01_Linear_Interpolation

Zarni Htet (zh938@nyu.edu)

Linear Interpolation

This Markdown is filling the missing data for BMI and media exposure at asynchronous time points using linear interpolation. The project is supervised by Professor Marc Scott and Professor Daphna Harel. The data is from the Belle Lab at the Bellevue Hospital. More details of the project scope in the repository under lit folder.

R Libraries

This code block has all the needed R libraries

```
#For the dta raw files
library(foreign)
```

```
## Warning: package 'foreign' was built under R version 3.3.2
```

```
#For importing different types of data set without specification
library(rio)
```

```
## Warning: package 'rio' was built under R version 3.3.2
```

```
#For processing long form data
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Uploading Raw data

In this code chunk, we are uploading raw .dta data and converting to it a csv. This will then be saved to a processing data folder to protect the integrity of the raw data.

```
#The BMI data extract
bmi <- read.dta("../../data/raw/MASextract1.dta")
#The Media data extract
media <- read.dta("../../data/raw/MASextract2.dta")
#Writing the BMI data to processing file
write.csv(bmi, "../../data/processing/bmi.csv")
#Writing the media data to processing file
write.csv(media, "../../data/processing/media.csv")
```

Loading the Data back from Processing Folder

This code chunk is loading the working version of the data extra to be used throughout the document.

```r
#processing bmi data
p_bmi <- import("../../data/processing/bmi.csv")
p_media <- import("../../data/processing/media.csv")
```

Data Exploration

This code chunks examine the two data sets. In particular, the focus here is on the key variables and the time intervals they are recorded. At the end of each code block for each data set, there is a short summary of what the data consists of.

The BMI data set overview

```r
head(p_bmi)
```

```
##   V1 ID_    AgeMos       zBMI
## 1  1   1 0.0000000 -3.5407891
## 2  2   1 0.1314168 -3.1878707
## 3  3   1 0.5585216 -0.2831618
## 4  4   1 1.5441478 -1.2716171
## 5  5   1 4.3039017 -1.1837007
## 6  6   1 6.3737168 -2.5585830
```

```r
tail(p_bmi)
```

```
##          V1   ID_    AgeMos       zBMI
## 10321 10321 91008 0.9199179  1.4600797
## 10322 10322 91008 1.2813141  1.8749880
## 10323 10323 91118 0.0000000 -0.9925033
## 10324 10324 91118 0.5913758  1.3112072
## 10325 10325 91118 1.5112936  1.9073267
## 10326 10326 91118 3.9096510  2.6738663
```

