B202395_Assignment1_CollectingData

June 18, 2022

Title: Collecting data using interactive Jupyter widgets
Author: B202395, *Contact details:* B202395
Notebook and data info: This Notebook provides the code to collect the NHS England accident and emergency attendances and admissions (ae_attendances) data for University Hospitals of Leicester NHS Trust using interactive jupyter-widgets.
Data: Data consists of date (period), numerical data (attendances, breaches, performance) and character data (type) from the subset of data from the NHSRdatasets package.

1 Data

The NHS England accident and emergency attendances and admissions data for University Hospitals of Leicester NHS Trust was extracted and partitioned into test and training data in R. Python in Jupyter was used for the data collection. The interactive Jupyter-widgets from *ipywidgets* package was used to collect the relevant data types from the test data.

1.1 Preparing for data collection and creating the data frame

[ ]:
```
#The training dataset created in R is loaded into Python.
#Load the 'pandas' package
import pandas as pd
testDatarwe=pd.read_csv("../Data/aerwe_attendances_4hr_perform_test.csv")
testDatarwe
#Checking the data type so we know what is required when creating the empty␣
 ↪data frame
result = testDatarwe.dtypes
print("Output:")
print(result)
```

[ ]:
```
#An empty data frame is required in the working data folder to collect the data␣
 ↪captured by the Juypter widgets.
#Setting up an empty data frame in the working data folder for collection of␣
 ↪the data via Jupyter widgets
dfTofillrwe = pd.DataFrame({'index': [0],# Integer
                    'period': [pd.Timestamp('20000101')], # Date
                    'type': ['NA'], # String
                    'attendances': [0], # Integer
                    'breaches': [0], # Integer
```

```
                    'performance': [0.0], # Float
                    'consent': [False]}) # Boolean
dfTofillrwe
```

[ ]:
```
#An empty data frame is saved in the working 'Data' folder (this step only␣
 ↪needs to be done once):
#dfTofillrwe.to_csv('../Data/CollectedDatarwe.csv', index=False)
```

[ ]:
```
#Reading in the empty data frame to collect the data from the Jupyter-widgets
CollectDatarwe=pd.read_csv("../Data/CollectedDatarwe.csv")
CollectDatarwe
```

[ ]:
```
#To view the test data
print(testDatarwe)
```

There are 10 rows and 6 columns of test data.

1.2 Indexing

The first variable (index) in the test data is to link it back to the original dataset "../Raw-Data/ae_attendances.csv". Indexing is added to the 'dfTofill' file to enable identification of individual entries by its position for selection of specific elements of choice. Indexing in Python starts with the count of zero ("zero-indexed").

[ ]:
```
#This line of code specifies the entry for selection/entry of interest
index_number=11425 #Changed for each record.
dfTofillrwe.iloc[0,0]=index_number
dfTofillrwe
```

2 Widgets

Python widgets are interactive browser controls that can be embedded into Jupyter Notebook to enable user engagement. The *ipywidgets* package and *IPython.display* package were used.

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

#Loading the 'IPython.display' package
from IPython.display import display
```

2.1 Creating the widgets

Each of these fields (consent, period, type, attendancs, breaches and performance) will require a separate widget to enable user input of the data.

2.2 Consent

Consent will be captured using a checkbox widget to record the Boolean (True/False) data subject consent status to ensure compliance with General Data Protection Regulation (GDPR).

[ ]:
```
##Checkbox widget to capture consent values (Boolean - True/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)
display(a)
```

[ ]:
```
#Add consent as the 6th column of the data frame
dfTofillrwe.iloc[0,6]=a.value
dfTofillrwe
```

2.3 The period variable

The period variable is stored as a date (1st of each month).

[ ]:
```
#Checking the data type for 'period'
print(result[1])
```

The data type object is a string.

2.3.1 DatePicker widget

The DatePicker widget will be used to collect the period data.

[ ]:
```
#This code creates the widget to collect the period data for the index selected␣
 ↪in section 1.5
b = widgets.DatePicker(
    description='Period',
    disabled=False)
display(b)
```

[ ]:
```
#This code is to ensure that the period (date) information entered in the␣
 ↪previous step is populated in the period column
dfTofillrwe.iloc[0,1]=b.value
dfTofillrwe
```

2.4 Type variable

As noted in section 1.3, type is a data type object. The data type object is a string.

