Statistics 506 Midterm

Name, uniquename:

October 23, 2018

Instructions

Answer each of the questions that follow in the space provided. If you need additional space, you may use a piece of plain white paper with the problem number and your name clearly labelled at the top.

You may bring as a reference a single, double sided, 8.5 by 11 inch page of notes. Please write your name and unique name on this sheet and submit it with your exam.

Your desk should be clear of all other materials except pens or pencils. No computer or cell phone use is permitted during the exam.

The exam has six questions organized into two parts. The four questions in part one are worth 20 points each. The two questions in part two are worth 35 points each. You should answer all six questions. There are 150 total points available.

You have 80 minutes to complete the exam. Try not to spend more than 10 minutes on any single question before beginning the others.

When finished, please sign the exam in the space below and then bring it to the front of the room.

Signature:

_____

Part One

Question 1

R basics. For each snippet of R code below, give the value or values held in `out` in the global environment after the code is run along with its *class* as returned by `class(out)` You should assume each code chunk starts in an empty global environment and that only base packages are loaded. [20 pts total; 4 pts each]

```
# i
out = seq(1L, 24L, 2L)
dim(out) = c(4, 3)
(out = matrix(1, nrow = 1, ncol = 4) %*%  out); class(out)
```

```
##      [,1] [,2] [,3]
## [1,]   16   48   80
```

```
## [1] "matrix"
```

Key concepts: Column major order, matrix operations, class inheritence.

```
# ii
(out = paste0(0:7 %/% 4, 0:7 %% 4)); class(out)
```

```
## [1] "00" "01" "02" "03" "10" "11" "12" "13"
```

```
## [1] "character"
```

Key concepts: Arithmetic operations. Many of you did not know the function "paste0". Although "paste" was used in an example, I am giving everyone +2 points on this question.

```
# iii
out = 10
f = function(x){
  out = 0
  for(i in 1:x){
   out = out + i
  }
  out
}
new = f(out)
out; class(out)
```

```
## [1] 10
```

```
## [1] "numeric"
```

Key conecpt: Variable scope

```r
# iv
is_even = function(x) {x %% 2} == 0
out = 1
while ( out < 5 ){
 if( is_even(out) ){
   out = 2*out
 } else {
   out = 3*out
 }
}
out; class(out)
```

```
## [1] 9
```

```
## [1] "numeric"
```

Key conecpt: While loops and function naming. If you did not know `%%` you should still be able to guess at what `is_even` does. You may even have chosen to use the `is_even` definition to help you with part ii.

```r
# v
start = list(x = TRUE, y = 'two', z = floor(pi) )
start$xyz = with(start, c(x, y, z) )
out = lapply(start, class)
out; class(out)
```

```
## $x
## [1] "logical"
##
## $y
## [1] "character"
##
## $z
## [1] "numeric"
##
## $xyz
## [1] "character"
```

```
## [1] "list"
```

Key concept: You should be familiar with `lapply` and `with`, but it appears many were not so I am adding +3 (more) points to your scores for this question.

Question 2

Linux shell skills. [20 points]

a. Match each of the command line tools on the left to the best description of how it operates on the file *file* on the right. Indicate your answers by writing your selection in the space ____ provided. [2 points each]

1. _iv_ head -n1 *file*
2. _iii_ tail -n1 *file*
3. _ii_ < *file* cut -f1 -d","
4. _vi_ < *file* grep -e
5. _v_ < *file* wc -l
6. _i_ gzip *file*

i. Creates a compressed version of *file*.

ii. Extracts the first column of *file*.

iii. Extracts the last row of *file*.

iv. Extracts the first row of *file*.

v. Counts the number of lines in *file*.

vi. Can be used to find lines matching an expression in *file*.

b. **Background.** A popular format for working with data too large to store in memory is the *bucketized*, *columnar* format. A *columnar* data storage format is one in which each column or variable of a rectangular data array is stored in its own file on disk. A *bucketized* or *chunked* data storage format is one in which the rows are split into *buckets* (aka chunks or groups) with rows from the same bucket being stored in different files on disk.

Task. Complete the *bash* script below by inserting appropriate commands into the places indicated by `<a>`, `<b>`, `<c>`, and `<d>`. Write your responses in the space below the script to write each command. Each of the answers should use a command from part "a", perhaps with different parameters. You do not need to understand the script below in its entirety; instead use comments and context to determine what command is needed in each of the indicated places.

