CollectingDataUsingInteractiveJupyterWidgets

June 24, 2022

1 Title: Collecting data using interactive Jupyter widgets

Author details: *Author:* Mairead Bermingham. *Contact details:* mairead.bermingham@ed.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 (your test data) and save it to your working 'Data' folder, and finally saving all the captured test data to your 'RawData'.
Data: Data consists of date, numerical data and character data from NHSRdatasets package.

2 Data

The data you will be managing on the course are from the NHSRdatasets package. This package has been created to support skills development in the NHS-R community and contains several free datasets. 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. However, for this lesson, we will use the full `ae_attendances` dataset to demonstrate how to use 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.

Note, you only need to set up widgets for the subset of the variables required for your data capture tool. We are using the full data set here, as you will be using interactive Jupyter widgets to collect different variables from your `ae_attendances` data subsets.

2.0.1 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.

[144]:
```
#Load the 'pandas' package
import pandas as pd
testData=pd.read_csv("../Data/ae_attendances_test.csv")
testData
```

```
[144]:     index      period org_code  attendances  breaches  admissions
        0    2881  2016-07-01      RXK        14488      2128        3141
        1    2896  2016-07-01      RNA         8947       596        2599
        2    4258  2018-03-01      RXK        13805      3556        3429
        3    4281  2018-03-01      RRK         9936      2154        3896
        4    5043  2018-01-01      RLQ         4532      1263        1437
        5    6471  2017-09-01      RWP         9817      2716        2921
        6    7137  2017-07-01      RJC         5811       297        1617
        7    7509  2017-06-01      RWP        10313      2824        3174
        8    9577  2018-12-01      RXK        13604      4432        3744
        9   10327  2018-10-01      RKB        12519      1937        4407
        10  12530  2018-04-01      RL4        10709      1704        2544
```

Data type We now need to check the data type in the testData data frame. Let us use the `dtypes` function from the Python *pandas* package to query the data types in the testData. The `dtypes` function returns the data types in the data frame.

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

```
Output:
index           int64
period         object
org_code       object
attendances     int64
breaches        int64
admissions      int64
dtype: object
```

The data type object is a string

Now let us collect the first row of data from the test data. Use the `df.head()` function to see the first row in the data frame(df).

The `head()` function The `head()` function lets you look at the top n rows of a data frame. By default, it shows the first five rows in a data frame. We can specify the number of rows we want to see in a data frame with the argument “n”. For example, look at the first row (n=1) of the test data:

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

```
[146]:    index      period org_code  attendances  breaches  admissions
        0   2881  2016-07-01      RXK        14488      2128        3141
```

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

```
[198]: dfTofill = pd.DataFrame({'index': [0],# Integer
                          'period': [pd.Timestamp('20000101')], # Date
                          'org_code': ['NA'], # String
                          'attendances': [0], # Integer
                          'breaches': [0], # Integer
                          'admissions': [0], # Integer
                          'breach_performance': [0.0], # Float
                          'admission_rate': [0.0], #Float
                          'consent': [False]}) # Boolean

       dfTofill
```

```
[198]:    index     period org_code  attendances  breaches  admissions  \
       0      0 2000-01-01       NA            0         0           0

          breach_performance  admission_rate  consent
       0                 0.0             0.0    False
```

Save the empty data frame to your working ‘Data’ folder:

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

The empty data frame is now saved to the working ‘Data’ folder. Now make sure to comment out the last cell (Ctrl+/), as you only need to do this once. Now let’s read in the empty data frame to collect the data from the Jupyter-widgets.

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

```
[200]:    index      period  org_code  attendances  breaches  admissions  \
       0      0  2000-01-01       NaN            0         0           0

          breach_performance  admission_rate  consent
       0                 0.0             0.0    False
```

Now let us collect the first row of data from the test data. Use the `df.head()` function to see the first row in the data frame(df).

The `head()` function The `head()` function lets you look at the top n rows of a data frame. By default, it shows the first five rows in a data frame. We can specify the number of rows we want to see in a data frame with the argument “n”. For example, look at the first row (n=1) of the test data:

```
[370]: testData.head(n=11)
```

```
[370]:     index      period org_code  attendances  breaches  admissions
       0     2881  2016-07-01      RXK        14488      2128        3141
       1     2896  2016-07-01      RNA         8947       596        2599
```

```
2    4258  2018-03-01      RXK        13805       3556         3429
3    4281  2018-03-01      RRK         9936       2154         3896
4    5043  2018-01-01      RLQ         4532       1263         1437
5    6471  2017-09-01      RWP         9817       2716         2921
6    7137  2017-07-01      RJC         5811        297         1617
7    7509  2017-06-01      RWP        10313       2824         3174
8    9577  2018-12-01      RXK        13604       4432         3744
9   10327  2018-10-01      RKB        12519       1937         4407
10  12530  2018-04-01      RL4        10709       1704         2544
```

3 Index variable

The first variable contains the index number, that allows us to connect the test data to the orginal data set "../RawData/ae_attendances.csv". We will have to use indexing to to add the index number to the 'dfTofill' file

3.0.1 Indexing in Python

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. In Python, objects are "zero-indexed" meaning the position count starts at zero.

