Our future health built with care

Introduction to SAS

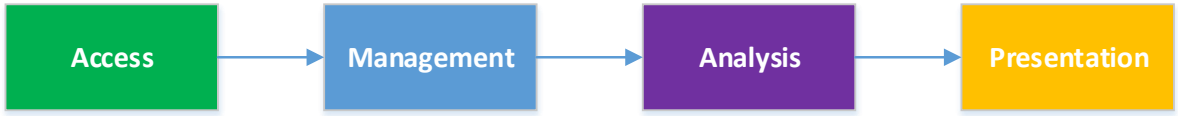
SEPTEMBER 27, 2018

Ontario
Cancer Care Ontario

Contents

- Typical analytics workflow
- SAS basics
- Applied example using hypothetical diabetes question
  - Import and access data
  - Explore data
  - Append data
  - Clean and standardize data
  - Merge data
  - Simple analysis
  - Alternative linking method using PROC SQL
  - Transforming data (wide to long, long to wide)

Basic Analytics Workflow



Workflow	Description
Access	Import or access data required for analysis.
Management	Clean, standardize, and prepare data for analysis.
Analysis	Analytical work to obtain necessary information from data.
Presentation	Prepare and communicate information for stakeholders.

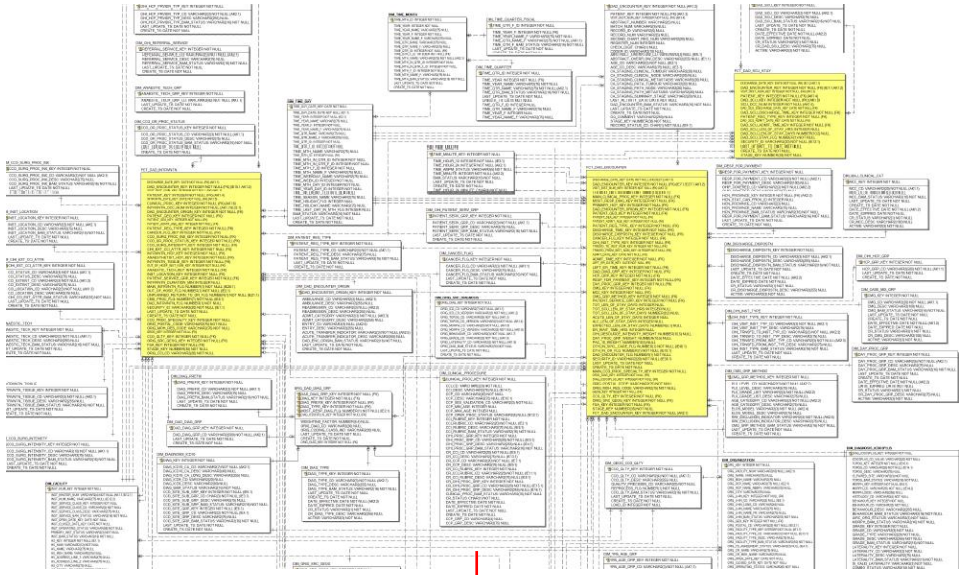
Comments on Data Management

Data preparation should be not overlooked

- SAS is often used for both data management and analytics
- Data is rarely clean and analytically ready for use
- Health administrative data often not standardized in the same way across datasets
- Data often stored in ways that are not analytically friendly (e.g., relational database)

Comments on Data Management *cont.*

Relational Database vs. Flat File



Efficient for storage, maintaining and updating, but not suitable for analyses.

Generally need data in this format to be "analytically ready".

SAS Basics

SAS Files

- Code can be stored in SAS programs → e.g., *Program.sas*
- Information on program execution can be stored in a log file → e.g., *Program Run.log*
- Data can be stored in native SAS files → e.g., *Program Dataset.sas7bdat*

SAS Syntax

- SAS code usually begins with a DATA or PROC step
- SAS steps usually ends with either a RUN or QUIT statement
- Statements end with a semicolon (**;**)
- Multiple statements can be written in a single line
- Statements are not case-sensitive, except inside quotations (e.g., “f” vs. “F”)

SAS Basics *cont.*

Example SAS Code

```
data new_data;
   set old_data;
run;
```

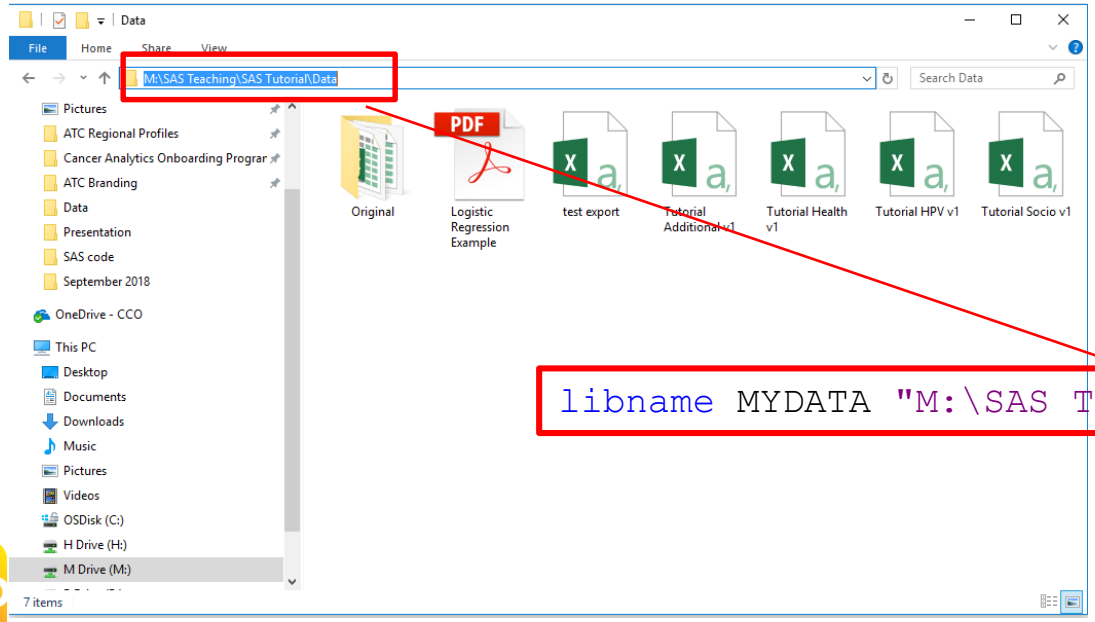
```
proc freq data=new_data;
   tables sex;
run;
```

- Names cannot be longer than 32 characters
- Can begin with a letter or underscore
- Spaces and other special characters cannot be used (e.g., !, ?, -. /, $, #, etc.)

SAS Basics *cont.*

SAS Libraries

- By default, SAS stores data in a temporary workspace (library referenced as WORK)
- To save or access permanent datasets, need to point SAS to directory
- LIBNAME tells SAS to create a reference to a directory
- Directory must already exist (i.e., LIBNAME cannot create a new folder)
- Contents in temporary workspace will be deleted when SAS session ends




```
libname MYDATA "M:\SAS Teaching\SAS Tutorial\Data";
```

