Working with data types and structures in Python and R: My First R Markdown Document

B208593

19 June, 2022

[Link to the GitHub repositary] (https://github.com/B208593/B208593_assessment.git)

I. Loading NHSR datasets

The data I used for this assessment are from the NHSRdatasets package: the NHS England accident and emergency attendances and admissions (ae_attendances) data set.

1. Load packages

First we need to load the packages we will use in the script.

```
library(NHSRdatasets)
```

```
library(tidyverse)
```

```
library(here)
```

```
library(knitr)
```

```
library(scales)
```

```
library(lubridate)
```

```
library(caret)
```

2. Load the NHS England accident and emergency attendances and admissions (ae_attendances) dataset

```
data(ae_attendances)
```

3. Viewing the ae_attendances data

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

Viewing the columns/variables in a data frame we use the *glimpse()* function

```
glimpse(ae)
```

Missing data

We need to check for missing data in the ae_attendances data as missing data can significantly impact any attempt to gain useful insight from data.

```
# Calculate how many NAs there are in each variable
ae %>%
  map(is.na) %>%
map(sum)
```

Adding index column to raw data

To develop and evaluate the data capture tool, the raw data is needed to be split into training and testing sets. We need to add an index column to the raw data so we can link the partitioned data sets to the raw data.

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

Tablulate the raw data for the report

To assists the reader in quickly determining the magnitude of the numbers they are looking at.

```
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()
```

Save the raw ae_attendances data to your 'RawData' folder

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

Creating a new variable

Calculate monthly four hour waiting time target performance

```
ae<- ae %>%
mutate(performance = round(1- breaches / attendances,2))
```

Tabulate the updated ae_attendances data

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

4. Selecting variables for data capture tool

The variables I have selected for the data capture tool: Index, period, org_ code, attendance, breaches and performance

Filter type 1 A&E departments and select variables for the data capture tool

```
ae2<-ae %>%
  filter(type=="1") %>%
  select(-type, -admissions)
```

Tabulate the raw data

```
ae2 %>%
  mutate_at(vars(period), format, "%b-%y") %>%
  mutate_at(vars(attendances, breaches), comma) %>%
  head(10) %>%
  kable()
```

Save csv file to Raw data folder

```
write_csv(ae2, here("RawData", "ae_type1_performance.full.csv"))
```

5. Partitioning the raw data into test and training data

We have to work out the proportion (`prop`) of the raw data to assign to the training data:

```
prop<-(1-(15/nrow(ae2)))
#The proportion of the raw that needs to be assigned to the training data to ensure there is only 10 to
print(prop)
```

Random number generator

The 'set.seed()' function is a random number generator which will make sure that every time we run this script, we will partition the raw data into the same test and training data.

```
set.seed(333)
```

Split dataset

We will use the createDataPartition() function from the caret package to split our raw data into test and training data sets.

```
trainIndex <- createDataPartition(ae2$index, p = prop,
                                  list = FALSE,
                                  times = 1)
```

We need the records that are in the trainIndex are assigned to the training data.

```
ae2Train <- ae2[ trainIndex,]
```

Tabulate ae_type1_performance training data for the report

```
ae2Train %>%
  mutate_at(vars(period), format, "%b-%y") %>%
  mutate_at(vars(attendances, breaches), comma) %>%
  head(10) %>%
  kable()
```

Save ae_type1_performance training data to the working data folder 'Data'

```
write_csv(ae2Train, here("Data", "ae_type1_performance_train_full.csv"))
```

Extract the ae_type1_performance test data that records not in the trainIndex (-trainIndex) are assigned to the test data.

```
ae2Test  <- ae2[-trainIndex,]
```

Marker test data

We need to set aside the first record from the ae_type1_performance test data so that your markers can test and evaluate your data-capture tool.

```
ae2TestMarker  <- ae2Test[1,]
```

Save ae_type1_performance marker test data to the Data folder

```
write_csv(ae2TestMarker, here("Data", "ae_type1_performance_test_marker_full.csv"))
```

Set aside the remaining records to test the data-capture tool.

```
ae2Test  <- ae2Test[2:nrow(ae2Test),]
```

Tabulate test data for the report

```
ae2Test  %>%
  mutate_at(vars(period), format, "%b-%y") %>%
  mutate_at(vars(attendances, breaches), comma) %>%
  head(10) %>%
  kable()
```

Finally save ae_type1_performance test data to the working data folder 'Data'

```
write_csv(ae2Test, here("Data", "ae_type1_performance_test_full.csv"))
```

II. Data capture tool in Python Jupyter notebook

1. Data upload

We need to upload the previously selected subset of the variables (ae_type1_performance_test_full) for the data capture tool

Load the pandas package

To import the data, we will need to load the pandas package which is used for data manipulation and analysis.

