Sequence Analysis: Time Use Data (ATUS-X)

Kamila Kolpashnikova*

Introduction

Traditionally, a tool for analysing life events in social sciences, sequence analysis is becoming more and more popular for dealing with time-use data. In this tutorial, I'll walk you through the steps how to do sequence analysis with time-use data from IPUMS (ATUS-X).

To follow this tutotial:

- change the directory to the location of cloned directory
- follow instructions in the code (R file is included)

Funding: This project has received funding from the CFREF-2015-00013: VISTA Postdoctoral Fellowship.

Example of Work Using Sequence Analysis

Here is an example how a published paper using sequence analysis looks like (my recent research in the Journal of Population Ageing):

Kolpashnikova, K., & Kan, M. Y. (2020). Eldercare in Japan: Cluster Analysis of daily time-use patterns of elder caregivers. Journal of Population Ageing 14(4), 441-463.

If you follow this tutorial, you will be able to recreate the analyses in the paper (but will be using another dataset – ATUS-X)

Loading Necessary Packages

```
#### Loading Necessary Packages ####
if (!require("pacman")) install.packages("pacman")
```

```
## Loading required package: pacman
```

```
## Warning: package 'pacman' was built under R version 4.1.3
```

```
library(pacman)

# load and install packages
pacman::p_load(TraMineR, TraMineRextras, downloader, cluster, RColorBrewer, devtools, haven,
               tidyverse, reshape2, WeightedCluster, nnet, plyr, ggplot2, ipumsr)
```

*York University

Load Datasets from the BLS website

Remember that in r, it's forward slashes. Here I create dataframes using the package called ipumsr to transform data from IPUMS.You can find explanations on how to use this package here.

```
# NOTE: To load data, you must download both the extract's data and the DDI
# and also set the working directory to the folder with these files (or change the path below).

if (!require("ipumsr")) stop("Reading IPUMS data into R requires the ipumsr package.
                             It can be installed using the following command:
                             install.packages('ipumsr')")

ddi <- read_ipums_ddi("data/atus_00003.xml")
df_ec <- read_ipums_micro(ddi)
```

```
## Use of data from IPUMS ATUS is subject to conditions including that users
## should cite the data appropriately. Use command 'ipums_conditions()' for more
## details.
```

```
head(df_ec)
```

```
## # A tibble: 6 x 45
##    YEAR  CASEID SERIAL REGION    STATEFIP  PERNUM LINENO MONTH   DAY     HOLIDAY
##   <dbl>   <dbl>  <dbl> <int+lbl> <int+lbl>  <int> <int+> <int+l> <int+l> <int+l>
## 1  2011 2.01e13      1 3 [South] 45 [Sout~      1 1      1 [Jan~ 7 [Sat~ 0 [No]
## 2  2011 2.01e13      1 3 [South] 45 [Sout~      1 1      1 [Jan~ 7 [Sat~ 0 [No]
## 3  2011 2.01e13      1 3 [South] 45 [Sout~      1 1      1 [Jan~ 7 [Sat~ 0 [No]
## 4  2011 2.01e13      1 3 [South] 45 [Sout~      1 1      1 [Jan~ 7 [Sat~ 0 [No]
## 5  2011 2.01e13      1 3 [South] 45 [Sout~      1 1      1 [Jan~ 7 [Sat~ 0 [No]
## 6  2011 2.01e13      1 3 [South] 45 [Sout~      1 1      1 [Jan~ 7 [Sat~ 0 [No]
## # ... with 35 more variables: WT06 <dbl>, AGE <dbl+lbl>, SEX <int+lbl>,
## #   RACE <int+lbl>, HISPAN <int+lbl>, MARST <int+lbl>, KIDUND18 <int+lbl>,
## #   HH_NUMOWNKIDS <int+lbl>, ECYEST <int+lbl>, ECPRIOR <int+lbl>,
## #   ECNUM <int+lbl>, ECFREQ <int+lbl>, ACTLINE <dbl>, ACTIVITY <int+lbl>,
## #   DURATION_EXT <dbl>, DURATION <dbl>, START <chr>, STOP <chr>,
## #   ACT_CAREHH <dbl>, ACT_CARENHH <dbl>, ACT_EDUC <dbl>, ACT_FOOD <dbl>,
## #   ACT_GOVSERV <dbl>, ACT_HHACT <dbl>, ACT_HHSERV <dbl>, ACT_PCARE <dbl>, ...
```

