# Bios 6301: Assignment 7

Lan Shi

*Due Thursday, 04 November, 1:00 PM*

$5^{n=day}$ points taken off for each day late.

40 points total.

Submit a single knitr file (named `homework7.rmd`), along with a valid PDF output file. Inside the file, clearly indicate which parts of your responses go with which problems (you may use the original homework document as a template). Add your name as `author` to the file's metadata section. Raw R code/output or word processor files are not acceptable.

Failure to name file `homework7.rmd` or include author name may result in 5 points taken off.

### Question 1

**21 points**

Use the following code to generate data for patients with repeated measures of A1C (a test for levels of blood glucose).

```
genData <- function(n) {
    if(exists(".Random.seed", envir = .GlobalEnv)) {
        save.seed <- get(".Random.seed", envir= .GlobalEnv)
        on.exit(assign(".Random.seed", save.seed, envir = .GlobalEnv))
    } else {
        on.exit(rm(".Random.seed", envir = .GlobalEnv))
    }
    set.seed(n)
    subj <- ceiling(n / 10)
    id <- sample(subj, n, replace=TRUE)
    times <- as.integer(difftime(as.POSIXct("2005-01-01"),
                                 as.POSIXct("2000-01-01"), units='secs'))
    dt <- as.POSIXct(sample(times, n), origin='2000-01-01')
    mu <- runif(subj, 4, 10)
    a1c <- unsplit(mapply(rnorm, tabulate(id), mu, SIMPLIFY=FALSE), id)
    data.frame(id, dt, a1c)
}
x <- genData(500)
```

Perform the following manipulations: (3 points each)

1. Order the data set by `id` and `dt`.

```
x1 = x[order(x$id,x$dt),]
```

2. For each `id`, determine if there is more than a one year gap in between observations. Add a new row at the one year mark, with the `a1c` value set to missing. A two year gap would require two new rows, and so forth.

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
addmissing = function(y){
  delta = as.numeric(difftime(tail(y$dt, -1),
                   head(y$dt, -1),units = "days"))
  #print(delta)
  # compute the number of gaps for each time difference
  ngap = as.numeric(delta)%/%365
  # get the row index for the gaps
  rowidx_gap = seq_along(ngap)[ngap!=0]

  for (i in rowidx_gap){
    # get row i
    row_i = unlist(y[i,]) #here, the datetime will be coerced to UNIX form
    new_rows = data.frame(matrix(rep(row_i,ngap[i]),
                      byrow=T,nrow=ngap[i]))
    names(new_rows) = names(y)
    new_rows$dt = as.POSIXct(new_rows$dt, origin="1970-01-01")
    new_rows$id = as.integer(new_rows$id)
    for (j in 1:ngap[i]) new_rows$dt[j]=new_rows$dt[j]+years(j)
    new_rows$a1c = NA
    y = rbind(y,new_rows)
  }
  y[order(y$dt),]
}

datById = split(x1,x1$id)
x2 = lapply(datById, addmissing)
#x2
```

3. Create a new column `visit`. For each `id`, add the visit number. This should be 1 to `n` where `n` is the number of observations for an individual. This should include the observations created with missing a1c values.

```
x3 = lapply(x2, function(y){
  y$visit = 1:nrow(y)
  y})
#x3
```

4. For each `id`, replace missing values with the mean `a1c` value for that individual.

```
x4 = lapply(x3, function(y){
  m = mean(y$a1c,na.rm = T)
  y$a1c[is.na(y$a1c)] = m
  y
})
#x4
```

5. Print mean `a1c` for each `id`.

```
x_final = do.call(rbind.data.frame, x4)
# mean `a1c` for each `id`
tapply(x_final$a1c,x_final$id,mean)
```

```
##        1        2        3        4        5        6        7        8
##  6.654444  9.789132  6.951820  8.191985  9.429694  7.133443  7.879138  6.244061
##        9       10       11       12       13       14       15       16
##  4.420523  6.028370  4.838279  6.691181  8.504632  9.122968  6.737092  7.420245
##       17       18       19       20       21       22       23       24
##  6.546329  6.151311  8.628037  8.923518  5.444430  5.763931  6.351112  9.377525
##       25       26       27       28       29       30       31       32
##  5.058097  8.692078  7.371831  4.243469  6.345254  4.135795  8.670622  5.130167
##       33       34       35       36       37       38       39       40
##  6.528153  8.445030  3.832195  9.514603  8.612608 10.160773  8.976697  7.583232
##       41       42       43       44       45       46       47       48
##  3.804325  6.787170  5.654235  5.613283  8.876623  7.485824  4.752133  7.415459
##       49       50
##  5.562809  4.970288
```

6. Print total number of visits for each `id`.

```
tapply(x_final$visit,x_final$id,length)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
##  7 16 13  9 14 11  7 12 15  8 12 12  9 12 10  8 10 14 10 11 13 12 10 12 16 11
## 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
## 10 15  3 13 11  9 12 12 11 10  8 14 14 11 14 11  8 12  6 12 10  5 11  9
```

7. Print the observations for `id = 15`.

