

Class Review

STAT 3505

Week 14 (April 18, 2024)

Gunes Fleming Ph.D.



TIPS AND TRICKS FOR RUNNING SAS

- Statements can begin and end anywhere
- Statements can continue over several lines, **ends with semi-colon**
- Several statements may be on the same line
- **Blanks**, as many as you want but at least one, separating the components (words) in a SAS program statement.
- **Case**, (lower and upper) **doesn't matter** in most SAS statements.
- **Case does make a difference in data and quoted information** . (such as M or m for “MALE” or “male”).



TIPS AND TRICKS FOR RUNNING SAS

- The most common error in SAS programming is a **misplaced (or missing) semicolon** .
- A second common error is a **missing RUN;** statement.
- A third common error in a SAS program is the presence of **unbalanced quotation marks**.
- Look for errors in a program log from the top down.
- Make the structure of your SAS programs easy to read.



File Extensions Referenced

- **SAS code file - (filename.sas)**
- **SAS log file - (filename.log)**
- **SAS listing file - (filename.lst)**
- **SAS data file - (filename.sasb7dat)**
- **Raw data files - (filename.dat or filename.txt or filename.csv)**
- **Excel file - (filename.xls or filename.xlsx)**



Programming Language

Rules for SAS Statements:

- **SAS statement MUST end with a semi-colon (;)!!!**
- SAS statements are not case-sensitive
- SAS statements may begin anywhere on a line and can be continued on more than one line
- You can write several SAS statements on a single line
- Words in SAS statements are separated by blanks or special characters



The SAS Data Step

- The DATA step is used to define a SAS data set, and to manipulate it and prepare it for analysis.
- In the SAS language, the DATA statement signals the creation of a new data set.

For example, a DATA statement in SAS code may look like this:

```
DATA MYDATA;
```



DATA Statement

- Signals the beginning of the DATA step.
- Assigns a name (that you chose) to the data set created in the DATA Step. The general form of the DATA statement is:
 - **DATA *datasetname*;**



SAS Basic Concepts

- SAS **data set names** must be as per following rules:
 - should be 1 to 32 characters in length and must not include any blank spaces
 - should begin with a letter (A-Z, including mixed case characters) or an underscore (_)
 - continue with any combination of numbers, letters, or underscores



SAS Basic Concepts

- SAS **variable names** must be as per following rules:
 - should be 1 to 32 characters in length and must not include any blank spaces
 - should begin with a letter (A-Z, including mixed case characters) or an underscore (_)
 - continue with any combination of numbers, letters, or underscores
 - may include upper- and lower- case characters (not case sensitive)
 - Recommended: should be descriptive of the variable



SAS Variable Types

- **Numeric variables (Default)** – A numeric variable is used to designate values that could be used in arithmetic calculations or are grouping codes for categorical variables. For example, the variables SBP (systolic blood pressure), AGE, and WEIGHT are numeric.
- **Character (text, string) variables** – Character variables are used for values that are not used in arithmetic calculations. For example, a variable that uses M and F as codes for gender would be a character variable. Some “number” are best represented as character -- social security, telephone number, etc. When designating a character variable in SAS, you must indicate to SAS that it is of character type. This is illustrated in upcoming examples.
- **Date Variables** – In SAS, a date value is stored in a special way. Technically, dates are stored as integers and specifically as the number of days since January 1, 1960. (More about this later.)



Methods of Reading Data Into SAS

- You can read data using
 - Freeform list input
 - Compact method
 - Column input
 - Formatted input



Freeform Data Entry

Open the file **DFREEFORM.SAS**

```
DATA MYDATA;  
INPUT ID $ SBP DBP GENDER $ AGE WT;  
DATALINES;  
1 120 80 M 15 115  
2 130 70 F 25 180  
3 140 100 M 89 170  
4 120 80 F 30 150  
5 125 80 F 20 110  
;  
PROC PRINT;  
RUN;
```

The INPUT statement defines the variables (some character, designated by \$ after name)

DATALINE indicates that data are listed next.

The data must match the INPUT statement – the same number of values per line, separated with blanks.

DATA ends with a semicolon.



Freeform Data Entry

Advantages:

- Easy, very little to specify.
- No rigid column positions which makes data
- Entry easy.
- If you have a data set where the data are separated by blanks, this is the quickest way to get your data into SAS.



Freeform Data Entry

Restrictions:

- Every variable on each data line must be in the order specified by the INPUT statement.
- Fields must be separated by at least one blank.
- Blank spaces representing missing variables are not allowed. If there are missing values in the data, a dot (.) should be placed in the position of that variable in the data line. For example, a data line with AGE missing might read:

```
4 120 80 F . 150
```

- No embedded blanks are allowed within the data value for a character field, like *MR ED*.
- A character field has a default maximum length of 8 characters in freeform input.



Compact Form

- This version of freeform input allows you to have several subjects' data on a single line. (Often used in textbooks to save space.)
- The data can be compacted using an @@ option in the INPUT statement to tell SAS to continue reading each line until it runs out of data.

```
DATA WEIGHT;  
INPUT TREATMENT LOSS @@;  
DATALINES;  
1 1.0 1 3.0 1 -1.0 1 1.5 1 0.5 1 3.5  
2 4.5 2 6.0 2 3.5 2 7.5 2 7.0 2 6.0 2 5.5  
3 1.5 3 -2.5 3 -0.5 3 1.0 3 .5  
;  
PROC PRINT;  
RUN;
```

The @@ symbol in the INPUT statement tells SAS to allow multiple rows of data on each line. You must be careful that the data matches the input definition.



Compact Form

- Realize the addition of the @@ indicator, otherwise this is same as the first entry method.
- Therefore, this data entry technique has the same benefits and restrictions as the previous freeform input method.



Column Input

- Should be used when data consist of fixed columns of values that are not necessarily separated by blanks.
- SAS allows you to specify which columns in the raw data set contain each variable's values. You must know the **starting column** and **ending columns** for each variable's values.
- We need to include of column ranges telling SAS where in the data set to find the information for each variable. Input would be in the following form:

INPUT variable startcol-endcol ...;



Column Input

```
DATA MYDATA;  
INPUT ID $ 1 SBP 2-4 DBP 5-7 GENDER $ 8 AGE 9-10 WT  
11-13;  
DATALINES;  
1120 80M15115  
2130 70F25180  
3140100M89170  
4120 80F30150  
5125 80F20110  
;  
RUN;  
PROC MEANS;  
RUN;
```

Note how data are in specific columns. In the INPUT statement, the columns are specified by the ranges following a variable name.

INPUT variable startcol-endcol ...;

- Note that the \$ after ID and GENDER specifies that they are character-type [text] variables.
- In the INPUT statement that each variable name is followed by a number or a range of numbers that tells SAS where to find the data values in the data set.

Column Input

Advantages:

- Data fields can be defined and read in any order in the INPUT statement, and unneeded columns of data can be skipped.
- Blanks are not needed to separate fields.
- For character data values, embedded blanks are no problem (e.g., John Smith).
- Character values can range from 1 to 200 characters. For example:

INPUT DIAGNOSE \$ 1-200;

- Only the variables needed can be read and the rest can be skipped. This is handy when your data set (like downloaded from a large database) contains variables you're not interested in using. Read only the variables you need.



Column Input

Restrictions:

- Data values must be in fixed column positions.
- Blank fields are read as missing.
- Column input has more specifications than list input (freeform and compact entries). You must specify the column ranges for each variable.



Formatted Data Entry

- Another technique of reading in data from specific columns. It allows you to specify information about how SAS will interpret data as they are read into the program.
- SAS uses the term **INFORMAT** to specify an input specification that tells it how to read data. (A **FORMAT** specification tells SAS how to output data into a report, this will be discussed later.)
- Using an **INFORMAT** is helpful when you are reading data that might be in an otherwise hard-to-read format, such as date and dollar values.
- Syntax:
DATA MYDATA;
INPUT @col variable1 format. @col variable2 format. ...;



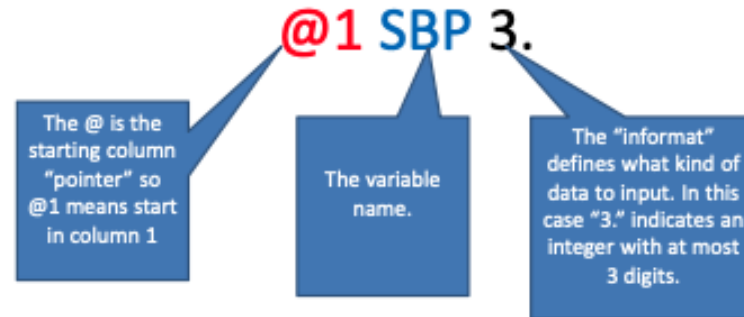
Formatted Data Entry

- Example:

```
DATA MYDATA
```

```
INPUT @1 SBP 3. @4 DBP 3. @7 GENDER $1. @8 WT 3. @12 OWE COMMA9.;
```

- Three Components:



Note: The `COMMA9.` INFORMAT tells SAS to read in numbers that contain dollar signs and commas and to convert the values into numbers. More on this later.



Formatted Data Entry

```
DATA MYDATA;  
INPUT @1 SBP 3. @4 DBP 3. @7 GENDER $1. @8 WT 3.  
@12 OWE COMMA9.;  
DATALINES;  
120 80M115 $5,431.00  
130 70F180 $12,122  
140100M170 7550  
120 80F150 4,523.2  
125 80F110 $1000.99  
;  
PROC PRINT;  
RUN;
```



Notes on Formats

- Formats usually end with a dot (.) or a dot followed by a number
 - 5.** : A number up to five digits, no decimals, so could take on values from -9999 to 99999.
 - 5.2** : A number up to five digits, and up to 2 decimals – so could take on values from -9.99 to 99.99
 - \$5.** : A character value of up to five digits, such as ABCDE, abcde, 12345 or (*&6%



Formatted Data Entry

Advantages and Restrictions:

- Advantages and restrictions are similar to those for column input.
- The primary difference is the ability to read in data using INFORMAT specifications.
- Is particularly handy for reading dates and dollar values.



Working with SAS Data Sets

- **Work** vs **Permanent** Data sets
- All of the data sets we've created have been “**Work**” data sets.
- **Work** datasets vanish when you end a SAS session.
- However, we can create permanent data sets
 - First, we need a location (**library** or folder)
 - Then, we store data sets in that **library** (or folder)



Temporary Vs. Permanent Data Sets

- A temporary SAS data set is named with a single level name such as MEASLES, or MAR2000. (technically the names are WORK.MEASLES and WORK.MAR2000)
- A permanent SAS data set is a file saved on your hard disk (Base SAS) or a [folder in server](#).
- There are two ways to refer to a permanent data set:

We can refer to a permanent SAS data set using a Windows filename such as “C:\SASDATA\SOMEDATA” or “C:\RESEARCH\MEASLES2009”.

Or by using a SAS library prefix: **MYSASDATA.SOMEDATA** or RESEARCH.MEASLES2009 where

MYSASDATA library name = “C:\SASDATA”

RESEARCH library name = “C:\RESEARCH”



Create a Library for this Class

- libname STAT3505 '~/my_shared_file_links/u63742093';
- To create a library, in general:

LIBNAME *yourlibraryname* "LOCATION";

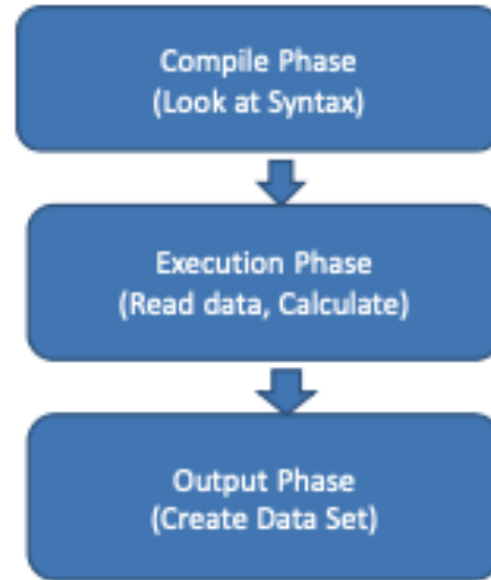


How does SAS think?

SAS data step overview:

When you submit a DATA step, SAS software processes the DATA step and creates a new SAS data set.

Let's see exactly how that happens.



How does SAS think?: Compilation Phase

- SAS Data step is processed in two distinct phases:

Compilations Phase

Execution Phase

Descriptor Portion

Data Portion

Compile Phase

```
DATA NEW;  
INPUT ID $ AGE TEMPC;  
TEMPF=TEMPC*(9/5)+32;  
DATALINES;  
0001 24 37.3  
0002 35 38.2  
;  
run;  
proc print;run;
```

SAS Checks the syntax of the program.

- Identifies type and length of each variable
- Does any variable need conversion?

If everything is okay, proceed to the next step.

If errors are discovered, SAS attempts to interpret what you mean. If SAS can't correct the error, it prints an error message to the log.



How does SAS think?: SAS Basic Concepts

- SAS Data step is processed in two distinct phases:
- During the **compilation phase**, each statement is scanned for syntax errors. Most syntax errors prevent further processing of the DATA step.
- If the DATA step compiles successfully, then the **execution phase** begins.

Parts of a SAS Data Set

- A SAS data set is a tabular collection of data (as variables and observations) whose contents are in a SAS file format. A SAS data set includes metadata that describes its attributes.
- The metadata (also known as *descriptor* information) make the file self-documenting. SAS can obtain the attributes directly from the data set.



Parts of a SAS Data Set

- **Descriptor Portion:**

- The descriptor portion of a dataset contains information about the dataset like: Name of dataset, date and time the dataset was created, number of Observations, number of variables.
- A example of SAS descriptor portion is as follows:

Data Set Name:	WORK.TEST1	Observations:	5
Member Type:	DATA	Variables:	4
Engine:	BASE	Indexes:	0
Created:	2:38 Tuesday, December 7, 2004	Observation Length:	32
Last Modified:	2:38 Tuesday, December 7, 2004	Compressed:	NO

- Try:

```
proc contents data=data2_weight;  
run;
```



How does SAS think?: SAS Basic Concepts

- SAS Data step is processed in two distinct phases:
- During the **compilation phase**, each statement is scanned for syntax errors. Most syntax errors prevent further processing of the DATA step.
- If the DATA step compiles successfully, then the **execution phase** begins. A DATA step executes once for each observation in the input data set, unless otherwise directed

How does SAS think?: SAS Basic Concepts

- When DATA step statements are compiled, SAS determines whether to create an input buffer.
- Input buffer, an area of memory, is created to hold a record from the external file.
- If the input file contains raw data, SAS creates an input buffer to hold the data before moving the data to the program data vector (PDV).
- If the input file is a SAS data set, however, SAS does not create an input buffer. SAS writes the input data directly to the PDV.
- The term input buffer refers to a logical concept and does not necessarily reflect the physical storage of data.
- Then the program data vector is created.



How does SAS think?

- SAS Data step is processed in two distinct phases:

Compilations Phase

Execution Phase

Descriptor Portion

Data Portion

Execution Phase

- SAS creates an input buffer
- INPUT BUFFER contains data as it is read in

```
DATALINES;  
0001 24 37.3  
0002 35 38.2  
;
```



1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	1		2	4		3	7	.	3

- PROGRAM DATA VECTOR (PDV) is created and contains information about the variables

N	_ERROR_	ID	AGE	TEMPC	TEMPF
1	0		.	.	.

- Two automatic variables `_N_` and `_ERROR_` and a position for each of the four variables in the DATA step.
- Sets `_N_ = 1` `_ERROR_ = 0` (no initial error) and remaining variables to missing.



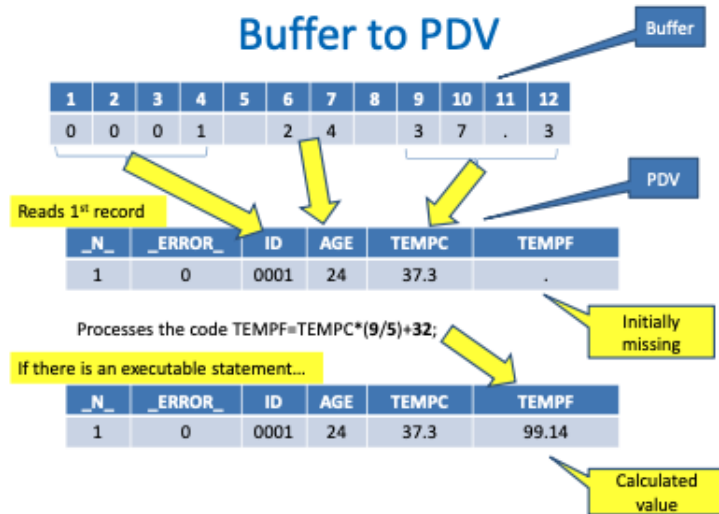
How does SAS think?: SAS Basic Concepts

Program Data Vector (PDV):

- PDV is a logical area in memory where SAS builds a data set, one observation at a time.
- When a program executes, SAS reads data values from the input buffer or creates them by executing SAS language statements. The data values are assigned to the appropriate variables in the program data vector. From here, SAS writes the values to a SAS data set as a single observation.
- Along with data set variables and computed variables, the PDV contains two automatic variables, `_N_` and `_ERROR_`.
- The `_N_` variable counts the number of times the DATA step begins to iterate.
- The `_ERROR_` variable signals the occurrence of an error caused by the data during execution. The value of `_ERROR_` is either 0 (indicating no errors exist), or 1 (indicating that one or more errors have occurred). SAS does not write these variables to the output data set.

