Data Analysis

The nonprofit foundation Alphabet Soup wants a tool that can help it select the applicants for funding with the best chance of success in their ventures. For this challenge, we are tasked to use our knowledge of machine learning and neural networks to create a binary classifier from the features in the provided dataset that can predict whether applicants will be successful if funded by Alphabet Soup.

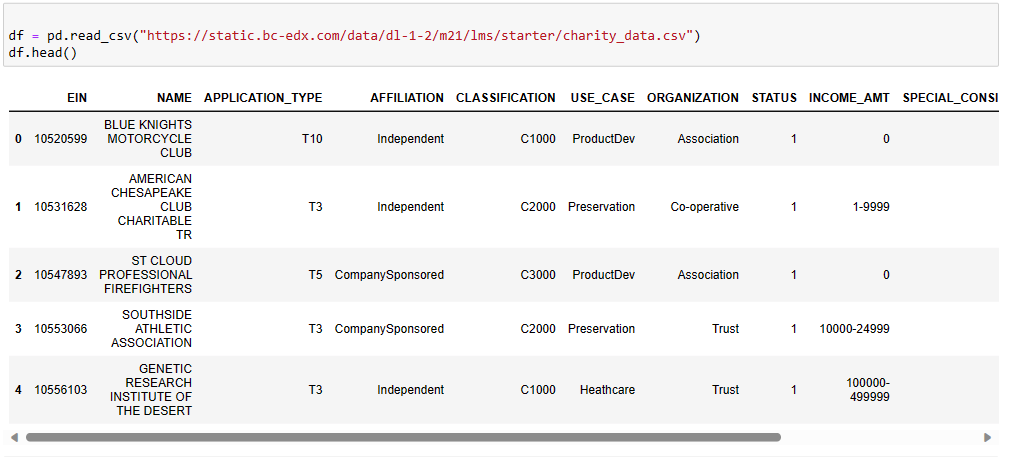
The dataset is in CSV format file and contains more than 34,000 organizations that have received funding from Alphabet Soup over the years. Within this dataset are a number of columns that capture metadata about each organization, such as:

* **EIN** and **NAME**—Identification columns
* **APPLICATION\_TYPE**—Alphabet Soup application type
* **AFFILIATION**—Affiliated sector of industry
* **CLASSIFICATION**—Government organization classification
* **USE\_CASE**—Use case for funding
* **ORGANIZATION**—Organization type
* **STATUS**—Active status
* **INCOME\_AMT**—Income classification
* **SPECIAL\_CONSIDERATIONS**—Special considerations for application
* **ASK\_AMT**—Funding amount requested
* **IS\_SUCCESSFUL**—Was the money used effectively

For this challenge, I used Google Colab instead of Jupyter Notebook.

