CAL Project - Knowledge Transfer Documentation - Data Cleaning

4/27/2020

This document outlines the data cleaning steps that the CAL team performed before beginning analysis. The six files that were provided and subsequently cleaned are Academic, Membership, Individual Info, Engagement, Events and Emails.

Preliminary Steps

First we load the required libraries for this script. We then set a global year variable.

```
## Libraries used
library(reticulate)
library(dplyr)
library(naniar)
library(ggplot2)
library(lubridate)
library(data.table)
library(tidyverse)
library(dtplyr)
library(stringr)
require(scales)

## Global Variables Used
# current fiscal year
current_fy = ifelse( month(Sys.Date()) > 6, year(Sys.Date()) + 1,
                     year(Sys.Date()))
```

Academic

Read the file

```
academic = read.csv('D:/Group Folder/Data/academic.csv')
```

There are 617,372 unique people in the `academic` file.

There are many duplicates in this file, meaning that one person might appear several times due to system change, due to different majors.

Per discussion, we noticed that, for those `duplicate` people, their college code should be less than 4 numbers.

So

we created a new column indicating the word count of the `CODE_CLG` and only remove those records whose `code_clg` greater than 4.

```
academic_group$leng = str_count(academic_group$CODE_CLG_QUERY)
academic_dedup = academic_group %>% filter(!((leng == 4) & (fre > 1)))
```

And then, there are still some people who appear multiple times, we random select one records.

```
# window rank creation
academic_dedup = academic_dedup %>% group_by(ID_DEMO) %>% mutate(rownum =
row_number())

# select final
academic_final = academic_dedup %>% filter(rownum == 1)
```

The last part for the `academic` cleaning is to remove missing values.

First, we checked the missing values for each column.

```
sapply(academic_dedup, function(x) sum(is.na(x)))
```

We found that when there are 35413 missing records in column `YEAR_GRAD_CODE`. After further confirmation, for those records missing with this value, other fields are also missing. So these records are meaningless for our analysis. We can directly remove them.

```
# keep non-missing rows
academic_export = academic_final[which(!is.na(academic_final$YEAR_GRAD)),]
sum(is.na(academic_export))

# remove those unknown people whose YEAR_GRAD is 9999
academic_export = academic_export %>% filter(!(YEAR_GRAD == 9999))

# remove those people whose YEAR_GRAD is great than the current year
academic_export = academic_export %>% filter(!(YEAR_GRAD >= year(Sys.Date())))
```

The last part is the export the file

```
# export
write.csv(academic_export %>%
            select(ID_DEMO,YEAR_GRAD,CODE_UM_DEGREE_LEVEL),
          'D:\\Group Folder\\Data\\Cleaned data
sets\\Academic_cleaned.csv',row.names = FALSE)
```

Membership

Read file.

```
membership = read.csv("D:/Group Folder/Data/membership.csv")
membership_clean = membership
```

First we filter out the record with ID_DEMO = 4B0BE2AB589CDD2B8ED0193018EB3015 due to it containing 5532 records and acting as a holding record

```
# ID_DEMO
membership_clean = membership_clean %>%
    filter(ID_DEMO != "4B0BE2AB589CDD2B8ED0193018EB3015")
```

Next we remove rows that do not have membership levels or appeal codes. We keep records that have membership level information but no appeal code, replacing the appeal code field with "None".

```
# APPEAL_CODE
# Replace rows without appeal codes but with membership level with "None"
membership_clean = membership_clean %>%
  mutate(APPEAL_CODE = ifelse((APPEAL_CODE == "") &
                              (MEMBERSHIP_LEVEL !=
""),"None",as.character(APPEAL_CODE)))

# Remove rows that remain without appeal codes or membership levels
membership_clean = membership_clean %>% filter(APPEAL_CODE != "")
```

We then filter our records that are above $1000 and below $0. There were 457 records of this type.

```
# AMOUNT_PAID
# Remove donations (> 1000) and refunds (< 0)
membership_clean = membership_clean %>% filter(membership_clean$AMOUNT_PAID >= 0
& membership_clean$AMOUNT_PAID < 1000)
```