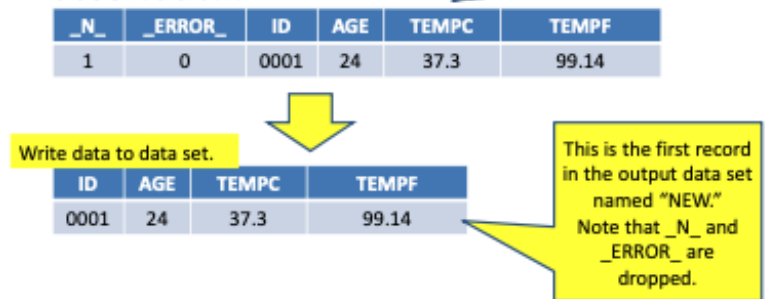


How does SAS think?: Execution Phase



Output Phase

- The values in the PDV are written to the output data set (NEW) as the first observation:



How does SAS think?: Execution Phase

- At the end of the execution phase, the SAS log confirms that the raw data file was read and displays the number of observations and variables in the data set.

NOTE: 9 records were read from the infile Myfile.

NOTE: The data set MYLIB.TEST has 5 observations and 5 variables.

Describing Data

- **Proc Means**: Calculates simple descriptive statistics for quantitative data.

```
PROC MEANS DATA= dataname options;
```

```
VAR varlist;
```

```
RUN;
```

- Some commonly used options:
N, NMISS, MEAN, MEDIAN, STD, Q1, Q3 etc.



Describing Data

- **Proc Freq**: Calculates simple descriptive statistics for categorical data by producing table of frequencies (counts, percentages etc.).

```
PROC FREQ DATA= dataname;  
  TABLES var(s)/options;  
RUN;
```

- Some commonly used options:
NOCUM, NOPERCENT, MISSING etc.



Describing Data

- **PROC PLOT/GPLOT**: Plot variable1 against variable2. Generates scatter plot(s)

PROC PLOT (or GPLOT) DATA= *dataname*;

PLOT *var1*var2*; **var1 is plotted on y-axis and var2 on x-axis*;

RUN;



Reading in External Data, Formatting and Manipulating Data Sets

INFILE STATEMENT

- Reads in external files from the computer/server.
- Data Step Syntax with INFILE statement:
DATA datasetname;
 INFILE filespecification options;
 INPUT variablelist;
RUN;
- INFILE statement frequently used with external dataset path name.:
 INFILE "EXTERNAL INPUT FILE PATH";



INFILE STATEMENT

- When INFILE statement is used to read external data in, there is no need for the DATALINES statement.
- INFILE statement must appear before INPUT statement that reads data lines.
- There are a number of INFILE statement options available to help read data into SAS in the exact form that you intend to read.



INFILE STATEMENT

- Some INFILE statement options are:
 - DLM: allows you to define a delimiter to be something other than a blank
 - FIRSTOBS: tells SAS what line should it start reading the raw data from
 - OBS: indicates which line in your raw data file should be treated as the last record to be read by SAS



INFILE STATEMENT

- Example 1:

```
DATA import_cars_txt1;  
  INFILE '~/my_shared_file_links/u63742093/cars_blank_txt.txt' FIRSTOBS=2;  
  INPUT make $ type $ origin $ MPG_Highway MPG_City EngineSize;  
RUN;
```

- Example 2:

```
DATA import_cars_txt2;  
  INFILE '~/my_shared_file_links/u63742093/cars_comma_txt.txt' DLM=',' FIRSTOBS=2 OBS=26;  
  INPUT make $ type $ origin $ MPG_Highway MPG_City EngineSize;  
RUN;
```



PROC IMPORT

- Imports external files such as excel, CSV, text etc. into SAS data sets.

- Syntax:

```
PROC IMPORT
    DATAFILE = "EXTERNAL INPUT FILE PATH"
    OUT = LIBNAME.DATASETNAME
    DBMS = FILETYPE
    REPLACE;
    <OPTIONS >;

RUN;
```

- REPLACE is a commonly used PROC IMPORT option. Tells SAS to replace an existing dataset with the same NAME.

- DBMS

- (As an alternative importing tool, Import Wizard can be used in SAS Base).



PROC IMPORT

- Commonly used options:
 - **SHEET**: Specifies the Excel worksheet to import (for Excel files).
 - **GETNAMES**: Specifies that the first row of the input file contains variable names. (YES is default).
 - **DATAROW**: Specifies the starting row in the input file where data begins.
 - **GUESSINGROWS**: Specifies the number of rows to use when determining variable types and lengths. (Default is 20).
 - **DELIMITER**: Specifies the delimiter for delimited files (for DBMS=DLM).



PROC IMPORT - Example

- Import a CSV file:

```
proc import
    datafile = "~/my_shared_file_links/u63742093/hdp.csv"
    out = work.data1_hdpimport
    dbms = CSV
    replace;
run;
```



PROC IMPORT - Example

- Import a txt with a | delimiter file:

```
proc import
  datafile = "~/my_shared_file_links/u63742093/senators.txt"
  out = data2_senatorimport
  dbms = DLM /* default delimiter is blank */
  replace;
  delimiter = "|"; * We must indicate the delimiter as it is different than the default;
run;
```



PROC IMPORT - Example

- Import an xlsx file:

```
proc import
    datafile = "~/my_shared_file_links/u63742093/Anamoly_Detection.xlsx"
    out = data3_AnDetimport
    dbms = XLSX
    replace;
    sheet = "Raw Data"; * Sheet name that we want to import;
run;
```



PROC EXPORT

- Exports SAS files as external files such as excel, CSV, text etc.

- Syntax:

```
PROC EXPORT
```

```
DATA = LIBNAME.DATASETNAME
```

```
OUTFILE = "EXTERNAL OUTPUT FILE PATH\OUTPUTFILENAME.EXTENSION"
```

```
DBMS = FILETYPE
```

```
REPLACE;
```

```
<OPTIONS >;
```

```
RUN;
```

- REPLACE is a commonly used PROC EXPORT option. Tells SAS to replace an existing dataset with the same name.



PROC EXPORT - Example

- Export as an xlsx file:

```
proc export
  data = stat3505.salary
  outfile = "~/my_shared_file_links/u63742093/SalaryExported.xlsx"
  dbms = xlsx
  replace;
run;
```



PROC EXPORT - Example

- Export as a txt file with a comma delimiter:

```
proc export
  data = stat3505.salary
  outfile = "~/my_shared_file_links/u63742093/SalaryExported2.txt"
  dbms = TAB
  replace;
  delimiter = ',';
run;
```

- Note that DBMS=TAB indicates that we want to export the data as a text file, and the tab character will be used as a delimiter.

Although it says "TAB," it's actually using a comma as the delimiter here.



FORMAT/ INFORMAT

- Informats – tell SAS how the data should be read in.
 - Formats – tell SAS how to print/display data.
-
- All informats and formats end with a dot (.) or contain a dot (.)
 - The number in SAS format/informat tells SAS the number of characters to read in



FORMAT/ INFORMAT

- A FORMAT statement within the DATA step is used to tell SAS how data will be presented in output.
- The FORMAT statement, which appears in the DATA step, uses the following syntax:

FORMAT VAR1 FORMAT1. VAR2 VAR3 VAR4 FORMAT2.etc... .;

- Example:

FORMAT BMI 5.2 name \$10.;



FORMAT/ INFORMAT

- Example:

```
DATA data4;  
  INPUT @1 NAME $11. @12 BDATE DATE9.;  
  FORMAT BDATE WORDDATE12.; * Assigns a format to display BDATE;  
DATALINES;  
Bill      08JAN1952  
Jane      02FEB1953  
Clyde     23MAR1949  
;  
proc print data=data4;  
run;
```

- [Date/time formats](#)



PROC FORMAT

- So far, we have only used pre-defined SAS formats.
- PROC FORMAT allows creating custom formats.
- These custom formats allow you to specify the information to be displayed for selected values of a variable.
- Once a FORMAT has been created, it must then be applied within a SAS procedure to take effect.
- The steps for using formatted values are:
 - Create a FORMAT definition in PROC FORMAT.
 - Apply the FORMAT to one or more variables in a DATA or PROC statement.



PROC FORMAT

- A user-generated format name should:
 - A numeric format name can be up to 32 characters in length.
 - A character format name can be up to 31 characters in length.
 - The name can include letters, numbers, and underscores.
 - Begin with a letter or an underscore and should not end with a number.
 - Not be same as SAS reserved words or keywords such as FORMAT, PROC, etc.
 - Not have special characters.



Modify/Manipulate Data in Data Step (See Examples in SAS)

- Some statements that we can and will use in the examples are:
 - KEEP
 - DROP
 - LABEL
 - RENAME
 - Using WHERE or IF for subsetting data.



Working with Dates, Creating Summary Datasets and Some SAS Functions

Modify/Manipulate Data in Data Step: **LABELS**

- In order to add variable labels, LABEL statement is used.
- Syntax:

```
LABEL  variablename_1 = 'description'  
       variablename_2 = 'description'  
       ...  
       variablename_n = 'description';
```

LABELS

- Labels can be up to 40 characters. Each blank counts as a character.
- Single or double quotes can be used for description (do not mix single and double quotes).
- LABEL statement can be placed anywhere in DATA step.



Variable LABELs:

Example:

Data *newdata*;

Set *olddata*;

Keep AGE WGTBL HGTBL BMI;

Label AGE = “Age at Baseline”

WGTBL = “Weight at Baseline”

HGTBL = “Height at Baseline”

BMI = “Body Mass Index”;

Run;



About SAS Functions

- The syntax for a function is:

NEWVARIABLE = FUNCTIONNAME(*argument1, argument2.., argumentk*);

where arguments can be constants, variables, expressions. or other functions.



SAS DATE AND TIME FUNCTIONS



DATE and TIME Functions

- SAS date and time variables are stored as integers and indicate the number of days since January 1, 1960.
- A positive number indicates a date after January 1, 1960, and a negative number indicates a date before January 1, 1960.
- Date values that contain both date and time are stored in SAS as the number of seconds since midnight on January 1, 1960

DATE and TIME Functions

Two ways to designate variables as a date value:

1. Read a value as a date in an INPUT statement.

For example:

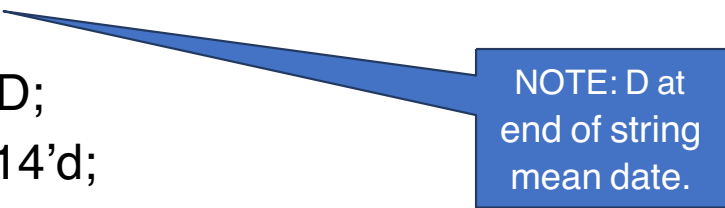
```
INPUT BDATE MMDDYY8.;
```

2. Assign a date value to a fixed date.

A “d” at the end of the value tells SAS to interpret this quoted string as a date.

For example:

```
BDATE = '12DEC2010'd;  
BEGINDATE="1jan2011"D;  
EXAMDATA='13-APR-2014'd;
```



NOTE: D at
end of string
mean date.



Difference in Dates

- Since SAS stores dates in number of days from January 1, 1960, the difference between two dates would be calculated in **days**.

Ex:

```
data data2;  
    bdate = '10JUL2014'd;  
    adodate = '01AUG2018'd;  
    diff_days = adodate - bdate;  
    format adodate bdate date9.;  
run;
```



Difference in Dates

- DATDIF Function returns the number of days between two dates.
- **Syntax:** DATDIF(start-date, end-date, < basis>)
- Examples to basis values:
 - 'ACT/ACT': Actual/actual method. This method calculates the exact number of days between two dates, taking leap years and month lengths into account.
 - 'ACT/360': This method uses the actual number of calendar days in a particular month, and 360 days as the number of days in a year, regardless of the actual number of days in a year.
 - For more info, check SAS [documentation](#).



Difference in Dates

- Examples for DATDIF:

- `DAYS=DATDIF('07JUL1976'd,'01JAN2013'd, '30/360');`

Returns the value (number of days) as 13134. Basis '30/360' specifies a 30-day month and 360-day year regardless of the actual number of calendar days in a month or year.

- `DAYS=DATDIF('07JUL1976'd,'01JAN2013'd, 'ACT/ACT');`

Returns the value (number of days) as 13327.

(Different BASIS are used in different disciplines.)



Difference in Dates

- To get the difference between two dates in **years**:
 - One way is to use arithmetic difference

$$\text{Year_diff} = (\text{Datevar2} - \text{Datevar1}) / 365.25$$

- Another way is to use YRDIF function:

$$\text{Year_diff} = \text{YRDIF}(\text{Datevar1}, \text{Datevar2}, \text{"ACTUAL"})$$



Difference in Dates

- YRDIF function returns the difference in years between two dates, taking into account fractional parts of a year if required.
- Syntax: YRDIF(start-date, end-date, < basis>)
- Examples to basis values:
 - 'ACT/ACT': Actual/actual method. This method calculates the exact number of years between two dates, taking leap years into account.
 - 'AGE': Age method. This method calculates the difference in years based on a 365-day year, ignoring leap years. This method is often used in financial calculations.
 - For more info, check SAS [documentation](#).



Some Other SAS Date/Time Functions:

- MDY(month,day,year): Creates and returns a SAS date from month, day, year.
- INTCK('interval',from, to): Returns the number of time intervals in a given time span, where interval can be DAY, WEEKDAY, YEAR etc.

Example:

```
data data4;  
  input @1 BDATE MMDDYY8.;  
  TARGET = MDY(08,25,2009);  
  AGE = INTCK('YEAR',BDATE, TARGET);  
datalines;  
07101952  
07041776  
01011900  
;
```



Some Other SAS Date/Time Functions:

- Rerun the same example with some changes:

- Change

TARGET = MDY(**08,25,2009**); to

TARGET = '**25-OCT-2009**'d;

- Change

AGE = INTCK('YEAR',BDATE,TARGET); to

WEEKS = INTCK('WEEK',BDATE,TARGET);



Some More SAS Functions:

- **UPCASE**(expression): copies a character expression, converts all lowercase letters to uppercase letters, and returns the altered value as a result.
- **LOWCASE**(expression): copies a character expression, converts all uppercase letters to lowercase letters, and returns the altered value as a result.
- **PROPCASE**(expression): copies a character argument and converts all uppercase letters to lowercase letters. It then converts to uppercase the first character of a word that is preceded by a blank, forward slash, hyphen, open parenthesis, period, or tab.



Special Use Functions

- **INPUT Function:**

- Syntax: `input(original_variable, informat.);`
- Converts a character expression to character or numeric using a specified informat.
- The informat tells SAS how to interpret the data in the original character variable.
- Example:

```
data data9;  
    char_var = '12345678';  
    numeric_var = input(char_var, 8.);  
run;
```

Special Use Functions

- **PUT Function:**

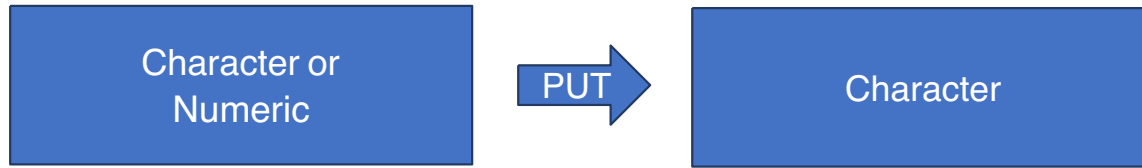
- Syntax: `put(original_variable, format.);`
- Converts a numeric (or character) expression to character using a specified format.
- The informat contains the SAS format that you want applied to the value that is specified in the original variable.
- Example:

```
data data10;  
  var1 = 379.43;  
  var2 = 481.56;  
  
  var1_char = put(var1, 8.2); * Format: 8 characters width, 2 decimal places;  
  var2_char = put(var2, dollar10.2); * Format: Dollar sign, 10 characters width, 2 decimal places;  
run;
```

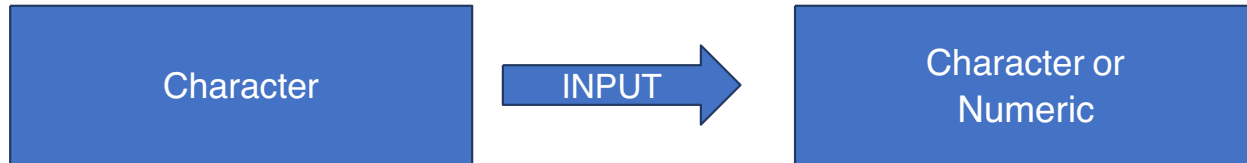


In Short, PUT and INPUT Functions:

- **PUT** – Converts from Character or Numeric to Character



- **INPUT** – Converts from Character to Character or Numeric



If original value is NUMERIC – you must use the PUT function.

If you want result to be NUMERIC -- you must use the INPUT function.

CREATING SUMMARY DATA SETS



CREATING SUMMARY DATASET WITH PROC MEANS

- We need to include OUTPUT statement in PROC MEANS to create a summary dataset.
- Syntax:

```
proc means data=your_dataset;  
    var var1 var2 ... vark; /*Numeric variables to calculate summary statistics of;  
    output out = output_dataset_name /* Specify the output dataset and the desired statistics to be saved */  
        mean = mean_var1 mean_vari2 ...;  
run;
```
- Note/recall that you can specify additional options for the desired statistics such as mean, sum, min, max, median, etc. If no statistics are specified, PROC MEANS calculates the default statistics: N, Mean, Std Dev, Minimum, and Maximum.



