Institute of Education

UCL

Millennium Cohort Study

Sweep 6

A guide to the parsed / raw Time Use Diary data (TUD)

Sunil Kumar Veeravalli

(First Edition)
December 2019

CENTRE FOR LONGITUDINAL STUDIES

E·S·R·C ECONOMIC & SOCIAL RESEARCH COUNCIL

First published in 2019

Centre for Longitudinal Studies
UCL Institute of Education
20 Bedford Way
London WC1H 0AL
www.cls.ucl.ac.uk

ISBN 978-1-906929-99-2

The Centre for Longitudinal Studies (CLS) is an ESRC Resource Centre based at the UCL Institution of Education. It provides support and facilities for those using the three internationally-renowned birth cohort studies: the National Child Development Study (1958), the 1970 British Cohort Study and the Millennium Cohort Study (2000). CLS conducts research using the birth cohort study data, with a special interest in family life and parenting, family economics, youth life course transitions and basic skills. The views expressed in this work are those of the author(s) (amend as necessary) and do not necessarily reflect the views of the Economic and Social Research Council. All errors and omissions remain those of the author(s).

This document is available in alternative formats.
Please contact the Centre for Longitudinal Studies.
tel: +44 (0)20 7612 6875
email: clsfeedback@ucl.ac.uk

Data queries: help@ukdataservice.ac.uk

Contents

1. Introduction

The Millennium Cohort Study (MCS) is the fourth of Britain's world-renowned national longitudinal birth cohort studies.[1] Each follows a large sample of individuals, born over a specific period of time, through the course of their lives. They provide insight into trajectories/mobility over the life course (e.g. in weight, poverty, education) for the population as a whole, and also into how these movements vary across different sub-populations. They are also uniquely placed to allow for an understanding of the relationship between early life events and circumstances and outcomes later on in life.

The MCS provides detailed information on approximately 19,000 children born at the start of the new century and their families, across the United Kingdom. In England and Wales, the cohort members were born over the 12 month period starting September 2000. In Scotland and Northern Ireland, they were born over 13½ months from November 2000. The sample design allowed for disproportionate representation of families living in areas of child poverty, and in areas of England with high ethnic minority populations. Information was first collected from parents, through a home-based survey, when the cohort members were aged nine months. This first survey recorded, amongst other things, the circumstances of pregnancy and birth, as well as those of the early months of life, and the social and economic background of the children's families.

These multidisciplinary baseline data reveal the diversity of starting points from which the 'Children of the New Century' set out. Subsequent surveys have taken place at ages 3, 5, 7, 11, 14 and 17. These surveys coincide with important moments in children's lives, including the preschool period (age 3), and the start, middle and end of primary school (ages 5, 7, 11) and in later school years at key transitional ages (14 and 17). From age 3 onwards, measured physical development and objective cognitive assessments have been carried out with the cohort members; surveys also include interviews with both parents (where co-resident), and, increasingly since age 7, with the cohort member.

Further details of the data available from the main surveys are available from the CLS website and, in particular, the MCS Guide to the Datasets at www.cls.ucl.ac.uk/cls-studies/millennium-cohort-study/

Sweep	Year of data collection	Cohort members' age
MCS 1	2001/2	9 months old
MCS 2	2003/4	3 years old
MCS 3	2005/6	5 years old
MCS 4	2008/9	7 years old
MCS 5	2012/3	11 years old
MCS 6	2015/6	14 years old
MCS 7	2018/9	17 years old

[1]The previous birth cohort studies, all still ongoing, are the 1946 National Birth Cohort, the 1958 British Birth Cohort (also known as the National Child Development Study) and the 1970 British Cohort Study.

2. Time-use diary data collection methods

The data collected is a snapshot of cohort members' daily life. Three different approaches were used to collect time-use data from 14 year old Millennium Cohort Study (MCS) members: paper, web and app. Paper-administered diaries were only offered to those with no access to a personal computer or a smartphone with internet access.

Each cohort member was asked to complete two diaries, one on a weekday and one on a weekend day where a day starts at 4am in the morning and ends at 4am the following day. Activities were grouped into 13 broader categories which account for a total of 44 activities. For paper and web based instruments, time was split into 10 minute slots such that the whole day starting 4AM and ending 4AM the following day had 144 slots. Instead of 10-minute slots, the app instruments allowed cohort members to assign the ending times of their activities.

