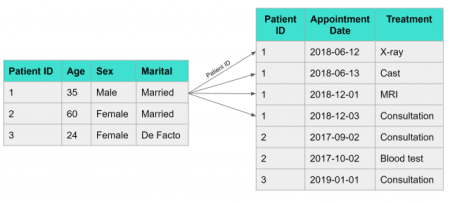


This post will build on the last post by tackling other complications when attempting to synthesise data. These challenges occur regularly in practice and this post will offer a couple of solutions, but there are plenty more. If you haven’t read my previous post, check that out first and swing back here. I’ll detail how more complex synthesis can be done using synthpop.

**Multi-grain**

When synthesising data, it is generally the finest grain which is synthesised. For population data the person is the finest grain, making synthesis relatively straight forward. But what happens when there are multiple tables to synthesise which are all linked by primary keys and have different grains finer than the person level? Lets look at an example.

Consider a health database which has two tables 1) a patient level table with demographic features and 2) an appointment level fact table consisting of the date of each appointment, including the diagnosis, treatment, prescriptions, etc, all the things you expect on a health record. In the second case the grain is patient x appointment.



Preserving this structure is important, otherwise there could be flow-on complications, plus there could be a loss of information. The obvious solution is to join the patient table onto the appointment table consolidating all the information, then synthesise. Right? Not always, for two key reasons,

1. Each synthesised row will essentially be a new patient and appointment date. We need to fix the simulated details for a patient and expand to the appointment level. It will be impossible to know how many appointments a patient has attended if the patient id is lost. This is a key feature of the data that needs to be preserved. Somehow each synthesised patient needs to have some number of synthesised appointment records.
2. The assumption of independence may no longer valid. This assumption is valid for the patient table as it’s reasonable to assume that one patient is not dependent on another in the dataset and therefore their treatments etc are also independent. At least in general, I’m sure we can think of scenarios where they are dependent e.g. family members, kids from the same school, etc. However, it’s reasonable to assume independence, fit a model, take random draws and synthesise the data items.

Some variables such as appointment dates should be treated as a sequence where the current appointment is dependent on the previous appointment date. The reason being, if a patient has an initial consultation, is diagnosed with some ailment e.g. fracture, fatigue, etc, it’s likely there are follow up appointments with the doctor for treatment. Synthesis should reflect this sequence.

Other examples where sequences should be preserved (not necessarily in the health domain) may include:

* + Pathology results of athletes taken at regular intervals
  + Messages sent and received by SMS (or some other chat) where the pattern may indicate a conversation
  + Weight measurements taken over time

or any other time series for that matter. It may not always be the case where the finest grain is a sequence, but it is likely to be. The key point is to know your data and what structure you need to preserve.

With these constraints the synthesised tables needs to retain the properties of properly synthesised data, meaning no variable is synthesised using the original data.

**Method**

Synthesising data at multiple grains can be done in a number of different ways. I’ll outline one particular way:

1. Use unsupervised learning to assign each patient to a cluster.
2. Synthesise the patient table including the cluster.
3. Randomly assign a synthesised patient to a real patients full appointment record within the same cluster (essentially stratified SRS- fancier methods could be used here).
4. Synthesise the remaining variables at the finer grains.

This requires some fancy footwork with synthpop but it can be done. There are many other ways to do this however where possible I opt for the path of least resistance.

**Construct the population data**

Appointment level data is first generated to use as original data. Key points,

1. Appointment date will be simulated to replicate the attendance pattern as outlined above. This pattern is likely to be correlated with other variables such as age and sex e.g. older people visit the doctor more. For this post 2 patterns will be simulated which are loosely correlated with age. Hidden Markov Models are used for the simulation. More complicated models such as state space models, ARIMA models or RNN’s can be used for simulation. (As a side note HMM’s are essentially a discrete form of state space models).
2. The cost of the appointment will be simulated from a multivariate normal distribution Y \sim N(\mathbf{\mu}, \mathbf{\Sigma})where \mathbf{\mu} = (100, 500, 1000)and \mathbf{\Sigma} = (75, 100, 125)^2(pulling some numbers out of the air) and truncated to 0 where appropriate. These are not correlated with any other variable (in the real world they would be but for lets keep it simple for now) and the assumption of independence is reasonable. This means we don’t have to do anything fancy with synthpop and simply use the built-in functions.