Filter the elder caregivers and recode activities

Before we create our sequence dataframes, we need to wrangle the original data to make it usable.

```
## cleaning the data and choosing only diaries
## you can merge these diaries later with socio-dem variables if needed
for( i in colnames(df_ec)){
  colnames(df_ec)[which(colnames(df_ec)==i)] = tolower(i)
}

df_ec <- df_ec[c('caseid', 'actline', 'activity', 'duration', 'start', 'stop', 'wt06')]

diary = df_ec[complete.cases(df_ec), ]
head(diary)
```

```
## # A tibble: 6 x 7
##    caseid actline activity                                durat~1 start stop    wt06
##     <dbl>   <dbl> <int+lbl>                                 <dbl> <chr> <chr>  <dbl>
## 1 2.01e13       1  10101 [Sleeping]                          330 04:0~ 09:3~ 1.42e6
## 2 2.01e13       2  20201 [Food and drink preparation]        20 09:3~ 09:5~ 1.42e6
## 3 2.01e13       3 110101 [Eating and drinking]               10 09:5~ 10:0~ 1.42e6
## 4 2.01e13       4 120303 [Television and movies (not~        90 10:0~ 11:3~ 1.42e6
## 5 2.01e13       5  10201 [Washing, dressing and groo~       30 11:3~ 12:0~ 1.42e6
## 6 2.01e13       6 120303 [Television and movies (not~      120 12:0~ 14:0~ 1.42e6
## # ... with abbreviated variable name 1: duration
```

```
## create activity dictionary
## this list contains information to reducing activity coding to 11 main categories
## (listed below this)
## you can rename the original categories yourself, but you have to figure it out on your own
act_dict <-  c("10101" = 1, "10102" = 1, "10199" = 1, "10201" = 2,
               "10299" = 2, "10301" = 2, "10399" = 2, "10401" = 2, "10499" = 2,
               "10501" = 2, "10599" = 2, "19999" = 2, "20101" = 3, "20102" = 3,
               "20103" = 3, "20104" = 3, "20199" = 3, "20201" = 3, "20202" = 3,
               "20203" = 3, "20299" = 3, "20301" = 3, "20302" = 3, "20303" = 3,
               "20399" = 3, "20400" = 3, "20401" = 3, "20402" = 3, "20499" = 3,
               "20500" = 3, "20501" = 3, "20502" = 3, "20599" = 3, "20600" = 3,
               "20601" = 3, "20602" = 3, "20603" = 3, "20681" = 10, "20699" = 3,
               "20700" = 3, "20701" = 3, "20799" = 3, "20800" = 3, "20801" = 3,
               "20899" = 3, "20900" = 3, "20901" = 3, "20902" = 3, "20903" = 3,
               "20904" = 3, "20905" = 3, "20999" = 3, "29900" = 3, "29999" = 3,
               "30100" = 4, "30101" = 4, "30102" = 4, "30103" = 4, "30104" = 4,
               "30105" = 4, "30106" = 4, "30107" = 4, "30108" = 4, "30109" = 4,
               "30110" = 4, "30111" = 4, "30112" = 4, "30199" = 4, "30200" = 4,
               "30201" = 4, "30202" = 4, "30203" = 4, "30204" = 4, "30299" = 4,
               "30300" = 4, "30301" = 4, "30302" = 4, "30303" = 4, "30399" = 4,
               "40100" = 4, "40101" = 4, "40102" = 4, "40103" = 4, "40104" = 4,
               "40105" = 4, "40106" = 4, "40107" = 4, "40108" = 4, "40109" = 4,
               "40110" = 4, "40111" = 4, "40112" = 4, "40199" = 4, "40200" = 4,
               "40201" = 4, "40202" = 4, "40203" = 4, "40204" = 4, "40299" = 4,
               "40300" = 4, "40301" = 4, "40302" = 4, "40303" = 4, "40399" = 4,
               "30186" = 4, "40186" = 4, "30000" = 5, "30400" = 5, "30401" = 5,
               "30402" = 5, "30403" = 5, "30404" = 5, "30405" = 5, "30499" = 5,
               "30500" = 5, "30501" = 5, "30502" = 5, "30503" = 5, "30504" = 5,
               "30599" = 5, "39900" = 5, "39999" = 5, "40000" = 5, "40400" = 5,
               "40401" = 5, "40402" = 5, "40403" = 5, "40404" = 5, "40405" = 5,
               "40499" = 5, "40500" = 5, "40501" = 5, "40502" = 5, "40503" = 5,
               "40504" = 5, "40505" = 5, "40506" = 5, "40507" = 5, "40508" = 5,
               "40599" = 5, "49900" = 5, "49999" = 5, "50000" = 6, "50100" = 6,
               "50101" = 6, "50102" = 6, "50103" = 6, "50104" = 6, "50199" = 6,
               "50200" = 6, "50201" = 6, "50202" = 6, "50203" = 6, "50204" = 6,
               "50205" = 6, "50299" = 6, "50300" = 6, "50301" = 6, "50302" = 6,
               "50303" = 6, "50304" = 6, "50305" = 6, "50399" = 6, "50400" = 6,
               "50401" = 6, "50403" = 6, "50404" = 6, "50405" = 6, "50499" = 6,
               "59900" = 6, "59999" = 6, "60000" = 6, "60100" = 6, "60101" = 6,
               "60102" = 6, "60103" = 6, "60104" = 6, "60199" = 6, "60200" = 6,
               "60201" = 6, "60202" = 6, "60203" = 6, "60204" = 6, "60299" = 6,
               "60300" = 6, "60301" = 6, "60302" = 6, "60303" = 6, "60399" = 6,
               "60400" = 6, "60401" = 6, "60402" = 6, "60403" = 6, "60499" = 6,
```

```
            "69900" = 6, "69999" = 6, "50481" = 6, "50389" = 6, "50189" = 6,
            "60289" = 6, "50289" = 6, "70000" = 7, "70100" = 7, "70101" = 7,
            "70102" = 7, "70103" = 7, "70104" = 7, "70105" = 7, "70199" = 7,
            "70200" = 7, "70201" = 7, "70299" = 7, "70300" = 7, "70301" = 7,
            "70399" = 7, "79900" = 7, "79999" = 7, "80000" = 7, "80100" = 7,
            "80101" = 7, "80102" = 7, "80199" = 7, "80200" = 7, "80201" = 7,
            "80202" = 7, "80203" = 7, "80299" = 7, "80300" = 7, "80301" = 7,
            "80302" = 7, "80399" = 7, "80400" = 7, "80401" = 7, "80402" = 7,
            "80403" = 7, "80499" = 7, "80500" = 7, "80501" = 7, "80502" = 7,
            "80599" = 7, "80600" = 7, "80601" = 7, "80602" = 7, "80699" = 7,
            "80700" = 7, "80701" = 7, "80702" = 7, "80799" = 7, "80800" = 7,
            "80801" = 7, "80899" = 7, "89900" = 7, "89999" = 7, "90000" = 7,
            "90100" = 7, "90101" = 7, "90102" = 7, "90103" = 7, "90104" = 7,
            "90199" = 7, "90200" = 7, "90201" = 7, "90202" = 7, "90299" = 7,
            "90300" = 7, "90301" = 7, "90302" = 7, "90399" = 7, "90400" = 7,
            "90401" = 7, "90402" = 7, "90499" = 7, "90500" = 7, "90501" = 7,
            "90502" = 7, "90599" = 7, "99900" = 7, "99999" = 7, "100000" = 7,
            "100100" = 7, "100101" = 7, "100102" = 7, "100103" = 7, "100199" = 7,
            "100200" = 7, "100201" = 7, "100299" = 7, "100300" = 7, "100303" = 7,
            "100304" = 7, "100399" = 7, "100400" = 7, "100401" = 7, "100499" = 7,
            "109900" = 7, "109999" = 7, "120303" = 8, "120304" = 8, "110000" = 9,
            "110100" = 9, "110101" = 9, "110199" = 9, "110200" = 9, "110201" = 9,
            "110299" = 9, "119900" = 9, "110289" = 9, "119999" = 9, "120000" = 10,
            "120100" = 10, "120101" = 10, "120199" = 10, "120200" = 10, "120201" = 10,
            "120202" = 10, "120299" = 10, "120300" = 10, "120301" = 10, "120302" = 10,
            "120305" = 10, "120306" = 10, "120307" = 10, "120308" = 10, "120309" = 10,
            "120310" = 10, "120311" = 10, "120312" = 10, "120313" = 10, "120399" = 10,
            "120400" = 10, "120401" = 10, "120402" = 10, "120403" = 10, "120404" = 10,
            "120405" = 10, "120499" = 10, "120500" = 10, "120501" = 10, "120502" = 10,
            "120503" = 10, "120504" = 10, "120599" = 10, "129900" = 10, "129999" = 10,
            "130000" = 10, "130100" = 10, "130101" = 10, "130102" = 10, "130103" = 10,
            "130104" = 10, "130105" = 10, "130106" = 10, "130107" = 10, "130108" = 10,
            "130109" = 10, "130110" = 10, "130111" = 10, "130112" = 10, "130113" = 10,
            "130114" = 10, "130115" = 10, "130116" = 10, "130117" = 10, "130118" = 10,
            "130119" = 10, "130120" = 10, "130121" = 10, "130122" = 10, "130123" = 10,
            "130124" = 10, "130125" = 10, "130126" = 10, "130127" = 10, "130128" = 10,
            "130129" = 10, "130130" = 10, "130131" = 10, "130132" = 10, "130133" = 10,
            "130134" = 10, "130135" = 10, "130136" = 10, "130199" = 10, "130200" = 10,
            "130201" = 10, "130202" = 10, "130203" = 10, "130204" = 10, "130205" = 10,
            "130206" = 10, "130207" = 10, "130208" = 10, "130209" = 10, "130210" = 10,
            "130211" = 10, "130212" = 10, "130213" = 10, "130214" = 10, "130215" = 10,
            "130216" = 10, "130217" = 10, "130218" = 10, "130219" = 10, "130220" = 10,
            "130221" = 10, "130222" = 10, "130223" = 10, "130224" = 10, "130225" = 10,
            "130226" = 10, "130227" = 10, "130228" = 10, "130229" = 10, "130230" = 10,
            "130231" = 10, "130232" = 10, "130299" = 10, "130300" = 10, "130301" = 10,
            "130302" = 10, "130399" = 10, "130400" = 10, "130401" = 10, "130402" = 10,
            "130499" = 10, "139900" = 10, "139999" = 10, "140000" = 10, "140100" = 10,
            "140101" = 10, "140102" = 10, "140103" = 10, "140104" = 10, "140105" = 10,
            "149900" = 10, "149999" = 10, "150000" = 10, "150100" = 10, "150101" = 10,
            "150102" = 10, "150103" = 10, "150104" = 10, "150105" = 10, "150106" = 10,
            "150199" = 10, "150200" = 10, "150201" = 10, "150202" = 10, "150203" = 10,
            "150204" = 10, "150299" = 10, "150300" = 10, "150301" = 10, "150302" = 10,
            "150399" = 10, "150400" = 10, "150401" = 10, "150402" = 10, "150499" = 10,
```

```
               "150500" = 10, "150501" = 10, "150599" = 10, "150600" = 10, "150601" = 10,
               "150602" = 10, "150699" = 10, "150700" = 10, "150701" = 10, "150799" = 10,
               "150800" = 10, "150801" = 10, "150899" = 10, "159900" = 10, "159999" = 10,
               "160000" = 10, "160100" = 10, "160101" = 10, "160102" = 10, "160103" = 10,
               "160104" = 10, "160105" = 10, "160106" = 10, "160107" = 10, "160108" = 10,
               "160199" = 10, "160200" = 10, "160201" = 10, "160299" = 10, "169900" = 10,
               "169999" = 10, "159989" = 10, "169989" = 10, "110281" = 10, "100381" = 10,
               "100383" = 10, "180000" = 11, "180100" = 11, "180101" = 11, "180199" = 11,
               "180200" = 11, "180201" = 11, "180202" = 11, "180203" = 11, "180204" = 11,
               "180205" = 11, "180206" = 11, "180207" = 11, "180208" = 11, "180209" = 11,
               "180280" = 11, "180299" = 11, "180300" = 11, "180301" = 11, "180302" = 11,
               "180303" = 11, "180304" = 11, "180305" = 11, "180306" = 11, "180307" = 11,
               "180399" = 11, "180400" = 11, "180401" = 11, "180402" = 11, "180403" = 11,
               "180404" = 11, "180405" = 11, "180406" = 11, "180407" = 11, "180482" = 11,
               "180499" = 11, "180500" = 11, "180501" = 11, "180502" = 11, "180503" = 11,
               "180504" = 11, "180599" = 11, "180600" = 11, "180601" = 11, "180602" = 11,
               "180603" = 11, "180604" = 11, "180605" = 11, "180699" = 11, "180700" = 11,
               "180701" = 11, "180702" = 11, "180703" = 11, "180704" = 11, "180705" = 11,
               "180782" = 11, "180799" = 11, "180800" = 11, "180801" = 11, "180802" = 11,
               "180803" = 11, "180804" = 11, "180805" = 11, "180806" = 11, "180807" = 11,
               "180899" = 11, "180900" = 11, "180901" = 11, "180902" = 11, "180903" = 11,
               "180904" = 11, "180905" = 11, "180999" = 11, "181000" = 11, "181001" = 11,
               "181002" = 11, "181099" = 11, "181100" = 11, "181101" = 11, "181199" = 11,
               "181200" = 11, "181201" = 11, "181202" = 11, "181203" = 11, "181204" = 11,
               "181205" = 11, "181206" = 11, "181283" = 11, "181299" = 11, "181300" = 11,
               "181301" = 11, "181302" = 11, "181399" = 11, "181400" = 11, "181401" = 11,
               "181499" = 11, "181500" = 11, "181501" = 11, "181599" = 11, "181600" = 11,
               "181601" = 11, "181699" = 11, "181800" = 11, "181801" = 11, "181899" = 11,
               "189900" = 11, "189999" = 11, "180481" = 11, "180381" = 11, "180382" = 11,
               "181081" = 11, "180589" = 11, "180682" = 11, "500000" = 11, "500100" = 11,
               "500101" = 11, "500102" = 11, "500103" = 11, "500104" = 11, "500105" = 11,
               "500106" = 11, "500107" = 11, "509900" = 11, "509989" = 11, "509999" = 11)


## this list is not used but it contains the codes for the 11 main activities
act = c("1" = "Sleep",
        "2" = "Personal Care",
        "3" = "Housework",
        "4" = "Child Care",
        "5" = "Adult Care",
        "6" = "Work and Education",
        "7" = "Shopping",
        "8" = "TV Watching",
        "9" = "Eating",
        "10" = "Leisure",
        "11" = "Travel and Other")

### recode the activities using the activity dictionary
diary$activity_i <- as.factor(diary$activity)
diary$lst_act <- revalue(diary$activity_i, act_dict)
```

```
## The following `from` values were not present in `x`: 10599, 20400, 20500, 20600, 20601, 20681, 20700
```

```
head(diary)
```

```
## # A tibble: 6 x 9
##    caseid actline activity            durat~1 start stop    wt06 activ~2 lst_act
##     <dbl>   <dbl> <int+lbl>             <dbl> <chr> <chr>  <dbl> <fct>   <fct>
## 1 2.01e13       1  10101 [Sleeping]       330 04:0~ 09:3~ 1.42e6 10101   1
## 2 2.01e13       2  20201 [Food and d~      20 09:3~ 09:5~ 1.42e6 20201   3
## 3 2.01e13       3 110101 [Eating and~      10 09:5~ 10:0~ 1.42e6 110101  9
## 4 2.01e13       4 120303 [Television~      90 10:0~ 11:3~ 1.42e6 120303  8
## 5 2.01e13       5  10201 [Washing, d~      30 11:3~ 12:0~ 1.42e6 10201   2
## 6 2.01e13       6 120303 [Television~     120 12:0~ 14:0~ 1.42e6 120303  8
## # ... with abbreviated variable names 1: duration, 2: activity_i
```

```
## print out unique values for activities
unique(diary$lst_act)
```

```
##  [1] 1  3  9  8  2  11 7  10 4  6  5
## Levels: 1 2 3 4 5 6 7 9 10 8 11
```

```
## rename some of the activities even to fewer categories
## the activity dictionary created above gives 11 categories of activities
## in case some of the activities need to be combined
## you can follow the same logic in the following lines:
diary$lst_act = revalue(diary$lst_act, c("2" = 1, "8" = 10, "9" = 10))
unique(diary$lst_act)
```

```
## [1] 1  3  10 11 7  4  6  5
## Levels: 1 3 4 5 6 7 10 11
```

