ApplyingFilters

August 8, 2023

1 Applying Filters

First, run the cell below to import the packages that we will use.

[1]:
```
import pandas as pd
import numpy as np
import os
```

1.1 Step 1: Inspect the Data

We will be working with the "adult" data set which contains Census information from 1994.

The code cell below loads the data set by using the Pandas `pd.read_csv()` function to read in the CSV file that contains the data. The file is located in a folder named "data" and has the name "adult.data.partial." The `pd.read_csv()` function returns a Pandas DataFrame. We will assign the data to a DataFrame object called `df`. Run the cell below to load the data.

[2]:
```
filename = os.path.join(os.getcwd(), "data", "adult.data.partial")
df = pd.read_csv(filename, header=0)
```

In the code cell below, use the Pandas DataFrame `head()` method to display the first few rows of the DataFrame `df`.

[3]:
```
# YOUR CODE HERE - this cell will not be graded
df.head()
```

[3]:
```
   age  workclass  fnlwgt     education  education-num      marital-status  \
0   36  State-gov  112074     Doctorate             16       Never-married
1   35    Private   32528       HS-grad              9  Married-civ-spouse
2   21    Private  270043  Some-college             10       Never-married
3   45    Private  168837  Some-college             10  Married-civ-spouse
4   39    Private  297449     Bachelors             13  Married-civ-spouse

          occupation   relationship   race  sex_selfID  capital-gain  \
0     Prof-specialty  Not-in-family  White  Non-Female             0
1  Handlers-cleaners        Husband  White  Non-Female             0
2      Other-service      Own-child  White      Female             0
3       Adm-clerical           Wife  White      Female             0
4     Prof-specialty        Husband  White  Non-Female             0

   capital-loss  hours-per-week native-country  label
```

```
0              0              45  United-States  <=50K
1              0              45  United-States  <=50K
2              0              16  United-States  <=50K
3              0              24         Canada   >50K
4              0              40  United-States   >50K
```

Use the Pandas shape property to display the number of rows and columns in the data. If you forgot the syntax, call `df.shape?` in the cell below to read the documentation.

How many examples (rows) do we have? How many features (columns)?

[4]:

```
# YOUR CODE HERE - this cell will not be graded
df.shape
```

[4]:

```
(7000, 15)
```

1.2 Step 2: Random Sampling of the Data

Random sampling from the data using `np.random.choice` and `loc`

We will start by sampling some of the data. You will learn more about sampling in a future exercise.

For now, imagine that you need to randomly select 30% of the data examples.

First, we will do this the 'NumPy' way.

In the cell below, some code is already pre-written to randomly select 30% of rows and save their indices to variable `indices`.

The variable `indices` only contains the indices of rows, not the actual data in the rows.

Complete the code below to obtain only the rows in `df` with the indices specified in variable `indices`.

You will recall that you can use `loc[]` to index into a DataFrame to acces rows. Use `loc[]` accomplish this task.

Save this result to a new DataFrame named `df_subset`.

1.2.1 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

[5]:

```
percentage = 0.3
num_rows = df.shape[0]
indices = np.random.choice(df.index, size=int(percentage*num_rows),␣
 ↪replace=False)



# YOUR CODE HERE
df_subset=df.loc[indices]
```

1.2.2 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell.

[6]:

```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testSubset

try:
    p, err = testSubset(df, df_subset)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Correct!
```

Note that you could write some of the code in the cell above in a single line, without creating a new array `indices`, which you likely won't use again. Note how the cell below accomplished that.

[7]:

```
percentage = 0.3
num_rows = df.shape[0]

df_subset = df.loc[np.random.choice(df.index, size=int(percentage*num_rows),␣
 ↪replace=False)]
```

This compressed style may seem a little bit bulky and intimidating at first, but will become easier to comprehend as you get more experience.

Let's check that our sampling worked. You should expect to see that the shape of the new object `df_subset` reflects that it has 30% of the original row number:

[8]:

```
print(df.shape) #original number of rows
print(df_subset.shape) #30% of the number of rows
```

```
(7000, 15)
(2100, 15)
```

But did you actually select the rows randomly? Look at the indices in the new DataFrame:

[9]:

```
df_subset.head()
```

[9]:

```
      age workclass  fnlwgt     education  education-num      marital-status  \
2003   28   Private  106951       HS-grad              9       Never-married
6369   42   Private   78765  Some-college             10  Married-civ-spouse
1337   25   Private  209428  Some-college             10  Married-civ-spouse
1027   30       NaN  164940       HS-grad              9           Separated
275    36   Private  115700       HS-grad              9  Married-civ-spouse

             occupation   relationship   race  sex_selfID  capital-gain  \
2003  Handlers-cleaners  Not-in-family  White  Non-Female             0
6369       Craft-repair        Husband  White  Non-Female          3103
1337              Sales        Husband  White  Non-Female             0
1027                NaN      Unmarried  Black      Female             0
275               Sales        Husband  White  Non-Female             0
```

```
      capital-loss  hours-per-week native-country  label
2003             0              42  United-States  <=50K
6369             0              45  United-States   >50K
1337             0              25    El-Salvador  <=50K
1027             0              25  United-States  <=50K
275              0              50  United-States  <=50K
```

It seems random. To convince yourself that it is, try running the sampling code above again, and then re-run the `head()` method to above and inspect the results. You should see a different random sample each time you re-run the sampling code cell.

We will now see how to perform sampling using the `Pandas` way:

[10]:
```
percentage = 0.3
num_rows = df.shape[0]

df_subset = df.sample(int(percentage*num_rows))
df_subset.head()
```

[10]:
```
      age workclass  fnlwgt     education  education-num      marital-status  \
6702   26   Private  181655     Assoc-voc             11  Married-civ-spouse
3748   18   Private  210828  Some-college             10       Never-married
4430   41   Private  187802  Some-college             10            Divorced
3923   46   Private  133616  Some-college             10            Divorced
5022   27   Private  142075  Some-college             10       Never-married

             occupation   relationship   race  sex_selfID  capital-gain  \
6702       Adm-clerical        Husband  White  Non-Female             0
3748  Handlers-cleaners      Own-child  Other  Non-Female             0
4430       Tech-support  Not-in-family  White  Non-Female             0
3923       Adm-clerical      Unmarried  White      Female             0
5022      Other-service      Own-child  White  Non-Female             0

      capital-loss  hours-per-week native-country  label
6702          2377              45  United-States  <=50K
3748             0              30  United-States  <=50K
4430             0              50  United-States  <=50K
3923             0              40  United-States  <=50K
5022             0              24  United-States  <=50K
```

1.3 Step 3: Filter a DataFrame by Column Values

Imagine that you want to examine only the private sector employees that we have in DataFrame `df`. The cell below contains a conditional statement `df['workclass'] =='Private'`

This will evaluate to a collection of True/False values per row. A value of True indicates that the corresponding row fulfills the condition. This collection of True/False values is of data type Pandas Series (a one-dimensional array). The array is assigned to variable `condition`.

Run the cell below and inspect the results.

```
[11]: condition = df['workclass'] =='Private'
      condition
```

```
[11]: 0       False
      1        True
      2        True
      3        True
      4        True
              ...
      6995     True
      6996     True
      6997    False
      6998     True
      6999     True
      Name: workclass, Length: 7000, dtype: bool
```

In the code cell below, use the `condition` variable to extract the private employee sector data from data DataFrame `df`. Hint: Index into `df` using bracket notation and supply it the variable `condition`. Save the results to variable `df_private`. Use the `head()` method to inspect the new DataFrame `df_private`.

1.3.1 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

```
[12]: # YOUR CODE HERE
      df_private=df[condition]
      df_private.head()
```

```
[12]:    age workclass  fnlwgt     education  education-num      marital-status  \
      1   35   Private   32528       HS-grad              9  Married-civ-spouse
      2   21   Private  270043  Some-college             10       Never-married
      3   45   Private  168837  Some-college             10  Married-civ-spouse
      4   39   Private  297449     Bachelors             13  Married-civ-spouse
      5   27   Private  233421  Some-college             10       Never-married

                occupation relationship   race  sex_selfID  capital-gain   \
      1  Handlers-cleaners      Husband  White  Non-Female             0
      2      Other-service    Own-child  White      Female             0
      3       Adm-clerical         Wife  White      Female             0
      4     Prof-specialty      Husband  White  Non-Female             0
      5       Adm-clerical    Own-child  White  Non-Female             0

         capital-loss  hours-per-week native-country  label
      1             0              45  United-States  <=50K
      2             0              16  United-States  <=50K
      3             0              24         Canada   >50K
      4             0              40  United-States   >50K
      5             0              20  United-States  <=50K
```

1.3.2 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell.

[13]:

```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testPrivate

try:
    p, err = testPrivate(df, df_private, condition)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Correct!
```

How many of the rows are in the new DataFrame `df_private`?

In the cell below, display the number of rows in DataFrame `df_private` using the `shape` property. Save the results to variable `num_rows` and print `num_rows`. Hint: Recall that the `shape` property returns a tuple, with the first value corresponding to the number of rows and the second value corresponding to the number of columns.

