Bios 6301: Assignment 7

Jonathan Lifferth

Due Thursday, 02 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).

```r
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`.

```
head(x)
```

```
##   id                  dt      a1c
## 1 39 2001-11-30 00:36:10 8.758993
## 2 27 2000-07-20 17:05:06 8.233413
## 3 47 2001-02-06 18:40:44 3.950582
## 4 50 2001-04-05 01:55:44 6.707085
## 5 41 2003-08-05 18:30:14 4.047338
## 6 31 2000-12-16 10:38:26 9.357902
```

```
x<-x[order(x[,'id'], x[,'dt']),]
head(x)
```

```
##     id                  dt      a1c
## 32   1 2001-05-08 16:22:52 7.309995
## 268  1 2001-06-17 22:42:23 8.310721
## 201  1 2001-08-17 16:51:46 6.548845
## 285  1 2001-12-14 14:50:29 5.985275
## 194  1 2002-08-19 13:51:47 6.011547
## 304  1 2003-03-22 03:51:36 7.243858
```

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
```

```
x <- genData(500)
x_ordered<-x[order(x[,'id'], x[,'dt']),]
rownames(x_ordered) <- NULL

x <- x[order(x[,'id'], x[,'dt']),]
rownames(x) <- NULL
gap_years <- data.frame()

nrow(x)
```

```
## [1] 500
```

```
for (i in 1:(nrow(x)-1)) {
  current_id <- x[i,'id']
  next_id <- x[i+1,'id']
```

```
    # only proceed if we're evaluating the same id
    if (next_id == current_id) {
        # what is the time gap between the current row and the next row?
        delta <- abs(difftime(x[i+1,'dt'], x[i,'dt'], units = 'days')[[1]])
        num_years <- trunc(delta / 365)

        # add new rows for each observation year gap
        if (num_years > 0) {
          for (j in 1:num_years) {
            new_row <- data.frame(id = current_id, dt = x[i,'dt']+years(j), a1c=NA)
    #        x <- rbind(x[1:i,], new_row, x[i+1:nrow(x),])
            gap_years <- rbind(gap_years, new_row)
        }
      }
    }

}
```

```
nrow(gap_years)
```

```
## [1] 45
```

```
x2 <- rbind(x, gap_years)
x2<-x2[order(x2[,'id'], x2[,'dt']),]
rownames(x2) <- NULL

head(x2)
```

```
##   id                  dt      a1c
## 1  1 2001-05-08 16:22:52 7.309995
## 2  1 2001-06-17 22:42:23 8.310721
## 3  1 2001-08-17 16:51:46 6.548845
## 4  1 2001-12-14 14:50:29 5.985275
## 5  1 2002-08-19 13:51:47 6.011547
## 6  1 2003-03-22 03:51:36 7.243858
```

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.

```
x2$visit <- 0
count <- 1 # this is the default value for visit number
for (i in 1:nrow(x2)) {

  # we need to exclude the very first row, because it can't be compared to any previous row ids
  if (i > 1) {
    if (x2[i, 'id'] == x2[i-1, 'id']) {
      count <- count + 1
      x2[i, 'visit'] <- count
    }
    else { # this indicates the first row for a new id
```

```
      count <- 1
      x2[i, 'visit'] <- count
    }

  } else { # this condition should only apply to the very first row in the data.frame
    x2[i, 'visit'] <- 1
  }
}
```

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

```
for (i in 1:nrow(x2)) {
  if (is.na(x2[i,'a1c'])) {
    x2[i,'a1c'] <- mean(subset(x2,id == x2[i,'id'])$a1c, na.rm=TRUE)
  }
}
```

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

```
for (i in unique(x2$id)) {
  print(paste('id: ', i, '   ' ,mean(subset(x2,id == i)$a1c, na.rm=TRUE)))
}
```

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

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

```
for (i in unique(x2$id)) {
  print(paste('id: ', i, '   ' ,max(subset(x2,id == i)$visit)))
}
```

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

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

```
subset(x2, id==15)
```

```
##     id                  dt      a1c visit
## 158 15 2000-10-21 01:08:17 7.401322     1
## 159 15 2001-08-08 14:23:08 5.896318     2
## 160 15 2001-08-15 07:03:29 7.457722     3
## 161 15 2002-03-15 21:23:10 5.330917     4
## 162 15 2002-04-14 09:08:25 6.484003     5
## 163 15 2002-10-10 18:27:43 8.139101     6
## 164 15 2003-02-19 12:58:53 6.446557     7
## 165 15 2003-03-02 06:58:10 7.432291     8
## 166 15 2003-06-30 07:20:49 7.113792     9
## 167 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.

