AI for Medicine Course 3 Week 1 lecture notebook

Pandas for a Medical Dataset

Welcome to this lecture notebook! In this week's graded assignment, you will be using pandas quite often to work with dataframes.

- To get you ready for assignment, you'll familiarize yourself with some objects in the `pandas` library, along with their data types.
- Then you'll see how you can leverage pandas to get information from a dataset.

Import Library

In [1]:
```
# import libraries
import pandas as pd
```

Load your data and check its shape

pandas.read_csv takes in a file name, assuming that the file is formatted as comma separated values (csv).

- You can choose one of the columns to be the row 'index', which is an ID associated with each row.

In [2]:
```
# Read the csv data, setting the 0th column as the row index
data = pd.read_csv("dummy_data.csv", index_col=0)

# Display the data's number of rows and columns
print(f"Data has {data.shape[0]} rows and {data.shape[1]} columns.\
n")
data.head()
```

```
Data has 50 rows and 5 columns.
```

Out[2]:

	sex	age	obstruct	outcome	TRTMT
1	0	57	0	1	True
2	1	68	0	0	False
3	0	72	0	0	True
4	0	66	1	1	True
5	1	69	0	1	False

Below is a description of all the fields:

- `sex (binary): 1 if Male, 0 otherwise`
- `age (int): age of patient at start of the study`
- `obstruct (binary): obstruction of colon by tumor`
- `outcome (binary): 1 if died within 5 years`
- `TRTMT (binary): patient was treated`

Introducing the DataFrame

In [3]:
```
# show the data type of the dataframe
print(type(data))
```

```
<class 'pandas.core.frame.DataFrame'>
```

You can see that your data is of type `DataFrame`. A `DataFrame` is a two-dimensional, labeled data structure with columns that can be of different data types. Dataframes are a great way to organize your data, and are the most common object in `pandas`. If you are unfamiliar with them, check the official documentation (https://pandas.pydata.org/pandas-docs/stable/index.html).

In case you're only interested in a single column (or feature) of the data, access that single column by using the "." dot notation, in which you specify the dataframe followed by a dot and the name of the column you are interested in, like this:

```
In [4]: data.TRTMT.head()
```

```
Out[4]: 1     True
        2    False
        3     True
        4     True
        5    False
        Name: TRTMT, dtype: bool
```

Notice the `head()` method. This method prints only the first five rows, so the output of the cell can be quickly and easily read. Try removing it and see what happens.

Introducing the Series

```
In [5]: print(type(data.TRTMT))
```

```
        <class 'pandas.core.series.Series'>
```

Each column of a DataFrame is of type `Series`, which are one-dimensional, labeled arrays that can contain any data type, plus its index. Series are similar to lists in Python, with one important difference: each Series can only contain one type of data.

Many of the methods and operations supported by DataFrames are also supported by Series. When in doubt, always check the documentation!

There are several ways of accessing a single column of a DataFrame. The methods you're about to see all do the same thing.

- Dot notation is simple to type, but doesn't work when the column name has a space. See some examples of where dot notation will not work (https://www.dataschool.io/pandas-dot-notation-vs-brackets/).
- Bracket notation always works.

In [6]:
```
# Use dot notation to access the TRTMT column
data.TRTMT

# Use .loc to get all rows using ":", for column TRTMT
data.loc[:,"TRTMT"]

# Use bracket notation to get the TRTMT column
data["TRTMT"]

print(data.TRTMT.equals(data.loc[:,"TRTMT"]))
print(data.TRTMT.equals(data["TRTMT"]))
```

```
True
True
```

Slicing the DataFrame

Most of the time you'll want a subset (or a slice) of the DataFrame that meets some criteria. For example, if you wanted to analyze all of the features for patients who are 50 years or younger, you can slice the DataFrame like this:

In [7]:
```
data[data.age <= 50]
```

Out[7]:

	sex	age	obstruct	outcome	TRTMT
6	1	43	0	1	True
15	1	46	1	0	False
19	0	34	1	1	True
24	0	50	0	0	True
32	0	33	1	0	True
33	0	49	0	1	False
34	1	47	0	0	False
42	0	39	1	0	False
45	1	40	0	0	True
67	1	49	0	0	True
70	0	40	0	0	False

What if you wanted to filter a DataFrame based on multiple conditions?