```r
dim(p_bmi) #10326, 4
```

```
## [1] 10326     4
```

```r
#check the number of unique subjects
length(unique(p_bmi$ID_)) #667
```

```
## [1] 667
```

```r
length(unique(p_bmi$AgeMos)) #1951
```

```
## [1] 1951
```

```r
print(sum(is.na(p_bmi$ID_))) #0 no values are missing here
```

```
## [1] 0
```

```r
print(sum(is.na(p_bmi$AgeMos))) #0 no values are missing here
```

```
## [1] 0
```

Each subject has different time points. For subject 1, months may be 0, 0.5, 1.0 while subject 2 has months in 0, 0.7, 1.2 etc.

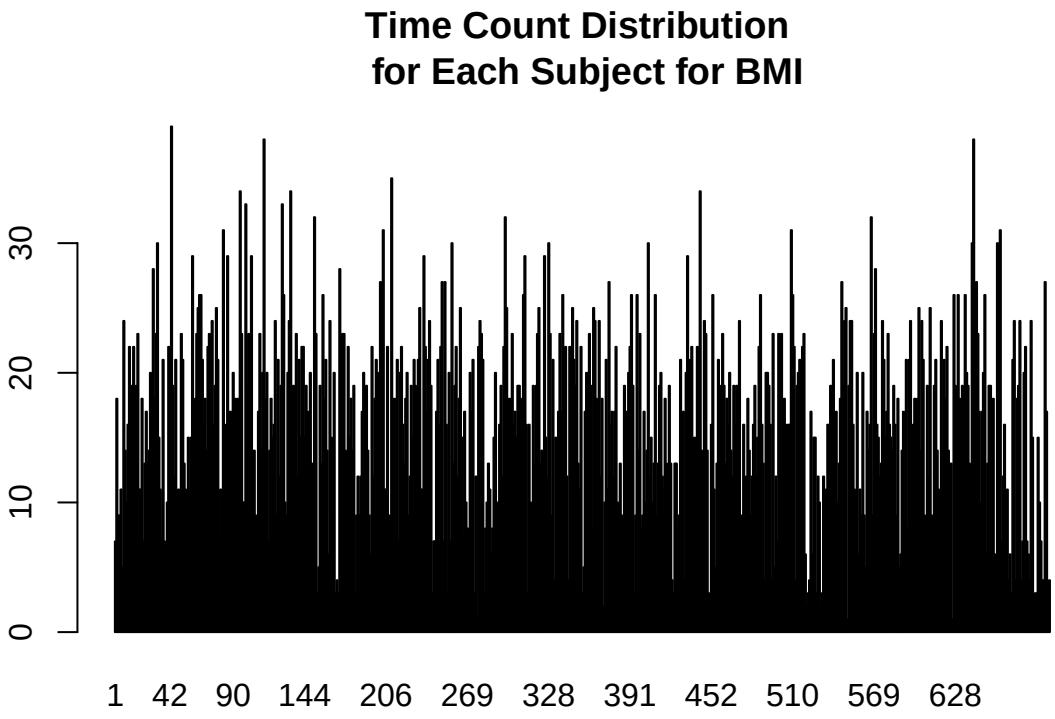
This is to explore the number of time intervals each subject has.

```
#This uses dplyr to group by each subject and count their instances. This effectively counts the number
bmi_timed <- p_bmi %>%
  group_by(ID_) %>%
  summarize(n = n())
print(bmi_timed)
```

```
## # A tibble: 667 × 2
##      ID_     n
##    <int> <int>
## 1      1     7
## 2      2    18
## 3      3     9
## 4      4     9
## 5      5    11
## 6      6     5
## 7      7    24
## 8      8    14
## 9      9    10
## 10    10    16
## # ... with 657 more rows
```

We will do a quick barplot to see the distribution of time points for each subject has

```
#Using the table function and barplot to draw the distribution of time.
barplot(table(bmi$ID_), main = "Time Count Distribution \n for Each Subject for BMI")
```



Check the Minimum/Maximum time intervals. This is to see if we have to explore edge cases later down the road for Last Value Carried forward at the end for each subject

```
min(bmi_timed$n) #1
```

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

```r
max(bmi_timed$n) #39
```

```
## [1] 39
```

At least 1 subject has only 1 time interval for BMI.

Media exposure data set overview

```r
head(p_media)
```

```
##   V1 ID_    AgeMos lnmediatimespent sqrtmediatimespent
## 1  1   1 15.244353         4.948760          11.832160
## 2  2   1  7.786448         5.463832          15.329710
## 3  3   2 24.147844         4.795791          10.954452
## 4  4   2 42.940453         3.433987           5.477226
## 5  5   2  6.735113         4.330733           8.660254
## 6  6   2 60.714581         4.795791          10.954452
```

```r
tail(p_media)
```

```
##        V1   ID_   AgeMos lnmediatimespent sqrtmediatimespent
## 1634 1634 90372 16.09856         5.602119          16.431677
## 1635 1635 90406 50.43121         5.198497          13.416408
## 1636 1636 90425 23.65503         3.433987           5.477226
## 1637 1637 90448 36.66530         4.510859           9.486833
## 1638 1638 90448 27.92608         4.110874           7.745967
## 1639 1639 90448 59.79466         5.198497          13.416408
```