SAS Basics *cont.*

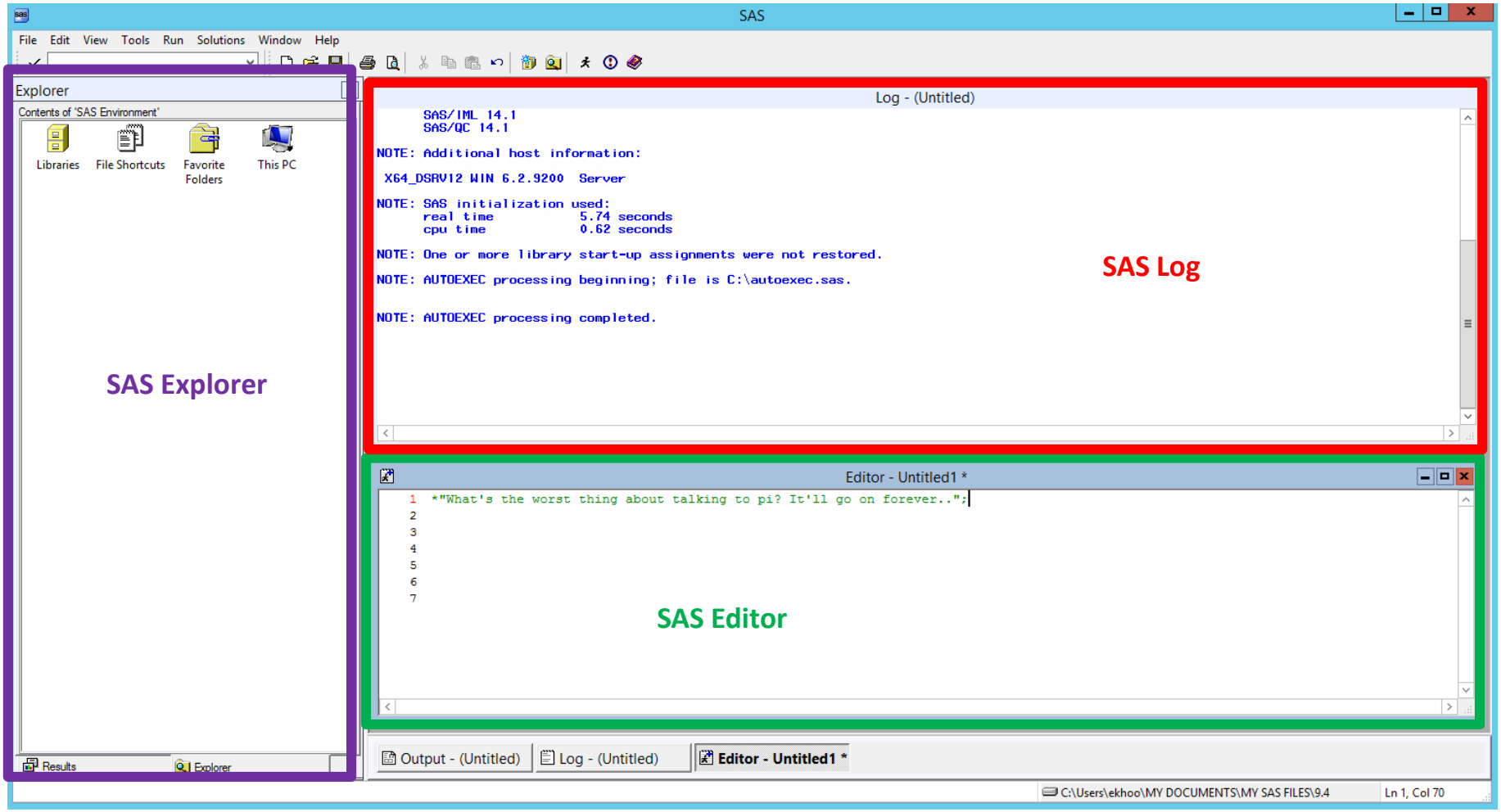
SAS Comments

- Good practice to annotate code by inserting comments
- Comments can be made using an asterisk and semicolon (e.g., `* comments ;`) or using a pair of forward slashes and asterisks (e.g., `/* comments */`)

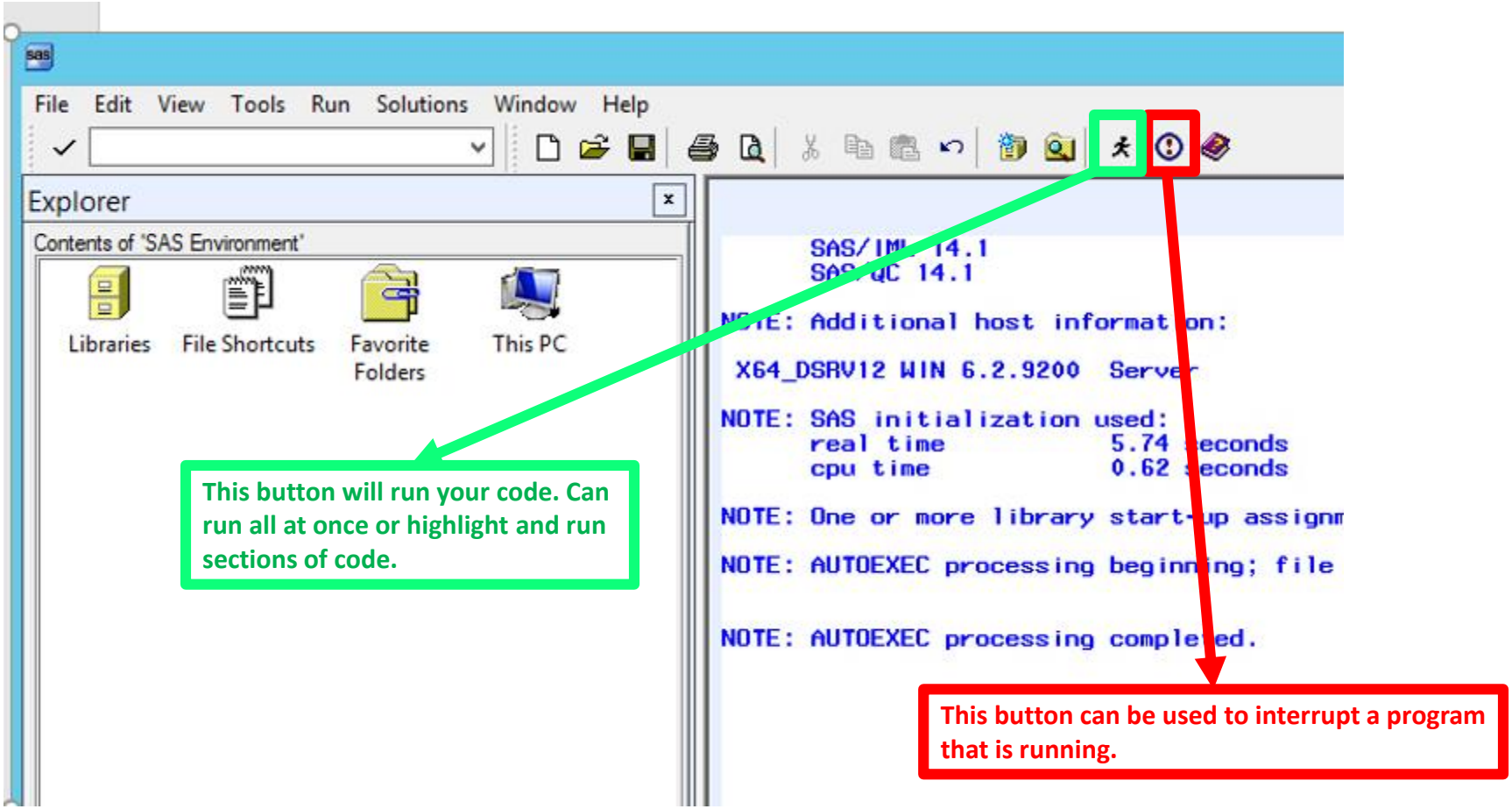
SAS Log

- **WARNINGS** (green) are non-critical errors and should be investigated even if program runs
  - E.g., reference a variable that does not exist in dataset
- **ERRORS** (red) are critical errors that prevent program from running
  - E.g. reference a dataset that does not exist
- Number of observations (or rows) and variables (or columns) for dataset (or table) will be displayed in the log
- Processing time also displayed – important for large datasets

SAS User Interface

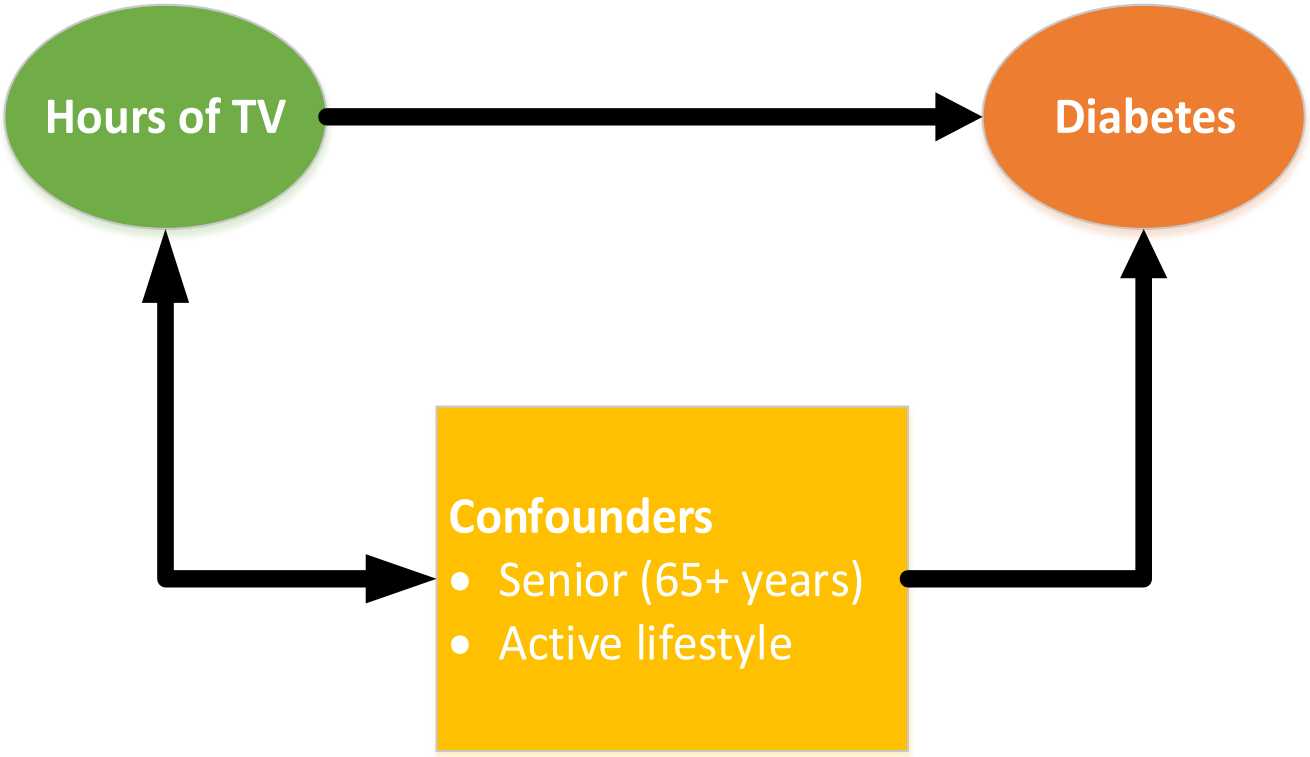


SAS User Interface *cont.*

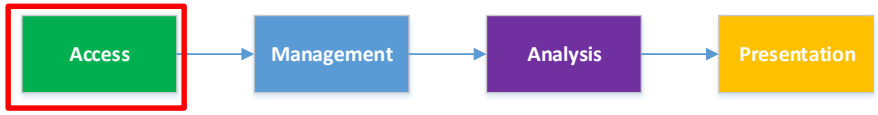


Hypothetical Scenario

- Researcher has a conceptual model of how TV viewing is associated with diabetes
- We will prepare data and build a statistical model to test this relationship



