PandasDescribe

August 8, 2023

1 Get Summary Statistics Using Pandas describe() Method

In this exercise, we will continue exploring the data and answering some initial questions about the dataset using the `describe()` data summarization method from the `Pandas` package.

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

1.0.1 Load the Dataset

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

1.0.2 Glance at the Dataset

[3]:
```
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

1.0.3 Get the Dimensions of the Dataset

[4]:
```
df.shape
```

[4]:
```
(7000, 15)
```

1.1 Step 1: Compute Summary Statistics Using Pandas `describe()` Method

The code cell below uses the Pandas DataFrame `describe()` method to get the summary statistics of the `df` DataFrame. It saves the resulting table as a new DataFrame named `df_summ`.

[5]:
```
df_summ = df.describe()
df_summ
```

[5]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	7000.000000	7.000000e+03	7000.000000	7000.000000	7000.000000	7000.000000
mean	38.596714	1.924335e+05	10.049857	1079.000429	84.970286	40.107143
std	13.745594	1.063365e+05	2.580982	7011.160679	400.142351	12.323946
min	17.000000	1.882700e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.202478e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.821170e+05	10.000000	0.000000	0.000000	40.000000
75%	47.000000	2.402370e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.268339e+06	16.000000	99999.000000	4356.000000	99.000000

We can see that the `fnlwgt` variable is scaled very differently from others. What does this variable represent? It is always a good idea to consult the data description before analyzing your data. This variable represents a weight of a given data point, which is the number of units in the target population that the data point represents. A weight is assigned to each observation depending on to which community or subgroup the represented person belongs. Within each state, people with similar demographic characteristics should have similar weights.

Recall that Pandas `describe()` ignores all non-numerical columns. This is why your summary table contains fewer columns than the original data. To fix this, the code cell below passes the `include = 'all'` parameter to the `describe()` method, and saves the results to DataFrame `df_summ_all`.

[6]:
```
df_summ_all = df.describe(include = 'all')
df_summ_all
```

[6]:

	age	workclass	fnlwgt	education	education-num
count	7000.000000	6625	7.000000e+03	7000	7000.000000
unique	NaN	7	NaN	16	NaN

top	NaN	Private	NaN	HS-grad	NaN	
freq	NaN	4879	NaN	2263	NaN	
mean	38.596714	NaN	1.924335e+05	NaN	10.049857	
std	13.745594	NaN	1.063365e+05	NaN	2.580982	
min	17.000000	NaN	1.882700e+04	NaN	1.000000	
25%	28.000000	NaN	1.202478e+05	NaN	9.000000	
50%	37.000000	NaN	1.821170e+05	NaN	10.000000	
75%	47.000000	NaN	2.402370e+05	NaN	12.000000	
max	90.000000	NaN	1.268339e+06	NaN	16.000000	

	marital-status	occupation	relationship	race	sex_selfID	\
count	7000	6625	7000	7000	7000	
unique	7	14	6	5	2	
top	Married-civ-spouse	Prof-specialty	Husband	White	Non-Female	
freq	3277	911	2878	5990	4731	
mean	NaN	NaN	NaN	NaN	NaN	
std	NaN	NaN	NaN	NaN	NaN	
min	NaN	NaN	NaN	NaN	NaN	
25%	NaN	NaN	NaN	NaN	NaN	
50%	NaN	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	NaN	
max	NaN	NaN	NaN	NaN	NaN	

	capital-gain	capital-loss	hours-per-week	native-country	label
count	7000.000000	7000.000000	7000.000000	6862	7000
unique	NaN	NaN	NaN	40	2
top	NaN	NaN	NaN	United-States	<=50K
freq	NaN	NaN	NaN	6233	5319
mean	1079.000429	84.970286	40.107143	NaN	NaN
std	7011.160679	400.142351	12.323946	NaN	NaN
min	0.000000	0.000000	1.000000	NaN	NaN
25%	0.000000	0.000000	40.000000	NaN	NaN
50%	0.000000	0.000000	40.000000	NaN	NaN
75%	0.000000	0.000000	45.000000	NaN	NaN
max	99999.000000	4356.000000	99.000000	NaN	NaN

We could also use `describe()` to get the statistics of only a few selected columns of interest.

The idea is to first filter the DataFrame, and then call `describe()` on the filtered object.

How would you get a summary table for only the *age*, *education (numerical)*, and *hours per week* data? First, create a Python list containing relevant column names. Then, use the list to retrieve a subset of the DataFrame `df` with just these columns. Finally, apply `describe()` to that subset.

The code cell below follows these steps and saves the result to DataFrame `df_summ_selected`.

[7]:
```
describe_vars = ['age', 'education-num', 'hours-per-week']
df_summ_selected = df[describe_vars].describe()
df_summ_selected
```

[7]:

	age	education-num	hours-per-week
count	7000.000000	7000.000000	7000.000000

