Your submitted pdf didnt have
most of your code, why?

40

Bios 6301: Assignment 7

Lydia Yao

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).

```
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.0.5

##
## Attaching package: 'lubridate'

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

```
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.

```
x1 <- x[order(x$id,x$dt),]
```

as.POSIXct(x1[i, “dt”] + years(1),origin=‘2000-01-01’)

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.

```
x1 <- data.frame(x1)
for(i in 1:nrow(x1)){
  if(x1[i,"id"] == x1[i+1,"id"]){
    if (difftime(x1[i+1, "dt"], x1[i, "dt"], "days") >= 365){

        iter = as.integer(difftime(x1[i+1, "dt"], x1[i, "dt"], "days")/365)
        for(j in 1:iter){
          x1[nrow(x1) + 1,1] = x1[i, "id"]
          x1[nrow(x1),2] = as.POSIXct(as.POSIXlt(x1[i, "dt"] + dyears(j)),origin='2000-01-01')
        }
    }else {next}
  } else{ next }
}
x1 <- x1[order(x1$id,x1$dt),]
```

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.

```
ix <- c(1,sapply(which(diff(x1$id) == 1), function(x) x+1), nrow(x1)+1)
count = 1
for(i in 1:nrow(x1)){
  num = ix[count+1]-ix[count]
  x1[i,"visit"] = num
  if(ix[count+1] == i+1 || i == nrow(x1)){
    count = count + 1
  }
}
```

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

```
for(i in 1:nrow(x1)){
  if(is.na(x1[i, "a1c"])){
    id = x1[i, "id"]
    lower_ix = ix[id]
    upper_ix = ix[id + 1] - 1
    m = mean(x1$a1c[lower_ix:upper_ix], na.rm = TRUE)
    x1[i, "a1c"] = m
  }
}
```

5. Print mean a1c for each id.

```
for(i in 1:(length(ix) -1)){
  lower_ix = ix[i]
  upper_ix = ix[i + 1] - 1
  m = mean(x1$a1c[lower_ix:upper_ix], na.rm = TRUE)
  print(paste("id", i, "has mean", m))
}
```

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

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

```r
for(i in 1:(length(ix) -1)){
  lower_ix = ix[i]
  m = x1[lower_ix, "visit"]
  print(paste("id", i, "has", m, "visits"))
}
```

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

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

```
lower_ix = ix[15]
upper_ix = ix[16] -1
x1[lower_ix:upper_ix,]
```

```
##     id                  dt      a1c visit
## 300 15 2000-10-21 01:08:17 7.401322    10
## 127 15 2001-08-08 14:23:08 5.896318    10
## 165 15 2001-08-15 07:03:29 7.457722    10
## 109 15 2002-03-15 21:23:10 5.330917    10
## 319 15 2002-04-14 09:08:25 6.484003    10
## 255 15 2002-10-10 18:27:43 8.139101    10
## 224 15 2003-02-19 12:58:53 6.446557    10
## 481 15 2003-03-02 06:58:10 7.432291    10
## 425 15 2003-06-30 07:20:49 7.113792    10
## 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`.

```
q1 <- grep("[A-Za-z]", sw_fry_1000, value=TRUE)
a <- sapply(q1, tolower)
```

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

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

```
length(a[grepl("ar", a) == TRUE])
```

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

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

```
grep("l.{4}r", a, value=TRUE)
```

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

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

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

```
##     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
length(grep("[^aeiou]{4,}", a, value=TRUE))
```

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

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

```r
grep("q(?!ui)", 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
q7 <- grep("k[a-z]", a, value=TRUE)
k_ix <- sapply(q7, function(x) unlist(gregexpr(pattern ='k',x)) + 1)
result = c(NULL)
for(i in 1:length(k_ix)){
  result <- append(result, substring(names(k_ix)[i], k_ix[i], k_ix[i]))
}
table(result)
```

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

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

```r
q8 <-table(gsub("[aeiou]", "", a))
length(q8[q8 == 1])
```

```
## [1] 582
```

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://raw.githubusercontent.com/couthcommander/Bios6301/main/datasets/haart.csv"
haart_df <- read.csv(url)
haart_df <- read.csv(url)[,c('death','weight','hemoglobin','cd4baseline')]
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) {
  others <- colnames(dat)[!grepl(response, colnames(dat))]
  others <- paste(others, collapse = " + ")
  form <- paste(response, "~", others)
  form <- as.formula(paste(response, "~", others))
  coef(summary(glm(form, data=dat, family=binomial(logit))))
}
```

```
oldfun <- 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(oldfun(haart_df, "death"), error = function(e) e)
```

```
## <simpleError in model.frame.default(formula = form, data = dat, drop.unused.levels = TRUE): variable
```

```
debugonce(oldfun)
```

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

Because we have not defined 'death', it is not recognized in the function. The debugger stops is seen to stop on the second line when trying to use form in the glm. To fix this, I make sure that `as.formula()` gets a string input. The input for response should also be a string.

5 bonus points

Create a working function.