p2+1);
  %end;
 %let p2=%eval(-4);
 %let p1=%eval(&p1+1);
 %end;
%mend fp2;

%fp2(scale_example,y,x,0,-4,-4); *-----------Enter data set name, outcome
variable name and name of variable being tested for scale;



data pvals2; do p1=-4 to 6; do p2=-4 to 6; output;end; end; run;
data pvals2; set pvals2; p1=p1/2; p2=p2/2; run;
data tres2; merge pvals2 tres2;
 if p1 in (-1.5, 1.5, 2.5) or p2 in (-1.5, 1.5, 2.5) then delete; run;
proc sort data=tres2; by InterceptAndCovariates; run;
data tres2; set tres2; if _N_=1; run;



data comb; set tres tres2; run;

data c1; set comb; if p1=1 and p2=.; rename
InterceptAndCovariates=Dev_linear;
                                     drop p1 p2; run;
data c2; set comb; if p1 ne 1 and p2=.; rename
InterceptAndCovariates=Dev_fp1;
                                        rename p1=e_fp1; drop p2; run;
data c3; set comb; if p2 ne .; rename InterceptAndCovariates=Dev_fp2;
                              rename p1=e1_fp2; rename p2=e2_fp2; run;

data c;
  merge c1 c2 c3;
  diff_lin_fp1=Dev_linear-Dev_fp1;
  diff_lin_fp2=Dev_linear-Dev_fp2;
  diff_fp1_fp2=Dev_fp1-Dev_fp2;

  p_lin_fp1=1-probchi(diff_lin_fp1,1);
  p_lin_fp2=1-probchi(diff_lin_fp2,3);
  p_fp1_fp2=1-probchi(diff_fp1_fp2,2);
run;

proc print noobs data=c;
```

```
  var Dev_linear e_fp1 Dev_fp1 e1_fp2 e2_fp2 Dev_fp2 p_lin_fp1 p_lin_fp2
p_fp1_fp2;
  format p_lin_fp1 p_lin_fp2 p_fp1_fp2 6.4;
run;

proc datasets; delete tres tres2 pvals pvals2 comb c c1 c2 c3; run; quit;

* End macro for fp assessment *;
```

fp MACRO for Multivariate Analyses

In order to run the fp macro for multivariate analyses, it is very similar to the univariate sense, but you need to add all other predictor variables to two parts of the fp macro, which are labeled below.

```
* Multivariate scale assessment, height *;
*___________________________________________________________________*;
*                                                                   *;
*                      FP METHOD                                    *;
*___________________________________________________________________*;

** Macro for fp assessment **;

%macro fp1(dset,y,var,lb,p1);
 %do %until(&p1=7);
 %put ***** &p1 *****;
 ODS output FitStatistics = mfs;
 data fpdat; set &dset; if &var>&lb; pc=&p1/2;
    if pc ne 0 then F1=&var**pc; else if pc = 0 then F1=log(&var);
 run;
 proc logistic descending data=fpdat;

   model &y=F1 SER CPR PO2 AGE; *--------------------F1 represents the
variable being tested for scale (SYS);

 run;
 data mfs; set mfs; if criterion='-2 Log L'; drop Criterion InterceptOnly;
run;
 proc append data=mfs base=tres; run;
 proc datasets; delete fpdat mfs; run;
 quit;
 %let p1=%eval(&p1+1);
 %end;
%mend fp1;


%fp1(ICU_altered,STA,SYS,0,-4); *-----------Enter data set name, outcome
variable name and name of variable being tested for scale;
```

Enter the model main effects here (1st instance)

```
data pvals; do p1=-4 to 6; output; end; run;
data pvals; set pvals; p1=p1/2; run;
data tres; merge pvals tres; if p1 in (-1.5, 1.5, 2.5) then delete; run;
proc sort data=tres; by InterceptAndCovariates; run;
data tres; set tres; if _N_=1 or p1=1; run;


%macro fp2(dset,y,var,lb,p1,p2);
 %do %until(&p1=7);
  %do %until(&p2=7);
 %put ***** &p1 &p2 *****;
 ODS output FitStatistics = mfs;
 data fpdat; set &dset; if &var>&lb; pc1=&p1/2; pc2=&p2/2;
    if pc1 ne 0 then F1=&var**pc1; else if pc1 = 0 then F1=log(&var);
    if pc1 ne pc2 then do; if pc2 ne 0 then F2=&var**pc2;
                           else if pc2 = 0 then F2=log(&var); end;
    if pc1=pc2 then F2=F1*log(&var);
 run;

 proc logistic descending data=fpdat;

   model &y=F1 F2 SER CPR PO2 AGE; *------------F1 and F2 represent the
variable being tested for scale (height);

 run;
 data mfs; set mfs; if criterion='-2 Log L'; drop Criterion InterceptOnly;
run;
 proc append data=mfs base=tres2; run;
 proc datasets; delete fpdat mfs; run;
 quit;
  %let p2=%eval(&p2+1);
  %end;
 %let p2=%eval(-4);
 %let p1=%eval(&p1+1);
 %end;
%mend fp2;

%fp2(ICU_altered,STA,SYS,0,-4,-4); *-----------Enter data set name, outcome
variable name and name of variable being tested for scale;



data pvals2; do p1=-4 to 6; do p2=-4 to 6; output;end; end; run;
data pvals2; set pvals2; p1=p1/2; p2=p2/2; run;
data tres2; merge pvals2 tres2;
 if p1 in (-1.5, 1.5, 2.5) or p2 in (-1.5, 1.5, 2.5) then delete; run;
proc sort data=tres2; by InterceptAndCovariates; run;
data tres2; set tres2; if _N_=1; run;



data comb; set tres tres2; run;

data c1; set comb; if p1=1 and p2=.; rename
InterceptAndCovariates=Dev_linear;
                                    drop p1 p2; run;
```

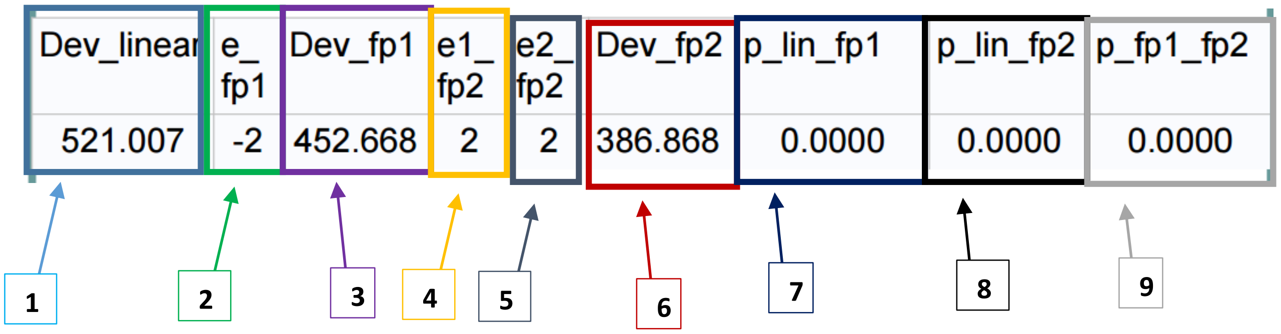
Enter the model main effects here (2^{nd} instance)

```
data c2; set comb; if p1 ne 1 and p2=.; rename
InterceptAndCovariates=Dev_fp1;
                                        rename p1=e_fp1; drop p2; run;
data c3; set comb; if p2 ne .; rename InterceptAndCovariates=Dev_fp2;
                               rename p1=e1_fp2; rename p2=e2_fp2; run;

data c;
  merge c1 c2 c3;
  diff_lin_fp1=Dev_linear-Dev_fp1;
  diff_lin_fp2=Dev_linear-Dev_fp2;
  diff_fp1_fp2=Dev_fp1-Dev_fp2;

  p_lin_fp1=1-probchi(diff_lin_fp1,1);
  p_lin_fp2=1-probchi(diff_lin_fp2,3);
  p_fp1_fp2=1-probchi(diff_fp1_fp2,2);
run;

proc print noobs data=c;
  var Dev_linear e_fp1 Dev_fp1 e1_fp2 e2_fp2 Dev_fp2 p_lin_fp1 p_lin_fp2
p_fp1_fp2;
  format p_lin_fp1 p_lin_fp2 p_fp1_fp2 6.4;
run;

proc datasets; delete tres tres2 pvals pvals2 comb c c1 c2 c3; run; quit;

* End macro for fp assessment *;
```

Interpreting the fp Macro output

If run successfully, the macro will output a lot of results. **THERE IS ONLY ONE TABLE YOU NEED TO VIEW FROM THIS OUTPUT**. The table you need to view is near the bottom and is shown below with selected explanations.

Figure 16.4: Output from fp Macro with numbers corresponding to explanations.



Dev_linear	e_fp1	Dev_fp1	e1_fp2	e2_fp2	Dev_fp2	p_lin_fp1	p_lin_fp2	p_fp1_fp2
521.007	-2	452.668	2	2	386.868	0.0000	0.0000	0.0000

1.) **Dev_Linear:** This number indicates how much the trend deviates from the linear trend. The lower the number here, the more the regression model lines up to the linear model.
2.) **e_fp1**: Recall from the "fp pertinent concepts" section from earlier in which we discussed one-power transformations. The number that is indicated here tells you what the best 1 power transformation is. If it is "-2" then the best 1 power transformation is x^{-2}. Essentially, when you see this it is indicating to you that x^{e_fp1} is the best one power transformation procedure to be carried out.
3.) **Dev_fp1**: This number indicates how much the trend deviates from the one power transformation trend noted in e_fp1. The lower the number here, the more the regression model lines up to the one power transformation model.
4.) **e1_fp2:** Again, recall to the "fp pertinent concepts" section from earlier. This number is referring to the **FIRST PART** of the two power transformation. Similar to the e_fp1 output, if it is "2" then the best 1 power transformation is x^2. Essentially, when you see this it is indicating to you that x^{e_fp2} is the best one power transformation procedure to be carried out for the FIRST part of the 2 power transformation.
5.) **e2_fp2:** Again, recall to the "fp pertinent concepts" section from earlier. This number is referring to the **SECOND PART** of the two power transformation. Similar to the e_fp1 output, If it is "2" then the best 1 power transformation is x^2. Essentially, when you see this it is indicating to you that x^{e_fp2} is the best one power transformation procedure to be carried out for the SECOND part of the 2 power transformation. **NOTE: if this number matches the number on e1_fp2, then it is indicating to take the natural log of x in addition to the suggestion in e1_fp2**.
6.) **Dev_fp2:** This number indicates how much the trend deviates from the two power transformation trend noted in e1_fp2 and e2_fp2. The lower the number here, the more the regression model lines up to the one power transformation model.
7.) **p_lin_fp1:** This is a significance test assessing whether the Linear model is statistically significant from the one power transformation model. If there IS statistical significance ($p<.05$) then refer the model with lower deviance is likely the better model.
8.) **p_lin_fp2:** This is a significance test assessing whether the Linear model is statistically significant from the two power transformation model. If there IS statistical significance ($p<.05$) then refer the model with lower deviance is likely the better model.
9.) **p_fp1_fp2:** This is a significance test assessing whether the one power transformation model is statistically significant from the two power transformation model. If there IS statistical significance ($p<.05$) then refer the model with lower deviance is likely the better model.

Note: When the linear scale is best, missing values appear in the fp results table for the best one-power model (e_fp1).

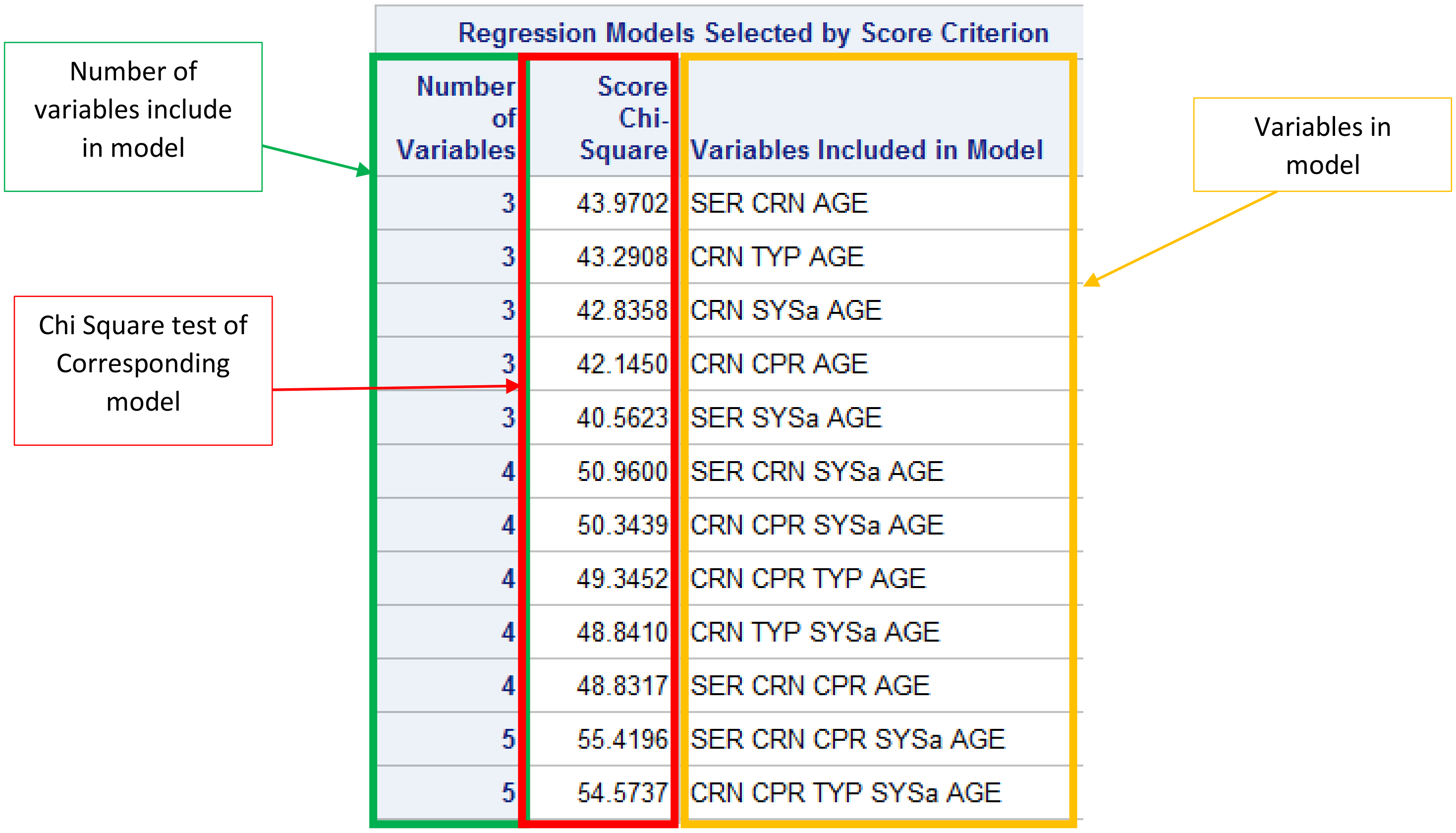
Automated Model Selection

Purposeful Selection