Dates are particularly annoying to deal with at the best of times and simulating them is no exception. Here is one way.

suppressPackageStartupMessages(library(synthpop))

suppressPackageStartupMessages(library(tidyverse))

suppressPackageStartupMessages(library(HMM))

suppressPackageStartupMessages(library(cluster))

suppressPackageStartupMessages(library(doParallel))

suppressPackageStartupMessages(library(foreach))

suppressPackageStartupMessages(library(mvtnorm))

suppressPackageStartupMessages(library(magrittr))

suppressPackageStartupMessages(library(gridExtra))

mycols <- c("darkmagenta", "turquoise")

myseed <- 20190201

# set data

patient.df <- SD2011 %>% dplyr::select(sex, age, socprof, income, marital, depress, sport, nofriend, smoke, nociga, alcabuse, bmi)

# set unique id

patient.df$patient\_id <- 1:nrow(patient.df)

head(patient.df)

## sex age socprof income marital depress sport nofriend smoke

## 1 FEMALE 57 RETIRED 800 MARRIED 6 NO 6 NO

## 2 MALE 20 PUPIL OR STUDENT 350 SINGLE 0 NO 4 NO

## 3 FEMALE 18 PUPIL OR STUDENT NA SINGLE 0 NO 20 NO

## 4 FEMALE 78 RETIRED 900 WIDOWED 16 YES 0 NO

## 5 FEMALE 54 SELF-EMPLOYED 1500 MARRIED 4 YES 6 YES

## 6 MALE 20 PUPIL OR STUDENT -8 SINGLE 5 NO 10 NO

## nociga alcabuse bmi patient\_id

## 1 -8 NO 30.79585 1

## 2 -8 NO 23.44934 2

## 3 -8 NO 18.36547 3

## 4 -8 NO 30.46875 4

## 5 20 NO 20.02884 5

## 6 -8 NO 23.87511 6

# patient group

low <- sample(1:2, nrow(patient.df), replace = TRUE, prob = c(0.25, 0.75))

high <- sample(1:2, nrow(patient.df), replace = TRUE, prob = c(0.75, 0.25))

patient.df$group <- ifelse(patient.df$age < median(patient.df$age), low, high)

# set my HMM simulation function

mysimHMM <- function (hmm, num.events, which.event = NULL){

hmm$transProbs[is.na(hmm$transProbs)] = 0

hmm$emissionProbs[is.na(hmm$emissionProbs)] = 0

states = c()

emission = c()

states = c(states, sample(hmm$States, 1, prob = hmm$startProbs))

i <- 2

if(is.null(which.event)) which.event <- hmm$Symbols[1]

while(sum(emission == which.event) < num.events) {

state = sample(hmm$States, 1, prob = hmm$transProbs[states[i-1], ])

states = c(states, state)

emi = sample(hmm$Symbols, 1, prob = hmm$emissionProbs[states[i-1], ])

emission = c(emission, emi)

i <- i + 1

}

return(list(states = states, observation = emission))

}

# initialise HMM

hmm.list <- list(initHMM(States = c("episode", "no episode"),

Symbols = c("Y", "N"),

transProbs = matrix(rbind(

c(0.92, 0.08),

c(0.02, 0.98)

)),

emissionProbs = matrix(rbind(

c(0.25, 0.75),

c(0.001, 0.999)

))),

initHMM(States = c("episode", "no episode"),

Symbols = c("Y", "N"),

transProbs = matrix(rbind(

c(0.92, 0.08),

c(0.1, 0.9)

)),

emissionProbs = matrix(rbind(

c(0.25, 0.75),

c(0.001, 0.999)

))))

# appointments start dates will be contained within a 3 year window

start.dates <- seq(as.Date("2014-01-01"), as.Date("2016-12-31"), 1)

simulate.appointment.dates <- function(){

n <- rpois(1, rgamma(1, 1)\*12) + 1

appointment.dates <- mysimHMM(hmm.list[[patient.df$group[k]]], n)$observation

l <- length(appointment.dates)

dts <- sample(start.dates, 1) + 0:(l-1)

appointment\_dates = dts[which(appointment.dates == "Y")]

out <- data.frame(patient\_id = rep(k, length = n), appointment\_dates,

cost = rmvnorm(n, mean = c(100, 500, 1000), sigma = diag(c(75, 100, 125)^2)) %>% sample(n) %>% pmax(0))

return(out)

}

ncores <- min(detectCores(), 4)

registerDoParallel(ncores)

appointment.df <- foreach(k = 1:nrow(patient.df), .combine = rbind, .export = "rmvnorm") %dopar% simulate.appointment.dates()

appointment.df %<>% dplyr::filter(appointment\_dates < as.Date("2019-01-31"))

Plotting the sequence of appointment dates, we can see the structure we want to preserve.