CREATING SUMMARY DATASET WITH PROC MEANS

- Example:

```
proc means data=datallupdated;
  class teacher;
  var pretest posttest change;
  output out=statdata
    mean = mean_pre mean_post mean_chg
    min  = min_pre min_post min_chg
    max  = max_pre max_post max_chg;
run;
```

- Output:

*Week 5 Prep.sas x WORK.STATDATA x

View: Column names Filter: (none)

Total rows: 6 Total columns: 12 Rows 1-6

	teacher	_TYPE_	_FREQ_	mean_pre	mean_post	mean_chg	min_pre	min_post	min_chg	max_pre	max_post	max_chg
1		0	25	68.92	83.36	14.44	38	63	-6	93	95	38
2	Black	1	5	73.4	86.8	13.4	41	79	-3	91	95	38
3	Charlotte	1	5	62.2	79.4	17.2	44	67	3	71	87	24
4	Jones	1	5	64.8	80.4	15.6	38	63	-1	93	92	25
5	Smith	1	5	72	84.4	12.4	56	77	-6	90	92	24
6	Wong	1	5	72.2	85.8	13.6	53	77	4	85	91	26



CREATING SUMMARY DATASET WITH PROC MEANS

- Output:

View: Column names | Filter: (none)

Total rows: 6 Total columns: 12

teacher	_TYPE_	_FREQ_	mean_pre	mean_post	mean_chg	min_pre	min_post	min_chg	max_pre	max_post	max_chg
	0	25	68.92	83.36	14.44	38	63	-6	93	95	38
2 Black	1	5	73.4	86.8	13.4	41	79	-3	91	95	38
3 Charlotte	1	5	62.2	79.4	17.2	44	67	3	71	87	24
4 Jones	1	5	64.8	80.4	15.6	38	63	-1	93	92	25
5 Smith	1	5	72	84.4	12.4	56	77	-6	90	92	24
6 Wong	1	5	72.2	85.8	13.6	53	77	4	85	91	26

- Note that

- `_FREQ_` gives number of observations (missing or non-missing)
- `_TYPE_` : the first observation with 0 is mean of all non-missing

the observations with 1 are means broken down by the class variable Teacher.



CREATING SUMMARY DATASET WITH PROC MEANS

- To get only the stats for each Teacher category, use NWAY option:

```
proc means data=datallupdated nway noprint;  
  class teacher;  
  var pretest posttest change;  
  output out=statdata2  
    mean = mean_pre mean_post mean_chg  
    min = min_pre min_post min_chg  
    max = max_pre max_post max_chg;  
run;
```

- Output:

Table: | View: | | Filter: (none)

Total rows: 5 Total columns: 12

◀ ◀ Rows 1-5 ▶ ▶										
TYPE	_FREQ_	mean_pre	mean_post	mean_chg	min_pre	min_post	min_chg	max_pre	max_post	max_chg
1	5	73.4	86.8	13.4	41	79	-3	91	95	38
1	5	62.2	79.4	17.2	44	67	3	71	87	24
1	5	64.8	80.4	15.6	38	63	-1	93	92	25
1	5	72	84.4	12.4	56	77	-6	90	92	24
1	5	72.2	85.8	13.6	53	77	4	85	91	26



Analyzing Counts and Tables, Merging, Concatenating and Transposing Data Sets

Analyzing Counts and Tables: **PROC FREQ**

- PROC FREQ is a multi-purpose procedure in SAS for analyzing count data.
- It can be used to obtain frequency counts for one or more single variables or to create two-way tables (crosstabulations) from two variables.
- PROC FREQ can also be used to perform statistical tests on count data.



Analyzing Counts and Tables: **PROC FREQ**

- PROC FREQ can also be used to perform statistical tests on count data.
- Syntax:
 - **PROC FREQ** <options(s)>; <statements> [TABLES](#) requests </ options> ;
 - Common Options
 - **DATA =** Specifies which data set to use.
 - **ORDER=** Specifies the order results is listed in the output table (FREQ, FORMATTED, ORDER (p164)).
 - **PAGE** Specifies that only one table will appear per page. (Not for HTML)
 - **COMPRESS** Begins next table on same page when possible. (Not for HTML)



Analyzing Counts and Tables: **PROC FREQ**

- PROC FREQ can be used for
 - Chi-Square
 - Relative Risk
 - CMH Test
 - Fisher's Exact Test
 - KAPPA
 - etc.



PROC FREQ Statements

- Common Statements
 - **BY**variable(s): Creates tables for each BY group
 - **EXACT**: Produces exact test for selected statistics
 - **OUTPUT=** Creates an output data set containing statistics from an analysis
 - **WEIGHT var**: Identifies a weight variable that contains summarized counts
- The **TABLES** statement is required for all of the examples in this section.
Its format is:

TABLES <variable-combinations/options>;

where variable-combinations specify frequency or crosstabulation tables.



TABLES Statement

Description	TABLE statement
Specify counts for a single table	TABLE A;
Specify a crosstabulation between two variables	TABLE A*B;
Specify several tables	TABLE A*B B*C X*Y; Also, TABLE A*(B C D); is the same as TABLE A*B A*C A*D;
Use a range of variables in a TABLE statement	TABLE (A - - C)*X; is the same as TABLE A*X B*X C*X;



Options for TABLES Statement

- Options for the TABLES statement follow a slash (/).

For example,

```
TABLES A*B / CHISQ;
```

- Requests that the chi-square and related statistics be reported for the crosstabulation A*B.



Options for TABLES Statement

- **AGREE** Request KAPPA statistic (inter-rater reliability) To include significance tests, add the following statement:
 - **TEST** option. For example: TABLES A*B / /AGREE; TEST KAPPA;
- **CHISQ** Requests chi-square and related tests
- **RELRISK** Requests relative risk calculations
- **FISHER** Requests Fisher's Exact test for tables greater than 2x2
- **SPARSE** Requests all possible combinations of variable levels
- **MISSING** Requests that missing values be treated as non-missing
- **CELLCHI2** Displays the contribution to chi-square for each cell
- **NOCOL** Suppresses column percentages for each cell
- **NOCUM** Suppresses cum. freq. and cum. % in one-way table
- **NOFREQ** Suppresses frequency count for each cell
- **NOPERCENT** Suppresses row %, and column % in tables
- **NOPRINT** Suppresses table display
- **NOROW** Suppresses row percentages



Testing Goodness-of-Fit in a One-Way Table

- A goodness-of-fit test of a single population is a test to determine if the distribution of observed frequencies in the sample data closely matches the expected number of occurrences under a hypothetical distribution for the population.
- The observations are assumed to be independent, and each data value can be counted in one and only one category.
- It is also assumed that the number of observations is fixed.
- The hypotheses being tested are:
 - H_0 : The population follows the hypothesized distribution.
 - H_a : The population does not follow the hypothesized distribution.
- A chi-square statistic is calculated, and a decision can be made based on the p-value associated with that statistic



Testing Goodness-of-Fit in a One-Way Table

- Example:

```
PROC FREQ data = carsdata2 order=freq; *descending order;  
    TABLES nationality/ nocum chisq testp=(0.7 0.15 0.1 0.05); * expected proportions;  
RUN;
```

(This test is Pearson chi-square goodness-of-fit test.)

- The TESTP= and TESTF= options in PROC FREQ specify the null hypothesis (expected) proportions or frequencies for a chi-square goodness-of-fit test on a one-way table.
- Note that you must use the ORDER= option to ensure that the hypothesized ratios listed in the TESTP= statement match up correctly with the categories in the table.
- If the TESTP= option is omitted, SAS will assume the proportions within the category are equal. For a categorical variable with 4 possible values, the SAS default would be TESTP = (0.25 0.25 0.25 0.25).



Testing Goodness-of-Fit in a One-Way Table

- Example:

```
PROC FREQ data = carsdata2 order=freq; *descending order;  
    TABLES nationality/ nocum chisq testp=(0.7 0.15 0.1 0.05); * expected proportions;  
RUN;
```

- Result:

The FREQ Procedure			
Nationality	Frequency	Percent	Test Percent
AMERICAN	61028	83.62	70.00
OTHER ASIAN	8033	11.01	15.00
TOP LINE ASIAN	3722	5.10	10.00
MANUAL	200	0.27	5.00

Chi-Square Test for Specified Proportions	
Chi-Square	7722.3995
DF	3
Pr > ChiSq	<.0001

- Notice that in this case, the p-value for the chi-square test is less than 0.05, which leads us to reject the null hypothesis and conclude there is evidence to conclude that the data does not follow the hypothesized distribution.



Another Example: Mendel's Pea Experiment

- We will use data from an experiment conducted by the nineteenth-century scientist Gregor Mendel. According to a genetic theory, crossbred pea plants show a 9:3:3:1 ratio of yellow smooth, yellow wrinkled, green smooth, and green wrinkled offspring.
- Out of 250 plants, under the theoretical ratio (distribution) of 9:3:3:1, you would expect:
 - $(9/16) \times 250 = 140.625$ yellow smooth peas (56.25%)
 - $(3/16) \times 250 = 46.875$ yellow wrinkled peas (18.75%)
 - $(3/16) \times 250 = 46.875$ green smooth peas (18.75%)
 - $(1/16) \times 250 = 15.625$ green wrinkled peas (6.25%)
- After growing 250 of these pea plants, Mendel observed 152 have yellow smooth peas, 39 have yellow wrinkled peas, 53 have green smooth peas, 6 have green wrinkled peas. Do these offspring support the hypothesized ratios? Run a goodness-of-fit test to assess whether the observed phenotypic frequencies seem to support the theory.



Another Example: Mendel's Pea Experiment

- Program:

```
proc freq data=genedata order=data;
  WEIGHT NUMBER;
  TITLE 'GOODNESS OF FIT ANALYSIS - MENDEL EXPERIMENT';
  TABLES COLOR / NOCUM PLOTS=NONE CHISQ TESTP=(0.5625 0.1875 0.1875 0.0625);
run;
```
- Notice here the WEIGHT statement. The WEIGHT statement names a numeric variable that provides a weight for each observation in the input data set. The WEIGHT statement is most commonly used to input cell count data.

- Results: **GOODNESS OF FIT ANALYSIS - MENDEL EXPERIMENT**

The FREQ Procedure			
COLOR	Frequency	Percent	Test Percent
YELLOWSMOOTH	152	60.80	56.25
YELLOWWRINKLE	39	15.60	18.75
GREENSMOOTH	53	21.20	18.75
GREENWRINKLE	6	2.40	6.25

Chi-Square Test for Specified Proportions	
Chi-Square	8.9724
DF	3
Pr > ChiSq	0.0297

Sample Size = 250

- The p-value for the chi-square test is less than 0.05, which leads us to reject the null hypothesis and conclude there is evidence to conclude that the peas do not come from a population having the 9:3:3:1 phenotypic ratios.



Proc Freq Output Data Sets

- PROC FREQ produces two types of output data sets that you can use with other statistical and reporting procedures. You can request these data sets as follows:
- Specify the OUT= option in a TABLES statement. This creates an output data set that contains frequency or crosstabulation table counts and percentages
- Specify an OUTPUT statement. This creates an output data set that contains statistics. More details can be found in [SAS documentation](#).

```
proc freq data=genedata order=data;  
  WEIGHT NUMBER;  
  TITLE 'GOODNESS OF FIT ANALYSIS - MENDEL EXPERIMENT';  
  TABLES COLOR / NOCUM PLOTS=NONE CHISQ TESTP=(0.5625 0.1875 0.1875 0.0625)  
    out=frequency_data;  
  OUTPUT OUT=stats_data chisq; *to include CHISQ includes the chi-square goodness-of-fit test in the outpu  
run;
```



Analyzing Two-Way Tables

- Recall: To create a cross-tabulation table using PROC FREQ for relating two variables, use the TABLES statement with both variables listed and separated by an asterisk (*), (for example, A * B).
- A cross-tabulation table is formed by counting the number of occurrences in a sample across two grouping variables. The number of columns in a table is usually denoted by c and the number of rows by r . Thus, a table is said to have $r \times c$ cells.
- Using Proc Freq, we can conduct a **test of independence** between two variables:

H_0 : The variables are independent (no association between them).

H_a : The variables are not independent.



Analyzing Two-Way Tables

- Example: Data come from a study performed by Karl Pearson in 1909 involving the relationship between criminal behavior and drinking alcoholic beverages. The category “Coining” refers to counterfeiting. For the DRINKER variable, 1 means yes and 0 means no.

```
PROC FREQ DATA=DRINKERS;WEIGHT COUNT;  
    TABLES DRINKER*CRIME/CHISQ;  
    TITLE 'Chi Square Analysis of a Contingency Table';  
    RUN;
```

- Results: The chi-square value is 48.7 and $p < 0.001$. Thus, you reject the null hypothesis of no association (independence) and conclude that there is evidence of a relationship between drinking status and type of crime committed.



Merging/ Concatenating Data Sets

- Concatenating/ Appending data sets with SET statement:
 - Appending data sets combines records from two or more data sets.
 - For example, suppose data are collected at two locations and you want to combine the data sets into one SAS data set for analysis.
 - The data sets must have at least some of the same variables in common.
 - Appending is accomplished by including multiple data set names in the SET statement.
 - Example:

```
DATA NEW;  
  SET OLD1 OLD2;  
RUN;
```

creates the data set NEW, which consists of the records in the OLD1 data set as well as the records from the OLD2 data set.



Merging/ Concatenating Data Sets

- Merging data sets with MERGE statement:
 - We can merge datasets by adding new variables from one data set to an existing data set.
 - To accomplish this, you must have a **matching identifier** in each data set that can be used by SAS to match the records during the merge.
 - For example, suppose you have data collected on patients taken on two separate occasions, and the data for each collection time are in separate data sets. If you want to combine the data so you can compare pre- and post-treatment values, the technique for merging the data sets using some key identifier (such as patient ID) is:
 1. Sort each data set by the key identifier.
 2. Within a DATA step, use the MERGE statement along with a BY statement to merge the data by the key identifier.



Merging/ Concatenating Data Sets

- Example:

Pre-Treatment Data

Obs	case	pretreat
1	1	1.02
2	2	2.10
3	4	2.20
4	5	1.44
5	3	1.88
6	11	1.55
7	13	1.61
8	14	2.61
9	15	1.56
10	16	0.99
11	22	1.53

Post-Treatment Data

Obs	case	posttreat
1	1	1.94
2	2	1.63
3	3	2.73
4	4	2.18
5	5	1.82
6	13	2.25
7	11	1.94
8	14	1.70
9	15	1.78
10	16	1.52
11	22	1.97

(Notice the sort for Case variable – We need to sort both datasets first.)

```
proc sort data=pre; by case; run;
proc sort data=post; by case; run;

data prepost;
  merge pre post;
  by case;
  diff = posttreat - pretreat;
run;
```

- Merged dataset:

Merged Pre- and Post- Treatment Data

Obs	case	pretreat	posttreat	diff
1	1	1.02	1.94	0.92
2	2	2.10	1.63	-0.47
3	3	1.88	2.73	0.85
4	4	2.20	2.18	-0.02
5	5	1.44	1.82	0.38
6	11	1.55	1.94	0.39
7	13	1.61	2.25	0.64
8	14	2.61	1.70	-0.91
9	15	1.56	1.78	0.22
10	16	0.99	1.52	0.53
11	22	1.53	1.97	0.44



Merging/ Concatenating Data Sets

- Merging data sets with MERGE statement:
 - When datasets are merged using the MERGE statement in a DATA step, a given record in one input dataset may not have corresponding counterparts with matching BY variable values in the other input datasets.
 - However, the DATA step merge selects both records with matching BY variable values *as well as* nonmatching records from any input dataset.
 - Any variables occurring only in datasets having no records to contribute for a given BY group will simply be missing.



Merging/ Concatenating Data Sets

- Merging data sets with MERGE statement:
 - To perform matching only for matching records in the DATA step, a common approach is to use the `IN=` dataset option in conjunction with a subsetting IF statement as follows:

```
data mergeddata;  
  merge olddata1(in=xxx) olddata2;  
  by commonvar;  
  if xxx;  
run;
```

- Here, the **IN=** option creates a temporary variable that indicates whether the corresponding dataset contributed to the current observation. We can specify any valid SAS variable name, but here we chose the name `xxx`.
- Any records with a value of `COMMONVAR` variable that did not appear in the `OLDDATA1` dataset will have a value of 0 for `xxx`. Those observations will fail the subsetting IF statement and will not be written to the output dataset.
- The resulting output dataset, `MERGEDDATA`, will have only the matching records desired.



Summarizing Longitudinal Data

- A longitudinal study is a research design that involves repeated observations of the same variables over long periods of time.
- Longitudinal data (sometimes called panel data) is data that is collected through a series of repeated observations of the same subjects over some extended time frame.
- This type of data is especially useful for measuring change over time.
- See VITALS dataset as an example.



Transposing Data Sets: PROC TRANSPOSE

- Proc Transpose allows you to restructure the values in a data set by transposing/ re-orienting the data.

- Syntax:

```
PROC TRANSPOSE DATA=Dataset-name OUT=New-dataset-name;  
    BY variable(s);  
    ID variable;  
    VAR variable(s);  
RUN;
```

- Example:

```
Proc Transpose Data=vitals out=vital_tr;  
    by subjid;  
    var sbp;  
    id visit;  
run;
```



Transposing Data Sets: PROC TRANSPOSE

- The OUT keyword says that the transposed dataset should be created as a new dataset called Vital_tr.
- The BY statement is used to determine the row structure of the transposed dataset. You can include more than one variable in the BY statement. Your data must be sorted on your BY variables before running PROC TRANSPOSE.
 - For long-to-wide transposes, the BY variable(s) should uniquely identify each row.
 - For wide-to-long transposes, the BY variable(s) determine the row structure of the long data; that is, it determines the repetition of the rows.



