Published in Towards Data Science

Ryan A. Mardani Follow
Jun 15, 2020 · 5 min read ★ · Listen

5 Steps in Pandas to Process Petrophysical Well Logs (part2)

I use these more advanced steps to process well log data in python

In the previous work, we implemented 10 simple steps using pandas to process petrophysical well logs in LAS format. In this project, we will go deeper to use more advanced approaches to process well log data.

These 5 steps are:
1) Function Definition
2) Apply Function
3) Lambda Function
4) Cut Function
5) Visualization

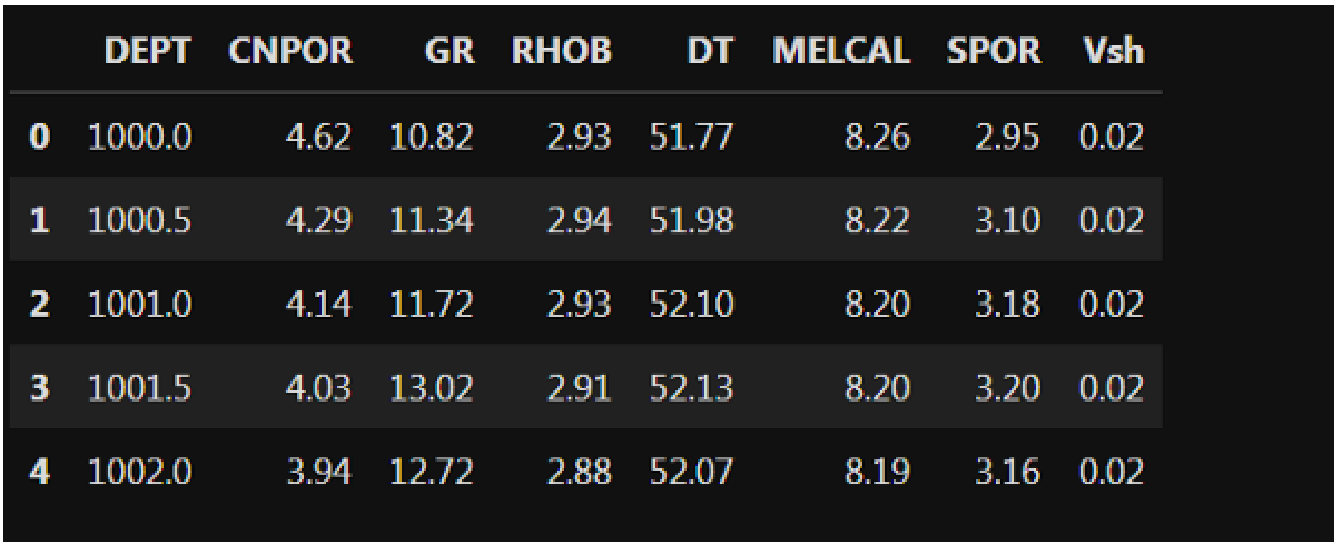
To avoid extra work that we already did on specific well data previously, we will use the output of that project. If you worked on a previous project you may write the DataFrame into csv file (use *to_csv* command) to use here. Otherwise, you can access through my github account and download the csv format called 1050383876v2.csv. You may also download the full Jupiter notebook file from my account, here.

Let's bring required libraries on the workbook first:

```
import seaborn as sns
%matplotlib inline
```

Then, read the csv data into df variable:

```
df = pd.read_csv('1050383876v2.csv')
df.head()
```



	DEPT	CNPOR	GR	RHOB	DT	MELCAL	SPOR	Vsh
0	1000.0	4.62	10.82	2.93	51.77	8.26	2.95	0.02
1	1000.5	4.29	11.34	2.94	51.98	8.22	3.10	0.02
2	1001.0	4.14	11.72	2.93	52.10	8.20	3.18	0.02
3	1001.5	4.03	13.02	2.91	52.13	8.20	3.20	0.02
4	1002.0	3.94	12.72	2.88	52.07	8.19	3.16	0.02

Dataset seems clean. I also dropped the upper part where there were some NaN values, so the starting depth is 1000 meters.

1. Function Definition

Function in python is a group of statements that perform a specific task. A function can make the larger program more readable. It also can help us to prevent repetition. A function has a special structure as:
1- keyword ***def***,
2- function name,
3- arguments that we pass values to a function,
4- a colon(:),
5- more statement in body,
6- Return statement which is optional.

wellbore conditions. Having knowledge of the fluid density (*f*) filling rock porosity and grain density(*ma*) can help us to calculate the percentage of void areas of the rocks. The equation is:

$$\Phi_D = \frac{\rho_{ma} - \rho_b}{\rho_{ma} - \rho_f}$$

Starting with *def* keyword, density porosity(*den*) function can be defined with the following arguments (parameters): *rb*, *rf*, *rm*.

```
def den(rb, rf, rm):
    # rb = bulk density from well log readings
    # rf = fluid density
    # rm = matrix density or grain density of rocks
    return (rm-rb)*100/(rm-rf)
```

Simply, the function will return the density porosity as written in the last line of code. Here we prefer to have porosity in percentage(multiplied by 100). After defining the density porosity function, we can apply it to the dataset.