To help you understand the problem, here are the first two lines of 'iris.csv.gz':

```
$ gunzip -c iris.csv.gz | head -n2
Sepal.Length,Sepal.Width,Petal.Length,Petal.Width,Species
5.1,3.5,1.4,0.2,setosa
```

`<a>: tail -n+2`

`<b>: cut -f5 -d","`

Note (a) and (b) could be given in either order.

`<c>: grep -e$spec | cut -f$k -d","`

The comment doesn't specifically address the species subsetting and the instructions were unclear regarding the use of multiple commands. Either side of the pipe is sufficient for full credit.

`<d>: gzip`

Notes: **`head`** *does not work in <a> or <b>, but +1 for thinking in the right direction. These are worth two points each: 1 point for the command and 1 point for its parameters. Please compare each of the lines these appear in to the solutions for the first quesiton on the first problem set.*

```
#!/bin/bash

## This script converts the data stored in iris.csv.gz
## to a bucketized, columnar format.
##
##  Buckets are determined by the value of the "species" variable
##  and columns are represented as "col".csv.gz
##
## Data is stored in the following file hierarchy:
##   iris/species/col.csv.gz
##
## Where `iris/species/col.csv.gz` contains all data for
## the column "col" from the subset of rows matching
## species "species" in a gzip compressed format.

wd0=$(pwd)
file=iris.csv.gz

## Get the unique values of species ignoring the header
species=$(gunzip -c $file | <a> | <b> | sort | uniq)

## Get the values of the column headers
cols=$(gunzip -c $file | head -n1 | cut -f'1-4' -d,)

## Loop over species
for spec in $species
do
 mkdir $wd0/$spec
 ## Loop over column names "col" and keep a counter "k"
 k=0
 for col in $(echo $variable | sed "s/,/ /g")
  do
    k=$(($k + 1))
    # Extract column "col" (using k), redirect, and then compress.
    colfile=$wd0/$spec/$col.csv
    gunzip -c $file | <c> > $colfile && <d> $colfile
  done
done
```

Question 3

S3 methods in R. [20 points]

Consider the R code below.

```
library(dplyr)
cars = mtcars %>% mutate(cyl = as.factor(cyl), wt = wt - mean(wt,  na.rm=TRUE))
fit = lm(mpg ~ cyl*wt, data = cars)
class(fit)
```

```
## [1] "lm"
```

```
print(fit)
```

```
##
## Call:
## lm(formula = mpg ~ cyl * wt, data = cars)
##
## Coefficients:
## (Intercept)         cyl6         cyl8           wt      cyl6:wt
##      21.403       -1.939       -4.589       -5.647        2.867
##     cyl8:wt
##       3.455
```

a. After the function call `print(fit)`, in what order are the following functions called? Indicate your solution as a call graph with arrows indicating the order, i.e. *first* → *second* → *third*. [3 pts]

`cat()`, `print.lm()`, `UseMethod()`

`UseMethod()` → `print.lm()` → `cat()`

Key concept: S3 method dispatch. You should know that `print()` is a generic function which calls `UseMethod()` to find the appropriate method in this case, `print.lm()`. You get 2 points for having `UseMethod()` before `print.lm()` in the sequence you gave. You do not need to know that `cat()` is called by `print.lm()` as the other two *must* come first.

b. Define a new S3 generic function `describe()`. [2 pts]

```
describe = function(x, ...){
  UseMethod('describe')
}
```

c. Define a default describe method that consists solely of a call to `str()` on the object passed. [2 pts]

```
describe.default = function(x){
  str(x)
}
```

d. Define an S3 method of the describe generic for class `lm` that sereves as an alias for the appropriate `summary` method. Your method function should directly call a specifc summary method and not the summary generic. [5 pts]

```
describe.lm = function(x, ...){
  stopifnot('lm' %in% class(x))
  summary.lm(x, ...)
}
```

e. Define an S3 generic `center` and methods for class `numeric` and `data.frame` that create centered versions of numeric variables. The data.frame method should only center columns of class `numeric`, `integer`, or `double` and leave other columns as is. [8 pts]

```
## Generic
center = function(x, ...){
  UseMethod('center')
}

## Method for class numeric
center.numeric(x, na.rm=TRUE){
  stopifnot(is.numeric(x))
  x - mean(x, na.rm=na.rm)
}

## Method for class data.frame
center.data.frame(x, na.rm=TRUE){
 stopifnot(is.data.frame(x))

 # Get columns with eligible class
 ind = sapply(x,
  function(x) any(class(x) %in% c('numeric', 'integer', 'double')) )

 for(i in ind){
    x[,i] = center.numeric(x, na.rm=na.rm)
 }

 x
}
```