Next, we adjust the the date column to represent UMAAs fiscal year

```
# TRANSACTION_DATE
membership_clean$TRANSACTION_DATE = as.Date(membership_clean$TRANSACTION_DATE,
format = "%m/%d/%Y")
membership_clean$FISCAL_YEAR = ifelse(month(membership_clean$TRANSACTION_DATE)
>= 7,
                                       year(membership_clean$TRANSACTION_DATE)+1,
                                       year(membership_clean$TRANSACTION_DATE))
```

Finally we keep only records with references to Annual or Life memberships. This removes 26,684 records with references to extensions, adjustments etc.

```
# MEMBERSHIP_LEVEL
# keep only rows pertaining to annual, lifetime memberships
membership_clean = membership_clean %>% filter(grepl("Annual", MEMBERSHIP_LEVEL)
| grepl("Life", MEMBERSHIP_LEVEL))
```

Write cleaned file.

```
write.csv(membership_clean, "D:/Group Folder/Data/Cleaned data
sets/membership_cleaned.csv",row.names = FALSE)
```

Individual Info

We first run the python script to transform the csv files from a double pipe separated file to a comma separated file. Following the script, we read the csv file.

```
py_run_file('D:/Group Folder/Code/Read individual_info.py')
individual_info = read.csv('D:/Group Folder/Data/individual_info.csv')
```

We select only those features that are required for our analysis.

```
individual_info_clean <- individual_info[,c("ID_DEMO",
                                              "ID_SPOUSE",
                                              "MARITAL_STATUS",
                                              "GENDER",
                                              "BIRTH_YEAR",
                                              "DEATH_YEAR",
                                              "AGE",
                                              "ZIP_CODE",
                                              "IN_TC_METRO_AREA",
                                              "HOUSEHOLD_INCOME",
                                              "IS_TC_GRAD",
                                              "MOST_RECENT_COLLEGE",
                                              "MOST_RECENT_GRAD_YEAR",
                                              "ATHLETIC_INTEREST",
                                              "TRAVEL_INTEREST",
                                              "MEMBERSHIP_TYPE_CODE",
                                              "MEMBERSHIP_STATUS_CODE",
                                              "FIRST_EVER_START_DATE",
                                              "EXP_DATE_LAST_MEMBERSHIP")]
```

We replace blanks in our dataset with `NA`. `NA` indicates that we do not have values for a given record and column.

```
# Replace '' with NA in the dataframe
individual_info_clean[individual_info_clean==''] <- NA
```

We keep the records of only those who are twin cities graduates.

```
individual_info_clean <-individual_info_clean %>% filter((IS_TC_GRAD %in% NA)|
(IS_TC_GRAD=='Y'))
```

We remove the records of alumni who are no longer with us. We retain the records of living alumni along with the records of those with death year value of 9999.

```
individual_info_clean <-individual_info_clean %>% filter((DEATH_YEAR==9999)|
(is.na(DEATH_YEAR)))
```

To determine the living status of those records with DEATH_YEAR value of 9999, we inspect the BIRTH_YEAR. For records with BIRTH_YEAR as NA, we impute values based on MOST_RECENT_GRAD_YEAR by subtracting 23 from MOST_RECENT_GRAD_YEAR.

```
individual_info_clean <- individual_info_clean%>%
  mutate(BIRTH_YEAR = ifelse(
    ((DEATH_YEAR==9999)&is.na(BIRTH_YEAR)&(MOST_RECENT_GRAD_YEAR==9999)),NA,
    ifelse((is.na(BIRTH_YEAR)&(DEATH_YEAR==9999)),
           MOST_RECENT_GRAD_YEAR-23,
           BIRTH_YEAR)) )
```

We now determine the AGE of all alumni using BIRTH_YEAR. The function `Sys.Date()` is an inbuilt R code function that gives us the current year.

```
# Calculate age using BIRTH_YEAR

individual_info_clean <- individual_info_clean %>%
  mutate(AGE = year(Sys.Date()) - BIRTH_YEAR)
```

We retain the records of only those alumni that have an age between 0 and 100.

```
individual_info_clean <- individual_info_clean %>% filter((AGE %in% NA) | ((AGE
>= 0) & (AGE  <= 100)))
```

We consider the remaining alumni as a living alumni.

```
# Consider all the remaining members as living

individual_info_clean$DEATH_YEAR <- na_if(individual_info_clean$DEATH_YEAR,9999)
```

We impute the records with missing AGE value using the mean value for AGE in the table.

```
individual_info_clean <- individual_info_clean%>% mutate(AGE = ifelse((AGE %in%
NA), mean(AGE, na.rm = TRUE),AGE) )
```

