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)

Hands-on HW: reading data from multiple files of the PTB

- As we have seen, the ECG information requires reading in multiple files
- A header file, `.hea`, a data file for the 12 standard leads, `.dat`, and a data file for the 3 Frank leads, `.xyz`
- Let's re-visit: why should we use the `IBw.d` informat?
- Create your own SAS library in the directory `~/libname/ecg` and read in all of the ECGs for controls into the permanent SAS data set `ecg.controls`
- And utilize your two's complement correction
  how many ECGs are there for which the sum total is incorrect?
- see the example for the cases: `PTB/cases.sas`
- BEWARE: the cases data set is quite large with almost 60M observations yet still surprisingly fast: run-time is about 5 minutes!
- The controls data set will be smaller, but if you have a mistake in your SAS code, then the error messages can create an ENORMOUS `.log` file

SAS and Emacs/ESS[SAS]

- Let's take a look at `autoexec.sas`
  `cp ~rsparapa/autoexec.sas ~`
- N.B. similarly, you have to copy `.sas` files to your home directory to edit/submit them (just like `.R` files)
- Submit `autoexec` with `F3`, goto the `.log` with `F5` return with `F4` and goto the `*shell*` buffer with `F8`
- Test of ESS[SAS] function keys
- see `PTB/ecg.sas` and `PTB/acf.sas`
- `F9` opens a SAS data set for viewing
- `F12` opens a graphics file for viewing
- `C-F10` makes the font smaller
- `C-F11` makes the font larger
- `TAB` and `C-TAB`
- Emacs and `tabbar`

DATASTEP STATEMENTS

- `data NEW; set OLD; STATEMENT; ... run;`
  the impact of `STATEMENT` is on the creation of `NEW`
  `OLD` is unchanged
- `drop VAR1 ... VARn;`
  drop these variables from the data set
  if the variable names actually end in numbers, then you can use a *variable list* which can be used in many places
  `drop VAR1-VARn;`
- `keep VAR1 VAR2 ... VARn;`
  keep ONLY these variables in the data set
- `rename OLD1=NEW1 ... OLDn=NEWn;`
  rename these variables in the data set
  if you can use a variable list
  `rename OLD1-OLDn=NEW1-NEWn;`
- `where CONDITION;`
  limit to observations from `OLD` satisfying `CONDITION`
  many PROCS also accept a `where` statement
  `proc NAME data=OLD; where CONDITION; ... run;`

SAS NUMBERS, CHARACTERS and CONDITIONS

- SAS makes no distinction for integers
  every number is double-precision floating point
  which obviously can represent integers as well
  SAS has no concept of `NaN` or `Inf`
  Division by zero results in a missing value
- if you attempt to perform arithmetic with a character expression, SAS will attempt to convert it to number
- if a CONDITION is simply a number or a numeric variable, then it is TRUE UNLESS the number is 0 or missing!
  `x=.;` is SAS' missing value for numeric
  Also `.a ... .z ._`
  `y=" ";` is SAS' missing value for character
- a CONDITION resulting in a character expression (that can't be converted into a numeric expression) is an error
- So these are all TRUE: `1, "1", 2.1`
- And these are all FALSE: `0, "0", ., ".", " "`

SAS NUMBERS, CHARACTERS and CONDITIONS

- Typically, a CONDITION is a comparison between two variables (or between a variable and a literal)
  = or EQ; < or LT; <= or LE; > or GT; >= or GE; and ^= or NE
- CONDITIONS can be chained together
  by either & and for AND; either | or for OR
- you can also chain together interval CONDITIONS like so
  3<x<=6 for x in the interval (3, 6]
- you can specify a subset by `x in(3, 4, 5)`
- you can compare a variable to missing (unlike R): `x=.`
  also `n(ARG1, ..., ARGn)` and `nmiss(ARG1, ..., ARGn)`
- BEWARE: .<x is TRUE unless x is also missing
  In comparisons, missing behaves like negative infinity!
- you can specify STARTS-WITH by the colon operator
  `y=:"a"` which is TRUE for all character strings starting with a
- alphabetic comparisons are based on the order of ASCII
  "A"<"Z" is TRUE and " "<"A" is also TRUE
  ASCII collating sequence on SbS pg. 204