```
proc logistic descending data=ICU_altered;
model STA= SER CRN INF CPR TYP PO2 PCO CRE SYSa age
/ selection=score start=3 stop=8 best=5;
run;
```

- `proc logistic descending data=ICU_altered;` = Running the proc logistic function
- `model STA= SER CRN INF CPR TYP PO2 PCO CRE SYSa age` = The variables in your dataset. (Enter in all of the variables in the dataset, not just the ones of interest)
- `/ selection=score start=3 stop=8 best=5;` = The type of models you would like the automated selection to spit out.
    - "start=3" means you would like it to start by creating the best 3 main effect models.
    - "stop=8" indicates you would like it to stop at making the best 8 main effect models.
    - "Best=5" indicates you would like the auto selection to only show the best 5 models for each # of main effects.

Table 16.01: Output from Automated Stepwise selection with selected explanations


Number of variables include in model
Chi Square test of Corresponding model
Variables in model

Regression Models Selected by Score Criterion

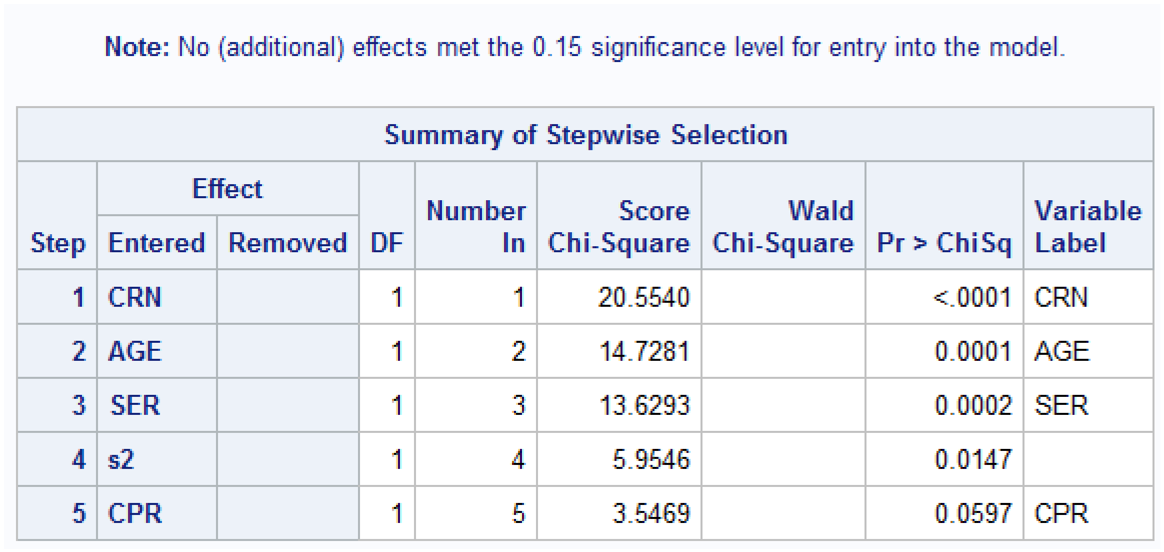
Number of Variables	Score Chi-Square	Variables Included in Model
3	43.9702	SER CRN AGE
3	43.2908	CRN TYP AGE
3	42.8358	CRN SYSa AGE
3	42.1450	CRN CPR AGE
3	40.5623	SER SYSa AGE
4	50.9600	SER CRN SYSa AGE
4	50.3439	CRN CPR SYSa AGE
4	49.3452	CRN CPR TYP AGE
4	48.8410	CRN TYP SYSa AGE
4	48.8317	SER CRN CPR AGE
5	55.4196	SER CRN CPR SYSa AGE
5	54.5737	CRN CPR TYP SYSa AGE

Stepwise Selection

```
proc logistic descending data=ICU_altered;
class SER CRN INF CPR TYP PO2 PCO CRE s1 s2/param=ref ref=first;
model STA= SER CRN INF CPR TYP PO2 PCO CRE s1 s2 age

/stepwise sle=0.15 sls=0.20 details;
run;
```

- `class SER CRN INF CPR TYP PO2 PCO CRE s1 s2/param=ref ref=first;`
  = the variables (again include ALL predictor variables in your dataset) that need to be classified. (DO not include continuous variables in here).
- `model STA= SER CRN INF CPR TYP PO2 PCO CRE s1 s2 age`
  = your model statement. Be sure and include the outcome and ALL variables!
- `/stepwise` = indicating that you would like SAS to perform stepwise selection
- `sle=0.15` = This is your "p-entry" only allows variables in the model that are less than this p-value (p<0.15 in this case)
- `sls=0.20 details;` = This is your "p-exit" this kicks any variables that have a p-value of greater than 0.20 out of the model.

Table: 16.02: Example output from Stepwise Selection.



Note: No (additional) effects met the 0.15 significance level for entry into the model.

Summary of Stepwise Selection

Step	Effect		DF	Number In	Score Chi-Square	Wald Chi-Square	Pr > ChiSq	Variable Label
	Entered	Removed						
1	CRN		1	1	20.5540		<.0001	CRN
2	AGE		1	2	14.7281		0.0001	AGE
3	SER		1	3	13.6293		0.0002	SER
4	s2		1	4	5.9546		0.0147	
5	CPR		1	5	3.5469		0.0597	CPR

Summary of table: Within this table, the variables running down the left side is the best model as determined by the stepwise selection. **PLEASE NOTE:** The inclusion of "s2" It should be noted that stepwise selection has the ability to split up design variables, therefore, you need to choose whether to include the other parts of the design variable or remove it entirely. IT DOES NOT MAKE SENSE TO INCLUDE PART OF A DESIGN VARIABLE.

Adding interaction terms

After you have tested for main effects, you can add all possible interactions to see if there are any significant interactions and determine which interactions are significant.

```
proc logistic descending data=ICU_altered;
class SER CRN CPR/param=ref ref=first;
model STA= SER CRN CPR age

SER*CRN SER*CPR SER*AGE
CRN*CPR CRN*AGE
CPR*AGE

/stepwise sle=0.15 sls=0.20 include=4 details;
run;
```

Reduced main effect model as determined by stepwise selection

Addition of all possible interactions

The "include" statement is necessary when including interactions. The number needs to include the total number of main effect variables you have within your model (in this case 4).

"Include" makes sure all of the main effects are included in the model.

Goodness of Fit (GOF) Tests

GOF test explanations

GOF tests are meant to determine if your predicted values truly are predicting the outcome. In summary, the tests below are a test to determine how close the predicted values are to the observed values. Thus, is the model truly predictive or is ready to fall apart?

What is a Covariate Pattern?

A covariate pattern = A set of values for the model covariate.

For example, If a model contains Age, Gender, and race then few examples of different covariate patterns would be:

1.) age=30, Gender=Female, Race=White
2.) Age=40, Gender=Female, Race =Black
3.) Age=22, Gender=Female, Race= Other

This is important for understanding which test should be used below. The tests below make assumptions about how many covariate patterns there are relative to the sample size. Therefore, certain ones should only be used in certain situations.

Couple notes:

1.) **Covariate pattern is often represented by a "J"**
2.) **None of these tests are truly good for sample sizes less than 400.**

Hosmer-Lemeshow (HL) GOF test

The Hosmer-Lemeshow test Groups covariate patterns using 10 groups.

Calculates the Pearson Chi Square on Groups, rather than the individuals.

P-values Greater than 0.05 (preferable greater than 0.25) indicate better model fit.

This test should be used when there are MANY covariate patterns (J) relative to the sample size (n).

Pearson Chi-Square (χ^2) GOF Test

The Pearson Chi-Square calculates the difference between the observed and the predicted value for each covariate pattern.

P-values Greater than 0.05 (preferable greater than 0.25) indicate better model fit.

This test should be used when there are FEW covariate patterns (J) relative to the sample size (n).

Deviance (D) GOF Test

The Deviance test calculates the deviance for each covariate pattern and squares it. Then adds the squared deviances over all covariate patterns.

P-values Greater than 0.05 (preferable greater than 0.25) indicate better model fit.

Use when there are FEW covariate patterns (J) relative to the sample size (n).

Osius-Rojek (OR) GOF Test

The Osius-Rojek test is a larger sample normal approximation to the Pearson Chi-Square test. → Sample size needs to be large! The results are likely to be wrong on a small sample size!

P-values Greater than 0.05 (preferable greater than 0.25) indicate better model fit.

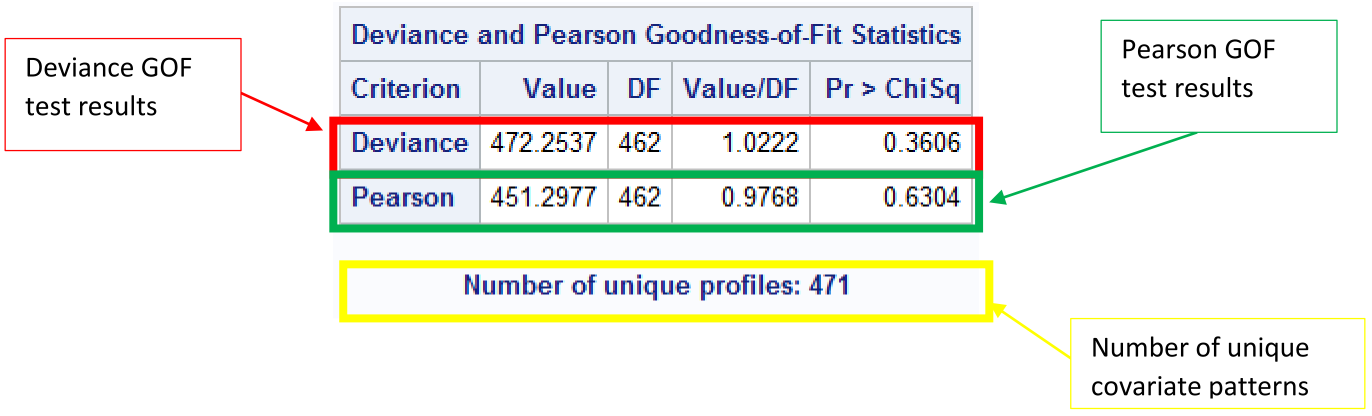
Use when there are fewer covariate patterns (J) than the sample size (n) (but not too few).

Hosmer-Lemeshow, Pearson Chi-square, and Deviance tests code