Along with the activities, cohort members were also asked to choose from the multi-coded options from three sets of questions: where they were (location) during that activity, who they were with and how much did they like doing that activity. In addition, each cohort member was asked to answer a set of final few questions at the end of the study. A detailed description of the time-use study is available at https://cls.ucl.ac.uk/wp-content/uploads/2017/04/CLS-WP-2015-5.pdf

3. Data management

The raw data generated from all three instruments were stored in SPSS format. The data for paper and web based instruments have the same structure since the cohort member can only choose from the 10-minute slots whereas the structure for app based approach is different. Therefore, the data handling and processing is different as discussed below.

3.1 Paper and Web raw data

The raw data file for paper have 329 rows and 28 columns whereas the raw data file for web have 1309 rows and 28 columns. In both cases, each row refers to a single cohort member and includes activity measures for both assigned days saved as columns 'day1_primary' and 'day2_primary'. These columns consists of binary arrays of size 8352 [(44 activities + 3 where questions + 6 who questions + 5 like questions) * 144 ten-minute slots = 8352].

3.2 App raw data

The raw data file for app is in wide format with 6122 rows and 1118 columns. Each row represents a single day activity of a cohort member. The data is in episode format i.e., cohort member has the flexibility to choose the end time for an activity.

4. Data parsing

The raw data from all three instruments were parsed using Python3.7.

4.1 Parsing of Paper and Web raw data

The binary array format has been converted to long format such that each row represents a 10-minute time slot with the information of the activity, location of the activity, who is accompanied in the activity and the likeliness of the activity in the

respective columns. So, a day of activity for a cohort member requires 144 rows (24 hours = 144 ten-minute slots).

4.2 Parsing of App raw data

The wide format of the raw data has been converted to long format. Data has been parsed in two ways:

4.2.1 Episode format

This doesn't follow the 10-minute time slot categorization. The end time specified by the cohort member for an activity has been used as it is. For example:

Time	Activity
08:52	Breakfast
09:20	Travel to school
12:32	Classroom

4.2.2 Calendar format

The time used for an activity is represented in the form of 10-minute time slots. The example shown in section 4.2.1 will be:

Time	Activity
……. – 08:50	Breakfast
08:50 – 09:00	Travel to school
09:00 – 09:10	Travel to school
09:10 – 09:20	Travel to school
09:20 – 09:30	Classroom
……. – …….	Classroom
……. – 12:30	Classroom

5. Data available

The data file names for the parsed data:

Instrument	File name
Paper	mcs6_tud_parsed_data_paper_calendar_format
Web	mcs6_tud_parsed_data_web_calendar_format
App	mcs6_tud_parsed_data_app_episode_format

IDs used in the dataset MCSID and FCNUM00 as a cohort member number within a MCS family.

