esmprep - An R package to prepare data from Experience-Sampling methods for statistical analysis

Marcel Miché
Mail: esmprep@gmail.com

September 1, 2017

Contents

Preface

The explicit goal in developing the **esmprep** package was to make it easy to use. Everybody who is able to install R (`https://www.r-project.org/`) and then within R first install a package (`install.packages('esmprep')`) and load it (`library(esmprep)`), should be able to use the **esmprep** package. For any further guidance use this document.

1 Introduction

The **Experience Sampling Methodology** (**ESM**) comprises research, in which people fill out 1+ questionnaires on an electronic mobile device for a certain period of time (1+ days) in their natural environment for 1+ consecutive days.
Research projects including ESM have increased in number, a trend which can be expected to continue. One of the purported reasons (cite) for this increase is the easy implementation of an ESM questionnaire app on a mobile device. However, it isn't 'easy' per se!
The justification as well as the motivation for developing the **esmprep** package is 3-fold:

1. To save researchers (students, doctoral candidates, postdocs, ...) precious time, which they otherwise might have to spend on preparing the raw ESM data[1].

2. To help preparing ESM datasets prior to the statistical analysis.

3. To be of help in improving the ESM questionnaire app, which optimally prevents unintended results in the raw ESM data.

On the subsequent pages I will introduce you to a hierarchy of procedures applied to raw ESM data. The data itself is simulated, the errors, on the other hand, stem from real ESM research projects! And so does the data structure. It is therefore not 'unrealistically' complex.

2 Applied ESM

The combinations involving ESM cannot exceed 3 scenarios:

1. Number of ESM questionnaire versions = 1; i.e. during the entire ESM period all participants always get to answer the same questions (same content, same amount, same order).

2. Number of ESM questionnaire versions $\geq$ 1; e.g. the questions in the morning are different and/or less in number than the questions during the day.

[1] Depending both on their own software skills and on the quality of the ESM questionnaire app.

3. The times of when the participant shall answer the daily ESM questionnaire(s) either are scheduled (an alarm signal prompts the participant) and/or the participant fills out ESM questionnaires due to some personal relevance (event contingent).

The third point might be split into 3 separate points, i.e. **all** daily ESM questionnaires are prompted, **none** is prompted or some are promted and some are not (**some**). The **esmprep** package can cover all of these cases. See table 1.

Table 1: Combinations (C) of ESM elements that **esmprep** can be used for.

	Daily ESM questionnaires prompted		
ESM versions	All	None	Some
1	C1	C2	C3
≥ 1	C4	C5	C6

Throughout this document I shall refer to table 1 when explaining which function makes sense to be used and how exactly it needs to be used, depending on the specific combination C1–C6 in your research project.

3 Install and load 'esmprep'

After having installed R, open R with the IDE[2] of your choice, e.g. RStudio. With an R IDE open, first install the 2 packages lubridate and **esmprep** by entering into the console:

```
install.packages(c("lubridate","esmprep"))
```

This needs to be done only once, not every time!

From now on use the **esmprep** package just by entering at the beginning of each session:

```
library(esmprep)
```

The package lubridate, from which a few functions are used, doesn't need to be loaded separately.

For additional help and the most up to date version of **esmprep** go to its GitHub page.

[2]IDE is short for integrated development environment.

4 Overview of the function hierarchy within the 'esmprep' package

Table 2: All functions within the **esmprep** package in hierarchical order.

No.	C1–C3	Function name	No.	C1–C3	Function name
01a		relevantREFVN	**11b**		**convertChars**
01b		**setREF**	**12**		**esAssign**
02a		relevantESVN	**13**		**missingEndDateTime**
02b		**setES**	**14**		**esIdentical**
03	–	esList	15a		suggestShift
04		**genKey**	**15b**		**makeShift**
05		**genDateTime**	15c		printSuggestedShift
05		**splitDateTime**	**16**	–	**expectedPromptIndex**
06a		**rmInvalid**	**17**	–	**intolerable**
06b		printRmInvalid	**18**		**randomMultSelection**
07	–	esItems	**19**		**computeTimeLag**
08	–	**esPlausible**	**20**		**computeDuration**
09		**esComplete**	**21**		**computeTimeBetween**
10	–	esMerge	**22**		**esFinal**
11a		findChars			

C1–C3 = –: Function for C1–C3 in table 1 is redundant or impossible. In **bold font**: Main functions. In normal font: Supporting functions. In red font: Error handling functions. Letters a–c behind numbers: Logically relating functions.

5 Setting things up

With the **esmprep** package 7 exemplary datasets are loaded automatically: referenceDf, morningControl, dayControl, eveningControl, morningTest, dayTest, and eveningTest. To find out about these datasets in the R console enter ?referenceDf, ?morningControl, etc.

5.1 Loading the relevant datasets in R

Display the exemplary reference dataset in R like this:

```
referenceDf
```

The first line of the exemplary reference dataset can be seen in table 3.

Table 3: Example of the first participant in the exemplary reference dataset.

id	imei	start_date	start_time	st1	st2	st3	st4	end_date	end_time
P001	526745224596318	2007-07-29	21:10:00	09:10:00	13:10:00	17:10:00	21:10:00	2007-08-05	17:10:00

The exemplary raw ESM datasets are called the same way, e.g.

```
morningControl
```

In our exemplary ESM project there are 6 raw ESM datasets (called ESM versions), i.e. 3 versions (morning, day, and evening) for each of 2 different study groups (test group vs control group).

5.2 Most relevant variable names

Before being able to make use of the main purpose of the **esmprep** package, the most relevant dataset information must be made known, according to your ESM study type. See C1–C6 in table 1.

5.2.1 relevantREFVN (Function 1 of 28)

There must be a reference (REF) dataset which must include variables, the content of which is relevant for subsequent functions. VN stands for variable names.
The variable names' meaning in table 3 are:

1. **id**: The unique(!) identification number of the participant.
2. **imei**: The IMEI number (also called MEID), which is the unique identification number (15 digits) of the mobile device[3].
3. **start_date**: The date at which the ESM period of the participant started.
4. **start_time**: The prompt at the start date when the participant was supposed to fill out the first ESM questionnaire on his/her own.
5. **st**: The scheduled time(s) (prompt(s)) at which the participant is supposed to fill out the daily ESM questionnaire(s). In the example there are 4 daily prompts.
6. **end_date**: The date at which the ESM period of the participant ended.
7. **end_time**: The prompt at the end date when the participant was supposed to fill out the last ESM questionnaire on his/her own.

[3]Entering *#06# on the dial keyboard of your smartphone should return its IMEI number.

The function `relevantREFVN` expects the above 7 arguments in the above order(!), i.e. the 7 most relevant variable names of your reference dataset. Your reference dataset of course might have more variable names but they are irrelevant for the **esmprep** package's purpose.
Arguments that have to be passed to the function `relevantREFVN`, for the exemplary reference dataset in table 3. `relevantREFVN` example 1:

```
relRef <- relevantREFVN("id", "imei", "st", "start_date", "start_time",
                        "end_date", "end_time")
```

If your reference dataset contains date-time objects, i.e. date and time combined in one variable, for the ESM start and end, respectively, please pass the arguments like in `relevantREFVN` example 2:

```
relRef <- relevantREFVN("id", "imei", "st",
                        START_DATETIME = "start_dateTime",
                        END_DATETIME = "end_dateTime")
```

In the `relevantREFVN` example 2 I assumed your variable names to be `start_dateTime` and `end_dateTime`. Of course they could be any character (7) string.
The result of the function `relevantREFVN` has been assigned to the variable `relRef`, which can be displayed:

```
> relRef
$REF_ID
[1] "id"
$REF_IMEI
[1] "imei"
$REF_ST
[1] "st"
$REF_START_DATE
[1] "start_date"
$REF_START_TIME
[1] "start_time"
$REF_END_DATE
[1] "end_date"
$REF_END_TIME
[1] "end_time"
$REF_DATETIMES_SEP
[1] TRUE
```

The last element `REF_DATETIMES_SEP` is relevant only for subsequent functions, which ask if it is `TRUE`, i.e. whether date and time are separated variables in the reference dataset, or if it is `FALSE`, i.e. whether date and time is combined in one variable.