Transposing Data Sets: PROC TRANSPOSE

- The ID statement assigns names to the transposed value columns that match the values in the variable listed in the ID statement.
 - For long-to-wide transposes, the ID variable(s) determine the structure of the columns in the transposed dataset. There will be one column for each unique value of the ID variable (or if multiple ID variables are present, one column for each unique combination of values).
 - For wide-to-long transposes, you typically do not need an ID variable.
- The VAR statement is where you actually tell SAS what variables you want transposed. These are the values that will appear in the cells of the transposed variables.
 - For long-to-wide datasets, there is usually one variable in the VAR statement.
 - For wide-to-long datasets, there are usually multiple variables in the VAR statement. The resulting dataset will have one row for each variable identified in the VAR statement.



Random Number Generators, Special Functions LAG and RETAIN, FIRST./LAST.variable

FIRST.variable and LAST.variable

- The FIRST.variable and LAST.variable automatic variables are available when you are using a BY statement in a DATA step.
- These are particularly useful when dealing with sorted data or when you want to perform certain calculations or actions only on the first or last observation within a group.
- The FIRST.variable will have a value of 1 when SAS is processing an observation with the first occurrence of a new value for that variable and a value of 0 for the other observations.
- The LAST.variable will have a value of 1 for an observation with the last occurrence of a value for that variable and the value 0 for the other observations.



FIRST.variable and LAST.variable: Example

- SchoolData contains Subject ID, Test Dates, Score1, Score2, Score3 and Term variables.
- To extract last term records for each subject:

```
proc sort data=SchoolData;  
  by subjid term;  
run;  
data SchoolData2;  
  set SchoolData;  
  by subjid term;  
  if last.subjid;  
run;
```

Obs	SUBJID	test_date	score1	score2	score3	term
1	1	01OCT2020	73	55	68	1
2	1	09JAN2021	94	72	66	2
3	1	19APR2021	82	54	54	3
4	2	01OCT2020	135	57	54	1
5	2	09JAN2021	81	93	84	2
6	2	19APR2021	135	96	74	3
7	3	01OCT2020	115	84	94	1
8	3	09JAN2021	124	81	91	2
9	3	19APR2021	81	94	86	3
10	4	01OCT2020	127	55	55	1
11	4	09JAN2021	75	98	75	2
12	4	19APR2021	77	69	65	3
13	5	01OCT2020	96	59	92	1
14	5	09JAN2021	134	107	64	2
15	5	19APR2021	138	82	65	3



LAG Function

- In SAS, the LAG function returns the value of its argument from the last time the LAG function executed.
- Often, LAG is used to retrieve the value of a variable from the previous observation within a BY group.
- It is useful for calculating differences between consecutive values, identifying trends, or detecting changes in a variable over time etc.



LAG Function: Example

- Get the difference between DBP value at each visit between the previous visit.

```
proc sort data=VitalsData; by subjid visit; run;  
data VitalsData3;  
  set VitalsData;  
  by subjid;  
  difference = dbp - lag(dbp);  
  if not first.subjid;  
run;
```



RETAIN Statement

- When SAS reads the DATA statement at the beginning of each iteration of the DATA step, SAS places missing values in the program data vector for variables that were assigned by either an INPUT statement or an assignment statement within the DATA step.
- A RETAIN statement effectively overrides this default.
- That is, a RETAIN statement tells SAS not to set variables whose values are assigned by an INPUT or assignment statement to missing when going from the current iteration of the DATA step to the next. Instead, SAS retains the values.
- You can specify as few or as many variables as you want in a RETAIN statement.



Random Number Generators

- Random number generators (pseudorandom number generators) in SAS are functions or procedures used to generate pseudo-random numbers.
- These numbers are typically used in different statistical and simulation tasks, such as Monte Carlo simulations, random sampling, and random assignment.
- Random number generators require an initial number, called a **seed**, that is used to calculate the first random number.



Random Number Generators

- The seed must be a nonnegative number less than 2,147,483,647.
- A given seed always produces the same results. That is, using the same seed, a random number generator would select the same observations.
- If you choose 0 as the seed, then the computer clock time at execution is used. In this case, it is very unlikely that a random number generator would produce the same results.
- Note that it is common practice when conducting research to use a non-zero seed, so that the results could be reproduced if necessary.



Random Number Generators

- **RANUNI** function generates uniform random numbers in the range from 0 to 1.
 - Syntax: RANUNI(seed)
- **RANNOR** function generates a series of random numbers from a standard normal distribution. You can scale these values to any mean and standard deviation.
 - Syntax: RANNOR(seed)

Random Number Generators

- **RAND** function can be used to generate random numbers from various distributions.
 - Syntax: RAND(distribution, parameter1, .., parameterk)
- RAND function does not have a parameter where a seed can be specified.
- **CALL STREAMINIT** statement can be used to assign a seed/ initialize the random number generation.
- Recall, the seed value determines the starting point for the sequence of random numbers generated.



Random Number Generators: **RAND** Function

Distribution	Function Call	Parameter Values
Bernoulli	<code>RAND('BERNoulli', prob)</code>	where $0 \leq \text{prob} \leq 1$
Beta	<code>RAND('BETA', alpha, beta)</code>	where $0.01 \leq \alpha \leq 1.5e6$ and $0.20 \leq \beta \leq 1.5e6$
Binomial	<code>RAND('BINOmial', prob, trials)</code>	where $0 \leq \text{prob} \leq 1$ and $0 \leq \text{trials}$
Cauchy	<code>RAND('CAUChy')</code>	
Chi-Square(d)	<code>RAND('CHISquare', df)</code>	where $0.03 \leq \text{df} \leq 4e9$
Erlang	<code>RAND('ERLAng', alpha)</code>	where $1 \leq \alpha \leq 2e9$
Exponential	<code>RAND('EXPOnential', scale)</code>	where $0 \leq \text{scale} \leq 1e300$

[SAS documentation](#)

Random Sampling

- Randomly selecting records from a large data set may be helpful if your data set is so large as to prevent or slow processing, or if one is conducting a survey and needs to select a random sample from some master database.
- When you select records randomly from a larger data set (or some master database), you can achieve the sampling in a few different ways, including:
 - Sampling Without Replacement
 - Sampling With Replacement
 - Selecting a Stratified Sample



Random Sampling

- **Sampling Without Replacement**: a subset of the observations are selected randomly, and once an observation is selected it cannot be selected again.
- **Sampling With Replacement**: a subset of observations are selected randomly, and an observation may be selected more than once.
- **Selecting a Stratified Sample**: a subset of observations are selected randomly from each group of the observations defined by the value of a stratifying variable, and once an observation is selected it cannot be selected again.

Approximate-Sized vs. Exact-Sized Samples

- One can indicate a certain percentage of the population to be chosen randomly, eliminating the need for a predetermined sample size.
- In this scenario, the samples obtained are known as **approximate-sized** samples since their size is not predetermined but rather determined by the specified percentage of the population.
 - For example, select approximately 25% of a data set
- A specific sample size can be predetermined and chosen from a population. Such samples are referred to as **exact-sized** samples.
 - For example, select 15 observations from a data set



Random Number Generators: Proc Surveyselect

```
PROC SURVEYSELECT data = data5 out = sample1B  
  method = SRS  
  seed = 1234  
  samprate = 0.30;  
RUN;
```

- **DATA:** tells SAS the name of the input data set from which observations should be selected.
- **OUT:** tells SAS the name of the output data set (sample1B) in which the selected observations should be stored.
- **METHOD:** Tells SAS the sampling method that should be used. For example, SRS tells SAS to use the simple random sampling method to select observations, that is, with equal probability and **without** replacement.
- **SAMPRATE:** tells SAS what proportion (0.30) of the input data set should be sampled.

[SAS Documentation](#)



Fox School of Business
TEMPLE UNIVERSITY®

Select Stratified Random Sample using Proc Surveyselect

- **Selecting a Stratified Sample** : a subset of observations are selected randomly from each group of the observations defined by the value of a stratifying variable, and once an observation is selected it cannot be selected again.

```
PROC SURVEYSELECT data = data5
    out = sample6A
    method = SRS
    seed = 12345678
    sampsize = (5 5 5);
    strata city notsorted;
RUN;
```

- The STRATA statement tells SAS to partition the input data5 into non-overlapping groups defined by the variable *city*.
- The NOTSORTED option does not tell SAS that the data set is unsorted. Instead, the NOTSORTED option tells SAS that the observations in the data set are arranged in city groups, but the groups are not necessarily in alphabetical order.
- The SAMPSIZE statement tells SAS that we are interested in sampling five observations from each of the city groups.



DO Loops and An Introduction to Proc SQL

DO LOOPS

- Commonly used to execute a block of code repetitively.
- Some of the do loops are unconditional. That is, they can run a code repeatedly. These DO loops are called iterative DO loops.
- Whereas some DO loops are conditional in that SAS needs to be told to execute a specific task *until* a particular condition is met or to do something *while* a particular condition is met.

We call the former a DO UNTIL loop and the latter a DO WHILE loop



DO LOOPS: ITERATIVE DO LOOPS

- Syntax: DO *index_variable* = *start_value* TO *end_value* BY *increment*;

SAS statements

END;

- In an iterative DO loop, keywords DO, TO, END and values *index_variable*, *start_value*, *end_value* are required.
- Any valid SAS variable name can be used for *index_variable*. Single letters like i, j, k, etc. are commonly used.
- Iterative DO loop starts at *start_value* and stops at *end_value*.
- Default increment, which is by how much SAS changes the *index_variable* after each iteration, is 1.
- If any increment other than 1 is desired, it needs to be specified in a BY clause.

DO LOOPS: DO UNTIL

- SAS executes the DO loop **UNTIL** the expression that is specified is true (instead of a fixed number of iterations like in the case of iterative DO loops).
- Syntax: DO UNTIL (*expression*);
 SAS statements
 END;
- SAS expression/condition specified can be any valid SAS expression enclosed in parentheses.
- This expression is not evaluated until the bottom of the loop. Therefore, a DO UNTIL loop always executes at least once. As soon as the expression is determined to be true, the DO loop does not execute again.



DO UNTIL Example:

- Calculate how many years it would take to accumulate \$100,000 if you deposit \$3000 each year into a savings account with 5.4% interest rate.

```
data investmentdata;  
  do until(value >= 100000);  
    value + 3000;  
    value + value * 0.054;  
    year + 1;  
    output;  
  end;  
run;
```



DO LOOPS: DO WHILE

- SAS executes the DO loop **WHILE** the expression that is specified is true (instead of a fixed number of iterations like in the case of iterative DO loops).
- Syntax: DO UNTIL (*expression*);
 SAS statements
 END;
- SAS expression/condition specified can be any valid SAS expression enclosed in parentheses.
- An important difference between the DO UNTIL and DO WHILE statements is that the DO WHILE expression is evaluated at the top of the DO loop. If the expression is false the first time it is evaluated, then the DO loop doesn't even execute once.



DO WHILE Example:

- Calculate how many years it would take to accumulate \$100,000 if you deposit \$3000 each year into a savings account with 5.4% interest rate.

```
data investmentdata2;  
value = 0;  
do while(value <= 100000);  
    value + 3000;  
    value + value * 0.054;  
    year + 1;  
    output;  
end;  
run;
```



An Introduction to PROC SQL

- PROC SQL is a powerful Base SAS Procedure that combines the functionality of DATA and PROC steps into a single step.
- PROC SQL can sort, summarize, subset, join (merge), and concatenate datasets, create new variables, and print the results or create a new table or view all in one step!
- PROC SQL can not only retrieve information without having to learn SAS syntax, but it can often do this with fewer and shorter statements than traditional SAS code.
- Additionally, SQL often uses fewer resources than conventional DATA and PROC steps. Further, the knowledge learned is transferable to other SQL packages.



An Introduction to PROC SQL

- In general, the Structured Query Language (SQL) is a standardized language used to retrieve and update data stored in relational tables (or databases).
- When coding in SQL, the user is not required to know the physical attributes of the table such as data location and type. SQL is non-procedural. The purpose is to allow the programmer to focus on what data should be selected and not how to select the data. The method of retrieval is determined by the SQL optimizer, not by the user.
- A table is a two-dimensional representation of data consisting of columns and rows. A table in SQL is simply another term for a SAS data set.
- Tables are logically related by values such as a key column.



An Introduction to PROC SQL

- Basic Proc SQL Syntax:

```
PROC SQL;  
  SELECT column, column ...  
  FROM tablename|viewname. ...
```

- In Proc SQL,
 - Statements (clauses) in the SQL procedure are not separated by semicolons, the entire query is terminated with a semicolon.
 - Items in an SQL statement are separated by a comma.
 - There is a required order of statements in a query.
 - One SQL procedure can contain many queries and a query can reference the results from previous queries.
 - The SQL procedure can be terminated with a QUIT statement, RUN statements have no effect.



An Introduction to PROC SQL

- Select Statement:
 - To retrieve and display data a SELECT statement is used.
 - The data will be displayed in the order you list the columns in the SELECT statement.
 - A column can be a variable, calculated value, assigned value or formatted value.
 - An asterisk (*) can be used to select all columns.
- SELECT Syntax:

```
PROC SQL options;  
  SELECT column(s)  
    FROM table-name | view-name  
   WHERE expression  
  GROUP BY column(s)  
  HAVING expression  
 ORDER BY column(s);
```



An Introduction to PROC SQL

- Simple Queries: A query is a request for information from a table or tables. The query result is typically a report but can also be another table.
- For instance: I would like to select last name, department, and salary from the employee table where the employee's salary is greater than 35,000.

How would this query (request) look in SQL?

```
SELECT LASTNAME, DEPARTMENT, SALARY  
FROM CLASS.EMPLOY  
WHERE SALARY GT 35000
```



An Introduction to PROC SQL

- *From* (Part of Select Statement): Specifies the input table(s).
- For example, to select social security number, salary and bonus (columns) for all employees (rows) from the employeedata table (data set):

```
PROC SQL;
```

```
    SELECT SSN, SALARY, BONUS
```

```
    FROM CLASS.EMPLOYEEDATA;
```

```
QUIT;
```



PROC SQL Basics

- An asterisk can be used on the SELECT statement to select all columns.
- To specify certain variables, variables can be listed on the SELECT statement.
- The AS keyword can be used to rename variables when selecting or creating new variables in the output.
- To refer to the results of calculations on columns within a select statement, the keyword CALCULATED must be used.



PROC SQL Basics

- The CASE statement is used to perform conditional logic within a SELECT statement.
- CASE statement allows you to evaluate conditions and return different values based on those conditions. It is useful for creating derived variables or transforming existing variables based on specific criteria.
- END is required when using the CASE.
- Coding the WHEN in descending order of probability will improve efficiency because SAS will stop checking the CASE conditions as soon as it finds the first true value.



PROC SQL Basics

- Summary functions such as AVG/MEAN, COUNT/FREQ, MAX, MIN, NMISS, STD, SUM, VAR can be used on the SELECT Statement.
- Unless GROUP BY is present when a summary function is used, the summary statistics would be remerged to the dataset.
- GROUP BY clause is used to group rows that have the same values into summary rows, often to perform aggregate summaries on the grouped data.
- It also sorts the data by the grouping variables.



PROC SQL Basics

- The ORDER BY clause will return the data in sorted order. In the absence of summary functions, ORDER BY can be used to sort data.
- In order to subset data, WHERE can be used similar to a DATA step.
- As part of WHERE, a calculated variable cannot be directly referred. Either the calculation can be repeated in WHERE clause, or the keyword CALCULATED can be used along with the derived column name.
- When grouping is in affect (GROUP BY), WHERE cannot be used for subsetting. Instead, HAVING can be used.

PROC SQL Basics

- To create a table, “CREATE TABLE *tablename* AS” must be used right before the SELECT statement.
- To concatenate two datasets, UNION operator can be used. The UNION operator keeps only unique observations. To keep all observations, the UNION ALL operator can be used.
- To merge two or more datasets, INNER/LEFT/RIGHT JOIN can be used.

Selected Statistical Methods in SAS and Output Delivery System (ODS)

Correlation and Regression

- Correlation measures the association between two quantitative variables.
- Simple linear regression creates an equation to predict the value of the dependent variable using a single independent variable.
- Multiple linear regression creates an equation to predict the value of the dependent variable using multiple independent variables.



Correlation

- The correlation coefficient is a measure of the linear relationship between two quantitative variables measured on the same subject (or entity).
- The sample correlation r is a unitless quantity (i.e. it does not depend on the units of measurement) that ranges from -1 to $+1$.
- The correlation coefficient ρ is typically estimated from data using the Pearson correlation coefficient and designated as r .
- In practice it is often of interest to test the hypotheses

$H_0 : \rho = 0$ (there is no linear relationship between the two variables)

$H_0 : \rho \neq 0$ (there is a linear relationship between the two variables)



Correlation

- In practice it is often of interest to test the hypotheses

$H_0 : \rho = 0$ (there is no linear relationship between the two variables)

$H_0 : \rho \neq 0$ (there is a linear relationship between the two variables)

- PROC CORR in SAS provides a test of the above hypotheses designed to determine whether the estimated correlation coefficient, r , is significantly different from zero.
- This test assumes that the data represent a random sample from some bivariate normal population. If normality is not a good assumption, nonparametric correlation estimates are available, the most popular of which is Spearman's rho, which PROC CORR also provides.
- To examine the nature of the relationship between two variables is always good practice to look at scatterplots of the variables.



Correlation

Syntax

PROC CORR <options>; <statements>;

- Common Options:
 - **DATA=datasetname**; Specifies dataset.
 - **SPEARMAN** - Requests Spearman rank correlations.
 - **NOSIMPLE** - Suppresses display of descriptive
 - **NOPROB** - Suppresses display of p-values
 - **PLOTS** – Produces plots

Common Statements

- **VAR variable(s);** Pair wise correlations are calculated for the variables listed.
- **BY variable(s);** Produces separate set of pairwise correlations for variables in the VAR list for each level of the categorical variable in the BY list. (Data must be sorted prior to using the BY statement in PROC CORR.)
- **WITH variable(s);** Correlations are obtained between the variables in the VAR list with variables in the WITH list



Simple Linear Regression

- The regression line that SAS calculates from the data is an estimate of a theoretical line describing the relationship between the independent variable (X) and the dependent variable (Y).
- The theoretical line is $Y = \alpha + \beta x + \varepsilon$

where α is the y-intercept, β is the slope, and ε is an error term that is normally distributed with zero mean and constant variance.