```r
dim(p_media) #1639, 5
```

```
## [1] 1639    5
```

```r
#check the number of unique subjects
length(unique(p_media$ID_)) #542
```

```
## [1] 542
```

```r
length(unique(p_media$AgeMos)) #745
```

```
## [1] 745
```

```r
print(sum(is.na(p_media$ID_))) #0
```

```
## [1] 0
```

```r
print(sum(is.na(p_media$AgeMos))) #0
```

```
## [1] 0
```

This is to explore the number of time intervals each subject has.

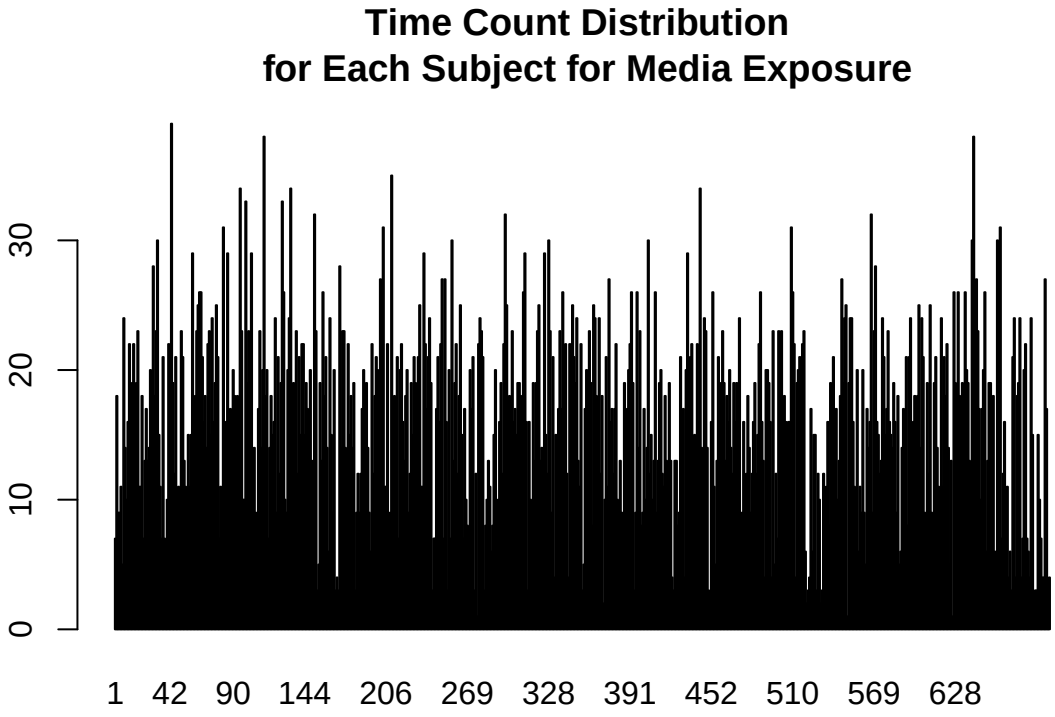
```r
#This uses dplyr to group by each subject and count their instances. This effectively counts the number
media_timed <- p_media %>%
  group_by(ID_) %>%
  summarize(n = n())
print(media_timed)
```

```
## # A tibble: 542 × 2
##      ID_     n
##    <int> <int>
```

```
## 1      1     2
## 2      2     4
## 3      3     1
## 4      4     2
## 5      5     5
## 6      6     3
## 7      7     3
## 8      8     4
## 9      9     2
## 10    10     3
## # ... with 532 more rows
```

Like the BMI from before, each subject has different count of time as well as time intervals where the data is collected.

```
#Using the table function and barplot to draw the distribution of time.
barplot(table(bmi$ID_), main = "Time Count Distribution \n for Each Subject for Media Exposure")
```



Cleaning data

In this section, I will attempt to discover the subjects that only have 1 data point for the BMI data set or the Media exposure data set. These will have to be removed from the working data set as Linear interpolation cannot happen unless we have more than 1 data point. Those subjects will be noted down and saved in a separate file.

BMI cleaning

Figuring out the 1 data point subjects and saving it them to an output file.

```
#Recalling the count by ID dataset from above for BMI.
head(bmi_timed)
```

```
## # A tibble: 6 × 2
##     ID_     n
```

```
##   <int> <int>
## 1     1     7
## 2     2    18
## 3     3     9
## 4     4     9
## 5     5    11
## 6     6     5
```

```
#Getting the indexes for which n = 1
#An assumption has been that for each row, there is no missing corresponding time value or bmi value. T
bmi_exclude <- bmi_timed[bmi_timed$n==1,]
print(bmi_exclude)
```

```
## # A tibble: 3 × 2
##     ID_     n
##   <int> <int>
## 1   276     1
## 2   550     1
## 3   626     1
```

```
write.csv(bmi_exclude, "../../data/final/bmisubjects_withonedatap.csv")
```

There are 3 subjects with 1 data point and has been saved to an output file.

Removing the 3 subjects from the processing BMI file by their ID.

