Assessment R Markdown

B203349

20/06/2022

Click here to link to my github repository

Loading the required packages required for this script

```
library(dataMeta)
library(tidyverse)
library(here)
library(NHSRdatasets)
library(knitr)
library(scales)
library(lubridate)
library(caret)
```

Loading NHS Datasets

Data

The **NHS England accident and emergency attendances and admissions (ae_attendances)** dataset will be loaded from the NHSRdatasets package. This dataset reports on the attendances, four-hour breaches and admissions for all A&E departments in England for the years 2016/17 through 2018/19 (Apr-Mar)

Loading ae_attendances data

ae_attendances data is loaded from the NHSdatasets package

```
data(ae_attendances)
ae<-ae_attendances
```

Viewing the data

The `head()` function is used to look at top n rows of a data frame (default n = 6 rows)

```
head(ae)
```

Check for missing data

Calculate how many NAs there are in each variable

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

Data is complete

Add an index link column to ae_attendances data

The `rowid_to_column()` function is used to convert row identities to a column named index. This facilitates the linking of partitioned datasets to the raw data if required in the future.

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

Preview dataset

Tabulate data, and create report with `kable()`

```
ae %>%
  mutate_at(vars(period), format, "%b-%y") %>%
  mutate_at(vars(attendances, breaches, admissions), comma) %>%
  head(10) %>%
  kable()
```

Save the raw ae_attendances data to 'RawData' folder

```
write_csv(ae, here("RawData", "ae_attendances.csv"))
```

Subsetting the data

The aim of my data capture tool is to help identify performance outliers with regard to the 4hr breaches. In order to do this effectively I will need use the entire dataset (i.e. the variables: index, period, attendances, breaches, organisation name, and organisation type).

Of note the **performance** variable was deliberately not created as this is a function of the **attendances** and **breaches** variables. The variable can be created after the collection of data at the data analysis stage.

Creating test and training datasets

Calculate the proportion (`prop`) of the raw data to assign to the training data.

The proportion of the raw data that needs to be assigned to the training data to ensure there is only 10 to 15 records in the test data is:

```
prop<-(1-(15/nrow(ae)))
print(prop)
```

Splitting the raw data

The createDataPartition() function from the *caret* package is used to splint the raw data into test and training data sets.

The `set.seed()` function allows a random number to be generated in a reproducible fashion. This ensures that every time the script is run the raw data will be partitioned into the same test and training datasets.

```
set.seed(333)
trainIndex <- createDataPartition(ae$index, p = prop,
                                  list = FALSE,
                                  times = 1)
head(trainIndex)
```

Assign all records in the trainIndex to the training data

```
aeTrain <- ae[ trainIndex,]
nrow(aeTrain)
```

Save the training data to the to working data folder 'Data' as ae_attendances_train.csv

```
write_csv(aeTrain, here("Data", "ae_attendances_train.csv"))
```

Extract the test data (all records that are not in the trainIndex (`-trainIndex`) are assigned to the test data).

```
aeTest  <- ae[-trainIndex,]
nrow(aeTest)
```

Set aside the fist record for markers to test and evaluate the data-capture tool.

```
aeTestMarker  <- aeTest[1,]
```

Save the marker test data to the to working data folder 'Data' as ae_attendances_test_marker.csv

```
write_csv(aeTestMarker, here("Data", "ae_attendances_test_marker.csv"))
```

The remaining records are set aside to test with the data capture tool. These are saved to the to working data folder 'Data' as ae_attendances_test.csv

```
aeTest  <- aeTest[2:nrow(aeTest),]
write_csv(aeTest, here("Data", "ae_attendances_test.csv"))
```

Data Dictionary for test data

Reading the collected data

`read_csv()` function from the *readr* package is used to read my collected data from the Raw data folder.

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

```
## Rows: 11 Columns: 8-- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): org_code, type
## dbl  (4): index, attendances, breaches, admissions
## lgl  (1): consent
## date (1): period
## 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.
```

Building a linker data frame

Variable descriptions

String vectors are created for the different variable descriptions

```
variable_description <- c("The index column that allows us to link the data collected to the original a
"The month that this activity relates to, stored as a date (1st of each month).",
"The Organisation data service (ODS) code for the organisation. If you want to know the organisation as
"The department type for this activity.",
"The number of attendances for this department type at this organisation for this month.",
"The number of attendances that breached the four-hour target.",
"The number of attendances that resulted in an admission to the hospital.",
"The consent from the end-user to process and share the data collected with the data capture tool.")
print(variable_description)
```

Variable types

A string vector is created representing the different variable types. It is a vector of integers with values 0 or 1. 0 is used for a variable with quantitative values (measured values) variables and 1 for fixed values (allowable values or codes) variables.