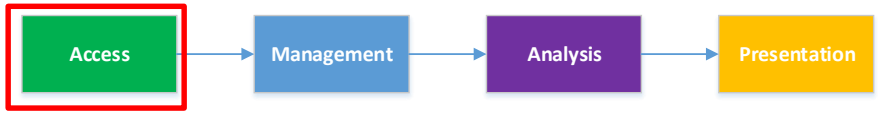
Fake Datasets



- Researcher has four datasets containing data to test conceptual model

CSV	Description	Record Granularity
Tutorial Socio	• Unique identifier = FAKE_HCN • Contains sociodemographics data for fake patients	• Patient-level
Tutorial Additional	• Unique identifier = FAKE_HCN • Contains sociodemographics data for additional fake patients – need to add these to Socio dataset	• Patient-level
Tutorial Health	• Contains clinical and health-related data for fake patients	• Patient-level
Tutorial HPV	• Contains HPV tests and cervical biopsy data for fake patients	• HPV Test-level

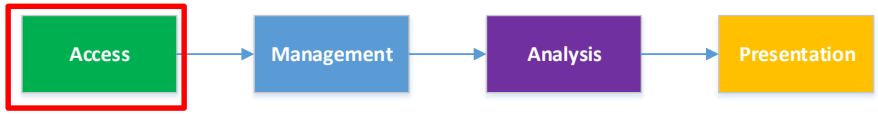
Setup Library



- Setup library so SAS knows where to read/write permanent data
- The library name acts as a reference for the folder path specified

```
libname TUTORIAL "M:\SAS Teaching\SAS Tutorial\Working";
```

Import Data



- We can import different file types into SAS to work with, including CSVs

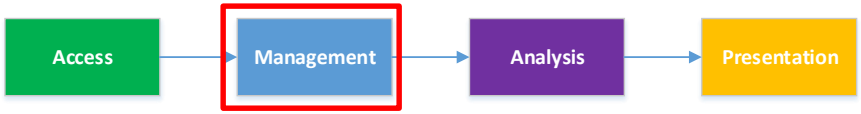
```
proc import datafile="M:\SAS Teaching\SAS Tutorial\Data\Tutorial Socio
v1.csv"
   out=SOCIO
   dbms=csv replace;
   guessingrows=100;
run;
```

Tells SAS the location of the file to import.

Tells SAS what to name file once imported.

Tells SAS the file type and to replace if the file already exists.
Change to excelcs if working with an Excel file.

Check Contents of Data

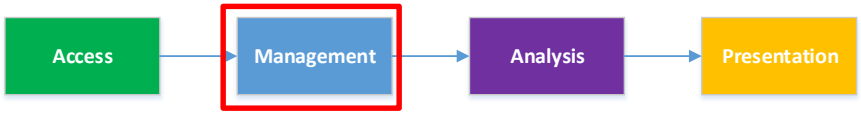


- We need to first append (add) data from Tutorial Additional to Tutorial Socio
- Before doing this, let's look at the metadata for the two datasets so we know what we're working with
- Pay attention to:
  - Number of observations and variables
  - Variable names, type, length

```
proc contents data=SOCIO order=varnum;
run;
```

Tells SAS to maintain order of variables in output. By default, SAS will order things alphabetically.

Append Datasets Together



- Code starting with DATA often referred to as a DATA step, ending with a RUN statement
- Usually used to read, write, or manipulate data
- Datasets can be appended together using the SET statement

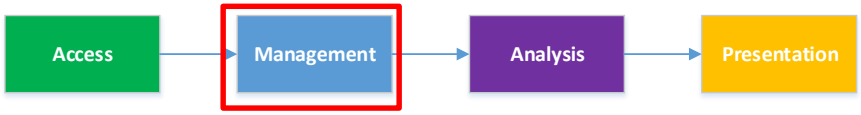
```
data COMBINE;
   length FNAME  $14.
          LNAME  $18.
          CITY $13.
          MARITAL_STATUS  $7.;
   set SOCIO ADDITIONAL;
run;
```

New dataset created named COMBINE

To prevent truncation, we specify the length of these variables since the lengths are different in the SOCIO and ADDITIONAL datasets.

SET tells SAS to read data from the SOCIO and ADDITIONAL datasets.

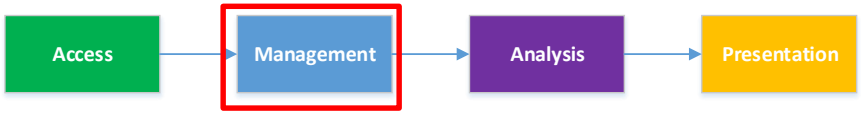
Explore Data



- Good practice to look at a sample of the data
- Two things are immediately apparent:
  - Values are not standardized (e.g., Homer vs. HOMER vs. HoMER)
  - Assuming FAKE_HCN is our unique identifier for a patient, we have duplicates

	FNAME	LNAME	CITY	MARITAL_STATUS	FAKE_HCN	DOB	SEX	EDUCATION
1	Homer	Simpson	Toronto	Y	9999999911	01JAN1960	M	2
2	HOMER	Simpson	TORONTO	Y	9999999911	01JAN1960	M	2
3	HOMER	SIMPSON	Toront	Y	9999999911	01JAN1960	M	2
4	HoMER	simpson	Tor	Y	9999999911	01JAN1960	M	2
5	HOMER	SIMPSON	6ix	Y	9999999911	01JAN1960	M	2
6	Marge	Simpson	Toronto	Y	9999999916	24MAR1970	F	3
7	Marge	Simpson	Toronto	Y	9999999916	24MAR1970	F	3

Explore Data *cont.*

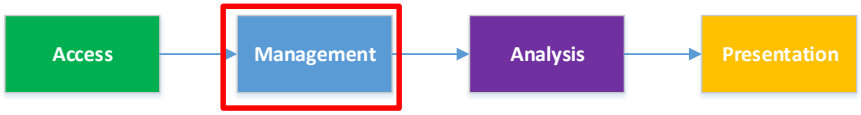


- PROC statements are used to run procedures or summarize data
- We can look at the unique values and frequency distribution of categorical data

```
proc freq data=COMBINE;
   tables CITY SEX EDUCATION MARITAL_STATUS;
run;
```

This just shows the basics. PROC FREQ is a powerful procedure with many options (e.g. cross-tabulate and conduct chi-sq tests, print results to dataset, etc.)

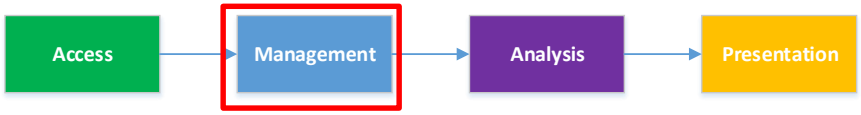
Explore Data *cont.*



CITY	Frequency	Percent	Cumulative Frequency	Cumulative Percent
6ix	1	0.55	1	0.55
?	1	0.55	2	1.10
Ajax	3	1.66	5	2.76
Aurora	2	1.10	7	3.87
Brampton	4	2.21	11	6.08
Fraserville	1	0.55	12	6.63
Georgetown	1	0.55	13	7.18
Guelph	3	1.66	16	8.84
Hamilton	6	3.31	22	12.15
Kirby	4	2.21	26	14.36
London	2	1.10	28	15.47
Markham	11	6.08	39	21.55
Milton	1	0.55	40	22.10
Mississauga	3	1.66	43	23.76
Newmarket	4	2.21	47	25.97
North York	14	7.73	61	33.70
NorthYork	1	0.55	62	34.25
Oakville	4	2.21	66	36.46
Oshawa	3	1.66	69	38.12
Peterborough	3	1.66	72	39.78
Pickering	4	2.21	76	41.99
Richmond Hill	8	4.42	84	46.41
Rigel VII	2	1.10	86	47.51
Scarborough	10	5.52	96	53.04
TOR	2	1.10	98	54.14
TORONTO	1	0.55	99	54.70
Tor	1	0.55	100	55.25
Toront	1	0.55	101	55.80
Toronto	64	35.36	165	91.16
Unknown	3	1.66	168	92.82
Uxbridge	2	1.10	170	93.92
Vaughan	7	3.87	177	97.79
Whitby	2	1.10	179	98.90
toronto	2	1.10	181	100.00

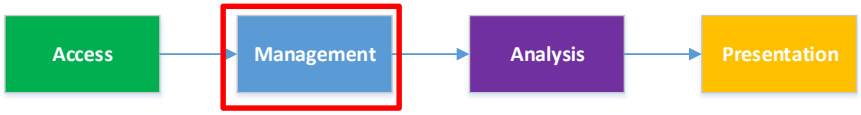
Frequency Missing = 5

Clean & Standardize Data



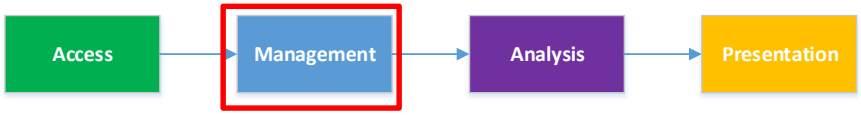
- There are many ways to clean/standardize data – key is to be consistent and transparent
- Noticed inconsistencies for:
  - FNAME
  - LNAME
  - CITY
  - MARITAL_STATUS
- We can recode values using if-then statements in a DATA step by either creating a new variable with the correct coding or overwriting the original values