5.2.2 setREF (Function 2 of 28)

The function `setREF` sets up all that the **esmprep** package requires concerning the reference dataset. `setREF` expects 2 arguments: The first argument is the number of daily prompts (in the example in table 3 there are 4 daily prompts). The second argument is the result from the function `relevantREFVN`.

```
RELEVANTVN_REF <- setREF(4, relRef)
```

`setREF` returns a list (7) to the user. Subsequent functions will use this list, therefore if this list is not created or if the name is deleted from the R workspace, subsequent functions can't (co-)operate. Recommended name for that list: `RELEVANTVN_REF`.

5.2.3 relevantESVN (Function 3 of 28)

`relevantESVN` has the same purpose as the function `relevantREFVN` (5.2.1), i.e. to make known the most relevant variable names, only this time of the raw ESM dataset(s). The meaning as well as the order of the arguments expected by `relevantESVN` is:

1. `svyName`: The column of the ESM dataset that holds the specific ESM questionnaire version as a character (7) string. This argument is mandatory for combinations C4–C6 in table 1, it is optional for combinations C1–C3.

2. `IMEI`: The column of the ESM dataset that holds the IMEI (MEID) numbers of the mobile devices that have been used by the research participants.

3. `STARTDATE`: The column of the ESM dataset that holds the date at which each ESM questionnaire was started.

4. `STARTTIME`: The column of the ESM dataset that holds the time at which each ESM questionnaire was started.

5. `ENDDATE`: The column of the ESM dataset that holds the date at which each ESM questionnaire was ended.

6. `ENDTIME`: The column of the ESM dataset that holds the time at which each ESM questionnaire was ended.

The result of the function `relevantESVN` is assigned to the variable `relEs`:

```
relEs <- relevantESVN("survey_name", "IMEI", "start_date", "start_time",
                      "end_date", "end_time")
```

The result of `relevantESVN` can be displayed:

```
$ES_SVY_NAME
[1] "survey_name"
$ES_IMEI
[1] "IMEI"
$ES_START_DATE
[1] "start_date"
$ES_START_TIME
[1] "start_time"
$ES_END_DATE
[1] "end_date"
$ES_END_TIME
[1] "end_time"
$ES_DATETIMES_SEP
[1] TRUE
```

The last element `ES_DATETIMES_SEP` fulfills the same purpose as does `REF_DATETIMES_SEP` in function `relevantREFVN` (5.2.1).

5.2.4 setES (Function 4 of 28)

The function `setES` sets up all that the **esmprep** package requires concerning the ESM dataset. `setES` expects 4 arguments, of which all 4 are mandatory for all combinations C1–C6 in table 3. The first argument is the number of daily prompts (in the example in table 3 there are 4 daily prompts). The second argument is a vector (7) of all IMEI numbers used in the ESM project. The IMEI numbers, however, must not be numeric (7), instead they must be character (7) strings, like this:

```
imeiNumbers <- c(
"526745224593822", "526745224599058", ..., "526745224592943")
```

Tip: Since there must be a column in the reference dataset, which holds the IMEI number of each participant, you don't have to retype all IMEI numbers, but instead use the already existing column in the reference dataset:

```
imeiNumbers <- unique(as.character(referenceDf[,"imei"]))
```

The third argument again must be a vector (7) of character (7) strings, specifying all the ESM questionnaire versions used in the study, like this:

```
surveyNames <- c(
"morningTestGroup", "dayTestGroup", "eveningTestGroup",
"morningControlGroup", "dayControlGroup", "eveningControlGroup")
```

The last argument is the result from the function `relevantESVN`, which in our examplary ESM data looks like this:

```
RELEVANT_ES <- setES(4, imeiNumbers, surveyNames, relEs)
RELEVANTINFO_ES <- RELEVANT_ES[["RELEVANTINFO_ES"]]
RELEVANTVN_ES <- RELEVANT_ES[["RELEVANTVN_ES"]]
```

`setES` returns 2 lists to the user. Subsequent functions will use both lists, therefore if they are not created or if the names are deleted from the R workspace, subsequent functions can't (co-)operate. Recommended names for that lists: `RELEVANTINFO_ES` and `RELEVANTVN_ES`.

5.3 esList (Function 5 of 28)

`esList`: Use for C4–C6 in table 1; optional, though redundant, for C1–C3.
`esList` includes all raw ESM datasets in one single data structure, in order to keep things together that will eventually be of a single form. For our 6 exemplary raw ESM datasets, of which each one must be of class data.frame[4] (7), it looks like this:

```
esLs <- esList(list(morningControl, dayControl, eveningControl, morningTest, dayTest,
       eveningTest), RELEVANTVN_ES)
```

The second argument RELEVANTVN_ES is one of the results of function `setES` 5.2.4.

5.4 genKey (Function 6 of 28)

`genKey`: Use for C1–C6 in table 1.
In order for the user to be in control of all the raw ESM data at any time, a unique number is assigned to each line of raw ESM data. For our exemplary ESM datasets this looks like this:

```
keyLs <- genKey(esLs)
```

The result of `genKey` is assigned to the variable `keyLs`. An excerpt of one of the raw ESM datasets with the new first column 'KEY' looks like this:

Table 4: Excerpt of a raw ESM dataset with the new first column 'KEY'.

KEY	survey_name	IMEI	start_date	start_time	end_date	end_time
1016	dayControlGroup	526745224596286	2007-07-30	21:00:28	2007-07-30	21:05:09
1017	dayControlGroup	526745224596286	2007-08-01	21:20:34	2007-08-01	21:29:57

[4]When using the function `read.table` as shown in section 5.1 the loaded dataset is of class data.frame.

5.5 genDateTime or splitDateTime (Function 7 of 28)

`genDateTime` and/or `splitDateTime`: Use for C1–C6 in table 1.
Some subsequent functions require the date and time information (e.g the time stamps of each ESM questionnaire) to be separate from one another. Other subsequent functions require date-time objects, i.e. the date and time information combined in one variable. The date and time information in both the reference dataset and the raw ESM datasets must contain either the separated or the combined form. Therefore if date and time information is separated the user must apply the function `genDateTime` (short for 'generate date-time'). For the reference dataset this looks like this:

```
referenceDfList <- genDateTime(referenceDf, "REF", RELEVANTINFO_ES,
RELEVANTVN_ES, RELEVANTVN_REF)
```

The first argument is either a single dataset (e.g. the reference dataset) or a list list (7) of datasets (e.g. the raw ESM datasets). The second argument is a character (7) string specifying whether it is the reference dataset or the (list (7) of) raw ESM dataset(s). The other arguments RELEVANTINFO_ES, RELEVANTVN_ES, and RELEVANTVN_REF are the results either of function `setES` or of function `setREF`. These latter 3 data objects are modified once, right here, by function `genDateTime`. Therefore, apart from the output dataset, they also need to be extracted from the result. For the reference dataset this looks like this:

```
# Extract reference dataset from output
referenceDfNew <- referenceDfList[["refOrEsDf"]]
# Extract extended list of relevant variables names of reference dataset
RELEVANTVN_REF <- referenceDfList[["extendedVNList"]]
```

The same procedure for the raw ESM datasets looks like this:

```
keyList <- genDateTime(keyLs, "ES", RELEVANTINFO_ES,
RELEVANTVN_ES, RELEVANTVN_REF)
# Extract list of raw ESM datasets from output
keyLsNew <- keyList[["refOrEsDf"]]
# Extract extended list of relevant variables names of raw ESM datasets
RELEVANTVN_ES <- keyList[["extendedVNList"]]
```

If, on the other hand, the date and time information already exists in combined form the user must apply the function `splitDateTime`. For the reference dataset this looks like this:

```
referenceDfList <- splitDateTime(referenceDf, "REF", RELEVANTINFO_ES,
RELEVANTVN_ES, RELEVANTVN_REF)
```

For the ESM dataset(s) this looks like this:

```
keyList <- splitDateTime(keyLs, "ES", RELEVANTINFO_ES,
RELEVANTVN_ES, RELEVANTVN_REF)
```

The resulting dataset will contain new variables. For instance, after `genDateTime` has been applied to the reference dataset the new variables are REF_START_DATETIME and REF_END_DATETIME.