1.3.3 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

[14]:

```
# YOUR CODE HERE
num_rows=df_private.shape[0]
print(num_rows)
```

```
4879
```

1.3.4 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell.

[15]:

```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testRows

try:
    p, err = testRows(num_rows)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Correct!
```

1.4 Step 4. Data Analysis using Filtering

The code cell below finds the average age of people who self-reported as female in DataFrame `df`.

[16]:
```
condition = df['sex_selfID']=='Female'
df[condition]['age'].mean()
```

[16]:
```
36.764213309828115
```

Notice that here we do not create a new DataFrame for the filtered data. Instead, we perform the computation and display the result. If you do not anticipate working further with a subset of your DataFrame
(e.g., querying it or finding more summary statistics about Females), then you don't need to save your results to a new DataFrame object.

As a practice, use the code cell below to play around with the statement that computes the mean: `df[condition]['age'].mean()`:

In particular: - Write `df[condition]` in the cell. run the cell and inspect the results. - Next, write `df[condition]['age']` in the cell. Run the cell and inspect the new DataFrame. - Next, write `df[condition]['age'].mean()` in the cell. Run the cell and inspect the results. ______

[19]:
```
# YOUR CODE HERE - this cell will not be graded
df[condition]['age'].mean
```

[19]:
```
<bound method Series.mean of 2       21
3       45
8       20
10      54
17      41
        ..
6980    57
6985    44
6986    27
6991    17
6996    19
Name: age, Length: 2269, dtype: int64>
```

Next you want to know how many people work for the local government for more than 40 hours per week. Using the code above as a guide, in the code cell below: 1. Define the conditions that will find the appropriate data from DataFrame `df`. 2. Apply the condition to DataFrame `df`. 3. Use the `shape` property to obtain the number of rows and assign the results to variable `rows`.

Follow these steps:

1. Create the first condition and name it `condition1`. `condition1` will look for the number of people who work for the local government. Employment information is found in the column `workclass`. The value is `Local-gov`.

2. Create the second condition and name it `condition2`. `condition2` will check whether the number of hours worked per week is more than 40 hours. The number of hours worked can be found in the column `hours-per-week`.

3. Combine these two conditions using the `&` operator to create a compound statement. Assign that to variable `condition` (`condition = condition1 & condition2`).

4. Apply `condition` to DataFrame `df` using bracket notation, and save the result to DataFrame`df_local`.
5. Use the `shape` property to obtain the number of rows in `df_local`. Assign the result to variable `rows`.

1.4.1 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

[21]:
```
# YOUR CODE HERE
condition1=df['workclass']=='Local-gov'
condition2=df['hours-per-week']>40
condition=condition1&condition2
df_local=df[condition]
rows=df_local.shape[0]
```

1.4.2 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell.

[22]:
```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testCondition

try:
    p, err = testCondition(df, condition1, condition2, condition, df_local, rows)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Correct!
```

Sometimes your data may contain missing values. One such column that contains missing values in DataFrame `df` is `native-country`. Not everyone's native country has been supplied. Such columns contain the value `Nan`.

The code cell below randomly samples 50% of rows for which the native country information is available and ignores missing values. It uses pandas `notnull()` method. You can read more about `notnull()` in the online documentation.

[23]:
```
percentage = 0.5

# obtain all rows in which the column 'native-country' contains a value
df_country_notnull = df[df['native-country'].notnull()]

# obtain the number of rows in df_country_notnull
num_rows = df_country_notnull.shape[0]
```

```
# obtain a 50% random sample of rows from df_country_notnull and save the␣
 ↪indices of these rows
indices = np.random.choice(df_country_notnull.index,␣
 ↪size=int(percentage*num_rows), replace=False)

# using the row indices, save these row values to new DataFrame df_filtered
df_filtered = df_country_notnull.loc[indices]
```

In the code cell below, find the mean age of individuals in DataFrame `df_filtered` and save the value to variable `mean_age`.

1.4.3 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

[24]:
```
# YOUR CODE HERE
mean_age=df_filtered['age'].mean()
```

1.4.4 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell.

[25]:
```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testMean
try:
    p, err = testMean(df_filtered, mean_age)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Correct!
```

You have been selecting a single column (e.g., 'age') by using bracket notation `df_filtered['age']`. You will sometimes also encounter columns being selected using dot notation `df_filtered.age`. Note that this won't work if the column name includes hyphens or any other special symbols. We will stick to providing names as strings in square brackets.

[ ]:
```

```