- It should be noted that $\beta = 0$ indicates that there is no linear relationship between X and Y.
- A simple linear regression analysis is used to develop an equation (a linear regression line) for predicting the dependent variable given a value (x) of the independent variable.



Simple Linear Regression

- The regression line calculated by SAS is given by $\hat{Y} = a + bx$
where a and b are the least-squares estimates of α and β .
- The null hypothesis that there is no predictive linear relationship between the two variables is that the slope of the regression equation is zero.
- Specifically, the hypotheses are:
 $H_0: \beta = 0$
 $H_a: \beta \neq 0$
- A low p-value for this test (say, less than 0.05) indicates significant evidence to conclude that the slope of the line is not 0; that is, that knowledge of X would be useful in predicting Y .



Simple Linear Regression: PROC REG

Syntax

- The general syntax for PROC REG is

PROC REG <options>; <statements>;

- Common OPTIONS are
 - **DATA=datasetname** - Specifies dataset.
 - **SIMPLE** - Displays descriptive statistics.

Common Statements

- **MODEL** dependentvar = independentvar</options>;
- **BY groupvariable;** Produces separate regression analyses for each value of the BY variable
- Note: Several MODEL options are available, but we will defer their discussion to the section on using SAS for multiple linear regression.



Simple Linear Regression: PROC REG

- **Example:** A random sample of fourteen elementary school students is selected from a school, and each student is measured on a creativity score (X) using a new testing instrument and on a task score (Y) using a standard instrument. The task score is the mean time taken to perform several hand–eye coordination tasks. Because administering the creativity test is much cheaper, the researcher wants to know if the CREATE score is a good substitute for the more expensive TASK score.



Simple Linear Regression: PROC REG

- Example (Continues):

SAS Results

Root MSE	1.60348	R-Square	0.3075
Dependent Mean	5.05000	Adj R-Sq	0.2498
Coeff Var	31.75213		

- R-Square: This is a measure of the strength of the association between the dependent and independent variables. (It is the square of the Pearson correlation coefficient.) The closer this value is to 1, the stronger the association. In this case $R^2 = 0.31$ indicates that 31% of the variability in TASK is explained by the regression with CREATE.

Simple Linear Regression: PROC REG

- Example (Continues): SAS Results

Parameter Estimates					
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > t
Intercept	1	2.16452	1.32141	1.64	0.1273
CREATE	1	0.06253	0.02709	2.31	0.0396

- Slope: The statistical test on the “CREATE” row is for a test of $H_0: \beta = 0$, and $p = 0.04$ provides evidence to reject the null hypothesis and conclude that the slope is not zero. (The statistical test on the Intercept is generally of little importance.)
- Estimates: The column labeled “Parameter Estimate” gives the least squares estimates a and b of the regression equation. In this case, the equation is: $TASK = 2.16452 + 0.06235 * CREATE$;
- Thus, from this equation you can predict a value of the TASK score from the CREATE score. However, before making any predictions using this equation, you should analyze the relationship further.



EXAMPLE

(data entered)

PROC REG;

MODEL TASK=CREATE;

**TITLE 'Example simple linear
regression using PROC REG';**

RUN;

QUIT;

The MODEL statement defines the linear regression equation you are calculating.

A QUIT statement is recommended for PROC REG to end the analysis.



Selected Output from PROC REG

R-Squared is a measure of the strength of the association.

Root MSE	1.60348	R-Square	0.3075
Dependent Mean	5.05000	Adj R-Sq	0.2498
Coeff Var	31.75213		

The regression equation from this analysis is

$$\text{TASK} = 2.16 + 0.0625 * \text{CREATE}$$

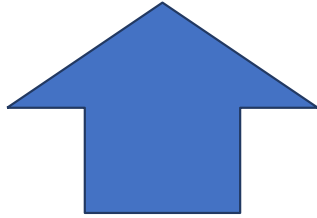
Parameter Estimates					
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > t
Intercept	1	2.16452	1.32144	1.637	0.1141
CREATE	1	0.06253	0.02709	2.31	0.0396

The parameter estimates are the estimates of alpha (Intercept) and beta (slope/CREATE).

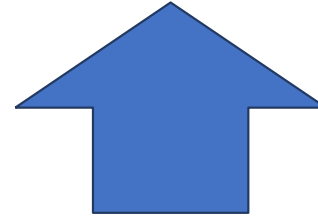


The SAS MODEL Statement

MODEL DEPENDENT VARIABLE(s) = INDEPENDENT VARIABLE(s)



LEFT side specifies
variable(s) to be
predicted.



RIGHT side specifies
predictor variable(s).



Predicting a New Value

- For this model, you might conclude that there is a moderate linear fit between CREATE and TASK, but it is not impressive ($R^2 = 0.3075$) or about 31% of the variation is accounted for by the regression using CREATE.
- Using the information in the regression equation, you could predict a value of TASK from CREATE=40.

$$4.67 = 2.16452 + 0.06235 * 40;$$

The value of TASK
predicted.

The value of
CREATE used for
prediction



MULTIPLE LINEAR REGRESSION USING PROC REG

- Multiple Linear Regression (MLR) is an extension of simple linear regression. In MLR, there is a single dependent variable (Y) and more than one independent (X_i) variable.
- As with simple linear regression, the multiple regression equation calculated by SAS is a sample-based version of a theoretical equation describing the relationship between the k independent variables and the dependent variable Y .

$$Y = a + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_k X_k + \epsilon$$



Hypotheses Tested

- As part of the analysis, the statistical significance of each of the coefficients is tested using a Student's t -test to determine if it contributes significant information to the predictor.
- These are tests of the hypotheses:
$$H_0 : \beta_i = 0$$
$$H_a : \beta_i \neq 0$$
- For these tests, if the p-value is low (say, <0.05), the conclusion is that the i th independent variable contributes significant information to the equation.



Using SAS PROC REG for Multiple Linear Regression

- As mentioned previously, the general syntax for PROC REG is

PROC REG <Options>; <Statements>;



Table. Additional Statement Options for the PROC REG MODEL statement
(Options follow /) (Relevant to Multiple Linear Regression)

Option	Explanation
P	Requests a table containing predicted values from the model.
R	Requests that the residuals be analyzed.
CLM	Prints the 95 percent upper and lower confidence limits.
CLI	Requests the 95 percent upper and lower confidence limits for an individual value.
INCLUDE=k	Include the first k variables in the variable list in the model (for automated selection procedures).
SELECTION=option	Specifies automated variable selection procedure: BACKWARD, FORWARD, and STEPWISE, etc.
SLSTAY=p	Specifies the maximum p-value for a variable to stay in a model during automated model selection.
SLENTY=p	Minimum p-value for a variable to enter a model for forward or stepwise selection.



EXAMPLE

(enter data)

PROC REG;

MODEL JOBScore=**TEST1 TEST2 TEST3 TEST4;**

TITLE 'Job Score Analysis using PROC REG';

RUN;

QUIT;

In this model all of the predictors (independent variables) are specified



Results

Root MSE	3.54777	R-Square	0.9754
Dependent Mean	75.10000	Adj R-Sq	0.9557
Coeff Var	4.72407		

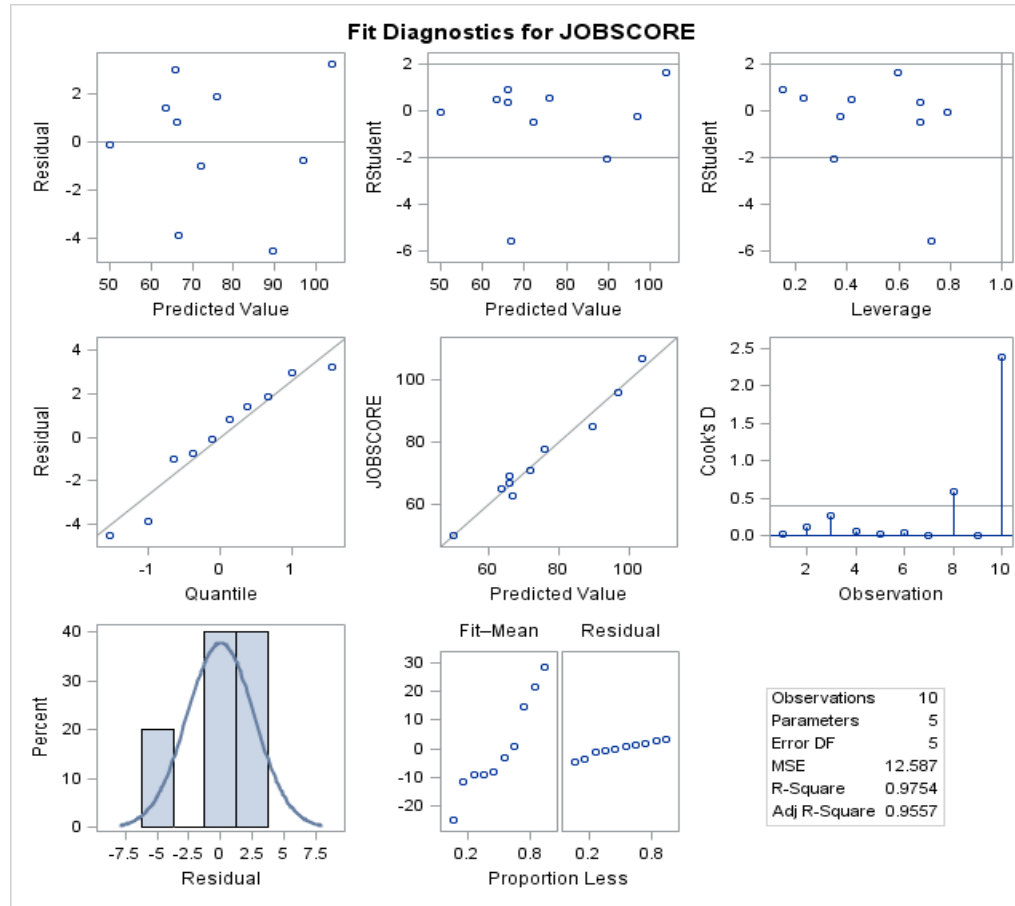
R-Square provides a measure of the strength of the prediction equation.

Parameter Estimates					
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > t
Intercept	1	-95.55939	12.82483	-7.45	0.0007
TEST1	1	0.17631	0.06616	2.66	0.0446
TEST2	1	-0.22344	0.14354	-1.56	0.1803
TEST3	1	1.74602	0.27770	6.29	0.0015
TEST4	1	0.26865	0.18424	1.46	0.2046

The Parameter Estimates are the estimates of the coefficients in the prediction equation.



Diagnostics for MLR Same as for SLR



PERFORMING A ONE-SAMPLE T-TEST

- A one-sample t-test is often used to compare an observed mean with a known or "gold standard" value.
- In general, for a one-sample t-test you obtain a random sample from some population and then compare the observed sample mean to some fixed value. The typical hypotheses for a one-sample t-test are as follows:

$H_0 : m = m_0$: The population mean is equal to a hypothesized value, m_0

$H_a : m \neq m_0$: The population mean is not equal to m_0



t -test Assumptions

- The key assumption underlying the one-sample t -test is that the population from which the random sample is selected is normal.
- If the data are non-normal, then nonparametric tests such as the sign test and the signed rank test are available.
- However, because of the central limit theorem, whenever the sample size is sufficiently large, the distribution of the sample mean is often approximately normal even when the population is non-normal. (See next slide)



Sample Size Recommendations

- The following are general guidelines:
- Small sample size ($N < 15$): You should not use the one-sample t-test if the data are clearly skewed or if outliers are present
- Moderate sample size ($N > 15$): The one-sample t-test can be safely used except when there are severe outliers.
- Large sample size ($N > 40$): The one-sample t-test can be safely used without regard to skewness or outliers.



Running the One-Sample t-Test in SAS

- Here are two ways to perform a one-sample t-test: First, using PROC UNIVARIATE, specify the value of m_0 for the test reported in the "Tests for Location." For example,

```
PROC UNIVARIATE MU0=4 ;VAR LENGTH ;RUN;
```

- does a *t-test* of the null hypothesis that $m_0 = 4$.
- A second method in SAS for performing this one-sample *t-test* is to use the PROC TTEST procedure. For this procedure, the corresponding SAS code is

```
PROC TTEST H0=4;VAR LENGTH; RUN;
```



EXAMPLE

```
DATA ONESAMPLE;  
INPUT LENGTH @@;  
DATALINES;  
4 3.95 4.01 3.95 4.00  
3.98 3.97 3.97 4.01 3.98  
3.99 4.01 4.02 4.02 3.98  
4.01 3.99 4.03 4.00 3.99  
;  
Title 'Single sample t-test, using PROC UNIVARIATE';  
PROC UNIVARIATE DATA=ONESAMPLE MU0=4;VAR LENGTH; RUN;  
Title 'Single sample t-test using PROC TTEST';  
PROC TTEST DATA=ONESAMPLE H0=4;var LENGTH;  
RUN;
```

Two ways to perform
the t-test – PROC
UNIVARIATE and PROC
TTEST.



Two Methods for a One Sample t -test

Title 'Using PROC UNIVARIATE';

```
PROC UNIVARIATE DATA=ONESAMPLE MU0=4;  
    VAR LENGTH;  
RUN;
```

Note indication of
hypothesized value

Title 'Using PROC TTEST';

```
PROC TTEST DATA=ONESAMPLE H0=4 ;  
    VAR LENGTH;  
RUN;
```

Note indication of
hypothesized value



Results for a One-Sample *t*-test – Two Ways

Tests for Location: Mu0=4				
Test	Statistic		p Value	
Student's t	t	-1.40593	Pr > t	0.1759
Sign	M	-1.5	Pr >= M	0.6291
Signed Rank	S	-26.5	Pr >= S	0.2240

PROC UNIVARIATE
RESULTS – See the
line for Student's t
where p=0.1759

N	Mean	Std Dev	Std Err	Minimum	Maximum
20	3.9930	0.0223	0.00498	3.9500	4.0300

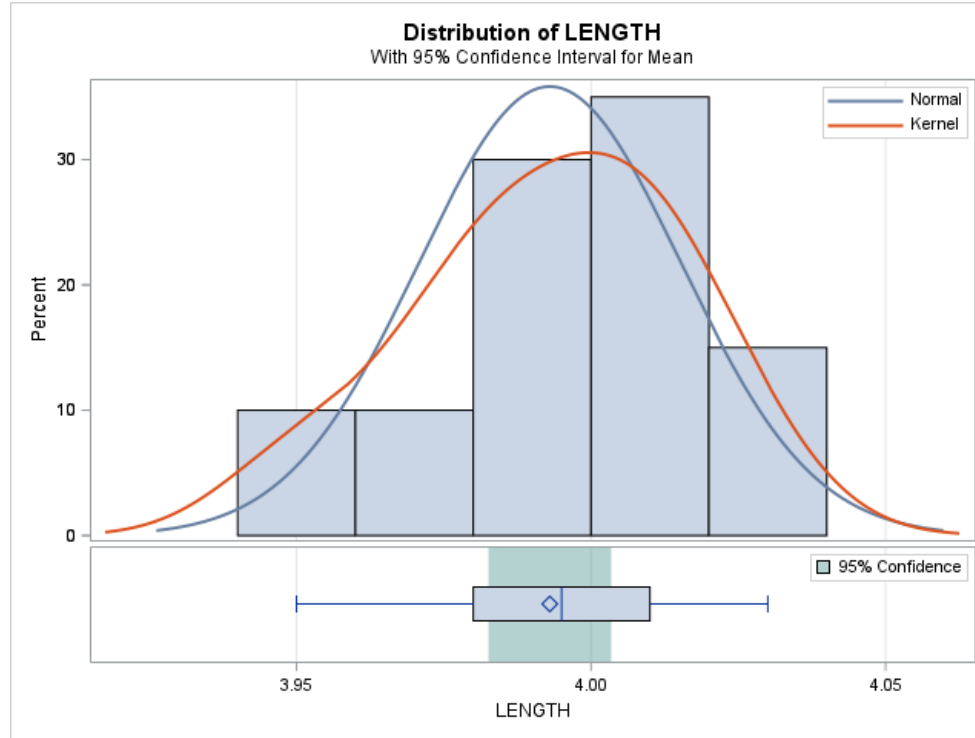
Mean	95% CL Mean	Std Dev	95% CL Std Dev
3.9930	3.9826 4.0034	0.0223	0.0169 0.0325

DF	t Value	Pr > t
19	-1.41	0.1759

PROC TTEST RESULTS – See
where p=0.1759



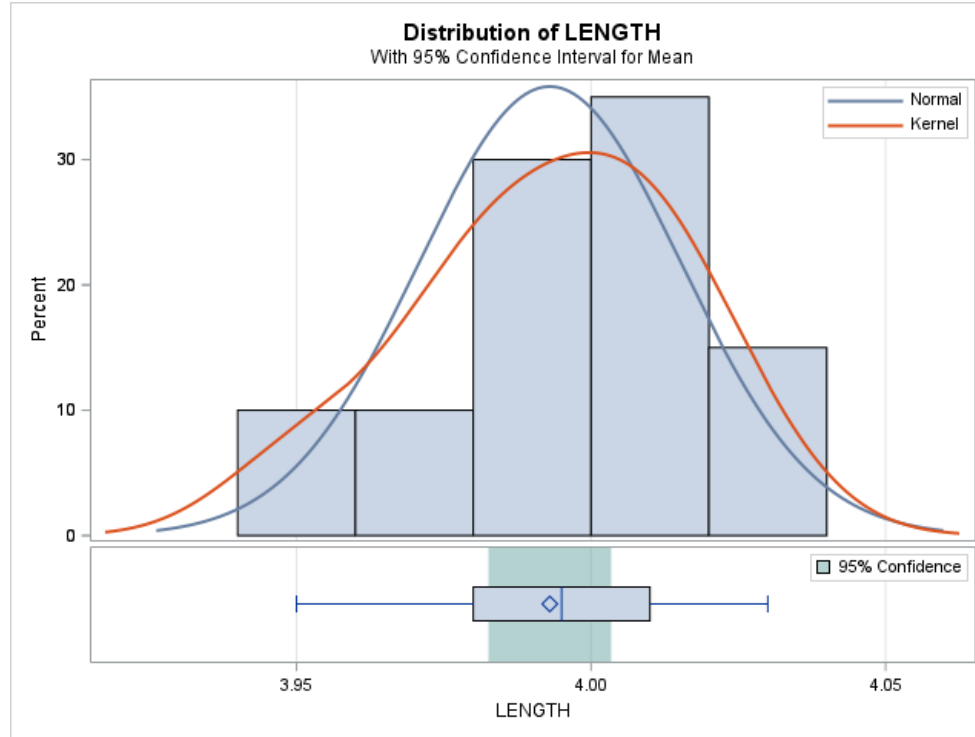
Plot Created by PROC TTEST



- The blue curve is a normal curve based on the mean and sd estimated from the data.
- The red curve is a kernel density estimator, which is a smoothed version of the histogram.
- If dramatic skewness were evident in the data, then the skewness would also be displayed in the kernel density estimator.