For both functions `genDateTime` and `splitDateTime` there are 2 further functions that might be helpful for the user: `dateTimeFormats` and `dateTimeFormats2`. The purpose or both functions is to give some orientation if needed. Just enter the function like this:

```
dateTimeFormats()
```

It will print to the console:

```
  dateTimeFormat     examplaryInput                   output
1        ymd_hms 2017-02-06 07:11:23  2017-06-02 07:11:23 UTC
2        mdy_hms   02-06-17 07:11:23  2017-06-02 07:11:23 UTC
3        dmy_hms   06-02-17 07:11:23  2017-06-02 07:11:23 UTC
4         ymd_hm      17-02-06 07:11  2017-06-02 07:11:00 UTC
5         mdy_hm    02-06-2017 07:11  2017-06-02 07:11:00 UTC
6         dmy_hm    06-02-2017 07:11  2017-06-02 07:11:00 UTC
```

Whereas `dateTimeFormats2()` will print to the console:

```
[1] "ymd_hms" "mdy_hms" "dmy_hms" "ymd_hm" "mdy_hm" "dmy_hm"
```

The meaning of it is: ymd = year month day. hms = hours minutes seconds.

6 ESM preparatory functions

Each function will be introduced by a very short explanation of its purpose, e.g. what kind of error it handles.

6.1 rmInvalid (Function 8 of 28)

`rmInvalid`: Use for C1–C6 in table 1.
Error: A line of raw ESM data does not contain any data whatsoever. Instead there is a warning message of some sort[5]. Instead of searching for the word 'Warning' `rmInvalid`

[5]According to an app developer a particular warning message was due to the participant using the quotes-symbol in a textfield when answering the respective ESM question.

registers a line of data to be invalid if there exists neither a start date nor a start time for a questionnaire. Applying the function to our exemplary ESM data:

```
rmInvLs <- rmInvalid(keyLsNew, RELEVANTVN_ES)
```

The second argument RELEVANTVN_ES is one of the results either of function `genDateTime` or of function `splitDateTime`.
`rmInvalid` displays to the console some rudimentary information about whether and how many lines of data were removed due to being invalid. The result of `rmInvalid`, here assigned to the variable **rmInvLs**, is a named list (7) with 4 elements. The elements' names and content are:

1. **dfValid**: Same content as in the result of esList, except that lines of data were removed when registered to be invalid.

2. **listInvalid**: A list (7) with all removed lines of data, which consists only of missing values and/or a warning message.

3. **rmInvalidDfFinished**: A logical value. `TRUE` if all datasets within the list (7) of raw ESM datasets where searched for invalid data, else `FALSE`.

4. **noLinesRemovedAtAll**: A vector (7) of logical values. `TRUE` whenever at least one line of data was registered to be invalid, else `FALSE`.

Relevant to the user is the first element only, **dfValid**, which is used in the subsequent function `esPlausible` (6.4). However, if the invalid data shall be displayed in the console, the user can do this by applying the function `printRmInvalid` (6.2).

6.2 printRmInvalid (Function 9 of 28)

`printRmInvalid`: Use for C1–C6 in table 1. Yet only if `rmInvalid` has been applied before.
Error: None. `printRmInvalid` prints to the console what the user wants to be displayed, concerning the questionnaires that have been removed by `rmInvalid`. The first argument is the result of the function `rmInvalid`. The last argument `smr` (short for summarize) can be `"tabulate"` (which is the default), it can be `"detail"`, and it can be `"both"`. In response to `"tabulate"` `printRmInvalid` displays a table, showing the number of removed questionnaires for each ESM version. The argument `"detail"` displays the removed questionnaires in full detail. The argument `"both"` displays both the table and the removed questionnaires in detail.

```
key_rmLs <- printRmInvalid(rmInvLs, RELEVANTVN_ES, smr="tabulate")
```

It's result for our exemplary datasets looks like this:

```
Summarizing table:
            versions notRemoved removed  Sum
1 morningControlGroup        14       0   14
2     dayControlGroup        89       1   90
3 eveningControlGroup        27       1   28
4    morningTestGroup        20       0   20
5        dayTestGroup        65       1   66
6    eveningTestGroup        13       1   14
7                Sum       228       4  232
```

Also `printRmInvalid` returns a list (7) to the user with as many elements as there are ESM versions. For each of the ESM versions, the value of the variable **KEY** (see function `genKey` (5.4)) is contained for all of the removed questionnaires. In case that there was an incorrect removal, the user will easily be able to reinsert the data.

6.3 esItems (Function 10 of 28)

`esItems`: Use for C4–C6 in table 1; optional, though redundant, for C1–C3.
Error: None. `esItems` is a preparatory function for the following function `esPlausible` (6.4), by extracting only the necessary information for it: The questionnaire items.
Applying the function for our exemplary data looks like this:

```
plausibItems <- esItems(dfList=rmInvLs[["dfValid"]], RELEVANTVN_ES)
```

In R you access the content of a list (7) with double square brackets and the name of the accessed element in quotes; optionally instead of the quoted name you can use the index number of the element, in this case it would be the first element (`rmInvLs[[1]]`).

6.4 esPlausible (Function 11 of 28)

`esPlausible`: Use for C4–C6 in table 1; optional and partly useful for C1–C3.
Errors: Variable names in at least 2 different ESM questionnaire versions are identical, althought the item content is different; The range of item values for items with identical names are different from one another; An item contains no values at all; etc.
Optimally `esPlausible` is applied in the pilot phase of the ESM project, when it makes most sense to have all combinations of variable names and their corresponding item content to be correct. This should be written down (e.g. paper, Excel, etc.) and be up to date! However, at the latest, `esPlausible` should be applied before searching any other errors in the raw ESM datasets. If this is not checked more errors will be generated!

Applying `esPlausible` for our exemplary ESM data looks like this:

```
plausibLs <- esPlausible(dfList=rmInvLs[["dfValid"]],
                         itemVecList=plausibItems)
```

`esPlausible` returns a list (7) with 4 elements:

1. **plausibNames**: Overview of the variable names as specified by the user.

   Table 5: Excerpt of list (7) element **plausibNames**.

namesAll	names1	names2	names3	names4	names5	names6	namesCheck
V1	V1	V1	V1	V1	V1	V1	TRUE
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
V2_1	V2_1	V2_1	V2_1	V2_1	V2_1	V2_1	TRUE
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
V8			V8		V8	V8	TRUE

   Note. Variable V8 only in versions 3, 5, and 6.

2. **plausibClass**: Overview of the variable classes as registered by R.

