Explaining the valag Functions

Mark White | markhwhiteii@gmail.com

2017 May 4th

I will discuss the `valag` ("Visit-Arrival Lag") and `valag1` functions in this document. The functions can be loaded by running the `valag.R` document, and the `valagGuide.pdf` file explains how to operate it. In case the CDC changes variable names, connections change, people are interested in doing something similar, etc., I will explain line-by-line what the code means. I will first present a segment of code and then explain it below. The code could probably be written computationally more efficiently, but I wanted it to be readable in case anyone needed to change the code later.

The `valag` function

```
valag <- function(channel, begin, end, table, location="KS") {
```

This first line simply defines `valag` as a function that has the arguments `channel`, `begin`, `end`, `table`, and `location` (which defaults to `KS`).

```
  require(dplyr)
```

This loads the `dplyr` package, in case the user has not done so already. If the user does not have it installed, the console will tell them that they need to install it.

Getting the data

```
  query <- paste0("SELECT C_Biosense_Facility_ID, C_BioSense_ID,
C_Visit_Date_Time,
                  Arrived_Date_Time, Chief_Complaint_Text,
Admit_Reason_Description
                  FROM ", location, "_", table, "_Processed
                  WHERE C_Visit_Date_Time >= '", begin, "' AND
C_Visit_Date_Time <= '", end, "'")
```

This creates a SQL query. The function is selecting six variables. Where from? It depends on what the user specified. It always is going to take it from a "Processed" table, but which table depends on the `location` and `table` arguments. The `paste0` function basically concatenates all of these elements, with no separation between them. The date range is specified by the visit, and it is larger than or equal to the `begin` argument and smaller than or equal to the `end` argument.

If I wanted to look at the Kansas (`KS`) production table (`PR`) for May 1st, 2017 (`begin` = "2017-05-01 00:00:00", `end` = "2017-05-01 23:59:59"), then the SQL query would look like:

```
"SELECT C_Biosense_Facility_ID, C_BioSense_ID, C_Visit_Date_Time,
        Arrived_Date_Time, Chief_Complaint_Text, Admit_Reason_Description
FROM KS_PR_Processed
```

```
WHERE C_Visit_Date_Time >= '2017-05-01 00:00:00' AND C_Visit_Date_Time <=
'2017-05-01 23:59:59'"
```

Now onto the next `R` code chunk:

```
  dat <- sqlQuery(channel, query)
```

This retrieves the data. It calls on the user's connection (`channel`) and runs the SQL query (`query`) that was specified in the code above. It places it into a `data.frame` called `dat`.

```
  names <- sqlQuery(channel, paste0("SELECT C_Biosense_Facility_ID,
Facility_Name
                                    FROM ", location, "_MFT"))
  names$C_Biosense_Facility_ID <- as.factor(names$C_Biosense_Facility_ID)
```

We want the facilities' names, not just their C_Biosense_Facility_ID numbers. Unfortunatley, the XX_PR_PRocessed table does not include the name. So we have to fetch it from the Master Facilities Table ("MFT"). Much like the SQL query above, the first line pulls down just the C_Biosense_Facility_ID numbers and the Facility_Name for all facilities in a given `location` specified by the user (again, this defaults to `KS`).

The second line in this chunk makes the list of C_Biosense_Facility_ID numbers a `factor`. This will come in handy later when we try to join a few tables.

Calculating lag for first message ("fm") overall

```
  datfm <- slice(group_by(dat, C_BioSense_ID), which.min(Arrived_Date_Time))
```