Zip code and household income information was taken from the United States Census Bureau(https://www.census.gov/search-results.html?q=minnesota+median+income&page=1&stateGeo=none&searchtype=web&cssp=SERP&_charset_=UTF-8), and a github site (https://gist.github.com/Radcliffe/5ba82ed06ba92fa5e27e), and merged with the individual info table to determine the TC metro area residents.

```
individual_info_clean$ZIP_CODE <- as.character(individual_info_clean$ZIP_CODE)
income_zipcode <- read_csv('D:/Group Folder/Data/Cleaned data
sets/income_zipcode.csv')
income_zipcode$zip_code = as.character(income_zipcode$zip_code)
# Join zipcode file
individual_info_clean = individual_info_clean %>% left_join(income_zipcode, by =
c("ZIP_CODE" = "zip_code"))
# Add new column to detect in TC area
individual_info_clean = individual_info_clean %>% mutate(IN_TC_METRO_AREA =
ifelse(TC_AREA == 1, "Y", "N"))
# Add new column for Household Income
individual_info_clean = individual_info_clean %>% mutate(HOUSEHOLD_INCOME =
ifelse(!is.na(HOUSEHOLD_INCOME), HOUSEHOLD_INCOME,

      ifelse(is.na(age_25), NA,

             ifelse(AGE < 26, age_25,

                    ifelse(AGE < 45, age_25_44,

                           ifelse(AGE < 65, age_45_64, age_65))))))

# Remove columns that are not required

individual_info_clean <- individual_info_clean %>% select(-c(TC_AREA, County,
age_25, age_25_44, age_45_64,age_65))
```

We remove the records of those alumni who don't have a value for IN_TC_METRO_AREA and ZIP_CODE.

```
individual_info_clean <- individual_info_clean%>%filter(!((IN_TC_METRO_AREA %in%
NA) & (ZIP_CODE%in%NA)))
```

We impute the records with missing HOUSEHOLD_INCOME value using the mean value for HOUSEHOLD_INCOME in the table.

```r
# Household income

individual_info_clean <- individual_info_clean %>%
group_by(AGE)%>%mutate(HOUSEHOLD_INCOME = ifelse(is.na(HOUSEHOLD_INCOME),
mean(HOUSEHOLD_INCOME, na.rm=TRUE),HOUSEHOLD_INCOME ) )
```

We replace `NA` in `IN TC METRO AREA` with 'N' indicating that the alumni are not from the TC metro area.

```r
individual_info_clean <- individual_info_clean %>% mutate(IN_TC_METRO_AREA =
ifelse(IN_TC_METRO_AREA %in% NA, "N","Y") )
```

We convert the FIRST_EVER_START_DATE and EXP_DATE_LAST_MEMBERSHIP to date format.

```r
individual_info_clean$FIRST_EVER_START_DATE <-
mdy(individual_info_clean$FIRST_EVER_START_DATE)

individual_info_clean$EXP_DATE_LAST_MEMBERSHIP <-
mdy(individual_info_clean$EXP_DATE_LAST_MEMBERSHIP)
```

We remove features that do not add value to the dataset.

```r
individual_info_clean <- individual_info_clean %>% select(-c(DEATH_YEAR,
IS_TC_GRAD))
```

We are remove the records of alumni with a gender value of "U".

```r
individual_info_clean <- individual_info_clean %>% filter(GENDER != "U")
```

We write the cleaned individual_info file to a new file `individual_info_cleaned.csv`.

```r
write.csv(individual_info_clean, "D:/Group Folder/Data/Cleaned data
sets/individual_info_cleaned.csv", row.names = FALSE)
```

Engagement

Read file.

```r
# read raw engagement score file
eng <- read.csv("D:\\Group Folder\\Data\\engagement.csv", header = TRUE)
```

There are several records with NULL in the fiscal year column in the engagement file. Below code removes such records.

```r
eng_no_blank <- eng[!is.na(eng$YEAR_FISCAL),]
```

Below code removes fiscal year 2014 records because all the scores are 0's where fiscal year = 2014.

```
eng_cleaned <- eng_no_blank[eng_no_blank$YEAR_FISCAL != 2014,]
```

Finally, saving the cleaned Engagement data frame into a csv file.

```
write.csv(eng_cleaned, "D:/Group Folder/Data/Cleaned data
sets/engagement_cleaned.csv", row.names = FALSE)
```

Events

```
py_run_file('D:/Group Folder/Code/Read events.py')
events <- read_csv('D:/Group Folder/Data/events.csv')
```

We want to make sure our date format is correct so we use `mdy` from `lubridate`.

