Bios 6301: Assignment 7

Catherine Greene

Due Thursday, 03 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`.

```
#table(x['id']) # each id has multiple instances, sort by dt within each id group
x_ordered <- x[order(x[,'id'], x['dt']),]
```

```
## Warning in xtfrm.data.frame(x): cannot xtfrm data frames
```

```
rownames(x_ordered) <- 1:nrow(x_ordered) # reset the indices
```

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.

```
install.packages("lubridate", repos = "http://cran.us.r-project.org") # install lubridate
```

```
##
## The downloaded binary packages are in
##  /var/folders/fn/3nmxs93x7wn7qxpwl0ky75t00000gn/T//RtmpNmLpxY/downloaded_packages
```

```
library("lubridate")
```

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

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

```
rows <- nrow(x_ordered)-1 # I was having trouble getting this to work inline

for (i in (1:rows)) {
  if (x_ordered[,'id'][i] == x_ordered[,'id'][i+1]) { # for the same id
    gap <-abs(difftime(x_ordered[,'dt'][i], x_ordered[,'dt'][i+1], units='days')) # what is the gap
  }
  if ((gap>365) && (gap<(2*365))) { # is the gap bigger than a year but less than 2 years
    gap_df <- data.frame(x_ordered[,'id'][i], x_ordered[,'dt'][i] %m+% years(1), 'NA')
    names(gap_df) <- c('id','dt','a1c')
    x_ordered <- rbind(x_ordered, gap_df)
    #print(gap)
  } else if (gap>(2*365)) { # is the gap bigger than 2 years
    gap_df1 <- data.frame(x_ordered[,'id'][i], x_ordered[,'dt'][i] %m+% years(1), 'NA')
    gap_df2 <- data.frame(x_ordered[,'id'][i], x_ordered[,'dt'][i] %m+% years(2), 'NA')
    names(gap_df1) <- c('id','dt','a1c')
    names(gap_df2) <- c('id','dt','a1c')
    x_ordered <- rbind(x_ordered, gap_df1, gap_df2) # I did it this way because it defaulted to wrong o
    #print (gap)
  }
}

x_ordered_update <- x_ordered[order(x_ordered[,'id'], x_ordered['dt']),]
```

```
## Warning in xtfrm.data.frame(x): cannot xtfrm data frames
```

```
rownames(x_ordered_update) <- 1:nrow(x_ordered) # reset the indices
```

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.

```
visit <- c()

for (i in 1:length(unique(x_ordered_update[,'id']))) {
  visit_sub <- c(seq(table(x_ordered_update[,'id'])[i]))
  visit <- c(visit, visit_sub)
}

x_visit <- cbind(x_ordered_update, visit)
```

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

```
# I know that each id is 1-50, so I'm going to make my life easier and just say that
a1c_update <- c()
for (i in 1:50) {
  subset <- suppressWarnings(as.numeric(x_visit[which(x_visit['id']==i),]['a1c']))
  subset[which(is.na(subset) == TRUE)] <- mean(subset, na.rm = TRUE)
  a1c_update <- c(a1c_update, subset)
}

x_final <- cbind(x_visit, a1c_update)[,-3]
```

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

```
# same as above, no NAs this time though
a1c_means <- c()

for (i in 1:50) {
  subset <- x_final[which(x_final['id']==i),]['a1c_update']
  a1c_means[i] <- mean(subset)
}

a1c_means
```

```
##  [1]  6.654444  9.789132  6.951820  8.191985  9.429694  7.133443  7.879138
##  [8]  6.244061  4.420523  6.028370  4.838279  6.691181  8.504632  9.122968
## [15]  6.737092  7.420245  6.546329  6.151311  8.628037  8.923518  5.444430
## [22]  5.763931  6.351112  9.377525  5.058097  8.692078  7.371831  4.243469
## [29]  6.345254  4.135795  8.670622  5.130167  6.528153  8.445030  3.832195
## [36]  9.514603  8.612608 10.160773  8.976697  7.583232  3.804325  6.787170
## [43]  5.654235  5.613283  8.876623  7.485824  4.752133  7.415459  5.562809
## [50]  4.970288
```

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

```
table(x_final['id'])
```

```
##
##  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  9 12 12  9 12 10  8 11 14 10 11 13 12 10 12 17 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 12  9 12 13 11 11  8 14 14 13 14 11  8 12  6 13 10  5 11  9
```

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

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

```
##     id                  dt visit a1c_update
## 159 15 2000-10-21 01:08:17     1   7.401322
## 160 15 2001-08-08 14:23:08     2   5.896318
## 161 15 2001-08-15 07:03:29     3   7.457722
## 162 15 2002-03-15 21:23:10     4   5.330917
## 163 15 2002-04-14 09:08:25     5   6.484003
## 164 15 2002-10-10 18:27:43     6   8.139101
## 165 15 2003-02-19 12:58:53     7   6.446557
## 166 15 2003-03-02 06:58:10     8   7.432291
## 167 15 2003-06-30 07:20:49     9   7.113792
## 168 15 2004-01-22 20:30:42    10   5.668897
```

Question 2

16 points

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

```
install.packages('lexicon', repos = "http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/fn/3nmxs93x7wn7qxpwl0ky75t00000gn/T//RtmpNmLpxY/downloaded_packages
```

```
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(grep("[^1-9]", sw_fry_1000, value=TRUE))
```

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

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

```
length(grep("ar", sw_fry_1000, value=TRUE))
```

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

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

```
grep("^l.*r${6}", sw_fry_1000, value=TRUE)
```

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

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

```
grep("^col|eck$", sw_fry_1000, 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.

```
grep("[b,c,d,f,g,h,j,k,l,m,n,p,q,r,s,t,v,w,x,y,z]{4}", sw_fry_1000, value=TRUE)
```

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

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

```
grep("q.[^ui]", sw_fry_1000, 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.

```
k <- grep("k[a-z]", sw_fry_1000, value=TRUE)
k_cut <- sub(".*k.*([a-z].*$)", "\\1", k)
table(k_cut)
```

```
## k_cut
## d e g l n p t w y
## 1 6 1 2 1 1 2 2 2
```

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

```
no_vowels <- gsub("[aeiou]", "", sw_fry_1000)
length(unique(no_vowels)) #743 unique character strings, once vowels are cut out
```

```
## [1] 743
```

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

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 is.data.frame(data): object 'haart_df' not found>
```

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

```
#debug(myfun)
#myfun(haart_df, death)

# This may just be a problem on my end, but it seems that the issue is originating with the url. The we
```

5 bonus points

Create a working function.