[371]:
```
index_number=12530 #Remember to change for each record.
dfTofill.iloc[0,0]=index_number
dfTofill
```

[371]:
```
   index      period org_code  attendances  breaches  admissions  \
0  12530  2018-10-01      RKB        12519      1937        4407

   breach_performance  admission_rate  consent
0            0.352025        0.352025     True
```

4 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. Widgets can transform your notebooks from static documents to dynamic dashboards, ideal for showcasing your data story.

To use the widget framework, you need to import the *ipywidgets* Python package. The *ipywidgets* package provides a list of widgets commonly used in web apps and dashboards like dropdown, checkbox, radio buttons, etc.

[9]:
```
#Load the 'ipywidgets' package
import ipywidgets as widgets
```

4.0.1 `display()`

The *IPython.display* package is used to display different objects in Jupyter. You can also explicitly display a widget using the `display()` function from the *IPython.display* package

[10]:
```
#Load the 'IPython.display' package
from IPython.display import display
```

5 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 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. Consent is defined in Article 4 as: "Consent of the data subject means any freely given, specific informed and unambiguous indication of the data subject's wishes by which he or she, by a statement or by a clear affirmative action, signifies agreement to the processing of personal data relating to him or her".

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.

5.1 Boolean widgets

Boolean widgets are designed to display a boolean value.

5.1.1 Checkbox widget

[11]:
```
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
)
```

[12]:
```
display(a)
```

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

[372]:
```
dfTofill.iloc[0,8]=a.value
dfTofill
```

[372]:
```
   index      period org_code  attendances  breaches  admissions  \
0  12530  2018-10-01      RKB        12519      1937        4407

   breach_performance  admission_rate  consent
0            0.352025        0.352025     True
```

6 The period variable

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

Data type We now need to check the data type in the testData data frame. Let us use the `dtypes` function from the Python *pandas* package to query the data types in the testData. The `dtypes` function returns the data types in the data frame.

[14]:

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

```
object
```

The data type object is a string.

The `head()` function The `head()` function lets you look at the top n rows of a data frame. By default, it shows the first five rows in a data frame. We can specify the number of rows we want to see in a data frame with the argument “n”. For example, look at the first row (n=1) of the test data:

[373]:

```
testData.head(n=11)
```

[373]:

```
    index      period org_code  attendances  breaches  admissions
0    2881  2016-07-01      RXK        14488      2128        3141
1    2896  2016-07-01      RNA         8947       596        2599
2    4258  2018-03-01      RXK        13805      3556        3429
3    4281  2018-03-01      RRK         9936      2154        3896
4    5043  2018-01-01      RLQ         4532      1263        1437
5    6471  2017-09-01      RWP         9817      2716        2921
6    7137  2017-07-01      RJC         5811       297        1617
7    7509  2017-06-01      RWP        10313      2824        3174
8    9577  2018-12-01      RXK        13604      4432        3744
9   10327  2018-10-01      RKB        12519      1937        4407
10  12530  2018-04-01      RL4        10709      1704        2544
```

6.0.1 DatePicker widget

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

[16]:

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

```
DatePicker(value=None, description='Period')
```

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

```
[374]:    index      period org_code  attendances  breaches  admissions  \
       0  12530  2018-04-01      RKB        12519      1937        4407

          breach_performance  admission_rate  consent
       0            0.352025        0.352025     True
```

6.1 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, 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. If you want 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`. For example, the organisation associated with the ODS code ‘AF003’ is Parkway health centre.

Data type We now need to check the data type in the testData data frame. Let us use the `dtypes` function from the Python *pandas* package to query the data types in the testData. The `dtypes` function returns the data types in the data frame.

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

```
object
```

The data type object is a string.

Describe the test data Here we are going to use the `describe()` function from the *numpy* Python package to calculate summary statistics for the testData data frame. The numpy package is the core package for scientific computing in Python. The `describe()` function from the *numpy* package computes the descriptive statistics.