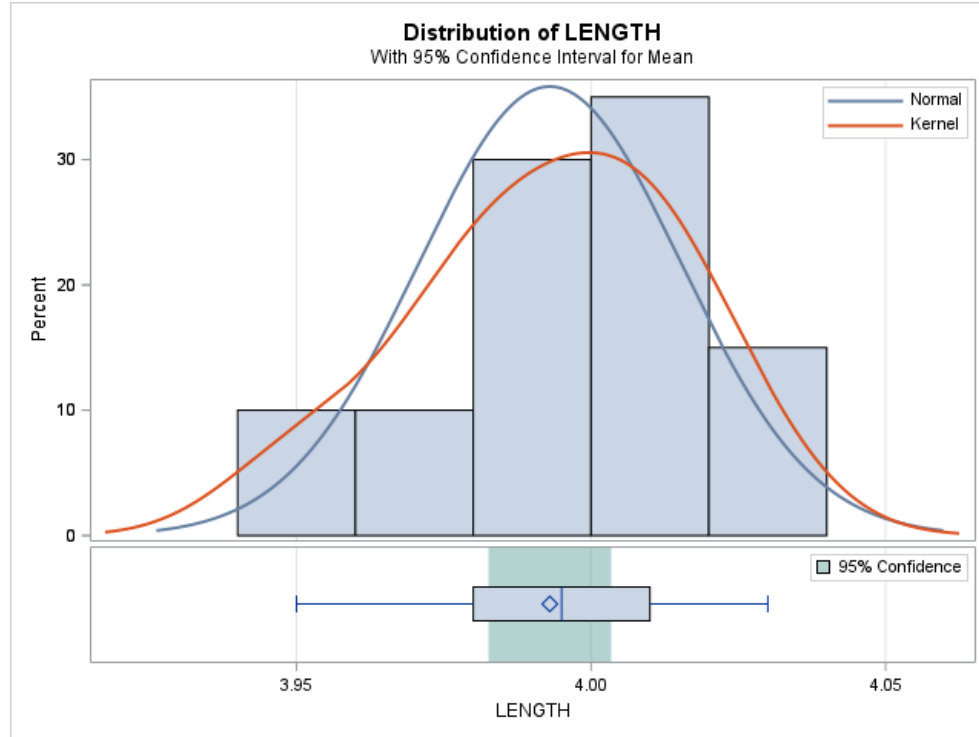
Plot Created by PROC TTEST



- These plots provide information about the normality assumption.
- There does not appear to be a dramatic departure from normality because the kernel density estimator is fairly bell-shaped.
- At the bottom of the plot is a boxplot.
- The boxplot is fairly symmetrical in shape with some tendency for the left whisker to be longer than the right one.



Plot Created by PROC TTEST



- Special graphics output such as the plots that follow are sometimes produced by SAS procedures
- Whenever you are in doubt about what type of graphics are available or what graphics may have been implemented in a new version of SAS, it is a good idea to run your code with ODS GRAPHICS ON/OFF.
- ```
ODS GRAPHICS ON;
```

```
before the procedure(s) and
```

```
ODS GRAPHICS OFF;
```





# One-tailed tests

- If you are in rejecting the null hypothesis if the population mean differs from the hypothesized value in a particular direction of interest, you may want to use a one-tailed test. For example, the hypotheses if there is sufficient evidence that the mean is smaller than the hypothesized value, the hypotheses become as follows:

**$H_0: \mu = \mu_0$** : The population mean is equal to a hypothesized value,  $\mu_0$ .

**$H_a: \mu < \mu_0$** : The population mean is less than  $\mu_0$ .

- In order to report the results of a one-tailed test you need to modify the reported p-value to fit a one-tailed test by dividing it by 2.



# PERFORMING A TWO-SAMPLE *T*-TEST

- The SAS PROC TTEST procedure is used to test for the equality of means for a two-sample (independent group) t-test.
- The purpose of the two-sample t-test is to determine whether your data provide you with enough evidence to conclude that there is a difference in means.
- For a two-sample t-test you obtain independent random samples of size  $N1$  and  $N2$  from the two populations and compare the observed sample means.
- The typical hypotheses for a two-sample t-test are as follows:

$H_0: \mu_1 = \mu_2$ : The population means of the two groups are equal.

$H_a: \mu_1 \neq \mu_2$ : The population means are not equal.



# Key Assumptions for a Two-Sample t-test

- Key assumptions underlying the two-sample t-test are

1. random samples are independent
2. populations are normally distributed with equal variances.

\*\*\*If the data are non-normal, then nonparametric tests such as the Mann-Whitney U are available.



# Guidelines regarding normality and equal variance assumptions

- **Normality:** As in the one-sample case, rules of thumb are available to help you determine whether to go ahead and trust the results of a two-sample t-test even when your data are non-normal. The sample size guidelines given earlier in this chapter for the one-sample test can be used in the two-sample case by replacing  $N$  with  $N1 + N2$ .
- **Equal variances:** There are two t-tests reported by SAS in this setting: one based on the assumption that the variances of the two groups are equal (and thus using a pooled estimate of the common variance) and one (Satterthwaite) not making that assumption.



# Running the Two-Sample $t$ -Test in SAS

- Two sample  $t$ -tests can be obtained using the PROC TTEST procedure which was previously introduced in the context of a one-sample test.
- The syntax for the TTEST procedure is as follows:

```
PROC TTEST <options>;
 CLASS variable; <Statements>;
RUN;
```



# Common Options for PROC TTEST

| Common Options for PROC TTEST |                                                                      |
|-------------------------------|----------------------------------------------------------------------|
| Option                        | Explanation                                                          |
| DATA =<br>datasetname         | Specifies which dataset to use.                                      |
| COCHRAN                       | Use Cochran and Cox probability approximation for unequal variances. |
| H0=n                          | Specifies the hypothesized value under $H_0$ .                       |



# Common statements for PROC TTEST

| Common statements for PROC TTEST |                                                                                                                          |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| CLASS variables;                 | For a two-sample t-test, specify the grouping variable for the analysis.<br><br>PROC TTEST;<br>CLASS GROUP; VAR SCORE;   |
| VAR variables;                   | Specify observed variables for test:<br>PROC TTEST;<br>CLASS GROUP; VAR SCORE WEIGHT HEIGHT;                             |
| PAIRED x*y;                      | Specifies that a paired t-test is to be performed and which variables to use.<br><br>PROC TTEST;<br>PAIRED BEFORE*AFTER; |
| BY, FORMAT, LABEL,<br>WHERE      | These statements are common to most procedures, and may be used here.                                                    |



# EXAMPLE

```
DATA TTEST;
INPUT BRAND $ HEIGHT;
DATALINES;
 A 20.00
 A 23.00
 A 32.00
etc
;
PROC TTEST;
 CLASS BRAND;
 VAR HEIGHT;
 Title 'Independent Group t-Test Example';
RUN;
QUIT;
```

Note that the CLASS statement identifies the “grouping” variable that specifies which groups are to be compared. In this case BRAND is A or B, the two groups to be compared.

The VAR statement identifies the observed variable.





# Understanding the output

THIRD— using either t-test, the p-value is small, so reject the null hypothesis and conclude that there is a difference in means.

| Method        | Variances | DF     | t Value | Pr >  t |
|---------------|-----------|--------|---------|---------|
| Pooled        | Equal     | 13     | -4.57   | 0.0005  |
| Satterthwaite | Unequal   | 9.3974 | -4.82   | 0.0008  |

| Equality of Variances |        |        |         |        |
|-----------------------|--------|--------|---------|--------|
| Method                | Num DF | Den DF | F Value | Pr > F |
| Folded F              | 7      | 6      | 6.33    | 0.0388 |

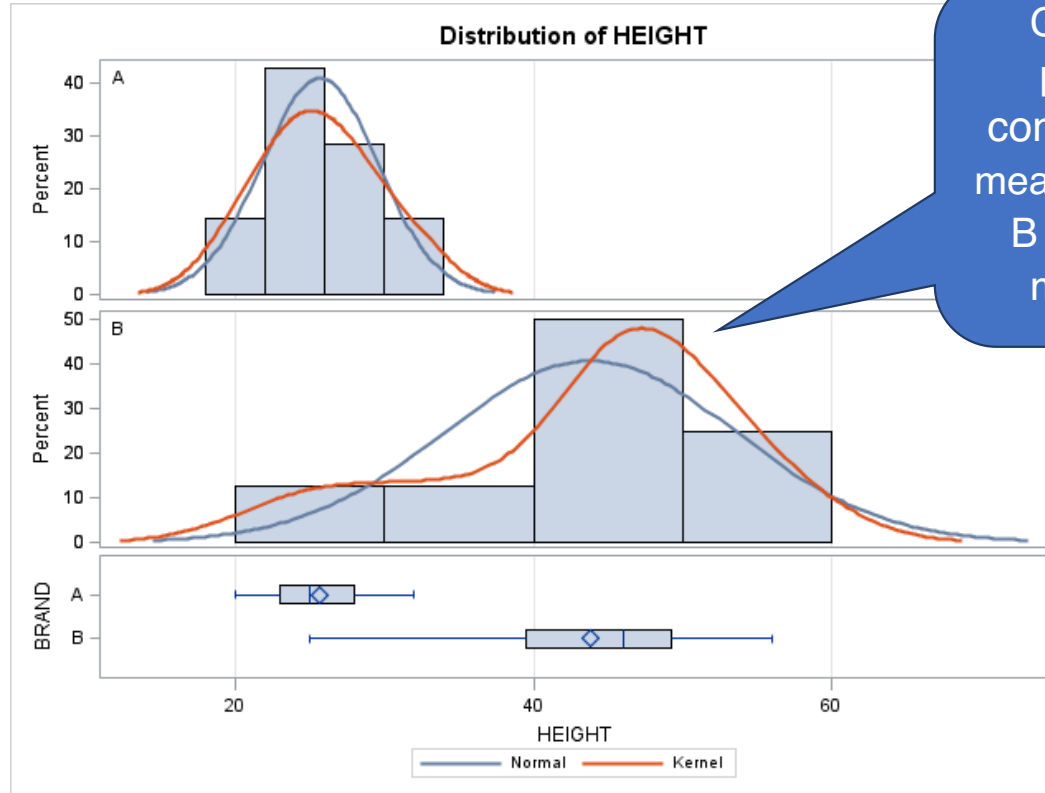
SECOND - If variances are equal, do the standard (Pooled) t-test. If variances unequal, do the unequal (Satterthwaite) version of the t-test

FIRST... use this test to determine if variances are "equal." A low p value  $p < .05$  indicates variances are unequal.

START HERE



# Graphical Results from Two Sample $t$ -test



Graphical results provide a visual confirmation that the mean HEIGHT of group B is larger than the mean of group A



# PERFORMING A PAIRED T-TEST

- To perform a paired t-test to compare two repeated measures (such as in a before–after situation) where both observations are taken from the same or matched subjects, use PROC TTEST with the PAIRED statement.
- Suppose your data contain the variables WBEFORE and WAFTER (before and after weight on a diet) for eight subjects. The hypotheses for this test are:

**$H_0: \mu_{\text{Loss}} = 0$** : The population average weight loss is zero.

**$H_a: \mu_{\text{Loss}} \neq 0$** : The population average weight loss is not zero.



# EXAMPLE

```
PROC TTEST;
PAIRED WBEFORE*WAFTER;
TITLE 'Paired t-test Example';
RUN;
```

Note the PAIRED statement – which indicates the two paired variables WBEFORE and WAFTER.



# Understanding the Paired $t$ -test output

**Difference: WBEFORE - WAFTER**

| N | Mean    | Std Dev | Std Err | Minimum | Maximum |
|---|---------|---------|---------|---------|---------|
| 8 | 15.2500 | 10.9381 | 3.8672  | -1.0000 | 35.0000 |

| Mean    | 95% CL Mean |         | Std Dev | 95% CL Std Dev |         |
|---------|-------------|---------|---------|----------------|---------|
| 15.2500 | 6.1055      | 24.3945 | 10.9381 | 7.2320         | 22.2621 |

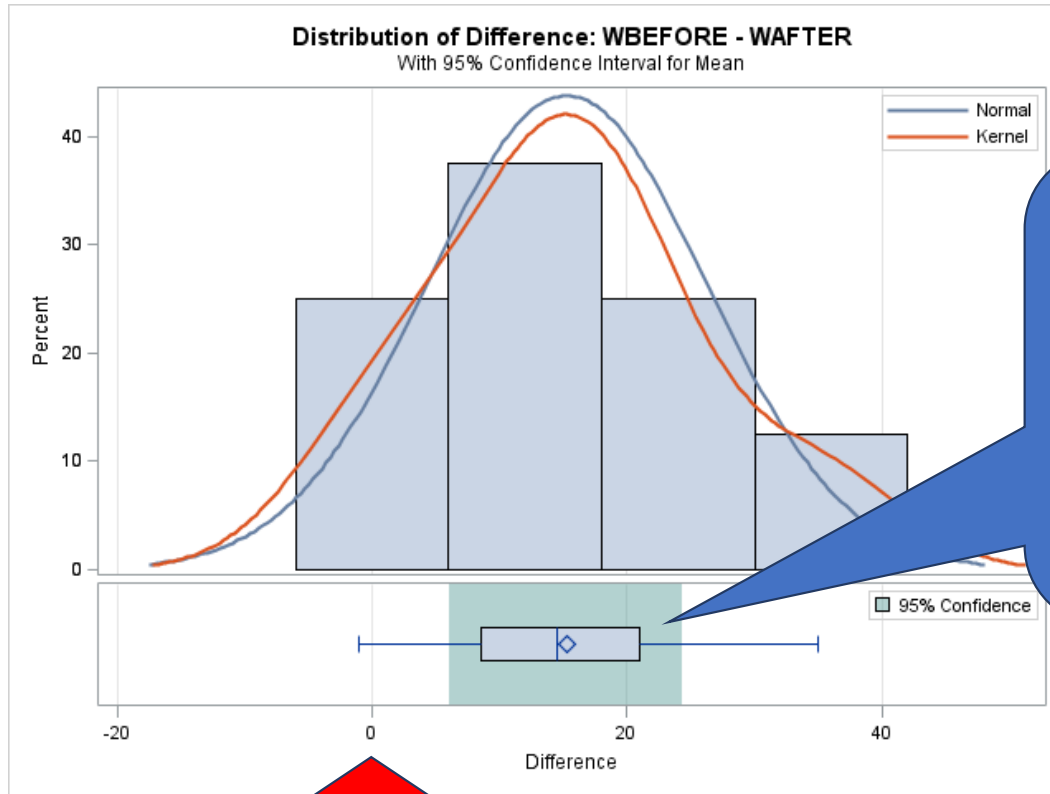
| DF | t Value | Pr >  t |
|----|---------|---------|
| 7  | 3.94    | 0.0056  |

Note the test is performed on the difference of the paired variables.

The results of the  $t$ -test reports  $p=0.0056$  which indicates to reject the null hypothesis of no difference and conclude that there was statistically significant weight loss.



# Graphical Results for Paired $t$ -test



Notice that the 95% confidence interval (indicated by the green highlight) does not include 0, which is another way to conclude that the mean difference is greater than 0.



# ANOVA

- To be able to compare three or more means , a one-way ANOVA with multiple comparisons can be used.
- **PROC ANOVA:** a basic procedure useful for one-way ANOVA or for multiway factorial designs with fixed factors and an equal number of observations per cell.
- **PROC GLM:** for one-way repeated measures analysis, and techniques not supported by PROC ANOVA.



# COMPARING THREE OR MORE MEANS USING ONE-WAY ANALYSIS OF VARIANCE

- A one-way ANOVA is an extension of the independent group t-test where there are more than two groups.
- Assumptions for this test are similar to those for the t-test:
  - Data within groups are normally distributed with equal variances across groups.
  - Groups are from independent samples.
- The hypotheses for the comparison of independent groups are as follows (k is the number of groups):

**$H_0: \mu_1 = \mu_2 = \dots = \mu_k$** : Means of all the groups are equal.

**$H_a: \mu_i \neq \mu_j$  for some  $i \neq j$** : At least two means are not equal.



# Simplified Syntax for PROC ANOVA

- The syntax for the statement is as follows:

PROC ANOVA *<Options>*,

**CLASS** variable;

**MODEL** dependentvar = independentvars;

**MEANS** independentvars / typecomparison  
*<meansoptions>;*

CLASS defines grouping variable.