Question 4

Regression formulas. [20 points]

Consider the data frame `mtcars` whose first few rows are given sbelow.

```
mtcars %>% as_tibble() %>% mutate(cyl = factor(cyl)) %>% head()
```

```
## # A tibble: 6 x 11
##     mpg    cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <fctr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21.0      6   160   110  3.90 2.620 16.46     0     1     4     4
## 2  21.0      6   160   110  3.90 2.875 17.02     0     1     4     4
## 3  22.8      4   108    93  3.85 2.320 18.61     1     1     4     1
## 4  21.4      6   258   110  3.08 3.215 19.44     1     0     3     1
## 5  18.7      8   360   175  3.15 3.440 17.02     0     0     3     2
## 6  18.1      6   225   105  2.76 3.460 20.22     1     0     3     1
```

```
with(mtcars, cyl)
```

```
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```

a. For each of the R formulas on the left, find its closest equivalent in Stata on the right. If there is no good match write an "X" in the space. Write your answer on the space provided. [2.5 pts each, 10 pts]

1. _iv_ 'mpg wt + cyl + carb'
2. _i/iii_ 'mpg wt + cyl + carb + wt:carb'
3. _X_ 'mpg wt*cyl + carb'
4. _X_ 'mpg wt*cyl*carb'

Assume the following follow 'regress'.

```
i. mpg wt i.cyl carb c.wt#c.carb

ii. mpg c.wt##i.cyl##i.carb

iii. mpg c.wt##c.carb i.cyl

iv. mpg wt i.cyl carb
```

Notes: Formula ii is nearly correct for 4, save for the i.carb (carb is numeric). You only lost .5 a point for ii instead of X here. Either i or iii is correct for 2.

b. Write R code to fit a linear model to the `mtcars` data using each of the (R) formulas 1-4 above, select the model with the lowest BIC, and then assign the fitted model object to `fit_best`. [10 pts]

Solution: My intent here was that you use functions or iteration to avoid excessive writing. However any working solution is acceptable and below is just one possibility.

```
formulas = list(mpg ~ wt + cyl + carb,
                mpg ~ wt + cyl + carb + wt:carb,
                mpg ~ wt*cyl + carb,
                mpg ~ wt*cyl*carb
          )
```

```r
fits = lapply(formulas, lm, data = mtcars)
bic = lapply(fits, BIC)

fit_best = fits[[ which(bic == min(bic)) ]]
```

Part Two

Question 5

Background. In epidemiology, an *incident rate* is the rate at which new events occur in a population. It is given by the number of new events divided by either the number of individuals at risk at a specific point in time period or the total person-time at risk over a set period.

A 95% confidence interval (L, U) for the incident rate θ can be computed as:

$$\hat{\theta} := \frac{a}{N}, \qquad se(\hat{\theta}) := \sqrt{\frac{1-\hat{\theta}}{a}}$$

$$L := e^{\log(\hat{\theta}) - m \cdot se(\hat{\theta})}, U := e^{\log(\hat{\theta}) + m \cdot se(\hat{\theta})}$$

where a is the number of events, N is the total person-time at risk and $m = \Phi^{-1}(.975) \approx 1.96$.

An *incident rate ratio* compares the incident rates between two groups.

$$\hat{\rho} := \frac{\hat{\theta}_2}{\hat{\theta}_1}, \qquad \hat{\theta}_1 := \frac{a_1}{N_1}, \qquad \hat{\theta}_2 := \frac{a_2}{N_2}$$

A 95% confidence interval for ρ is given by computing the standard error of $\log \hat{\rho}$

$$se(\log \hat{\rho}) = \sqrt{\frac{1}{a_1} + \frac{1}{a_2}}$$

and then forming the confidence interval

$$(e^{\log(\hat{\rho}) - m \cdot se(\log \hat{\rho})}, e^{\log(\hat{\rho}) + m \cdot se(\log \hat{\rho})})$$

where a_1 and a_2 are the number of events used in computing $\hat{\theta}_1$ and $\hat{\theta}_2$, respectively.

Task. Write R code to carry out each of the the analysis described below. Your code should used vectorized operations, dplyr functions, and pipes where possible. [35 pts]

Consider a tab seprated file of the following form (this is the first two rows only):