```
#install.packages('lexicon')
library('lexicon')
```

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

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

```r
length(sw_fry_1000)
```

```
## [1] 1000
```

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

```r
sw_fry_1000[which(grepl("[^[:alpha:]]", sw_fry_1000, ignore.case = TRUE))]
```

```
## [1] "don’t" "won’t"
```

```r
a <- gsub("[^[:alpha:]]", "", sw_fry_1000)
head(a)
```

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

```r
a <- sapply(a, tolower)
```

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

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

```r
sum(grepl("ar", a))
```

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

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

```r
# Create a regular expression to match the letter "l" at the beginning of the word
l_regex <- "^l"

# Create a regular expression to match the letter "r" at the end of the word
r_regex <- "r$"

indices <- which(grepl(l_regex, a) & grepl(r_regex, a))

print(length(a[indices[1]]))
```

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

```r
print(a[indices[1]])
```

```
##   letter
## "letter"
```

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

```r
a[which(grepl("^col", a) | grepl("eck$", a))]
```

```
##     color       cold      check    collect     colony     column       neck
##   "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.

```r
print(a[which(grepl('[bcdfghjklmnpqrstvwxyz]{4,}', a))])
```

```
##      country       system     syllable       length   instrument     industry
##    "country"     "system"   "syllable"     "length" "instrument"   "industry"
##       symbol       supply
##     "symbol"     "supply"
```

```r
print(length(which(grepl('[bcdfghjklmnpqrstvwxyz]{4,}', a))))
```

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

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

```r
grep("\\b\\w*q(?!ui)\\w*\\b", a, value = TRUE, perl = TRUE)
```

```
##   question     equate     square      equal      quart   quotient
## "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.

```r
k_words <- grep("\\b\\w*k\\w+\\b", a, value=TRUE)
k_words
```

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

```r
first_chars <- character(0)

for (word in k_words) {
  match <- regexpr("k.", word)
  if (match != -1) {
    first_char <- substr(word, match + 1, match + 1)
    first_chars <- c(first_chars, first_char)
  }
}
```

```r
# Use the table function to count the occurrences of each unique first character
char_count <- table(first_chars)

# Print the character count
print(char_count)
```

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

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

```r
remove_vowels <- function(string) {
  return(gsub("[aeiouAEIOU]", "", string, ignore.case = TRUE))
}
no_vowels <- lapply(a, remove_vowels)
length(unique(no_vowels))
```

```
## [1] 741
```

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.

```r
url <- "https://github.com/couthcommander/Bios6301/raw/master/datasets/haart.csv"
url <- "https://github.com/couthcommander/Bios6301/blob/main/datasets/haart.csv"

haart_df <- read.csv(url)[,c('death','weight','hemoglobin','cd4baseline')]
```

```
## Warning in read.table(file = file, header = header, sep = sep,
## quote = quote, : incomplete final line found by readTableHeader on
## 'https://github.com/couthcommander/Bios6301/blob/main/datasets/haart.csv'
```

```r
# the urls weren't loading so I used a copy of the dataset that I already had downloaded

haart_df <- read.csv('/Users/jonathanlifferth/R_projects/Bios6301/datasets/haart.csv')[,c('death','weig

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.

```
# debug(myfun(dat=haart_df, response=death))
```

Debugging revealed the following error: "Error in eval(predvars, data, env) : object 'death' not found" Meaning that the death column in haart_df is not accessible in the scope of this function.

5 bonus points

Create a working function.

eval() can't find death in the current scope/environment

```
myfun2 <- function(dat, response) {
  form <- paste0(substitute(response), " ~.")
  coef(summary(glm(form, data=dat, family=binomial(logit))))
}
myfun2(dat=haart_df, response=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
```