- To do this, use the "&" as the 'and' operator. Don't use `and`.
- You can use '|' as the 'or' operator. Don't use `or`.

```
# Trying to combine two conditions using `and` won't work
data[(data.age <= 50) and (data.TRTMT == True)]

ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bo
ol(), a.item(), a.any() or a.all().
```

- Don't forget the parentheses around each condition!
- Without parentheses, this won't work.

```
# Trying to combine two conditions without parentheses results in an er
ror
data[ data.age <= 50 & data.TRTMT == True]

ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bo
ol(), a.item(), a.any() or a.all().
```

In [8]:
```
# Get patients aged 50 or less who received treatment
data[(data.age <= 50) & (data.TRTMT == True)]
```

Out[8]:

	sex	age	obstruct	outcome	TRTMT
6	1	43	0	1	True
19	0	34	1	1	True
24	0	50	0	0	True
32	0	33	1	0	True
45	1	40	0	0	True
67	1	49	0	0	True

When slicing a DataFrame the resulting type will be a DataFrame as well:

In [9]:
```
type(data[(data.age <= 50) & (data.TRTMT == True)])
```

Out[9]: `pandas.core.frame.DataFrame`

More Advanced Operations

Now let's dive into some useful properties of DataFrames and Series that allow for more advanced calculations.

In [10]:
```
# Applying len() to the df yields the number of rows
print(f"len: {len(data[(data.age <= 50)])}")

# Accessing the 'shape' attribute of the df yields a tuple of the f
orm (rows, cols)
print(f"shape (rows, cols) {data[(data.age <= 50)].shape}")

# Accessing the 'size' attribute of the df yields the number of ele
ments in the df:
print(f"size: {data[(data.age <= 50)].size}")
```

```
len: 11
shape (rows, cols) (11, 5)
size: 55
```

In [11]:
```
# Applying len() to the df yields the number of rows
print(f"{len(data.TRTMT)}")

# Accessing the 'shape' attribute of the df yields a tuple of the f
orm (rows, cols)
print(f"{data.TRTMT.shape}")

# Accessing the 'size' attribute of the df yields the number of ele
ments in the df:
print(f"{data.TRTMT.size}")
```

```
50
(50,)
50
```

Exercise

Using what you've seen so far, can you calculate the proportion of the patients who are male?

In [12]:
```
prop_male_patients = len(data[(data.sex == 1)])/len(data.sex)
print(f"Your answer: {prop_male_patients}, Expected: {21/50}")
```

```
Your answer: 0.42, Expected: 0.42
```

mean( ) Method

One handy hack you can use when dealing with binary data is to use the `mean()` method of a Series to calculate the proportion of occurrences that are equal to 1.

Note this should also work with bool data since Python treats booleans as numbers when applying math operations.

- `True` is treated as the number `1`
- `False` is treated as the number `0`

In [13]:
```
# Calculate the proportion of the `sex` column that is `True` (1).
data.sex.mean()
```

Out[13]: 0.42

Updating Values

So far you've only accessed values of a DataFrame or Series. Sometimes you may need to update these values.

Let's look at the original DataFrame one more time:

In [14]:
```
# View dataframe
data.head()
```

Out[14]:

	sex	age	obstruct	outcome	TRTMT
1	0	57	0	1	True
2	1	68	0	0	False
3	0	72	0	0	True
4	0	66	1	1	True
5	1	69	0	1	False

Let's say you detected an error in the data, where the second patient was actually treated.

- To update the data, you can use .loc[row, col] and specify the row and column you want to update.
- Notice that because the dataframe's index is defined, the first row is at index 1 and not 0.
- If the index was not set, then indexing would start at 0.

```
# Try to access patient 0, and note the error message
data.loc[0,'TRTMT']

KeyError: 0
```

In [15]:
```
data.loc[2,'TRTMT']
```

Out[15]: False

In [16]:
```
data.loc[2, "TRTMT"] = True
data.head()
```

Out[16]:

	sex	age	obstruct	outcome	TRTMT
1	0	57	0	1	True
2	1	68	0	0	True
3	0	72	0	0	True
4	0	66	1	1	True
5	1	69	0	1	False

Now, you've found out that there was another issue with the data that needs to be corrected. This study only includes females, so the `sex` column should be set to 0 for all patients.

You can update the whole column (or Series) using .loc[row, col] once again, but this time using ":" for rows.

In [17]:
```
data.loc[:, "sex"] = 0
data.head()
```

Out[17]:

	sex	age	obstruct	outcome	TRTMT
1	0	57	0	1	True
2	0	68	0	0	True
3	0	72	0	0	True
4	0	66	1	1	True
5	0	69	0	1	False

You can access a range of rows by specifying the `start:end`, where the `end` index is included.

- Note that the range is inclusive of the end (other functions in Python exclude the end of the range from the output).

In [18]:
```
# Access patients at index 3 to 4, including 4.
data.loc[3:4,:]
```

Out[18]:

	sex	age	obstruct	outcome	TRTMT
3	0	72	0	0	True
4	0	66	1	1	True

Congratulations, you have completed this lecture notebook!

Welcome to the wonderful world of Pandas! You will be using these pandas functions in this week's graded assignment.