```
proc logistic descending data=glow500;
model fracture=priorfrac momfrac armassist raterisk2
height age priorfrac*age momfrac*armassist
/ scale=n aggregate lackfit;
run;
```

- `/ scale=n` = Perform Pearson chi square test
- `Aggregate` = Model by covariate pattern rather than by subject
- `lackfit;` = Perform Hosmer Lemeshow test

Table 17.01: Deviance and Pearson Output from above code with selected explanations.

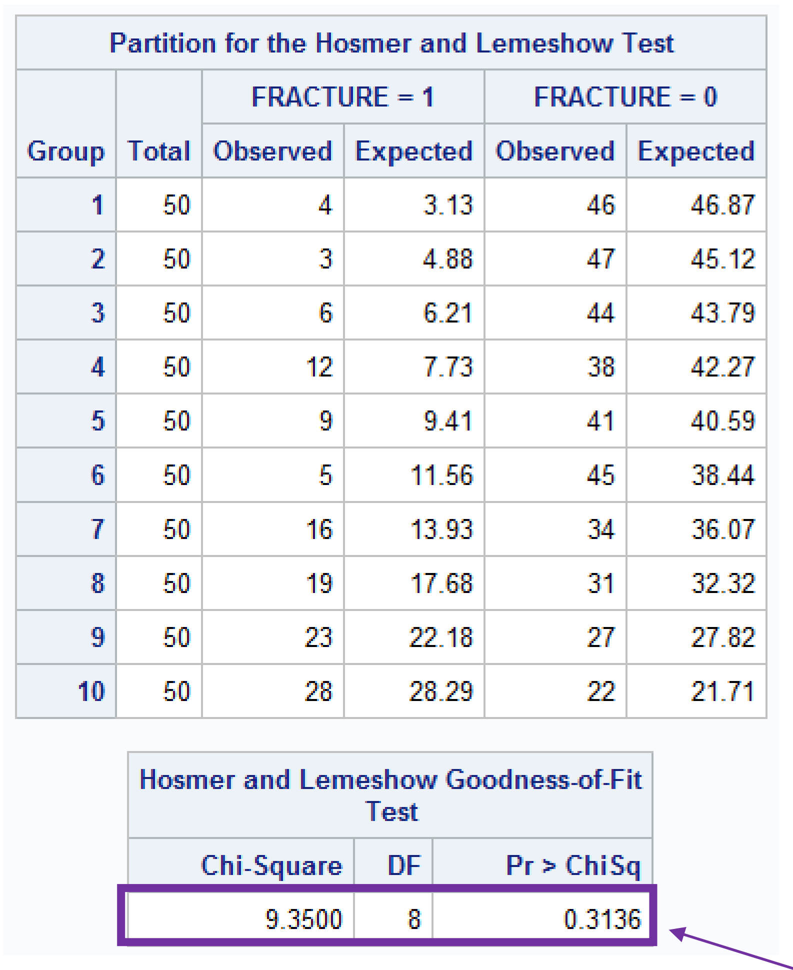

Deviance GOF test results
Pearson GOF test results
Number of unique covariate patterns

Deviance and Pearson Goodness-of-Fit Statistics

Criterion	Value	DF	Value/DF	Pr > ChiSq
Deviance	472.2537	462	1.0222	0.3606
Pearson	451.2977	462	0.9768	0.6304

Number of unique profiles: 471

Table 17.02: Hosmer-Lemeshow output from above code with selected explanations



Partition for the Hosmer and Lemeshow Test

Group	Total	FRACTURE = 1		FRACTURE = 0	
		Observed	Expected	Observed	Expected
1	50	4	3.13	46	46.87
2	50	3	4.88	47	45.12
3	50	6	6.21	44	43.79
4	50	12	7.73	38	42.27
5	50	9	9.41	41	40.59
6	50	5	11.56	45	38.44
7	50	16	13.93	34	36.07
8	50	19	17.68	31	32.32
9	50	23	22.18	27	27.82
10	50	28	28.29	22	21.71

Hosmer and Lemeshow Goodness-of-Fit Test

Chi-Square	DF	Pr > ChiSq
9.3500	8	0.3136

Hosmer-Lemeshow GOF test results

Osius-Rojek test code

Please note: this is a very long code, and there are only certain parts that need to be entered in. The parts that need to be changed, dependent on the data are described in various text boxes when scanning the code. In the end, you will only be outputted with one p-value. This is the p-value to determine your GOF using the Osius-Rojek test.

```
************************************************************;
**                   Osius-Rojek test                    **;
************************************************************;

* If interaction terms are in the model, create the interaction terms  ;
* If categorical variables with more than 2 categories are in the model;
* create design variables                                              ;
* Sort new data set by model covariates;

proc sort data=ICU_altered;                    ****  for a different data set
change independent variable names in by statement ****************;
    by SER PO2 age CAN;
run;
```

Insert Dataset name here

```
* For each covariate pattern, j, save m_j= # with covariate pattern j and;
* y_j = # with outcome=1 in covariate pattern j                          ;

proc means n sum noprint data=ICU_altered;     ****  for a different data set
change independent variable names in by statement ***************;
                                          ****   and outcome variable name in var
statement                                     ***************;
      by SER PO2 age CAN;
      var STA; output out=jdat n=m_j sum=y_j;   *←- indicate outcome;
run;

*LOOK ABOVE! After "var" is where your outcome should be;

* Run proc logistic for covariate patterns rather than individuals
* outcome=y_j / m_j (not 0 or 1) , save fitted values

proc logistic noprint descending data=jdat;    ****  for a different data set
change independent variable names in model statement. Keep jdat as jdat. Do
NOT change this*****;
  model y_j/m_j= SER PO2 age CAN;
  output out=pdat p=p_j;
run;
```

Insert Predictor variables here.

NOTE: They need to remain in the same order.

Insert outcome variable here

```
* Create v_j, c_j, the chi-square terms and the terms in the sum in A;

data pdat;  set pdat;
  v_j=m_j*p_j*(1-p_j);   c_j=(1-2*p_j)/v_j;   chisq_j=(y_j-m_j*p_j)**2/v_j;
  m_j_inv=1/m_j;
run;
```

Insert Predictor variables here.

NOTE: They need to remain in the same order.

```
* Create and save chi-square & sum for A                  ;
* Perform weighted linear regression, save SS             ;
* Calculate RSS,A,z & p-val for z                         ;

proc means sum noprint data=pdat;
  var chisq_j m_j_inv;  output out=cdat sum=chisq m_inv; run;

proc reg noprint data=pdat 66utset=ss;  ****  for a different data set change
independent variable names in model statement *****;
  model c_j=SER PO2 age CAN;
  weight v_j; run;
```

```
data zdat; merge cdat (keep=_freq_ chisq m_inv) ss (keep=_rmse_);
  rss=(_freq_-4-1)*_rmse_**2;  A=2*(_freq_-m_inv); ****  for a different data
set change 4 to number of variables in the model *****;
  z=(chisq-(_freq_-4-1))/sqrt(A+rss);  z=abs(z);   ****  for a different data
set change 4 to number of variables in the model *****;
  pval=(1-probnorm(z))*2;
run;

proc print noobs data=zdat; var pval; run;
```

Change these number to the total number of predictor variables.

i.e. there are 4 predictor variables in this model, so the number changes to 4

When you run the code, you will get only p-value (Displayed below).

REMEMBER a higher p=value indicates a *better* model of fit for the Osius-Rojek test.

Table 17.03: P-value output from the Osius-Rojek test code.

pval
0.010042

Testing for the Tails

Stukel "Goodness of Fit" Test

The Stukel test is technically not a Goodness of Fit test, but it produces a test to see if there are any extreme pihat observations.

Below you will find the code for the Stukel test. With instructions as to what to change within the code.

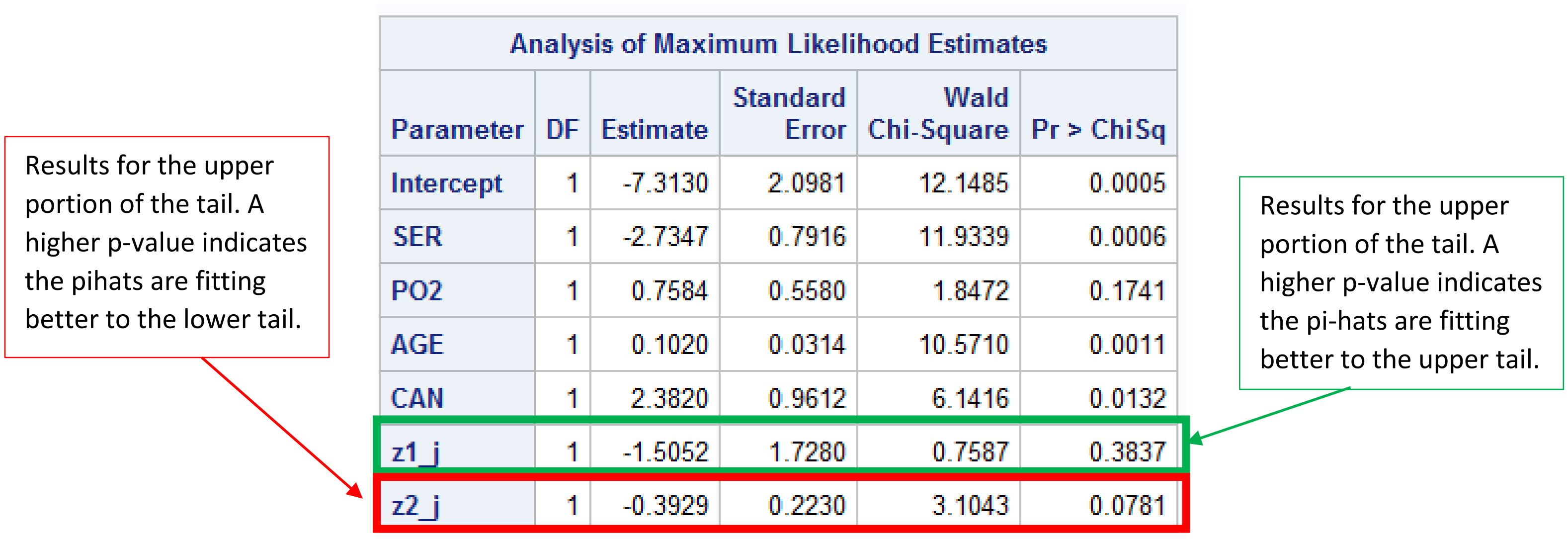
```
*************************************************************;
***** Stukel test of logistic regression model assumption ***;
*************************************************************;
ODS trace on;
ODS output GlobalTests=gt1;
proc logistic descending data=ICU_altered;      ****  for a different data
set change outcome and independent variable names *****;
                                                ****  in model statement
*****;
  model STA=SER PO2 age CAN;
  output out=pdat2 xbeta=g_j p=p_j;
run;

data pdat2;
  set pdat2;
  if p_j>=0.5 then ind1=1; else ind1=0;
  if p_j< 0.5 then ind2=1; else ind2=0;
  z1_j=0.5*g_j**2*ind1;
  z2_j=-0.5*g_j**2*ind2;
run;
```

Insert dataset name here.

Insert model statement here.

```
ODS output GlobalTests=gt2;
proc logistic descending data=pdat2 ; ****  for a different data set change
outcome and independent variable names *****;
                                       ****  in model statement but keep z1_j
and z2_j                               *****;
  model STA=SER PO2 age CAN z1_j z2_j;
run;

data pval;
  merge gt1(rename=(ChiSq=ChiSq1))
        gt2(rename=(ChiSq=ChiSq2));
  if _N_=1;
  drop Test df ProbChisq;

  lr=ChiSq2-ChiSq1;
  pval=(1-probchi(lr,2));
run;

proc print noobs data=pval; var pval; run;
```

Table 17.04: Output from Stukel Analysis of Maximum Likelihood Estimates



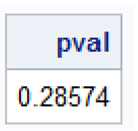
Analysis of Maximum Likelihood Estimates

Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	-7.3130	2.0981	12.1485	0.0005
SER	1	-2.7347	0.7916	11.9339	0.0006
PO2	1	0.7584	0.5580	1.8472	0.1741
AGE	1	0.1020	0.0314	10.5710	0.0011
CAN	1	2.3820	0.9612	6.1416	0.0132
z1_j	1	-1.5052	1.7280	0.7587	0.3837
z2_j	1	-0.3929	0.2230	3.1043	0.0781

Results for the upper portion of the tail. A higher p-value indicates the pihats are fitting better to the lower tail.

Results for the upper portion of the tail. A higher p-value indicates the pi-hats are fitting better to the upper tail.

Table 17.05: Final p-value output from the Stukel code.
NOTE: Remember, higher p-value indicates the tails are fitting to the model more correctly.



pval
0.28574

GOF Diagnostics

When running a GOF test, there are times that the errors lie within your data. Like, for example, when there are extreme outliers. There is no real test to actually view which actual data points are the ones that are causing the problems without actually viewing them on a graph statement. Below are some ways in order to properly assess outliers within your data.

NOTE: It is really not wise to remove too much from your data…. Seems like cheating to get your data to fit how you want it… right? Always have acaution and be sure that the outlier patterns you are faced with should truly be removed.

Creating Outlier Graphs

The code below is used to create outlier graphs. There will be 4 graphs outputted and the graphs created check the changes in certain outputs if a certain covariate pattern is removed. You can use the graphs below to assess if there are certain patterns that should be removed. Following the outputted graphs are ways to properly deal with any extreme outliers and observed values.

