Title: ‘**Combined R Markdown document**’

Author: “B210741”

Date: “20 June, 2022”

1. Link to Github repository:

https://github.com/B210741/B210741_assessment.git (https://github.com/B210741/B210741_assessment.git)

2. Constructing a data dictionary and appending it to the data

Loading NHSRdatasets

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

#Load the ae_attendances data.
data(ae_attendances)
```

Exploring the ae_attendances data

The dataset set I have chosen to manage from the NHSRdatasets package is the NHS England accident and emergency (A&E) attendances and admissions (ae_attendances) data. The ae_attendances data includes reported attendances, four-hour breaches and admissions for all A&E departments in England for 2016/17 through 2018/19 (Apr-Mar). We previously selected a subset of the variables needed for my data capture tool, including period, attendances and breaches, and subsetted the data into test and training data.

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

```
glimpse(ae)
```

```
head(ae)
```

```r
#There are no missing data. The data is complete.
ae %>%
  map(is.na) %>%
map(sum)
```

```r
#Add an index link column to ae_attendances data
ae <- rowid_to_column(ae, "index")

#Tabulate the raw data for the report

ae %>%
  # Set the period column to show in month-year format
  mutate_at(vars(period), format, "%b-%y") %>%
  # Set the numeric columns to have a comma at the 1000's place
  mutate_at(vars(attendances, breaches, admissions), comma) %>%
  # Show the first 10 rows
  head(10) %>%
  # Format as a table
  kable()
```

```r
#Select the ae_attendances data subset for further exploratory analysis

ae %>%
  # set the period column to show in Month-Year format
  mutate_at(vars(period), format, "%b-%y") %>%
  # set the numeric columns to have a comma at the 1000's place
  mutate_at(vars(attendances, breaches), comma) %>%
  # show the first 10 rows
  head(10) %>%
  # format as a table
  kable()
```

```r
#save provisional subsetted ae_attendances data to the ‘RawData’ folder

write_csv(ae, here("RawData", "ae_attendances_ENG_4hr_perfom.csv"))
```

Separating provisional ae_attendances_ENG_4hr_perfom data into training and testing sets

Splint the raw data into test and training data sets

```r
#The ae_attendances_ENG_4hr_perfom dataset is large
nrow(ae) #rows of data
```

```
#[1] 12765

prop<-(1-(15/nrow(ae)))
#The proportion of the raw that needs to be assigned to the training data to ensure there is
 only 10 to 15 records in the test data is:

print(prop)
```

```
#[1] 0.9988249

set.seed(333)

#Partitioning the raw data into the test and training data.
trainIndex <- createDataPartition(ae$index, p = prop,
                                  list = FALSE,
                                  times = 1)
head(trainIndex)
```

```
# All records that are in the trainIndex are assigned to the training data.

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

```
#[1] 12753 records
```

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

glimpse(CollectedData)
```

Let's tabulate ae_attendances_ENG_4hr_perfom training data for your report

```
aeTrain %>%
  # set the period column to show in Month-Year format
  mutate_at(vars(period), format, "%b-%y") %>%
  # set the numeric columns to have a comma at the 1000's place
  mutate_at(vars(attendances, breaches), comma) %>%
  # show the first 10 rows
  head(10) %>%
  # format as a table
  kable()
```

```
write_csv(aeTrain, here("Data", "ae_attendances_ENG_4hr_perfom_train.csv"))
```

Let's extract the ae_attendances_ENG_4hr_perfom test data

```
#All records that are not in the trainIndex (`-trainIndex`) are assigned to the test data.
aeTest  <- ae[-trainIndex,]
nrow(aeTest)
```

```
#[1] 12
#There are 12 records in your test data
```

Set aside the first record from the ae_attendances_ENG_4hr_perfom test data

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

Tabulate ae_attendances_ENG_4hr_perfom marker test data for the report

```
aeTestMarker  %>%
  # set the period column to show in Month-Year format
  mutate_at(vars(period), format, "%b-%y") %>%
  # set the numeric columns to have a comma at the 1000's place
  mutate_at(vars(attendances, breaches), comma) %>%
  # show the first 10 rows
  head(10) %>%
  # format as a table
  kable()
```

Save the ae_attendances_ENG_4hr_perfom marker test data to the working data folder 'Data'

```
write_csv(aeTestMarker, here("Data", "ae_attendances_ENG_4hr_perfom_test_marker.csv"))
```

Setting aside the remaining records for testing data-capture tool.