DATASTEP STATEMENTS: `length`

- The `length` statement can be used for both numeric and character variables
- BEWARE: it is risky to reduce the length of numeric variables
  do NOT lessen the length below the default which is 8
  can create errors which are VERY difficult to debug since no error message is generated
  to set, you would do: `length NUMBER 8;`
- The default length for character variables is also 8
- Typically, longer character variables are either 40 or 200 characters
  `length SHORT $ 40 LONG $ 200;`
- if you need to change the length of a variable already created then place the `length` statement BEFORE the `set` statement
  `data NEW; length VAR $ 200; set OLD; ...`

DATASTEP STATEMENTS: `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;
```

DATASTEP STATEMENTS: `output`

- You can selectively remove or save observations
- The default is to save each observation when the program reaches the `run;` statement
  or you can force that behavior with the `output` statement
  these are the same

```
data NEW; set OLD; run;
data NEW; set OLD; output; run;
```

- You can selectively pick the saved observations

```
data NEW; set OLD; if CONDITION then output; run;
```

- You can selectively save observations to different data sets

```
data NEW1 NEW2; set OLD;
if CONDITION1 then output NEW1;
else if CONDITION2 then output NEW2;
run;
```

DATASTEP STATEMENTS: `delete` and subsetting `if`

- You can selectively remove observations
  if not removed, then they are saved
  `data NEW; set OLD; if CONDITION1 then delete;`
  `run;`
- Similarly, you can use an `if` statement
  (don't confuse with an `if-then-else`)
- This is the same as above
  `data NEW; set OLD; if CONDITION2; run;`
  where `CONDITION2` is equivalent to `not(CONDITION1)`
  the `not` function reverses the predicate
- Typically, a `where` statement is preferable to an `if` for
  computational efficiency especially with large data sets
  a `where` clause omits observations from even being considered
  while `if` has to selectively process each one
  `data NEW; set OLD; where CONDITION2; run;`

SAS data set options

- Operate on a data set: `NAME(OPTION)`
  and these can be combined: `NAME(OPTION1 ... OPTIONn)`
- Generally, can be used anywhere a data set `NAME` can appear
  i.e., in PROCS and the DATASTEP (NOT just the DATASTEP like most DATASTEP statements)
- `NAME(drop=VAR1 ... VARn)`
  drop these variables from the data set
  if you can use a *variable list*: `NAME(drop=VAR1-VARn)`
- `NAME(keep=VAR1 VAR2 ... VARn)`
  keep ONLY these variables in the data set
- `NAME(rename=(OLD1=NEW1 ... OLDn=NEWn))`
  rename these variables in the data set
  or if you can use variable lists
  `NAME(rename=(OLD1-OLDn=NEW1-NEWn))`

SAS data set options

- `NAME(firstobs=N)`
  start with observation number `N` by skipping 1 to `N-1`
- `NAME(obs=N)`
  limit to observations number 1 to `N`
  `NAME(firstobs=M obs=N)`
  OR limit to observations number `M` to `N`
- `NAME(where=(CONDITION))`
  limit to observations satisfying `CONDITION`
  very useful for those PROCS that don't accept a `where` statement
- `INDEX=(KEY=(VAR1 ... VARn))`
  create a new `KEY` to retrieve observations in `KEY` order
  `INDEX=(KEY=(VAR1 ... VARn)/unique)`
  use the `unique` keyword if it is a unique key
  What is a unique key?

Creating output with `proc print`

- ```
  proc print data=NAME;
  var VAR1 ... VARn;
  run;
  ```

```
proc print UNIFORM N data=ntdb.elder;
    where sbp=0 | pulse=0;
    var new_doa doa dead sbp pulse;
run;
```
```