The MODEL statement  
defines the model tested.

The MEANS statement defines  
post hoc multiple comparisons.



Table. Common Options for PROC ANOVA and PROC GLM for performing a One-Way ANOVA or simple Repeated Measures

| Option           | Explanation                                                                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DATA = dataname  | Specifies which data set to use.                                                                                                                                                |
| NOPRINT          | Suppresses output. This is used when you want to extract information from ANOVA results but don't want SAS to produce output in the Results Viewer.                             |
| OUTSTAT=dataname | Names an output data set that saves a number of the results from the ANOVA calculation.                                                                                         |
| PLOTS=options    | Specify PLOTS=NONE to suppress plots that are generated by default.                                                                                                             |
| ORDER=option     | Specifies order in which to display the CLASS variable (similar to what was covered in Chapter 10:Analyzing Counts and Tables.) Options are DATA, FORMATTED, FREQ, or INTERNAL. |
| ALPHA=p          | Specifies alpha level for a Confidence Interval (GLM only)                                                                                                                      |



|                          |                                                                                                                                                                 |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (Table Continued)        |                                                                                                                                                                 |
| CLASS variable list;     | This statement is required and specifies the grouping variable(s) for the analysis.                                                                             |
| MODEL specification      | Specifies the dependent and independent variables for the analysis. More specifically, it takes the form<br><br>MODEL dependentvariable=independentvariable(s); |
| FREQ var                 | Specifies that a variable represents the count of values for an observation. Similar to the WEIGHT statement for PROC FREQ.                                     |
| MEANS vars               | Calculates means for dependent variables and may include comparisons.                                                                                           |
| LSMEANS vars             | Calculates least square means for a dependent variable & to request comparisons. (GLM Only)                                                                     |
| REPEATED vars            | Used to specify repeated measure variables.                                                                                                                     |
| TEST specification       | Used to specify a hypothesis test value.                                                                                                                        |
| CONTRAST specification   | Allows you to create customized posthoc comparisons. (GLM Only)                                                                                                 |
| BY, FORMAT, LABEL, WHERE | These statements are common to most procedures, and may be used here.                                                                                           |



# Using the MEANS or LSMEANS Statement

- When you perform a one-way ANOVA, typically there is a two-step procedure:
  - (1) test the null hypothesis to determine whether any significant differences exist, and
  - (2) if  $H_0$  is rejected, run subsequent multiple comparison tests to determine which differences are significantly different.
- Pairwise comparison of means can be performed using one of several multiple comparison tests specified using the MEANS statement, which has the following format (where independentvar is a CLASS variable):

**MEANS independentvar/typecomparison <meansoptions>;**

- For PROC GLM, use the LSMEANS statement:

**LSMEANS independentvar / typecomparison <meansoptions>;**



Table. Common type comparison options for the PROC ANOVA or GLM MEANS Statement (Options following the slash /)

| <b>Option</b>     | <b>Explanation</b>                                                                               |
|-------------------|--------------------------------------------------------------------------------------------------|
| BON               | Bonferroni t-tests of difference                                                                 |
| DUNCAN            | Duncan's multiple range test                                                                     |
| SCHEFFE           | Scheffe multiple comparison                                                                      |
| SNK               | Student Newman Keuls multiple range test                                                         |
| LSD               | Fisher's Least Significant Difference                                                            |
| TUKEY             | Tukey's studentized range test                                                                   |
| DUNNETT ( ' x ' ) | Dunnett's test—compare to a single control, where 'x' is the category value of the control group |
| ALPHA=pvalue      | Specifies the significance level for comparisons (default: 0.05)                                 |
| CLDIFF            | Requests that confidence limits be included in the output.                                       |



|                                                                                                         |                                                                                          |
|---------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| (Table continued)                                                                                       |                                                                                          |
| Common type comparison options for the <b>PROC GLM</b> LSMEANS Statement (options following the slash / |                                                                                          |
| ADJUST=option                                                                                           | Specify type of multiple comparison. Examples are BON, DUNCAN, SCHFEE, SNK, LSD, DUNNETT |
| PDIF=                                                                                                   | Calculates p-values base (default is T). You can also specify TUKEY or DUNNETT options.  |

# EXAMPLE

```
PROC ANOVA DATA=ACHE;
```

```
CLASS BRAND;
```

```
MODEL RELIEF=BRAND;
```

```
MEANS BRAND/TUKEY;
```

```
TITLE 'COMPARE RELIEF ACROSS MEDICINES - ANOVA
EXAMPLE';
```

```
RUN;
```

```
QUIT;
```

CLASS defines the grouping variable, BRAND.

The MODEL statement indicates you are wanting to test if BRAND can predict mean RELIEF.

The MEANS statement is used for a post hoc test (if  $H_0$  is rejected) to determine which means are different.



# Results of a One-Way ANOVA

- The primary results for a One-Way ANOVA test are in the following table:

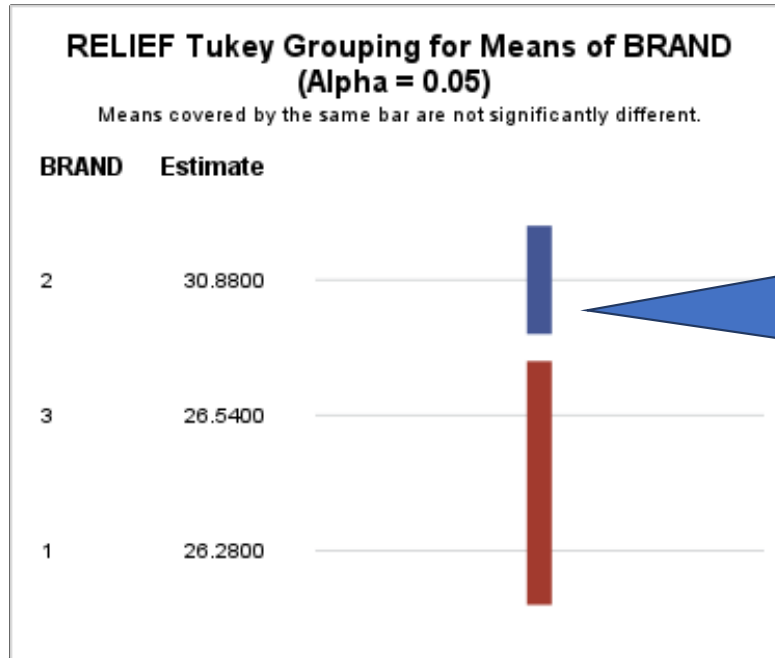
| Source | DF | Anova SS    | Mean Square | F Value | Pr > F |
|--------|----|-------------|-------------|---------|--------|
| BRAND  | 2  | 66.77200000 | 33.38600000 | 7.14    | 0.0091 |

The p-value is used to decide whether or not to reject the null hypothesis. Typically, if  $p < 0.05$ , you reject  $H_0$ . If you reject  $H_0$ , it indicates that some means (by group) are different, so you proceed to look at the post hoc results.



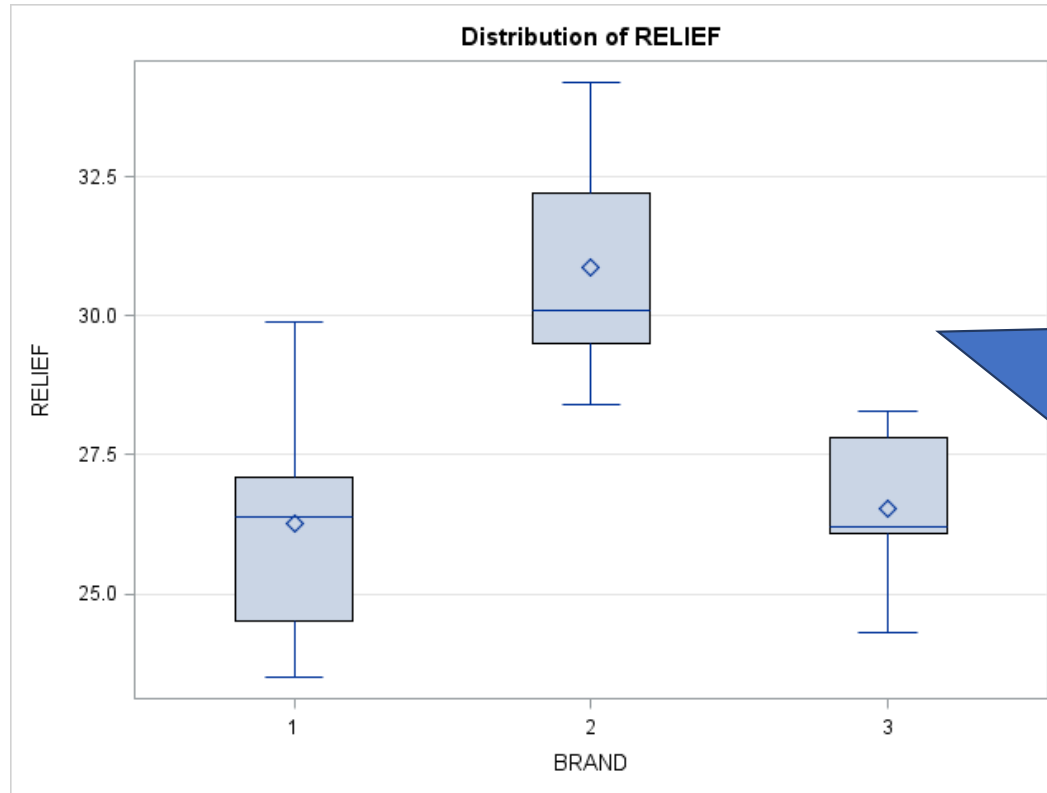
# Post Hoc Multiple Comparisons test – Tukey Test

- Depending on your SAS version, you may get this description of grouping using TUKEY comparisons – both come to the same conclusion.



In this graph, (at the 0.05 significance level) the mean for BRAND 2 (30.89) is different from the means of group 3 and 1 (26.54 and 26.28).

# Graphical Comparison of Groups



This graph reinforces the statistical results --- that groups 1 and 3 are very similar, but the mean for group 2 is larger than for either groups 1 or 2.

# Multiple Comparison Test Using Confidence Limits

- Using this code for the comparison test:  
**MEANS BRAND/TUKEY CLDIFF;**
- Results in this table

| Comparisons significant at the 0.05 level<br>are indicated by ***. |                                |                                       |        |     |
|--------------------------------------------------------------------|--------------------------------|---------------------------------------|--------|-----|
| BRAND<br>Comparison                                                | Difference<br>Between<br>Means | Simultaneous 95% Confidence<br>Limits |        |     |
| 2 - 3                                                              | 4.340                          | 0.691                                 | 7.989  | *** |
| 2 - 1                                                              | 4.600                          | 0.951                                 | 8.249  | *** |
| 3 - 2                                                              | -4.340                         | -7.989                                | -0.691 | *** |
| 3 - 1                                                              | 0.260                          | -3.389                                | 3.909  |     |
| 1 - 2                                                              | -4.600                         | -8.249                                | -0.951 | *** |
| 1 - 3                                                              | -0.260                         | -3.909                                | 3.389  |     |

In this table, mean differences are compared. For example, the first line tests the difference between means for groups 2 minus 3 = 4.340 and reports a 95% CL of 0.691 to 7,989. Since this range does not include 0.0, the difference is considered statistical different at the 0.05 significance level. The \*\*\* indicates a 0.05 significant difference for that comparison



# Multiple Comparisons using p-values

- Using **PROC GLM** instead of PROC ANOVA, and using this code for the comparison test:

**LSMEANS BRAND/ PDIFF;**

- Results in this table:

| Least Squares Means for effect BRAND<br>Pr >  t  for H0: LSMean(i)=LSMean(j)<br>Dependent Variable: RELIEF |        |        |        |
|------------------------------------------------------------------------------------------------------------|--------|--------|--------|
| i/j                                                                                                        | 1      | 2      | 3      |
| 1                                                                                                          |        | 0.0056 | 0.8524 |
| 2                                                                                                          | 0.0056 |        | 0.0080 |
| 3                                                                                                          | 0.8524 | 0.0080 |        |

This table reports the results of mean comparisons. For example, the comparison of mean 1 vs 3 reports a p-value of 0.8524, indicating that the difference in means is NOT statistically different.

The comparison of means 2 vs 3 is statistically different at  $p=0.0080$ .



# MODEL TYPES

- Typical Model Statement

MODEL dependentvar = independentver(s)

| Type       | Dependent  | Independent                   |
|------------|------------|-------------------------------|
| ANOVA      | Group      | Quantitative                  |
| Linear     | Continuous | Quantitative and /or grouping |
| Logistic   | Binary     | Quantitative and /or grouping |
| Chi-Square | Group      | Group                         |



# Binary Logistic Regression

- Binary logistic regression models are based on a dependent variable that can take on only one of two values, such as presence or absence of a disease, deceased or not deceased, married or unmarried, and so on.
- In this setting, the independent (sometimes called explanatory or predictor) variables **are used for predicting the probability of occurrence of an outcome** (such as mortality).



# LOGISTIC ANALYSIS BASICS: Logistic Regression Model

- The basic form of the logistic equation is

$$p = \frac{e^{b_0 + b_1X_1 + b_2X_2 + \dots + b_kX_k}}{1 + e^{b_0 + b_1X_1 + b_2X_2 + \dots + b_kX_k}}$$

where  $X_1, \dots, X_k$  are the  $k$  independent variables,  $p$  is the probability of occurrence of the outcome of interest (which lies between 0 and 1),  $\beta_i$  is the coefficient on the independent variable  $X_i$ , and  $\beta_0$  is a constant term. As in linear regression, the parameters of this theoretical model are estimated from the data, resulting in the prediction equation

$$\hat{p} = \frac{e^{b_0 + b_1X_1 + b_2X_2 + \dots + b_kX_k}}{1 + e^{b_0 + b_1X_1 + b_2X_2 + \dots + b_kX_k}}$$



# Hypotheses for the Logistic Model

- Any variable with a zero coefficient in the theoretical model is not useful in predicting the probability of occurrence.
- SAS reports tests of the **null hypothesis** that all of the  $\beta_i$ 's,  $i = 1, \dots, k$  are zero.
- If this null hypothesis is **not rejected**, then there is **no statistical evidence that the independent variables as a group are useful in the prediction**.
- If the overall test is rejected, then we conclude that at least some of the variables are useful in the prediction. For each  $\beta_i = 1, \dots, k$ , SAS reports the results of the tests.
- The hypotheses test are thus

$H_0: \beta_i = 0$ : The  $i$ th independent variable is not predictive of the probability of occurrence.

$H_a: \beta_i \neq 0$ : The  $i$ th independent variable is predictive of the probability of occurrence.





# Understanding Odds and Odds Ratios

- Another use of the logistic model is the calculation of odds ratio (OR) for each independent variable.
- **The odds of an event measures the expected number of times an event will occur relative to the number of times it will not occur .** Thus, if the odds ratio of an event is 5, this indicates that we expect five times as many occurrences as non-occurrences. An odds of 0.2 ( $=1/5$ ) would indicate that we expect five times as many non occurrences as occurrences.



# PERFORMING A LOGISTIC ANALYSIS USING PROC LOGISTIC

- PROC LOGISTIC is the SAS procedure that allows you to analyze the data using a binary logistic model.
- An abbreviated syntax for this statement

CLASS variables are categorical  
such as Gender “Male” and  
“Female” or Cancer Stage 1, 2, 3

**PROC LOGISTIC** *<options>*;

**CLASS** variables;

**MODEL** dependent\_var *<(variable\_options)>* =  
*<(independent\_variables)>* *<options>*;

The dependent variable is binary – that is it  
takes on only one of two values.



# What You Are Predicting

- By default, SAS assumes that the outcome predicted (with p) in the logistic regression equation corresponds to the case in which the dependent variable is 0. (or the lowest number or alphabetic character.)
- If, for example, you have a variable such as DISEASE with DISEASE=0 indicating the disease is absent and DISEASE= 1 indicating the disease is present, then SAS will predict the probability of "disease absent" by default.



