Bios 6301: Assignment 7

Megan Taylor

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`.

```
x <- x[order(x[,'id'], x[,'dt']),]
rownames(x) = NULL
```

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.

```
# set vector to keep track of all time differences
alldiff <- c()
# make an index column so can sort by it later
x[,'ind'] <- seq_len(nrow(x))
# loop for each id
for (id in 1:length(unique(x[,'id']))){
  # initialize vectors of different times
  difftimes = c()
  # for each row of the id
  for (n in 1:length(which(x[,'id']==id))-1){
    # get relevant rows
    rows <- which(x[,'id']==id)
    # get time difference in days for each date of the current id
    difftimes <- c(difftimes, difftime(x[rows[n+1],'dt'], x[rows[n], 'dt']))
    # also store them in the overall time differences vector
    alldiff <- c(alldiff, difftime(x[rows[n+1],'dt'], x[rows[n], 'dt']))
  }
  # if there is at least a one year gap
  if (max(difftimes) > 365){
    # get the indexes within the id for gaps of 1 year or more
    num1 <- which(difftimes > 365)
    # get the indexes within the id for gaps of two years or more
    num2 <- which(difftimes >730)
    # get indexes from main data frame
    inds <- which(x[,'id']==id)
    # for each gap of at least a year
    for (i in num1){
      # add a row at 1 year point with an index between the indexes of the gap times and add missing a1
      x <- rbind(x,data.frame('id' = id, 'dt' = x[inds[i],'dt']+31536000, 'a1c' = NA, 'ind'= inds[i]+.1
    }
    # add another row at 2 year point with an index higher than 1 year gap but before next visit for ga
    for (i in num2){
      x <- rbind(x,data.frame('id' = id, 'dt' = x[inds[i],'dt']+31536000*2, 'a1c' = NA, 'ind'= inds[i]+
    }
  }
}
# sort by index so everything is in order
x <- x[order(x[,'ind']),]
# remove index column
x <- x[,-4]
# this is how I knew only needed to account for gaps of 2 years
max(alldiff)/365
```

```
## [1] 2.659525
```

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.

```r
x[,'visit'] = NA
for (id in 1:length(unique(x[,'id']))){
  vs = length(which(x[,'id']==id))
  inds <- which(x[,'id']==id)
  counter = 1
  for (v in 1:vs){
    x[inds[v],'visit'] = counter
    counter = counter + 1
  }
}
```

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

```r
for (id in 1:length(unique(x[,'id']))){
  ns <- which(x[,'id']==id)
  m <- mean(x[ns, 'a1c'], na.rm=TRUE)
  for (n in ns){
    if (is.na(x[n, 'a1c'])){
      x[n,'a1c'] = m
    }
  }
}
```

