Data Discretization

So far, we have done the categorical data treatment using encoding and numerical data treatment using scaling.

**Data discretization** is the process of converting continuous data into discrete buckets by grouping it. Discretization is also known for easy maintainability of the data. Training a model with discrete data becomes faster and more effective than when attempting the same with continuous data. Although continuous-valued data contains more information, huge amounts of data can slow the model down. Here, discretization can help us strike a balance between both. Some famous methods of data discretization are **binning** and using a histogram. Although data discretization is useful, we need to effectively pick the range of each bucket, which is a challenge.

The main challenge in discretization is to choose the number of intervals or bins and how to decide on their width.

Here we make use of a function called **pandas.cut()**. This function is useful to achieve the bucketing and sorting of segmented data.

Discretization of Continuous Data

In this We will load the **Student.csv** dataset and perform bucketing. The dataset consists of student details such as **Student\_id**, **Age**, **Grade**, **Employed**. Follow these steps to complete this exercise:

Open a Jupyter notebook and add a new cell. Write the following code to import the required libraries and load the dataset into a pandas dataframe:

import pandas as pd

import numpy as np

from google.colab import files

data = files.upload()

#Reading data from csv file

df = pd.read\_csv("Student.csv")

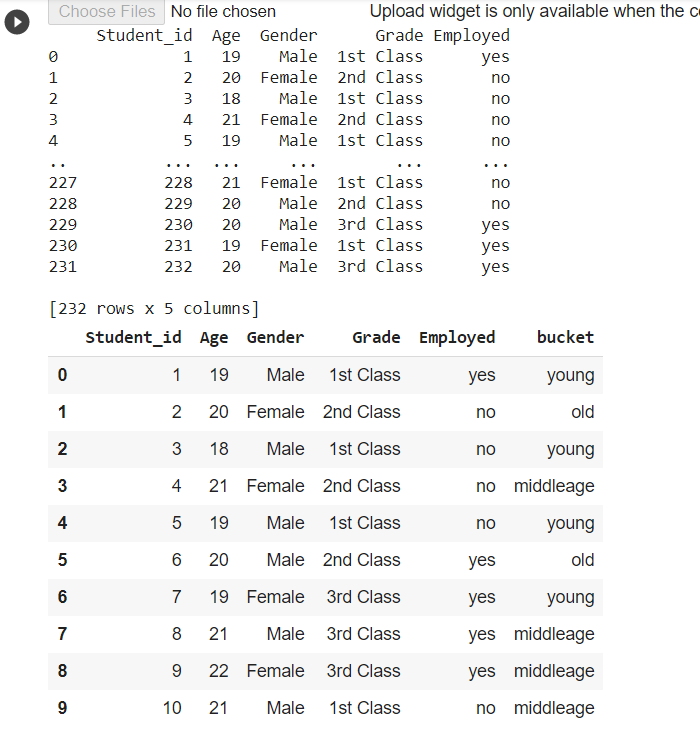
print(df)

#print("no.of rows:",len(my\_data))

df['bucket']=pd.cut(df['Age'],3,labels=['young','old','middleage'])

df.head(10)

Output:



SOURCE: <https://cmdlinetips.com/2019/12/how-to-discretize-bin-a-variable-in-python/>

**How to Discretize or Bin with Numpy’s digitize() function?**

|  |
| --- |
| # load numpy  import numpy as np  # load pandas  import pandas as pd |

Let us create a numpy array with 10 integers. We will use NumpPy’s random module to generate random numbers in between 25 and 200. We will also use random seed to reproduce the random numbers.

|  |  |
| --- | --- |
| 1  2  3  4 | # set a random seed to reproduce  np.random.seed(123)  # create 10 random integers  x = np.random.randint(low=25, high=200, size=10) |

Let us sort the numbers for convenience.

|  |  |
| --- | --- |
| 1 | x = np.sort(x) |

We can see the numbers we generated 10 numbers for height ranging from 42 to 151.

|  |  |
| --- | --- |
| 1  2 | print(x)  array([ 42,  82,  91, 108, 121, 123, 131, 134, 148, 151]) |

We can use NumPy’s digitize() function to discretize the quantitative variable. Let us consider a simple binning, where we use 50 as threshold to bin our data into two categories. One with values less than 50 are in the 0 category and the ones above 50 are in the 1 category.