Then, let's create sequence dataframe, specify column names, and add ids.

```
## create sequences of activities per activity (result: long list of combined sequences)
sequences = rep(diary$lst_act, diary$duration)

## separate the long list into sequences (1440 min in each sequence)
seq = matrix(sequences, nrow=length(sequences)/1440, ncol=1440, byrow=T)
seq[1:5, 1:10]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,] "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"
## [2,] "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"
## [3,] "10" "10" "10" "10" "10" "10" "10" "10" "10" "10"
## [4,] "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"
## [5,] "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"
```

```
## specify the names for the activity variables
activities<-c()
for(i in 0:1439) {
  activities<-c(activities, paste("var", i, sep = ""))
}
```

```
## transform matrix to dataframe
data <- as.data.frame(seq, row.names = unique(diary$caseid))

## create id
data$id <- as.numeric(row.names(data))

## used created vars names to name the columns in the dataframe
colnames(data) <-c(activities, "caseid")

## merge with weights wt06
df1 <- as.data.frame(unique(diary %>% select(caseid, wt06) ))
df1 <- data.frame(df1[,-1], row.names = df1[,1])
colnames(df1) <-"wt06"
head(df1)
```

```
##                    wt06
## 20110101110010  1418123
## 20110101110072  8091097
## 20110101110074 25198135
## 20110101110081 12544499
## 20110101110082  6573078
## 20110101110101  5910557
```

```
data <-cbind(data, df1)
```

Sequence Analysis

I create an object with intervals' labels. Sequences start at 04:00 AM: Depending on your own sequence intervals (if you are not using the ATUS) these labels need to be adjusted

```
t_intervals_labels <-  format( seq.POSIXt(as.POSIXct("2021-11-08 04:00:00 GMT"),
                                          as.POSIXct("2021-11-09 03:59:00 GMT"), by = "1 min"),
                               "%H:%M", tz="GMT")
```

Colour Palette

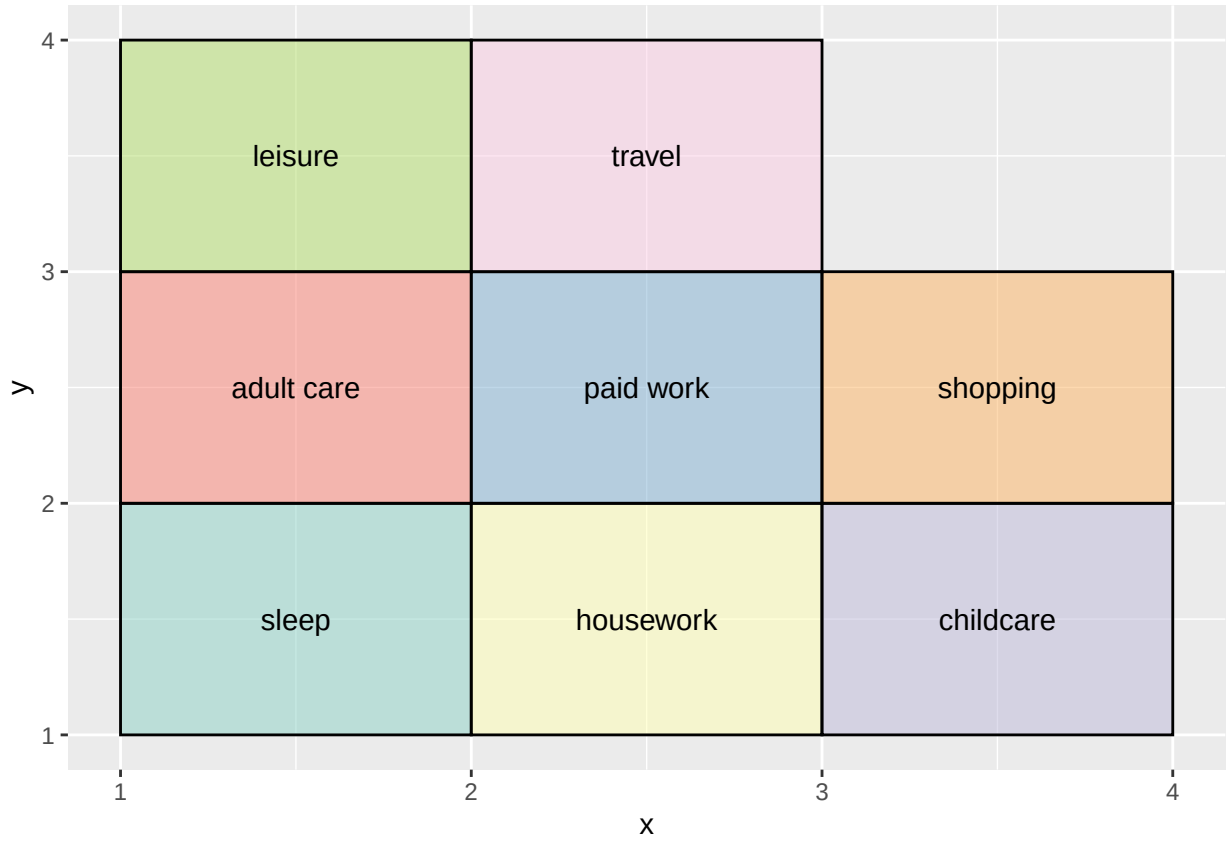
In my experience, brewing colours in R is one of the most painful experiences, especially if you have many categories of activities. Things can get even worse if you have to go grayscale if publishing articles with colour images is too expensive.