Clean & Standardize Data *cont.*



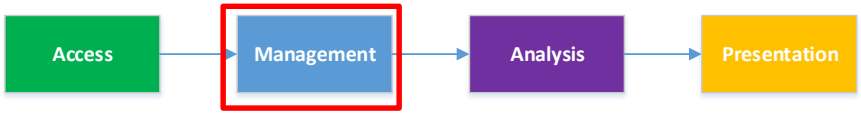
- Can make FNAME and LNAME consistent by using built in SAS functions
- `FNAME=propcase(FNAME);`
  - This will recode all values in FNAME to be propercase
  - E.g., HOMER becomes Homer
- The same can be done to LNAME so that SIMPSON becomes Simpson

Clean & Standardize Data *cont.*



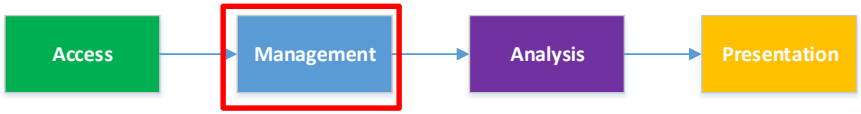
- CITY is a bit trickier since there are numerous values for Toronto (6ix, TORONTO, Tor, Toront, Toronto)
- One option is to use if-then statements for each variation
- Another option is to standardize values (e.g., make all values capitalized), and then use a search function
- `if upcase(CITY) in: ('6IX','TOR') then CITY='Toronto';`
  - Upcase(CITY) temporarily makes all values capitalized
  - In: (‘6IX’,’TOR’) tells SAS to search for values that start with either 6IX or TOR
  - The statement then converts these values to ‘Toronto’ to make things consistent

Clean & Standardize Data *cont.*



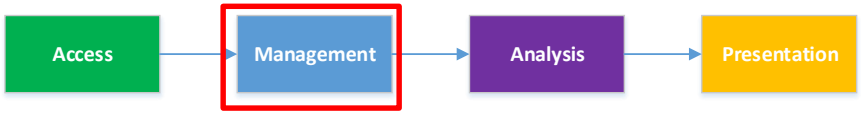
- We should also identify observations that might be missing FAKE_HCN or DOB since these are important variables we need
- `if missing(FAKE_HCN) then HCN_MISSING=1;`
- `else HCN_MISSING=0;`
- `if missing(DOB) then DOB_MISSING=1;`
- `else DOB_MISSING=0;`
  - Missing() can be used to identify missing values in a variable
  - If a value is missing, we flag that observation by creating a new variable for each

Clean & Standardize Data *cont.*




```
data COMBINE2 (drop=EDUCATION rename=(EDUCATION2=EDUCATION));
  set COMBINE;
  length EDUCATION2 $25.;
  if missing(FAKE_HCN) then HCN_MISSING=1;
  else HCN_MISSING=0;
  if missing(DOB) then DOB_MISSING=1;
  else DOB_MISSING=0;
  FNAME=propcase(FNAME);
  LNAME=propcase(LNAME);
  if upcase(CITY) in: ('6IX','TOR') then CITY='Toronto';
  if CITY in ('?','Unknown') then CITY='';
  if CITY='NorthYork' then CITY='North York';
  if SEX='U' then SEX='';
  if upcase(MARITAL_STATUS) in: ('U') then MARITAL_STATUS='';
  if EDUCATION=1 then EDUCATION2='1 - Less than high school';
  else if EDUCATION=2 then EDUCATION2='2 - High school';
  else if EDUCATION=3 then EDUCATION2='3 - Undergraduate';
  else if EDUCATION=4 then EDUCATION2='4 - Post-graduate';
  else EDUCATION2='';
run;
```

Check Results



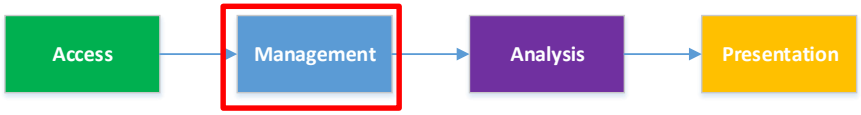
HCN_MISSING	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	186	100.00	186	100.00

DOB_MISSING	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	183	98.39	183	98.39
1	3	1.61	186	100.00

CITY	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Ajax	3	1.69	3	1.69
Aurora	2	1.13	5	2.82
Brampton	4	2.26	9	5.08
Fraserville	1	0.56	10	5.65
Georgetown	1	0.56	11	6.21
Guelph	3	1.69	14	7.91
Hamilton	6	3.39	20	11.30
Kirby	4	2.26	24	13.56
London	2	1.13	26	14.69
Markham	11	6.21	37	20.90
Milton	1	0.56	38	21.47
Mississauga	3	1.69	41	23.16
Newmarket	4	2.26	45	25.42
North York	15	8.47	60	33.90
Oakville	4	2.26	64	36.16
Oshawa	3	1.69	67	37.85
Peterborough	3	1.69	70	39.55
Pickering	4	2.26	74	41.81
Richmond Hill	8	4.52	82	46.33
Rigel VII	2	1.13	84	47.46
Scarborough	10	5.65	94	53.11
Toronto	72	40.68	166	93.79
Uxbridge	2	1.13	168	94.92
Vaughan	7	3.95	175	98.87
Whitby	2	1.13	177	100.00

Frequency Missing = 9

Remove Duplicates



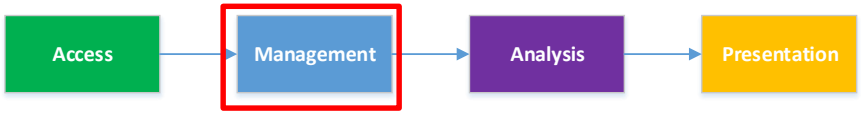
- Earlier we noticed that there were duplicate records
- We can remove duplicates using PROC SORT
- This procedure can also just sort data
- Good practice to create a new dataset when you remove duplicates so you still have the original data

```
proc sort data=COMBINE2 out=COMBINE3 nodupkey;
  by FAKE_HCN;
run;
```

Tells SAS to create a new dataset with the duplicates removed.

Nodupkey tells SAS to remove duplicates based on the specified BY variable (FAKE_HCN).

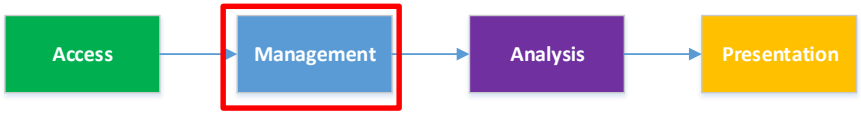
SAS Dates



- We need to calculate age to identify seniors for our model
- SAS dates are stored as the number of days (+/-) from January 1, 1960
- You will need to tell SAS that a numeric value is a date otherwise SAS will display the value as an integer
- There are many date formats, but a common one is date9. (e.g., 01JAN1960)

Date	SAS Value
December 31, 1959	-1
January 1, 1960	0
January 2, 1960	1

Calculating Age



- Simplest way is to use arithmetic in a DATA step
- There are more sophisticated functions that can be used to accurately account for leap years
- We can get today's date using TODAY() and the patient's DOB to calculate their age
- Additionally, let's apply an exclusion criteria where we only keep patients with a DOB

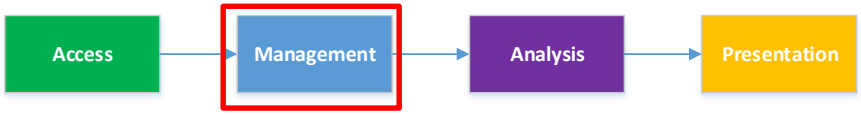
Apply date9. format to display current date as a date instead of numeric value.

```
data COMBINE4;
   set COMBINE3;
   format TODAY date9.;
   where DOB_MISSING=0;
   TODAY=today();
   AGE=int((TODAY-DOB)/365.25);
run;
```

WHERE statement tells SAS to only read data from COMBINE3 that are not missing DOB.

The INT() function tells SAS to keep only the integer.

Save Permanent Copy of Dataset



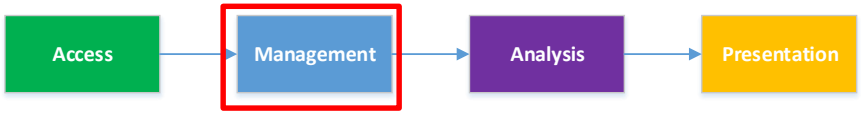
- Now that we've cleaned up the data, we can save it as a permanent dataset
- We need to use the library reference to tell SAS where to save the dataset

TUTORIAL is the library reference we used earlier.

```
data TUTORIAL.COMBINE4;
   set COMBINE4;
   drop HCN_MISSING DOB_MISSING;
run;
```

We can also drop variables in the new dataset created.

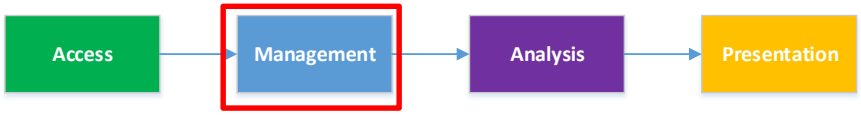
Merging Datasets



- There are a couple ways to merge/link datasets together
- One common method is to use a MERGE statement in a DATA step
- We will need to sort both datasets together in exactly the same way first

```
proc sort data=TUTORIAL.COMBINE4 out=COMBINE4;
   by FAKE_HCN;
run;

proc sort data=HEALTH;
   by FAKE_HCN;
run;
```

Merging Datasets *cont.*



FAKE_HCN	DOB
9999997760	19NOV1958
9999998743	15APR1932
9999998748	15APR1952
9999998749	25MAR1950
9999998750	15FEB1985

FAKE_HCN	DIABETES
9999997760	N
9999998743	Y
9999998748	Y
9999998749	N
9999998750	N

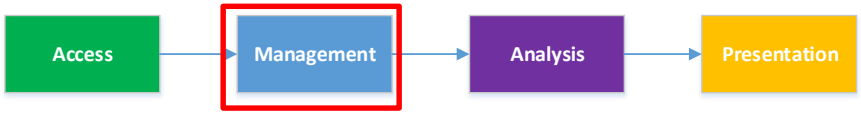
This tells SAS to merge COMBINE4 and HEALTH. We also tell SAS to reference COMBINE4 as 'a' and HEALTH as 'b'.