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

```
[156]:                index      period org_code   attendances     breaches  \
       count     11.000000          11       11     11.000000    11.000000
       unique          NaN           9        8           NaN          NaN
       top             NaN  2016-07-01      RXK           NaN          NaN
       freq            NaN           2        3           NaN          NaN
       mean    6628.181818         NaN      NaN  10407.363636  2146.090909
       std     3160.349089         NaN      NaN   3183.123286  1218.150356
       min     2881.000000         NaN      NaN   4532.000000   297.000000
       25%     4269.500000         NaN      NaN   9382.000000  1483.500000
```

```
50%      6471.000000          NaN      NaN  10313.000000  2128.000000
75%      8543.000000          NaN      NaN  13061.500000  2770.000000
max     12530.000000          NaN      NaN  14488.000000  4432.000000

          admissions
count      11.000000
unique           NaN
top              NaN
freq             NaN
mean     2991.727273
std       911.047868
min      1437.000000
25%      2571.500000
50%      3141.000000
75%      3586.500000
max      4407.000000
```

Applying *pandas* `unique()` function We must first use the *pandas* package `unique()` function to get the unique Organisation data service (ODS) codes in the test data.

[157]:

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

[157]:

```
['RXK', 'RNA', 'RRK', 'RLQ', 'RWP', 'RJC', 'RKB', 'RL4']
```

The `head()` function The `head()` function lets you look at the top n rows of a data frame. By default, it shows the first five rows in a data frame. We can specify the number of rows we want to see in a data frame with the argument "n". For example, look at the first row (n=1) of the test data:

[375]:

```
testData.head(n=11)
```

[375]:

```
    index      period org_code  attendances  breaches  admissions
0    2881  2016-07-01      RXK        14488      2128        3141
1    2896  2016-07-01      RNA         8947       596        2599
2    4258  2018-03-01      RXK        13805      3556        3429
3    4281  2018-03-01      RRK         9936      2154        3896
4    5043  2018-01-01      RLQ         4532      1263        1437
5    6471  2017-09-01      RWP         9817      2716        2921
6    7137  2017-07-01      RJC         5811       297        1617
7    7509  2017-06-01      RWP        10313      2824        3174
8    9577  2018-12-01      RXK        13604      4432        3744
9   10327  2018-10-01      RKB        12519      1937        4407
10  12530  2018-04-01      RL4        10709      1704        2544
```

6.2 Selection widgets

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

[159]:

```
c=widgets.Select(
    options=org_code,
    value='RXK',
    rows=len(org_code),
    description='ODS code:',
    disabled=False
)
display(c)
```

```
Select(description='ODS code:', options=('RXK', 'RNA', 'RRK', 'RLQ', 'RWP', 'RJC', 'RKB', 'RL4
```

[376]:

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

[376]:

```
   index      period org_code  attendances  breaches  admissions  \
0  12530  2018-04-01      RL4        12519      1937        4407

   breach_performance  admission_rate  consent
0            0.352025        0.352025     True
```

6.3 The type variable

The type variable contains 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 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 an appointment. A service mainly or entirely appointment-based (for example, a GP Practice or Outpatient clinic) is excluded even though it may treat a number of patients with minor illnesses or injury. Excludes NHS walk-in centres.(National Health Service, 2020)

Data type We now need to check the data type in the testData data frame. Let us use the `dtypes` function from the Python *pandas* package to query the data types in the testData. The `dtypes` function returns the data types in the data frame.

[82]:

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

The data type object is a string.

Applying *pandas* `unique()` function We must first use the *pandas* package `unique()` function to get the unique department type in the test data.

[83]:
```
#type=list(testData['type'].unique())
#type
```

The `head()` function The `head()` function lets you look at the top n rows of a data frame. By default, it shows the first five rows in a data frame. We can specify the number of rows we want to see in a data frame with the argument "n". For example, look at the first row (n=1) of the test data:

[84]:
```
#testData.head(n=1)
```

6.3.1 RadioButtons

[85]:
```
#d=widgets.RadioButtons(
#    options=type,
#     value='other',
#    description='Type:',
#    disabled=False
#)
#display(d)
```

[86]:
```
#dfTofill.iloc[0,3]=d.value
#dfTofill
```

7 The attendances variable

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

Data type We now need to check the data type in the testData data frame. Let us use the `dtypes` function from the Python *pandas* package to query the data types in the testData. The `dtypes` function returns the data types in the data frame.