Let's brew some colours first.The number of colours is dictated by the number of states (in the alphabet). We reduced it to 8, so it won't create difficulties (difficulties arise when there are more than 12 categories).

Interesting resource on colors (cheatsheet):

```
## define labels first and count:
labels = c("sleep", "housework",
           "childcare", "adult care",
           "paid work", "shopping", "leisure",
           "travel")
colourCount = length(labels)
```

```
getPalette = colorRampPalette(brewer.pal(8, "Set3"))

## let's see how our colours look like
axisLimit <- sqrt(colourCount)+1
colours=data.frame(x1=rep(seq(1, axisLimit, 1), length.out=colourCount),
                   x2=rep(seq(2, axisLimit+1, 1), length.out=colourCount),
                   y1=rep(1:axisLimit, each=axisLimit,length.out=colourCount),
                   y2=rep(2:(axisLimit+1), each=axisLimit,length.out=colourCount),
                   t=letters[1:colourCount], r=labels)

## check the created pallette:
ggplot() +
  scale_x_continuous(name="x") +
  scale_y_continuous(name="y") +
  geom_rect(data=colours, mapping=aes(xmin=x1, xmax=x2, ymin=y1, ymax=y2, fill=t),
            color="black", alpha=0.5) +
  geom_text(data=colours, aes(x=x1+(x2-x1)/2, y=y1+(y2-y1)/2, label=r), size=4) +
  scale_fill_manual(values = getPalette(colourCount)) + theme(legend.position = "none")
```



Defining Sequence Object

Before defining a sequence object, I reduced the dataframe to 2000 sequences. This is because I wanted for the tutorial demostration to run faster. You can use all sequences as long as they are less than about 46000 in total. On my laptop 46000's dissimilarity matrix calculation ran for 4 hours but the cluster analysis

crushed. So, I would recommend to use a random smaller sample if you have huge ones. Or, run the analyses on chunks and then combine the similar-looking clusters.

```
## subsetting is not necessary, but for the sake of efficiency
## in here I'll subset it to 2000 observations
MyData <- as_tibble(data[1:2000,])


## you want to use the full categories of states:
## (you need to change if you only focus on specific activities)
seq <- seqdef(MyData,
              var = activities,
              cnames = t_intervals_labels,
              alphabet = c("1", "3", "4", "5",
                           "6", "7", "10",
                           "11"),
              labels = labels,
              cpal = getPalette(colourCount),
              xtstep = 18, ##step between displayed tick-marks and labels on the time x-axis
              id = MyData$caseid,
              weights = MyData$wt06)

## check how the sequence looks like
print(seq[1:5, ], format = "STS")
```

```
##                Sequence
## 20110101110010 1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-
## 20110101110072 1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-
## 20110101110074 10-10-10-10-10-10-10-10-10-10-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-
## 20110101110081 1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-
## 20110101110082 1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-
```

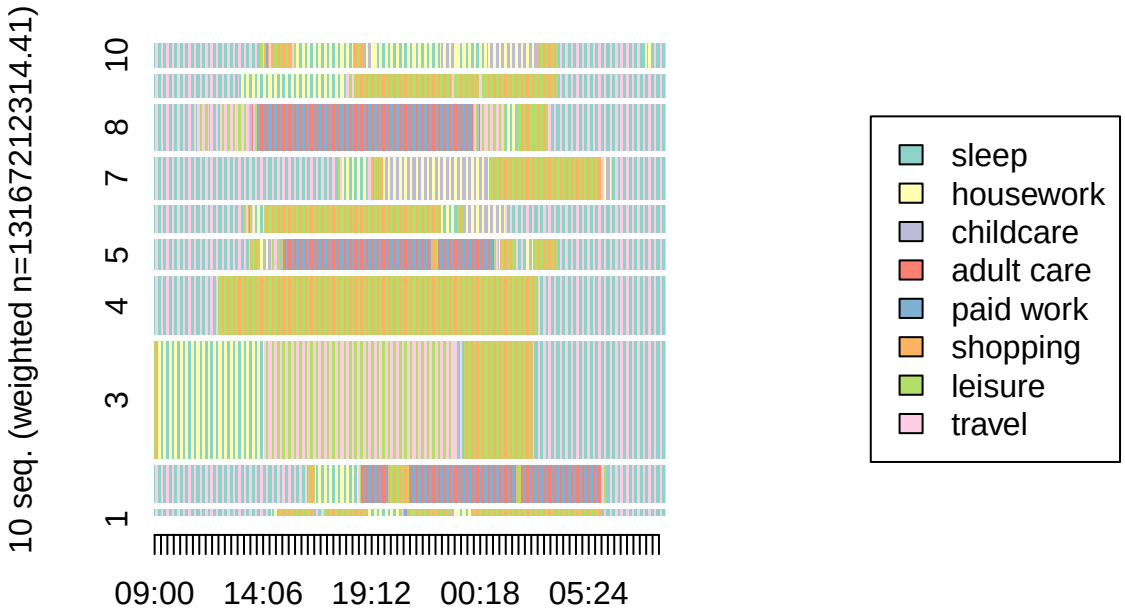
```
## "STS" format shows each step
```

Plotting Sequences

- sequence index plots

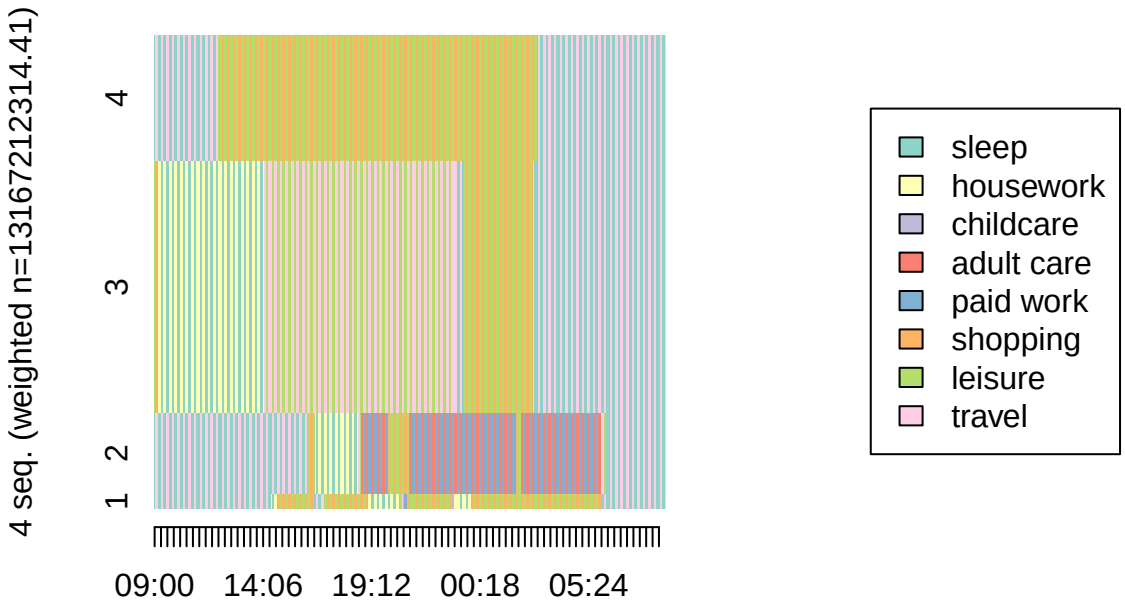
With a small i, the default for idxs is 1:10, plotting the first 10 sequences. If you set idxs to 0, it plots all sequences (might take a long time).

```
seqiplot(seq, border = NA, with.legend = "right", legend.prop=0.4)
```



You can also use the same command with a capital I. It wiil plot all unless you specify idxs option.

```
seqIplot(seq, border = NA, with.legend = "right", legend.prop=0.4, idxs = 1:4)
```



