Midterm SolutionsStatistics 506

Stats 506 F23 Midterm Solutions

Problem 1 - 18 points

For each of the following snippets of code, determine the value of `q` after execution.

a.

```
x <- 1:10
y <- x[which(x %% 2 == 1)]
q <- sum(y)
```

`x %% 2 == 1` identifies all odd values of `x`, so `y` contains only odd values through 9, which sum to 25.

```
q = 25
```

b.

```
x <- c(1, 5, 10, 15, 20)
y <- x*c(2, 1)
```

```
Warning in x * c(2, 1): longer object length is not a multiple of shorter
object length
```

```
q <- y[1] + y[5]
```

With recycling, both `x[1]` and `x[2]` are multipied by 2, so their sum is 42.

```
q = 42
```

c.

```r
x <- matrix(c(1, 3, 2, 2, 2, 1), nrow = 3, byrow = FALSE)
q <- apply(x, 1, sum) - apply(x, 2, sum)
```

```
Warning in apply(x, 1, sum) - apply(x, 2, sum): longer object length is not a
multiple of shorter object length
```

After obtaining the rowsums and colsums, there is recycling again.

```
q = c(-3, 0, -3)
```

d.

```r
f <- function(x) {
  ifelse(x > 2, f(x/2), x)
}
q <- f(10)
```

Function `f` divides its input in half until it goes below 2.

```
q = 1.25
```

e.

```r
tibble(a = 1:4, b = c(1, 2, 1, 2)) %>%
  mutate(c = a - b,
         d = c + b) %>%
  group_by(b) %>%
  summarize(e = mean(d)) %>%
  ungroup %>%
  arrange(-e) %>%
  filter(row_number() == 1) %>%
  select(e) %>%
  as.numeric -> q
```

```
q = 3
```

f.

```r
end = 1
goal = 30
```

```
total = 0
while ( total <= goal ) {
  for ( i in 1:end ) {
    total <- total + i
  }
  end <- end + 1
}
q <- total
```

This adds a increasing sequence of integers: `sum(1:1) + sum(1:2) + sum(1:3) + sum(1:4) + sum(1:5)`.

`q = 35`

Problem 2 - 14 points

Assume I have the following data.

first	last	year	age	owns_home
Afonso	Pelletier	2012	36	0
Putu	Contreras	2013	24	1
Lisbet	Gruber	2007	62	1
Catrine	Smit	2020	18	0
Vilté	Proudfoot	2017	42	1

The data is stored in "work.housing". Write SAS code to

a. Sort the file by "age"
b. Generate a variable "birthyear"
c. Compute the average birthyear amongst home owners and non-homeowners.

(Hint: One or more of these may require multiple `data` and/or `proc` steps.)

```
/* You didn't have to manually input the data, doing it here so
   the example runs to completion. */
data housing;
    input year age owns_home;
    datalines;
2012 36 0
2013 24 1
2007 62 1
```

```
2020 18 0
2017 42 1
run;

/* a. */
/* the out= is optional, required if you want to
   store the table in a different place */
proc sort data=housing out=housing;
    by age;
run;

/* b. */
data housing;
    set housing;
    birthyear = year - age;
run;

/* c. */
/* proc means needs the data sorted by `by` prior
   to running */
proc sort data=housing;
    by owns_home;
run;
proc means data=housing;
    var birthyear;
    by owns_home;
run;
```

Problem 3 - 20 points

For each of the following `regress` calls from Stata, produce R code to replicate the model. You may assume the data is stored in equivalent fashion in both software, with the R data being stored in a `data.frame` named "dat", with all columns being numeric. **You should write only one line of R code.**

a. `regress y x1 i.x2 c.x3##c.x4`

```
lm(y ~ x1 + as.factor(x2) + x3*x4, data = dat)
```

b. `regress y i.x1##i.x2##x3, nocons`

```
lm(y ~ as.factor(x1)*as.factor(x2)*as.factor(x3) + 0, data = dat)
```

Using `- 1` to remove the intercept works as well. In Stata, when a variable is involved in an interaction, Stata treats it as categorical by default.

c. `regress y x1 c.x1#c.x2`

```
lm(y ~ x1 + x1:x2, data = dat)
```

d.

```
generate z = log(x3)
regress y c.x1#c.x1 x2 z
```

```
lm(y ~ I(x1*x1) + x2 + log(x3), data = dat)
```

Complete each of the following pieces of Stata code according to the comments.

e.

```
* Loop over each of the following binary variable currently stored as
* 1/2 and convert to 0/1
foreach var of varlist bin1-bin20 {
    replace `var' = `var' - 1
}
```

f.

```
regress depress i.race##c.age
* Draw an interaction plot with `age` on the x-axis at values 20
* through 80 by 10s, plotting line per `race`
margins race, at(age = (20(10)80))
marginsplot
```

g.

```
mean x1 x2, by(group)
* calculate squared difference between `x1` and `x2`
generate sd = (x1 - x2)^2
* replace the data with a dataset that contains the mean squared
```

```
* difference between `x1` and `x2` per `group`
collapse (mean) sd, by(group)
```

h.

```
mata:
X = st_matrix("x")
// find the largest value on the diagonal of `X`
Q = max(diagonal(X))
st_matrix("q", Q)
end
matrix list q
```

Problem 4 - 20 points

Fill out the following table. Place an X in each cell which the regular expression will match the string. Leave the remaining cells blank.