```
id  group month year days_at_risk new_events
A02 1   Jan 2018  20  2
A02 1   Feb 2018  18  0
...
```

that has been read into a new R session as follows

```r
library(dplyr); libary(tidyr)

df = readr::read_delim('./event_data.tsv', delim = '\t')
```

Notes: Any working pipe is fine and the below are just one way to approach this using the most common dplyr functions. a. Write a dplyr pipe to count the total number of unique ids with at least one day at risk. [5 pts]

```
df %>% group_by(id) %>% summarize(days_at_risk = sum(days_at_risk)) %>%
  filter(days_at_risk > 0) %>% summarize(n())
```

b. Write a dplyr pipe to compute the total number of events and the total days at risk within each group and store the result in an object `total_events`. [5 pts]

```
total_events = df %>%
  group_by(group) %>%
  summarize(days_at_risk = sum(days_at_risk), new_events = sum(new_events))
```

c. Use your output from part (b) to compute the incident rate ~~ratio~~ and its standard error for each group. Then, form the point estimate and 95% CI as a nicely formated string. Store only the group and your nice string as `total_events_pretty`. [5 pts]

In the original background the parentheses were in the wrong place suggesting the log be taken after adding/subtracting the SE (they read correctly now). You get full credit for a solution either as written or as corrected here.

```
m = qnorm(.975)
total_events_pretty = total_events %>% # .5 pt
 mutate(
  ir = new_events / days_at_risk,   # 1pt
  se = sqrt( {1 - ir} / new_events),# 1pt
  L = exp(log(ir) - m*se),           # 1pt
  U = exp(log(ir) + m*se),
  ir_pretty = sprintf('%4.2f (%4.2f-%4.2f)', ir, se, L) # .5 pt
 ) %>%
 select(group, ir_pretty) # 1pt
```

d. Use your output from part (b) to compute incident rate ratios and their standard errors for all groups relative to group 1. Append a nicely formatted string with the point estimate and 95% CI to `total_events_pretty` [10 points]. *Hint: Use* `left_join()`.

```
# Create G?/G1 pairs
paired_totals =
  # Create a common variable to merge on
  total_events %>%
   mutate(help_merge = 1) %>%   # Create a common variable to merge on
   filter(group != 1) %>%       # Don't need group 1
  left_join(
     # Group 1 with new names for columns
     total_events %>% filter(group == 1) %>%
       mutate(new_events_1 = new_events,
              days_at_risk_1 = days_at_risk,
              help_merge = 1
       ), by = 'help_merge'
```

```
  )

# Compute IRR and its CI
paired_totals = paired_totals %>%
  mutate(
   IRR = {new_events / days_at_risk} / {new_events+1 / days_at_risk_1},
   SE = sqrt(1/days_at_risk + 1/days_at_risk_1),
   L = exp( log(IRR) - m*SE),
   U = exp( log(IRR) + m*SE),
   irr_with_group1 = sprintf('%4.2f (%4.2f-%4.2f)', IRR, L, U)
  ) %>%
  select(group, irr_with_group1)

# Merge with total_events_pretty
total_events_pretty = total_events_pretty %>%
 left_join(paired_totals, by = 'group')
```

e. Compute incident rate ratios and standard errors for all pairs of groups with the lower numbered group always in the denominator (you can assume integer-valued groups). Organize your code so that you compute each pair only once. [10 points]

- One way to accomplish this task is to use a "Cartesian" join by combining all pairs and then filtering the duplicate or inverse pairs. An implementation of this approach is left as an exercise.*
- A second approach is to turn your code from "d" into a function with "group"" as parameter. To save time and avoid writing the best way to do this is to go back and modify your answer to part d in place to also use this function.*

We will treat question 5e as a bonus.