Appendix

Python syntax to convert episode format to calendar format

Before executing the code below, fill the variable sav_file_path in the code with the path to the location of the file mcs6_tud_parsed_data_app_episode_format on your computer. Upon execution, the calendar format data will be saved to the same location with file name:
'mcs6_tud_parsed_data_app_calendar_format_conversion.sav'.

```
import pandas as pd
```

```
import savReaderWriter as srw
import warnings
warnings.filterwarnings('ignore')
import numpy as np


# Write down between the double quotes below, the path where the file
‘mcs6_tud_parsed_data_app_episode_format.sav’ is located
sav_file_path = ""

# importing the data
episode = list(srw.SavReader(savFileName = sav_file_path +
'\\mcs6_tud_parsed_data_app_episode_format.sav',
                             returnHeader = True,
                             ioUtf8 = True))
episode = pd.DataFrame(episode[1:], columns = episode[0])


# Importing the metadata
with srw.SavHeaderReader(savFileName = sav_file_path +
'\\mcs6_tud_parsed_data_app_episode_format.sav') as header:
    metadata = header.all()

# Generating timeslots
time_slots = []
hours = ['04','05','06','07','08','09','10','11','12',
         '13','14','15','16','17','18','19','20','21',
         '22','23','00','01','02','03']
minutes = ['00','10','20','30','40','50']

for i in hours:
    for j in minutes:
        time_slots.append(i + ':' + j)

time_slots = time_slots[1:]
time_slots.append('04:00')

mod_time_slots = []
for i, j in zip(range(101,245), time_slots):
    mod_time_slots.append(str(i) + '_' + j)

dict_mod_time_slots = {i:j for i, j in zip(time_slots, mod_time_slots)}
dict_mod_time_slots_rev = {i:j for i, j in zip(mod_time_slots, time_slots)}
encode_time_slots = {i:j for i, j in zip(mod_time_slots, range(1,145))}

# trimming time to xx:xx from xx:xx:xx
episode.FCTUDSLOT = episode.FCTUDSLOT.str.slice(start = 0, stop = 5)

# An empty dataframe that will populate all the output from the loops below
final_table = pd.DataFrame()

uniq_mcsid = episode.MCSID.unique()

for mcsid_seq, mcsid in enumerate(uniq_mcsid):
    temp_dataset = episode.loc[episode.MCSID == mcsid, :].copy()
    no_of_days = temp_dataset.FCTUDAD.unique()
```

```
    # Loop that runs for each day
    for day in no_of_days:
        temp_dataset2 = temp_dataset.loc[temp_dataset.FCTUDAD == day, :]
        temp_dataset2
        temp_dataset2.reset_index(drop = True, inplace = True)
        # Creating the dummy table
        dummy_table = pd.DataFrame(columns = ['activity'], index =
time_slots)
        dummy_table.index.name = 'time_slots'
        # Converting the times. For example, if time given by user is
09:12, we convert into 09:20. This is because, our time_slots value for
09:20 mean 09:10 to 09:20
        for i in range(temp_dataset2.shape[0]):
            if i == temp_dataset2.shape[0] - 1:
                if temp_dataset2.FCTUDSLOT[i] in [' ', '']:
                    temp_dataset2.FCTUDSLOT[i] = '04:00'
            if temp_dataset2.FCTUDSLOT[i] not in dummy_table.index:
                if temp_dataset2.FCTUDSLOT[i] > '23:50':
                    temp_dataset2.FCTUDSLOT[i] = '00:00'
                else:
                    temp_dataset2.FCTUDSLOT[i] =
dummy_table.index[dummy_table.index >
temp_dataset2.FCTUDSLOT[i]].sort_values(ascending = True)[0]
            else:
                pass
        # There will be issue working with time because, a day is running
from 04:00AM to 03:59AM next day.
        # So, it will be difficult to sort or order the rows by time.
Hence, I need to add a numeric value before the time, so that we can sort
or order.
        temp_dataset2.FCTUDSLOT =
temp_dataset2.FCTUDSLOT.map(dict_mod_time_slots)
        # There are instances where the last but one row time is 23:00 and
the last row is 09:00 for the next day. So, such data has to be converted
to 23:00 and 04:00 as done below.
        total_rows = temp_dataset2.shape[0]
        if total_rows > 1:
            if temp_dataset2.FCTUDSLOT[total_rows-1] <
temp_dataset2.FCTUDSLOT[total_rows-2]:
                temp_dataset2.FCTUDSLOT[total_rows-1] = '244_04:00'

        # Sometime, we see the time of entry not in an order. For example,
09:00, then 10:00, then 09:30.