```
aeTest  <- aeTest[2:nrow(aeTest),]
```

Tabulate ae_attendances_ENG_4hr_perfom test data for the report

```
aeTest  %>%
  # set the period column to show in Month-Year format
  mutate_at(vars(period), format, "%b-%y") %>%
  # set the numeric columns to have a comma at the 1000's place
  mutate_at(vars(attendances, breaches), comma) %>%
  # show the first 10 rows
  head(10) %>%
  # format as a table
  kable()
```

Save the ae_attendances_ENG_4hr_perfom test data to the working data folder 'Data'

```
write_csv(aeTest, here("Data", "ae_attendances_test.csv"))
```

The CollectedData dataset contains:

index: the index column that allows us to link the data collected to the original ae_attendances data in the 'RawData' folder.

period: the month that this activity relates to, stored as a date (1st of each month).

org_code: the Organisation data service (ODS) code for the organisation.

type: the Department Type for this activity, either

*1: Emergency departments are a consultant-led 24-hour service with full resuscitation facilities and designated accommodation for the reception of accident and emergency patients,

*2: Consultant-led mono speciality accident and emergency service (e.g. ophthalmology, dental) with designated accommodation for the reception of patients, or

*other: Other types of A&E/minor injury activity with designated accommodation for the reception of accident and emergency patients.

attendances: the number of attendances for this department type at this organisation for this month.

breaches: the number of attendances that breached the four-hour target.

admissions: the number of attendances that resulted in an admission to the hospital.

performance: the performance ([1 - breaches]/attendances) calculated for the whole of England.

consent: the consent from the end-user to process and share the data collected with the data capture tool.

3. Constructing a data dictionary and appending it to the data

Build a linker data frame

Variable descriptions

Create a string vector representing the different variable descriptions

```
variable_description <- c("The index column that allows us to link the data collected to the
 original ae_attendances data in the 'RawData' folder.",
"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 organ
isation associated with a particular ODS code, you can look it up from the following address:
https://odsportal.digital.nhs.uk/Organisation/Search.",
"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 performance ([1 - breaches]/attendances) calculated for the whole of England.",
"The consent from the end-user to process and share the data collected with the data capture
 tool.")

print(variable_description)
```

Variable types

```
glimpse(CollectedData)
```

We have five quantitative values (measured values) variables and four fixed values (allowable values or codes) variables.

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

```
## [1] 0 1 1 1 0 0 0 0 1
```

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

print(linker)
```

Data dictionary

Using the build_dict() function from the dataMeta to constructs a data dictionary for a CollectedData data frame with the aid of the linker data frame between.

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

```
dictionary[6,4]<-"C82010: Prescribing Cost Centre - OAKHAM MEDICAL PRACTICE."
dictionary[7,4]<-"RDZ: NHS Trust - The Royal Bournemouth and Christchurch Hospitals NHS Found
ation 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 TRUS
T."
dictionary[15,4]<-"NLO12 - Independent Sector H/c Provider Site - OAKHAM URGENT CARE CENTRE."
dictionary[27,4] <-"other: Other types of A&E/minor injury activity with designated accommoda
tion for the reception of accident and emergency patients."
dictionary[28,4] <-"1: Emergency departments are a consultant-led 24-hour service with full r
esuscitation facilities and designated accommodation for the reception of accident and emerge
ncy patients."
dictionary[29,4] <-"2: Consultant-led mono speciality accident and emergency service (e.g. op
hthalmology, dental) with designated accommodation for the reception of patients."
```