This tells SAS to merge on FAKE_HCN

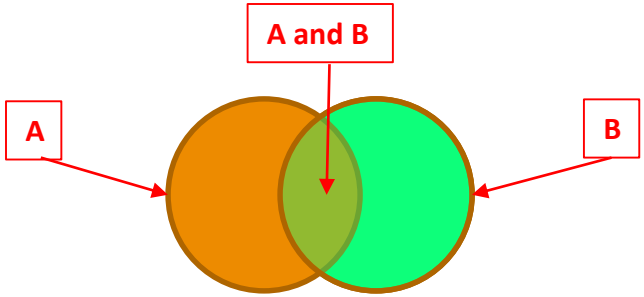
```
data COMBINE_HEALTH;
  merge COMBINE4 (in=a) HEALTH (in=b);
  by FAKE_HCN;
  if a=1 and b=1;
run;
```

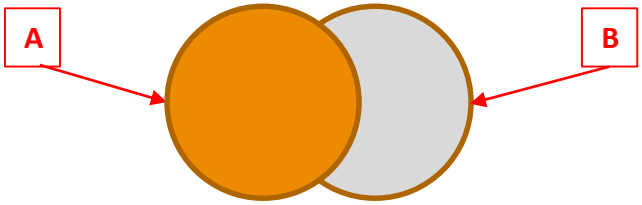
This tells SAS to merge observations only if the same FAKE_HCN is in both datasets.

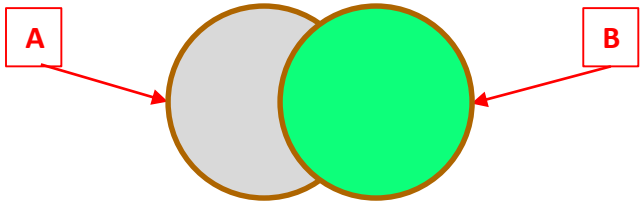
Merging Datasets *cont.*



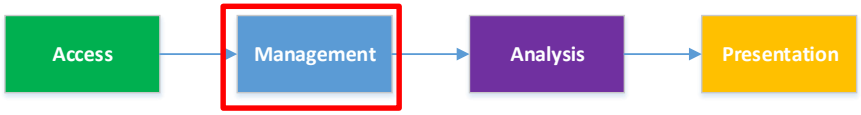

```
data inner;
   merge COMBINE4 (in=a) HEALTH (in=b);
   by FAKE_HCN;
   if a=1 and b=1;
run;

data left;
   merge COMBINE4 (in=a) HEALTH (in=b);
   by FAKE_HCN;
   if a=1;
run;

data right;
   merge COMBINE4 (in=a) HEALTH (in=b);
   by FAKE_HCN;
   if b=1;
run;
```







Explore, Clean, Standardize..Again

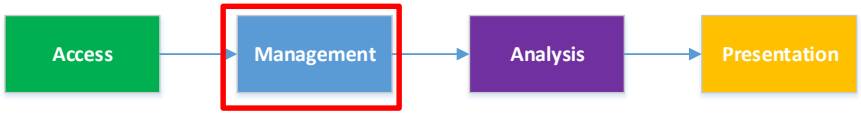


- Since we are working with new data and variables, we should check the values to see if cleaning/standardization is required
- We can use PROC FREQ again on categorical values
- To keep things simple, we previously converted unknown ('U') values to missing
- Let's do the same for this new data

DIABETES	Frequency	Percent	Cumulative Frequency	Cumulative Percent
N	87	58.00	87	58.00
U	2	1.33	89	59.33
Y	61	40.67	150	100.00

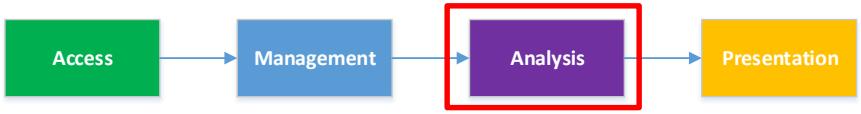
HYPER	Frequency	Percent	Cumulative Frequency	Cumulative Percent
N	92	61.33	92	61.33
U	5	3.33	97	64.67
Y	53	35.33	150	100.00

Explore, Clean, Standardize..Again *cont.*




```
data COMBINE_HEALTH2 (drop=DIET OBE DEP rename=(DIET2=DIET));
  length DIET2 $10.;
  set COMBINE_HEALTH;
  if DIABETES='U' then DIABETES='';
  if HYPER='U' then HYPER='';
  OBESITY=OBE;
  if OBESITY='U' then OBESITY='';
  if CANCER='U' then CANCER='';
  DEPRESSION=DEP;
  if DEPRESSION='U' then DEPRESSION='';
  if DIET=1 then DIET2='MIXED';
  else if DIET=2 then DIET2='VEGETARIAN';
  else if DIET=3 then DIET2='VEGAN';
  else DIET2='';
  if SUPPLEMENTS='U' then SUPPLEMENTS='';
  if DAIRY='U' then DAIRY='';
run;
```

Descriptive Statistics



- We can use PROC MEANS or PROC UNIVARIATE to look at continuous variables
- CLASS statement can be used to tell SAS to stratify the analysis by a variable (e.g., sex)

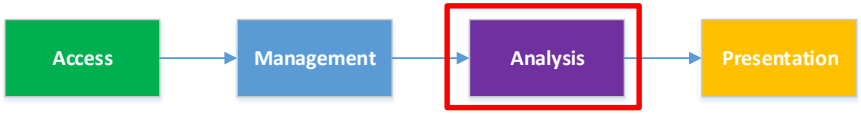
```
proc means data=COMBINE_HEALTH2 n mean median min max clm std p25 p50 p75
maxdec=2;
  var AGE EXERCISE TV;
  class DIABETES;
run;
```

Round output to two decimal places.

Specify which summary statistics to display.

```
proc univariate data=COMBINE_HEALTH2 plot;
  title "Analysis of Age, Exercise, & TV by Diabetes Status";
  var AGE EXERCISE TV;
  class DIABETES;
run;
```

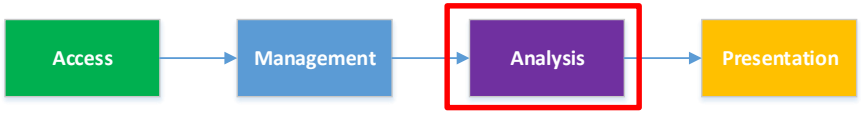
PROC MEANS Output



DIABETES	N Obs	Variable	N	Mean	Median	Minimum	Maximum
N	87	AGE	87	50.02	49.00	8.00	91.00
		EXERCISE	87	13.39	16.00	0.00	27.00
		TV	87	12.83	7.00	1.00	65.00
Y	61	AGE	61	64.89	68.00	18.00	88.00
		EXERCISE	61	7.44	5.00	0.00	25.00
		TV	61	27.90	32.00	1.00	60.00

DIABETES	N Obs	Variable	Lower 95% CL for Mean	Upper 95% CL for Mean	Std Dev	25th Pctl	50th Pctl
N	87	AGE	46.19	53.86	17.99	33.00	49.00
		EXERCISE	11.81	14.96	7.39	6.00	16.00
		TV	10.12	15.54	12.72	5.00	7.00
Y	61	AGE	60.85	68.92	15.74	65.00	68.00
		EXERCISE	5.71	9.17	6.75	2.00	5.00
		TV	23.38	32.42	17.65	9.50	32.00

PROC UNIVARIATE Output



For Hours of TV where DIABETES='Y'

Moments

N	61	Sum Weights	61
Mean	27.9016393	Sum Observations	1702
Std Deviation	17.6530969	Variance	311.631831
Skewness	-0.0186843	Kurtosis	-1.2722672
Uncorrected SS	66186.5	Corrected SS	18697.9098
Coeff Variation	63.2690312	Std Error Mean	2.26024745

