BIOS 6301: Assignment 7

Charles Rhea

38/40 for some issues in code for first question

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).

```
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`.

```
x2 <- 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
```

```
x3 <- x2
x3$tdiff <- unlist(tapply(x3$dt, INDEX = x3$id,
                          FUN = function(x) c(0, `units<-`(diff(x), "days"))))
x3$tdiff_years <- as.numeric(floor((x3$tdiff / 365.4)))

#Subset by rows with 1 and 2 missing years
one_miss <- subset(x3, c(x3$tdiff_years == 1))
two_miss <- subset(x3, c(x3$tdiff_years == 2))

#Adding 1 additional row to one_miss subset
one_miss_2 <- one_miss[rep(1:nrow(one_miss), (one_miss$tdiff_years == 1) +1), ]
one_miss_2$a1c[duplicated(one_miss_2$a1c)] <- NA
one_miss_2$dt[duplicated(one_miss_2$dt)] = one_miss_2$dt[duplicated(one_miss_2$dt)] - years(1)

#Adding 2 additional rows to two_miss subset
two_miss_2 <- two_miss[rep(1:nrow(two_miss), (two_miss$tdiff_years == 2) +2), ]
two_miss_2$a1c[duplicated(two_miss_2$a1c)] <- NA
two_miss_2$dt[duplicated(two_miss_2$dt)] = two_miss_2$dt[duplicated(two_miss_2$dt)] - years(1)
two_miss_2$dt[duplicated(two_miss_2$dt)] = two_miss_2$dt[duplicated(two_miss_2$dt)] - years(1)

#Binding subsets back into the original dataset
bind1 <- rbind(x3, one_miss_2)
bind2 <- rbind(bind1, two_miss_2)

x4 <- bind2[order(bind2$id, bind2$dt), ]
```

you have 40 extra rows, i am guessing you may have double counted the ones with 2 years missing
the rest of your numbers are a bit off and i think it is because of this

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.

```
library(plyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag

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

```
x4$number <- 1
x4 <- x4 %>%
  group_by(id) %>%
  mutate(visit = cumsum(number))
x4 <- x4[-c(6)]
head(x4)
```

```
## # A tibble: 6 x 6
## # Groups:   id [1]
##      id dt                   a1c tdiff tdiff_years visit
##   <int> <dttm>             <dbl> <dbl>       <dbl> <dbl>
## 1     1 2001-05-08 16:22:52  7.31   0             0     1
## 2     1 2001-06-17 22:42:23  8.31  40.3           0     2
## 3     1 2001-08-17 16:51:46  6.55  60.8           0     3
## 4     1 2001-12-14 14:50:29  5.99 119.            0     4
## 5     1 2002-08-19 13:51:47  6.01 248.            0     5
## 6     1 2003-03-22 03:51:36  7.24 215.            0     6
```

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

```
impute.mean <- function(x) replace(x, is.na(x), mean(x, na.rm = TRUE))
x_final <- ddply(x4, ~ id, transform, a1c = impute.mean(a1c))
x_final <- x_final[order(x_final$id, x_final$dt),]
head(x_final)
```

```
##   id                  dt      a1c     tdiff tdiff_years visit
## 1  1 2001-05-08 16:22:52 7.309995   0.00000           0     1
## 2  1 2001-06-17 22:42:23 8.310721  40.26355           0     2
## 3  1 2001-08-17 16:51:46 6.548845  60.75652           0     3
## 4  1 2001-12-14 14:50:29 5.985275 118.95744           0     4
## 5  1 2002-08-19 13:51:47 6.011547 247.91757           0     5
## 6  1 2003-03-22 03:51:36 7.243858 214.62487           0     6
```

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

```
group_mean <- aggregate(x = x_final$a1c, by=list(x_final$id), FUN=mean)
print(group_mean)
```

```
##    Group.1        x
## 1        1 6.654444
## 2        2 9.723766
## 3        3 7.152055
## 4        4 8.317773
```

```
## 5        5  9.429694
## 6        6  7.161733
## 7        7  7.878521
## 8        8  6.472878
## 9        9  4.475878
## 10      10  6.111711
## 11      11  4.838279
## 12      12  6.634796
## 13      13  8.565237
## 14      14  9.122968
## 15      15  6.737092
## 16      16  7.420245
## 17      17  6.632733
## 18      18  6.181420
## 19      19  8.628037
## 20      20  8.923518
## 21      21  5.444430
## 22      22  5.763931
## 23      23  6.476453
## 24      24  9.377525
## 25      25  5.047863
## 26      26  8.671435
## 27      27  7.581578
## 28      28  4.243469
## 29      29  6.345254
## 30      30  4.135795
## 31      31  8.700384
## 32      32  5.175777
## 33      33  6.528153
## 34      34  8.444519
## 35      35  3.884199
## 36      36  9.568271
## 37      37  8.541850
## 38      38 10.160773
## 39      39  8.965639
## 40      40  7.607973
## 41      41  3.804325
## 42      42  6.891402
## 43      43  5.699373
## 44      44  5.523107
## 45      45  8.728996
## 46      46  7.500455
## 47      47  4.835777
## 48      48  7.674268
## 49      49  5.480910
## 50      50  5.145822
```