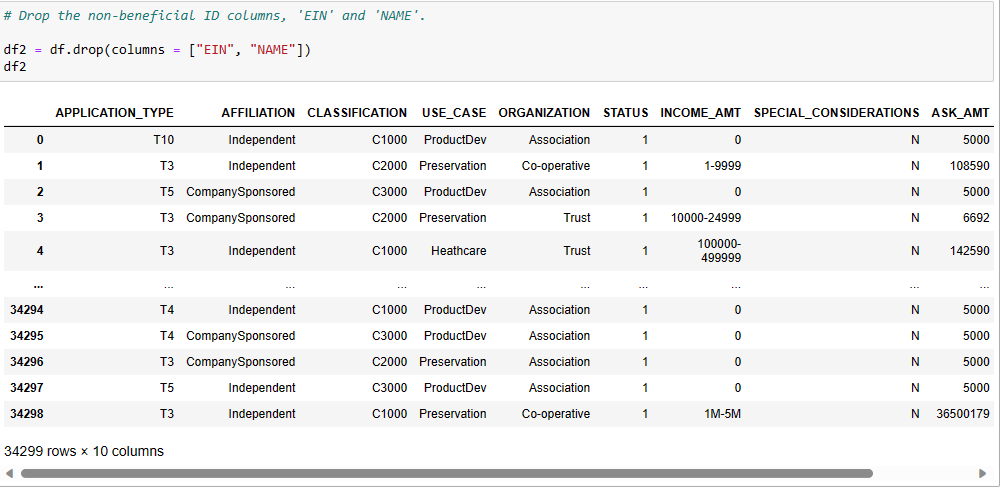
Step 1: Preprocess the Data

1. Read in the charity\_data.csv to a Pandas DataFrame, and be sure to identify the following in your dataset:

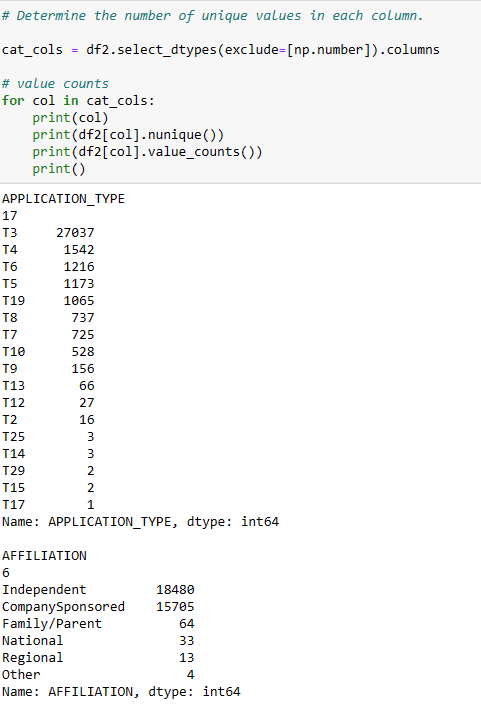


* What variable(s) are the target(s) for your model? IS\_SUCCESSFUL
* What variable(s) are the feature(s) for your model? APPLICATION TYPE\_TYPE, AFFILIATION, CLASSIFICATION, USE\_CASE, ORGANIZATION, STATUS, INCOME\_AMT, SPECIAL\_CONSIDERATIONS, ASK\_AMT.

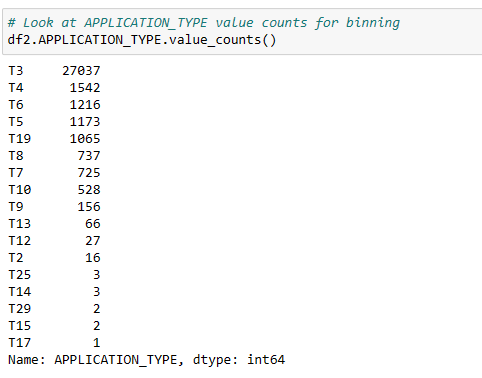
1. Drop the EIN and NAME columns.

