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

B208593

18 June, 2022

To set global options that apply to every chunk in the file

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

1. Loading NHSRdatasets

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.

Load packages and data

```
library(NHSRdatasets)
library(tidyverse)
```

```
library(here)
```

```
library(knitr)
```

```
library(scales)
```

```
library(lubridate)
```

```
library(caret)
```

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

```
data(ae_attendances)
```

Take a look at the ae_attendances data

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

An overview of your data

```
ae
```

The original dataset contains

- **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. The ODS code is a unique code created by the Organisation data service within NHS Digital, and used to identify organisations across health and social care. ODS codes are required in order to gain access to national systems like NHSmail and the Data Security and Protection Toolkit.
- **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 specialty accident and emergency service (e.g. ophthalmology, dental) 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. The department may be doctor led or nurse led and treats at least minor injuries and illnesses and can be routinely accessed without appointment. A service mainly or entirely appointment based (for example a GP Practice or Out-patient clinic) is excluded even though it may treat a number of patients with minor illness or injury. Excludes NHS walk-in centres.(National Health Service, 2020)
- **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.(Chris Mainey, 2021)

The `glimpse()` function is from *tibble* package and is great to view the columns/variables in a data frame.

```
glimpse(ae)
```

Missing data

Missing data can significantly impact any attempt to gain useful insight from data. Therefore need to check for missing data in the ae_attendances 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")
```

Let's 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()
```

Let's save the raw ae_attendances data to your 'RawData' folder

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

Calculate monthly four hour waiting time target performance for England as a whole

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

Let's tabulate the ae_attendances data

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

Selecting variables for 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

```
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"))
```

Check number of rows in subset

```
nrow(ae2)
```

A test data set of 10-15 records to capture with and evaluate your data capture tool is sufficient for the purpose of the course assignment. Here is how 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 15 records in the test data is:

```
print(prop)
```

Use the createDataPartition() function from the caret package to splint our raw data into test and training data sets. The 'set.seed()' function is a random number generator, which is useful for creating random objects that can be reproduced. This 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)
```

Partitioning the raw data into the test and training data.

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

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

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

Check rows

```
nrow(ae2Train)
```

Let's tabulate ae_attendances_ENG_4hr_perfom training data for your report

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

Our next task, it to save ae_attendances_ENG_4hr_perfom training data to your working data folder 'Data'

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

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

Check number or rows

```
nrow(ae2Test)
```

#You now need to set aside the first record from the ae_attendances_ENG_4hr_perfom test data so that your markers can test and evaluate your data-capture tool.

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

```
ae2TestMarker  %>%
  mutate_at(vars(period), format, "%b-%y") %>%
  head(10) %>%
  kable()
```

Let's tabulate marker test data for your report

Our next task, it to save our ae_attendances_ENG_4hr_perfom marker test data to our working data folder 'Data'

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

We then need to set aside the remaining records for you to test (or collect with your) your data-capture tool.

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

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

Let's tabulate test data for your report

Our final task, is to save our ae_attendances_ENG_4hr_perfom test data to our working data folder 'Data'

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

2. Data capture tool in Python Jupyter notebook

Data upload

The previously selected subset of the variables for the data capture tool in Raw data folder

Load the pandas package

To import the data, you will need to load the pandas package. The Python pandas package is used for data manipulation and analysis.

Data type

We now need to check the data type in the testData data frame in order to know what type of widget we need to use. Let us use the dtypes function from the Python pandas package to query the data types in the testData.

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 containgin 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

Now let's read in the empty data frame to collect the data from the Jupyter-widgets.

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_attendancesxxx.csv". Indexing in Python is a way to refer the individual items by its position. In other words, you can directly access your elements of choice.

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. Widgets can be embedded in the Notebook and provide a user-friendly interface to collect the user input and see the impact the changes have on the data/results without interacting with your code.

To use the widget framework, you need to import the ipywidgets Python package.

Consent

Consent is a vital area for data protection compliance. Consent means giving data subjects genuine choice and control over how you process their data. If the data subject has no real choice, consent is The consent data type has 2 options, TRUE or FALSE For consent boolean widget (Checkbox widget) was used that are designed to display a boolean value: TRUE or FALSE.

Period

The data type object is a string (represent text rather than numbers) We used DatePicker widget to capture/collect date variable

Org_code

The org_code variable includes the Organisation data service (ODS) code for the organisation. We now need to check the data type: The data type object is also a string.

Selection widget used to capture this variable to display single selection list

Attendances, breaches

The attendances variable includes the number of attendances for this department type at this organisation for this month. The breaches variable includes the number of attendances that breached the four hour target. check the data type - integer, float

Numeric widget (IntText) for displaying integers and floats

performance

The performance variable was calculated for the whole of England as (1 - breaches)/ attendances. It is a float variable. Numeric widget - FloatText

Concatenating the collected data to the CollectData data frame.

The concat() function is used to concatenate pandas objects.

Have you consent to process and share the data before you save it to the working data folder?

Before we save our data to file, we must make sure we have consent to do so. The following line of code, will ensure that you have consent to save data.

Saving the CollectData data frame

Saving the data collected by your data-capture tool to the working data folder:../Data/CollectedData.csv

You need to iterate through this Notebook until you have collected all of your test data and then save the captured test data to your 'RawData' folder.

Data dictionary

A data dictionary is an important tool in data management. A data dictionary is a supplementary document that details the information about variables, data collection, and other important features of a data set, i.e. metadata, data that describes other data.

linker

adding metadata

Metadata describes other data in order to help others to find, access, understand, and reuse the data.

Load packages and read in the data

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

Data

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. The Jupyter Notebook "./ipynbScripts/CollectingDataUsingInteractiveJupyterWidgets.ipynb" was used to to collect the data.

Let us use the `read_csv()` function from the *readr* package to read your 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 a date but we need to be character for the Data dictionary

Change from date to character

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

check again class

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

Now it's a character variable

Let's view the CollectedData ae_attendances data

```
glimpse(CollectedData)
```

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.
- **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.
- **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.

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)
```

Now let us use the `build_linker()` function from the *dataMeta* package to constructs an intermediary (linker) data frame between the CollectedData and the data dictionary.

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

Data dictionary

We are now going to 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)
```

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

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. 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 = 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

It was partitioned the data subset into training and testing data and save them to your working 'Data' folder for downstream exploratory analysis

1. Data dictionary for test data

linker

adding metadata

Metadata describes other data in order to help others to find, access, understand, and reuse the data.

2. Data capture tool in Python

Variables

Indexing

Consent

Widgets

3. Loading NHS Dataset

Introduction what the dataset is eg. Data

Upload

Viewing the data

Subsetting the data

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.