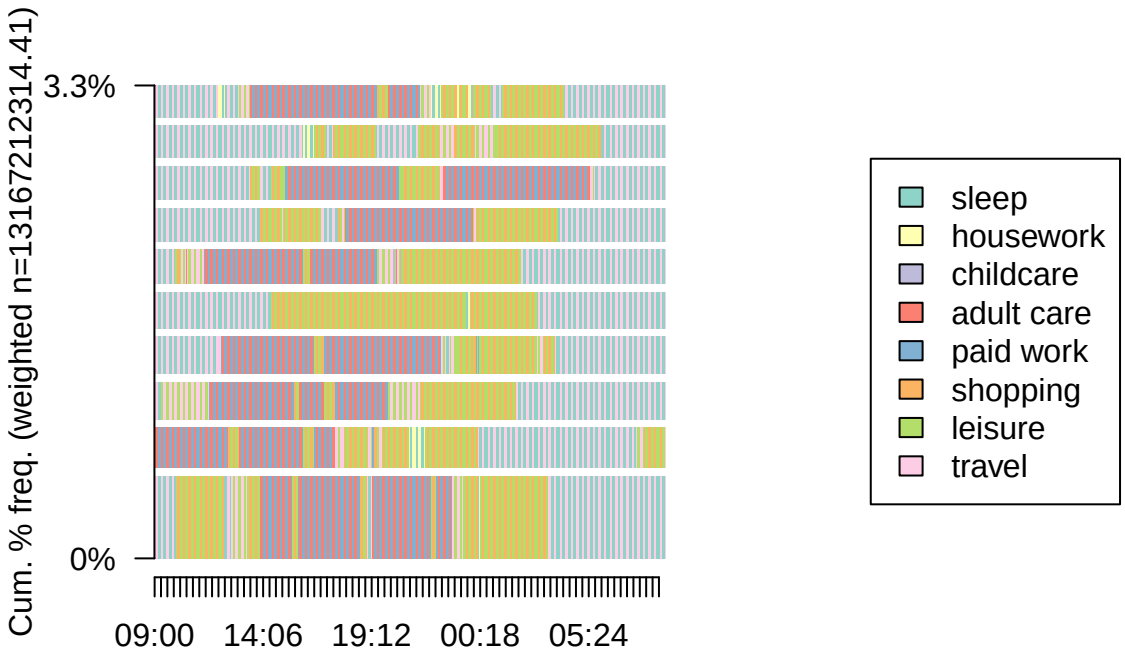
- the most frequent sequences

Unfortunately, for time-use data, it is usually useless to plot the most frequent sequences. If you tabulate 4 frequent sequences you will see what I mean. It is because there are 1440 steps and there are barely any same sequences. This command is more useful for shorter sequences with many commonalities (as in life-course research).

```
## tabulate 4 frequent sequences:
## because there are 1440 steps there are barely any
## this is more useful for shorter sequences with many commonalities (as in life-course research)
seqtab(seq, idxs = 1:4)
```

```
##
## 1/63-10/135-1/20-11/45-10/37-6/90-10/15-6/175-10/20-1/15-6/165-10/15-6/45-11/30-10/45-3/5-10/190-1/33
## 6/210-10/30-6/180-10/30-6/60-11/30-10/60-11/15-7/15-11/15-10/75-3/45-10/150-1/445-11/12-5/1-11/10-10/
## 1/20-11/135-6/240-10/15-6/70-10/30-6/150-11/90-10/270-1/420
## 1/180-11/10-6/260-10/30-6/330-11/10-1/15-11/20-10/45-11/2-7/10-11/2-7/2-10/164-11/20-10/30-1/310
##
## 1/63-10/135-1/20-11/45-10/37-6/90-10/15-6/175-10/20-1/15-6/165-10/15-6/45-11/30-10/45-3/5-10/190-1/33
## 6/210-10/30-6/180-10/30-6/60-11/30-10/60-11/15-7/15-11/15-10/75-3/45-10/150-1/445-11/12-5/1-11/10-10/
## 1/20-11/135-6/240-10/15-6/70-10/30-6/150-11/90-10/270-1/420
## 1/180-11/10-6/260-10/30-6/330-11/10-1/15-11/20-10/45-11/2-7/10-11/2-7/2-10/164-11/20-10/30-1/310
```

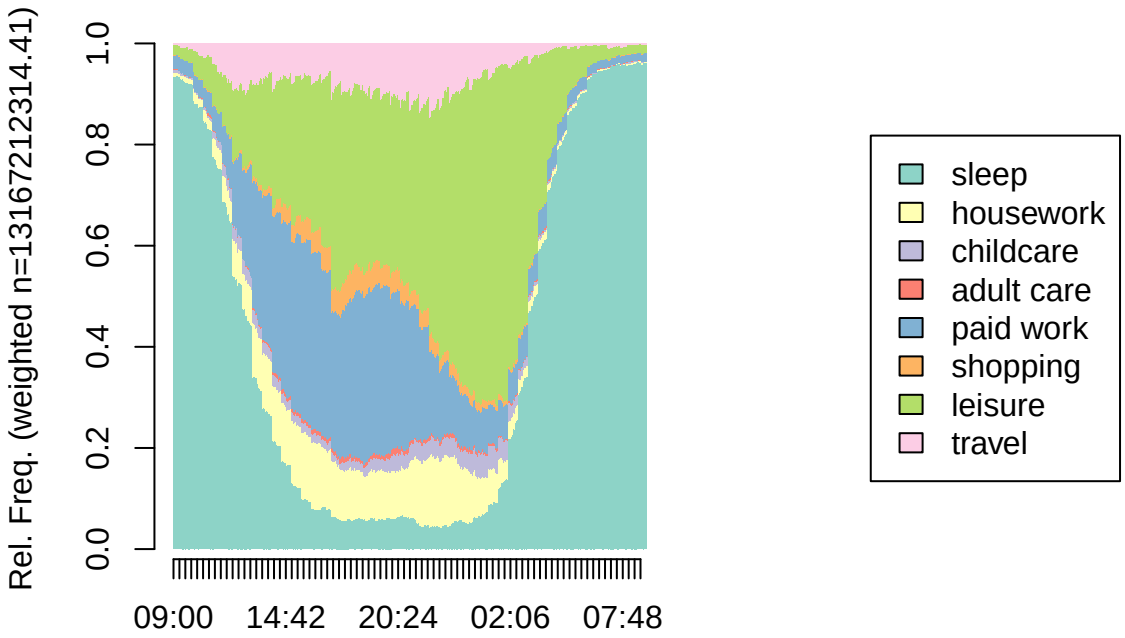
```
##also can plot frequencies using seqfplot
seqfplot(seq, border = NA, with.legend = "right", legend.prop=0.4)
```



- tempograms

State distribution plots (aka tempogram aka chronogram) This is an easy way to plot a tempogram (compared to area plots).

```
## this is an easy way to plot a tempogram
seqdplot(seq, border = NA, with.legend = "right", legend.prop=0.4)
```



Transitions

```
# transitions from state to state (in probabilities)
trate <- seqtrate(seq)
```

```
##  [>] computing transition probabilities for states 1/3/4/5/6/7/10/11 ...
```