Data type

We now need to check the data type in the test Data data frame in order to know what type of widget we need to use.

2. Set up empty data frame

We need to set up an empty data frame in the working data folder to collect the data captured by the Juypter widgets containing only the variables we want to capture

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

Only use once otherwise it will overwrite our data we already input

Next we have to read in the empty data frame to collect the data from the Jupyter-widgets.

3. Indexing in Python

The first variable contains the index number, that allows us to connect the test data to the original data set "../RawData/ae_type1_performance_test_full". Indexing in Python is a way to refer the individual items by its position.

4. Widgets for the variables

We need to select widgets (eg. button, dropdown or textbox) to collect our test data in Python. To use the widget framework, we need to import the ipywidgets Python package.In order to select the widgets we need to check the data type of the variables.

Consent variable

Before we collect the data , we need to get consent from the end-user to process and share the data we will collect with the data capture tool. The consent data type has 2 options, TRUE or FALSE (logical data type) so boolean widget (Checkbox widget) will be used that are designed to display a boolean value.

Period variable

Data type is string or character. DatePicker widget was selected to capture/collect period variable.

Org_code variable

Data type is also a string. Selection widget was selected to capture this variable to display single selection list.

Attendances, breaches variables

Data type for these variables is numeric, integer. Numeric widget was selected (IntText) for displaying integers.

Performance variable

Data type is numeric, float. Numeric widget (FloatText) was selected.

5. Concatenating the collected data to the CollectData data frame.

Every row needs to be filled in the empty data frame by using concat() function to concatenate pandas objects (11 rows of data).

Consent check

Before we save our data to file, we must make sure we have consent to do so (we have consent filled in as TRUE).

Saving the CollectData data frame

We need to save the data collected by the data-capture tool to the Data and Raw folders.

III. Data dictionary

1. Load packages

```
library(dataMeta)
library (tidyverse)
library(here)
```

2. Data used

Test data was collected by the Jupyter Notebook "./ipynbScripts/CollectingDataUsingInteractiveJupyterWidgets.ipynb".

Read the collected data from the Raw data folder.

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

Check type of the period variable

```
class(CollectedData$period)
```

It's identified as *date* but we need the period to be *character* for the Data dictionary.

Change period variable from date to character

```
CollectedData$period<-as.character(CollectedData$period)
```

Check 'type' of period variable

```
class(CollectedData$period)
```

Now it's a character variable.

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

Build a linker data frame

We first need to build a linker data frame. To do this, we need to create two string vectors representing the different variable descriptions and the different variable types.

Variable descriptions We need to 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 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.",
"The number of attendances for type 1 A&E department at this organisation for this month.",
"The number of attendances that breached the four-hour target.",
"The performance ([1 - breaches]/attendances) calculated for type 1 A&E departments.",
"The consent from the end-user to process and share the data collected with the data capture tool.")
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. We need to use 0 for a variable with quantitative values (measured values) variables and 1 for fixed values (allowable values or codes) variables.

To view the variable types in the CollectedData data frame.

```
glimpse(CollectedData)
```

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

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

4. Linker

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

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

5. Data dictionary

We will use 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)
glimpse(dictionary)
```

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

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

6. Append data dictionary to the CollectedData

We will now incorporate 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 type 1 accident and emergency (A&E) attendances and
main_string
```

Create main_string for attributes

Incorporate attributes as metada We are using the 'incorporate_attr()' function to return an R dataset containing metadata stored in its attributes.

```
complete_CollectedData <- incorporate_attr(my.data = CollectedData, data.dictionary = dictionary,
main_string = main_string)
#Change the author name
attributes(complete_CollectedData)$author[1]<-"B208593"
complete_CollectedData
attributes(complete_CollectedData)
```

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

Save the CollectedData with attributes