homework7

Haoyang Yi

10/28/2022

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`.

```
x = arrange(x,id,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 unique(x$id)){
  for (j in 2:length(x$dt[x$id==i])){
    dtnow = x$dt[x$id==i][j]
    dtbefore = x$dt[x$id==i][j-1]
    ytdiff = as.numeric(dtnow-dtbefore)%/%365
    while (ytdiff>0){
      ytdiff = ytdiff-1
      newrow = data.frame(id=i,dt=dtbefore+years(1),a1c=NA)
      x = rbind(x,newrow)
    }
  }
```

```
}
x = arrange(x,id,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.

```
x = x %>%
  group_by(id) %>%
  mutate(visit = 1:length(id))
```

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

```
x = x %>%
  group_by(id) %>%
  mutate(a1c = replace_na(a1c,mean(a1c,na.rm = T)))
```

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

```
x %>%
  group_by(id) %>%
  summarize(mean_a1c=mean(a1c))
```

```
## # A tibble: 50 x 2
##       id mean_a1c
##    <int>    <dbl>
##  1     1     6.65
##  2     2     9.79
##  3     3     6.95
##  4     4     8.19
##  5     5     9.43
##  6     6     7.13
##  7     7     7.88
##  8     8     6.24
##  9     9     4.42
## 10    10     6.03
## # ... with 40 more rows
```

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

```
x %>%
  group_by(id) %>%
  summarize(visits=n())
```

```
## # A tibble: 50 x 2
##       id visits
##    <int>  <int>
##  1     1      7
##  2     2     16
##  3     3     13
##  4     4      9
##  5     5     14
##  6     6     11
##  7     7      7
##  8     8     12
##  9     9     15
## 10    10      8
```

```
## # ... with 40 more rows
```

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

```
x[x$id==15,]
```

```
## # A tibble: 10 x 4
## # Groups:   id [1]
##       id dt                   a1c visit
##    <int> <dttm>               <dbl> <int>
##  1    15 2000-10-21 01:08:17  7.40     1
##  2    15 2001-08-08 14:23:08  5.90     2
##  3    15 2001-08-15 07:03:29  7.46     3
##  4    15 2002-03-15 21:23:10  5.33     4
##  5    15 2002-04-14 09:08:25  6.48     5
##  6    15 2002-10-10 18:27:43  8.14     6
##  7    15 2003-02-19 12:58:53  6.45     7
##  8    15 2003-03-02 06:58:10  7.43     8
##  9    15 2003-06-30 07:20:49  7.11     9
## 10    15 2004-01-22 20:30:42  5.67    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`.

```
a = tolower(gsub("[^A-Za-z ]", "", sw_fry_1000))
```

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

2. How many words contain the string "ar"? 64

```
length(grep("ar",a))
```

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

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

```
for (i in unique(a)){
if (nchar(i)==6 &startsWith(i,"l")&endsWith(i,"r"))res = i}
res
```

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

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

```
res = NULL
for (i in unique(a)){
if (startsWith(i,"col")|endsWith(i,"eck"))res = append(res,i)
}
res
```

```
## [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

```
str_subset(a, "[^aeiou]{4}")
```

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

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

```
str_subset(a, "q[u][^i]") # q must be followed by a u
```

```
## [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.

```
a1 = str_subset(a,"k[^k]")
a2=str_split(a1,"k")
res = NULL
for (i in 1:length(a2)){
res = c(res,a2[[i]][2])
}
table(substr(res,1,1))
```

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

8. Remove all vowels. How many character strings are found exactly once? There are 581 strings found exactly once.

```
a3= gsub("[aeiou]", "", a)
t = table(a3)
count = 0
for (i in 1:length(t)){
if (as.numeric(t[i])==1) count = count+1}
count
```

```
## [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.

```
haart_df = read.csv('haart.csv')[,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:

```
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.

The input of the function response cannot be directly assigned as death because it's not a valid value.

5 bonus points

Create a working function.

```
myfun <- function(dat, response) {
  y = dat[,response]
  other = dat[,!(colnames(dat) %in% response)]
  coef(summary(glm(y~other[,1]+other[,2]+other[,3], family=binomial(logit))))
}
myfun(haart_df, "death")
```

```
##                 Estimate  Std. Error   z value     Pr(>|z|)
## (Intercept)  3.576411744 1.226870535  2.915069 0.0035561039
## other[, 1]  -0.046210552 0.022556001 -2.048703 0.0404911395
## other[, 2]  -0.350642786 0.105064078 -3.337418 0.0008456055
## other[, 3]   0.002092582 0.001811959  1.154872 0.2481427160
```