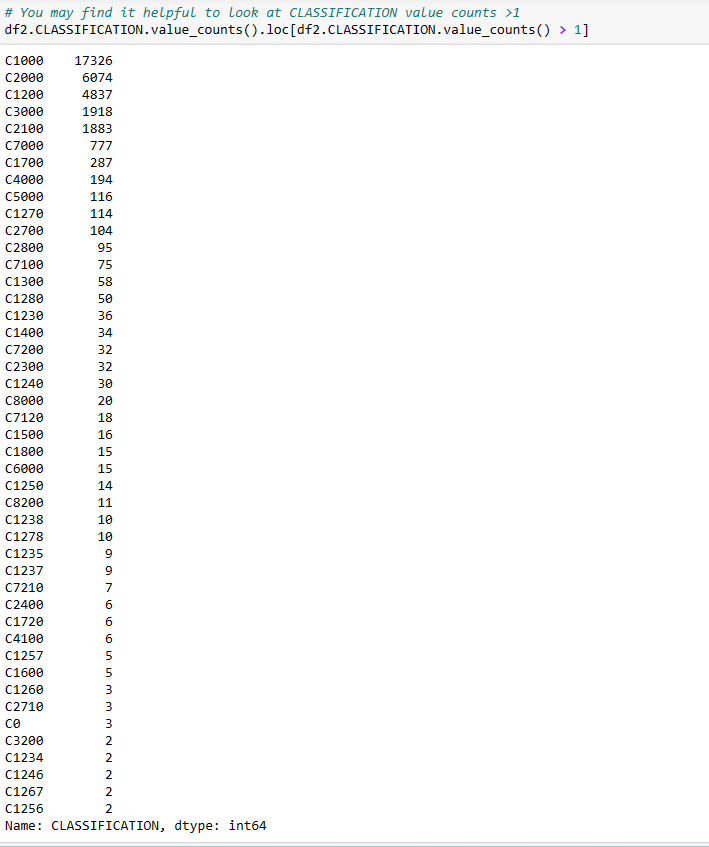


1. Determine the number of unique values for each column.

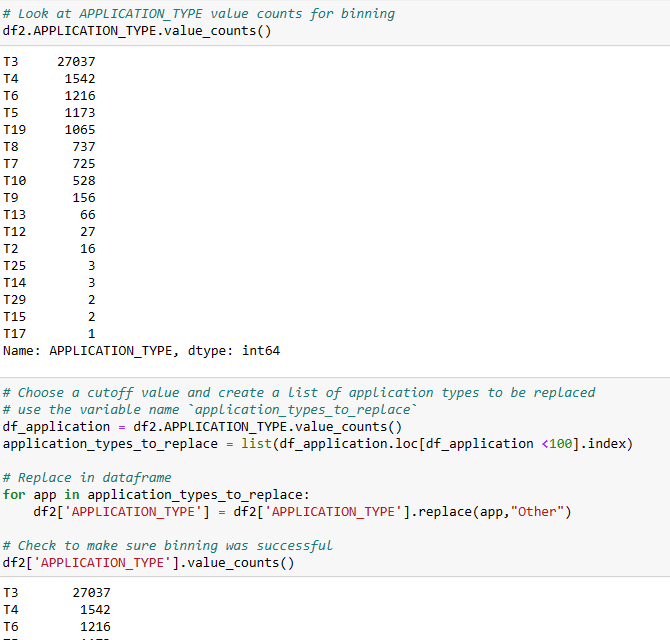


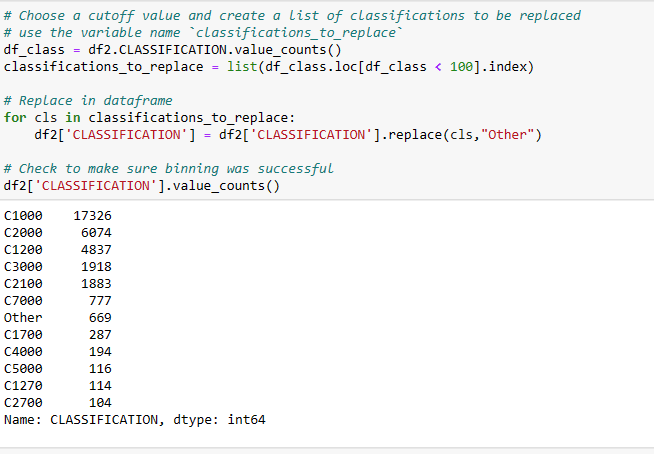
1. For columns that have more than 10 unique values, determine the number of data points for each unique value.
   * Application and Classification have more than 10 unique values.



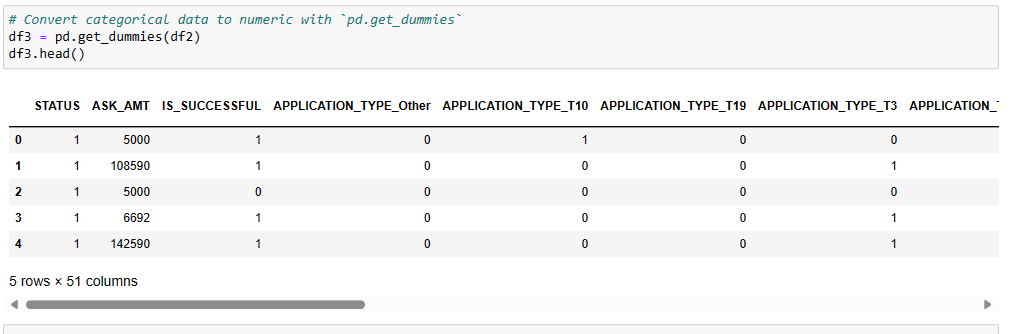


1. Use the number of data points for each unique value to pick a cutoff point to bin "rare" categorical variables together in a new value, Other, and then check if the binning was successful.





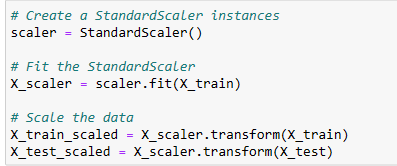
1. Use pd.get\_dummies() to encode categorical variables.