```
mean      38.596714      10.049857      40.107143
std       13.745594       2.580982      12.323946
min       17.000000       1.000000       1.000000
25%       28.000000       9.000000      40.000000
50%       37.000000      10.000000      40.000000
75%       47.000000      12.000000      45.000000
max       90.000000      16.000000      99.000000
```

Going forward, we will use the first summary table `df_summ` to answer some of the questions that can help us explore and understand our (numerical) data better.

1.2 Step 2: Data Analytics Using Summary Statistics

Let's print our summary data again:

[8]:
```
df_summ
```

[8]:
```
                age        fnlwgt  education-num  capital-gain  capital-loss  \
count  7000.000000  7.000000e+03    7000.000000   7000.000000   7000.000000
mean     38.596714  1.924335e+05      10.049857   1079.000429     84.970286
std      13.745594  1.063365e+05       2.580982   7011.160679    400.142351
min      17.000000  1.882700e+04       1.000000      0.000000      0.000000
25%      28.000000  1.202478e+05       9.000000      0.000000      0.000000
50%      37.000000  1.821170e+05      10.000000      0.000000      0.000000
75%      47.000000  2.402370e+05      12.000000      0.000000      0.000000
max      90.000000  1.268339e+06      16.000000  99999.000000   4356.000000

       hours-per-week
count     7000.000000
mean        40.107143
std         12.323946
min          1.000000
25%         40.000000
50%         40.000000
75%         45.000000
max         99.000000
```

1.2.1 What is the 25th percentile of feature 'age'?

The code cell below uses column and row indices to get a particular cell of the summary table that answers this initial data exploration question. Recall that you can call a value from a row named `r1` and a column named `c1` by using `loc[]`.

[9]:
```
age_25p = df_summ.loc['25%']['age']
print(f"The 25th percentile of the feature 'age' is {age_25p}")
```

```
The 25th percentile of the feature 'age' is 28.0
```

1.2.2 Which feature has the most variation?

We will need to use both `loc[]` and a new method: `idxmax()`. Consult the online documentation for more information. The method `idxmax()` retrieves the index (or a name) of the location where the maximum value in a series was found. We need to first get a vector of `std` values, and then pass it to `idxmax()` to identify the name of a column which has the maximum value. We must specify `idxmax(axis = 1)` to indicate that the search for the highest value must occur *column-wise*.

[10]:
```
df_summ.loc['std'].idxmax(axis=1)
```

[10]:
```
'fnlwgt'
```

Note: Many Pandas methods can be applied to both Series and DataFrame objects. The `idxmax()` method is one such method. Therefore, this could have been done in a different order: You can apply the `idxmax()` method to the DataFrame `df_summ` to find the name of the column that contains the max value *for all of the rows*, and then select only the row (`std`) of interest:

[11]:
```
df_summ.idxmax(axis = 1)['std']
```

[11]:
```
'fnlwgt'
```

Use the same approach as the code cell above to answer the same question for the `mean` statistic: which feature in our data has the highest mean value? Save your result to variable `column_name`. Hint: Use the same code as in the code cell above, but change the column name.

1.2.3 Graded Cell

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

[12]:
```
# YOUR CODE HERE
column_name=df_summ.idxmax(axis=1)['mean']
```

1.2.4 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.

[13]:
```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testColumnName

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

```
Correct!
```

1.2.5 Do any features have negative values?

The code cell below uses the appropriate row name, `loc[]`, and the `np.any()` function to get the `True/False` answer to the question.

[14]:
```
np.any(df_summ.loc['min'] < 0)
```

[14]:
```
False
```

1.2.6 Which feature has the highest range?

In the code cell below, write code to find the feature with the highest range. Follow the steps below: 1. Construct a vector of *differences* using `df_summ.loc[]` to find the difference between the `max` and `min` columns. Save the result to variable `column_ranges`. 2. Apply the `idxmax()` method to `column_ranges` to find the column with the maximum range. Save the result to variable `column_range_name`.

1.2.7 Graded Cell

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

[21]:
```
# YOUR CODE HERE
column_ranges=df_summ.loc['max']-df_summ.loc['min']
column_range_name=column_ranges.idxmax()
```

1.2.8 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 testRange
try:
    p, err = testRange(df, df_summ, column_ranges, column_range_name)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Correct!
```

[ ]:
```

```