# look at the clustering of appointments

df <- appointment.df %>% dplyr::filter(patient\_id < 7)

g.date.sequences <- ggplot(df, aes(x = appointment\_dates, y = patient\_id)) +

geom\_hline(yintercept = 1:6, col = "grey80") +

geom\_point(col = "turquoise", size = 4) +

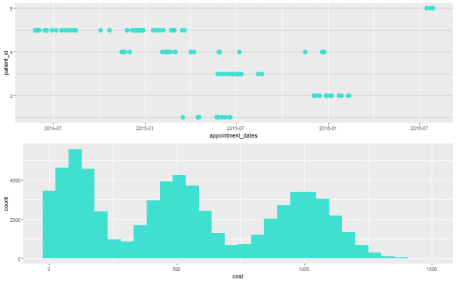
coord\_cartesian(xlim = range(df$appointment\_dates))

# costs

g.costs <- ggplot(appointment.df, aes(x = cost)) + geom\_histogram(fill = "turquoise")

# plot them

grid.arrange(g.date.sequences, g.costs)



**Cluster the patient data and create new link**

To create a link between the patient level data and the appointment level data the patient table is clustered using a simple hierarchical clustering procedure (it can be whatever method you think is suitable but I’m choosing hierarchical for this example).

Here 20 clusters are defined. In practice, more thought may need to go into how many clusters are defined given the problem. The clusters are joined to the appointment table to create the link.

# --------------- CLUSTER PATIENTS TO EXPAND TO APPOINTMENT LEVEL

dsim <- patient.df %>%

dplyr::select(-patient\_id) %>%

dist()

clust <- hclust(dsim, method = "ward.D2")

nc <- 20

cut.clust <- cutree(clust, nc)

table(cut.clust)

## cut.clust

## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

## 385 179 800 626 569 532 464 277 349 92 60 35 165 199 74 4 168 11

## 19 20

## 8 3

# put cluster on patient table

patient.df$cluster <- cut.clust

appointment.df %<>% left\_join(patient.df %>% dplyr::select(patient\_id, cluster, group), by = "patient\_id")

**Synthesis**

**Synthesise patient records**

The patient table is now synthesised, including the clusters and a new patient id created.

# --------------- SYNTHESISE PATIENT DATA FRAME

p.df <- patient.df %>% dplyr::select(-patient\_id, -group)

# apply rules to ensure consistency

rules.list <- list(marital = "age < 18", nociga = "smoke == 'NO'")

rules.value.list <- list(marital = "SINGLE", nociga = -8)

# setting continuous variable NA list

cont.na.list <- list(income = c(NA, -8), nofriend = c(NA, -8), nociga = c(NA, -8))

# synth

patient.synth <- syn(p.df, cont.na = cont.na.list, rules = rules.list, rvalues = rules.value.list)$syn

##

## Unexpected values (not obeying the rules) found for variable(s): nociga.

## Rules have been applied but make sure they are correct.

## Synthesis

## -----------

## sex age socprof income marital depress sport nofriend smoke nociga

## alcabuse bmi cluster

# --------------- NEW PATIENT ID

patient.synth$patient\_id <- 1:nrow(patient.synth)

head(patient.synth)

## sex age socprof income marital depress sport

## 1 FEMALE 31 EMPLOYED IN PUBLIC SECTOR 1240 MARRIED 4 YES

## 2 FEMALE 49 OTHER ECONOMICALLY INACTIVE 600 MARRIED 7 YES

## 3 FEMALE 20 OTHER ECONOMICALLY INACTIVE -8 SINGLE 0 YES

## 4 FEMALE 26 400 SINGLE 4 NO

## 5 MALE 27 OTHER ECONOMICALLY INACTIVE 900 SINGLE 0 YES

## 6 FEMALE 53 EMPLOYED IN PRIVATE SECTOR -8 SINGLE 5 YES

## nofriend smoke nociga alcabuse bmi cluster patient\_id

## 1 3 NO -8 NO 20.06920 7 1

## 2 20 YES 10 NO 25.14861 8 2

## 3 10 NO -8 NO 18.02596 3 3

## 4 5 NO -8 NO 22.49135 2 4

## 5 7 YES 20 NO 23.56663 4 5

## 6 2 YES 10 NO 26.53376 3 6

The next step is to randomly select a true patient id from within the same cluster. By doing so each synthesised patient gets a random patient’s appointment history. Since the history is randomly selected within the same cluster we can be confident the structure of the data is retained e.g. a 50 year old male that doesn’t smoke gets a history which is realistic and supported by the data.

