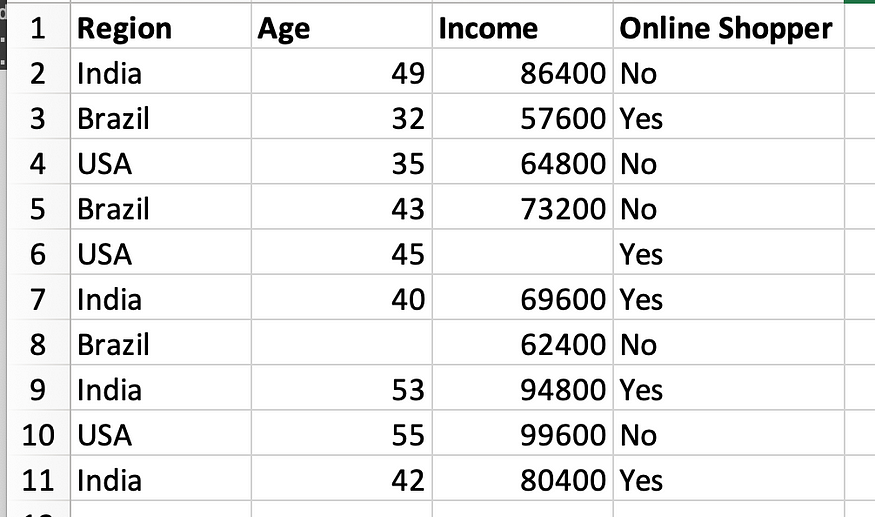
|  |  |
| --- | --- |
| DATE | 9MAY2023 |
| TEAM ID | NM2023TMID06904 |
| PROJECT | PYTHON CODE |

Preprocess Dataset

==================================

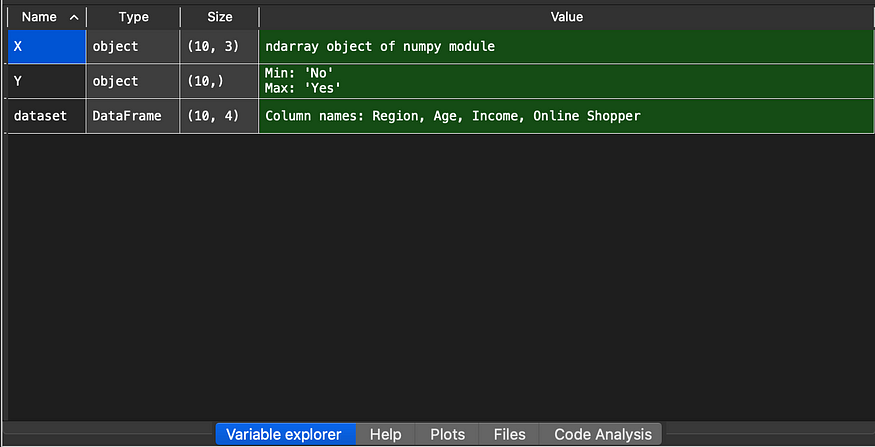
First of all, let us have a look at the dataset we are going to use for this particular example.



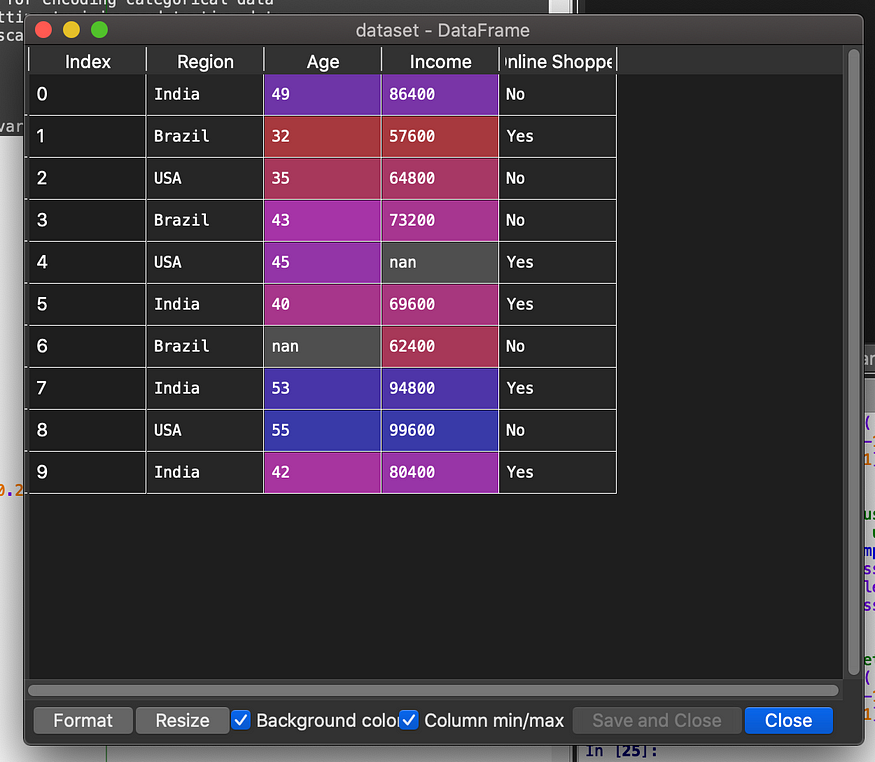
In order to import this dataset into our script, we are apparently going to use pandas as follows.