We specify the threshold to digitize or discretize as a list to bins argument.

|  |  |
| --- | --- |
| 1  2 | # digitize examples  np.digitize(x,bins=[50]) |

We can see that except for the first value all are more than 50 and therefore get 1.

|  |  |
| --- | --- |
| 1 | array([0, 1, 1, 1, 1, 1, 1, 1, 1, 1]) |

The bins argument is a list and therefore we can specify multiple binning or discretizing conditions. In the example below, we bin the quantitative variable in to three categories.

|  |  |
| --- | --- |
| 1 | np.digitize(x,[50,100]) |

It gives us three categories as we wanted, 0 category with values less than 50, 1 category with value less than 100 and category 3 with more than 100.

|  |  |
| --- | --- |
| 1 | array([0, 1, 1, 2, 2, 2, 2, 2, 2, 2]) |

We can also bin/categorize/discretize the variable into multiple categories. Here is an example with four categories using digitize.

|  |  |
| --- | --- |
| 1  2 | np.digitize(x,[25,50,100])  array([1, 2, 2, 3, 3, 3, 3, 3, 3, 3]) |

**How to Discretize or Bin with Pandas cut() function?**

Now let us use Pandas cut function to discretize/categorize a quantitative variable and produce the same results as NumPy’s digitize function.

Pandas cut function is a powerful function for categorize a quantitative variable. The way it works is bit different from NumPy’s digitize function.

Let us first make a Pandas data frame with height variable using the random number we generated above.

|  |  |
| --- | --- |
| **Output:** | df = pd.DataFrame({"height":x})  df.head()         height  0   42  1   82  2   91  3   108  4   121 |

Let us categorize the height variable into four categories using Pandas cut function. Pandas cut function takes the variable that we want to bin/categorize as input. In addition to that, we need to specify bins such that height values between 0 and 25 are in one category, values between 25 and 50 are in second category and so on.

|  |  |
| --- | --- |
|  | df['binned']=pd.cut(x=df['height'], bins=[0,25,50,100,200]) |

Let us save the binned variable as another variable in the original dataframe. When we apply Pandas’ cut function, by default it creates binned values with interval as categorical variable. Check the type of each Pandas variable using df.dtypes.

Note how we specify the bins with Pandas cut, we need to specify both lower and upper end of the bins for categorizing.

|  |  |
| --- | --- |
|  | df.head()  **Output:**     height      binned       42    (25, 50]      82   (50, 100]       91   (50, 100]       108  (100, 200]      121  (100, 200] |

**Pandas Cut Example**

Let us see another Pandas cut example, but this time let us specify labels for each categorical variable that Pandas cut provides. We can specify the labels or the names of the categorical group we want using the argument “labels”.

In this Pandas cut example, we provide the labels as integers. Since we want to have four bins or categories, we provide the bin labels as [0,1,2,3].

|  |  |
| --- | --- |
|  | df['height\_bin']=pd.cut(x = f['height'],   bins = [0,25,50,100,200],   labels = [0, 1, 2,3])  df |

We save the new bins for height as a variable and it perfectly matches with our Numpy’s digitize example above.

**Output**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | height binned  height\_bin  0   42  (25, 50]    1  1   82  (50, 100]   2  2   91  (50, 100]   2  3   108 (100, 200]  3  4   121 (100, 200]  3 |

In the above Pandas cut example, we used integers as labels. However, we can use more descriptive categories like this as well

|  |  |
| --- | --- |
|  | df['height\_bin']=pd.cut(x=df['height'], bins=[0,25,50,100,200],                          labels=["very short", " short", "medium","tall"])  print(df.head())  **Output**     height      binned height\_bin  0      42    (25, 50]      short  1      82   (50, 100]     medium  2      91   (50, 100]     medium  3     108  (100, 200]       tall  4     121  (100, 200]       tall |