Week_10

Dan Brooks

April 2, 2016

```
library(RJSONIO)
library(plyr)
library(stringr)
library(RCurl)
```

```
## Loading required package: bitops
```

```
library(svDialogs)
```

```
## Loading required package: svGUI
```

```
#This is the overall function. It takes all of the information that is created by the other
#parts of the program and creates the URL that is sent to the data frame function that will go
#out and get the data from the NY API website.

#Parameters: NONE

#Returns:
# 1) The finished URL for the NY times API
# 2) The page number that the user has specified

#Function Calls:
# 1) Population(), gets the population paramter from the user
# 2) Elevation(), gets the elevation parameter from the user
# 3) Feature_Class(), gets the feature class parameter from the user
# 4) Feature_Code(), gets the featire code parameter from the user
# 5) DataFrame(), passes the finished URL for the final data fram processing

GeoInterface <- function()
{
key <- dlgInput(message = "What is your Geographic API Key", default = "325ac5b2fac111f7e51b00df6f61b68
apikey <- paste("api-key=", key, sep = '')
base <- "http://api.nytimes.com/svc/semantic/v2/geocodes/query.json?"

choices <- c("Population", "Elevation", "Feature_Class", "Feature_Code")
i <- dlgList(choices, title = "Choose Query Parameter", multiple = FALSE, preselect = NULL)$res

switch(i,
       "Population" = {poppop <- population()
                       pagenum <- perpage()
                       url <- paste(base, poppop, pagenum[1], apikey, sep='')},

       "Elevation" = {popele <- elevation()
                      pagenum <- perpage()
                      url <- paste(base, popele, pagenum[1], apikey, sep='')},
```

```
       "Feature_Class" = {popfeat <- feature_class()
                          pagenum <- perpage()
                          url <- paste(base, popfeat[1], pagenum[1], apikey, sep='')},
       "Feature_Code" = {popcode <- feature_code()
                         pagenum <- perpage()
                         url <- paste(base, popcode, pagenum[1], apikey, sep='')}

      )

dataframe(url, pagenum[2])
}
```

```
#This is the elevation function. It will deal with all of the information needed for
#elevation portion of the users choice. It will ask the user for the comparison
#greater than, less than, or between, and then ask for one or two input parameters
#depending on the choice. It will then ask for the feature class to pair with the
#elevation parameter.

#Parameters: NONE

#Returns:
# 1) The finished URL for the NY times API

#Function Calls:
# 1) Feature_Class(), gets the feature class parameter from the user

elevation <- function()
{
  choices <- c("Greater_Than", "Less_Than", "Between")
  i <- dlgList(choices, title="Elevation", multiple = FALSE, preselect = NULL)$res

  if (i == "Between")
  {
    min <- dlgInput(message = "Minimum Number:", default = '100', gui = .GUI)$res
    max <- dlgInput(message = "Maximum Number:", default = '1000', gui = .GUI)$res

    if (as.numeric(min) >= as.numeric(max))
    {
      dlgMessage(message = "Minimum Number must be less that Maximum Number", type = "ok")
      elevation()
    }
    else
    {
      popfeat <- feature_class()
      url <- paste("elevation=", as.numeric(min), "_", as.numeric(max), "&", "feature_class=", popfeat[1
      return(url)
    }
  }
  else if (i == "Less_Than")
  {
    min <- dlgInput(message = "Maximum Number (Less Than):", default = '1000', gui = .GUI)$res
    popfeat <- feature_class()
    url <- paste("elevation=_", as.numeric(min), "&", "feature_class=", popfeat[2], "&", sep = '')
```

```
    return(url)
  }
  else
  {
    max <- dlgInput(message = "Minimum Number (Greater Than):", default = '100', gui = .GUI)$res
    popfeat <- feature_class()
    url <- paste("elevation=", as.numeric(max), "_", "&", "feature_class=", popfeat[2], "&", sep = '')
    return(url)
  }
}
```

```
#This is the population function. It will deal with all of the information needed for
#population portion of the users choice. It will ask the user for the comparison
#greater than, less than, or between, and then ask for one or two input parameters
#depending on the choice. It will then ask for the feature class to pair with the
#population parameter.

#Parameters: NONE

#Returns:
# 1) The finished URL for the NY times API

#Function Calls:
# 1) Feature_Class(), gets the feature class parameter from the user

population <- function()
{
  choices <- c("Greater_Than", "Less_Than", "Between")
  i <- dlgList(choices, title="Population", multiple = FALSE, preselect = NULL)$res

  if (i == "Between")
  {
    min <- dlgInput(message = "Minimum Number:", default = '100', gui = .GUI)$res
    max <- dlgInput(message = "Maximum Number:", default = '1000', gui = .GUI)$res

    if (as.numeric(min) >= as.numeric(max))
    {
      dlgMessage(message = "Minimum Number must be less that Maximum Number", type = "ok")
      population()
    }
    else
    {
      popfeat <- feature_class()
      url <- paste("population=", as.numeric(min), "_", as.numeric(max), "&", "feature_class=", popfeat
      return(url)
    }
  }
  else if (i == "Less_Than")
  {
    min <- dlgInput(message = "Maximum Number (Less Than):", default = '1000', gui = .GUI)$res
    popfeat <- feature_class()
    url <- paste("population=_", as.numeric(min), "&", "feature_class=", popfeat[2], "&", sep = '')
    return(url)
```

```
  }
  else
  {
    max <- dlgInput(message = "Minimum Number (Greater Than):", default = '100', gui = .GUI)$res
    popfeat <- feature_class()
    url <- paste("population=", as.numeric(max), "_", "&", "feature_class=", popfeat[2], "&", sep = '')
    return(url)
  }
}
```

```
#This is the feature class function. This function will produce a list of possible
#choices to the user. The user can then pick whichever class they would like. The chosen
#class is then taken and, depending on the choice, the switch statement will create the URL
#with the proper class choice.