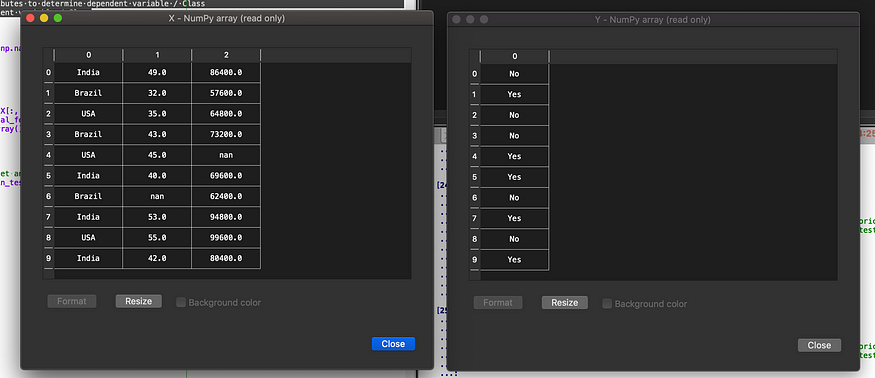
dataset = pd.read\_csv('Data.csv') # to import the dataset into a   
variable# Splitting the attributes into independent and dependent attributes  
X = dataset.iloc[:, :-1].values # attributes to determine dependent variable / Class  
Y = dataset.iloc[:, -1].values # dependent variable / Class

When you run this code section, you should not see any errors, if you do make sure the script and the *Data.csv*are in the same folder. When successfully executed, you can move to variable explorer in the Spyder UI and you will see the following three variables.



When you double click on each of these variables, you should see something similar.





If you face any errors in order to see these data variables, try to upgrade Spyder to Spyder version 4.

Build the ANN Model

==================================

# Import python libraries required in this example:

import numpy as np

from scipy.special import expit as activation\_function

from scipy.stats import truncnorm

# DEFINE THE NETWORK

# Generate random numbers within a truncated (bounded)

# normal distribution:

def truncated\_normal(mean=0, sd=1, low=0, upp=10):

   return truncnorm(

       (low - mean) / sd, (upp - mean) / sd, loc=mean, scale=sd)

# Create the ‘Nnetwork’ class and define its arguments:

# Set the number of neurons/nodes for each layer

# and initialize the weight matrices:

class Nnetwork:

   def \_\_init\_\_(self,

                no\_of\_in\_nodes,

                no\_of\_out\_nodes,

                no\_of\_hidden\_nodes,

                learning\_rate):

       self.no\_of\_in\_nodes = no\_of\_in\_nodes

       self.no\_of\_out\_nodes = no\_of\_out\_nodes

       self.no\_of\_hidden\_nodes = no\_of\_hidden\_nodes

       self.learning\_rate = learning\_rate

       self.create\_weight\_matrices()

   def create\_weight\_matrices(self):

       """ A method to initialize the weight matrices of the neural network"""

       rad = 1 / np.sqrt(self.no\_of\_in\_nodes)

       X = truncated\_normal(mean=0, sd=1, low=-rad, upp=rad)

       self.weights\_in\_hidden = X.rvs((self.no\_of\_hidden\_nodes,

                                      self.no\_of\_in\_nodes))

       rad = 1 / np.sqrt(self.no\_of\_hidden\_nodes)

       X = truncated\_normal(mean=0, sd=1, low=-rad, upp=rad)

       self.weights\_hidden\_out = X.rvs((self.no\_of\_out\_nodes,

                                       self.no\_of\_hidden\_nodes))

def train(self, input\_vector, target\_vector):

       pass # More work is needed to train the network

   def run(self, input\_vector):

       """

       running the network with an input vector 'input\_vector'.

       'input\_vector' can be tuple, list or ndarray

       """

       # Turn the input vector into a column vector:

       input\_vector = np.array(input\_vector, ndmin=2).T

       # activation\_function() implements the expit function,

       # which is an implementation of the sigmoid function:

       input\_hidden = activation\_function(self.weights\_in\_hidden @   input\_vector)

       output\_vector = activation\_function(self.weights\_hidden\_out @ input\_hidden)

       return output\_vector

# RUN THE NETWORK AND GET A RESULT

# Initialize an instance of the class:

simple\_network = Nnetwork(no\_of\_in\_nodes=2,

                              no\_of\_out\_nodes=2,

                              no\_of\_hidden\_nodes=4,

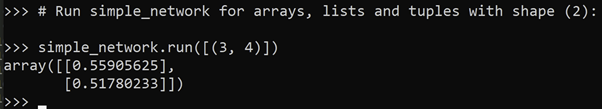
                              learning\_rate=0.6)

# Run simple\_network for arrays, lists and tuples with shape (2):

# and get a result:

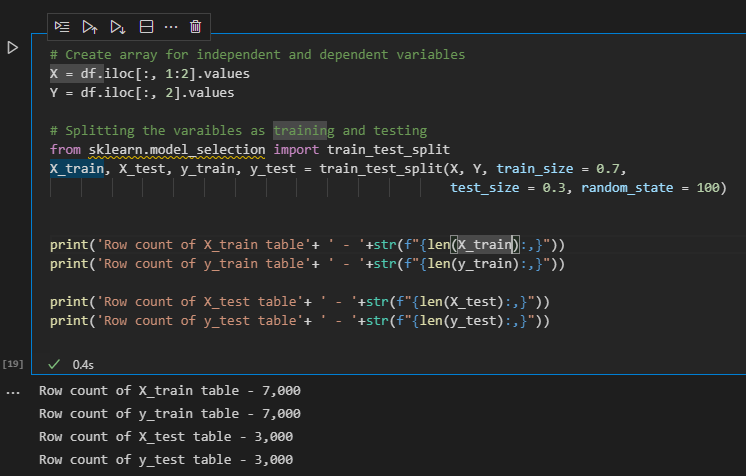
simple\_network.run([(3, 4)])

Output :



Test The Model :

==================================



We are first splitting the dataset into train and test data sets. We will build the model based on the train data set and test it on the test dataset.