1. Split the preprocessed data into a features array, X, and a target array, y. Use these arrays and the train\_test\_split function to split the data into training and testing datasets.

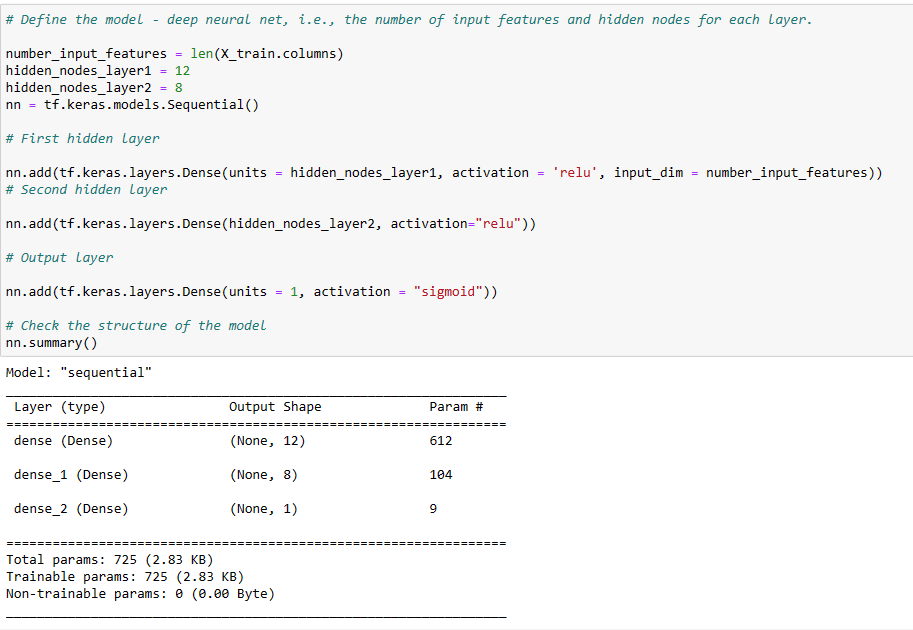


1. Scale the training and testing features datasets by creating a StandardScaler instance, fitting it to the training data, and then using the transform function.

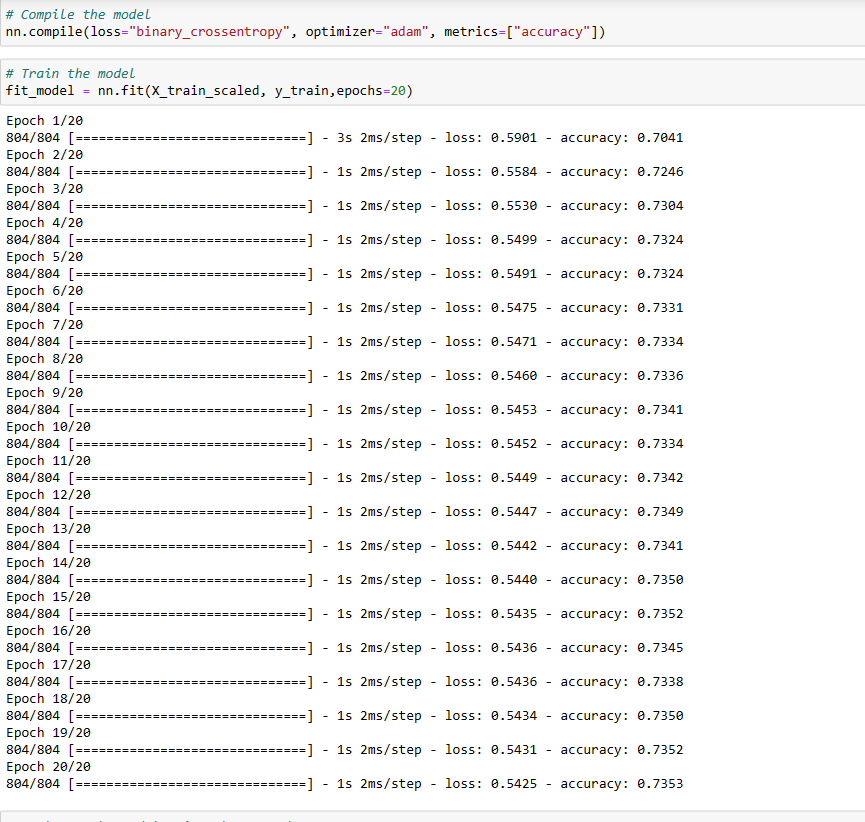


Step 2: Compile, Train, and Evaluate the Model.