[87]:
```
print(result[3])
```

```
int64
```

The `head()` function The `head()` function lets you look at the top n rows of a data frame. By default, it shows the first five rows in a data frame. We can specify the number of rows we want to see in a data frame with the argument "n". For example, look at the first row (n=1) of the test data:

[377]:
```
testData.head(n=11)
```

[377]:

```
    index      period org_code  attendances  breaches  admissions
0    2881  2016-07-01      RXK        14488      2128        3141
1    2896  2016-07-01      RNA         8947       596        2599
2    4258  2018-03-01      RXK        13805      3556        3429
3    4281  2018-03-01      RRK         9936      2154        3896
4    5043  2018-01-01      RLQ         4532      1263        1437
5    6471  2017-09-01      RWP         9817      2716        2921
6    7137  2017-07-01      RJC         5811       297        1617
7    7509  2017-06-01      RWP        10313      2824        3174
8    9577  2018-12-01      RXK        13604      4432        3744
9   10327  2018-10-01      RKB        12519      1937        4407
10  12530  2018-04-01      RL4        10709      1704        2544
```

7.1 Numeric widgets

There are many widgets distributed with ipywidgets that are designed to display numeric values. Widgets exist for displaying integers and floats, both bounded and unbounded. The integer widgets share a similar naming scheme to their floating point counterparts. By replacing Float with Int in the widget name, you can find the Integer equivalent.

7.1.1 IntText

[26]:

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

```
IntText(value=0, description='Attendances:')
```

[378]:

```
dfTofill.iloc[0,3]=e.value
dfTofill
```

[378]:

```
   index      period org_code  attendances  breaches  admissions  \
0  12530  2018-04-01      RL4        10709      1937        4407

   breach_performance  admission_rate  consent
0            0.352025        0.352025     True
```

8 The breaches variable

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

Data type We now need to check the data type in the testData data frame. Let us use the `dtypes` function from the Python *pandas* package to query the data types in the testData. The `dtypes` function returns the data types in the data frame.

[28]:

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

```
int64
```

[379]:

```
testData.head(11)
```

[379]:

```
    index      period org_code  attendances  breaches  admissions
0    2881  2016-07-01      RXK        14488      2128        3141
1    2896  2016-07-01      RNA         8947       596        2599
2    4258  2018-03-01      RXK        13805      3556        3429
3    4281  2018-03-01      RRK         9936      2154        3896
4    5043  2018-01-01      RLQ         4532      1263        1437
5    6471  2017-09-01      RWP         9817      2716        2921
6    7137  2017-07-01      RJC         5811       297        1617
7    7509  2017-06-01      RWP        10313      2824        3174
8    9577  2018-12-01      RXK        13604      4432        3744
9   10327  2018-10-01      RKB        12519      1937        4407
10  12530  2018-04-01      RL4        10709      1704        2544
```

8.0.1 IntText

[30]:

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

```
IntText(value=0, description='Breaches:')
```

[380]:

```
dfTofill.iloc[0,4]=f.value
dfTofill
```

[380]:

```
   index      period org_code  attendances  breaches  admissions  \
0  12530  2018-04-01      RL4        10709      1704        4407

   breach_performance  admission_rate  consent
0            0.352025        0.352025     True
```

The admissions variable The admissions variable includes the number of attendances that resulted in an admission to the hospital.(Chris Mainey, 2021)

Data type We now need to check the data type in the testData data frame. Let us use the `dtypes` function from the Python *pandas* package to query the data types in the testData. The `dtypes` function returns the data types in the data frame.

[32]:

```
print(result[5])
```

```
int64
```

It is an integer variable.

The `head()` function The `head()` function lets you look at the top n rows of a data frame. By default, it shows the first five rows in a data frame. We can specify the number of rows we want to see in a data frame with the argument “n”. For example, look at the first row (n=1) of the test data:

```
[381]: testData.head(n=11)
```

```
[381]:     index      period org_code  attendances  breaches  admissions
       0    2881  2016-07-01      RXK        14488      2128        3141
       1    2896  2016-07-01      RNA         8947       596        2599
       2    4258  2018-03-01      RXK        13805      3556        3429
       3    4281  2018-03-01      RRK         9936      2154        3896
       4    5043  2018-01-01      RLQ         4532      1263        1437
       5    6471  2017-09-01      RWP         9817      2716        2921
       6    7137  2017-07-01      RJC         5811       297        1617
       7    7509  2017-06-01      RWP        10313      2824        3174
       8    9577  2018-12-01      RXK        13604      4432        3744
       9   10327  2018-10-01      RKB        12519      1937        4407
       10  12530  2018-04-01      RL4        10709      1704        2544
```

8.0.2 IntText

```
[34]: g=widgets.IntText(
          value=0,
          description='Admissions:',
          disabled=False)
      display(g)
```

```
IntText(value=0, description='Admissions:')
```

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

```
[382]:    index      period org_code  attendances  breaches  admissions  \
       0  12530  2018-04-01      RL4        10709      1704        2544

          breach_performance  admission_rate  consent
       0            0.352025        0.352025     True
```

9 The performance variable

The performance variable was calculated for the whole of England as (1 - breaches)/ attendances.