The `group_by` function is simply taking the data frame we've already made, `dat`, and grouping it by the ID that is unique to a patient's visit, `C_BioSense_ID`. I take this grouped data and `slice` it, which will extract rows within a group. What rows do I want? I want the row in which the *very first* message arrived. The `which.min` function here will give me the row number for the arrived time that has the earliest date. Since it is using grouped data, I will be extracting the row that has the earliest arrival time for each unique visit. Here is a small demonstration of how that works:

```
demo <- data.frame(group=factor(c("A","A","A","B","B","B")), x=c(1:6))
demo

##   group x
## 1     A 1
## 2     A 2
## 3     A 3
## 4     B 4
## 5     B 5
## 6     B 6

group_by(demo, group)

## Source: local data frame [6 x 2]
## Groups: group [2]
##
```

```
##    group     x
##   <fctr> <int>
## 1      A     1
## 2      A     2
## 3      A     3
## 4      B     4
## 5      B     5
## 6      B     6

slice(group_by(demo, group), which.min(x))

## Source: local data frame [2 x 2]
## Groups: group [2]
##
##    group     x
##   <fctr> <int>
## 1      A     1
## 2      B     4
```

And onto the next `R` chunk in the function...

```
  datfm$lag <- as.numeric(difftime(datfm$Arrived_Date_Time,
datfm$C_Visit_Date_Time, units="hours"))
```

This line creates a new variable, `lag`, in the data frame for first message, `datfm`. This variable is the difference between the arrival and visit times (remember: now we are only including cases that represent the *first* message overall for a patient's visit). This uses the `difftime` function and specifies to always return the number of hours. This is created as a numeric variable.

```
  outputfm <- data.frame(C_Biosense_Facility_ID=
                           rownames(data.frame(Hours=with(datfm, tapply(lag,
C_Biosense_Facility_ID, mean, na.rm=TRUE)))),
                         First_Message=
                           as.numeric(round(with(datfm, tapply(lag,
C_Biosense_Facility_ID, mean, na.rm=TRUE)),2)), row.names=NULL)
```

This chunk makes up the first part of the output we will be displaying to the user; it creates a new data frame that has two variables: C_Biosense_Facility_ID and First_Message. Both variables are using the `tapply` function. What this does it applies a function (in this case, `mean`) to a variable (in this case, `lag`), grouped by another variable (in this case, `C_Biosense_Facility_ID`). When the result of this `tapply` is made into a `data.frame`, then the groups (in this case, `C_Biosense_Facility_ID`) become the row names. All I'm doing to create `outputfm$C_Biosense_Facility_ID` is taking the row names from this output and making them their own row. This will tell users what the ID is for each First_Message. This will come in handy later when it comes to joining the different outputs.

`outputfm$First_Message` simply takes the reslt of this `tapply` as a vector of numeric values (and I round them to two decimal points). I tell it `row.names=NULL` so that the Facility ID numbers are not repeated.

```
  outputfm$C_Biosense_Facility_ID <- factor(outputfm$C_Biosense_Facility_ID,
levels=levels(names$C_Biosense_Facility_ID))
```

The last thing that I'm doing here is re-factoring the ID numbers to specify that the levels for this factor can include *all* facility ID numbers, not just the ones that are listed here. R will be fussy later on if you try to merge variables that have different factors.

Calculating lag for first chief complaint or admit reason (ccar) message

```
  datccar <- dat[-which(is.na(dat$Chief_Complaint_Text)==TRUE &
is.na(dat$Admit_Reason_Description)==TRUE),]
```

For this section, I only want to look at messages that include *either* a chief complaint *or* an admit reason. This first line of code simply removes any rows that have an NA value for *both* of these fields.

```
  datccar <- slice(group_by(datccar, C_BioSense_ID),
which.min(Arrived_Date_Time))
  datccar$lag <- as.numeric(difftime(datccar$Arrived_Date_Time,
datccar$C_Visit_Date_Time, units="hours"))
  outputccar <- data.frame(C_Biosense_Facility_ID=
                             rownames(data.frame(Hours=with(datccar,
tapply(lag, C_Biosense_Facility_ID, mean, na.rm=TRUE)))),
                           CC_or_AR=
                             as.numeric(round(with(datccar, tapply(lag,
C_Biosense_Facility_ID, mean, na.rm=TRUE)),2)), row.names=NULL)
  outputccar$C_Biosense_Facility_ID <-
factor(outputccar$C_Biosense_Facility_ID,
levels=levels(names$C_Biosense_Facility_ID))
```