Basic Statistical Measures

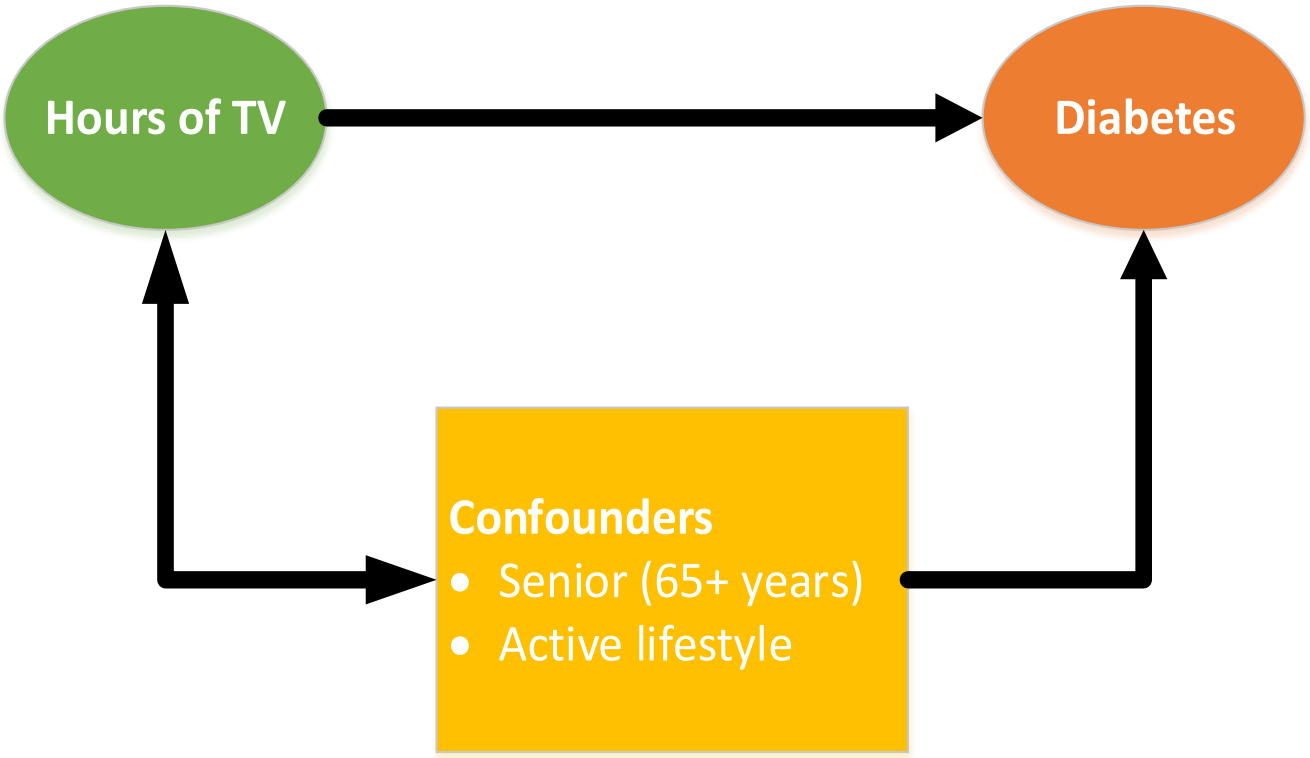
Location		Variability	
Mean	27.90164	Std Deviation	17.65310
Median	32.00000	Variance	311.63183
Mode	45.00000	Range	59.00000
		Interquartile Range	35.50000

Extreme Observations

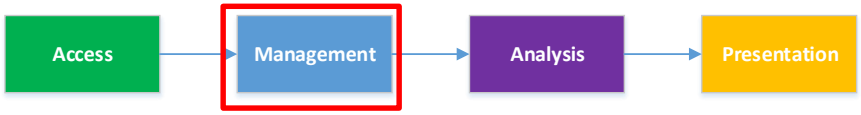
----Lowest----		----Highest---	
Value	Obs	Value	Obs
1	97	50	66
2	142	55	12
2	134	59	10
2	40	60	8
3	41	60	9

Returning to Research Question

- We have Hours of TV and Diabetes in our dataset
- Let's create Senior status (based on age) and Active lifestyle status variables



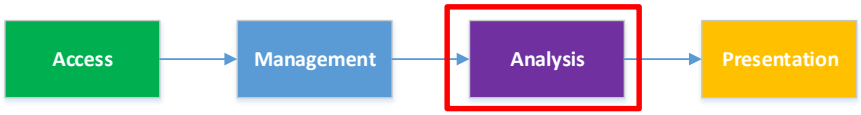
Create New Variables



- We can use a DATA step to create new variables for Senior and Active status
- Senior is defined as being 65+ years old
- Active status is defined as either doing weight lifting or cardio, and also exercising more than 15 hours a week
- COMBINE_HEALTH3 will be our “analytically ready dataset”

```
data COMBINE_HEALTH3;
   set COMBINE_HEALTH2;
   if AGE>=65 then SENIOR=1;
   else SENIOR=0;

   if (WEIGHT_LIFTING='Y' or CARDIO='Y') and EXERCISE>15 then ACTIVE=1;
   else ACTIVE=0;
run;
```

X^2 Test and Fischer's Exact Test



- Before we build our multivariate model, we might want to do a simple bivariate analysis first
- PROC FREQ can be used again to cross-tabulate categorical values and test expected vs. observed outcomes
- SAS will give you a warning if 25% of cells have expected counts less than 5 for a chi-square test

```
proc freq data=COMBINE_HEALTH3;
  tables DIABETES*SENIOR /chisq fisher norow nocol;
run;
```

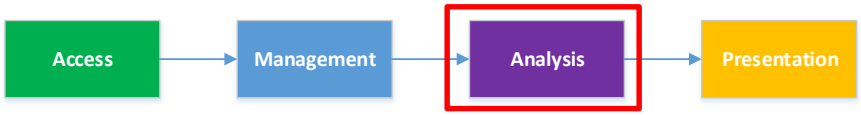
Table of DIABETES by SENIOR

DIABETES SENIOR

Frequency Percent	0	1	Total
N	55 37.16	32 21.62	87 58.78
Y	15 10.14	46 31.08	61 41.22
Total	70 47.30	78 52.70	148 100.00

Frequency Missing = 2

X^2 Test and Fischer's Exact Test *cont.*



Statistics for Table of DIABETES by SENIOR

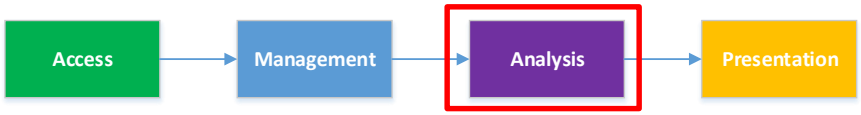
Statistic	DF	Value	Prob
Chi-Square	1	21.4648	<.0001
Likelihood Ratio Chi-Square	1	22.2346	<.0001
Continuity Adj. Chi-Square	1	19.9432	<.0001
Mantel-Haenszel Chi-Square	1	21.3198	<.0001
Phi Coefficient		0.3808	
Contingency Coefficient		0.3559	
Cramer's V		0.3808	

Fisher's Exact Test

Cell (1,1) Frequency (F)	55
Left-sided Pr <= F	1.0000
Right-sided Pr >= F	<.0001
Table Probability (P)	<.0001
Two-sided Pr <= P	<.0001

Effective Sample Size = 148
Frequency Missing = 2

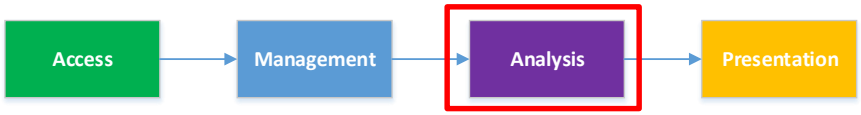
Multivariate Logistic Regression



- There are multiple procedures that can be used to build a regression model
- PROC LOGISTIC works in most situations
- By default, beta coefficient estimates will be produced but SAS can also output odds ratios

```
proc logistic data=COMBINE_HEALTH3;
   class  SENIOR (ref="0") ACTIVE (ref="0") /param=ref;
   model  DIABETES (event='Y') = TV SENIOR ACTIVE /clodds=wald cl;
   units  TV=1;
run;
```

Multivariate Logistic Regression *cont.*



CLASS statement tells SAS that SEX, SENIOR, and ACTIVE are categorical variables.

Ref tells SAS which value to use as the reference point.

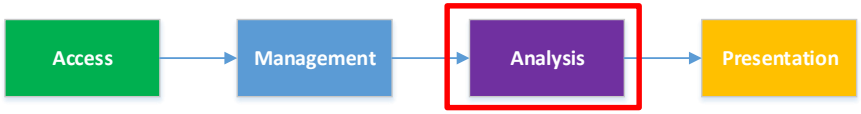
```
proc logistic data=COMBINE_HEALTH3;
  class  SENIOR (ref="0") ACTIVE (ref="0") /param=ref;
  model  DIABETES (event='Y') = TV SENIOR ACTIVE /clodds=wald cl;
  units  TV=1;
run;
```

MODEL statement tells SAS how to build the model with the outcome on the left of the equal sign followed by the covariates.

Event='Y' states which value in DIABETES is of interest.

CLODDS=wald cl tells SAS to output odds ratios and confidence limits using the Wald test.