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

```
total_visits <- aggregate(x = x_final$visit, by=list(x_final$id), FUN=length)
print(total_visits)
```

```
##    Group.1  x
```

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

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

```
print(x_final[x_final$id ==15, ])
```

```
##     id                  dt      a1c      tdiff tdiff_years visit
## 170 15 2000-10-21 01:08:17 7.401322   0.000000           0     1
## 171 15 2001-08-08 14:23:08 5.896318 291.551979           0     2
## 172 15 2001-08-15 07:03:29 7.457722   6.694687           0     3
## 173 15 2002-03-15 21:23:10 5.330917 212.638669           0     4
## 174 15 2002-04-14 09:08:25 6.484003  29.448090           0     5
## 175 15 2002-10-10 18:27:43 8.139101 179.388403           0     6
## 176 15 2003-02-19 12:58:53 6.446557 131.813310           0     7
## 177 15 2003-03-02 06:58:10 7.432291  10.749502           0     8
## 178 15 2003-06-30 07:20:49 7.113792 119.974063           0     9
## 179 15 2004-01-22 20:30:42 5.668897 206.590197           0    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')
```

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

```
a <- tolower(sub('[^a-zA-Z]', '', sw_fry_1000))
head(a)
```

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

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

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

64 words contain the string "ar".

```
length(unique(grep('ar', a)))
```

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

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

Using the 'grep' command, we find that "letter" meets these criteria.

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

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

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

Using the 'grep' command, we find the words "color", "cold", "check", "collect", "colony", "column", and "neck" meets these criteria.

```
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.

There are 8 words in this vector which contain 4 or more adjacent consonants

```
length(unique(grep("[^aeiou]{4,}", a)))
```

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

```
#a[c(206, 467, 529, 537, 575, 642, 708, 785)]
```

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

Using the 'grep' command, we find the words question, equate, square, equal, quart, and quotient meet these criteria

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

Using the 'grep' command, we find the words like, make, know, take, kind, keep, knew, king, sky, kept, broke, kill, lake, key, skin, spoke, skill, and market meet these criteria

```
grep("[k][a-z]", a, value = TRUE)
```

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

We extracted the letter after 'k' within each of these 10 words, then ran the 'table' function on this vector. We found there are three unique characters: e (4 times), i (4 times), and n (2 times).

```
a2 <- grep("[k][a-z]", a, value = TRUE)
a2 <- sub("^.*[k](.).*$", '\\1', a2)
table(a2)
```

```
## a2
##  e  i  n  y
## 10  5  2  1
```

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

There are 581 character strings that are found once after removing the vowels

```
a3 <- gsub("[aeiou]", "", a)
length(which(table(a3) == 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.

```
#Note - I could not get this URL to work/connect to the dataset; will import the dataset using 'read_cs
#url <- "https://github.com/couthcommander/Bios6301/raw/master/datasets/haart.csv"
library(readr)
haart.df <- read_csv("~/Desktop/BIOS 6301 - Introduction to Statistical Computing/datasets/haart.csv")
```

```
## Rows: 1000 Columns: 12
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): init.reg, init.date, last.visit, date.death
## dbl (8): male, age, aids, cd4baseline, logvl, weight, hemoglobin, death
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
haart_df2 <- haart.df[,c('death','weight','hemoglobin','cd4baseline')]
coef(summary(glm(death ~ ., data=haart_df2, 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_df2, death), error = function(e) e)
```

```
## <simpleError in eval(expr, envir, enclos): object 'death' not found>
```

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

After evaluating, the issue appears to lies within the "form <- as.formula(response ~ .)" line of code. This is where we are establishing the first element that will go in the 'glm' function, the dependent variable - for example, in the first part of this problem this was "death". However, within this line of code, we do not direct R to a data source for this variable. Again, if 'response' = 'death' in this line, there is not a way for us to direct R to know these 'death' is coming from. As a result, when we run this function, it will return an error stating that the 'response' variable assigned cannot be found.

```
#debug(myfun)
#myfun()
```

5 bonus points

Create a working function.