Table Common Options for PROC LOGISTIC

| Option          | Explanation                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DATA = dataname | Specifies which dataset to use.                                                                                                                                                                                                                                                                                                                                                                                                  |
| DESCENDING      | Reverses the sorting order for the levels of the response variable. By default, the procedure will predict the outcome corresponding to the lower value of the dichotomous dependent variable. So, if the dependent variable takes on the values 0 and 1, then by default SAS predicts the probability that the dependent variable is 0 unless you use the DESCENDING option. (See information about the (EVENT=) option below.) |
| ALPHA= value    | Specifies significance level for confidence limits.                                                                                                                                                                                                                                                                                                                                                                              |
| NOPRINT         | Suppresses output.                                                                                                                                                                                                                                                                                                                                                                                                               |
| SIMPLE          | Displays descriptive statistics.                                                                                                                                                                                                                                                                                                                                                                                                 |
| PLOTS= option   | In current versions of SAS, the Odds Ratio plots is displayed by default. Use PLOTS=NONE; to suppress this plot. PLOTS=ALL produces a number of plots include ROC, and influence diagnostics.                                                                                                                                                                                                                                    |



| Common Statements for PROC LOGISTIC (Table continued) |                                                                                                                                                                     |
|-------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MODEL<br>depvar=indvar(s);                            | Specifies the dependent and independent variables for the analysis. More specifically, it takes the form<br><br>MODEL depvariable=indvariable(s);                   |
| CLASS variable<br>list;                               | Specifies classification (either categorical character or discrete numeric) variables for the analysis. They can be numeric or character. See text for more details |
| ODDSRATIO 'label'<br>var;                             | Creates a separate table with Odds Ratio Estimates and Wald Confidence Intervals. See text for more details                                                         |
| OUTPUT out=NAME;                                      | Creates an output dataset with all predictors and response probabilities. For example<br><br>OUTPUT OUT=MYFILE P=PRED;                                              |
| BY, FORMAT,<br>LABEL, WHERE                           | These statements are common to most procedures, and may be used here.                                                                                               |



# The DESCENDING Option in the MODEL Statement

- The MODEL statement specifies the dependent (outcome) variable as well as the independent variables. For example,

**PROC LOGISTIC;**

**MODEL** DEPVAR = INDVAR1 INDVAR2 etc/options;

- Care must be taken as to how the DEPVAR is defined.
- For example, if your dependent variable is FAIL (0 means not failed & 1 means failed), then SAS will predict FAIL=0.
- To reverse the default prediction, use the **DESCENDING** option. When that option is included in the PROC LOGISTIC statement, FAIL=1 will be modeled instead of FAIL=0. Thus:

**PROC LOGISTIC** **DESCENDING**;

**MODEL** DEPVAR = INDVAR1 INDVAR2 etc/options;



## Another Way to Specify What is Predicted

- Another way to choose the value modeled is to explicitly define it in the MODEL statement. For example,

**MODEL FAIL(EVENT='1') = *independentvars*;**

- Causes SAS to use 1 as the value to model for the dependent variable FAIL.
- We recommend that you choose to use either the DESCENDING option or the EVENT= option to specify a value of the response variable to predict.



Table. Common MODEL statement options for PROC Logistic

| Option                 | Explanation                                                                         |
|------------------------|-------------------------------------------------------------------------------------|
| EXPB                   | Displays the exponentiated values of parameter, (the odds ratios.)                  |
| SELECTION=type         | Specifies variable selection method (examples are STEPWISE, BACKWARD, and FORWARD). |
| SLENTY=value           | Specifies significance level for entering variables. Default is 0.05.               |
| SLSTAY=value           | Specifies significance level for removing variables. Default is 0.05.               |
| LACKFIT                | Requests Hosmer-Lemershow test                                                      |
| RISKLIMITS             | Requests confidence limits for odds ratios.                                         |
| CTABLE<br>PPROB=(list) | Requests a classification table report. PPROB specifies cutpoints to display.       |
| INCLUDE=n              | Includes first n independent variables in model.                                    |
| OUTROC=name            | Outputs ROC values to a dataset.                                                    |





# The CLASS Statement

- If a model includes independent variables that are categorical, they must be indicated in a CLASS statement.
- For example, suppose the variable CATNUM is (i.e., 1, 2, 3) and CARALPH is character (i.e., A, B, C). Your LOGISTIC code might be:

```
CLASS CATNUM CATALPH;
```

Categorical variables identified in the CLASS statement.

```
MODEL Y = X1 X2 . . . Xk CATNUM CATALPH;;
```

And those same categorical variables are used as independent variables in the model.



# USING SIMPLE LOGISTIC ANALYSIS

- A simple logistic model is one that has only one predictor (independent) variable. This predictor variable can be either a binary or a quantitative measure.

```
PROC LOGISTIC DATA=STAT3505.ACCIDENTS DESCENDING;
```

```
MODEL DEAD=PENETRATE / RISKLIMITS;
```

```
RUN;
```

DEAD (which is coded 0 and 1) is what is being modeled. The DESCENDING option tells SAS to model DEAD=1

PENETRATE is a 0, 1 dichotomous variable

RISKLIMITS requests ORs to be output.



# EXAMPLE

```
PROC LOGISTIC DATA=STAT3505.ACCIDENTS DESCENDING;
 MODEL DEAD=PENETRATE / RISKLIMITS;
TITLE 'Trauma Data Model Death by Penetration Wound';
RUN;
```

- In this example, the independent variable PENETRATE, which is a 0,1 variable, is used to predict death (DEAD=1), so the DESCENDING option is used.

# Exercise Continued...

- Run the program. Pay special attention to this statement in the output “**Probability modeled is dead=1.**”
- It indicates what is modelled – make sure it is the output you want to model. In this case you are predicting death.
- Also note the “Response Profile”

| Response Profile |      |                 |
|------------------|------|-----------------|
| Ordered Value    | dead | Total Frequency |
| 1                | 1    | 103             |
| 2                | 0    | 3580            |

In this case there are 103 deaths and 3580 non-deaths. IMPORTANT: Make sure these numbers are what you expect from your data.



# Continued... Logistic Model Results

- Your primary tables of interest in the output are the estimates for the model:

| Analysis of Maximum Likelihood Estimates |    |          |                |                 |            |
|------------------------------------------|----|----------|----------------|-----------------|------------|
| Parameter                                | DF | Estimate | Standard Error | Wald Chi-Square | Pr > ChiSq |
| Intercept                                | 1  | -3.6988  | 0.1111         | 1108.0853       | <.0001     |
| penetrate                                | 1  | 1.2697   | 0.2584         | 24.1519         | <.0001     |

Indicates if the variable (PENETRATE) is a good predictor. Since  $p < 0.001$ , we conclude that it is (when  $p < 0.05$ )

- And the Odds Ratios

| Odds Ratio Estimates and Wald Confidence Intervals |        |          |                       |       |
|----------------------------------------------------|--------|----------|-----------------------|-------|
| Effect                                             | Unit   | Estimate | 95% Confidence Limits |       |
| penetrate                                          | 1.0000 | 3.560    | 2.145                 | 5.906 |

If the predictor is shown to be important, the OR gives us an idea of its strength in predicting the outcome. In this case  $OR = 3.56$

$OR = 3.56$  indicates that the odds of a person's dying who had a penetrating wound is 3.56 greater than that for a person who did not suffer this type of wound.



# Change DEAD=PENETRATE to DEAD=ISS, Which is a Continuous Variable

```
PROC LOGISTIC DATA=STAT3505.ACCIDENTS DESCENDING;
```

```
MODEL DEAD=ISS / RISKLIMITS;
```

```
RUN;
```

- In this model only ISS (injury Severity Score) has changed – it is a **continuous variable** whereas PENETRATE was dichotomous.

| Analysis of Maximum Likelihood Estimates |    |          |                |                 |            |
|------------------------------------------|----|----------|----------------|-----------------|------------|
| Parameter                                | DF | Estimate | Standard Error | Wald Chi-Square | Pr > ChiSq |
| Intercept                                | 1  | -5.4444  | 0.2105         | 668.7126        | <.0001     |
| ISS                                      | 1  | 0.1056   | 0.00721        | 214.5334        | <.0001     |

| Odds Ratio Estimates and Wald Confidence Intervals |        |          |                       |       |
|----------------------------------------------------|--------|----------|-----------------------|-------|
| Effect                                             | Unit   | Estimate | 95% Confidence Limits |       |
| ISS                                                | 1.0000 | 1.111    | 1.096                 | 1.127 |

ISS is also an important predictor of death...

The odds ratio for ISS is 1.11



# OR for Dichotomous vs Continuous Variables

## IMPORTANT

- OR is interpreted *differently* for PENETRATE than for ISS as ISS is a quantitative measure and PENETRATE is a binary measure. ***Pay close attention to this difference.***
- For PENETRATE OR=3.56 indicates that the odds of a person's dying who had a penetrating wound is 3.56 times greater than that for a person who did not suffer this type of wound.
- For ISS OR=1.11 indicates that **for each unit increase** in ISS, the odds of dying increases by 1.11. (or 11%)



# When OR is Less than 1

- An Odds Ratio less than 1 can also be important.
- For example, suppose a significant OR in this dataset (say AGE) was .89.
- It would be interpreted as – for each increase in AGE year, the odds of dying is LESS by about 11%.
- One way to look at it is variables with a high OR are predictive of death (the predicted outcome) and variables with an OR less than 1 are **protective** of death (the predicted outcome).





# Results – First 3 Tables

| Model Information         |                      |
|---------------------------|----------------------|
| Data Set                  | C:/SASDATA/ACCIDENTS |
| Response Variable         | dead                 |
| Number of Response Levels | 2                    |
| Model                     | binary logit         |
| Optimization Technique    | Fisher's scoring     |

|                             |      |
|-----------------------------|------|
| Number of Observations Read | 3683 |
| Number of Observations Used | 3683 |

| Response Profile |      |                 |
|------------------|------|-----------------|
| Ordered Value    | dead | Total Frequency |
| 1                | 1    | 103             |
| 2                | 0    | 3580            |

Probability modeled is dead=1.

- Top table – Summary information about model (note Response Variable)
- Middle Table – Make sure observations are as expected
- Bottom Table – Make sure the key variable (In this case DEAD) has the expected number of obs.
- Note the Probability modeled is DEAD = 1



# Results Continued

| Analysis of Maximum Likelihood Estimates |    |          |                |                 |            |
|------------------------------------------|----|----------|----------------|-----------------|------------|
| Parameter                                | DF | Estimate | Standard Error | Wald Chi-Square | Pr > ChiSq |
| Intercept                                | 1  | -5.4444  | 0.2105         | 668.7126        | <.0001     |
| ISS                                      | 1  | 0.1056   | 0.00721        | 214.5334        | <.0001     |

Note these values

| Odds Ratio Estimates and Wald Confidence Intervals |        |          |                       |       |
|----------------------------------------------------|--------|----------|-----------------------|-------|
| Effect                                             | Unit   | Estimate | 95% Confidence Limits |       |
| ISS                                                | 1.0000 | 1.111    | 1.096                 | 1.127 |

- Maximum Likelihood table  
– Note the ISS row – the p-value is  $p < .0001$  indicating that ISS is a good predictor of DEAD. (Recall ISS is a continuous Variable)
- The ODDS RATIO estimate is 1.111



# Simple Logistic Results

- The logistic equation based on estimates given in the Maximum-Likelihood Estimates tables is

$$\hat{p} = \frac{e^{-5.444.1056 * ISS}}{1 + e^{-5.444.1056 * ISS}}$$

- Where p-hat is the prediction calculated for a value of ISS.
- Because the code

**OUTPUT OUT=LOGOUT PREDICTED=PROB;**

was used in the program, a file named LOGOUT contains the values of p-hat (labeled PROB) for each value of ISS.

# What is ODS?

- ODS, or **Output Delivery System**, is a method for controlling the output from SAS® procedures. ODS began with version 8 and continues with added enhancements in more recent versions.
- Prior to SAS 9.3, SAS output appeared in the Output Window. This output listing is like a monospaced typewriter font (with no graphics) and there are few options that allow you to control the “look” of the listing.
- Beginning with 9.3 **default** output is HTML type output.



# SPECIFYING THE ODS OUTPUT FORMAT AND DESTINATION

- The SAS ODS is set up so that you "turn on" or initiate output into a designated output format. Once the output format has been initiated, SAS procedures send information to that output format. You can send output from one or more procedures to the output stream.

ODS **OUTDESTINATION** <OPTIONS>;

Used to tell SAS to start outputting results to a specified **output** type.

- To end the ODS output, use the **CLOSE** statement:

ODS **OUTDESTINATION** **CLOSE**;

Tells SAS that you are finished outputting results.



What do you do if your output is not showing up in the Results Viewer?

**ODS PREFERENCES;**

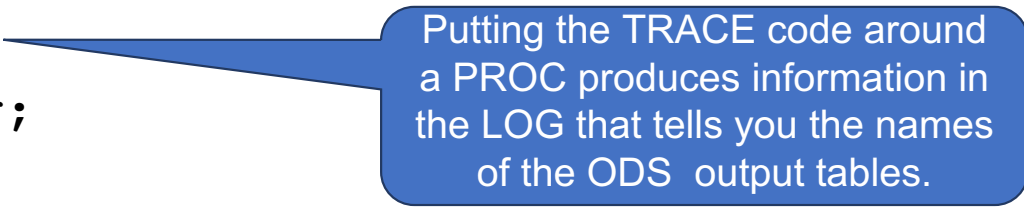
(Or end SAS and restart)



# USING ODS TO SELECT SPECIFIC OUTPUT TABLES

- SAS procedures often output a lot of information you don't want or need. In ODS output, each part of the output is contained in a table.
- Using ODS options, you can customize **which tables you want SAS to output** to the ODS
- To include or exclude a table from the output, you first need to know the table's name. You can discover this information by using the ODS TRACE command in the following way:

```
ODS TRACE ON;
PROC whatever;
ODS TRACE OFF;
```

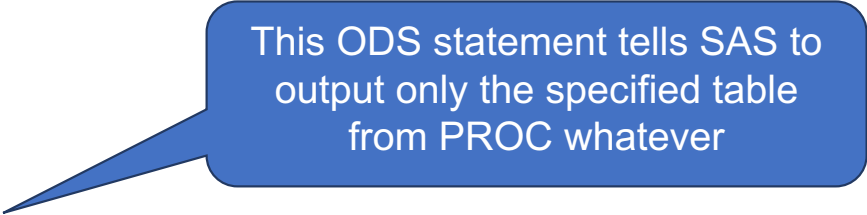


Putting the TRACE code around a PROC produces information in the LOG that tells you the names of the ODS output tables.



# Specifying which tables to display

- Once you know the names of the tables you want to display (using TRACE), use the following code to make that request:



This ODS statement tells SAS to output only the specified table from PROC whatever

```
ODS SELECT name-of-tables-to-include;
PROC whatever;
Etc...
```



# EXAMPLE:

```
DATA TABLE;
 INPUT A B COUNT;
 DATALINES;
0 0 12
0 1 15
1 0 18
1 1 3
;
ODS TRACE ON;

PROC FREQ; WEIGHT COUNT;
 TABLES A*B /CHISQ;
 TITLE 'CHI-SQUARE ANALYSIS FOR A 2X2 TABLE';
RUN;

ODS TRACE OFF;
```

Use TRACE to  
discover names of  
output tables.



# Results of running this code:

```
DATA TABLE;
INPUT A B COUNT;
DATALINES;
0 0 12
0 1 15
1 0 18
1 1 3
;
ODS TRACE ON;
PROC FREQ; WEIGHT COUNT;
 TABLES A*B /CHISQ;
 TITLE 'CHI-SQUARE ANALYSIS FOR A 2X2 TABLE';
RUN;
ODS TRACE OFF;
```

Note names of  
tables in the SAS  
Log file

```
Output Added:

Name: CrossTabFreqs
Label: Cross-Tabular Freq Table
Template: Base.Freq.CrossTabFreqs
Path: Freq.Table1.CrossTabFreqs

Output Added:

Name: ChiSq
Label: Chi-Square Tests
Template: Base.Freq.ChiSq
Path: Freq.Table1.ChiSq

Output Added:

Name: FishersExact
Label: Fisher's Exact Test
Template: Base.Freq.ChiSqExactFactoid
Path: Freq.Table1.FishersExact

```



# Use that information:

- **STEP 2:** Use **SELECT *tablename*s** to produce output that ONLY contains the tables of interest:

**ODS SELECT CROSSTABFREQS CHISQ;**

**PROC FREQ;WEIGHT COUNT;**

**TABLES A\*B /CHISQ;**

**RUN;**

- (You can also use **ODS EXCLUDE** to exclude certain tables from output.)

| Frequency<br>Percent<br>Row Pct<br>Col Pct | Table of A by B |       |       |        |
|--------------------------------------------|-----------------|-------|-------|--------|
|                                            | A               | B     |       |        |
|                                            |                 | 0     | 1     | Total  |
| 0                                          |                 | 12    | 15    | 27     |
|                                            |                 | 25.00 | 31.25 | 56.25  |
|                                            |                 | 44.44 | 55.56 |        |
|                                            |                 | 40.00 | 83.33 |        |
| 1                                          |                 | 18    | 3     | 21     |
|                                            |                 | 37.50 | 6.25  | 43.75  |
|                                            |                 | 85.71 | 14.29 |        |
|                                            |                 | 60.00 | 16.67 |        |
| Total                                      |                 | 30    | 18    | 48     |
|                                            |                 | 62.50 | 37.50 | 100.00 |

| Statistic                   | DF | Value   | Prob   |
|-----------------------------|----|---------|--------|
| Chi-Square                  | 1  | 8.5841  | 0.0034 |
| Likelihood Ratio Chi-Square | 1  | 9.1893  | 0.0024 |
| Continuity Adj. Chi-Square  | 1  | 6.9136  | 0.0086 |
| Mantel-Haenszel Chi-Square  | 1  | 8.4053  | 0.0037 |
| Phi Coefficient             |    | -0.4229 |        |
| Contingency Coefficient     |    | 0.3895  |        |
| Cramer's V                  |    | -0.4229 |        |



# CAPTURING INFORMATION FROM ODS TABLES

- Once you know the name of an output table, you can use ODS to save the table contents into a SAS data file using:
- **ODS OUTPUT *NAMEOFTABLE*=*OUTPUTDATASET*;**

You learn the name of the table using the TRACE statement

When you output a table using ODS OUTPUT, you are creating a SAS data set.



# EXAMPLE

- This example shows how to capture a specific statistics from the output.

```
DATA WT;
INPUT WEIGHT @@;
DATALINES;
64 71 53 67 55 58
77 57 56 51 76 68
;
ODS TRACE ON;
PROC MEANS DATA=WT;
RUN;
ODS TRACE OFF;RUN;
QUIT;
```

- Run this program



# Capturing Output Continued...

- The ODS TRACE outputs this information in Log:

Output Added:

-----

**Name: Summary**

Label: Summary statistics

Template: base.summary

Path: Means.Summary

Summary is the name of the output table containing the statistics from the PROC.