```
[ ]: #Checking the data type for 'type'
print(result[2])
#String data type
```

```
[ ]: #The `unique()` function from the *pandas* package is used to obtain the list␣
 ↪of unique department types in the test data.
type=list(testDatarwe['type'].unique())
type
#'other','2', '1'
```

The RadioButtons widget will be used to collect data for the 'type' variable.

```
[ ]: #Creating the radio button for the user to input the 'type' data for the index␣
 ↪selected in section 1.5
c=widgets.RadioButtons(
    options=type,
    description='Type:',
    disabled=False)
display(c)
```

```
[ ]: #This code is to ensure that the 'type' information entered in the previous␣
 ↪step is populated in the 'type' column
dfTofillrwe.iloc[0,2]=c.value
dfTofillrwe
```

2.5 Attendances variable

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

```
[ ]: #Checking the data type for 'attendances'
print(result[3])
```

IntText from the ipywidgets is used to enable the inputting of the numeric value of the attendances variable.

```
[ ]: #Creating the IntText widget for the user to input the 'attendances' data for␣
 ↪the index selected in section 1.5
d=widgets.IntText(
    value=0,
    description='Attendances:',
    disabled=False)
display(d)
```

```
[ ]: #This code is to ensure that the 'attendances' information from the previous␣
 ↪step is populated in the 'attendance' column
dfTofillrwe.iloc[0,3]=d.value
dfTofillrwe
```

2.6 Breaches variable

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

[ ]:
```
#Checking the data type for 'breaches'
print(result[4])
```

IntText from the ipywidgets is used to enable the inputting of the numeric value of the breaches variable.

[ ]:
```
#Creating the IntText widget for the user to input the 'breaches' data for the␣
 ↪index selected in section 1.5
e=widgets.IntText(
    value=0,
    description='Breaches:',
    disabled=False)
display(e)
```

[ ]:
```
#This code is to ensure that the 'breaches' information entered in the previous␣
 ↪step is populated in the breaches column
dfTofillrwe.iloc[0,4]=e.value
dfTofillrwe
```

2.7 Performance variable

The performance variable was calculated as (1 - breaches)/ attendances.

[ ]:
```
#Checking the data type for 'performance'
print(result[5])
```

FloatText from the ipywidgets is used to enable the inputting of the float value of the performance variable.

[ ]:
```
#Creating the IntText widget for the user to input the 'performance' data for␣
 ↪the index selected in section 1.5
f=widgets.FloatText(
    value=0.0,
    description='Performance:',
    disabled=False)
display(f)
```

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

3 Concatenating the collected data to the CollectDatarwe data frame

[ ]:
```
#The `concat()` function from the *pandas* package is used to append the␣
↪CollectDatarwe to dfTofillrwe data frames.
# CollectDatarwe is the first data frame, dfTofillrwe is the second data frame
CollectDatarwe  = pd.concat([CollectDatarwe, dfTofillrwe])
display(CollectDatarwe)
```

[ ]:
```
#The code below checks that consent has been obtained before it is saved into␣
↪the working data folder.
CollectDatarwe=CollectDatarwe[CollectDatarwe['consent'] == True]
display(CollectDatarwe)
```

3.1 Saving the CollectDatarwe data frame

[ ]:
```
#Saving the data collected by the data-capture tool to the 'Data' folder:
CollectDatarwe.to_csv('../Data/CollectedDatarwe.csv', index=False)
```

[ ]:
```
#The process is repeated until all the data has been captured from the test␣
↪data.
#Once that is completed, the data is saved in the 'RawData' folder.
CollectDatarwe.to_csv('../RawData/CollectedDatarweFinal.csv', index=False)
```