```
compute_irr = function(total_events, g){
 # Pass an initial data frame "total events" and compute IRR
 # for all rows (except g) relative to group g.

 paired_totals =
   # Create a common variable to merge on
   total_events %>%
      mutate(help_merge = 1) %>%   # Create a common variable to merge on
      filter(group != g) %>%       # Don't need group 1
      left_join(
       # Group 1 with new names for columns
       total_events %>% filter(group == g) %>%
         mutate(new_events_1 = new_events,
              days_at_risk_1 = days_at_risk,
              help_merge = 1
         ), by = 'help_merge'
       )
```

```
  # Compute IRR and its CI
  paired_totals =
   paired_totals %>%
     mutate(
      IRR = {new_events / days_at_risk} / {new_events+1 / days_at_risk_1},
      SE = sqrt(1/days_at_risk + 1/days_at_risk_1),
      L = exp( log(IRR) - m*SE),
      U = exp( log(IRR) + m*SE),
      irr_with_group_g = sprintf('%4.2f (%4.2f-%4.2f)', IRR, L, U)
     ) %>%
     select(group, irr_with_group_g)

  names(paired_totals)[2] = sprintf('irr_with_group_%i', g)

  paired_totals
}

## Now use our function above to accomplish the task
max_g = max(df$group)
for(g in 2:{max_g - 1}){
  total_events_pretty = total_events_pretty %>%
    left_join( compute_irr(total_events %>% filter(group > g), g), by = 'group')
}
```

Question 6

Below is a short analysis script written in Stata. First, read the script and determine what output it produces. Use this information to provide a succinct but informative header. Then add comments in the four spaces indicated by [Comment X].

Finally, on the following page *write* an R script to produce the same output. [35 points]

```
*------------------------------------------------------------*
* [Add a header in the space below]
* Compute (n-1) times univariate regression coefficients for mpg vs
* other continuous variables by cylinder (cyl) groups.
*
* Updated: October 23, 2018
* Author: stata coder, stata_coder@analysis.com
*------------------------------------------------------------*

// [Comment one]
// Read data and reduce to needed variables
import delimited mtcars.csv

local vars = "disp hp wt"
keep mpg cyl `vars'

// [Comment two]
// Sort by cyl and then compute centered variables
// cross products with mpg.
gsort+ cyl

foreach var in `vars' {
 by cyl: egen `var'_grp_mean = mean(var)
 generate `var'_gc = `var' - `var'_grp_mean
 generate `var'Xmpg = mpg*`var'_gc
}

// Compute the cross products, variance, and (n-1)
// (n-1)*regression coefficients
collapse (sum) *Xmpg (var) *_gc, by(cyl)

// To make these accurate beta hats, there should also be a scaling factor
// of 1/{n-1}. This was missing from the script you were given.
quietly describe
local n = r(N)

// There was a typo here before: `varXmpg' vs `var'Xmpg.
foreach var in vars {
 generate beta_cyl_`var' = `var'Xmpg / (`var'_gc * (`n' - 1))
}
```

```
drop *Xmpg *_gc

// [Comment 4]: Export values (a better name would be mpg_beta_by_cyl.csv)
export delimited mpg_cor_by_cyl.csv
*60: ------------------------------------------------------*
```

Note: If you didn't recognize that these are (n-1) times regression coefficients within each group it was acceptable to describe the computed quantities directly using words or notation.

Write your R script for question 6 in the space below.

This is a fairly literal translation. You can create a more compact version (and have less to write/type) by directly centering in the "summarize" call.

```r
## Import data and select key terms
df = readr::read_csv('mtcars.csv')
df = df %>% select(mpg, cyl, disp, hp, wt)

## Center disp, hp, and wt within cyl and then compute "beta-hats"
beta_cyl = df %>%
 group_by(cyl) %>%
 mutate(
  disp_gc = disp - mean(disp),
  hp_gc = hp - mean(hp),
  wt_gc = wt - mean(wt)
 ) %>%
 # Compute cross products and variances
 summarize(
  dispXmpg = sum(mpg*disp_gc), vdisp = var(disp_gc),
  hpXmpg = sum(mpg*hp_gc), vhp = var(hp_gc),
  wtXmpg = sum(wt*wt_gc), vwt = var(wt_gc),
  n = n() ## Not needed in your solution.
 ) %>%
 # Compute betas
 # Scaling by (n-1) was not done in the
 # original Stata so is not expected in your solution.
 mutate(
   beta_cyl_disp = dispXmpg / {vdisp*{n-1}},
   beta_cyl_hp = hpXmpg / {vhp*{n-1}},
   beta_cyl_wt = wtXmpg / {vwt*{n-1}}
 )

readr::write_csv(beta_cyl, path = 'mpg_cor_by_cyl.csv')
```