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

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

Incorporating attributes as metadata to the CollectedData as metadata using the 'incorporate_attr()' function from the dataMeta package.

```
main_string <- "This data describes the NHS England accident and emergency (A&E) attendances
 and breaches of four-hour wait time target data from the *NHSRdatasets* package collected by
the data capture tool."

main_string
```

Incorporate attributes as metadata

Using the 'incorporate_attr()' function to return an R dataset containing metadata stored in its attributes. The attributes we are going to add include: * 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 = diction
ary,
main_string = main_string)

#Change the author name
attributes(complete_CollectedData)$author[1]<-"B210741"
complete_CollectedData
```

attributes(complete_CollectedData)

Save the CollectedData with attributes

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

complete_CollectedData<-readRDS(here("RawData", "complete_CollectedData.rds"))
```

4. Colleting data using interactive Jupyter widgets

Description of the code:

The data capture tool is created using Python coding language and interactive Jupyter widgets. Using the pandas package to import the data and setting an empty data frame to collect the data captured by the Jupyter widgets. Before any data is collected and saved, we need to ensure there is consent.

Description of the interactive widgets:

Boolean widgets for consent; DatePicker widget for Period variable; Selection widgets for Org_code variable; Radio buttons for Type variable; and Numeric widgets (IntText) for attendances and breaches variables.

Storage and archive:

The data is stored in the RawData folder to follow best practice for archiving and storing data.

****

ModifiedCollectingDataUsingInteractiveJupyterWidgets

June 20, 2022

1 Title: Collecting data using interactive Jupyter widgets

Author details: *Author:* B210741. *Contact details:* s2271734@edu.ac.uk. **Notebook and data info:** This Notebook provides an example of using interactive jupyter-widgets and to collect the NHS England accident and emergency attendances and admissions (ae_attendances) data. **Data:** Data consists of date, numerical data and character data from NHSRdatasets package. **Copyright statement:** This Notebook is the product of The University of Edinburgh.

1.1 Data

I have chosen the NHS England accident and emergency (A&E) attendances and admissions (`ae_attendances`) data from the NHSRdatasets dataset, this includes reported attendances, four-hour breaches and admissions for all A&E departments in England for 2016/17 through 2018/19 (Apr-Mar).

I selected a subset of the variables needed for my data capture tool, including period, attendances and breaches, and subsetted the data into test and training data. I used interactive Jupyter-widgets from the *ipywidgets* package to collect all data types from the `ae_attendances` data. The R script "./RScripts/LoadingNHSRdatasets_fulldata.R" was used to subset the full `ae_attendances` data into test and training data.

1.1.1 Loading the data

[ ]:
```
#Load the 'pandas' package
import pandas as pd
testData=pd.read_csv("../Data/ae_attendances_ENG_4hr_perfom_test_full.csv")
testData
```

Data type Using the `dtypes` function from the Python *pandas* package to query the data types in the testData.

[ ]:
```
result = testData.dtypes
print("Output:")
print(result)
#The data type object is a string
```

Setting up an empty data frame Setting up an empty data frame in the working data folder to collect the data captured by the Juypter widgets.

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

Save the empty data frame to your working 'Data' folder:

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

Reading in the empty data frame to collect the data from the Jupyter-widgets.

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

1.1.2 Index variable

I will use indexing to to add the index number to the 'dfTofill' file. The first variable contains the index number, indexing allows us to connect the test data to the orginal data set "../Raw-Data/ae_attendances.csv".

```
[ ]: index_number=1155 #Remember to change for each record.
     dfTofill.iloc[0,0]=index_number
     dfTofill
```

1.1.3 Setting up Widgets

Widgets are interactive Python objects that have a representation in the browser. A widget is a graphical user interface element, such as a button, dropdown or textbox.

```
[ ]: #Load the 'ipywidgets' package
     import ipywidgets as widgets
```

display() The *IPython.display* package is used to display different objects in Jupyter.

```
[ ]: #Load the 'IPython.display' package
     from IPython.display import display
```

2 Consent

Before we collect any data, we need to get consent from the end-user to process and share the data we will collect with the data capture tool. If the data subject has no real choice, consent is not freely given, and it will be invalid. The General Data Protection Regulation sets a high standard for consent and contains significantly more detail than previous data protection legislation.