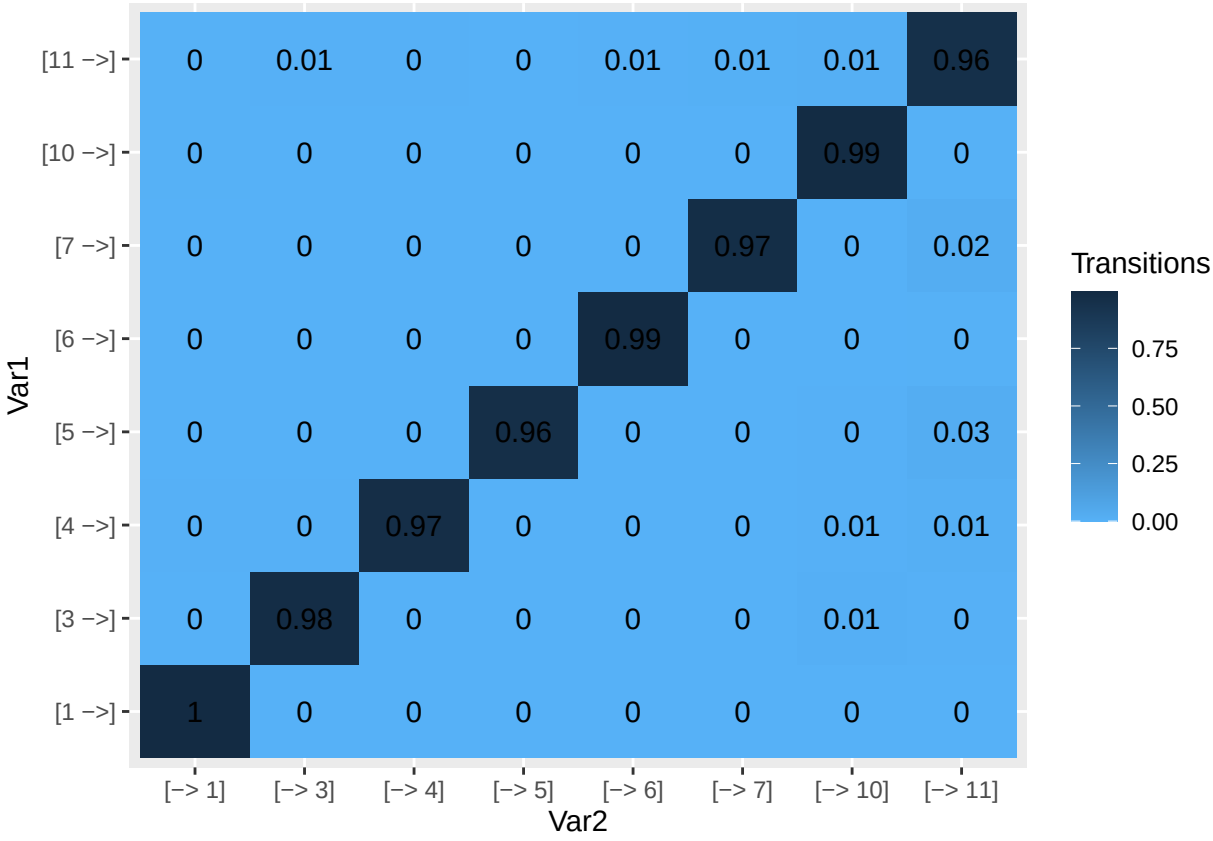
```
round(trate, 2)
```

```
##          [-> 1] [-> 3] [-> 4] [-> 5] [-> 6] [-> 7] [-> 10] [-> 11]
## [1 ->]        1   0.00   0.00   0.00   0.00   0.00    0.00    0.00
## [3 ->]        0   0.98   0.00   0.00   0.00   0.00    0.01    0.00
## [4 ->]        0   0.00   0.97   0.00   0.00   0.00    0.01    0.01
## [5 ->]        0   0.00   0.00   0.96   0.00   0.00    0.00    0.03
## [6 ->]        0   0.00   0.00   0.00   0.99   0.00    0.00    0.00
## [7 ->]        0   0.00   0.00   0.00   0.00   0.97    0.00    0.02
## [10 ->]       0   0.00   0.00   0.00   0.00   0.00    0.99    0.00
## [11 ->]       0   0.01   0.00   0.00   0.01   0.01    0.01    0.96
```

```
## heatmap of the transitions matrix
heatTrate=melt(trate)
head(heatTrate)
```

```
##     Var1   Var2        value
## 1 [1 ->] [-> 1] 9.966837e-01
## 2 [3 ->] [-> 1] 2.179312e-03
## 3 [4 ->] [-> 1] 3.932838e-03
## 4 [5 ->] [-> 1] 1.806305e-03
## 5 [6 ->] [-> 1] 2.746448e-04
## 6 [7 ->] [-> 1] 7.920407e-05
```

```
## plot the heatmap
ggplot(heatTrate, aes(Var2, Var1)) +
  geom_tile(aes(fill = value)) +
  geom_text(aes(label = round(value, 2))) +
  scale_fill_continuous(high = "#132B43", low = "#56B1F7", name="Transitions")
```



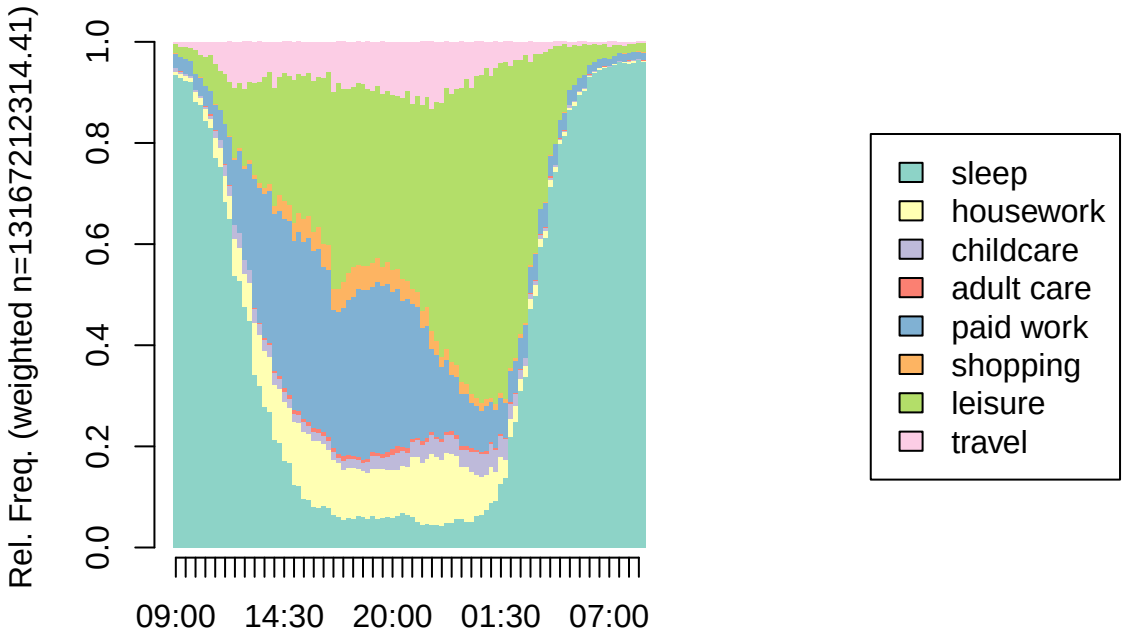
Changing Granularity (Minutes to Hours, etc.)

TraMineR made it very easy to change the number of steps in a sequence. *seqgranularity* is the command that will help you do it.

To use the first state to represent all, use method = "first", the last = "last", or the most frequent = "mostfreq".

In the following chunk of code, tspan = 15 means to transform the step to every 15 min (instead of every minute).

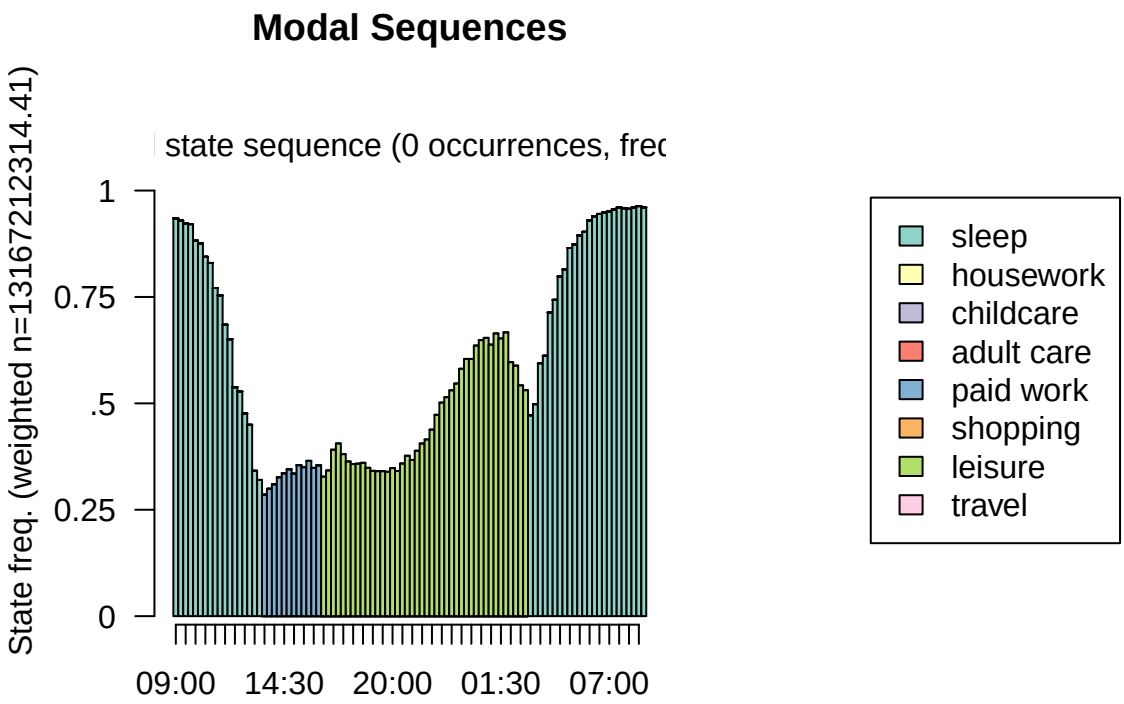
```
time15_seq <- seqgranularity(seq, tspan=15, method="mostfreq")

## plot the tempogram
seqdplot(time15_seq, border = NA, with.legend = "right", legend.prop=0.4)
```



You can see on the tempograms that the granularity decreased and now each step is a 15-minute slot.

