Collecting data using interactive Jupyter widgets

June 20, 2022

Author details: Author: B209223. Contact details: s2272326@ed.ac.uk. **Notebook and data info**: This Notebook uses interactive jupyter-widgets and to collect the stranded patient data from NHSRdatasets and save it to working 'Data' folder, and finally saving all the captured test data to 'RawData' folder.

1 Data

The data is subsetted from stranded patient data from NHSRdatasets, including one outcome variable (stranded vs. non-stranded) and 3 predicting variables, synthetically generated by Gary Hutson g.hutson@nhs.net, Mar-2021. Data consists of strings, integers and date.

1.1 The *pandas* package

Python *pandas* package is needed to import and manipulate data.

[2]:
```
#To load the 'pandas' package
import pandas as pd
testData=pd.read_csv("../Data/stranded_test.csv")
testData
```

[2]:

	index	stranded.label	age	mental_health_care	admit_date
0	7	Not Stranded	26	0	2021-01-02
1	135	Not Stranded	26	0	2021-01-20
2	157	Not Stranded	21	0	2021-01-02
3	269	Stranded	21	0	2021-02-05
4	299	Not Stranded	67	0	2021-01-27
5	335	Stranded	79	1	2020-12-18
6	434	Not Stranded	72	1	2021-01-21
7	496	Not Stranded	66	0	2021-02-04
8	540	Not Stranded	32	0	2021-01-21
9	663	Not Stranded	72	1	2021-01-28
10	720	Stranded	67	0	2021-01-15
11	723	Stranded	42	1	2020-12-12

1.2 Data type

Firstly we should examine the data type in the testData data frame with dtypes function.

[29]:
```
result = testData.dtypes
print("Output:")
print(result)
```

```
Output:
index                   int64
stranded.label         object
age                     int64
mental_health_care      int64
admit_date             object
dtype: object
```

Index, age and mental_health_care are integers. Stranded.label is a string. Admit_date is a date.

Next an empty data frame will be set up in the "Data" folder to collect data captured by Jupyter widgets.

[86]:
```
dfTofill = pd.DataFrame({'index': [0],# Integer
                    'stranded.label': ['NA'], # String
                    'age': [0], # Integer
                    'mental_health_care': [0], # Integer
                    'admit_date': [pd.Timestamp('20000101')], # Date
                    'consent': [False]}) # Boolean

dfTofill
```

[86]:

	index	stranded.label	age	mental_health_care	admit_date	consent
0	0	NA	0	0	2000-01-01	False

The empty dataframe will be saved to "Data" folder.

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

The empty dataframe is saved. Saving the empty dataframe should be done only once at the beginning of data capture, so a # is added.

We can start to capture data by importing the empty frame.

[96]:
```
CollectData=pd.read_csv("../Data/CollectedData.csv")
CollectData
```

[96]:

	index	stranded.label	age	mental_health_care	admit_date	consent
0	7	NaN	0	0	2000-01-01	False

1.3 The head() function

Next the first row of data will be captured.

The first row of data can be accessed with a head() function.

[97]:
```
testData.head(1)
```

[97]:

	index	stranded.label	age	mental_health_care	admit_date
0	7	Not Stranded	26	0	2021-01-02

We will work on this row as an example.

2 Index variable

The first variable index serves to link the test data with the original dataset.

2.1 Indexing in Python

Python is zero-indexed, meaning it starts in zero. We will use indexing in python to access the position where the data should be entered.

[98]:
```
index_number=7 #Remember to change for each record.
dfTofill.iloc[0,0]=index_number
dfTofill
```

[98]:

	index	stranded.label	age	mental_health_care	admit_date	consent
0	7	NA	0	0	2000-01-01	False

2.2 Widgets

Widgets are interative python elements that provides user-friendly interface to input data. Based on data types, different widgets are employed. Here we will use checkbox widget, selection widget, numeric widgets and DatePicker widgets.

[99]:
```
#Load the 'ipywidgets' package
import ipywidgets as widgets
```

2.3 display

The display() function in *IPython.display* package helps to display different objects and widgets in Jupyter.

[100]:
```
#Load the 'IPython.display' package
from IPython.display import display
```

3 Consent

Data protection guidances and laws require consent must be obtained before any processing of data. So the first step of the data capture tools is to obtain consent from users that they agree to process and share the data with us.

3.1 Boolean widgets

Boolean widgets are designed to display a boolean value.

3.1.1 Checkbox widget

[101]:
```
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)
```

```
Checkbox(value=False, description='I consent for the data I have provided to be processed and s
```

[102]:
```
dfTofill.iloc[0,5]=a.value
dfTofill
```

[102]:

	index	stranded.label	age	mental_health_care	admit_date	consent
0	7	NA	0	0	2000-01-01	True

4 The stranded.label variable

The stranded.label variable marks whether the patient is stranded or not, including *Stranded* and *Not Stranded*. It is an outcome variable for supervised machine learning model.

4.1 Data type

Let's examine the data type of the variable with dtypes function.

[103]:
```
print(result[1])
```

```
object
```

The data type object is a string.

4.2 Describe the stranded.label data

Given the outcome variable is expected to be binary, it is a good idea to check what response is allowed in this dataset. If there are only a few options, a selection widget will be employed.

[104]:
```
#Load the 'numpy' package
import numpy as np
testData["stranded.label"].describe(include='all')
```

[104]:
```
count               12
unique               2
top       Not Stranded
freq                 8
Name: stranded.label, dtype: object
```

As expected, there are only two response allowed so a selection widget is preferred. The two responses are:

[105]:
```
label=list(testData["stranded.label"].unique())
label
```

[105]:
```
['Not Stranded', 'Stranded']
```

Let's access our example row with a head() function.