	`[Aa]*[a-zA-Z]+.{2}o\$`	`^..?...$`	`^[^AS]..?[aeiou]`	`(.).\\1`
Adedayo	X			X
Dinah		X	X	
Frida		X	X	
Liam		X	X	
Marlyn				
Milko	X	X		
Mirka		X		
Rupert			X	
Rustik				
Sebastian				

The first expression matches anything that ends in an "o" as long as it is at least 4 letters and the first of those four isn't an "A" or "a". (Technically the 2nd and 3rd don't have to be letters, but we're only dealing with names here.)

The second expression matches anything exactly 4 or 5 characters long.

The third expression matches anything with a vowel in the 3rd or 4th position, as long as it doens't start with "A" or "S".

The fourth expression matches any string containing a repeated character seperated by one position.

Problem 5 - 20 points

Let `data` be the following table:

```
  x y q z
1 2 7 1 1
2 9 8 1 2
3 7 5 2 3
4 3 5 2 4
5 1 8 2 5
6 6 4 1 6
```

and `data2` be

```
  z p
1 1 4
2 7 8
```

For each of the following SQL queries, what will the output table be? Be sure to provide column names as appropriate.

a.

```
SELECT *
  FROM data
 WHERE y > 7
```

```
  x y q z
1 9 8 1 2
2 1 8 2 5
```

(Row numbers were not required.)

b.

```
SELECT x, x - 1 AS x1
  FROM data
 LIMIT 2
```

```
  x x1
1 2  1
2 9  8
```

c.

```
SELECT q, sum(x) AS xx
  FROM data
 GROUP BY q
HAVING xx < 15
```

Without the "HAVING" clause, we'd have

```
  q xx
1 1 17
2 2 11
```

so the final table is

```
  q xx
1 2 11
```

d.

```
SELECT x, p
  FROM data AS d1
  LEFT JOIN data2 AS d2 ON d1.z = d2.z
```

```
  x  p
1 2  4
2 9 NA
3 7 NA
4 3 NA
5 1 NA
6 6 NA
```

e.

```
SELECT x, p
  FROM data AS d1
 RIGHT JOIN data2 AS d2 ON d1.z = d2.z
```

```
   x p
1  2 4
2 NA 8
```

Problem 6 - 20 points

Suppose we have an unfair coin. We know that $P(\text{heads}) = p$, $P(\text{tails}) = q$ and $p \neq q$ but we do not know p or q. One way to obtain a fair result from such a coin is to carry out the following procedure. Flip the coin twice, recording the first and second flip in order as a pair. If the pair has different results, return the result of the first flip. If the pair has the same results, repeat the procedure. (This is called the "Von Neumann Extractor".)

a. Write an R function that takes in a proportion p for a biased coin, and returns a single heads or tails result using the procedure above. Name your function "vonneumann". You do not need to check your input; you can assume the proportion is a valid number strictly between 0 and 1. You can use the `rbinom` function to generate the random coin flips, it takes in two arguments: `n` and `p`. The output should be a binary where 0 represents tails, and 1 represents heads. Do not worry about whether your function finishes in finite time.

Solution:

There are of course many *many* ways you can solve this. Here's one:

```r
vonneumann <- function(p) {
  while(TRUE) {
    flips <- rbinom(2, 1, p) # The actual `rbinom` function takes in an
                             # additional argument that I suppressed for the
                             # exam
    if (flips[1] != flips[2]) {
      return(flips[1])
    }
  }
}
vonneumann(.5)
```

```
[1] 1
```

```r
vonneumann(.99)
```

```
[1] 1
```

b. Using your `vonneumann` function, write a Monte Carlo simulation to demonstrate that it does in fact produce fair results. Start with the following parameters:

```
p <- .7
reps <- 10000
```

Your code should produce an estimate of the proportion of heads in `reps` loops of the procedure.

Solutions:

Again there, are many *many* possible solutions. The shortest is probably:

```
mean(sapply(1:reps, function(x) vonneumann(p)))
```

```
[1] 0.4919
```

A more manual (and completely acceptable solution) would be

```
save <- rep(-1, reps) # Make sure that your "default" value is nonsensical
for (i in 1:reps) {
  save[i] <- vonneumann(p)
}
mean(save)
```

```
[1] 0.5011
```