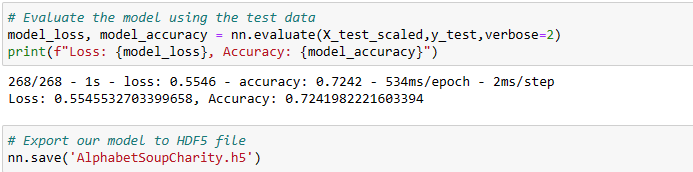
* Create a neural network model with a defined number of input features and nodes for each layer. Create hidden layers and an output layer with appropriate activation functions. Check the structure of the model.



* Compile the model / train the model



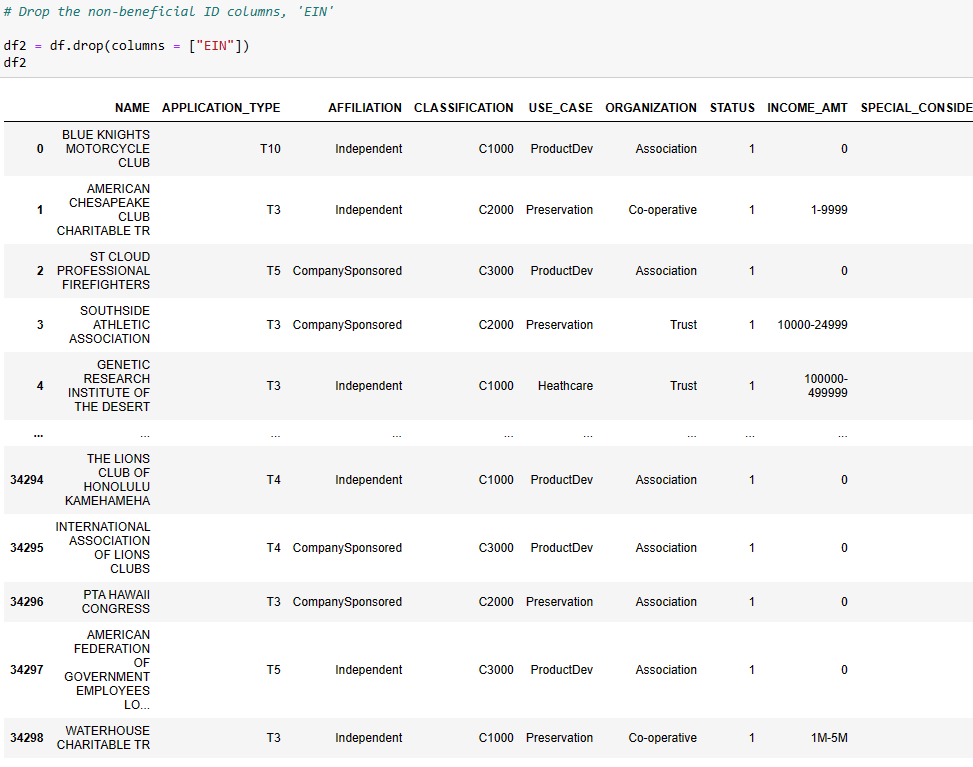
* Evaluate the model using the test data to determine the loss and accuracy, then export our model to HDF5 file.