   Table 6: Excerpt of list (7) element **plausibClass**.

namesAll	class1	class2	class3	class4	class5	class6	classCheck
V1	num	num	num	num	num	num	TRUE
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
V2_1	log	int	int	int	int	int	FALSE
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
V8			cha		cha	cha	TRUE

   Note. Inconsistent classes in variable V2_1.

3. **plausibRowNa**: Overview of number of datalines and percentage of missing values.

   Table 7: Excerpt of list (7) element **plausibRowNa**.

namesAll	nRows1	naPercent1	...	nRows5	naPercent5	nRows6	naPercent6
V1	14	0	...	65	0.0	13	0.0
⋮	⋮	⋮	...	⋮	⋮	⋮	⋮
V2_1	14	100	...	65	3.1	13	7.7
⋮	⋮	⋮	...	⋮	⋮	⋮	⋮
V8			...	65	0.0	13	0.0

   Note. In variable V2_1 (version 1) all values (100%) are missing. The values TRUE, FALSE, and NA have the class logical. nRow specifies the number of questionnaires.

4. **plausibMinMax**: Overview of minimum and maximum numeric values.

   Table 8: Excerpt of list (7) element **plausibMinMax**.

namesAll	min1	max1	min2	max2	min3	max3	min4	max4	min5	max5	min6	max6
V1	0	3	−1	3	0	3	0	3	0	3	0	3
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
V2_1			0	98	0	99	1	97	0	100	12	98
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
V8												

Note. V1 contains a negative value (version 2), V8 contains contains text (no min, max possible).

Note that `esPlausible` can merely give hints about errors. Only someone with knowledge about the items' content and about the expected values – i.e. their range – in the raw ESM dataset can find out whether implausibilities point to actual errors or not.
In our exemplary ESM dataset some of the implausibilities have turned out to be errors as described at the beginning of subsection 6.4. One hint can be seen in table 6.
It turned out that V2_1 in version 1 contains no values, i.e. all values are NA. Since `class(NA)` returns the class logical, the variable correctly is of that class.
If, however, the plausibility check turned out that this variable contained values, TRUE and FALSE in the case of the class logical, then the variable name V2_1 would have been pointed to an error, because the other variables of the same name contain integers, i.e. numbers. In such a hypothetical case, when there are identical variable names for different items, prior to any proceedings a new variable name must be assigned, e.g.

```
# Change column name in version 1 at index 6 (which so far is V2_1)
colnames(rmInvLs[["dfValid"]][[1]])[6] <- "V2_2"
```

During the pilot phase of the ESM project the app developer should change the relevant variable name(s). However, if the ESM project was over, somebody else would have to change the variable name(s) accordingly[6].
Another error turned up only by inspecting **plausibMinMax** (see table 8). There was a negative value in version 2 of item V1.
After all errors were found and corrected, again apply `esItems` (6.3) and `esPlausible` (6.4), in order to see whether all corrected errors don't show up any more.
Note in particular that inspecting all 4 list (7) elements returned by `esPlausib` is important, not just inspecting 1 or 2 elements.

6.5 esComplete (Function 12 of 28)

`esComplete`: Use for C1–C6 in table 1.
Error: An ESM questionnaire has neither an end date nor an end time, nonetheless the questionnaire might be complete, i.e. the last item of the questionnaire that is expected to contain a value in order to be considered complete, acually contains a value.

[6] Choose some variable name **that so far isn't** in any of the ESM dataset(s).

`esComplete` expects 2 arguments, the first of which is the current state of the raw ESM dataset(s). The second argument might seem difficult at first glance. It isn't.

```
lastItemList <- list(
                      list("morningTestGroup", "V6", 0, "V6_1"),
                      list("dayTestGroup", NA, NA, "V7"),
                      list("eveningTestGroup", "V9", 1, "V9_1"),
                      list("morningControlGroup", "V6", 0, "V6_1"),
                      list("dayControlGroup", NA, NA, "V7"),
                      list("eveningControlGroup", "V8_1", 1:5, "V8_3"),
                      list("eveningControlGroup", "V8_1", 0, "V8_2"))
```

The second argument in `esComplete` will be the variable **lastItemList**. It is a list (7) of lists, i.e. (inner) lists nested within one (outer) list. The upper example is undoubtably a complex one. However, once understood, all simpler cases can then be applied very easily. Let's translate the core principle:

Example 1:

```
                      list("morningTestGroup", "V6", 0, "V6_1"),
```

In the ESM version `morningTestGroup`, if the variable `V6` is 0, then variable `V6_1` is the last item that is expected to contain a value, otherwise `V6` is the last item that is expected to contain a value.

What if there is no such condition? That makes things much easier!

Example 2:

```
                      list("dayTestGroup", NA, "V7", NA),
```

In the ESM version `dayTestGroup` the variable `V7` is the last item that is expected to contain a value. Because there is no condition, the user must set the second and third list (7) element to NA.

But what if there are 2 or even more conditions? That makes things a bit more difficult.

Example 3:

```
                      list("eveningControlGroup", "V8_1", 1:5, "V8_3"),
                      list("eveningControlGroup", "V8_1", 0, "V8_2")
```

Since the ESM version `eveningControlGroup` has two conditions, just repeat the same procedure for both conditions, as learned in **example 1**.

If variable `V8_1` is between 1 and 5, then variable `V8_3` is expected to be the last variable to contain a value.

If variable `V8_1` is 0, then variable `V8_2` is expected to be the last variable to contain a

value.

Now that the second argument can be generated, apply the function `esComplete` like this[7]:

```
isCompleteLs <- esComplete(rmInvLs[["dfValid"]], lastItemList)
```

The result is assigned to the variable **isCompleteLs**. It is a named list (7) with 7 elements, because the procedure for the last ESM version was applied twice, whereas for the remaining 5 ESM versions it was applied just once. This is also represented in the names of the resulting list:

```
names(isCompleteLs)
[1]"morningTestGroup_1"  "dayTestGroup_1"      "eveningTestGroup_1"
[4]"morningControlGroup_1" "dayControlGroup_1" "eveningControlGroup_1"
[7]"eveningControlGroup_2"
```

Each single raw ESM dataset contains the new column **INCOMPLETE_i**, where **i** represents the incrementing number of how often the procedure was applied within `esComplete`. For our exemplary ESM datasets the procedure was applied once for the versions 1–5, which is why the new column is **INCOMPLETE_1**. Only version 6 has the column **INCOMPLETE_1** and **INCOMPLETE_2**, because the procedure was applied twice. The column **INCOMPLETE_i** is dichotomous, representing a questionnaire to be complete by the value 0. Incompleteness therefore is represented by the value 1. See version 2 (`"dayTestGroup_1"`). Of 65 lines of data 63 are complete, 2 are incomplete.

```
table(isCompleteLs[["dayTestGroup_1"]]["INCOMPLETE_1"])
 0  1
63  2
```

If for every ESM version the procedure must be applied once (unlike **example 3**), then the new column's name will be **INCOMPLETE**, i.e. without any number at the end.

Table 9: Incomplete ESM questionnaires in version **dayTestGroup_1**.

KEY	survey_name	end_date	end_time	V7	INCOMPLETE_1
1165	dayTestGroup				1
1166	dayTestGroup				1

Note. Neither end date nor end time and no value in variable V7.

[7]The first argument is the result from function `rmInvalid` (6.1). More specifically, it is list (7) element `"dfValid"`, since `rmInvalid` returns a list with 4 elements. Also `esPlausible` (6.4) should have been applied as advised in this vignette, before applying `esComplete`!

6.6 esMerge (Function 13 of 28)

`esMerge`: Use for C4–C6 in table 1, not for C1–C3.