        # So, we need to sort the time.
        temp_dataset2.sort_values(by = 'FCTUDSLOT', inplace = True)
        # There are instances where we see an activity for 09:00 and
another activity for the same 09:00.
        # So, removing those duplicate entries for the same time. Keeping
only the first activity.
        temp_dataset2 =
temp_dataset2.loc[~temp_dataset2.FCTUDSLOT.duplicated(), :]
        temp_dataset2.reset_index(drop = True, inplace = True)
        dummy_table.index = dummy_table.index.map(dict_mod_time_slots)
        for i in range(temp_dataset2.shape[0]):
            if i == 0:
```

```
                end = '101_04:10'
            else:
                end = dummy_table.index[dummy_table.index >
temp_dataset2.FCTUDSLOT[i-1]][0]
            ids = dummy_table.index[(dummy_table.index >= end) &
(dummy_table.index <= temp_dataset2.FCTUDSLOT[i])]
            ids
            for j in ids:
                dummy_table.activity[j] = temp_dataset2.FCTUDACT[i]

        # for those times whose value is NaN, it means the cohort member
did not record the activity. Such NaN's are to be represented as missing
values with -1.
        dummy_table.activity.fillna(value = -1, inplace = True)
        dummy_table.reset_index(drop = False, inplace = True)

        # Reversing the time from 101_04:00 to 1,2,3...
        dummy_table.time_slots =
dummy_table.time_slots.map(encode_time_slots)

        # Renaming the columns
        dummy_table.rename(columns = {'time_slots':'FCTUDSLOT',
'activity':'FCTUDACT'}, inplace = True)

        # Adding rest of the columns
        dummy_table['MCSID'] = [temp_dataset2.MCSID[0]] * 144
        dummy_table['FCNUM00'] = [temp_dataset2.FCNUM00[0]] * 144
        dummy_table['FCTUDAD'] = [temp_dataset2.FCTUDAD[0]] * 144
        dummy_table['FCTUDMOD'] = [temp_dataset2.FCTUDMOD[0]] * 144
        dummy_table['FCTUDMONTH'] = [temp_dataset2.FCTUDMONTH[0]] * 144
        dummy_table['FCTUDYEAR'] = [temp_dataset2.FCTUDYEAR[0]] * 144
        dummy_table['FCTUDWHE'] = [temp_dataset2.FCTUDWHE[0]] * 144
        dummy_table['FCTUDWHO01'] = [temp_dataset2.FCTUDWHO01[0]] * 144
        dummy_table['FCTUDWHO02'] = [temp_dataset2.FCTUDWHO02[0]] * 144
        dummy_table['FCTUDWHO03'] = [temp_dataset2.FCTUDWHO03[0]] * 144
        dummy_table['FCTUDWHO04'] = [temp_dataset2.FCTUDWHO04[0]] * 144
        dummy_table['FCTUDWHO05'] = [temp_dataset2.FCTUDWHO05[0]] * 144
        dummy_table['FCTUDWHO06'] = [temp_dataset2.FCTUDWHO06[0]] * 144
        dummy_table['FCTUDWHO07'] = [temp_dataset2.FCTUDWHO07[0]] * 144
        dummy_table['FCTUDHOW'] = [temp_dataset2.FCTUDHOW[0]] * 144
        dummy_table['FCTUDFINQ101'] = [temp_dataset2.FCTUDFINQ101[0]] * 144
        dummy_table['FCTUDFINQ102'] = [temp_dataset2.FCTUDFINQ102[0]] * 144
        dummy_table['FCTUDFINQ201'] = [temp_dataset2.FCTUDFINQ201[0]] * 144
        dummy_table['FCTUDFINQ3'] = [temp_dataset2.FCTUDFINQ3[0]] * 144

        # Appending the dummy_table to final_table
        final_table = pd.concat(objs = [final_table, dummy_table], axis =
0, ignore_index = True)

# converting FCTUDSLOT and FCTUDACT columns to float data type
final_table.FCTUDSLOT = final_table.FCTUDSLOT.astype(np.float64)
final_table.FCTUDACT = final_table.FCTUDACT.astype(np.float64)

# reodering the columns
final_table = final_table.reindex(columns = metadata.varNames)
```

```
# Generating a sav file
savFileName = sav_file_path +
'\\mcs6_tud_parsed_data_app_calendar_format_conversion.sav'
varNames = metadata.varNames
varTypes = metadata.varTypes
varTypes['FCTUDSLOT'] = 0
varLabels = metadata.varLabels
missingValues = metadata.missingValues
valueLabels = metadata.valueLabels
start = ['04:00'] + time_slots[:-1]
varLab = {i:f'Slot {j} from {k} to {l}' for i, j, k, l in
zip(np.arange(1.0,145.0), range(1,145), start, time_slots)}
valueLabels.update({'FCTUDSLOT':varLab})

formats = metadata.formats
formats['FCTUDSLOT'] = 'F8.2'

# Writing to sav file
with srw.SavWriter(savFileName = savFileName, varNames = varNames,
varTypes = varTypes, varLabels = varLabels, missingValues = missingValues,
valueLabels = valueLabels, formats = formats, ioUtf8 = True) as writer:
    for i in range(len(final_table)):
        writer.writerow(list(final_table.iloc[i, :]))
```