2. Apply Function

Using the *apply* function, a predefined function (here, density porosity) can be applied to each element of DataFrame either in column or row direction. If we leave it on default, it will on the column.

```
df['DNPOR'] = df['RHOB'].apply(den, rf=1, rm=2.71 )
```

As *den* function takes 3 values (*rb*, *rf*, *rm*), we can introduce the main one from the dataset (*rb =df['RHOB']*) and use the apply function to define *rf* and *rm* constants manually. Here I assume that fluid content is water with a density of 1 and the dominant mineral is Calcite with a density of 2.71. The Density porosity is stored in a

Lambda function is a simple 1-line function that does not have *def* or *return* keywords. In fact, they are implicit. To use this function, we need to type lambda followed by parameters. Then, the colon comes before the return argument.

Total Porosity Calculation from Density and Neutron porosity

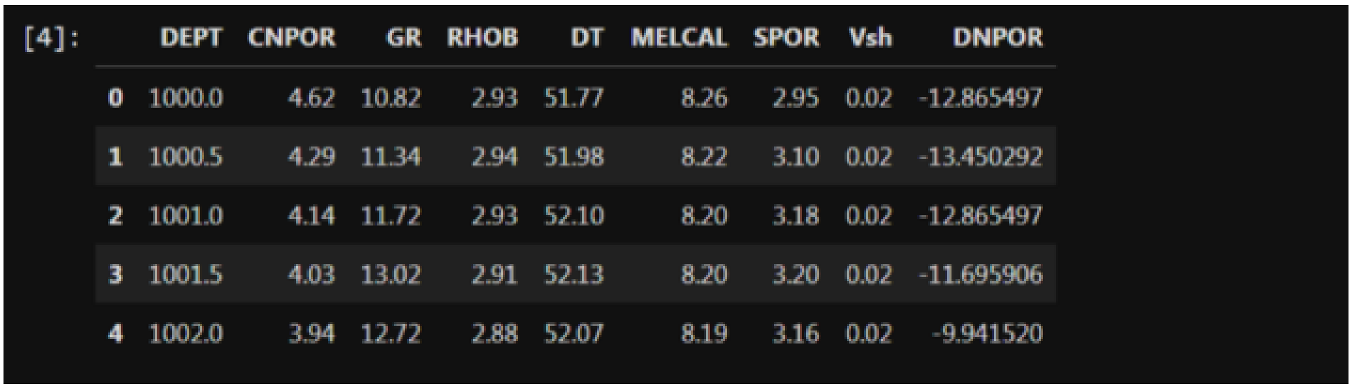
Total porosity is defined as the average of Density and Neutron porosity.

```
Tot_por = lambda DN,CN: (DN+CN)/2
```

The function name is *Tot_por*. For *lambda* function, DN and CN are density and neutron parameters followed by a colon. The average of these two inputs will be returned by the function.

```
df['TPOR'] = Tot_por(df['CNPOR'], df["DNPOR"])
df.head()
```

Calling the *Tot_por* function and introducing corresponding columns of DataFrame as input will create total porosity that will be stored in a new column called TPOR in the dataset.



[4]:

	DEPT	CNPOR	GR	RHOB	DT	MELCAL	SPOR	Vsh	DNPOR
0	1000.0	4.62	10.82	2.93	51.77	8.26	2.95	0.02	-12.865497
1	1000.5	4.29	11.34	2.94	51.98	8.22	3.10	0.02	-13.450292
2	1001.0	4.14	11.72	2.93	52.10	8.20	3.18	0.02	-12.865497
3	1001.5	4.03	13.02	2.91	52.13	8.20	3.20	0.02	-11.695906
4	1002.0	3.94	12.72	2.88	52.07	8.19	3.16	0.02	-9.941520

4. Cut Function

When we need to segment and sort data values into specific bins, we can use the Cut function. Here we will use this function to define simple rock facies based on

Facies classification is a huge topic in geoscience and various metrics can come to play but here, we look at it very simple. Based on GR reading in well logs, we can identify clean from shaly formations.