Error: None. This function merges all raw ESM datasets into one single dataset, with which the error handling procedure will then proceed. Apply `esMerge` like this:

```
esMerged <- esMerge(isCompleteLs, RELEVANTVN_ES)
# If preferred convert the 15 digit IMEI number from scientfic notation to text.
esMerged[,RELEVANTVN_ES[["ES_IMEI"]]] <-
as.character(esMerged[,RELEVANTVN_ES[["ES_IMEI"]]])
```

After merging the raw ESM datasets, lets take a look at an excerpt of the structure of our exemplary ESM dataset:

```
'data.frame': 228 obs. of 32 variables:
 $ KEY             : int 1015 1153 1154 1016 1155 1156 1157 1158 1017 1159 ...
 $ V1              : num 1 0 1 3 1 0 2 3 2 1 ...
 $ V1_1            : int 1 1 0 1 1 0 0 0 1 0 ...
 $ survey_name     : chr "dayControlGroup" "dayTestGroup" "dayTestGroup" ...
 $ IMEI            : chr "526745224596286" "526745224596286" "526745224596286" ...
 $ ES_START_DATETIME: POSIXct, format: "2007-07-29␣21:00:28" "2007-07-30␣13:01:46" ...
 $ ES_END_DATETIME : POSIXct, format: "2007-07-29␣21:07:52" "2007-07-30␣13:06:40" ...
 $ INCOMPLETE_1    : num 0 0 0 0 0 0 0 0 0 0 ...
 $ INCOMPLETE_2    : num NA NA NA NA NA NA NA NA NA NA ...
```

6.7 findChars (Function 14 of 28)

`findChars`: Use for C1–C6 in table 1.

Error: None. `findChars` is a preparatory function for the following function `convertChars` (6.8), in that it helps to find the variables within the (merged) raw ESM dataset which contain text. For our exemplary ESM data apply `findChars` like this:

```
findTextIdx <- findChars(esMerged)
```

The result (as printed by `findChars` to the console) might look like this[8]:

```
findTextIdx <- findChars(esMerged)
'data.frame': 228 obs. of 10 variables:
 $ V1_2      : chr "erklärt" "Börse" "erklärt" "Schloß" ...
 $ V3_1      : chr "Jahr" "Börse" "entrée" "Übung" ...
 $ survey_name: chr "dayControlGroup" "dayTestGroup" "dayTestGroup" ...
 $ IMEI      : chr "526745224596286" "526745224596286" "526745224596286" ...
 $ start_date : chr "2007-07-29" "2007-07-30" "2007-07-30" "2007-07-30" ...
```

[8]Since R doesn't permit umlauts in a package, there are no umlauts in the exemplary ESM datasets.

```
$ start_time : chr "21:00:28" "13:01:46" "17:00:11" "21:00:28" ...
$ end_date   : chr "2007-07-29" "2007-07-30" "2007-07-30" "2007-07-30" ...
$ end_time   : chr "21:07:52" "13:06:40" "17:08:19" "21:05:09" ...
$ V8         : chr NA "" "" NA ...
$ V9_1       : chr NA NA NA NA ...
```

In the exemplary ESM datasets delivered with the **esmprep** package there are some meaningless words, representing participant's text answers. Some of these words might contain letters like ä, ü, or ß – they very often appear in German text. In other countries these might be other words like 'entrée' in French.
`findChars` returns the search result for all variables of class character (7). However, the user must specify which of these variables contain letters that need to be converted. In our exemplary ESM dataset the variables `V1_2`, `V3_1`, `V8`, and `V9_1` might need conversion. `findChars` returns the indices of the variables containing text ...

```
findTextIdx
      V1_2       V3_1 survey_name      IMEI  start_date start_time   end_date
         3         11           20        21          22         23         24
  end_time         V8       V9_1
        25         30         41
```

... the user must pick from them, like this:

```
findTextIdx1 <- findTextIdx[c(1,2,9,10)]
```

Elements 1, 2, 9, and 10 within the variable **findTextIdx** are the target variables within the ESM dataset to pick. Notice the new assignment to variable **findTextIdx1**:

```
names(findTextIdx1)
[1] "V1_2" "V3_1" "V8" "V9_1"
```

Next, before applying the function `convertChars` we must generate one of the arguments, which `convertChars` needs. This argument must be a data.frame (7). The first column of it includes the letters that shall be converted, the second column of it includes the letters after the conversion. This data.frame (7) might look like this:

Table 10: On = before conversion, Off = after conversion.

On	Off
ä	ae
ß	ss
é	e
è	e
Ä	Ae

6.8 convertChars (Function 15 of 28)

`convertChars`: Use for C1–C6 in table 1.

Error: Depending on the encoding system somebody uses when reading data files, some letters can cause trouble. Sometimes the data file is not even imported, instead the user might receive an error message about so-called escape sequences. In order to avoid any trouble that might be caused by encoding systems, troublesome letters as specified by the user might be converted to letters known not to cause any trouble. For our exemplary ESM dataset this looks like this:

```
esMerged1 <- convertChars(esMerged, textColumns, convertCharsDf)
```

The first argument in `convertChars` is the current state of the ESM dataset. The second argument are the column names of the variables that contain text within which there might be letters that need to be conversed. The second argument can easily be obtained by using the result of `findChars`, like this:

```
textColumns <- names(esMerged)[findTextIdx1]
```

The third argument is the conversion data.frame (7), an example of which is table 10. There is a fourth argument which is optional: `ignoreCase`. Per default it is set to `FALSE`. If upper/lower case can be ignored when applying the conversion, the user has to set it to `TRUE`, like this:

```
esMerged1 <- convertChars(esMerged, textColumns, convertCharsDf, TRUE)
```

After the conversion the result in our exemplary ESM dataset (only the relevant variables) looks like this:

```
str(esMerged1[,findTextIdx1])
'data.frame':  228 obs. of 4 variables:
 $ V1_2      : chr "erklaert" "Boerse" "erklaert" "Schloss" ...
 $ V3_1      : chr "Jahr" "Boerse" "entree" "Uebung" ...
 $ V8        : chr NA "" "" NA ...
 $ V9_1      : chr NA NA NA NA ...
```

6.9 esAssign (Function 16 of 28)

`esAssign`: Use for C1–C6 in table 1.

Error: None. `esAssign` can be considered to be the centre of the **esmprep** package. `esAssign` assigns the lines of data within the raw ESM dataset to the participants which are listed in the reference dataset (see subsection 5.1 as well as table 3).

Applying `esAssign` to our exemplary ESM dataset looks like this:

```
esAssigned <- esAssign(esDf = esMerged1, refDf = referenceDfNew, RELEVANTINFO_ES,
RELEVANTVN_ES, RELEVANTVN_REF)
```

The first argument in `esAssign` is the ESM dataset in its current state. The second argument is the reference dataset. `esAssign` with these 2 arguments will do its job as described above.
`esAssign` comes with 5 optional arguments:

- **singleParticipant**: If the user wishes to have the ESM data to be assigned to one specific participant, then the participant's ID – as specified in the reference dataset – must be passed to the argument **singleParticipant**. Per default the ESM data is assigned to all participants listed in the reference dataset.

- **prompted**: If none of the ESM questionnaires were prompted, i.e. all ESM questionnaires were event contingent (see ESM combination C2 and C5 in table 1), then the user must pass `TRUE` to the argument **prompted**.

- **promptTimeframe**: If a single questionnaire must have been started within a specified time frame around each single prompt, e.g. 30 minutes, in order to be valid, then the user must pass that time frame in minutes to the argument **promptTimeframe**.

- **midnightPrompt**: If the survey app enabled the participant to start a questionnaire after midnight even though it was prompted before midnight, then the user must pass `TRUE` to the argument **midnightPrompt**. Per default it is set to `FALSE`.