```
*************************************************************;
*****                    Diagnostics                     ***;
*************************************************************;
proc sort data=ICU_altered; by SER PO2 age CAN; run;

proc means n sum noprint data=ICU_altered;
      by SER PO2 age CAN;
      var STA; output out=jdat n=m_j sum=y_j;
run;

proc logistic descending data=jdat plots(only label)=(phat);
  model y_j/m_j=SER PO2 age CAN;
  output out=diag h=h difchisq=difchisq difdev=difdev c=db p=pihat;
run;
```

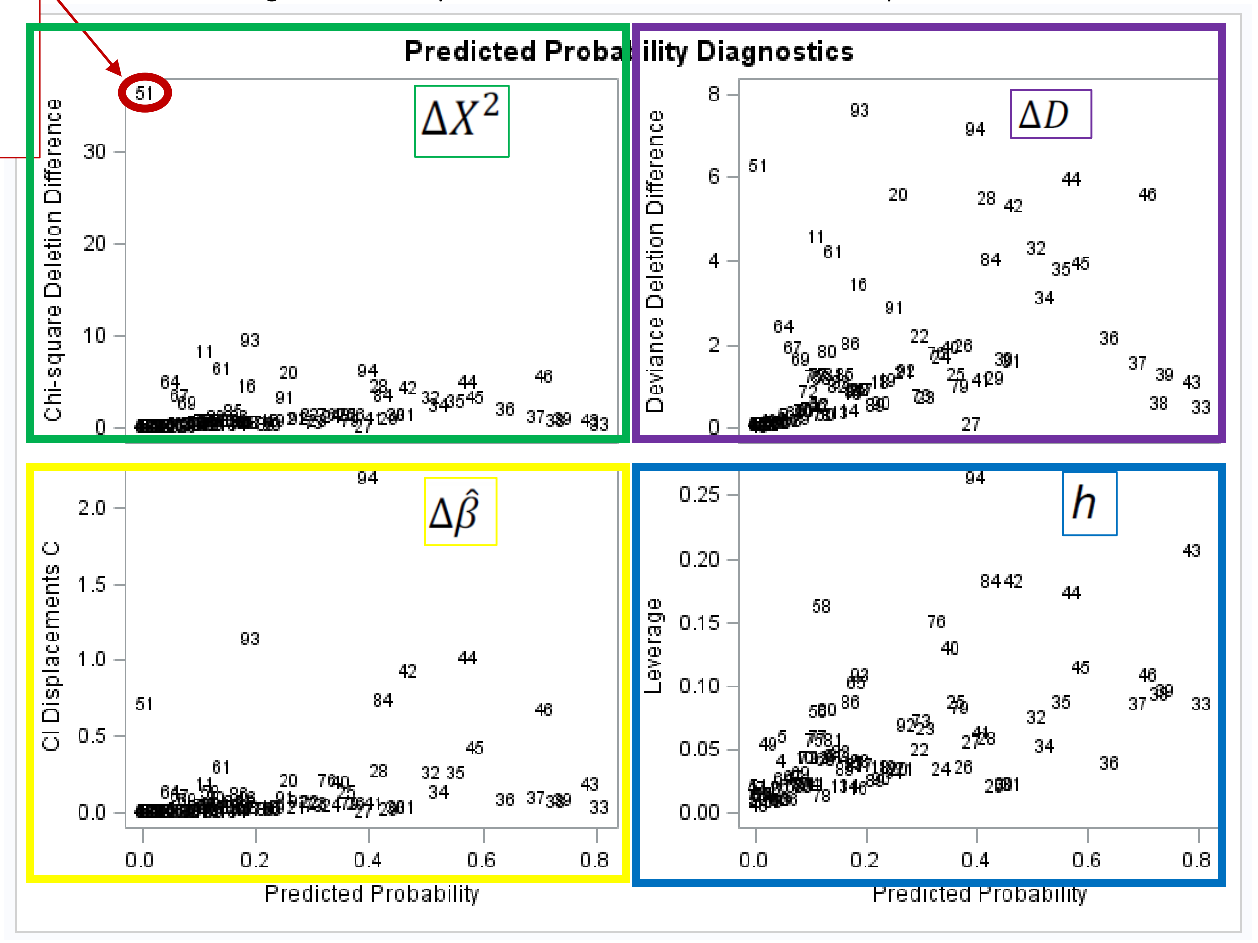
- `proc sort data=ICU_altered; by SER PO2 age CAN; run;` = sorting your data in order of your predictor variables.

-
  ```
  proc means n sum noprint data=ICU_altered;
  by SER PO2 age CAN;
  var STA; output out=jdat n=m_j sum=y_j;
  run;
  ```
  = outputting the n & sum from your model into a new dataset called "jdat"

-
  ```
  proc logistic descending data=jdat plots(only label)=(phat);
  output out=diag h=h difchisq=difchisq difdev=difdev c=db p=pihat;
  run;
  ```
  = Outputting the proc logistic function into a new output file name "diag" and printing out graphs indicating the model changes if outlier covariate patterns are removed. (explained and shown in the next figure)

Numbers refer to covariate patterns. Look to see which ones stand out to identify outliers.

Figure 17.1: Output from above code with selected explanations



Green Box Indicates the changes in the Pearson Chi-Square GOF test if the numbered covariate pattern is removed

Purple Box indicate Indicates the changes in the Deviance GOF test if the numbered covariate pattern is removed

Yellow box indicates the changes in the Beta Coefficients if the certain covariate pattern is deleted.

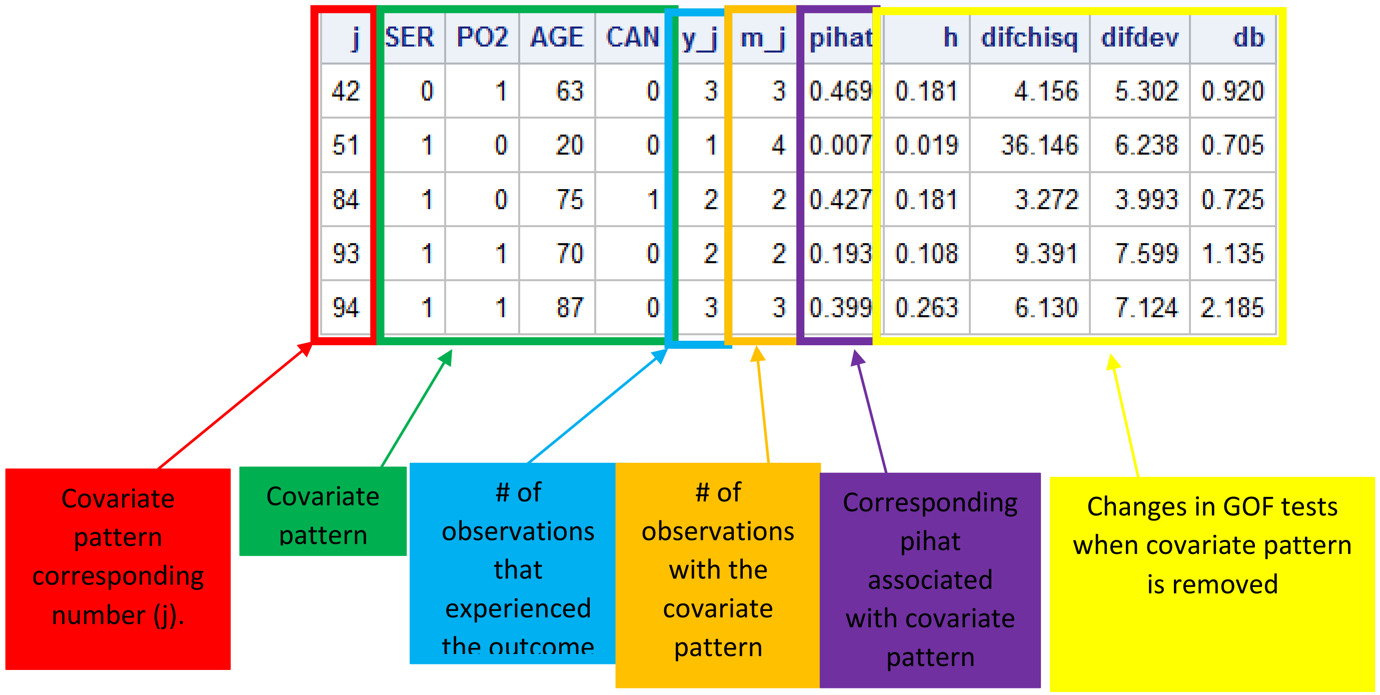
Blue Box indicates the changes in the leverage if the covariate pattern is deleted.

Identifying Outlier Covariate Patterns

The code should be used after you have identified all potential outliers from the graphs created in the "Creating Outlier graphs section. Only change the parts of the code that are indicated in the green text.

```
data diag; set diag; j=_N_; run;

proc print data=diag noobs; where j in (51, 94, 93, 84, 42); *<-- you change
these numbers based on the predicted probability dignostics. You are looking
for outliers;
  var j SER PO2 age CAN y_j m_j pihat h difchisq difdev db;  *<-- Change to
your variables do NOT include interactions keep everything else that is not a
variable;
  format pihat h difchisq difdev db 7.3; *<-- keep this the same, this is
just making your table look nice.;
run;
```

Table 17.05: Selected covariate patterns (j) that appear to be outliers from the above diagnostic graphs with selected explanations.



j	SER	PO2	AGE	CAN	y_j	m_j	pihat	h	difchisq	difdev	db
42	0	1	63	0	3	3	0.469	0.181	4.156	5.302	0.920
51	1	0	20	0	1	4	0.007	0.019	36.146	6.238	0.705
84	1	0	75	1	2	2	0.427	0.181	3.272	3.993	0.725
93	1	1	70	0	2	2	0.193	0.108	9.391	7.599	1.135
94	1	1	87	0	3	3	0.399	0.263	6.130	7.124	2.185

Covariate pattern corresponding number (j).
Covariate pattern
# of observations that experienced the outcome
# of observations with the covariate pattern
Corresponding pihat associated with covariate pattern
Changes in GOF tests when covariate pattern is removed

Getting OR’s and p-values with removed covariate patterns Macro

The following Macro code allows you to See How much the OR’s and corresponding p-values change if the outlier covariate patterns are removed from the dataset.

Instructions and explanations are indicated by text boxes in the code below:

Starting Macro

```
%macro outliers(j);
ODS output Logistic.ContrastEstimate=ors;
```

Indicate Model and Contrast statements (These are required)

```
proc logistic descending data=diag; where j ne &j;
  model y_j/m_j=SER PO2 age CAN;
  contrast 'SER 1 vs 0' SER 1/estimate=exp; *<- insert one contrast per
effect;
  contrast 'PO2 1 vs 0' PO2 1/estimate=exp; *<- insert one contrast per
effect;
  contrast 'AGE increase of 10 years' AGE 10/estimate=exp; *<- insert one
contrast per effecct;
  contrast 'CAN 1 vs 0' CAN 1/estimate=exp; *<- insert one contrast per
effect;

run;
```

```
data ors&j; set ors;
  drop type row stderr alpha lowerlimit upperlimit waldchisq;
  rename estimate=OR&j probchisq=p&j; run;
proc print data=ors&j; run;
```

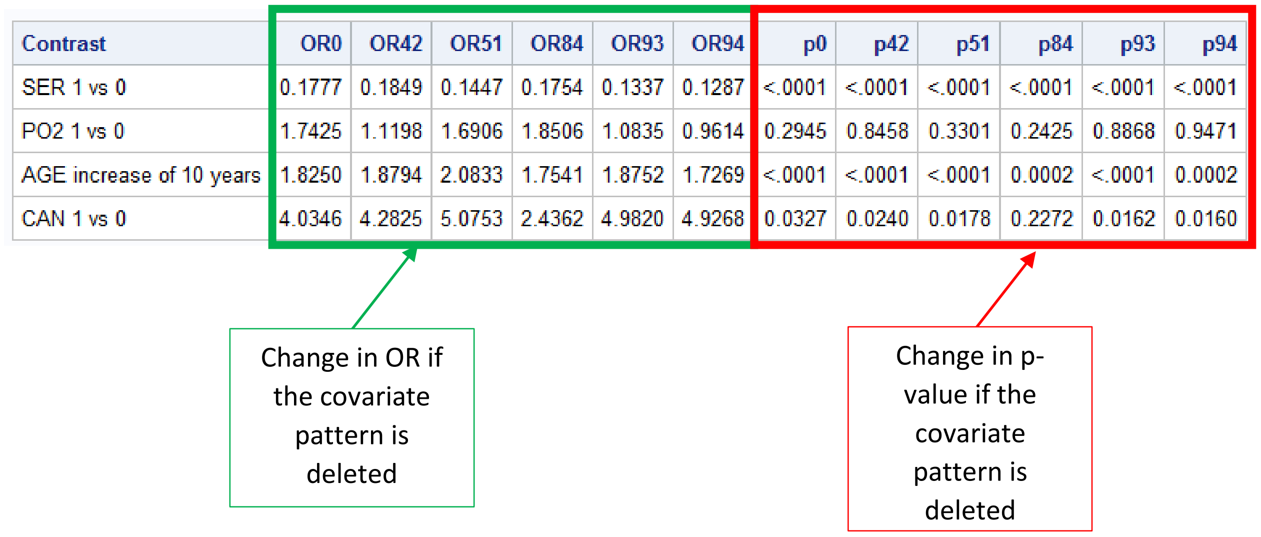
Part of the Macro, ignore and do NOT change

