Assessment R Markdown file

B209223

21/6/2022

Link to Github Repository

https://github.com/B209223/B209223_assessment

Loading NHSRdatasets

Script and data information:

This script will load the Stranded Patient dataset from the NHSRdatasets package. It will explore and tabulate data on stranded patient in NHS services and save it to "RawData" folder.A subset of variables will be selected. It will then be splited into training and test data and save it to "Data" folder, which will be available for furhter analysis. Data consists of character data, numeric data and date from NHSRdatasets. These datasets have been synthetically generated. (Chris Mainey, 2021)

Load packages and data

To load the packages

```
library(NHSRdatasets) #data is from NHSRdatasets
library(tidyverse) #for data read-in and manipulation
library(here) #for data workflows
library(lubridate) #to deal with dates
library(caret) #to split data into training and testing dataset
library(knitr) #for dynamic report generation
```

To load the Stranded Patient dataset from NHSRdataset

```
data(stranded_data)
stranded <- stranded_data
```

To check for its class

```
class(stranded)
```

Class of stranded is "tbl_df", which means it's a tibble.

To get an overview of the stranded patient dataset.

```
stranded
```

The dataset has 768 rows and 9 columns, including three character variables (stranded.label, admit_date and frailty_index), as well as 6 numeric variables(age, care.home.referral, medicallysage, hcop, menatal_health_care and periods_of_previous_care). Description of selected variables is available on Data Dictionary.

To view the stranded patient data

A glimpse() function is used to view variables in this dataframe.

```
glimpse(stranded)
```

To look at top and bottom rows

```
head(stranded)
tail(stranded)
```

To check if there's missing data

```
stranded %>%
  map(is.na) %>%
  map(sum)
```

There's 69 data missing on periods_of_previous_care. We will leave them here because this variable is irrelevant to this study and will not be included in further analysis.

To add an index column to standed patient data.

Index column is set to link raw data with partitioned datasets.The rowid_to_column() can set index.

```
stranded <- rowid_to_column(stranded, "index")
```

To convert variable admit_date into a date variable

The admit_date is a character variable but we would like it to be a date and in date format, so a conversion is done here with a as.Date() function

```
stranded$admit_date <- as.Date(stranded$admit_date, format = "%d/%m/%Y")
```

To tabulate the raw data

```
kable(stranded)
```

To save the raw stranded patient data to "RawData" folder

```
write_csv(stranded, here("RawData", "stranded.csv"))
```

To select variables for the data capture tool

The project intends to explore relationship between one outcome (stranded vs non-stranded) and 3 predictors, namely age, mental_health_care and admit_date. So only 4 variables will be selected, namely stranded.label, age, mental_health_care and admit_date along with index.

```
stranded_4var <- stranded %>% select(index, stranded.label, age, mental_health_care, admit_date)
```

To tabulate and overview the subsetted data

```
stranded_4var %>% head(5) %>% kable()
glimpse(stranded_4var)
```

To save the subsetted data to the "RawData" folder

```
write_csv(stranded_4var, here("RawData", "stranded_4var.csv"))
```

To split the stranded_4var data into training and testing datasets

How many rows are there in the dataframe?

```
nrow(stranded_4var)
```

The dataset has 768 rows of data.

A testing dataset of 10-15 records is sufficient to evaluate the data capture tool. So the proportion of the raw data assigning to the training dataset is:

```
prop<-(1-(15/nrow(stranded_4var))) #0.9804688, so 2% to testing dataset and 98% to training dataset.
print(prop)
```

To split the stranded_4var data into testing and training dataset

A random seed is generated to ensure outputs are reproducible.

```
set.seed(42)
```

To set up the training dataset. The function createDataPartition() helps to split datasets randomly with indexing.

```
trainIndex <- createDataPartition(stranded_4var$index, p = prop,
                                  list = FALSE,
                                  times = 1)
head(trainIndex)
stranded_train <- stranded_4var[ trainIndex,]
nrow(stranded_train)
```

There are 756 rows in the training dataset.

To tabulate the training dataset and save to "Data" folder.

```
kable(stranded_train)
write_csv(stranded_train, here("Data", "stranded_train.csv"))
```

To set up the testing dataset by removing the training dataset from the original dataset.

```
stranded_test <- stranded_4var[-trainIndex,]
nrow(stranded_test)
```

There are 12 rows in the testing dataset, sufficient to evaluate the data capture tool.