Multivariate Logistic Regression *cont.*

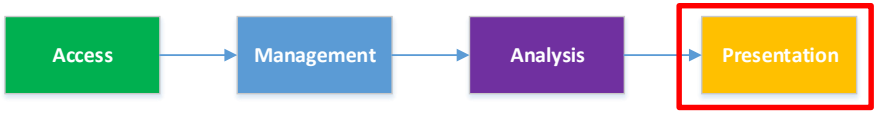


- Looks like a one hour increase in TV watching per week is associated with a 1.067 (95% CL: 1.036, 1.099) increase in odds of being diabetic, controlling for senior status and active lifestyle status
- The effect sizes and 95% CL for active lifestyle and especially senior status are very large
  - This should be investigated (could be issues with data or sample size)

Odds Ratio Estimates and Wald Confidence Intervals

Effect	Unit	Estimate	95% Confidence Limits	
TV	1.0000	1.067	1.036	1.099
SENIOR 1 vs 0	1.0000	8.003	2.989	21.430
ACTIVE 1 vs 0	1.0000	0.267	0.103	0.691

Exporting Output or Data

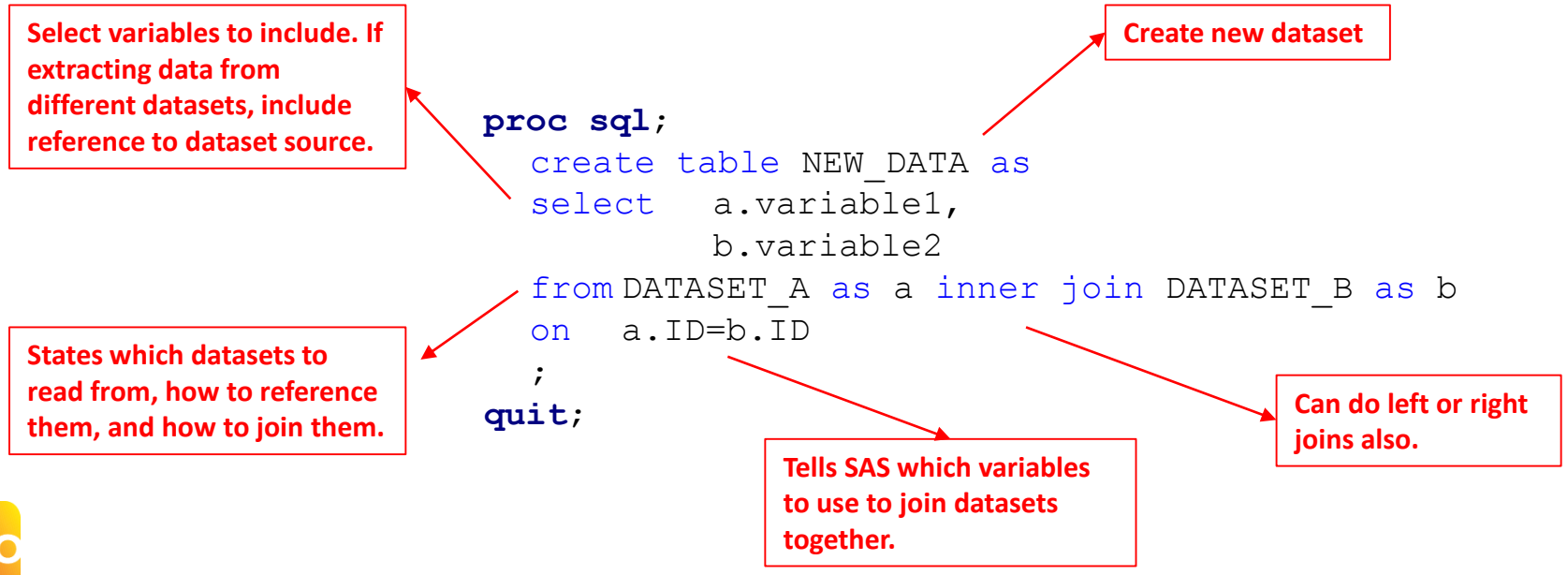


- SAS has a couple of ways to export data or analytical outputs
- PROC EXPORT can be used to export datasets in different file formats (e.g., CSV, XLSX)
- You can also run a procedure and have its outputs printed to a PDF

```
proc export data=COMBINE_HEALTH3
       outfile='M:\SAS Teaching\SAS Tutorial\Data\test export.csv'
       dbms=csv replace;
run;

ods pdf file="M:\SAS Teaching\SAS Tutorial\Data\Logistic Regression
Example.pdf";
proc logistic data=COMBINE_HEALTH3;
  class  SENIOR (ref="0") ACTIVE (ref="0") /param=ref;
  model  DIABETES (event='Y') = TV SENIOR ACTIVE /clodds=wald cl;
  units  TV=1;
run;
ods pdf close;
```

More on Merging Data

- Using a DATA step is an easy way to merge data, but one drawback is that data needs to be sorted in the same way first
- Works fine for small datasets but can be problematic with health administrative data (e.g., millions of observations)
- Additionally, sometimes we need to merge multiple datasets using multiple variables (or keys)
- PROC SQL is a good alternative to a DATA step



Another Hypothetical Scenario

- Suppose a program is interested in evaluating HPV screening tests for the cohort of patients we worked with earlier
- The researcher is only interested in subsequent screening tests (i.e., not the very first one), and the following criteria must apply:
  - 1) The HPV screening tests must be positive or there must have been a positive biopsy
  - 2) The HPV screening tests must be at least 30 days apart but not more than 2 years apart

	TEST_ID	FAKE_HCN	HPV_TEST_DATE	HPV_RESULT	BIOPSY
1	100251	9999999916	27JAN1999	0	0
2	100252	9999999916	18JUN2005	0	0
3	100253	9999999916	10SEP2006	1	0
4	100254	9999999916	25FEB2016	0	0
5	100255	9999999925	01JUN2018	0	0
6	100256	9999999937	19JAN2012	0	0
7	100257	9999999937	20NOV2014	0	0
8	100258	9999999938	07NOV1998	0	0
9	100259	9999999938	17DEC2002	1	0
10	100260	9999999938	12MAY2005	1	0

PROC SQL to Merge Data

- First we need to merge data from our analytically ready dataset to Tutorial HPV, which contains HPV screening data

```
proc sql;
   create table COMBINE_HPV as
   select   a.FAKE_HCN,
            b.TEST_ID,
            b.HPV_TEST_DATE,
            b.HPV_RESULT,
            b.BIOPSY as CERVICAL_BIOPSY,
            count(*) as TEST_COUNT,
            sum(HPV_RESULT) as POSITIVE_HPV,
            sum(CERVICAL_BIOPSY) as POSITIVE_BIOPSY
   from COMBINE_HEALTH3 as a inner join HPV as b
   on   a.FAKE_HCN=b.FAKE_HCN
   group by a.FAKE_HCN
   order by a.FAKE_HCN, b.HPV_TEST_DATE
   ;
quit;
```

PROC SQL can easily rename variables using the "as" statement followed by the new variable name.

Count(*) will count all observations based on the GROUP BY variable.

Sum(HPV_RESULT) will add the values in HPV_RESULT for each FAKE_HCN.

Group by tells SAS how to sum or count values.

Order by will sort the output.

PROC SQL to Merge Data *cont.*

- The record granularity is still at the HPV-test level
- TEST_COUNT, POSITIVE_HPV, and POSITIVE_BIOPSY are vertical summaries of data based on FAKE_HCN

Total number of records by patient.

Sum of the number of positive HPV tests by patient.

	FAKE_HCN	TEST_ID	HPV_TEST_DATE	HPV_RESULT	CERVICAL_BIOPSY	TEST_COUNT	POSITIVE_HPV	POSITIVE_BIOPSY
1	9999999916	100251	27JAN1999	0	0	4	1	0
2	9999999916	100252	18JUN2005	0	0	4	1	0
3	9999999916	100253	10SEP2006	1	0	4	1	0
4	9999999916	100254	25FEB2016	0	0	4	1	0
5	9999999925	100255	01JUN2018	0	0	1	0	0
6	9999999937	100256	19JAN2012	0	0	2	0	0
7	9999999937	100257	20NOV2014	0	0	2	0	0
8	9999999938	100258	07NOV1998	0	0	6	3	1
9	9999999938	100259	17DEC2002	1	0	6	3	1
10	9999999938	100260	12MAY2005	1	0	6	3	1

Identify Subsequent HPV Tests

- Next we need to identify HPV tests that are not the very first test a patient had
- There are multiple ways to do this, but we can use a BY statement in a DATA step
- This identifies the first and last observation for a patient
- You can also create a counter with this method that has other uses

If the first observation for a person is true, then TEST_NUM=0.

Tells SAS to add 1 to TEST_NUM.

Retain the value in TEST_NUM. When SAS reads the next observation, if it is the same person, SAS will add 1. If it is a new person, SAS will reset TEST_NUM to 0 and the process starts over again.

SAS keeps track of the first and last record for each patient.

```
data COMBINE_HPV2;
  set COMBINE_HPV;
  by FAKE_HCN;
  if first.FAKE_HCN then TEST_NUM=0;
  TEST_NUM=TEST_NUM+1;
  retain TEST_NUM;
run;
```

Identify Subsequent HPV Tests *cont.*

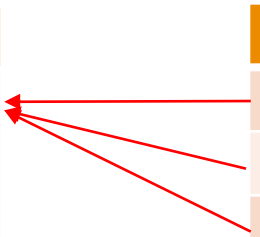
Counter for each observation by patient.