Modal states sequence

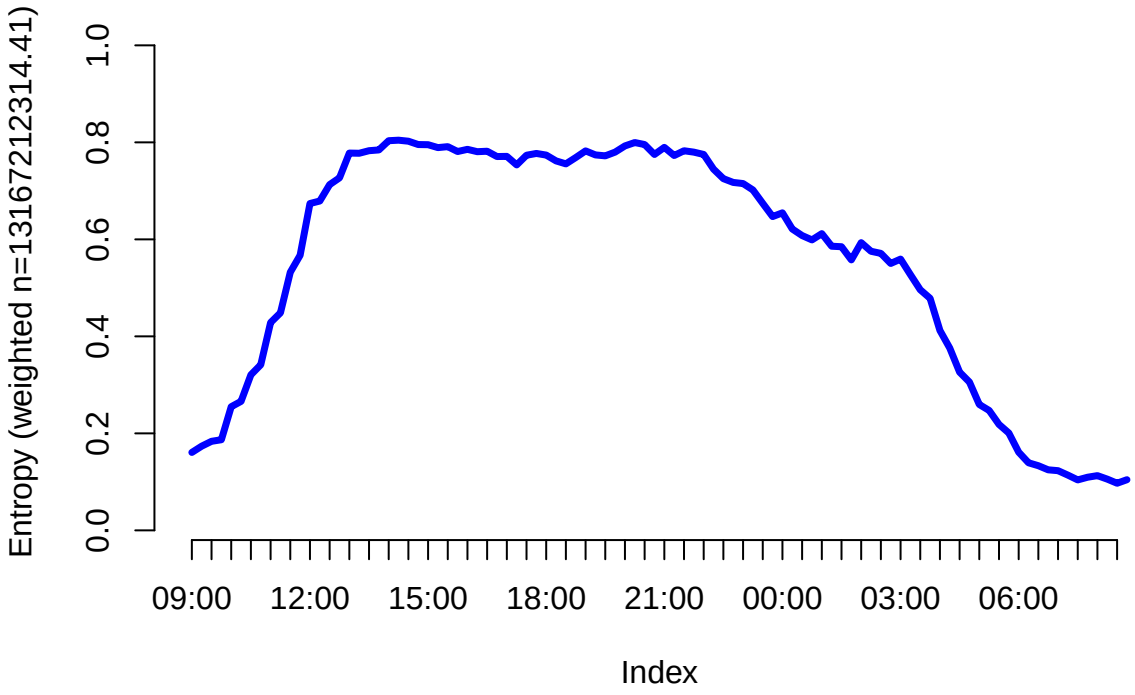
```
#seqplot(time15_seq, type="ms", with.legend = "right", legend.prop=0.4)
## same as
seqmsplot(time15_seq, with.legend = "right", legend.prop=0.4, main="Modal Sequences")
```



Embrace entropy

The higher the value, the more diverse are activities at that time.

```
#transversal entropy of state distributions
#the number of valid states and the Shannon entropy of the transversal state distribution
# shows the measure of 'chaos' (diversity of activities) in the diaries
seqHtplot(time15_seq, with.legend = "right", legend.prop=0.4)
```



Calculating Dissimilarities

Substitution Cost Matrix

We need to define the substitution cost for all the transitions (it can be a constant or user-defined).

```
# seqdist() = for pairwise dissimilarities
# seqsubm() = to compute own substitution matrix
#"TRATE" option, the costs are determined from the estimated transition rates
scost <- seqsubm(time15_seq, method = "TRATE")
```

```
##  [>] creating substitution-cost matrix using transition rates ...
```

```
##  [>] computing transition probabilities for states 1/3/4/5/6/7/10/11 ...
```

```
round(scost, 3)
```

```
##        1     3     4     5     6     7    10    11
## 1  0.000 1.963 1.943 1.978 1.993 1.996 1.931 1.956
## 3  1.963 0.000 1.918 1.972 1.988 1.959 1.812 1.904
## 4  1.943 1.918 0.000 1.994 1.987 1.994 1.897 1.895
## 5  1.978 1.972 1.994 0.000 1.985 1.994 1.921 1.875
## 6  1.993 1.988 1.987 1.985 0.000 1.993 1.943 1.892
```

```
## 7  1.996 1.959 1.994 1.994 1.993 0.000 1.950 1.756
## 10 1.931 1.812 1.897 1.921 1.943 1.950 0.000 1.797
## 11 1.956 1.904 1.895 1.875 1.892 1.756 1.797 0.000
```

```
## calculated in this way, all are close to 2 anyway (for this dataset) 2 is default
## or we can use the usual default one of constant 2:
ccost <- seqsubm(time15_seq, method="CONSTANT", cval=2)
```

```
##  [>] creating 8x8 substitution-cost matrix using 2 as constant value
```

```
round(ccost, 3)
```

```
##    1 3 4 5 6 7 10 11
## 1  0 2 2 2 2 2  2  2
## 3  2 0 2 2 2 2  2  2
## 4  2 2 0 2 2 2  2  2
## 5  2 2 2 0 2 2  2  2
## 6  2 2 2 2 0 2  2  2
## 7  2 2 2 2 2 0  2  2
## 10 2 2 2 2 2 2  0  2
## 11 2 2 2 2 2 2  2  0
```

Optimal Matching

Optimal matching for calculating dissimilarities between sequences need the specification of both substitution and indel costs. The algorithm is developed by Needleman and Wunsch (1970). For the illustration how the algorithm works please link to my explanation of optimal matching

If the sequence file is heavy, calculate only the upper part of the matrix by full.matrix = FALSE The usual default is that substitution cost is twice the indel cost, and default indel cost is 1.

```
om_time <- seqdist(time15_seq, method = "OM", indel = 1, sm = scost)
```

```
##  [>] 2000 sequences with 8 distinct states
```

```
##  [>] checking 'sm' (size and triangle inequality)
```

```
##  [>] 1996 distinct  sequences
```

```
##  [>] min/max sequence lengths: 96/96
```

```
##  [>] computing distances using the OM metric
```

```
##  [>] elapsed time: 41.78 secs
```

```
## this results in a dissimilarity matrix which you can look at using:
round(om_time[1:10, 1:10], 1)
```

```
##        1     2     3     4     5     6     7     8     9    10
## 1    0.0  92.3 117.7  47.7 103.2  62.3  61.9 126.3  52.6  82.8
## 2   92.3   0.0 137.1 119.7  49.5 111.5  88.7  65.3 106.2 101.4
## 3  117.7 137.1   0.0 106.8 123.9 106.8 129.1 105.0  73.3 103.8
## 4   47.7 119.7 106.8   0.0 102.5  49.1  97.8 110.8  51.4 104.3
## 5  103.2  49.5 123.9 102.5   0.0  94.0 108.5  34.8  93.0  97.2
## 6   62.3 111.5 106.8  49.1  94.0   0.0  84.3 107.7  47.4  80.1
## 7   61.9  88.7 129.1  97.8 108.5  84.3   0.0 127.3  79.9  80.3
## 8  126.3  65.3 105.0 110.8  34.8 107.7 127.3   0.0 108.9 114.3
## 9   52.6 106.2  73.3  51.4  93.0  47.4  79.9 108.9   0.0  73.0
## 10  82.8 101.4 103.8 104.3  97.2  80.1  80.3 114.3  73.0   0.0
```

Cluster Analysis

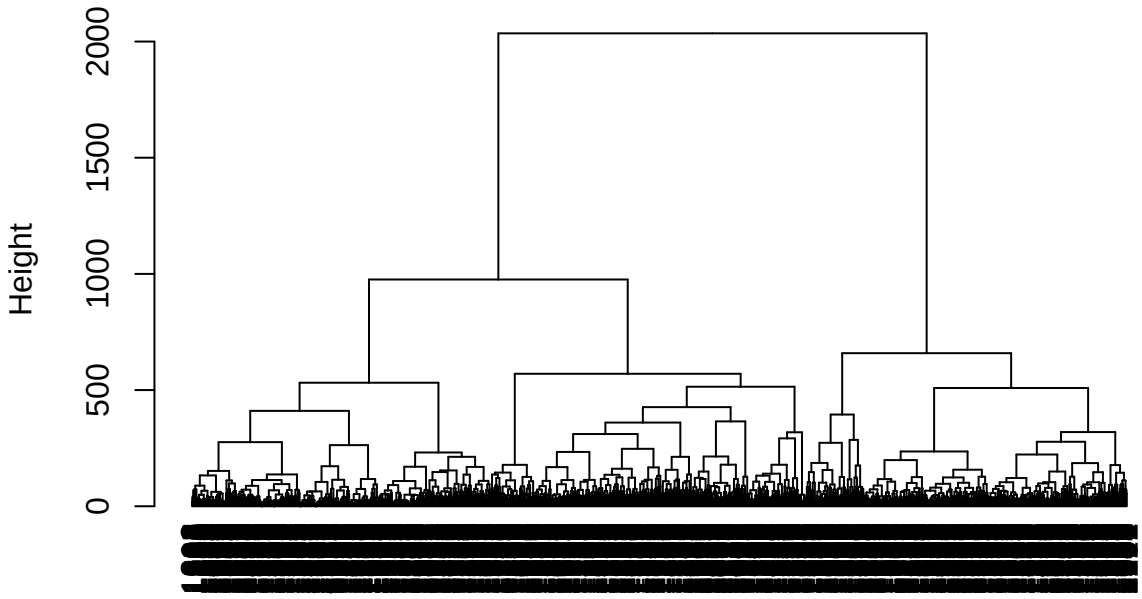
Let's run cluster analysis on our dissimilarity matrix