All of this code is the same as above, except now it is only performing it on messages that have a chief complaint *or* admit reason.

```
  output <- left_join(outputfm, outputccar, by="C_Biosense_Facility_ID")
```

This code takes the outputfm object and adds the outputccar object to it, matching the two by C_Biosense_Facility_ID. If the time frame is small enough, a facility might not report any chief complaints or admit reasons yet. This will return an NA in the "CC_or_AR" column of the output.

```
  output <- right_join(names, output, by="C_Biosense_Facility_ID")
```

This last part takes the output and attaches the names of the facilities that we retrieved earlier. If the facility name is not found in the output object, then it is not included in the output.

```
  cat("Average hours between visit and (a) arrival of first message
(First_Message)
and (b) arrival of first message with chief complaint or admit reason
(CC_or_AR), by facility.
Visits between:", begin, "and", end, "(inclusive).\n\n")
```

```
  print(output)
}
```

This prints the final output to the user.

The `valag1` function

I will discuss this function more briefly, since it is derivative of the `valag1` function.

```
valag1 <- function(channel, begin, end, table, facility, location="KS") {
  require(dplyr)
```

These lines accomplish the same purpose as described above for `valag`.

```
  query <- paste0("SELECT C_BioSense_ID, C_Visit_Date_Time,
Arrived_Date_Time, Chief_Complaint_Text, Admit_Reason_Description
                   FROM ", location, "_", table, "_Processed
                   WHERE C_Visit_Date_Time >= '", begin, "' AND
C_Visit_Date_Time <= '", end, "' AND C_Biosense_Facility_ID=", facility)
  dat <- sqlQuery(channel, query)
  name <- sqlQuery(channel, paste0("SELECT Facility_Name
                                    FROM ", location, "_MFT
                                    WHERE C_Biosense_Facility_ID=",
facility))
```

These lines are similar to those above, except now we are limiting the data we are retrieving to just one facility.

```
  datfm <- slice(group_by(dat, C_BioSense_ID), which.min(Arrived_Date_Time))
  datfm$lag <- as.numeric(difftime(datfm$Arrived_Date_Time,
datfm$C_Visit_Date_Time, units="hours"))
```

To get the first message overall, we group the data by patient's visit (i.e., C_BioSense_ID) and then take only the row (i.e., message) with the earliest time for this visit. Just like above, we then calculate the difference between arrived and visit times.

```
  datccar <- dat[-which(is.na(dat$Chief_Complaint_Text)==TRUE &
is.na(dat$Admit_Reason_Description)==TRUE),]
  datccar <- slice(group_by(dat, C_BioSense_ID),
which.min(Arrived_Date_Time))
  datccar$lag <- as.numeric(difftime(datccar$Arrived_Date_Time,
datccar$C_Visit_Date_Time, units="hours"))
```

To get the first message that had a chief complaint or admit reason, we again first limit the data by eliminating any messages that included neither a chief complaint nor an admit reason. Then we cut the data down to only rows that had the earliest date and time for each message and perform the time difference on those rows.

```
  if (is.nan(mean(datfm$lag))==TRUE) {
    cat("There were no visits to this facility in this time frame.")
```

I found that, if one is looking at too small of a time frame, a facility might have had no visits within that time point. If this is the case, the mean lag time will be `NaN`. I have written the code so that if this is the case, the function tells the user that no visits happened in that facility in that time frame. Otherwise...

```
      } else {
    cat(as.character(name[1,1]),
"\nC_Biosense_Facility_ID:", facility,
"\nVisits between:", begin, "and", end, "(inclusive)
The average time between visit and first message arrival:",
round(mean(datfm$lag),2), "hours
The average time between visit and first message with chief complaint or
admit reason:", round(mean(datccar$lag),2), "hours")
  }
}
```

The name (`as.character(name[1,1])`) is returned to the user, as well as the facility number, the start and end date, as well as the two mean lag times, in hours, rounded to the second decimal point.