We are also making the `NAME_GROUP` lower case so its easier to text-match and then selecting just each distinct event. This will help speed things up.

```
events <- events %>%
  mutate(DATE_EVENT = mdy(DATE_EVENT)) %>%
  mutate(YEAR_FISCAL = ifelse(month(DATE_EVENT) > 6,            year(DATE_EVENT)
+ 1, year(DATE_EVENT))) %>%
  mutate(NAME_GROUP = str_to_lower(events$NAME_GROUP))


### Applying broad categories to email names
events_distinct <- events %>%
    distinct(NAME_GROUP)
```

These are the categories we have discussed before. If a word from `NAME_GROUP` matches one of these items, the event falls into one of the following categories.

```
network <- c('day of
service|fundraiser|mcan|epn|connect|summit|leader|network|event|alumni|job|caree
r|aaw')


social <-
c('mnu|behind|bts|dta|brunch|interest|date|gradfest|engage|resched|postpo|valley
fair|invite|holiday|matchmaker|gala|birthday|farewell|volunteer|potluc|picnic|ni
ght|reunion|social|happy
hour|mixer|brew|wine|tour|travel|movie|breakfast|lunch|dinner|meet|skiumania|ser
vice|trip|gathering|awards|travel|celebration|party|ceremony|bike|ski-u')


laws <- c('affordable
care|dc|candid|state|climate|legis|laws|election|senate|vote|board|counc|capitol
|voting|forum|pre-ele|advocacy')
```

```
sports <-
c('btn|goldy|fleck|coach|sparks|sports|football|basketball|twins|bowl|game|hocke
y|golf|soccer|baseball|volleyball|halftime|timber|wild|rockies|gophers|lynx|tick
ets|homecoming|ncaa|frozen|athletics|bball|gymnastics|giants|big ten')


learning1 <-
c('webinar|negotiate|branding|web|learning|coursera|minnecollege|minne-
coll|minne coll')


learning2 <- c('librar|talk|web|learning|coursera|lecture|speaker')
```

Text Matching (Events)

Now we need to perform the actual text-matching.

- grepl - gives us either T or F if a word matches
- paste - separates each item from a list. "Collapse" just tells R what separator we used.
    - For example, we have `giants|big ten` at the end of the sports list. These will become two separate items.
- NAME_GROUP - the original column from our dataset that we are searching in

This chunk is to break apart the ugly text-matching function.

```
# Creating a new column
mutate(broad_cat =
    ifelse(..., 'Learning')

# Separates each item from the list
paste(learning2, collapse='|')

# Gives us either T or F if any item from the list matches a word from
NAME_GROUP
grepl(..., NAME_GROUP)
```

Here is the whole ugly function.

```
events_distinct <- events_distinct %>%
  mutate(broad_cat =
    ifelse(grepl(paste(learning1, collapse='|'),                    NAME_GROUP),
'Learning',
    ifelse(grepl(paste(sports, collapse='|'),                       NAME_GROUP),
'Sports',
    ifelse(grepl(paste(social, collapse='|'),                       NAME_GROUP),
'Social',
    ifelse(grepl(paste(laws, collapse='|'),                         NAME_GROUP),
'Legislature',
    ifelse(grepl(paste(learning2, collapse='|'),                    NAME_GROUP),
'Learning',
    ifelse(grepl(paste(network, collapse='|'),                      NAME_GROUP),
'Networking','Other'))))))) %>%
  select(-NAME_GROUP)
```

Now that we have categorized each event, we can add the categories into the full `events` file.

```
events <-
    events %>%
    inner_join(events_distinct, by = 'NBR_GROUP')

write.csv(events, 'D:/Group Folder/Data/Cleaned data sets/events_cleaned.csv',
row.names = FALSE)
```

Emails

`fread` is what we use when a data file is very large. This comes from the package `data.table`.

```
emails <- fread('D:/Group Folder/Data/emails.csv')
```

This chunk just gets the initial cleaning done.