```
x4$`15`
```

```
##     id                  dt      a1c visit
## 300 15 2000-10-21 01:08:17 7.401322     1
## 127 15 2001-08-08 14:23:08 5.896318     2
## 165 15 2001-08-15 07:03:29 7.457722     3
## 109 15 2002-03-15 21:23:10 5.330917     4
## 319 15 2002-04-14 09:08:25 6.484003     5
## 255 15 2002-10-10 18:27:43 8.139101     6
## 224 15 2003-02-19 12:58:53 6.446557     7
## 481 15 2003-03-02 06:58:10 7.432291     8
## 425 15 2003-06-30 07:20:49 7.113792     9
## 259 15 2004-01-22 20:30:42 5.668897    10
```

**Question 2**

**16 points**

Install the `lexicon` package. Load the `sw_fry_1000` vector, which contains 1,000 common words.

```
library(lexicon)
data('sw_fry_1000', package = 'lexicon')
#head(sw_fry_1000)
```

1. Remove all non-alphabetical characters and make all characters lowercase. Save the result as `a`.

```
a = tolower(gsub('[^a-zA-Z]','',sw_fry_1000))
```

Use vector `a` for the following questions. (2 points each)

2. How many words contain the string “ar”?

```
#grep('ar',sw_fry_1000,value = T)
length(grep('ar', a))
```

```
## [1] 64
```

3. Find a six-letter word that starts with “l” and ends with “r”.

```
grep('^l[a-z]{4}r$', a, value = T)
```

```
## [1] "letter"
```

4. Return all words that start with “col” or end with “eck”.

```
grep('^col|eck$', a, value = T)
```

```
## [1] "color"   "cold"    "check"   "collect" "colony"  "column"  "neck"
```

5. Find the number of words that contain 4 or more adjacent consonants. Assume “y” is always a consonant.

```
grep('[^aeiou]{4,}', a, value = T)
```

```
## [1] "country"    "system"     "syllable"   "length"     "instrument"
## [6] "industry"   "symbol"     "supply"
```

6. Return all words with a “q” that isn’t followed by a “ui”.

```
# if q then not u => not qui
# if qu => then not i => not qui
# or ends with q or qu
grep('q[^u]|qu[^i]|qu$|q$', a, value = T)
```

```
## [1] "question" "equate"   "square"   "equal"    "quart"    "quotient"
```

7. Find all words that contain a “k” followed by another letter. Run the `table` command on the first character following the first “k” of each word.

```
ks = grep('k[a-z]', a, value = T)
ks
```

```
##  [1] "like"   "make"   "know"   "take"   "kind"   "keep"   "knew"   "king"
##  [9] "sky"    "kept"   "broke"  "kill"   "lake"   "key"    "skin"   "spoke"
## [17] "skill"  "market"
```

```
table(substr(sub('[^k]*k',"",ks),1,1))
```

```
##
##  e  i  n  y
## 10  5  2  1
```

8. Remove all vowels. How many character strings are found exactly once?

```
novowels = gsub('[aeiou]','',a)
#novowels
sum(table(novowels)==1)
```

```
## [1] 581
```

**Question 3**

**3 points**

The first argument to most functions that fit linear models are formulas. The following example defines the response variable `death` and allows the model to incorporate all other variables as terms. `.` is used to mean all columns not otherwise in the formula.

```
url <- "https://raw.githubusercontent.com/couthcommander/Bios6301/main/datasets/haart.csv"
haart_df <- read.csv(url)[,c('death','weight','hemoglobin','cd4baseline')]
coef(summary(glm(death ~ ., data=haart_df, family=binomial(logit))))
```

```
##                 Estimate  Std. Error   z value     Pr(>|z|)
## (Intercept)  3.576411744 1.226870535  2.915069 0.0035561039
## weight      -0.046210552 0.022556001 -2.048703 0.0404911395
## hemoglobin  -0.350642786 0.105064078 -3.337418 0.0008456055
## cd4baseline  0.002092582 0.001811959  1.154872 0.2481427160
```

Now imagine running the above several times, but with a different response and data set each time. Here's a function:

```
myfun <- function(dat, response) {
  form <- as.formula(response ~ .)
  coef(summary(glm(form, data=dat, family=binomial(logit))))
}
```

Unfortunately, it doesn't work. `tryCatch` is "catching" the error so that this file can be knit to PDF.

```
tryCatch(myfun(haart_df, death), error = function(e) e)
```

```
## <simpleError in eval(predvars, data, env): object 'death' not found>
```

What do you think is going on? Consider using `debug` to trace the problem.

```
  In "as.formula(response ~ .)", the response is fixed as "response",
  but what actually needed here is the name of our real response, i.e., death.
```

```
debugonce(myfun)
myfun(haart_df, death)
```

**5 bonus points**

Create a working function.

```
myfun <- function(dat, response) { # response should be a character
  form <- as.formula(paste(response,'~ .'))
  coef(summary(glm(form, data=dat, family=binomial(logit))))
}
myfun(haart_df, "death")
```

```
##                 Estimate  Std. Error   z value     Pr(>|z|)
## (Intercept)  3.576411744 1.226870535  2.915069 0.0035561039
## weight      -0.046210552 0.022556001 -2.048703 0.0404911395
## hemoglobin  -0.350642786 0.105064078 -3.337418 0.0008456055
## cd4baseline  0.002092582 0.001811959  1.154872 0.2481427160
```