```
%mend outliers;

%outliers(0);
%outliers(42);
%outliers(51);
%outliers(84);
%outliers(93);
%outliers(94);
```

Indicate outlier cavariate pattern (do NOT delete the 0)

```
data ors; merge ors0 ors42 ors51 ors84 ors93 ors94; run;
proc print data=ors noobs;
  var contrast or0 or42 or51 or84 or93 or94 p0 p42 p51 p84 p93 p94;
  format ors0 ors42 ors51 ors84 ors93 ors94 6.2;
run;

proc datasets; delete ors; run; quit;
```

Table 17.06: Table indicating The Change OR's and p-values if the assigned covariate pattern is deleted with selected explanations



Contrast	OR0	OR42	OR51	OR84	OR93	OR94	p0	p42	p51	p84	p93	p94
SER 1 vs 0	0.1777	0.1849	0.1447	0.1754	0.1337	0.1287	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001
PO2 1 vs 0	1.7425	1.1198	1.6906	1.8506	1.0835	0.9614	0.2945	0.8458	0.3301	0.2425	0.8868	0.9471
AGE increase of 10 years	1.8250	1.8794	2.0833	1.7541	1.8752	1.7269	<.0001	<.0001	<.0001	0.0002	<.0001	0.0002
CAN 1 vs 0	4.0346	4.2825	5.0753	2.4362	4.9820	4.9268	0.0327	0.0240	0.0178	0.2272	0.0162	0.0160

Change in OR if the covariate pattern is deleted
Change in p-value if the covariate pattern is deleted

Removing Outliers

Once you have determined you need to remove an outlier with a certain covariate patterns you will need to create separate datasets, removing the unwanted covariate patterns and re-run all necessary GOF tests on the newly created dataset.

Hosmer-Lemeshow, Pearson Chi-Square & Deviance Outlier removal

After you have run the macro to identify change in OR's and p-values. You can now designate a proc logistic function in order to run the model again without the unwanted covariate pattern. This will give you the Hosmer-Lemeshow, Pearson Chi-Square and Deviance GOF tests while removing all observations with the unwanted covariate pattern.

Below is the code to do this:

```
proc logistic descending data=diag;
where j ne (42);
model y_j/m_j = SER PO2 age CAN/scale=n aggregate lackfit;
run;
```

- `proc logistic descending data=diag;` = Running the logistic function on the newly created diag dataset
- `where j ne (42);` = Telling SAS to run the function without any observations that have covariate pattern 43 (j = covariate pattern, ne= not equal, 42 = the designated covariate pattern)
- `model y_j/m_j = SER PO2 age CAN` = Your model statement.
- `/scale=n aggregate lackfit;` = Telling SAS to run the GOF tets.
- `run;` = Run statement

******NOTE: if you would like to remove MULTIPLE Covariate patterns, change "ne" to "not in" and separate all unwanted covariate patterns by a space.******************

- i.e `where j not in (42 51 93);`

Complete Identification and removal of Outliers from data set (For OR & Stukel GOF tests)

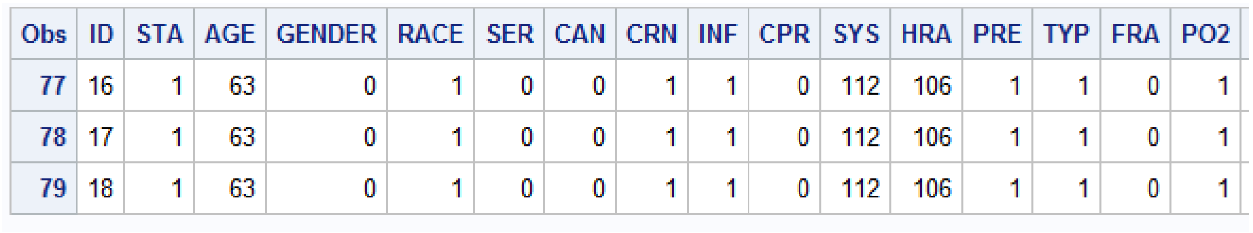
- Step 1: Find Observations containing covariate patterns in the dataset.
    - In order to do this, you will need to create a proc print statement in which the covariate pattern is identified. (Shown below).

```
Proc print data=ICU_altered;
where SER=0 and PO2=1 and AGE=63 and CAN=0;
run;
```

- `Proc print data=ICU_altered;` = Data to be printed
- `where SER=0 and PO2=1 and AGE=63 and CAN=0;` = covariate pattern (as designated in the "Identifying outlier covariate patterns section. Make sure the numbers match the corresponding covariate pattern.
- `run;` = Run procedure

Once run, You will then be outputted with all observations that contain the specified covariate pattern (Shown below).

Table 17.07: Proc Print of observations that contain the covariate pattern of interest.



Obs	ID	STA	AGE	GENDER	RACE	SER	CAN	CRN	INF	CPR	SYS	HRA	PRE	TYP	FRA	PO2
77	16	1	63	0	1	0	0	1	1	0	112	106	1	1	0	1
78	17	1	63	0	1	0	0	1	1	0	112	106	1	1	0	1
79	18	1	63	0	1	0	0	1	1	0	112	106	1	1	0	1

- Step 2: Using the ID (or whatever the observation identifier is), we now need to create unique data sets where those id's are NOT included.
    - To do this we create a subset of our original data set (in this case, the ICU_altered dataset. The code to do this is below:

```
data ICU_42;
set ICU_altered;
where ID not in (16, 17, 18);
run;
```

  - `data ICU_42;` = New dataset being created (In this case we are naming the new dataset "ICU_42")
  - `set ICU_altered;` = The dataset that is being pulled from
  - `where ID not in (16, 17, 18);` = Indicating to SAS not to include ID#'s 16, 17, & 18 in the new data set.
- Step 3: Using the new dataset, re-run the desired GOF test (in reality, this can be used on all GOF tests… Not just the OR & Stukel). Just make sure to run the test with the new dataset you created above.

Classification Tables

What is a classification table?

A classification table is simple a table that compares the how often the predicted outcomes aligned with the true observed outcome. With these tables, we can indicate Sensitivity, Specificity, Positive Predictive Values (PPV) and Negative Predictive Values (NPV). The steps to getting all of this information is outlined below.

Classification Table Terminology

Below is a list of important terminology to the section below.

Sensitivity

- Sensitivity is the proportion of individuals that have an outcome were correctly identified as having the outcome.

Specificity

- Specificity is the proportion of individuals that were correctly identified as *NOT* having the outcome.

Positive Predictive Value (PPV)

- The probability that those indicated with having the outcome truly have the outcome.

Negative Predictive Value (NPV)

- The probability that those indicated with having the *NOT* having the outcome truly do *NOT* have the outcome.

The Receiver Operating Characteristic (ROC) curve

Before we start creating a Classification table. It is nice to see how well our data is truly discriminating the data.

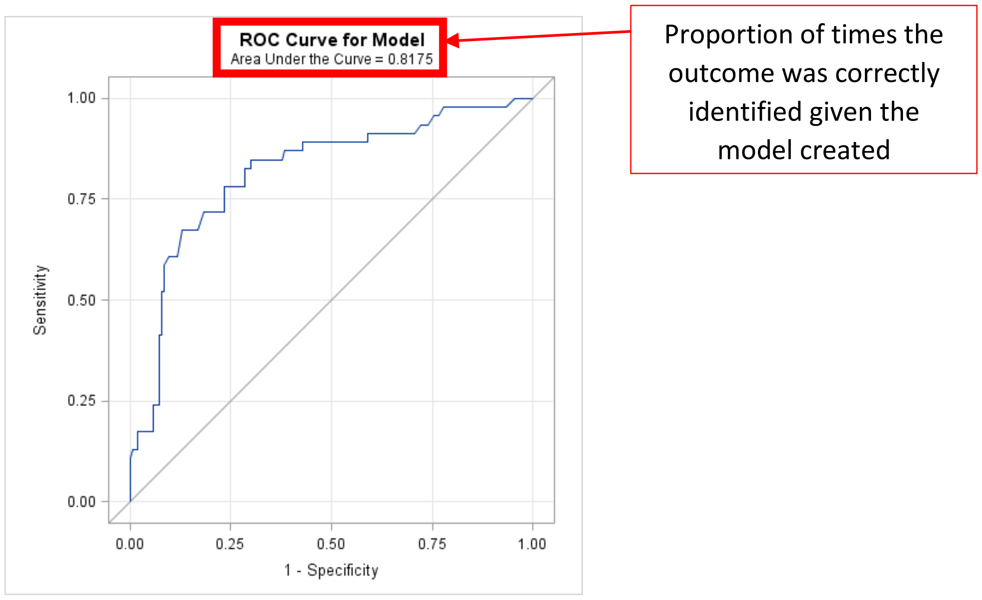
The ROC curve measures the receivers ability to discriminate between true and false signals. In other words, the area under the ROC curve the proportion that the model correctly identified the outcome. The code to create it is shown below:

```
proc logistic descending data=ICU_altered;
model STA=SER PO2 age CAN / outroc=rocdat;
run;
```

- `model STA=SER PO2 age CAN` = The model being tested.
- `/ outroc=rocdat;` = creating a new dataset & printing the ROC curve.

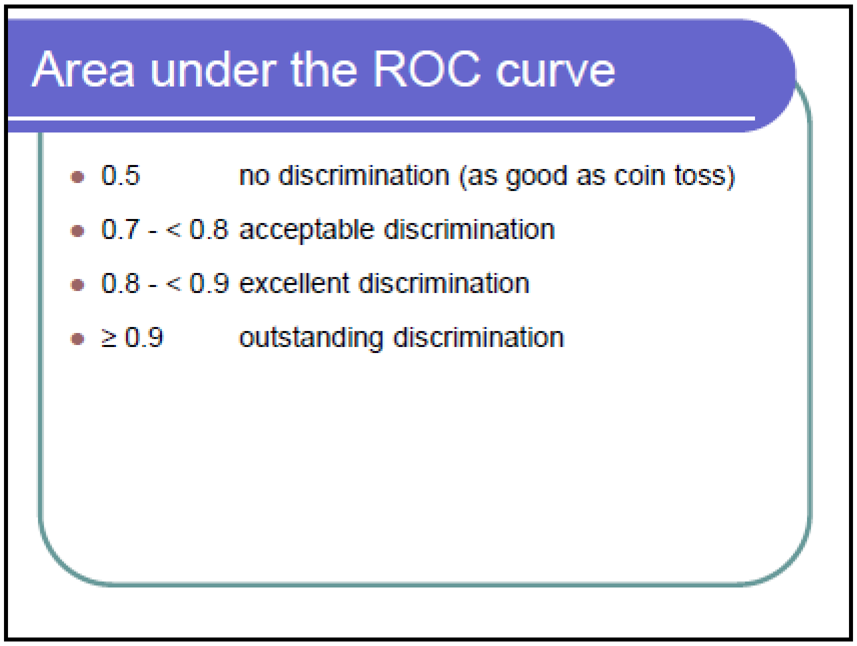
If ran correctly, the ROC curve will appear as so:

Figure 18.01: Output ROC curve form above with selected explanations



Determining a "Good" Area under the ROC Curve

It is common for people to want to see your ROC curve looks like if you are creating a model, therefore, below you will find a table with explanations of a way that the area is commonly defined:



Area under the ROC curve

- 0.5 no discrimination (as good as coin toss)
- 0.7 - < 0.8 acceptable discrimination
- 0.8 - < 0.9 excellent discrimination
- ≥ 0.9 outstanding discrimination

PLEASE NOTE: For a model to have "outstanding discrimination", this would require Quasi-complete separation of your model (This indicating a bad GOF test). We will talk more about the relation (or rather, non-relation) of GOF and Outcome predictability later.

Creating the Classification Table

Before doing this **PLEASE NOTE** that you will first have to creat your ROC curve, there is a creation of a dataset within the above “ROC Curve” step that we will need.

- **Step 1:** Note that the graph created in the ROC curve from the “Rocdat” dataset gave us a “1-Specificity” on the X-axis of the graph. Unfortunately, this is not the number we want, there fore, we need to manipulate that variable so that it is just back to the normal specificity. Therefore, we will do “1-1-specificity” in order to be back to transform it back to the normal specificity in the dataset and continue. The procedure for this is shown below:

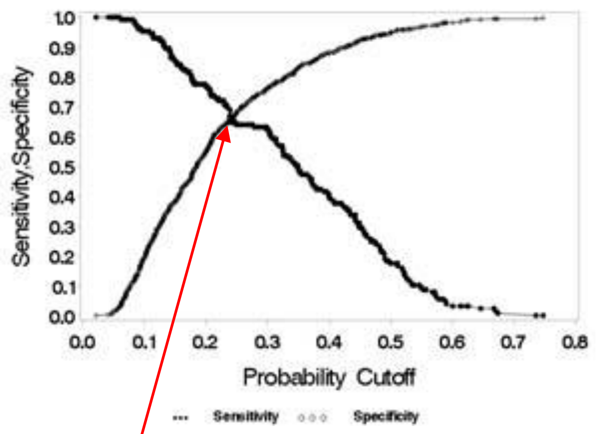
```
data rocdat;
set rocdat;
spec=1-_1mspec_;
run;
```

    - `data rocdat;` = Using the rocdat dataset created above
    - `set rocdat;` = setting the rocdat dataset for use (This is a datastep)
    - `spec=1-_1mspec_;` = Getting 1 -1- specificity
    - `run;` = Run statement to manipulate the variable.
- **Step 2:** Now we need to create a graph, This graph shows where the sensitivity and Specificity Values meet up. Below, there will not be explanation of the code. Be sure you are using the “rocdat” dataset created from below and if you run this universal graph statement as well as the proc gplot, it will give you the desired output. The code is below:

```
axis1 label=(f=swiss h=2.5 'Probability Cutoff')
minor=none;
axis2 label=(f=swiss h=2.5 a=90 'Sensitivity,Specificity')
minor=none;
goptions FTEXT=swissb HTEXT=2.0 HSIZE=8 in VSIZE=6 in;
symbol1 v=dot i=join c=black h=1;
symbol2 v=diamond i=join c=black h=1;
footnote1 c=black f=special h=1 'J J J' f=swissb h=1.5 '
Sensitivity'
c=black f=special h=1 ' D D D' f=swissb h=1.5 '
Specificity';

proc gplot data=rocdat;
plot (_sensit_ spec)*_prob_
/overlay haxis=axis1 vaxis=axis2;
run; quit;
footnote;
```

If done correctly, the output will appear similar to this:

Figure 18.02: Graph of Sensitivity and Specificity and the corresponding probability cutoff.



- **Step 3:** Now it is important to identify the best cutpoint so that you can best correctly calculate the specificity and sensitivity. The best way to do this is to look the above graph (Figure 18.02) and identify where they meet. In this case, the best cutpoint 0.25.

- **Step 4:** Next, we will create the classification table. The code will be provided below with text box explanations as to what each part of the code means. The table created gives you the sensitivity, Specificity, PPV's and NPV's for the cutpoint selected and the population prevalence of the event.

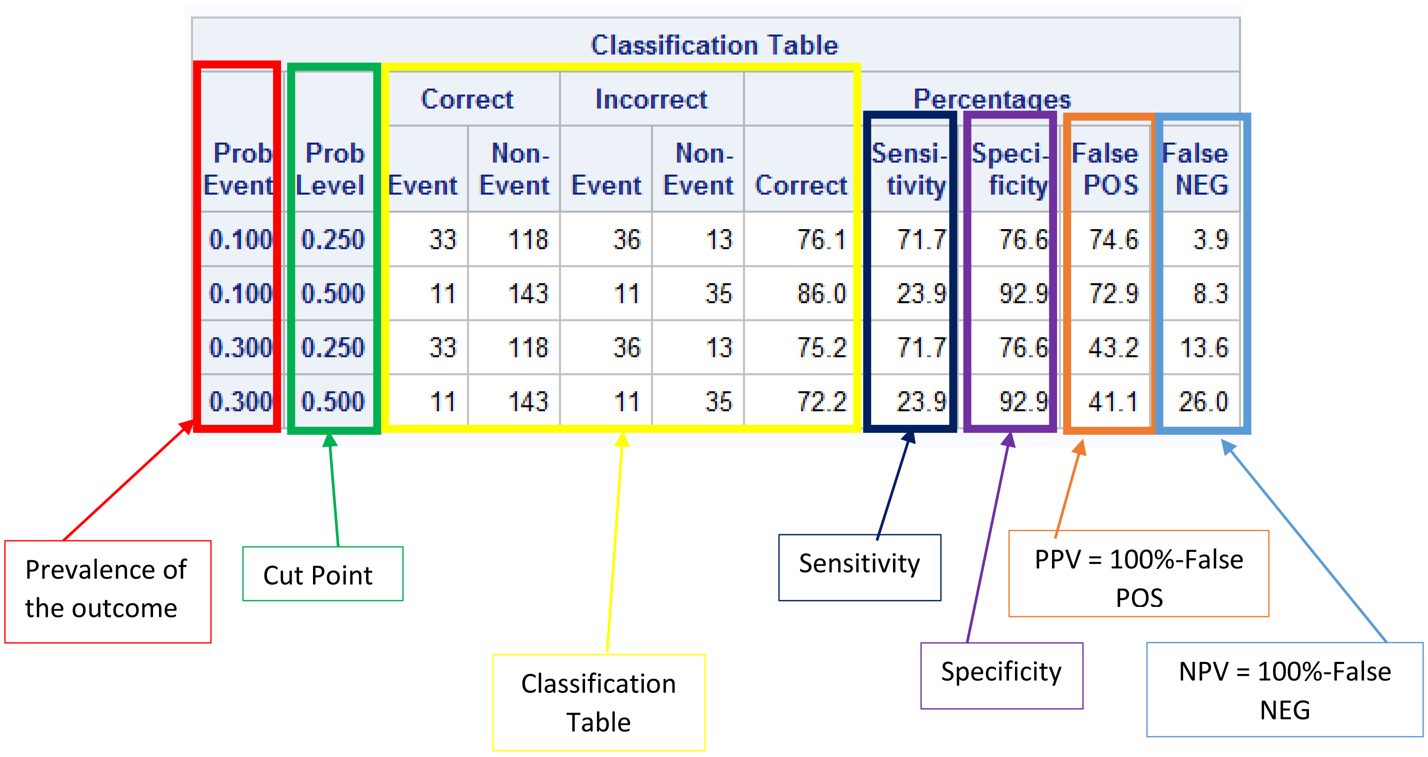
```
proc logistic descending data=ICU_altered;
model STA=SER PO2 age CAN
/ctable pprob=(0.25 0.5) pevent=0.10 0.30;
run;
```

Calculates Se, Sp, PPV and NPV for cutpoints 0.25 and 0.5
0.25 is the best cutpoint here
0.5 is the standard cutpoint and should always be tested

Determines PPV and NPV for prevalence's 10% and 30% (Change dependent on what you know about the outcome)

The Output from the above code is below:

Table 18.01: Classification Table created with Prevalence of outcome set at 0.1 & 0.3 & the Cut points set at 0.25 & 0.50 with selected explanations.



Classification Table

		Correct		Incorrect		Percentages				
Prob Event	Prob Level	Event	Non-Event	Event	Non-Event	Correct	Sensi-tivity	Speci-ficity	False POS	False NEG
0.100	0.250	33	118	36	13	76.1	71.7	76.6	74.6	3.9
0.100	0.500	11	143	11	35	86.0	23.9	92.9	72.9	8.3
0.300	0.250	33	118	36	13	75.2	71.7	76.6	43.2	13.6
0.300	0.500	11	143	11	35	72.2	23.9	92.9	41.1	26.0

Prevalence of the outcome
Cut Point
Classification Table
Sensitivity
Specificity
PPV = 100%-False POS
NPV = 100%-False NEG

A final note concerning Classification and Goodness of Fit

It should be noted that Specificity, Sensitivity, PPV, & NPV are not in any way indicative of how well the model is fitting.

Identifying Power and Sample Size

Remember back in epidemiologic methods when we used Openepi to identify our power and sample size? Good news! You are able to do all this through SAS! Below is a summary on how to do that.

Some things to know…

Please note, in order to identify sample size and power there is NO reason to have an initial data step. We are NOT working with data here.

REMEMBER! These calculations are meant to done as preliminary analyses. In other words, these are things you should know before you apply for a grant to conduct a research study. Some of the information can be obtained either by looking into the literature or running a pilot study.

For a Univariate Analysis

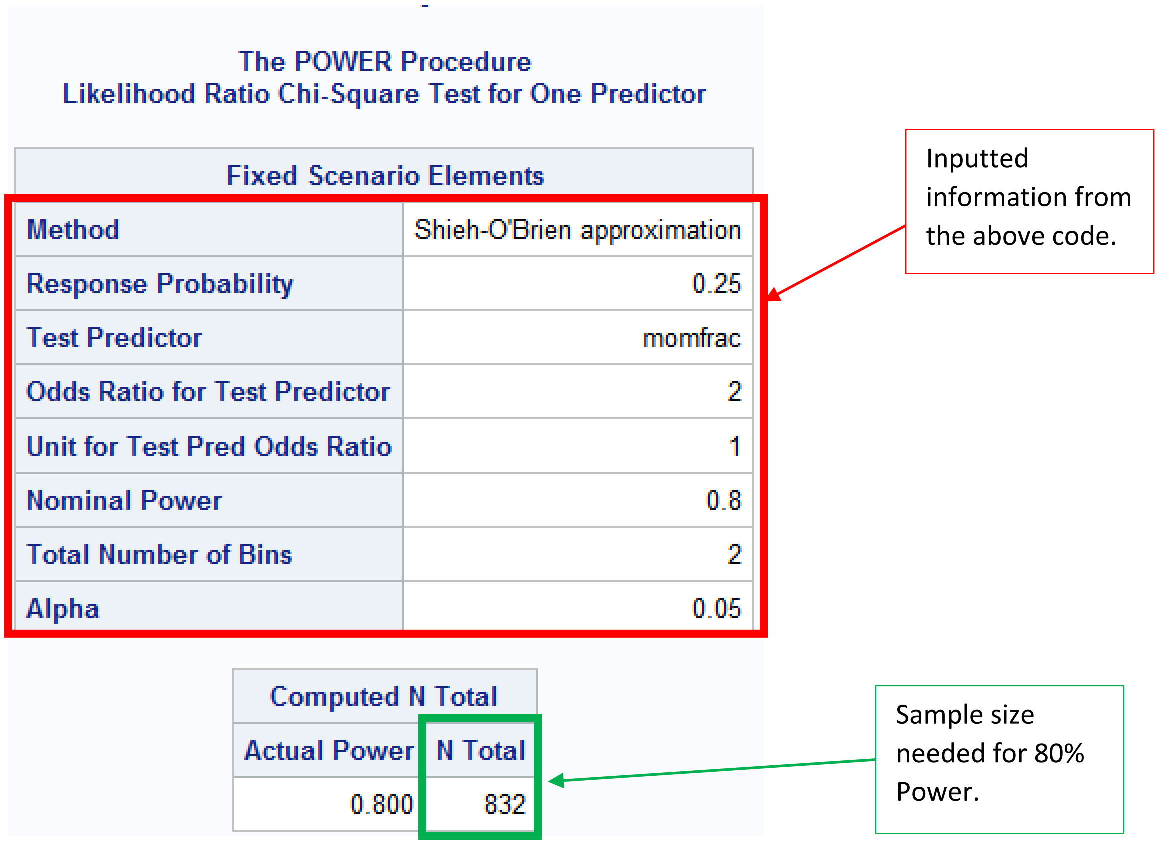
Getting Sample Size

Below is the code needed in order to conduct a power analysis for a study that will be using a logistic regression model (or rather, a study that has a 0-1 outcome) with selected explanations.