To set aside one record for test markers, tabulate and save to "Data" folder.

```
stranded_TestMarker <- stranded_test[1, ]
kable(stranded_TestMarker)
write_csv(stranded_TestMarker, here("Data", "stranded_TestMarker.csv"))
```

To tabulate the testing dataset and save to "Data" folder.

```
write_csv(stranded_test, here("Data", "stranded_test.csv"))
```

End of Loading NHSRdatasets

Constructing a data dictionary and appending it to your data

Overview

The data dictionary contains information about variables, data collection and metadata, faciliating access and communication with other users. To create this data dictionary, the *dataMeta* R package is employed to document information for a subset of stranded patient data from NHSRdatasets. It then will be saved to "RawData" folder.

Load packages and read in the data

```
library(dataMeta) # to construct a data dictionary
library (tidyverse) # to read in datasets
library(here) # to reference in data workflows
```

Data

The data was subsetted from the stranded patient data from NHSRdatasets and provided information on stranded patients from 2020 Dec to 2021 Feb. Data was captured by the Jupyter Notebook "./ipynb-Scripts/CollectingDataUsingInteractiveJupyterWidgets.ipynb" and then splited into a training dataset and a testing dataset. It contains one outcome variable (stranded.label) and 3 predictors, namely age, mental_health_care and admit_date.

Let's use the `read_csv()` function from the *readr* package to read the collected data from the "RawData" folder.

```
CollectedData=read_csv(here("RawData", "CollectedDataFinal.csv"))
```

```
## Rows: 1 Columns: 6
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): stranded.label
## dbl  (3): index, age, mental_health_care
## lgl  (1): consent
## date (1): admit_date
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Let's view the CollectedData stranded patient data

The glimpse() function helps to view variables and their corresponding data types.

```
glimpse(CollectedData)
```

There are 6 columns in total: * **index:** the index column that serves to link the testing dataset with the original dataset.

- **stranded.label:** an binary outcome variable - whether patient is stranded or not.
- **age:** age of patients when they were admitted to hospital.
- **mental_health_care:** whether patients need mental health care.
- **admit_date:** dates when patients were admitted to hospital.
- **consent:** consent from users that they allow us to share and process data captured by the data capture tool.

Build a data dictionary for the data collected by the data capture tool

Build a linker data frame

A dataframe should be set up to present information of data type and description of each variable.

Variable descriptions

The description of each variable will be presented in a string vector.

```
variable_description <- c("The index column that serves to link the testing dataset with the original da
print(variable_description)
```

Variable types

We need to create a string vector representing the different variable types. It is a vector of integers with values 0 or 1, 0 for a variable with quantitative values and 1 for fixed values.

A glimpse() function is used to view data types.

```
glimpse(CollectedData)
```

We have 4 quantitative and 2 fixed values.

```
variable_type <- c(1, 1, 0, 0, 1, 1)
print(variable_type)
```

Now the`build_linker()` function from the *dataMeta* package is used to construct an intermediary (linker) data frame between the CollectedData and the data dictionary. For this function to run, it requires the CollectedData data frame and variable_description and variable_type string vectors as inputs.

```
linker<-build_linker(CollectedData, variable_description, variable_type)
print(linker)
```

Data dictionary

Now we are ready to build a data dictionary with `build_dict()` function from the *dataMeta*. The function requires the CollectedData and linker data frames and variable_description as inputs.

```
dictionary <- build_dict(my.data = CollectedData, linker = linker, option_description = NULL, prompt_va
glimpse(dictionary)
```

Let's save the data dictionary for CollectedData to the 'RawData' folder

```
glimpse(dictionary)
write_csv(dictionary, here("RawData", "CollectedData_DataDictionary.csv"))
```

Append data dictionary to the CollectedData

The 'incorporate_attr()' function from the *dataMeta* package is used to incorporate attributes as metadata.

```
main_string <- "This data describes the stranded patient data from 2020 Dec to 2021 Feb from the *NHSRd
main_string
```

Incorporate attributes as metadata The attributes to be included:

- a data dictionary
- number of columns
- number of rows
- the name of the author who created the dictionary and added it,
- the time when it was last edited
- a brief description of the original dataset.

```
complete_CollectedData <- incorporate_attr(my.data = CollectedData, data.dictionary = dictionary,
main_string = main_string)
#Change the author name
attributes(complete_CollectedData)$author[1]<-"B209223"
complete_CollectedData
attributes(complete_CollectedData)
```

Save the CollectedData with attributes The 'save_it()' function is used to save the metadata in "RawData" folder.

```
save_it(complete_CollectedData, here("RawData", "complete_CollectedData"))
```

End of Constructing a data dictionary and appending it to your data

Data capture tool: Collecting data using interactive Jupyter widgets

The data capture tool will use interative widgets written in Python language in Jupyter to collect data and then store them in "RawData" folder.