2.1 Boolean widgets

Boolean widgets are designed to display a boolean value. ### Checkbox widget

```
[ ]: 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)
     display(a)
```

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

3 The period variable

The period variable includes the month this activity relates to, stored as a date (1st of each month).

Data type

```
[ ]: print(result[1])
     #String data type
```

3.0.1 DatePicker widget

We next need to set up a DatePicker widget to collect the period data.

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

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

4 The org_code variable

The org_code variable includes the Organisation data service (ODS) code for the organisation. The ODS code is a unique code created by the Organisation data service within [NHS Digital] To know the organisation associated with a particular ODS code, you can look it up from the following address: https://odsportal.digital.nhs.uk/Organisation/Search.

Data type

```
[ ]: print(result[2])
     #String data type
```

Using the describe() function from the numpy Python package to calculate summary statistics for the testData data frame.

```
[ ]: #Load the 'numpy' package
     import numpy as np
     testData.describe(include='all')
```

Using the pandas package unique() function to get the unique Organisation data service (ODS) codes in the test data.

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

4.1 Selection widgets

Several widgets can be used to display single selection lists. You can specify the selectable options by passing a list.

```
[ ]: 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
```

5 The type variable

The type variable contains the department type for this activity, either
* **1:** Emergency departments are a consultant-led 24-hour service * **2:** Consultant-led mono speciality accident and emergency service with designated accommodation for the reception of patients, or
* **other:** Other type of A&E/minor injury activity with designated accommodation for the reception of accident and emergency patients.

Data type

```
[ ]: print(result[3])
     #String data type
```

Applying pandas unique() function Using the pandas package unique() function to get the unique department type in the test data.

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

5.0.1 RadioButtons

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

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

6 The attendances variable

The attendances variable includes the number of attendances for this department type at this organisation for this month.

Data type

```
[ ]: print(result[4])
     #int64
```

6.1 Numeric widgets

There are many widgets distributed with ipywidgets that are designed to display numeric values.

6.1.1 IntText

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

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

7 The breaches variable

The breaches variable includes the number of attendances that breached the four hour target.

Data type

```
[ ]: print(result[5])
     #int64
```

7.0.1 IntText

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

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

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

7.0.2 Ensuring we have consent to save the data

Before we save our data to file, we must make sure we have consent to do so.

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

7.0.3 Filling the Data frame

For each subsequent row of the test data, I use the selected widgets to overwrite the dfTofill data frame with the next row of data from the test data, ensuring that there is consent before saving.

```
[ ]: #Second row
     index_number=2059
     display(a)
     display(b)
     display(c)
     display(d)
     display(e)
     display(f)
     #Select the data for each widget to capture referencing testData.
```

```
[ ]: dfTofill.iloc[0,0]=index_number
     dfTofill.iloc[0,6]=a.value
     dfTofill.iloc[0,1]=b.value
     dfTofill.iloc[0,2]=c.value
     dfTofill.iloc[0,3]=d.value
     dfTofill.iloc[0,4]=e.value
     dfTofill.iloc[0,5]=f.value
     dfTofill
     #Save to dfTofill dataframe.
```

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

```
#Ensuring we have consent to save the data
CollectData=CollectData[CollectData['consent'] == True]
display(CollectData)
```

8 Concatenating the collected data to the CollectData data frame.

Using the `concat()` function from the Python *pandas* package to append the CollectData and dfTofill data frames.

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

8.1 Ensuring we have consent to save the data

Before we save our data to file, we must make sure we have consent to do so.

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

8.1.1 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)
```

Collected all of your test data and then save the captured test data to your 'RawData' folder. This is to ensure secure storage and archiving purposes.

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