`glimpse()` function from *tibble* package is used to view the variable types in the CollectedData data frame

```
glimpse(CollectedData)
```

This indicated that there are four quantitative values (measured values) variables and four fixed values (allowable values or codes) variables.

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

Building the linker

`build_linker()` function from the *dataMeta* package is used to construct an intermediary (linker) data frame between the CollectedData and the data dictionary.

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

Data dictionary

the `build_dict()` function from the *dataMeta* package is used to construct a data dictionary for a CollectedData data frame with the aid of the linker data frame.

```
dictionary <- build_dict(my.data = CollectedData, linker = linker, prompt_varopts = FALSE)
```

Descriptions are added for variables that entered as abreviations

```
dictionary[6,4]<-"C82010: Prescribing Cost Centre - OAKHAM MEDICAL PRACTICE"
dictionary[7,4]<-"RDZ: NHS Trust - THE ROYAL BOURNEMOUTH AND CHRISTCHURCH HOSPITALS NHS FOUNDATION TRUST
dictionary[8,4]<-"RVR: NHS Trust - EPSOM AND ST HELIER UNIVERSITY HOSPITALS NHS TRUST"
dictionary[9,4]<-"RQM: NHS Trust - CHELSEA AND WESTMINSTER HOSPITAL NHS FOUNDATION TRUST"
dictionary[10,4]<-"R1F: NHS Trust - ISLE OF WIGHT NHS TRUST"
dictionary[11,4]<-"RE9: NHS Trust - SOUTH TYNESIDE NHS FOUNDATION TRUST"
dictionary[12,4]<-"RNL: NHS Trust - NORTH CUMBRIA UNIVERSITY HOSPITALS NHS TRUST"
dictionary[13,4]<-"RJ1: NHS Trust - GUY'S AND ST THOMAS' NHS FOUNDATION TRUST"
dictionary[14,4]<-"RKB: NHS Trust - UNIVERSITY HOSPITALS COVENTRY AND WARWICKSHIRE NHS TRUST"
dictionary[15,4]<-"NLO12: Independent Sector H/c Provider Site - OAKHAM URGENT CARE CENTRE"
dictionary[26,4] <-"other: Other types of A&E/minor injury activity with designated accommodation for t
dictionary[27,4]<- "1: Emergency departments are a consultant-led 24-hour service with full resuscitati
dictionary[28,4] <- "2: Consultant-led mono speciality accident and emergency service (e.g. ophthalmolo
names(dictionary)[4] <- "notes"
```

Save the data dictionary for CollectedData to the 'RawData' folder

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

Appending the data dictionary to data

Incorporate attributes as metadata to the CollectedData as metadata using the `incorporate_attr()` function from the *dataMeta* package. This requires the CollectedData, dictionary, and main_string as inputs (main_string is a character string describing the CollectedData data frame)

Create main_string for attributes

```
main_string <- "This data describes the NHS England accident and emergency (A&E) attendances and breache
```

Incorporate attributes as metadata

```
complete_CollectedData <- incorporate_attr(my.data = CollectedData, data.dictionary = dictionary,
main_string = main_string)
attributes(complete_CollectedData)$author[1]<-"B203349"
complete_CollectedData
attributes(complete_CollectedData)
```

Save the CollectedData with attributes

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

Data Capture Tool (Python)

A data capture tool was created. The code was written in Python using the Jupyter notebook. Data was collected using an interactive graphic user interface that was built using widgets from the `ipywidgets` package. During the active collection process data was saved to the Data folder as *CollectedData*. Once all data was collected the final dataset wa saved to the RawData folder as *CollectedDataFinal*.

B203349_data_collection_tool

June 20, 2022

0.1 Loading of initial data

0.1.1 Load requisite python packages

[ ]:
```
#Load the 'pandas' package
import pandas as pd
#Load the 'ipywidgets' package
import ipywidgets as widgets
#Load the 'IPython.display' package
from IPython.display import display
#Load the 'numpy' package
import numpy as np
```

0.1.2 Load test data

[ ]:
```
testData=pd.read_csv("../Data/ae_attendances_test.csv")
testData
```

[ ]:
```
#confirm data types
result = testData.dtypes
print("Output:")
print(result)
```

0.1.3 Set up empty data frame

[ ]:
```
dfTofill = pd.DataFrame({'index': [0],# Integer
                    'period': [pd.Timestamp('20000101')], # Date
                    'org_code': ['NA'], # String
                    'type': ['NA'], # String
                    'attendances': [0], # Integer
                    'breaches': [0], # Integer
                    'admissions': [0], # Integer
                    'consent': [False]}) # Boolean

dfTofill
```