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

```r
for (id in 1:length(unique(x[,'id']))){
  ns <- which(x[,'id']==id)
  m <- mean(x[ns, 'a1c'])
  print(paste("Mean of id", id, "=", m))
}
```

```
## [1] "Mean of id 1 = 6.65444426795186"
## [1] "Mean of id 2 = 9.78913246074151"
## [1] "Mean of id 3 = 6.95182045895334"
## [1] "Mean of id 4 = 8.19198450682839"
## [1] "Mean of id 5 = 9.42969414135007"
## [1] "Mean of id 6 = 7.13344348656912"
## [1] "Mean of id 7 = 7.87913801432509"
## [1] "Mean of id 8 = 6.24406099245875"
## [1] "Mean of id 9 = 4.42052304020483"
## [1] "Mean of id 10 = 6.02836978936866"
## [1] "Mean of id 11 = 4.83827911476455"
## [1] "Mean of id 12 = 6.69118108424096"
## [1] "Mean of id 13 = 8.50463215686808"
## [1] "Mean of id 14 = 9.12296781957672"
## [1] "Mean of id 15 = 6.73709205512209"
## [1] "Mean of id 16 = 7.42024462564604"
## [1] "Mean of id 17 = 6.54632858730216"
## [1] "Mean of id 18 = 6.1513112940644"
## [1] "Mean of id 19 = 8.62803745758515"
## [1] "Mean of id 20 = 8.92351824057672"
## [1] "Mean of id 21 = 5.444430006372"
## [1] "Mean of id 22 = 5.76393126014759"
## [1] "Mean of id 23 = 6.35111217834161"
## [1] "Mean of id 24 = 9.37752492553745"
## [1] "Mean of id 25 = 5.05809652490457"
## [1] "Mean of id 26 = 8.69207762927627"
## [1] "Mean of id 27 = 7.37183147872539"
## [1] "Mean of id 28 = 4.24346852483802"
## [1] "Mean of id 29 = 6.34525429737664"
## [1] "Mean of id 30 = 4.13579498572139"
## [1] "Mean of id 31 = 8.67062198152496"
## [1] "Mean of id 32 = 5.1301670704902"
## [1] "Mean of id 33 = 6.52815306924961"
## [1] "Mean of id 34 = 8.44503021368734"
## [1] "Mean of id 35 = 3.83219482233089"
## [1] "Mean of id 36 = 9.51460255980355"
## [1] "Mean of id 37 = 8.61260794411042"
## [1] "Mean of id 38 = 10.160772908825"
## [1] "Mean of id 39 = 8.97669727861485"
## [1] "Mean of id 40 = 7.58323173368407"
## [1] "Mean of id 41 = 3.8043252144796"
## [1] "Mean of id 42 = 6.78716991115953"
## [1] "Mean of id 43 = 5.65423470328969"
## [1] "Mean of id 44 = 5.61328261848045"
## [1] "Mean of id 45 = 8.8766234785112"
## [1] "Mean of id 46 = 7.4858240579994"
## [1] "Mean of id 47 = 4.75213278333204"
## [1] "Mean of id 48 = 7.41545866940117"
## [1] "Mean of id 49 = 5.56280902415056"
## [1] "Mean of id 50 = 4.97028797276639"
```

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

```r
for (id in 1:length(unique(x[,'id']))){
  v = length(which(x[,'id']==id))
  print(paste("Visits for id", id, "=", v))
}
```

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

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

```
x[which(x[,'id']==15),]
```

```
##     id                  dt      a1c visit
## 144 15 2000-10-21 01:08:17 7.401322     1
## 145 15 2001-08-08 14:23:08 5.896318     2
## 146 15 2001-08-15 07:03:29 7.457722     3
## 147 15 2002-03-15 21:23:10 5.330917     4
## 148 15 2002-04-14 09:08:25 6.484003     5
## 149 15 2002-10-10 18:27:43 8.139101     6
## 150 15 2003-02-19 12:58:53 6.446557     7
## 151 15 2003-03-02 06:58:10 7.432291     8
## 152 15 2003-06-30 07:20:49 7.113792     9
## 153 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.

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

```
## [1] "the" "of"  "to"  "and" "a"   "in"
```

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

```
a <- tolower(sub("[^[:alpha:]]","", sw_fry_1000))
```

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

2. How many words contain the string "ar"?

```
length(grep("ar", a))
```

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

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

```
grep("^l.*r$", a, value=TRUE)
```

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

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

```
grep("^col|eck$", a, value=TRUE)
```

```
## [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.

```
length(grep("[^a|e|i|o|u]{4}", a, value=TRUE))
```

```
## [1] 8
```

6. Return all words with a "q" that isn't followed by a "ui".

```
grep('q[^u]', a, value=TRUE) # all q's are followed by a u
```

```
## character(0)
```

```
grep('qu[^i]', a, value=TRUE)
```

```
## [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, value=TRUE)
s = c()
for (word in ks){
  l <- regexpr('k.', word)
  s <- c(s, substr(word, l+1, l+attr(l, 'match.length')-1))
}
table(s)
```

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

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

```
b <- sub("a|e|i|o|u", "", a)
tb <- table(b)
length(which(tb==1))
```

```
## [1] 806
```

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 didn't work so I read it in from my downloaded files
# url <- "https://github.com/couthcommander/Bios6301/raw/master/datasets/haart.csv"
haart_df <- read.csv("C:\\Users\\megan\\Documents\\Fall 2021\\Statistical Computing\\Bios6301-main\\Bios
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:

```r
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.

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

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

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

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

The as.formula function is returning response ~ . instead of death ~ . . The argument to as.formula should be "death ~ .".

5 bonus points

Create a working function.

```r
myfun1 <- function(dat, response) {
  c <- substitute(response)
  form <- paste(c, "~ .")
  form <- as.formula(form)
  coef(summary(glm(form, data=dat, family=binomial(logit))))
}

myfun1(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
```