[106]:
```
testData.head(n=1)
```

[106]:

	index	stranded.label	age	mental_health_care	admit_date
0	7	Not Stranded	26	0	2021-01-02

4.3 Selection widget

Selection widget is used to present a list of options. In this case, the two responses are included in a list *label* which could be passed to options.

[107]:
```
b=widgets.Select(
    options=label,
    value='Not Stranded',
    rows=len(label),
    description='Stranded or not:',
    disabled=False
)
display(b)
```

```
Select(description='Stranded or not:', options=('Not Stranded', 'Stranded'), rows=2, value='Not
```

[108]:
```
dfTofill.iloc[0,1]=b.value
dfTofill
```

[108]:

	index	stranded.label	age	mental_health_care	admit_date	consent
0	7	Not Stranded	0	0	2000-01-01	True

5 The age variable

The age variable records patient age on admission.

5.1 Data type

Let's examine the data type of the variable with dtypes function.

[63]:
```
print(result[2])
```

```
int64
```

The data type object is integer so a numeric widget will be applied.

Let's access our example row with a head() function.

[109]:
```
testData.head(n=1)
```

[109]:

	index	stranded.label	age	mental_health_care	admit_date
0	7	Not Stranded	26	0	2021-01-02

5.2 Numeric widgets

5.2.1 IntText

[110]:
```
c=widgets.IntText(
    value=26,
    description='age:',
    disabled=False)
display(c)
```

```
IntText(value=26, description='age:')
```

[111]:
```
dfTofill.iloc[0,2]=c.value
dfTofill
```

[111]:

	index	stranded.label	age	mental_health_care	admit_date	consent
0	7	Not Stranded	26	0	2000-01-01	True

6 The mental_health_care variable

The mental_health_care variable flags whether patients need mental health care or not.

6.1 Data type

Let's examine the data type of the variable with dtypes function.

[66]:
```
print(result[3])
```

```
int64
```

Although the data type is integer, given it is expected to be binary, it is a good idea to check unique response and decide whether a selection widget or a numeric widget should be applied.

6.2 Unique value of the mental_health_care data

[67]:
```
#Load the 'numpy' package
import numpy as np
testData["mental_health_care"].unique()
```

[67]:
```
array([0, 1])
```

There are only two responses allowed. 0 indicates patients do not need mental health service while 1 indicates they do. With limited options a selection widget should be used.

6.3 Selection widget

First we should generate a list of the two responses and pass it to the option parameter.

A list of options

[68]:
```
options=list(testData["mental_health_care"].unique())
options
```

[68]:
```
[0, 1]
```

Let's access our example row with a head() function.

[112]:
```
testData.head(n=1)
```

[112]:

	index	stranded.label	age	mental_health_care	admit_date
0	7	Not Stranded	26	0	2021-01-02

[116]:
```
d=widgets.Select(
    options=options,
    value=0,
    rows=len(options),
    description='Mental health care needed:',
    disabled=False
)
display(d)
```

```
Select(description='Mental health care needed:', options=(0, 1), rows=2, value=0)
```

[117]:
```
dfTofill.iloc[0,3]=d.value
dfTofill
```

[117]:

	index	stranded.label	age	mental_health_care	admit_date	consent
0	7	Not Stranded	26	0	2000-01-01	True

7 The admit_date variable

The admit_date variable records dates when patients were admitted to hospital.

7.1 Data type

Let's examine the data type of the variable with dtypes function.

[71]:
```
print(result[4])
```

```
object
```

The data type object is a string.

7.2 DatePicker widget

A DatePicker widget is placed to select a date.

Let's access our example row with a head() function.

[118]:
```
testData.head(n=1)
```

[118]:

	index	stranded.label	age	mental_health_care	admit_date
0	7	Not Stranded	26	0	2021-01-02

[119]:
```
e = widgets.DatePicker(
    description='Admit date',
    disabled=False
)
display(e)
```

```
DatePicker(value=None, description='Admit date')
```

[121]:
```
dfTofill.iloc[0,4]=e.value
dfTofill
```

[121]:

	index	stranded.label	age	mental_health_care	admit_date	consent
0	7	Not Stranded	26	0	2021-01-02	True

8 Concatenating the collected data to the CollectData data frame.

The concat() function from the Python *pandas* package is used to append the CollectData and dfTofill data frames.

[122]:
```
# CollectData is the first data frame
# dfTofill is the second data frame
CollectData  = pd.concat([CollectData, dfTofill])
display(CollectData)
```

```
   index stranded.label  age  mental_health_care  admit_date  consent
0      7            NaN    0                   0  2000-01-01    False
0      7   Not Stranded   26                   0  2021-01-02     True
```

9 To obtain consent to process and share the data before you save it to the working data folder?

Before moving to process data, a consent must be given, so rows without consent will be removed.

[123]:
```
CollectData=CollectData[CollectData['consent'] == True]
display(CollectData)
```

```
   index stranded.label  age  mental_health_care  admit_date  consent
0      7   Not Stranded   26                   0  2021-01-02     True
```

10 Saving the CollectData data frame

It is to save the collected data captured by the tool to “Data” folder.

[124]:
```
CollectData.to_csv('../Data/CollectedData.csv', index=False)
```

That is the CollectData data frame saved to the working ‘Data’ folder. 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.

[125]:
```
CollectData.to_csv('../RawData/CollectedDataFinal.csv', index=False)
```

When all data was done inputting, a final dataframe will be saved to “RawData” folder.