```
dim(p_bmi)#checking the dimensions of p_bmi before removing
```

```
## [1] 10326     4
```

```
p_bmi <- p_bmi[!(p_bmi$ID_ %in% bmi_exclude$ID_), ] #removing by a logical vector where we want all the
dim(p_bmi)
```

```
## [1] 10323     4
```

Media exposure cleaning

Figuring out the 1 data point subjects and saving it them to an output file.

```
#Recalling the count by ID dataset from above for Media exposure.
head(media_timed)
```

```
## # A tibble: 6 × 2
##     ID_     n
##   <int> <int>
## 1     1     2
## 2     2     4
## 3     3     1
## 4     4     2
## 5     5     5
## 6     6     3
```

```
#Getting the indexes for which n = 1
#An assumption has been that for each row, there is no missing corresponding time value or media value.
media_exclude <- media_timed[media_timed$n==1,]
print(media_exclude)
```

```
## # A tibble: 100 × 2
##      ID_     n
##    <int> <int>
```

```
## 1      3     1
## 2     21     1
## 3     38     1
## 4     40     1
## 5     49     1
## 6     66     1
## 7     70     1
## 8     79     1
## 9     94     1
## 10    98     1
## # ... with 90 more rows
```

```
write.csv(media_exclude, "../../data/final/mediasubjects_withonedatap.csv")
```

There are 100 subjects in media exposure file that only has 1 data point.

Removing the 100 subjects from the processing BMI file by their ID.

```
dim(p_media)#checking the dimensions of p_bmi before removing
```

```
## [1] 1639    5
```

```
p_media <- p_media[!(p_media$ID_ %in% media_exclude$ID_), ] #removing by a logical vector where we want
dim(p_media)
```

```
## [1] 1539    5
```

Output

This is to build redundancy of data sets should code be accidentally changed etc. The data will be saved in the final data folder.

Output

bmi_clean_1 has removed 3 subjects with 1 data point media_clean_1 has removed 100 subjects with 1 data point

```
#write.csv(p_bmi,"../../data/final/bmi_clean_1.csv")
#write.csv(p_media, "../../data/final/media_clean_1.csv")
```

Merging the two data sets

Check the number of ID matches between the two files

The two ids to merge across the data sets are not of equal length. We will use the smaller one as base and see how much more are missing.

```
#Writing in the Media Exposure Smaller one first to fill in
matched_index <- match(p_media$ID_, p_bmi$ID_)
#Checking the number of non matches using is.na
non_matches <- sum(is.na(matched_index))
print(non_matches) #19
```

```
## [1] 19
```

```
#Checking total matches by substracting from maximum unique ID of smaller set to the missing ones
total_matches <- length(unique(p_media$ID_)) - non_matches
print(total_matches) #423
```

```
## [1] 423
```

```
#This is not perfect total number. The best way is to match it to match the IDs by innerjoin and captur
```

Join the two tables to fill in missing Xs and missing Ys for Each Subject

This is where we join by the patient ID to only those subjects that exist in both datasets. Therefore, we will only be using innerjoin. We will also be expanding BMI time points for where Media exposure exists and vice versa.

Step 1: Figure out all the IDs that match Step 2: Extract only those IDs from each table *Step 3: For each ID, expand by the row number by the total number of unique time intervals.* Step 4: For BMI, add in those missing time intervals, (sort in order) and expand for each subject ID. *Step 5: For Media, add in those missing time intervals,(sort in order) and expand for each subject ID.

```
p_bmi_media <- p_bmi %>% inner_join(p_media, by =c("ID_" = "ID_"))
```

Base Linear Interpolation Function

The linear interpolation equation to be used in the base function is below. The y_0 and y_1 would be either BMI or Media exposure variable. The x_0 and x_1 would be the time variable.

The y variable is the missing value we are looking for at time x. For BMI variable, the x corresponds to time from Media exposure that is missing between the x_0 and the x_1 intervals. The converse can be said of the Media Exposure variable to BMI as well.

Source: Linear Interpolation, Wikipedia

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0}$$

Use ApproxFun: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/approxfun.html

START: Testing Out Linear Interpolation approx/approxfun

Both approx and approxfun looks fairly similar. There are a couple of **key parameters** to consider * x,y => input vectors * xout => we specify which indexes we want to interpolate values for * yleft => this is specifying the last value to be carried to the left or backward if x values are less than min(x)

- yright => this is specifying the last value to be carried to the right or forward if x values are more than max(x)
- rule => Two options. 1 is to get NA for yleft, yright case. 2 is to output yleft, yright cases

Test Case 1

This is a simple case of some missing Ys with X values. A manual calculation is done below to verify the answer.

Simulated data 1