	FAKE_HCN	TEST_ID	HPV_TEST_DATE	HPV_RESULT	CERVICAL_BIOPSY	TEST_COUNT	POSITIVE_HPV	POSITIVE_BIOPSY	TEST_NUM
1	9999999916	100251	27JAN1999	0	0	4	1	0	1
2	9999999916	100252	18JUN2005	0	0	4	1	0	2
3	9999999916	100253	10SEP2006	1	0	4	1	0	3
4	9999999916	100254	25FEB2016	0	0	4	1	0	4
5	9999999925	100255	01JUN2018	0	0	1	0	0	1
6	9999999937	100256	19JAN2012	0	0	2	0	0	1
7	9999999937	100257	20NOV2014	0	0	2	0	0	2
8	9999999938	100258	07NOV1998	0	0	6	3	1	1
9	9999999938	100259	17DEC2002	1	0	6	3	1	2
10	9999999938	100260	12MAY2005	1	0	6	3	1	3
11	9999999938	100261	19AUG2006	1	1	6	3	1	4
12	9999999938	100262	30DEC2006	0	0	6	3	1	5
13	9999999938	100263	15MAY2016	0	0	6	3	1	6
14	9999999939	100264	11AUG2006	0	0	3	0	0	1
15	9999999939	100265	01SEP2015	0	0	3	0	0	2
16	9999999939	100266	16JUL2018	0	0	3	0	0	3

Identify Records of Interest

- We can use PROC SQL to identify records that are 30-730 days apart
- This can be accomplished by joining the dataset to itself

TEST_ID	HPV_TEST_DATE
100251	27JAN1999
100252	18JUN2005
100253	10SEP2006



TEST_ID	HPV_TEST_DATE
100251	27JAN1999
100252	18JUN2005
100253	10SEP2006

TEST_ID	HPV_TEST_DATE	TEST_ID2	HPV_TEST_DATE2	DATE_DIFF
100251	27JAN1999	100251	27JAN1999	0
100251	27JAN1999	100252	18JUN2005	2,334
100251	27JAN1999	100253	10SEP2006	6,238

Identify Records of Interest *cont.*

- Using PROC SQL, we can easily identify all combinations of HPV screening dates that are of interest

FAKE_HCN	TEST_ID	TEST_NUM	HPV_TEST_DATE	TEST_ID2	TEST_NUM2	HPV_RESULT2	CERVICAL_BIOPSY2	HPV_TEST_DATE2	DATE_DIFF
9999999943	100288	1	19APR2008	100288	1	0	0	19APR2008	0
9999999943	100288	1	19APR2008	100289	2	1	1	10SEP2009	509
9999999943	100288	1	19APR2008	100290	3	0	0	14FEB2014	2127
9999999943	100288	1	19APR2008	100291	4	1	0	13MAY2018	3676
9999999943	100289	2	10SEP2009	100288	1	0	0	19APR2008	-509
9999999943	100289	2	10SEP2009	100289	2	1	1	10SEP2009	0
9999999943	100289	2	10SEP2009	100290	3	0	0	14FEB2014	1618
9999999943	100289	2	10SEP2009	100291	4				
9999999943	100290	3	14FEB2014	100288	1				
9999999943	100290	3	14FEB2014	100289	2				
9999999943	100290	3	14FEB2014	100290	3				
9999999943	100290	3	14FEB2014	100291	4				
9999999943	100291	4	13MAY2018	100288	1				
9999999943	100291	4	13MAY2018	100289	2				
9999999943	100291	4	13MAY2018	100290	3				
9999999943	100291	4	13MAY2018	100291	4				

For patient 9999999943, test 100289 is of interest since there was one instance where it was:

1) **Not the first HPV test (TEST_NUM is 2)**
2) **HPV result was positive or cervical biopsy was positive**
3) **It was 509 days from a previous HPV test**

FAKE_HCN	TEST_ID	TEST_NUM	HPV_TEST_DATE	TEST_ID2	TEST_NUM2	HPV_RESULT2	CERVICAL_BIOPSY2	HPV_TEST_DATE2	DATE_DIFF
9999999943	100288	1	19APR2008	100289	2	1	1	10SEP2009	509

Identify Records of Interest *cont.*

- Using a WHERE statement we can keep only subsequent HPV tests that are either HPV positive or there was a positive biopsy, and are at least 30 days apart but not more than 2 years apart

```
proc sql;
   create table COMBINE_HPV3 as
   select b.*

   from   COMBINE_HPV2 as a inner join COMBINE_HPV2 as b
   on     a.FAKE_HCN=b.FAKE_HCN

   where  (b.TEST_NUM>1 and (b.HPV_RESULT=1 or b.CERVICAL_BIOPSY=1))
          and 30<=(b.HPV_TEST_DATE- a.HPV_TEST_DATE)<=730

order by a.FAKE_HCN, a.HPV_TEST_DATE, b.HPV_TEST_DATE
   ;
quit;
```

Transpose Data – Long to Wide

- Sometimes data needs to be transformed prior to analysis
- Transposing data is very common
- SAS provides a few different ways to convert data from long to wide and vice versa

	TEST_ID	FAKE_HCN	HPV_TEST_DATE	HPV_RESULT	BIOPSY
1	100251	9999999916	27JAN1999	0	0
2	100252	9999999916	18JUN2005	0	0
3	100253	9999999916	10SEP2006	1	0
4	100254	9999999916	25FEB2016	0	0
5	100255	9999999925	01JUN2018	0	0
6	100256	9999999937	19JAN2012	0	0
7	100257	9999999937	20NOV2014	0	0
8	100258	9999999938	07NOV1998	0	0

	FAKE_HCN	TEST_ID1	TEST_ID2	HPV_TEST_DATE1	HPV_TEST_DATE2	HPV_RESULT1	HPV_RESULT2	BIOPSY1	BIOPSY2	SUM_HPV	SUM_BIOPSY
1	9999999937	100256	100257	19JAN2012	20NOV2014	0	0	0	0	0	0

Transpose Data – Long to Wide *cont.*

- One way to transform data is to use a DATA step along with ARRAYS and a DO loop
- ARRAYs in SAS allow you to reference a group of variables
- This allows you to do an operation on a group of variables without having to write separate code for each variable
- Suppose we want to automate some code instead of typing out each line
- In the example below, creating MONEY_1 to MONEY_5 results in 5 lines of code we have to program

```
data example;
   MONEY=100000;

   MONEY_1=MONEY*1.035**1;
   MONEY_2=MONEY*1.035**2;
   MONEY_3=MONEY*1.035**3;
   MONEY_4=MONEY*1.035**4;
   MONEY_5=MONEY*1.035**5;
run;
```

	MONEY	MONEY_1	MONEY_2	MONEY_3	MONEY_4	MONEY_5
1	100000	103500	107122.5	110871.7875	114752.30006	118768.63056

Transpose Data – Long to Wide *cont.*

- We can replicate the SAS program previously in a more efficient manner with the code below
- First we create an array called MONEY_
  - [5] tells SAS that there are 5 variables in the array
  - MONEY_1-MONEY_5 specifies which variables are in the array
- Do a=1 to 5 creates a loop where SAS will cycle through the specified code changing a=1 to a=5 each iteration

```
data array_example;
   MONEY=100000;
   array  MONEY_[5] MONEY_1-MONEY_5;

   do a=1 to 5;
      MONEY_[a]=MONEY*1.035**a;
   end;
   drop a;
run;
```

Tell SAS to reference variables MONEY_1 to MONEY_5 as MONEY_.

MONEY_ is arbitrary here.

This tells SAS to loop through a=1 to 5.

"A" acts as a reference that changes from 1 to 5 as SAS goes through the DO loop.

Transpose Data – Long to Wide *cont.*

Iteration 1 where a=1

```
data array_example;
   MONEY=100000;
   array  MONEY_[5] MONEY_1-MONEY_5;

   do a=1 to 5;
     MONEY_1=MONEY*1.035**1;
   end;
   drop a;
run;
```

Iteration 2 where a=2

```
data array_example;
   MONEY=100000;
   array  MONEY_[5] MONEY_1-MONEY_5;

   do a=1 to 5;
     MONEY_2=MONEY*1.035**2;
   end;
   drop a;
run;
```

Iteration 3 where a=3

```
data array_example;
   MONEY=100000;
   array  MONEY_[5] MONEY_1-MONEY_5;

   do a=1 to 5;
     MONEY_3=MONEY*1.035**3;
   end;
   drop a;
run;
```

Etc. until iteration 5 where a=5

Transpose Data – Long to Wide

- We can use arrays and a do loop to transpose data from long to wide format
- There are 21 variables because that is the maximum number of tests a patient had

```
data HPV_WIDE (drop=a TEST_ID HPV_TEST_DATE HPV_RESULT BIOPSY);
  retain FAKE_HCN;
  array  TEST_ID_[21];
  array  HPV_TEST_DATE_[21];
  array  HPV_RESULT_[21];
  array  BIOPSY_[21];
  format HPV_TEST_DATE_1-HPV_TEST_DATE_21 date9.;

  do a=1 to 21 until (last.FAKE_HCN);
    set HPV;
    by FAKE_HCN;
    TEST_ID_[a]=TEST_ID;
    HPV_TEST_DATE_[a]=HPV_TEST_DATE;
    HPV_RESULT_[a]=HPV_RESULT;
    BIOPSY_[a]=BIOPSY;
  end;
  POSITIVE_HPV=sum(of HPV_RESULT_1-HPV_RESULT_21);
  POSITIVE_BIOPSY=sum(of BIOPSY_1-BIOPSY_21);
run;
```

This tells SAS to sum the values of HPV_RESULT_1 to HPV_RESULT_21 to give the total number of positive HPV results for each patient.

Transpose Data – Wide to Long

- We can use a DATA step and arrays to reverse the process also

```
data HPV_LONG;
  retain TEST_ID FAKE_HCN HPV_TEST_DATE HPV_RESULT BIOPSY;
  set HPV_WIDE;
  format HPV_TEST_DATE date9.;

  array _TEST_ID [21] TEST_ID_1-TEST_ID_21;
  array _HPV_TEST_DATE [21] HPV_TEST_DATE_1-HPV_TEST_DATE_21;
  array _HPV_RESULT [21] HPV_RESULT_1-HPV_RESULT_21;
  array _BIOPSY [21] BIOPSY_1-BIOPSY_21;

  do a=1 to 21;
    if _TEST_ID[a]^=. then do;
      TEST_ID=_TEST_ID[a];
      HPV_TEST_DATE=_HPV_TEST_DATE[a];
      HPV_RESULT=_HPV_RESULT[a];
      BIOPSY=_BIOPSY[a];
      output;
    end;
  end;
  keep TEST_ID FAKE_HCN HPV_TEST_DATE HPV_RESULT BIOPSY;
run;
```

The OUTPUT statement tells SAS to write a new observation with each iteration of the do loop. This is the key to transforming the data back to long format.

Questions?

Image source: https://157ofgemma.com/