Graduate School Class Reminders

- Maintain six feet of distancing
- Please sit in the same chair each class time
- Observe entry/exit doors as marked
- Use hand sanitizer when you enter/exit the classroom
- Use a disinfectant wipe/spray to wipe down your learning space before and after class
- Media Services: 414 955-4357 option 2

Documentation on the web

- CRAN: `http://cran.r-project.org`
- R manuals: `https://cran.r-project.org/manuals.html`
- SAS: `http://support.sas.com/documentation`
- SAS 9.3: `https://support.sas.com/en/documentation/documentation-for-SAS-93-and-earlier.html`
- Step-by-Step Programming with Base SAS 9.4 (SbS): `https://documentation.sas.com/api/docsets/basess/9.4/content/basess.pdf`
- SAS 9.4 Programmer s Guide: Essentials (PGE): `https://documentation.sas.com/api/docsets/lepg/9.4/content/lepg.pdf`
- Wiki: `https://wiki.biostat.mcw.edu` (MCW/VPN)

ISO 8601 and the Proleptic Gregorian Calendar Again

- Creating a SAS format for ISO 8601 dates seemed like a logical approach
- However, this created a rather large formats catalog with 2.3M records and containing 169MB
- SAS is fairly good at handling large data sets, but this seems very inefficient
- So, we are going to create new SAS functions instead
- Typically, SAS macros have filled this niche
- However, SAS macros mainly generate SAS code to be run Creating text and performing integer arithmetic which is fast
- Recently (circa 2004) SAS added a function compiler with PROC FCMP so we can write our own DATASTEP functions, but they are relatively slow
- Now, we have some options to consider
- N.B. you can create functions in C/C++ and link to them with `PROC PROTO` (circa 2004): see `protolibs.sas` (but `PROC PROTO` is restricted in v9.4 unlike 9.0 to 9.3)

SAS macros

- You can create macro variables by `%let VAR=VALUE;`
- Access their values by `&VAR` which generates `VALUE`
  quotes are not typically used so if you need quotes
  double quotes allow expansion: `"&VAR"` yields `"VALUE"`
  but single quotes don't: `'&VAR'` yields `'&VAR'` which can be useful if you need to generate a literal with an ampersand
- You can create array-like macro variables like so
  `%let VAR&i=VALUEi;`
  `&&VAR&i` yields `VALUEi`
  when the macro variable `&i` is the number `i`
- Macro variable are often lists
  `%let var=dead male black other hispanic sbp;`
  and you access them all at once by `&var` or individually by
  `%let var&i=%scan(&var, &i, %str( ));`
  with `&&var&i`

SAS macros: %if-%then-%else

RED and BLUE lines optional

```
%if IF-CONDITIONAL-EXPRESSION %then IF-THEN-BLOCK
%else %if ELIF-CONDITIONAL-EXPRESSION-1
%then ELSE-IF-BLOCK-1
⋮
%else %if ELIF-CONDITIONAL-EXPRESSION-M
%then ELSE-IF-BLOCK-M
%else ELSE-BLOCK
```

- BLOCK can either be a single line followed by a semi-colon
- or it can be several lines surrounded by %do; and %end;

```
%do;
    LINE1;
    ⋮
    LINEn;
%end;
```

SAS macros: macro expressions

- Simpler expressions than DATASTEP expressions,
  but follow somewhat similar rules so it is relatable
- Numeric: only integer expressions by default
  the `%eval()` function performs integer arithmetic
  `%eval(&i+1)` produces 11 if `&i` is 10
  instead of `&i+1` which produces 10+1
  `%if %eval(&i+1)=11 %then ...;` is TRUE/executed
- `%sysevalf()` for floating point expressions
  `%if %sysevalf(2.5=(age/10), boolean) %then ...;`
- Character: often helpful to place these in quotes
  `%if "&&var&i"="dead" %then ...;`

SAS macros: %DO loops

- The %do statement has other variants besides a block
- `%do VAR=VALUE1 %to VALUE2 %by VALUE3; ...; %end;`
  if `VALUE3` not given, it defaults to 1

```
%let var=dead male black other hispanic sbp;
%do i=1 %to %_count(&var);
%let var&i=%scan(&var, &i, %str( )); %end;
```

- `%do %until(CONDITION); ...; %end;`
  executes until the `CONDITION` is true
- `%do %while(CONDITION); ...; %end;`
  executes until the `CONDITION` is false

SAS macros

- You can create macros as follows
  `%macro NAME; ...; %mend NAME;`
  and call it by `%NAME;`
- These macros can have arguments as well
  `%macro NAME(ARG1, ..., ARGn); ...; %mend NAME;`
  and call it by `%NAME(VALUE1, ..., VALUEn);`
  within the macro, the argument values are accessed as `&ARGi`
- These arguments can have defaults which is very convenient

```
%macro NAME(ARG1, ..., ARGn,
     ARGn+1=VALUE1, ..., ARGn+m=VALUEm);
     ...;
%mend NAME;
```

and call it by `%NAME(VALUE1, ..., VALUEn,`
`ARGn+i=VALUEi, ARGn+j=VALUEj, ...);`

SAS macros

- Our SAS installation can be found in the directory `/usr/local/sas/SAS18w47/SASHome/SASFoundation/9.4`
- And the SAS macros can be found in the `sasautos` sub-directory
- However, there are big gaps in their capabilities
- I have created a supplementary library called RASmacro (my middle name is Allen) that is GPL free software
- Most of them are on `github` (or can be soon I haven't uploaded recently): `https://github.com/rsparapa/rasmacro`
- Most of them start with an underscore so that they are distinct from what SAS provides
- On gouda they are installed in `/usr/local/sasmacro` and the documentation is provided within the files themselves it cannot get separated from the SAS program that way

RASmacro

- We have seen some of these already
- `%_nobs` returns the number of observations in a data set
- `%_list` creates a SAS list with slightly more general rules than `VAR1-VARn`
- Let's take a look `%_julian` and `%_gregory`
- These provide a mathematical formula to calculate ISO 8601 dates with the SAS origin of 01/01/1960

PROC FCMP: the function compiler

- A SAS macro is impractical for ISO 8601 dates since the arguments would have to be integer literals
- We would like to have the flexibility for the arguments to be DATASTEP variables
- See my example of doing just that by creating the function `_julian` with the SAS program `_julian.sas` based on the `%_julian` RASmacro

HW 1: create the DATASTEP function `_gregory`

- Hint: base this on the RASmacro `%_gregory`
- Verify that this function is creating ISO 8601 compatible dates by comparing it with the SAS format that you created for ISO 8601
- BEWARE: user-defined functions are comparatively slow which is magnified by the size of the data set like we have here with 2.3M observations

HW 2: create the DATASTEP subroutine: `call _permute`

- This subroutine will generate random permutations since such capability does not appear to exist within SAS currently
- It just needs to take 1 argument: the max value `M`
  i.e., create random permutations of `1, ..., M`
- Hint: base this on your stratified random sampling code
- Test to make sure that the seed value works the way it should
- It is easier if we consider `M` has some maximum value like 100
- You can use `M=5` for testing
- `subroutine _permute(_w[*]) varargs; outargs _w;`
- For array initialization of the uniform random variables, use the `_repeat` macro

```
%let strata=100;
array _u(&strata) u1-u&strata (%_repeat(%str( 1), &strata))
```