- **dstDates**: If the user wishes to make a check on whether the time frame around each single prompt was influenced by the daylight saving time (the shifting of 1 hour at the last weekend in March and in October), then the user must pass the daylight saving date to the argument **dstDates**[9].

Additional details on why the arguments **promptedTimeframe** and **dstDates** contribute important information:
The argument **dstDates** makes sense in case that the time app on the mobile device, used by the participants, is assumed not to have the correct time all the time. The argument **dstDates** checks if the ESM period of each single participant includes the **dstDates**. An example of why this can be important information is when the participant gets reimbursed for the ESM questionnaires that have been filled out within a certain time frame, say 30

[9] **dstDates** can also be a vector of dates, in case the ESM project spans over an entire year or longer.

minutes. If the mobile device of the participant does not have the correct time all the time and the **dstDates** is included in the individual's ESM period, then some questionnaires might not get reimbursed for being outside of the 30 minutes time frame. However, they might be inside the time frame, only the 1 hour shift might make them seem to be outside of it.
Applying `esAssign` not only assigns the ESM data to the selected participants, it also prints information to the console, which for our exemplary ESM dataset looks like this:

```
P004
No. 4 of 8
Daylight saving time is contained in the ESM period.
Event sampling period - completion rates per prompt
   1   2   3   4
% 100 100 100 87.5
```

The printed information concerns the participant's ID, whether the daylight saving date (**dstDates**) is included in the participant's ESM period, and how many of each of the prompts have been answered by the participant (in percent) during the ESM period.
By applying the function to the ESM dataset the user receives a list with 3 elements:

1. **ES**: a data.frame (7). This dataset contains all the ESM questionnaires that were assigned to the participant(s), which are listed in the reference dataset. Also this dataset now contains additional columns:

    - **ID**: Unique identification code of each participant.
    - **CV_ES**: CV is short for count variable. It counts all the questionnaires that have been filled out by the participant during the ESM period. In incrementing order it starts at 1 and skips a number, whenever a questionnaire is missing.
    - **CV_ESDAY**: This variable counts the single ESM days. In incrementing order it starts at 1 and skips a number, whenever all questionnaires of that day are missing.
    - **CV_ESWEEKDAY**: This variable counts the weekday, with Monday represented by the value 1, ..., Sunday = 7.
    - **PROMPT**: Correspondance of the actual start time of the questionnaire to its prompt (in our examplary dataset this ranges between 1 and 4).
    - **PROMPTEND**: Correspondance of the actual end time of the questionnaire to its prompt (in our examplary dataset this ranges between 1 and 4).
    - **ES_MULT**: Dichotomous variable. The value 1 represents a questionnaire that has been filled out repeatedly at one specific prompt.

- **ES_MULT2**: Alternative representation of ES_MULT. The very first questionnaire at a prompt is represented by the value 1, the second questionnaire (i.e. the first repeatedly filled out q.) is represented by the value 2, etc.
- **ST**: Assigns the prompt/scheduled time (ST) to the actual start time of a questionnaire, by choosing the miminal time difference between all possible prompts (per participant) and the actual start time of the single ESM questionnaire.
- **STDATE**: Variable is returned only if argument midnightPrompt is set to TRUE. Possible values and meaning: -1 = scheduled start date is prior to actual start date; 0 = scheduled start date and actual start date are equal to one another; 1 = scheduled date is subsequent to actual start date.
- **TFRAME**: Dichotomous variable. The value 1 represents a questionnaire that is within the time frame, as specified by the user.
- **DST**: Dichotomous variable. The value 1 represents a questionnaire's date to be equal or later than the daylight saving date.
- **QWST**: Dichotomous variable. The value 1 represents a questionnaire to be fully within the scheduled time, i.e. the time differences of both the actual start time and the actual end time are minimal relative to the same scheduled time.

2. **ESopt** a data.frame (7). This dataset contains the most relevant information for each participant in terms of comparing the actual ESM dataset (after `esAssign` has been applied) with the ESM dataset considered to be optimal (short: opt). The optimal dataset exists when each and every participant filled out each of the prompted questionnaires during the ESM period. The user might want to find out how many questionnaires (in percent) were filled out on average across all the participants of the ESM study.

3. **ESout**: a data.frame (7). This dataset contains all the ESM questionnaires that were not assigned to the participant(s), which are listed in the reference dataset. One of the possible reasons ESM questionnaires were not assigned is that a participant's information has not yet been added to the reference dataset.

4. **ESrate**: a data.frame (7). This dataset contains the average completion rates per participant, both per prompt and overall.

6.10 missingEndDateTime (Function 17 of 28)

`missingEndDateTime`: Use for C1–C6 in table 1.
Error: Some of the ESM questionnaires' end date and end time are missing. `missingEndDateTime` finds out about this.
For our exemplary ESM dataset applying `missingEndDateTime` looks like this:

```
noEndDf <- missingEndDateTime(esAssigned[["ES"]], RELEVANTVN_ES)
```

`missingEndDateTime` expects the ESM dataset in its current state as its first argument. Applying the function to the ESM dataset adds 2 additional columns to it, named **NOENDDATE** and **NOENDTIME**. They are dichotomous, representing a missing end date and end time by the value 1, existing end date and end time by the value 0.

6.11 esIdentical (Function 18 of 28)

`esIdentical`: Use for C1–C6 in table 1.
Error: Some ESM questionnaires, after having downloaded them, exist in duplicated form, i.e. one specific line of ESM data identically exists more than once in the raw ESM dataset. `esIdentical` finds out about this.
For our exemplary ESM dataset applying `esIdentical` looks like this:

```
identDf <- esIdentical(noEndDf, RELEVANTVN_ES)
```

`esIdentical` expects the ESM dataset in its current state as its first argument. Applying the function to the ESM dataset adds 1 additional column to it, named **IDENT**. It is dichotomous, representing identical questionnaires by the value 1, unique questionnaires by the value 0.

6.12 suggestShift (Function 19 of 28)

`suggestShift`: Use for C1, C3, C4, and C6 in table 1.
Error: Some ESM questionnaires that were registered by `esAssign` to have been filled out repeatedly at a given prompt, might be shifted backward or forward by the user – due to some condition. In the following example this condition is: If between the scheduled time and the actual start time of a questionnaire 100 minutes or more have passed, suggest that questionnaire to be shifted to a neighboring prompt index. Applying `suggestShift` for our exemplary dataset looks like this:

```
sugShift <- suggestShift(identDf, 100, RELEVANTINFO_ES, RELEVANTVN_ES)
```

What is printed to the console by `suggestShift` looks like this:

```
     ID  KEY    survey_name CV_ES CV_ESDAY ES_START_DATETIME    ST
159 P001 1161   dayTestGroup   19       6    2007-08-03 13:07:46 13:00:00
19  P001 1019 dayControlGroup 21       6    2007-08-03 19:17:31 21:00:00
20  P001 1020 dayControlGroup 21       6    2007-08-03 21:05:21 21:00:00

   PROMPT PROMPTEND ES_MULT SHIFT SHIFTKEY LAG_MINUTES
159      2         2       0     0       NA          NA
19       4         4       0     1     1019        -102
20       4         4       1     0       NA          NA
```

Let's take a look at the reference dataset for P001.

Table 11: Extraction of the reference dataset.

id	st1	st2	st3	st4
P001	09:00:00	13:00:00	17:00:00	21:00:00

Note. id = identification code, st = scheduled time

When comparing what was printed to the console with the relevant data in the reference dataset (see table 11) we notice for P001 that the assigned **PROMPT** that is suggested to be shifted (see column **SHIFT** = 1) belongs to a questionnaire that was started 2 hours and 17 minutes after the scheduled time of 17 o'clock. The interval of 4 hours between 2 subsequent prompts mean that the questionnaire 'almost' is in the middle between 2 prompts, therefore it might be justified to manually shift this particular questionnaire to the prior promt, i.e. from index 4 to index 3, since there is no questionnaire already assigned to prompt index 3.
If at least one questionnaire is suggested to be shifted to a neighboring prompt, the output of `suggestShift` is a list with 3 elements:

1. **esDf**: The raw ESM dataset in its current state.