```
x_1 <- c(1,2,3,4,5,6)
y_1 <- c(3,NA,5,NA,NA,10)
xout_1 <- which(is.na(y_1)) #which returns the indexes where y_1 vector has NA values
```

Specifying y_left and y_right

This code chunk will tackle the case of last carried left/backward and last carried right/forward. The goal is to find the furthest left y index that is not NA and save the value. The same goes for the furthest right.

```
y_nmis_1 <- which(!is.na(y_1)) #indexes of non-missing y values
y_min_1 <- y_1[min(y_nmis_1)] #get the value from the furtherest left index of y
y_max_1 <- y_1[max(y_nmis_1)] #get the value from the furthest right index of y
```

Applying the function

This code chunk applies the function

```
out_1 <- approx(x_1, y_1, xout = xout_1,  method = "linear", yleft = y_min_1, yright = y_max_1, rule = 1
```

Interpolated results

```
print(out_1$y)
```

```
## [1] 4.000000 6.666667 8.333333
```

Manual calculation to confirm it.

Notetoself: In the future, helper functions should be in a separate source file. Seek permission from MS/DH.

Base interpolation helper function

```
#Note: Come back and write more comments later.

#The function takes in two pairs of point and the point you want to interpolate
lin_interpol <- function(y0,y1,x0,x1,x){
  y <- y0 + (x-x0) * ((y1-y0)/(x1-x0))
  return(y)
}
```

Manually outputting the three NA values from above

```
res_1_1 <- lin_interpol(3,5,1,3,2)
print(res_1_1)
```

```
## [1] 4
```

```
res_2_1 <- lin_interpol(5,10,3,6,4)
print(res_2_1)
```

```
## [1] 6.666667
```

```
res_3_1 <- lin_interpol(5,10,3,6,5)
print(res_3_1)
```

```
## [1] 8.333333
```

All the results matches up. We only have a case of Last Value Carried forward and backward to test

Test Case 2

Simulated data 2

We are testing the case of last value carried forward with 1 value missing on the left and 2 values missing on the right

```
x_2 <- c(1,2,3,4,5,6)
y_2 <- c(NA,3,5,10,NA,NA)
xout_2 <- which(is.na(y_2)) #which returns the indexes where y_1 vector has NA values
```

Same as above (Comments to merge or fill in later)

```
y_nmis_2 <- which(!is.na(y_2)) #indexes of non-missing y values
y_min_2 <- y_2[min(y_nmis_2)] #get the value from the furtherest left index of y
y_max_2 <- y_2[max(y_nmis_2)] #get the value from the furthest right index of y
```

This code chunk applies the function

```
out_2 <- approx(x_2, y_2, xout = xout_2,  method = "linear", yleft = y_min_2, yright = y_max_2, rule = 2
```

Interpolated results

```
print(out_2$y)
```

```
## [1]  3 10 10
```

Perfect. Left value carried forward and right value carried forward works like a charm.

END: Testing Out Linear Interpolation approx/approxfun

Applying approx to Curated Data Set

To Be Archived as R function is working as it should

Filling using Base Function above Function

```
#Parameters, X and Y values of each subject with missing NAs for the Y values

#Within function
## skip the first time point
## from the second time point and onwards
### if a missing NA is encountered for Y, go to the non-missing X and Y pair and the next one before.
### Calculate the time points in between.

#This could be much easily done if I use indexes of missing and non-missing.
##Have an index vector with the two X and Ys.
## Missing indexes can be two types
### It could be an index of count 1 and multiple
### For either case, pick the x0 and y0 and fill it up
```

Applying for Last Value Carried Forward Function

```
#Apply Last Value Carried Forward/Backward for the values
```

Applying to all the subjects function

Execution of the Functions

```
#All BMI subjects
```

```
#All Media Exposure subjects
```