We filter our "bounced" emails because they don't help us understand membership. We manually create a fiscal year. We also create a column for if an email was clicked or opened.

```
# Filter bounced
emails_revised <- emails %>% lazy_dt() %>%
    mutate(DATE_CONTACT = mdy(DATE_CONTACT)) %>%
    filter(CODE_OUTCOME != 'BO') %>%

    # Create fiscal year
    mutate(YEAR_FISCAL = ifelse(month(DATE_CONTACT) > 6,
        year(DATE_CONTACT) + 1, year(DATE_CONTACT))) %>%

    # Opened or clicked
    mutate(Status_Clicked =
           ifelse(CODE_OUTCOME == 'CL' |
                  CODE_OUTCOME == 'OE', 1, 0))
```

Like we had done for events, we make `DESC_PATH` (the pseudo-subject line) lowercase to help us text-match.

We only want to perform text-matching one time per email, so we select only distinct email names.

```
emails$DESC_PATH <- str_to_lower(emails$DESC_PATH)

# Just getting distinct emails to create categories
emails_distinct <-
    emails_revised %>% distinct(DESC_PATH)
```

These are the categories we have discussed before. If a word from `DESC_PATH` matches one of these, the email falls into one of the following categories.

```
clothes <- c('shirt|pants|clothes|socks|shopping|apparel|cybermonday|presale')


laws <-
c('dc|candid|state|climate|legis|laws|election|senate|vote|board|counc|capitol|v
oting|forum|pre-ele|advocacy')


sports <-
c('btn|mnu|fleck|coach|sparks|sports|football|basketball|twins|bowl|game|hockey|
golf|soccer|baseball|volleyball|halftime|timber|wild|rockies|gophers|lynx|ticket
s|homecoming|ncaa|frozen|athletics|bball|gymnastics|giants|big ten')


regions <- c('swfl|france|japan|phoenix|beijing|vietnam|milwaukee|korea|new
jersey|asia|florida|arizona|california|washington|seattle|india|vietnam|colorado
|atlanta|internation|china|london|nyc|chicago|new mexico|houston')


learning <- c('minnecollege|minne-
|mcan|epn|career|tech|mentor|librar|talk|webinars|web|learning|coursera|lecture|
speaker')
```

```
solicit <- c('gtmd|deal|opt|new
grad|redeem|give|gift|appeal|donor|solicit|membership|survey|at the
u|spring|fall|promo|offer|fundraiser|renew')


mass_updates <- c('aaw|system|news|angle|mem|non-
|nonmem|announce|report|email|update|birthday')


social <- c('bts|dta|gala|brunch| at | in
|suncoast|interest|date|gradfest|engage|resched|postpo|farewell|remind|volunteer
|valleyfair|invite|holiday|connect|summit|leader|potluc|picnic|join|night|reunio
n|social|happy
hour|mixer|networking|event|network|brew|wine|tour|travel|movie|breakfast|lunch|
dinner|meet|alumni|skiumania|service|trip|gathering|awards|travel|celebration|pa
rty|ceremony|bike|ski-u')
```

Text Matching (Emails)

Please see the "Text Matching" section from Events above for further understanding of this function.

```
emails_distinct <- emails_distinct %>%
  mutate(broad_cat =
    ifelse(grepl(paste(clothes, collapse='|'),
                 DESC_PATH), 'Solicitations',
    ifelse(grepl(paste(laws, collapse='|'),
                 DESC_PATH), 'Legislature',
    ifelse(grepl(paste(sports, collapse='|'),
                 DESC_PATH), 'Sports',
    ifelse(grepl(paste(learning, collapse='|'),
                 DESC_PATH), 'Learning',
    ifelse(grepl(paste(solicit, collapse='|'),
                 DESC_PATH), 'Solicitations',
    ifelse(grepl(paste(mass_updates, collapse='|'),
                 DESC_PATH), 'Mass Updates',
    ifelse(grepl(paste(social, collapse='|'),
                 DESC_PATH), 'Social',
           'Other'))))))))
```

We can now apply the email names to every single email in our dataset.

`merge.data.table` is how we perform an inner join when we use `data.table`.

```
emails_revised <- emails_revised %>% merge.data.table(emails_revised,
emails_distinct, by = 'CODE_PATH')
```

```
fwrite(emails_revised, file='D:/Group Folder/Data/Cleaned data
sets/emails_cleaned.csv',row.names = FALSE)
```