2. **suggestShiftDf**: Relevant excerpt of each questionnaire that is suggested to be shifted to a neighboring prompt.

3. **printShiftDf**: Relevant information (concerning indices within the current raw ESM dataset), that is going to be used by the function `makeShift` (6.13).

However, if no questionnaire is suggested to be shifted to a neighboring prompt, the output of `suggestShift` is the data.frame that was passed to it as input.
Notice that `suggestShift` has added 2 columns to the current raw ESM dataset, named **SHIFT** and **SHIFTKEY**. If no questionnaire was suggested to be shifted, then **SHIFT** only contains 0, whereas **SHIFTKEY** only contains NA.

6.13 makeShift (Function 21 of 28)

`makeShift`: Use for C1, C3, C4, and C6 in table 1. Yet only if at least one questionnaire shall justifiably be shifted from one prompt index to another.
Error: None. `makeShift` executes the shifting. For our exemplary ESM dataset this looks like that:

```
madeShift <- makeShift(sugShift, referenceDfNew, keyPromptDf,
RELEVANTINFO_ES, RELEVANTVN_REF)
```

In order to execute the shifting `makeShift` expects as first argument the output of function `suggestShift` and as second argument the reference dataset. The third argument must be generated. To this end, part of the ouput of function `suggestShift` is of help. Select from the dataset **suggestShiftDf** the columns NEW_PROMPT and SHIFTKEY. If you don't want to shift all suggested questionnaires, select the rows of **suggestShiftDf**, that show the questionnaires you want to shift.

```
keyPromptDf <- sugShift$suggestShiftDf[,c("NEW_PROMPT", "SHIFTKEY")]
```

6.14 printSuggestedShift (Function 20 of 28)

`printSuggestedShift`: Use only if `suggestShift` or `makeShift` has been applied before.
Error: None. `printSuggestedShift` prints to the console what is printed by the function `suggestShift`. Printing is the sole purpose of `printSuggestedShift`! This is useful to check if the shifting has been executed as ordered by the user. For our exemplary dataset this looks like this:

```
printSuggestedShift(madeShift, RELEVANTVN_ES)
```

What is printed to the console by `printSuggestedShift` now looks like this:

```
     ID  KEY    survey_name CV_ES CV_ESDAY ES_START_DATETIME    ST
159 P001 1161   dayTestGroup   19       6    2007-08-03 13:07:46 13:00:00
19  P001 1019 dayControlGroup 20       6    2007-08-03 19:17:31 17:00:00
20  P001 1020 dayControlGroup 21       6    2007-08-03 21:05:21 21:00:00

    PROMPT PROMPTEND ES_MULT SHIFT SHIFTKEY LAG_MINUTES
159      2         2       0     0       NA          NA
19       3         3       0     1     1019        -102
20       4         4       0     0       NA          NA
```

The prompt index for P001, which before was 4 now is 3, both for **PROMPT** and for **PROMPTEND** (see line index number `19`).

6.15 expectedPromptIndex (Function 22 of 28)

`expectedPromptIndex`: Use for C1, C3, C4, and C6 in table 1.

Error: A questionnaire that belongs to a certain time of day (say morning) has been filled out at another time of day, when it **might** make no sense (say day or evening).

Before applying `expectedPromptIndex`, the second argument of the function must be generated. An example of the second argument four our exemplary ESM dataset looks like this:

```
expIdxList <- list(
                      list("morningTestGroup", 1, 1),
                      list("dayTestGroup", c(2,3), 2),
                      list("eveningTestGroup", 4, 3),
                      list("morningControlGroup", 1, 1),
                      list("dayControlGroup", c(2,3), 2),
                      list("eveningControlGroup", 4, 3))
```

General explanation: The second argument is a list (7) of lists (compare the second argument in subsection 6.5). In the inner lists, the first element is a character (7) string, specifying the ESM version, which in our exemplary ESM study refers to categories of the day, i.e. morning, day, and evening, each for a test and a control group. The second element in the inner lists either is a single numeric (7) value or a vector (7) of values, specifying the prompt indices that belong to the ESM version, which was specified as first element. The third element always is a single numeric (7) value, specifying what value we expect the second element of the inner lists to be combined with.

Specific explanation: For the morning version we expect the prompt index 1 (which belongs to the ESM version of 'morning') to be combined with the value 1, which I – as user – specify as the first category (morning category) of an ESM day.

```
                      list("morningTestGroup", 1, 1),
```

For the day version we expect the prompt indices 2 and 3 (which belong to the ESM version of 'day') to be combined with the value 2, which I – as user – specify as the second category (day category) of an ESM day.

```
                      list("dayTestGroup", c(2,3), 2),
```

For the evening version we expect the prompt index 4 (which belongs to the ESM version of 'evening') to be combined with the value 3, which I – as user – specify as the third category (evening category) of an ESM day.

```
                      list("eveningTestGroup", 4, 3),
```

`expectedPromptIndex` will then discover all prompt indices that diverge from the expection, as it was specified by the user.
Note that the chosen categories 1, 2, and 3 for the third element of the inner lists is arbitrary, as `expectedPromptIndex` only checks whether there are any divergences from the specified combinations that the user expects to occur under optimal conditions. The user could also have chosen 45, 29, and 2 instead of 1, 2, and 3, or any other values, as long as they differ from one another.
With the second argument now generated, applying `expectedPromptIndex` in our exemplary ESM dataset looks like this:

```
expectedDf <- expectedPromptIndex(madeShift$esDf, expIdxList,
RELEVANTINFO_ES, RELEVANTVN_ES)
```

The first argument is the ESM dataset in its current state. The second argument is the list (7) of lists we have just generated. An excerpt of the result of `expectedPromptIndex` for our exemplary ESM dataset looks like this:

Table 12: Excerpt of the result of `expectedPromptIndex`.

ID	survey_name	ES_START_DATETIME	PROMPT	PROMPTFALSE	EXPCATEGORY
P005	dayControlGroup	2007-11-06 09:06:56	1	1	2
P005	dayControlGroup	2007-11-06 13:03:06	2	0	2
P005	dayControlGroup	2007-11-06 17:14:19	3	0	2

Note. ID = identification code, EXPCATEGORY = expected category as specified by the user. The first questionnaire is unexpected: A day questionnaire (category 2) filled out in the morning (prompt 1).

Note that `expectedPromptIndex` adds 2 new columns to the ESM dataset in its current state, named **PROMPTFALSE** and **EXPCATEGORY**.
PROMPTFALSE is dichotomous. The value 1 represents a divergence from the expectation as specified by the user. The value 0 represents things to be as expected.
EXPCATEGORY displays the category that the user specified to be combined with the prompt index, which belongs to the ESM version (compare the explanation for how the second argument was generated for `expectedPromptIndex`). The ESM version in our exemplary ESM dataset is represented by the variable **survey_name** (see table 12).

6.16 intolerable (Function 23 of 28)

`intolerable`: Use for C1, C3, C4, and C6 in table 1.
Error: A questionnaire that belongs to a certain time of day (say morning) has been filled out at another time of day, when it **certainly** makes no sense at all (say day or evening).
Before applying `intolerable`, the second argument of the function must be generated.