# --------------- SAMPLE PATIENT ID WITHIN CLUSTER IN ORDER TO LINK TO APPOINTMENT DATASET

sample.patient <- function(x){

z <- which(patient.df$cluster == x)

n <- length(which(patient.synth$cluster == x))

out <- sample(patient.df$patient\_id[z], n, replace = TRUE)

return(out)

}

patient.list <- lapply(as.list(1:nc), sample.patient)

patient.synth$link.patient <- NA

for(k in 1:nc){

z <- which(patient.synth$cluster == k)

patient.synth$link.patient[z] <- patient.list[[k]]

}

# -------------- LINK TO APPOINTMENT

patient.synth %<>%

left\_join(appointment.df %>% dplyr::select(-cluster), by = c("link.patient" = "patient\_id")) %>%

arrange(patient\_id, appointment\_dates)

head(patient.synth %>% dplyr::select(patient\_id, sex, age, socprof, appointment\_dates, cost, cluster)) # only displaying a few variables for convenience

## patient\_id sex age socprof appointment\_dates

## 1 1 FEMALE 31 EMPLOYED IN PUBLIC SECTOR 2015-06-18

## 2 2 FEMALE 49 OTHER ECONOMICALLY INACTIVE 2016-04-26

## 3 2 FEMALE 49 OTHER ECONOMICALLY INACTIVE 2016-05-30

## 4 2 FEMALE 49 OTHER ECONOMICALLY INACTIVE 2016-06-02

## 5 2 FEMALE 49 OTHER ECONOMICALLY INACTIVE 2016-07-02

## 6 2 FEMALE 49 OTHER ECONOMICALLY INACTIVE 2016-07-03

## cost cluster

## 1 1063.82122 7

## 2 380.20829 8

## 3 492.43475 8

## 4 1085.44437 8

## 5 1038.01058 8

## 6 68.89496 8

**Synthesise appointment records**

This data is suitable to be simulated using HMM’s (we know this for sure since we created the data set!). A new function is written to be consumed by syn() and the data synthesised during the same process.

**Training your models**

This is where things can get tricky and it is important to have a good handle on your data. To synthesise the sequence data, a model needs to be trained on the original data first. In practice syn() takes care of the training in most situations. Either the function is defined to train the model before synthesis or consume pre-trained models. This will depend on the problem at hand.

I am going to skip the training step here. Training HMM’s (or any state space models) can be tricky and require careful tuning. Instead I am simply going to use the predefined HMM’s above that were used to create the original data. The primary aim is to demonstrate how to synthesise the data rather than train HMM’s.

**The syn.hmm() function**

The function to pass to syn() is given below. The list of HMM’s is passed directly to the function which returns an array of dates.

syn.hmm <- function (y, x, xp, id = patient.synth$link.patient, hmm\_list = hmm.list, group = patient.synth$group, ...){

start.dates <- seq(as.Date("2014-01-01"), as.Date("2016-12-31"), 1)

uniq.id <- unique(id)

n <- length(uniq.id)

gr <- unique(cbind(id, group))

new.y <- vector(mode = "numeric", length = nrow(xp))

for(k in 1:n){

ii <- which(id == uniq.id[k])

gr <- group[k]

obs <- mysimHMM(hmm = hmm\_list[[gr]], length(ii))$observation

l <- length(obs)

dts <- sample(start.dates, 1) + 0:(l-1)

new.y[ii] <- dts[which(obs == "Y")]

}

return(list(res = as.Date(new.y, origin = "1970-01-01"), fit = "hmm"))

}

**The final synthesis**

To synthesise only the appointment dates and cost but ignore the remaining variables, we define the predictor matrix and methods vector.

Firstly, syn() will switch the method to “sample” if the yvariable does not have a predictor. To trick the function into doing the what we want we can set the appointment dates variable to have at least 1 predictor which it won’t use because we haven’t told it to in syn.hmm().

Secondly, cost can be synthesised as normal and for this example is synthesised using age, sex and income for demonstration.