Save the empty data frame to your working 'Data' folder (commented out out to prevent wiping file when re-running code)

[ ]:
```
#dfTofill.to_csv('../Data/CollectedData.csv', index=False)
```

Read empty 'CollectedData' dataframe

[ ]:
```
CollectData=pd.read_csv("../Data/CollectedData.csv")
CollectData
```

1 Indexing

Add the index number to the *'dfTofill'* file

[ ]:
```
index_number=11767 #Remember to change for each record.
dfTofill.iloc[0,0]=index_number
dfTofill
```

2 Widgets

2.1 Inserting consent

2.1.1 Checkbox widget

To capture the value for consent which is Boolean (i.e. True or False)

[ ]:
```
a = widgets.Checkbox(
    value=False,
    description='I consent for the data I have provided to be processed and␣
↪shared in accordance with data protection regulations with the purpose of␣
↪improving care service provision across the UK.',
    disabled=False,
    layout=widgets.Layout(width='1300px')) #layout ammended to allow display of␣
↪full text
display(a)
```

Add result from checkbox to 'dataTofill' dataframe

[ ]:
```
dfTofill.iloc[0,7]=a.value
dfTofill
```

3 Inserting the date

3.0.1 DatePicker widget

To input the period which is an object (string)

[ ]:
```
testData.head(n=1)
```

[ ]:
```
b = widgets.DatePicker(
    description='Period',
    disabled=False
)
display(b)
```

[ ]:
```
dfTofill.iloc[0,1]=b.value
dfTofill
```

3.1 Inserting *org_code* and *type*

Compute descriptive statistics for testData to identify how many unique options there will be for the variables *org_code* and *type*

[ ]:
```
testData.describe(include='all')
```

Obtain the unique ODS code for the organisations in the testData

3.1.1 Selection widget

Selection widget for the 'org_code' was chosen as there are 11 options

Apply the `unique()` function to create the object 'org_code' which contains the options for the 'org_code' variable

[ ]:
```
org_code=list(testData['org_code'].unique())
org_code
```

Apply the **Selection** widget

[ ]:
```
c=widgets.Select(
    options=org_code,
    value='C82010',
    rows=len(org_code),
    description='ODS code:',
    disabled=False
)
display(c)
```

[ ]:
```
dfTofill.iloc[0,2]=c.value
dfTofill
```

3.1.2 Radio button widget

Radio button for the 'Type Variable' as this is one of only three options

Apply the `unique()` function to create the object 'type' which contains the options for the 'Type Variable'

[ ]:
```
type=list(testData['type'].unique())
type
```

Apply the **Radio button** widget

[ ]:
```
d=widgets.RadioButtons(
    options=type,
    value='other',
    description='Type:',
    disabled=False
)
display(d)
```

[ ]:
```
dfTofill.iloc[0,3]=d.value
dfTofill
```

3.2 Inserting Attendances, Breaches, and Admissions

These are all whole numbers, therefore the **IntText** widget will be used

3.2.1 IntText widget

3.2.2 Attendances

Insert value for ED attendances

[ ]:
```
e=widgets.IntText(
    value=0,
    description='Attendances:',
    disabled=False)
display(e)
```

[ ]:
```
dfTofill.iloc[0,4]=e.value
dfTofill
```

3.2.3 Breaches

Insert value for 4hr breaches for the defined period

[ ]:
```
f=widgets.IntText(
    value=0,
    description='Breaches:',
    disabled=False)
display(f)
```

[ ]:
```
dfTofill.iloc[0,5]=f.value
dfTofill
```

3.2.4 Admissions

Insert value for admissions from the ED

[ ]:
```
g=widgets.IntText(
    value=0,
    description='Admissions:',
    disabled=False)
display(g)
```

[ ]:
```
dfTofill.iloc[0,6]=g.value
dfTofill
```

4 Saving the collected data

4.1 Concatenating the collected data to the CollectData data frame.

`concat()` function is used to append the CollectData and dfTofill data frames.

[ ]:
```
CollectData  = pd.concat([CollectData, dfTofill])
display(CollectData)
```

4.2 Confirming that consent is obtained

[ ]:
```
CollectData=CollectData[CollectData['consent'] == True]
display(CollectData)
```

4.3 Saving the CollectData data frame

Saving the data collected by your data-capture tool to the working data folder:

[ ]:
```
CollectData.to_csv('../Data/CollectedData.csv', index=False)
```

Once all captured data collected, then save captured test data to your 'RawData' folder.

[ ]:
```
CollectData.to_csv('../RawData/CollectedDataFinal.csv', index=False)
```

[ ]:
```

```