Explaining the rc Functions

Mark White | markhwhiteii@gmail.com

5/8/2017

I will discuss the `rc` ("Records Count") functions for feeds (`feedrc`) and facilities (`facilityrc`) in this document. The functions can be loaded by running the `rc.R` document, and the `rcGuide.pdf` file explains how to operate it. In case the CDC changes variable names, connections change, people are interested in doing something different with the code, etc., I will explain here line-by-line what the code means. I will first present a segment of code and then explain it below.

The `feedrc` function

```
feedrc <- function(channel, visit, arrived, month, year) {
```

This first line names the function `feedrc` and specifies the five different arguments that it will take (see `rcGuide.pdf` for a description of these arguments).

```
  suppressPackageStartupMessages(require(lubridate))
  suppressPackageStartupMessages(require(data.table))
  suppressPackageStartupMessages(require(dplyr))
  suppressPackageStartupMessages(require(Hmisc))
```

These lines load the required packages if they are not already loaded. Some packages give a bunch of annoying messages when they are loaded, so I suppressed these for this function.

```
  if (visit==TRUE) {
```

This code chunk will only be evaluated if the user specifies they want to get records for patient visits during that month. If not, it skips to the next `if` statement.

```
    dv <- sqlQuery(channel, paste0("SELECT C_Visit_Date_Time, Feed_Name
                                   FROM KS_PR_PRocessed
                                   WHERE MONTH(C_Visit_Date_Time) = ", month,
" AND YEAR(C_Visit_Date_Time) = ", year))
```

This code chunk generates a SQL query and pastes in the `month` and `year` that the user has specified, using the `RODBC` connection specified with the `channel` argument. This data is placed into a data frame called `dv` (for "data visits").

```
    dv$Visit_Date <- gsub(" UTC", "",
floor_date(ymd_hms(dv$C_Visit_Date_Time), unit="day"))
```

When we pull the date down from the database, it includes hours, minutes, and seconds. We want to generalize this to the day, and this is what this line does. It takes the variable—`dv$C_Visit_Date_Time`—and tells R that it is in the format of `ymd_hms` (year, month, day, hours, minutes, seconds). I use the `floor_date()` function to round all times down to the

nearest `unit="day"`. This function will automatically add a time zone to the end. I do not want this time zone information, so I simply delete it using the `gsub()` function. This function says I want to take the pattern " UTC" and replace it with nothing (i.e, "").

```
    outv <- setDT(as.data.frame.matrix(table(dv$Visit_Date, dv$Feed_Name)),
keep.rownames = TRUE)
  }
```

Here, I take a frequency table of visit dates (`dv$Visit_Date`) and feeds (`dv$Feed_Name`), I save it as a data frame matrix, and then I set it as a data table, with the row names (dates) now stored as the first variable. It saves this into a new data frame, called `outv` (which stands for "output visits"). The last curly bracket means that I have closed the `if` statement. If the user did not specify `visit` as `TRUE`, then it has skipped over this whole thing.

```
  if (arrived==TRUE) {
    da <- sqlQuery(channel, paste0("SELECT Arrived_Date_Time, Feed_Name
                                   FROM KS_PR_PRocessed
                                   WHERE MONTH(Arrived_Date_Time) = ", month,
" AND YEAR(Arrived_Date_Time) = ", year))
    da$Arrived_Date <- gsub(" UTC", "",
floor_date(ymd_hms(da$Arrived_Date_Time), unit="day"))
    outa <- setDT(as.data.frame.matrix(table(da$Arrived_Date, da$Feed_Name)),
keep.rownames = TRUE)
  }
```

This code chunk runs if the user specified `TRUE` for wanting the records that arrived during the month. This is the same exact code as above, except the data and output are suffixed with "a" (which stands for "arrived"), and we are now pulling down and working with `Arrived_Data_Time` instead of `C_Visit_Date_Time`.

```
  if (visit==TRUE & arrived==FALSE) {
    setnames(outv, 1, "Visit_Date")
    return(outv)
```

This code chunk is run if the user asked for visits *only*. The first column is named "Visit_Date", and the output is returned to the user.

```
  } else if (visit==FALSE & arrived==TRUE) {
    setnames(outa, 1, "Arrived_Date")
    return(outa)
```

This chunk does the same as the previous chunk of code, except it runs if the user asked for arrived *only*.

```
  } else if (visit==TRUE & arrived==TRUE) {
    setnames(outa, 1, "Date")
    setnames(outv, 1, "Date")
    outav <- full_join(outa, outv, by="Date", suffix=c("_A", "_V"))
    outav[is.na(outav)] <- 0
    return(outav[order(outav$Date),c(colnames(outav)[1],
sort(colnames(outav)[2:length(colnames(outav))]))])
```

This runs if the user asked for both visits and arrivals. It renames the first column to "Date", and then the data are joined (using `full_join()`). The `suffix` argument is where the "_A" and "_V" are attached to the end of the column names. The next line specifies any missing values as a zero, and the last line returns the output. The arguments inside the brackets say that we want the rows ordered by `Date`, and the columns ordered with the data first and the rest done alphabetically.

```
  } else {
    stop(cat("Please specify visit and/or arrived as TRUE."))
  }
}
```

This last piece of code runs if the user specified `FALSE` for both `visit` and `arrived`.