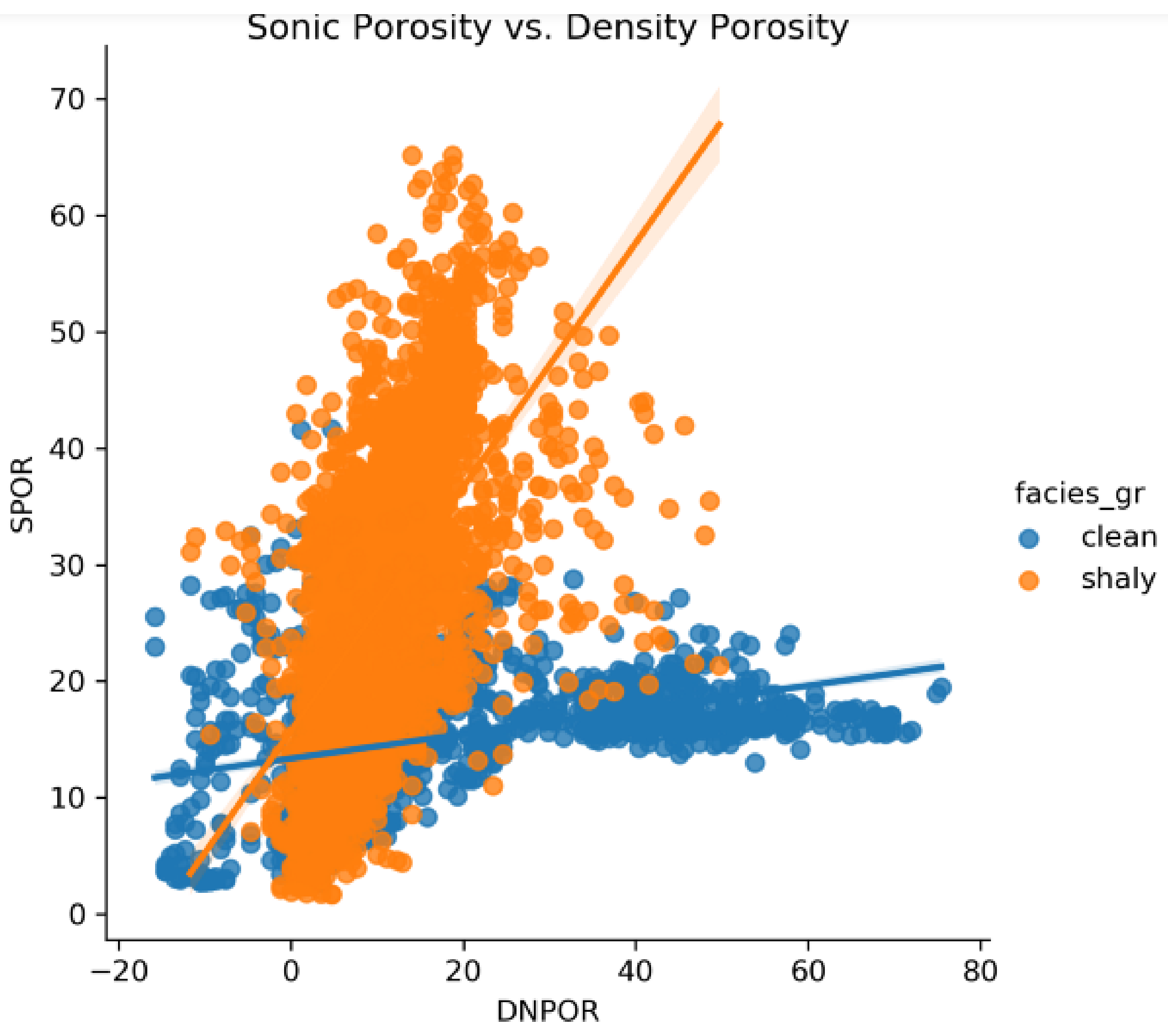
```
df['facies_gr'] = pd.cut(df['GR'], bins=[0,40,300], labels=['clean',
'shaly'] )
```

A new column is added to the dataset based on GR readings which are binned between 0 and40(clean), and between 40and 300(shaly).

5. Visualization

Commonly, the matplotlib library is used in python for plotting. In this work, I prefer to use the seaborn library for its simplicity. I want to visualize a scatter plot of density porosity vs. neutron porosity with the legend color of facies that in the previous part we defined. This is simply accessible by one line of code while in matplotlib it requires for loop.

```
sns.lmplot(x='DNPOR', y='DT', hue='facies_gr', data=df)
```



Conclusion

In this work, I have tried to use more advanced steps in pandas to process petrophysical well log data. Functions are a convenient way of using programming to avoid repetition. Either python's functions like *apply* and *cut* or self-defined functions(*den* & *Tot_por* in this work) can be useful to process well log data.

If you have any suggestions, I'm gladly open to see your comments!