Bios 6301: Assignment 7

Haley Yaremych

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`.

```
ord = 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.

```
for (i in 1:length(unique(ord$id))){
  idx = ord[which(ord$id==i),]
  # differences in seconds
  diffs = diff(as.numeric(idx$dt))
  # differences in days
  diffdays = diffs/86400

  which = which(diffdays>365)


  # couldn't figure this one out, sorry!


}
```

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.

```
ord$visit = numeric(dim(ord)[1])
for (i in 1:length(unique(ord$id))){
  n = nrow(ord[which(ord$id==i),])
  ord[which(ord$id==i),"visit"] = seq(n)
}
```

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

```
for (i in 1:length(unique(ord$id))){
  meana1c = mean(ord[which(ord$id==i),"a1c"], na.rm=TRUE)

  ord[which(ord$id==i),is.na("a1c")] = meana1c
}
```

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

```
tapply(ord$a1c, ord$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(ord$visit, ord$id, max)
```

```
##  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 15 11  8 14  9  5 10 14  7 12 11  8 12 10  8  8 12 10 11 13 12  9 12 15 10
## 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
##  9 15  3 13 10  8 12 10 10  8  6 14 13  9 14 10  6 11  5 11  9  4 10  7
```

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

```
ord[which(ord$id==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.

```
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`.

```
for (i in 1:1000){
  sw_fry_1000[i] = tolower(sw_fry_1000[i])
  sw_fry_1000[i] = gsub("[^a-z]", "", sw_fry_1000[i])
}
a = 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.{4}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("[^aeiou]{4}", a))
```

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

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

```
grep("q[u?][^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.

```
lst = grep("k[a-z]{1}", a, value=TRUE)

new = numeric(length(lst))
for (i in 1:length(lst)){

  letters = strsplit(lst[i], split="")
  idx = grep("k", letters)
  new[i] = letters[[1]][idx+1]
}

table(new)
```

```
## new
## a e i k n p r
## 4 3 4 3 2 1 1
```

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

```
a_new = numeric(length(a))
for (i in 1:length(a)){
  a_new[i] = gsub("[aeiou]", "", a[i])
}
length(unique(a_new))
```

the length of the uniques is not really the same as the length of those that occur only once, 1/3

```
## [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.

```
url <- "https://github.com/couthcommander/Bios6301/blob/main/datasets/haart.csv"
tryCatch(haart_df <- read.csv(url)[,c('death','weight','hemoglobin','cd4baseline')], error=function(e) c
```

```
## <simpleError in `[.data.frame`(read.csv(url), , c("death", "weight", "hemoglobin",     "cd4baseline"]
```

```
haart_df = read.csv("~/Documents/Fall 2021/Statistical Computing/datasets/haart.csv")
coef(summary(glm(death ~ ., data=haart_df, family=binomial(logit))))
```

```
##                  Estimate  Std. Error       z value  Pr(>|z|)
## (Intercept)  2.356607e+01 1945929.490  1.211044e-05 0.9999903
## male         3.236768e-08  661616.837  4.892210e-14 1.0000000
## age         -6.038576e-10   57363.578 -1.052685e-14 1.0000000
## cd4baseline -5.391171e-11    4290.562 -1.256519e-14 1.0000000
```

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.

(Note: I had to read in the data set from my local machine because the URL method was not working.)

When I run `debug()` on `myfun`, I see that, although we specify that `response = death`, when `form` is created, it is created as `response ~ .` rather than `death ~ .` which makes me think there is an issue in how the formula is created. When I try to probe `response` it tells me `object 'death' not found`. Perhaps when we specify the response variable, we need to specify that it's a column of the data set, not some global object.

5 bonus points

Create a working function.

‘‘‘