An example of the second argument four our exemplary ESM dataset looks like this:

```
intoleranceDf <- data.frame(prompts=c(2:4, 1, 1), expect=c(rep(1,times=3), 2, 3))
```

The second argument for `intolerable` is a data.frame (7).

Table 13: Second argument for function `intolerable`.

prompt	expect
2	1
3	1
4	1
1	2
1	3

Note. Exactly 2 columns.

Table 13 translates to:
Day questionnaires (prompts 2 and 3) are not tolerated with category 1 (morning category). An evening questionnaire (prompt 4) is also not tolerated with category 1. A morning questionnaire (prompt 1) is not tolerated with category 2 (day category), or with category 3 (evening category). All other combinations therefore are tolerable and will thus not be removed from the ESM dataset.
Applying the function `intolerable` to our exemplary ESM dataset looks like this:

```
intolLs <- intolerable(expectedDf, intoleranceDf, RELEVANTINFO_ES)
```

The first argument expected by `intolerable` is the ESM dataset in its current state. The second argument is the data.frame (7) of intolerable combinations as we have generated it above. `intolerable` returns a list (7) to the user, containing 2 data.frame (7) s.

1. **cleanedDf**: This data.frame (7) contains all tolerable ESM data.
2. **intoleranceDf**: This data.frame (7) contains all intolerable ESM data.

An excerpt of the **intoleranceDf** for our exemplary ESM dataset looks like this:

Table 14: Excerpt of the result of `intolerable`, list element no.2: **intoleranceDf**.

ID	survey_name	ES_START_DATETIME	PROMPT	PROMPTFALSE	EXPCATEGORY
P003	dayControlGroup	2007-09-15 08:50:27	1	1	2

Note. ID = identification code, EXPCATEGORY = expected category as specified by the user.

Note for P003 that a day questionnaire (EXPCATEGORY = 2) was filled out at morning time (PROMPT = 1), which can't be tolerated in any further data analyses.

6.17 randomMultSelection (Function 24 of 28)

`randomMultSelection`: Use for C1, C3, C4, and C6 in table 1.
Error: At least one participant repeatedly filled out a questionnaire at one specific prompt and one specific day during the ESM period.
Multilevel statistical analyses deal with repeated measurements, yet each measurement is expected to be unique/single. Therefore multiple measurements per prompt and day usually are not wanted. Unless such multiple measurements per prompt and day are valid (see combination C5 in table 1), the repeatedly filled out questionnaires have to be removed from the ESM dataset.
`randomMultSelection` uses the variable **ES_MULT**, which was generated by the function `esAssign` (6.9), and (for each participant) randomly selects one questionnaire among the repeatedly filled out questionnaires (for each relevant prompt and day). The unselected questionnaires get discarded.
Applying the function `randomMultSelection` to our exemplary ESM dataset looks like this:

```
randSelLs <- randomMultSelection(intolLs[["cleanedDf"]])
```

`randomMultSelection` expects as only argument the ESM dataset in its current state. As result the user receives a list with 2 elements:

1. **esRandSelIn**: a data.frame (7). This dataset contains all the questionnaires of the ESM dataset at its current state, except for the ones that were discarded by the random deselection.

2. **esRandSelOut**: a data.frame (7). This dataset contains all the discarded questionnaires of the ESM dataset at its current state.

Note that `randomMultSelection` replaces the column **ES_MULT2** in the ESM dataset. It recomputes it, since the shifting of questionnaires to a neighboring prompt index has changed the number of questionnaires that were registered as being multiples in function `esAssign` (6.9).

6.18 computeTimeLag (Function 25 of 28)

`computeTimeLag`: Use for C1, C3, C4, and C6 in table 1.
Error: None. `computeTimeLag` computes the time difference between the scheduled start time (the prompt) of a questionnaire and the actual start time of it.
Applying the function `computeTimeLag` to our exemplary ESM dataset looks like this:

```
lagDf <- computeTimeLag(randSelLs[["esRandSelIn"]], RELEVANTVN_ES)
```

`computeTimeLag` expects as its first argument the ESM dataset in its current state. As result it is returned to the receiver with 3 new columns. The one column's name is **TIME_LAG**, which contains the time difference as described above. The next column's name is **LAG_PA**, which is dichotomous. A questionnaire that has been started **A**fter the scheduled time is represented by the value 1, else the actual start was **P**rior to the scheduled time, represented by the value 0. The last column's name is **ST_DATETIME**, which is the date-time object of the scheduled time/prompt.

6.19 computeDuration (Function 26 of 28)

`computeDuration`: Use for C1–C6 in table 1.
Error: None. `computeDuration` computes the time difference between the actual start time of a questionnaire and the actual end time of it. If there is no end time the function will return `NA` for this questionnaire.
Applying the function `computeDuration` to our exemplary ESM dataset looks like this:

```
durDf <- computeDuration(lagDf, RELEVANTVN_ES)
```

`computeDuration` expects as its first argument the ESM dataset in its current state. As result it is returned to the receiver with 1 new column, named **DUR**, which contains the time difference as described above.

6.20 computeTimeBetween (Function 27 of 28)

`computeTimeBetween`: Use for C1–C6 in table 1.
Error: None. For each participant `computeTimeBetween` computes the time difference between the actual end time of a questionnaire and the actual start time of the subsequent questionnaire. If there is no end time the function will return `NA` for the subsequent questionnaire.
Applying the function `computeTimeBetween` to our ESM dataset looks like this:

```
tbsqDf <- computeTimeBetween(esDf = durDf, refDf = referenceDfNew, RELEVANTVN_ES,
RELEVANTVN_REF)
```

`computeTimeBetween` expects as its first argument the ESM dataset in its current state and as second argument the reference dataset. As result the ESM dataset is returned to the user with 1 new column, named **TBSQ** (T_ime B_etween S_ubsequent Q_uestionnaires).

6.21 esFinal (Function 28 of 28)

`esFinal`: Use for C1–C6 in table 1.

Error: None. `esFinal` generates the final ESM dataset, i.e. for each missed questionnaire an empty line will be inserted. Also each participant in the final ESM dataset will have an equal amount of lines. To this end `esFinal` searches across all participants in the ESM dataset for the maximum number of valid questionnaires per participant. This maximum number then will determine the number of lines for all participants. Applying the function `esFinal` to our exemplary ESM dataset looks like this:

```
esDfFin <- esFinal(tbsqDf, RELEVANTINFO_ES, RELEVANTVN_ES, maxRows=28)
```

As its first argument `esFinal` expects the ESM dataset at its current state. A short explanation is necessary for what the last argument means: The number of questionnaires must be equal for all participants. Usually the actual number of questionnaires is not equal for all participants. Therefore the final ESM dataset must contain some empty lines of data (i.e. empty questionnaires) at the end of those participants who filled out fewer questionnaires than the participant(s) who have filled out the maximum number of questionnaires. If for some reason the user needs to constrain this maximum number of questionnaires to be less than what the actual maximum number is (if there is one heavy outlier participant), this constraint will be achieved by entering the (user-defined) maximum number as second argument. If the user doesn't need to make such a constraint, then the argument `maxRows` can be ignored:

```
esDfFin <- esFinal(tbsqDf, RELEVANTINFO_ES, RELEVANTVN_ES)
```

As result the final ESM dataset is returned to the user with 2 new columns, both being dichotomous. The one column's name is **MISSED**, i.e. all questionnaires that should have been answered by the participant but weren't (the value 1 represents a missing questionnaire). The other column's name is **FILLER**, i.e. the empty questionnaires at the end of each participant, that are needed to get the number of questionnaires to be equal for all participants.

Don't forget to save the final ESM dataset by writing it to the hard disc of your computer!

7 R glossary (in relation to the esmprep vignette)

Link: data.frame

Link: character

Link: list

Link: vector

Link: numeric