Other common methods are:

- "average",
- "single",
- "complete"

The "average" and "single" options do not work well for time-use data (check). The "complete" option is a possibility (check).

```
## run cluster analysis on the calculated dissimilarity matrix
clusterward <- agnes(om_time, diss = TRUE, method = "ward")
## other common methods are "average", "single", "complete" (instead of "ward")
## "average" and "single" do not work well for time-use data (check if you want)
## "complete" can be an option
## "ward" is the "industry standard"

# Convert hclust into a dendrogram and plot
hcd <- as.dendrogram(clusterward)

# plot the dendrogram
plot(hcd, type = "rectangle", ylab = "Height")
```



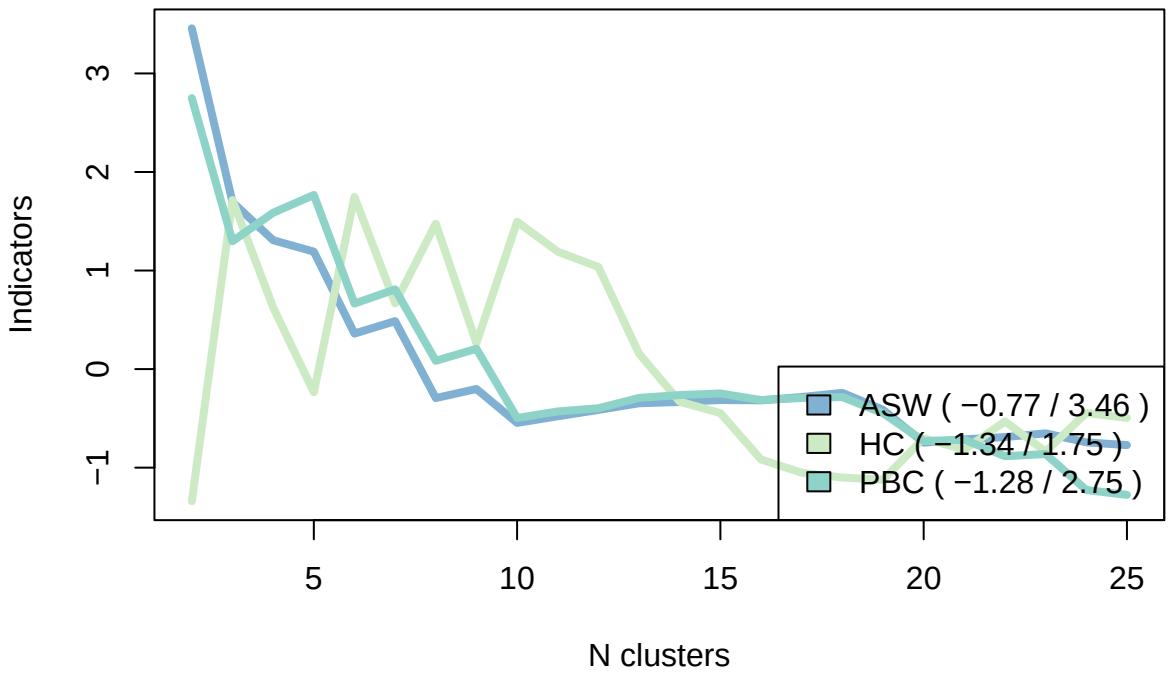
How good is our clustering?

Let's inspect the splitting tree branches:

```
#inspect the splitting steps
ward.tree <- as.seqtree(clusterward, seqdata = time15_seq,
                            diss = om_time,
                            ncluster = 25)
## plot the tree of tempograms to check how it splits the diaries
seqtreedisplay(ward.tree, type = "d", border = NA, show.depth = TRUE, gvpath='C:/Program Files/GraphViz
## use this one instead of the above:
# seqtreedisplay(ward.tree, type = "d", border = NA, show.depth = TRUE)
```

There are many possible tests for the cluster solution quality: - Point Biserial Correlation (PBC). - Hubert's Gamma (HG). - Hubert's Gamma (Somers'D) (HGSD). - Average Silhouette Width (ASW). - Average Silhouette Width (weighted) (ASWw). - Calinski-Harabasz index (CH). - R2. - Calinski-Harabasz Index squared (CHsq). - R2sq. - Hubert's C coefficient (HC).

```
#test cluster solution quality
wardtest <- as.clustrange(clusterward,
                          diss = om_time,
                           ncluster = 25)

#plot the quality criteria
plot(wardtest, stat = c("ASW", "HC", "PBC"), norm = "zscore", lwd = 4)
```



Let's say that our solution is pretty good for 8 clusters.

1. Cut the tree

```
c8 <- cutree(clusterward, k = 8)

## bind with the dataset
MyData<-cbind(MyData, c8)
```

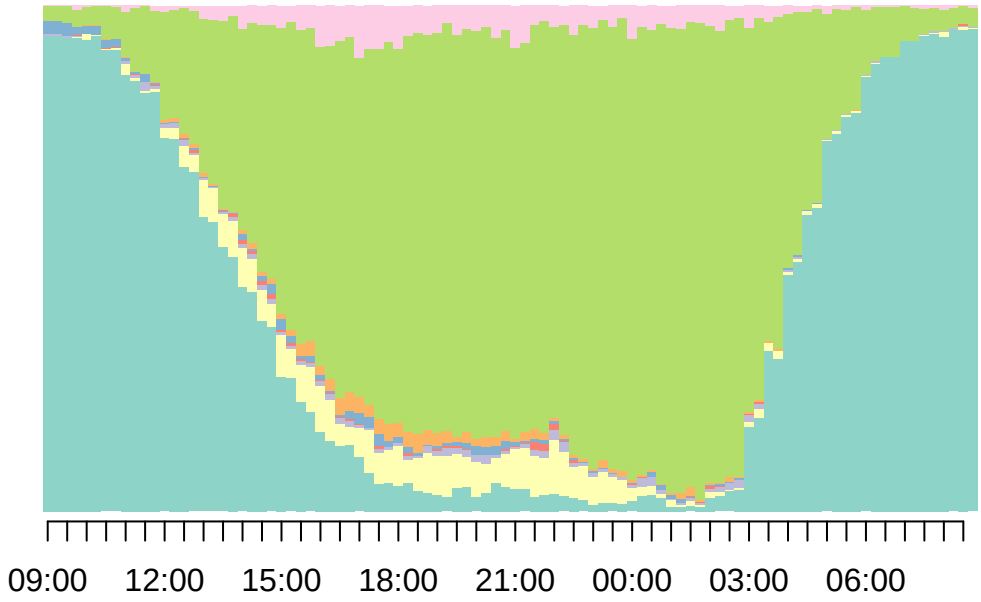
2. Plot the cluster solution (will save in the working directory)

```
#plot cluster solution
png("plot_clusters.png", 1200, 800)
seqdplot(time15_seq, group = c8, border = NA)
dev.off()
```

```
## pdf
##   2
```

How to plot a cluster

```
# subset data by cluster
cl1<-(time15_seq[MyData$c8 ==  "1",])

# plot the selected cluster
par(mfrow=c(1,1))
seqdplot(cl1, main = "",
         cex.main = 1.7,
         with.legend = FALSE,
         yaxis = FALSE,
         cpal = getPalette(colourCount),
         ylab = "",
         border = NA)
```




```
## write new data to csv if needed
#write.csv(MyData, "clustered_EC.csv")
```

References

Flood, S.M., Sayer, L.C., & Backman, D. American Time Use Survey Data Extract Builder: Version 3.1 dataset. College Park, MD: University of Maryland and Minneapolis, MN: IPUMS, 2022.

Kolpashnikova, K., & Kan, M. Y. (2020). Eldercare in Japan: Cluster Analysis of daily time-use patterns of elder caregivers. Journal of Population Ageing 14(4), 441-463.