#Parameters: NONE

#Returns:
# 1) The finished URL for the NY times API
# 2) The class choice that will be used in the population and elevation functions

#Function Calls: NONE

feature_class <- function()
{
  choices <- c("A", "H", "L", "P", "S", "T")
  i <- dlgList(choices, title="Choose Feature_Class", multiple = FALSE, preselect = NULL)$res

switch(i,
       "A" = {url <- paste("feature_class=", i, "&", sep ='')},
       "H" = {url <- paste("feature_class=", i, "&", sep ='')},
       "L" = {url <- paste("feature_class=", i, "&", sep ='')},
       "P" = {url <- paste("feature_class=", i, "&", sep ='')},
       "S" = {url <- paste("feature_class=", i, "&", sep ='')},
       "T" = {url <- paste("feature_class=", i, "&", sep ='')}
      )
return(c(url,i))
}
```

```
#This is the page number function. Thos function allows the user to specify the
#number of items they want to see in the final data frame. If no number is specified
#then 20 will be the number that is returned. Sometimes there are less items than
#the user specifies. There is a prompt that will tell the user if there is less than
#the number the specified.

#Parameters: NONE

#Returns:
# 1) The finished URL for the NY times API
# 2) The number of pages that the user specified

#Function Calls: NONE
```

```
perpage <- function()
{
 pages <- dlgInput(message = "How many data frame items would you like?", default = '20', gui = .GUI)$r
 if (as.numeric(pages) < 0)
 {
   dlgMessage(message = "Please input a positive number", type = "ok")
   perpage()
 }
 else
 {
   url <- paste("perpage=", as.numeric(pages), "&", sep = '')
   return(c(url, pages))
 }
}
```

```
#This is the feature code function. It reads in a list of feature codes from the API website.
#I takes the codes and creates a list for the user to select. The list contains the code
#and the class associated with each other, because each code has a specific class
#associated with it. Once the user picks the desired code the URL is created.

#Parameters: NONE

#Returns:
# 1) The finished URL for the NY times API

#Function Calls: NONE

feature_code <- function()
{
options(warn=-1)
feature_code_table <- read.table("http://download.geonames.org/export/dump/featureCodes_en.txt", sep =
choices <- as.character(feature_code_table$V1)

i <- select.list(choices, graphics = TRUE, title = "Choose a feature code", multiple = FALSE)
class <- str_extract(i, "[[:upper:]]")
code <- str_extract(i, "[[:upper:]]{2,7}")

url <- paste("feature_class=", class, "&", "feature_code=", code, "&", sep = '')
return(url)
}
```

```
#This is the dataframe function. This funtion takes the URL that is made
#throughout the entire process and goes out to the API website to get the JSON data.
#It takes the JSON data that is returned and converts it to a data frame. It could
#then return the data frame for further analysis if it is neccessary. There had to be
#a little manipulations for the JSON data to get it to work with the dat farme conversion,
#and there were a few that returned no data at all. There is error catching to ensure the
#user will not break the code

#Parameters:
# 1) uRL, that will be used to go and get the JSON data fro, the NY API
# 2) pagenum, the number of items the user would like to see
```

```
#Returns:
# 1) The data frame from the JSON data

#Function Calls: NONE

dataframe <- function(url, pagenum)
{

data <- fromJSON(url, simplifyDataFrame = TRUE)
remove <- c(data$status, data$copyright, data$num_results)
data <- data[! data %in% remove]

geo.unlist <- sapply(data[[1]], unlist)

if(length(geo.unlist) == 0)
{
  i <- dlgMessage(message = "There appears to be no Data available, would you like to try again?", type
  if (i == "yes")
   {
     GeoInterface()
   }
  else
   {
    dlgMessage(message = "Thank You Very Much!", type = "ok")
   }
}
else if (length(geo.unlist) < pagenum)
{
 geo.df <- do.call("rbind.fill", lapply(lapply(geo.unlist,t), data.frame))
 if (colnames(geo.df[1]) == "X..i..")
 {
  i <- dlgMessage(message = "There appears to be no Data available, would you like to try again?", type
  if (i == "yes")
   {
     GeoInterface()
   }
  else
   {
    dlgMessage(message = "Thank You Very Much!", type = "ok")
   }
 }
 else
 {
  dlgMessage(message = paste("There is only", length(geo.unlist), "items worth of data"), sep = "", typ
  View(geo.df)
 }
}
else
{
geo.df <- do.call("rbind.fill", lapply(lapply(geo.unlist,t), data.frame))
if (colnames(geo.df[1]) == "X..i..")
 {
  i <- dlgMessage(message = "There appears to be no Data available, would you like to try again?", type
```

```
  if (i == "yes")
   {
     GeoInterface()
   }
  else
   {
    dlgMessage(message = "Thank You Very Much!", type = "ok")
   }
 }
 else
 {
  View(geo.df)
 }
}
}
```