Data type We now need to check the data type in the testData data frame. Let us use the `dtypes` function from the Python *pandas* package to query the data types in the testData. The `dtypes` function returns the data types in the data frame.

```
[243]: # print(result[6])
```

It is a float variable.

The `head()` function The `head()` function lets you look at the top n rows of a data frame. By default, it shows the first five rows in a data frame. We can specify the number of rows we want to see in a data frame with the argument "n". For example, look at the first row (n=1) of the test data:

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

9.0.1 FloatText

10 And the admission rate variable

The admission rate variable was calculated as admissions/attendances.

```
[383]: # h=widgets.FloatText(
       #     value=0.0,
       #     description='Performance:',
       #     disabled=False
       # )
       # display(h)

       def performance(row):
           return 1-dfTofill.iloc[row,4]/dfTofill.iloc[row,3]
       h=float(performance(0))
       def admission(row):
           return dfTofill.iloc[row,5]/dfTofill.iloc[row,3]
       h=float(admission(0))
       dfTofill.iloc[0,6]=h
       dfTofill.iloc[0,7]=h
       dfTofill
```

```
[383]:    index      period org_code  attendances  breaches  admissions  \
       0  12530  2018-04-01      RL4        10709      1704        2544

          breach_performance  admission_rate  consent
       0            0.237557        0.237557     True
```

11 Concatenating the collected data to the CollectData data frame.

Let us use the `concat()` function from the Python *pandas* package to append the CollectData and dfTofill data frames. The concat() function is used to concatenate *pandas* objects.

[384]:
```
# CollectData is the first data frame
# dfTofill is the second data frame
CollectData  = pd.concat([CollectData, dfTofill])
display(CollectData)
```

```
   index      period org_code  attendances  breaches  admissions  \
0   2881  2016-07-01      RXK         1488      2128        3141
0   2896  2016-07-01      RNA         8947       596        2599
0   4258  2018-03-01      RXK        13805      3556        3429
0   4281  2018-03-01      RRK         9936      2154        3896
0   5043  2018-01-01      RLQ         4532      1263        1437
0   6471  2017-09-01      RWP         9817      2716        2921
0   7137  2017-07-01      RJC         5811       297        1617
0   7509  2017-06-01      RWP        10313      2824        3174
0   9577  2018-12-01      RXK        13604      4432        3744
0  10327  2018-10-01      RKB        12519      1937        4407
0  12530  2018-04-01      RL4        10709      1704        2544

   breach_performance  admission_rate  consent
0           -0.430108        2.110887     True
0            0.933385        0.290488     True
0            0.742412        0.248388     True
0            0.783213        0.392110     True
0            0.317079        0.317079     True
0            0.297545        0.297545     True
0            0.278265        0.278265     True
0            0.307767        0.307767     True
0            0.275213        0.275213     True
0            0.352025        0.352025     True
0            0.237557        0.237557     True
```

11.1 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.

[385]:
```
CollectData=CollectData[CollectData['consent'] == True]
display(CollectData)
```

```
   index      period org_code  attendances  breaches  admissions  \
0   2881  2016-07-01      RXK        1488      2128         3141
0   2896  2016-07-01      RNA        8947       596         2599
0   4258  2018-03-01      RXK       13805      3556         3429
0   4281  2018-03-01      RRK        9936      2154         3896
0   5043  2018-01-01      RLQ        4532      1263         1437
0   6471  2017-09-01      RWP        9817      2716         2921
0   7137  2017-07-01      RJC        5811       297         1617
0   7509  2017-06-01      RWP       10313      2824         3174
0   9577  2018-12-01      RXK       13604      4432         3744
0  10327  2018-10-01      RKB       12519      1937         4407
0  12530  2018-04-01      RL4       10709      1704         2544

   breach_performance  admission_rate  consent
0           -0.430108        2.110887     True
0            0.933385        0.290488     True
0            0.742412        0.248388     True
0            0.783213        0.392110     True
0            0.317079        0.317079     True
0            0.297545        0.297545     True
0            0.278265        0.278265     True
0            0.307767        0.307767     True
0            0.275213        0.275213     True
0            0.352025        0.352025     True
0            0.237557        0.237557     True
```

11.1.1 Saving the CollectData data frame

Saving the data collected by your data-capture tool to the working data folder:

```
[386]: 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.

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

That is the final CollectData data frame saved to the 'RawData' folder.

I hope these examples help you to improve your Python programming skills. Happy Coding!