```
proc power;
logistic
vardist("momfrac") = binomial( .1 ,1)
testpredictor = "momfrac"
responseprob = 0.25
testoddsratio = 2.0
ntotal = .
power = 0.8;
run;
```

- `proc power;` = Beginning the power function
- `logistic` = Indicating that the power analysis for a logistic regression model
- `vardist("momfrac")` = Indicating what the predictor variable is
- `= binomial( .1 ,1)` = indicating that 10% of the controls have MOMFRAC=yes
- `testpredictor = "momfrac"` = This is noting that our variable of interest is MOMFRAC
- `responseprob = 0.25` = Noting that 25% of the sample are cases
- `testoddsratio = 2.0` = Indicating that our expected OR that we are trying to detect is 2.0 (found through the literature or a pilot study)
- `ntotal = .` = The "." Is telling SAS that this is the value we want it to detect (ntotal = sample size)
- `power = 0.8;` = Telling SAS that you want to the sample size needed to have 80% power
- `run;` = Run the Proc Power statement.

Table 19.01: Sample size calculator with selected explanations.



Inputted information from the above code.
Sample size needed for 80% Power.

The POWER Procedure
Likelihood Ratio Chi-Square Test for One Predictor

Fixed Scenario Elements	
Method	Shieh-O'Brien approximation
Response Probability	0.25
Test Predictor	momfrac
Odds Ratio for Test Predictor	2
Unit for Test Pred Odds Ratio	1
Nominal Power	0.8
Total Number of Bins	2
Alpha	0.05

Computed N Total	
Actual Power	N Total
0.800	832

Getting Power

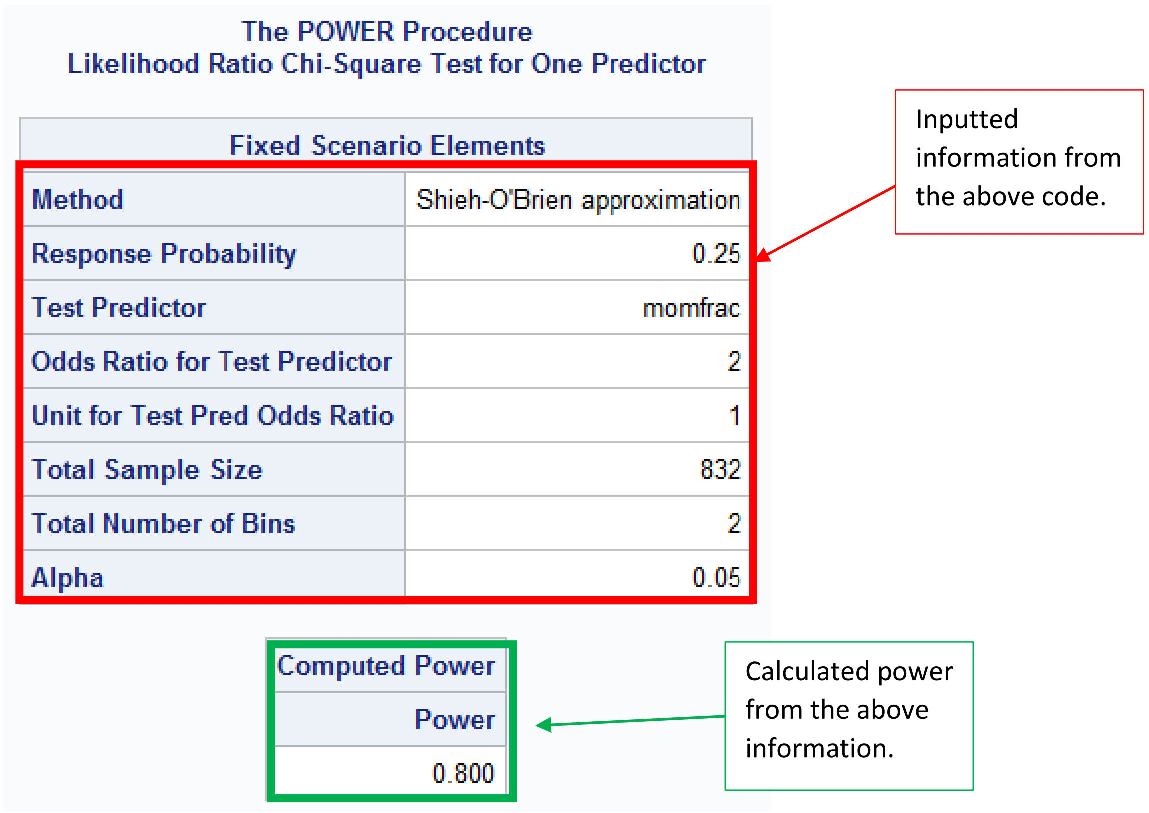
Just a note: To get power is very simple, the only difference between getting the sample size is the identification of the sample size and putting period in place of the desired power.

```
proc power;
logistic
vardist("momfrac") = binomial( .1 ,1)
testpredictor = "momfrac"
responseprob = 0.25
testoddsratio = 2.0
ntotal = 832
power = .;
run;
```

- `proc power;` = Beginning the power function
- `logistic` = Indicating that the power analysis for a logistic regression model
- `vardist("momfrac")` = Indicating what the predictor variable is

- `= binomial( .1 ,1)` = indicating that 10% of the controls have MOMFRAC=yes
- `testpredictor = "momfrac"` = This is noting that our variable of interest is MOMFRAC
- `responseprob = 0.25` = Noting that 25% of the sample are cases
- `testoddsratio = 2.0` = Indicating that our expected OR that we are trying to detect is 2.0 (found through the literature or a pilot study)
- `ntotal = 832` = Designating that the sample size of the study is 832.
- `power = .;` = The "." Is telling SAS that this is the value we want it to detect (power = detected power)
- `run;` = Run the Proc Power statement.

Table 19.02: Power calculator with selected explanations



The POWER Procedure
Likelihood Ratio Chi-Square Test for One Predictor

Fixed Scenario Elements	
Method	Shieh-O'Brien approximation
Response Probability	0.25
Test Predictor	momfrac
Odds Ratio for Test Predictor	2
Unit for Test Pred Odds Ratio	1
Total Sample Size	832
Total Number of Bins	2
Alpha	0.05

Computed Power
Power
0.800

Inputted information from the above code.

Calculated power from the above information.

For a Multivariate Analysis

Getting Sample Size

Below is the code needed in order to conduct a power analysis for a study that will be using a logistic regression model (or rather, a study that has a 0-1 outcome) with selected explanations for multivariate analysis

```
proc power;
logistic
vardist ("momfrac") = binomial(0.1, 1)
vardist ("priorfrac") = binomial(0.2,1)
vardist ("armassist") = binomial(0.3, 1)
vardist ("raterisk2") = binomial(0.25,1)
vardist ("age") = normal(67.5,9)
vardist ("height") = normal(162,6)
testpredictor = "momfrac"
covariates = ("priorfrac" "armassist" "raterisk2"
"age" "height")
responseprob = 0.25
testoddsratio = 2.0
covoddsratios = (2.0 1.5 1.5 1.5 0.6)
units = ("age"=10 "height"=10)
ntotal = .
power = 0.8;
run;
```

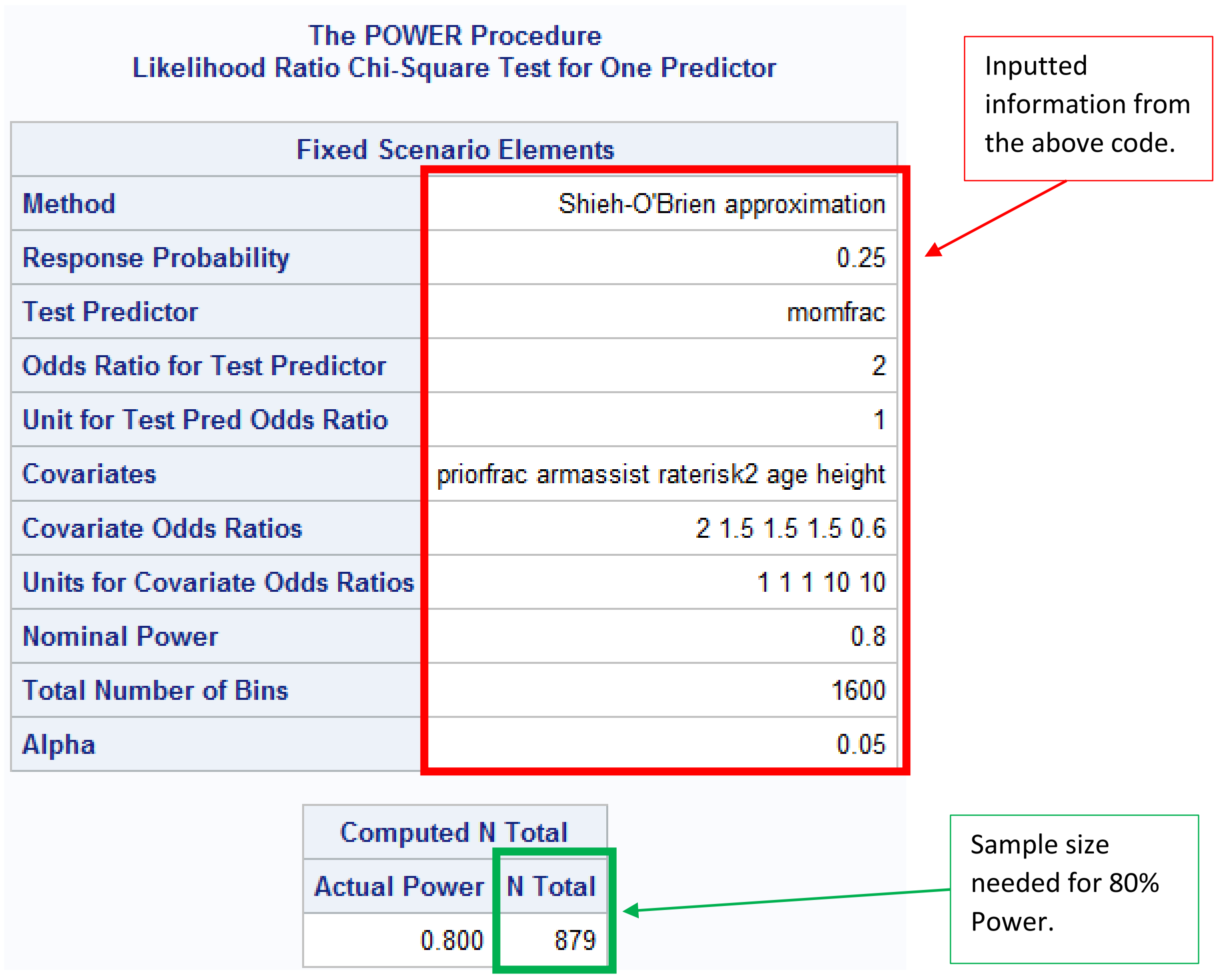
≈10% of controls have momfrac=yes
≈20% of controls have priorfrac=yes
≈30% of controls have armassist=yes
≈25% of controls have raterisk2=yes

Age: Among controls, mean=67.5, std=9
Height: Among controls, mean=162, std=6
Continuous variables may have a distribution other than normal (e.g. lognormal)

- `proc power;` = Beginning the power function
- `logistic` = Indicating that the power analysis for a logistic regression model
- `vardist ("momfrac") = binomial(0.1, 1)`
- `vardist ("priorfrac") = binomial(0.2,1)`
- `vardist ("armassist") = binomial(0.3, 1)`
- `vardist ("raterisk2") = binomial(0.25,1)`
- `vardist ("age") = normal(67.5,9)`
- `vardist ("height") = normal(162,6)`
- `testpredictor = "momfrac"` = This is noting that our variable of interest is MOMFRAC
- `covariates = ("priorfrac" "armassist" "raterisk2" "age" "height")` = Including all of the model covariates
- `responseprob = 0.25` = Noting that 25% of the sample are cases
- `testoddsratio = 2.0` = Indicating that our expected OR that we are trying to detect is 2.0 (found through the literature or a pilot study [Specifically for the variable of interest, which is "MOMFRAC in this case])
- `covoddsratios = (2.0 1.5 1.5 1.5 0.6)` = The expected OR's for model covariates. NOTE: These should be in the same order as the variables listed above.
- `units = ("age"=10 "height"=10)` = The Units statements for your continuous variable covariates.

- `ntotal = .` = The "." Is telling SAS that this is the value we want it to detect (ntotal = sample size)
- `power = 0.8;` = Telling SAS that you want to the sample size needed to have 80% power
- `run;` = Run the Proc Power statement.

Table 19.03: Sample size calculator with selected explanations for multivariate model.



The POWER Procedure
Likelihood Ratio Chi-Square Test for One Predictor

Fixed Scenario Elements	
Method	Shieh-O'Brien approximation
Response Probability	0.25
Test Predictor	momfrac
Odds Ratio for Test Predictor	2
Unit for Test Pred Odds Ratio	1
Covariates	priorfrac armassist raterisk2 age height
Covariate Odds Ratios	2 1.5 1.5 1.5 0.6
Units for Covariate Odds Ratios	1 1 1 10 10
Nominal Power	0.8
Total Number of Bins	1600
Alpha	0.05

Inputted information from the above code.

Computed N Total	
Actual Power	N Total
0.800	879

Sample size needed for 80% Power.

Getting Power

Just a note: To get power is very simple, the only difference between getting the sample size is the identification of the sample size and putting period in place of the desired power.