The `facilityrc` function

Most of the code here is the same as above, so I will only go into detail where it changes.

```
facilityrc <- function(channel, visit, arrived, month, year) {

  suppressPackageStartupMessages(require(lubridate))
  suppressPackageStartupMessages(require(data.table))
  suppressPackageStartupMessages(require(dplyr))
  suppressPackageStartupMessages(require(Hmisc))

  if (visit==TRUE) {
    dv <- sqlQuery(channel, paste0("SELECT C_Visit_Date_Time,
C_Biosense_Facility_ID
                                   FROM KS_PR_PRocessed
                                   WHERE MONTH(C_Visit_Date_Time) = ", month,
" AND YEAR(C_Visit_Date_Time) = ", year))
    dv$Visit_Date <- gsub(" UTC", "",
floor_date(ymd_hms(dv$C_Visit_Date_Time), unit="day"))
```

All of this code is the same as above, except that we are now pulling down `C_Biosense_Facility_ID` instead of `Feed_Name`.

```
    dv <- left_join(dv, sqlQuery(channel, "SELECT C_Biosense_Facility_ID,
Facility_Name FROM KS_MFT"),
                    by="C_Biosense_Facility_ID")
```

The `Processed` table does not have the facility name, but the `MFT` table does. This code joins the data with another data frame that has two columns: the facility ID and name from the `MFT` table. It is a `left_join`, so it drops all rows from the `MFT` data frame that is not in the `Processed` data we first pulled down (i.e., it does not include rows for facilities that were not represented in the original data set we pulled down).

```
    dv$Facility_Name <- droplevels(dv$Facility_Name)
```

Since `Facility_Name` was saved as a factor, it wants to return values for *all* of the facilities, even if they aren't found in the data set. We simply drop unused levels here, so that we don't have a bunch of zeros in our data frame from facilities that aren't in production or didn't report any records during the time frame we are looking at.

```
    namekey <- unique(dv[,c("C_Biosense_Facility_ID","Facility_Name")])
```

This line generates a key for IDs and names: It takes the unique parts of the list of IDs matched with names (i.e., the facility ID and name are only represented once). We will use this in a second to match IDs with facility names

```
    outv <- setDT(as.data.frame.matrix(table(dv$Visit_Date,
dv$C_Biosense_Facility_ID)), keep.rownames = TRUE)
```

Just like in the `feedrc` function, this creates the output we want.

```
    for (i in 2:length(colnames(outv))) {
      id <- colnames(outv)[i]
      label(outv[[id]]) <-
as.character(namekey[which(namekey$C_Biosense_Facility_ID==id),"Facility_Name
"])
    }
  }
```

This is a `for` loop that performs an action for every column *except* for the first one (which is always date). It takes the ID number from the name of columns and saves it into an object called `id`. Then the `label` for this column is set to the `Facility_Name` that is on the same row as the ID number in the `namekey`. The curly brackets end the `for` loop and then end the `if` statement, respectively.

```
  if (arrived==TRUE) {
    da <- sqlQuery(channel, paste0("SELECT Arrived_Date_Time,
C_Biosense_Facility_ID
                                   FROM KS_PR_PRocessed
                                   WHERE MONTH(Arrived_Date_Time) = ", month,
" AND YEAR(Arrived_Date_Time) = ", year))
    da$Arrived_Date <- gsub(" UTC", "",
floor_date(ymd_hms(da$Arrived_Date_Time), unit="day"))
    outa <- setDT(as.data.frame.matrix(table(da$Arrived_Date,
da$C_Biosense_Facility_ID)), keep.rownames = TRUE)

    da <- left_join(da, sqlQuery(channel, "SELECT C_Biosense_Facility_ID,
Facility_Name FROM KS_MFT"),
                    by="C_Biosense_Facility_ID")
    da$Facility_Name <- droplevels(da$Facility_Name)
    namekey <- unique(da[,c("C_Biosense_Facility_ID","Facility_Name")])
    outa <- setDT(as.data.frame.matrix(table(da$Arrived_Date,
da$C_Biosense_Facility_ID)), keep.rownames = TRUE)
    for (i in 2:length(colnames(outa))) {
      id <- colnames(outa)[i]
```

```
      label(outa[[id]]) <-
as.character(namekey[which(namekey$C_Biosense_Facility_ID==id),"Facility_Name
"])
    }
  }
```

This code chunk is the same as above, except it runs for the `Arrived_Date_Time`, and only will be executed if the user specifies `TRUE` for `arrived`.

```
  if (visit==TRUE & arrived==FALSE) {
    setnames(outv, 1, "Visit_Date")
    return(outv)

  } else if (visit==FALSE & arrived==TRUE) {
    setnames(outa, 1, "Arrived_Date")
    return(outa)

  } else if (visit==TRUE & arrived==TRUE) {
    setnames(outa, 1, "Date")
    setnames(outv, 1, "Date")
    outav <- full_join(outa, outv, by="Date", suffix=c("_A", "_V"))
    outav[is.na(outav)] <- 0
    return(outav[order(outav$Date),c(colnames(outav)[1],
sort(colnames(outav)[2:length(colnames(outav))]))])

  } else {
    stop(cat("Please specify visit and/or arrived as TRUE."))
  }
}
```

Lastly, all of this code is the same as it was for the `feedrc` function; it is simply preparing the output and returning it to the user.