Thirdly, to tell the function to ignore all other variables we set the visit sequence to be appointment dates followed by cost and nothing else. It will ignore the other already synthesised variables and attach them to the data frame.

# set data frame

df <- patient.synth %>% dplyr::select(-cluster, -group)

synth.obj <- syn(df, method = rep("sample", ncol(df)), m = 0)

# create new predictor matrix

new.pm <- synth.obj$predictor.matrix

new.pm["cost", c("sex", "age", "income")] <- 1

new.pm["appointment\_dates", c("link.patient")] <- 1

# set visit sequences

vseq <- c(which(colnames(new.pm) == "appointment\_dates"), which(colnames(new.pm) == "cost"))

# set methods array

mth <- rep("", ncol(new.pm))

mth[vseq] <- c("hmm", "cart")

# synthesise data

synth.obj <- syn(df, method = mth, visit.sequence = vseq, predictor.matrix = new.pm)

##

## Variable(s): socprof, marital, depress, sport, nofriend, smoke, nociga, alcabuse, bmi, patient\_id not synthesised or used in prediction.

## CAUTION: The synthesised data will contain the variable(s) unchanged.

##

## Synthesis

## -----------

## appointment\_dates cost

head(synth.obj$syn %>% dplyr::select(sex, age, income, appointment\_dates, cost)) # only displaying a few variables for convenience

## sex age income appointment\_dates cost

## 1 FEMALE 31 1240 2016-04-30 74.43372

## 2 FEMALE 49 600 2016-06-25 68.89496

## 3 FEMALE 49 600 2016-07-27 475.31540

## 4 FEMALE 49 600 2016-07-28 126.08026

## 5 FEMALE 49 600 2016-11-03 1009.78769

## 6 FEMALE 49 600 2016-11-04 1038.01058

synth.obj$method

## sex age socprof income

## "" "" "" ""

## marital depress sport nofriend

## "" "" "" ""

## smoke nociga alcabuse bmi

## "" "" "" ""

## patient\_id link.patient appointment\_dates cost

## "" "" "hmm" "cart"

synth.obj$visit.sequence

## appointment\_dates cost

## 15 16

# compare cost

# look at the clustering of appointments

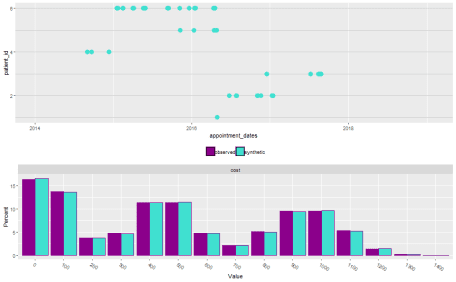
g.date.sequences <- ggplot(synth.obj$syn %>% dplyr::filter(patient\_id < 7), aes(x = appointment\_dates, y = patient\_id)) +

geom\_hline(yintercept = 1:6, col = "grey80") +

geom\_point(col = "turquoise", size = 4) +

coord\_cartesian(xlim = range(df$appointment\_dates))

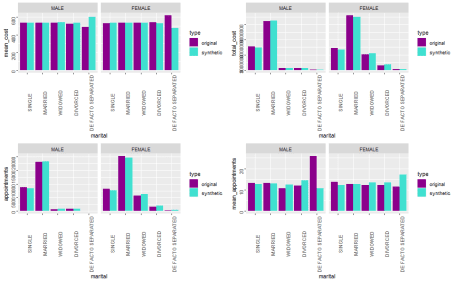
grid.arrange(g.date.sequences, compare(synth.obj, df, vars = "cost", cols = mycols)$plot)



Again, a solid effort. Appointment dates display a similar patterns and the cost variable matches the original distributions.

**Comparing outputs**

The plots below show the comparison between the synthetic and original data for the mean and total cost, appointment count and mean appointment count per patient. All given by sex and marital status.



The outputs are very similar with the exception of the mean number of appointments for those in de facto relationships. This is likely a property of the small sample size in this group.

**Takeaways**

When synthesising data from any relational database these challenges will be present. There will be a business need to preserve the same relational form to ensure the synthetic data is used in the same way as the original.

There are many ways of dealing with the problem of different grains and synthesising sequences – this is only one. With respect to sequences and time series data, careful thought is needed to use the right model for synthesis and define the appropriate function.

This example is relatively straight forward to highlight a couple of key challenges. The synthpop package is flexible enough to handle these more complex real world scenarios.

If you have solved this in other ways, let me know in the comments!