```
proc power;
logistic
vardist ("momfrac") = binomial(0.1, 1)
vardist ("priorfrac") = binomial(0.2,1)
vardist ("armassist") = binomial(0.3, 1)
vardist ("raterisk2") = binomial(0.25,1)
vardist ("age") = normal(67.5,9)
vardist ("height") = normal(162,6)
```

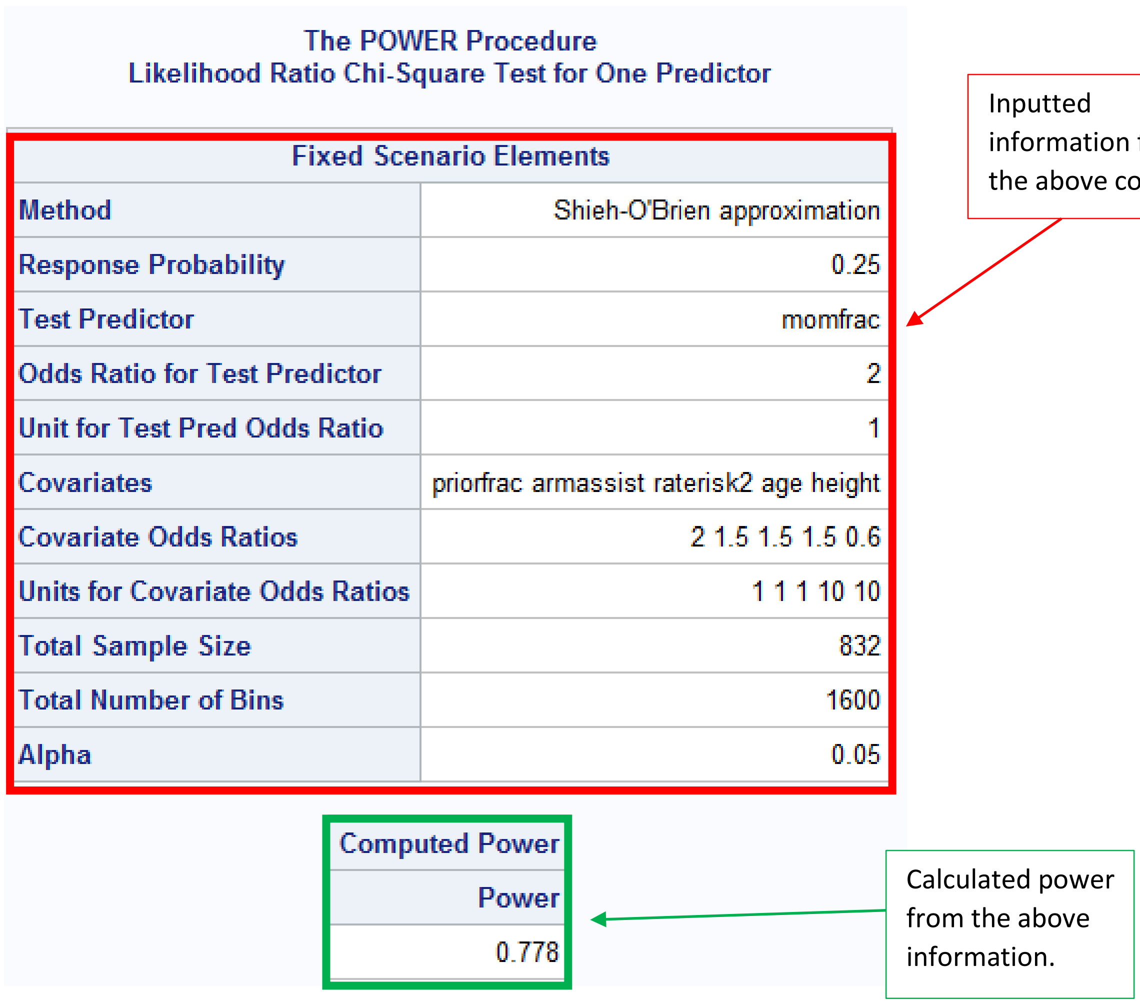
```
testpredictor = "momfrac"
covariates = ("priorfrac" "armassist" "raterisk2"
"age" "height")
responseprob = 0.25
testoddsratio = 2.0
covoddsratios = (2.0 1.5 1.5 1.5 0.6)
units = ("age"=10 "height"=10)
ntotal = .
power = 0.8;
run;
```

≈10% of controls have momfrac=yes
≈20% of controls have priorfrac=yes
≈30% of controls have armassist=yes
≈25% of controls have raterisk2=yes

Age: Among controls, mean=67.5, std=9
Height: Among controls, mean=162, std=6
Continuous variables may have a distribution other than normal (e.g. lognormal)

- `proc power;` = Beginning the power function
- `logistic` = Indicating that the power analysis for a logistic regression model
- `vardist ("momfrac") = binomial(0.1, 1)`
- `vardist ("priorfrac") = binomial(0.2,1)`
- `vardist ("armassist") = binomial(0.3, 1)`
- `vardist ("raterisk2") = binomial(0.25,1)`
- `vardist ("age") = normal(67.5,9)`
- `vardist ("height") = normal(162,6)`
- `testpredictor = "momfrac"` = This is noting that our variable of interest is MOMFRAC
- `covariates = ("priorfrac" "armassist" "raterisk2" "age" "height")` = Including all of the model covariates
- `responseprob = 0.25` = Noting that 25% of the sample are cases
- `testoddsratio = 2.0` = Indicating that our expected OR that we are trying to detect is 2.0 (found through the literature or a pilot study [Specifically for the variable of interest, which is "MOMFRAC in this case])
- `covoddsratios = (2.0 1.5 1.5 1.5 0.6)` = The expected OR's for model covariates. NOTE: These should be in the same order as the variables listed above.
- `units = ("age"=10 "height"=10)` = The Units statements for your continuous variable covariates.
- `ntotal = .` = Designating that the sample size of the study is 832.
- `power = .;` = The "." Is telling SAS that this is the value we want it to detect (power = detected power)
- `run;` = Run the Proc Power statement.

Table 19.04: Power calculator with selected explanations for multivariate analysis



The POWER Procedure
Likelihood Ratio Chi-Square Test for One Predictor

Fixed Scenario Elements	
Method	Shieh-O'Brien approximation
Response Probability	0.25
Test Predictor	momfrac
Odds Ratio for Test Predictor	2
Unit for Test Pred Odds Ratio	1
Covariates	priorfrac armassist raterisk2 age height
Covariate Odds Ratios	2 1.5 1.5 1.5 0.6
Units for Covariate Odds Ratios	1 1 1 10 10
Total Sample Size	832
Total Number of Bins	1600
Alpha	0.05

Computed Power
Power
0.778

Inputted information from the above code.
Calculated power from the above information.

Matched Case Control Studies

When performing Matched Case control studies, we need to think about our analyses differences. In the situations in which Cases are matched with controls on certain variables, The data needs to be set-up differently, the traditional Goodness of Fit (GOF) Tests do not apply because they rely on Chi-square analyses, which cannot be performed with a matched case-control analyses.

Terminology

Below is a list of terminology associated with Matched case-control study analyses that you should be aware of.

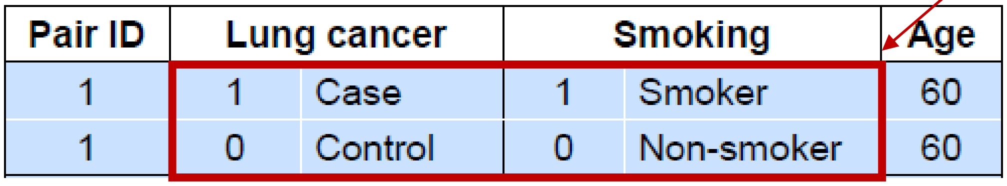
Matched Data

When Controls are matched with cases based on one or more confounders (i.e age or smoking status).

Discordant pairs

When matched pairs have different values for the main variable of interest.

Table 20.01: Example if Discordant pair as used in analyses matched for age.


Discordant pair

Pair ID	Lung cancer		Smoking		Age
1	1	Case	1	Smoker	60
1	0	Control	0	Non-smoker	60

Concordant pair

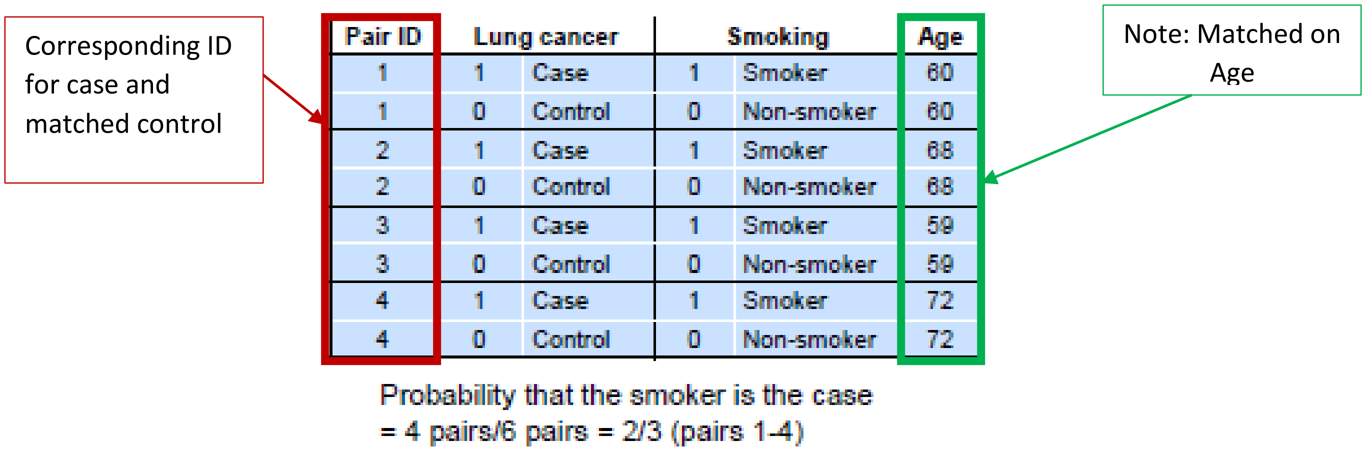
When matched pairs have the same values on the main variable of interest.

NOTE: unfortunately these values cannot be used for analyses for estimating coefficients.

Setting up Matched Case–Control study data

If the cases and control have been matched, it is important to set-up your data differently as a way to indicate they have been matched. In the instance of cases and control that have been matched, you assign them the same id number and put them right next to each other in the dataset. An example is shown below:

Table 20.02: Example dataset of matched Case-Control study data with selected explanations.


Corresponding ID for case and matched control
Note: Matched on Age

Pair ID	Lung cancer		Smoking		Age
1	1	Case	1	Smoker	60
1	0	Control	0	Non-smoker	60
2	1	Case	1	Smoker	68
2	0	Control	0	Non-smoker	68
3	1	Case	1	Smoker	59
3	0	Control	0	Non-smoker	59
4	1	Case	1	Smoker	72
4	0	Control	0	Non-smoker	72

Probability that the smoker is the case
= 4 pairs/6 pairs = 2/3 (pairs 1-4)

Proc Logistic for Matched Case Control study

Below you will find the code for of how to get MAximimum likelihood estimates and Odds ratios for Matched case control studies. But first, here is the dataset we are using for this code:

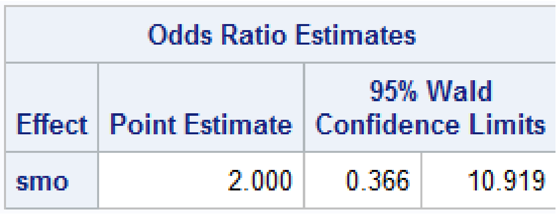
```
data test; input pair lc smo age;
cards;
1 1 1 60
1 0 0 60
2 1 1 68
2 0 0 68
3 1 1 59
3 0 0 59
4 1 1 72
4 0 0 72
5 1 0 83
5 0 1 83
6 1 0 69
6 0 1 69
run;
```

On this dataset, here is the code in order to get results for matched case-control datasets:

```
proc logistic descending data=test;
model lc=smo;
strata pair;
run;
```

- `proc logistic` = run proc logistic function
- `descending data=test;` = sort the data in descending order (The data is "test" as created from the cards statement above)
- `strata pair;`= indicating that the matched pairs are in the variable called "pair" and that is the matched variable
- `run;` ;= run proc logistic function

Table 20.03: Odds ratio for the paired case-control analyses.



Odds Ratio Estimates			
Effect	Point Estimate	95% Wald Confidence Limits	
smo	2.000	0.366	10.919