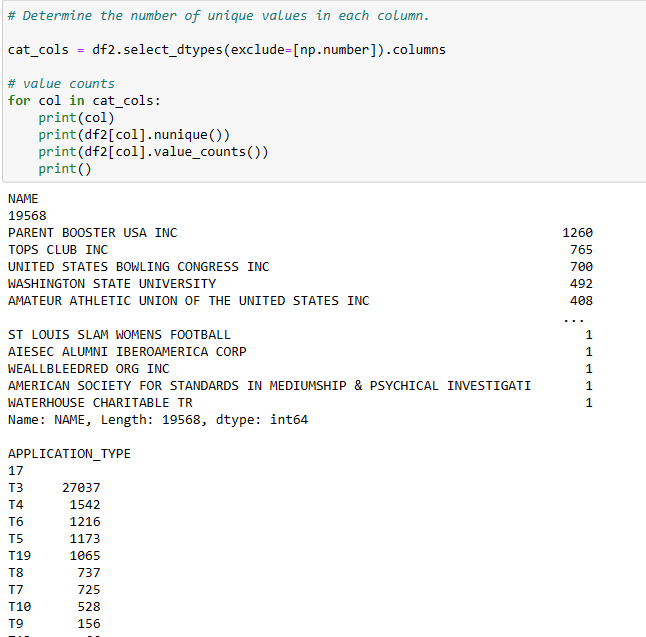
Step 3 – Optimize the model

Another ibynb file was created to optimize the dataset and then will run the model again to see if it would improve the accuracy score.

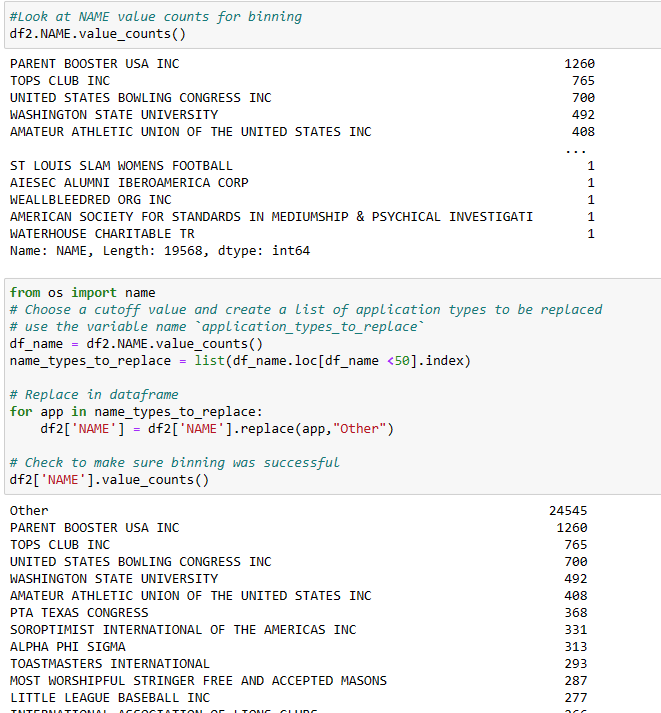
We keep the same steps 1-4 as the original file, then on step 5, instead of dropping ‘Name” and “EIN”, only EIN was dropped.



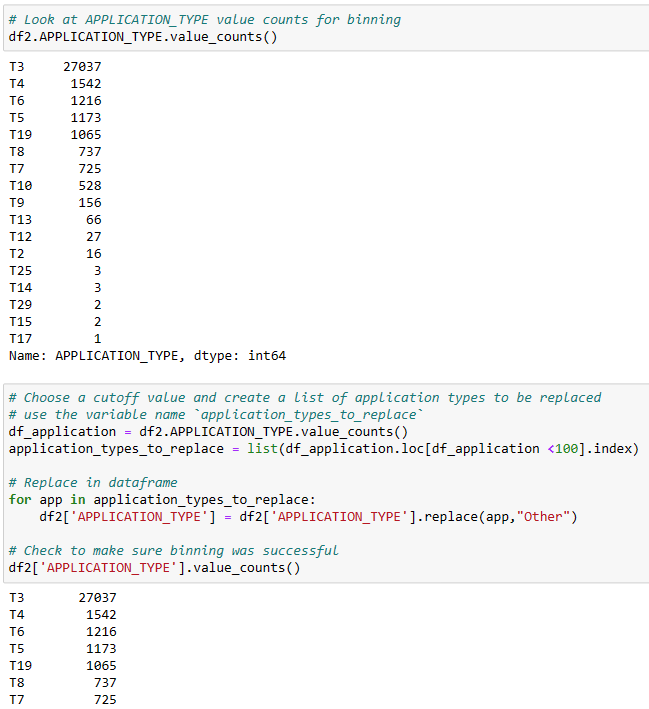
* Determine the number of unique values in each columns, “Name”, “Application”, and “Classification” came up with more than 10 unique values

****

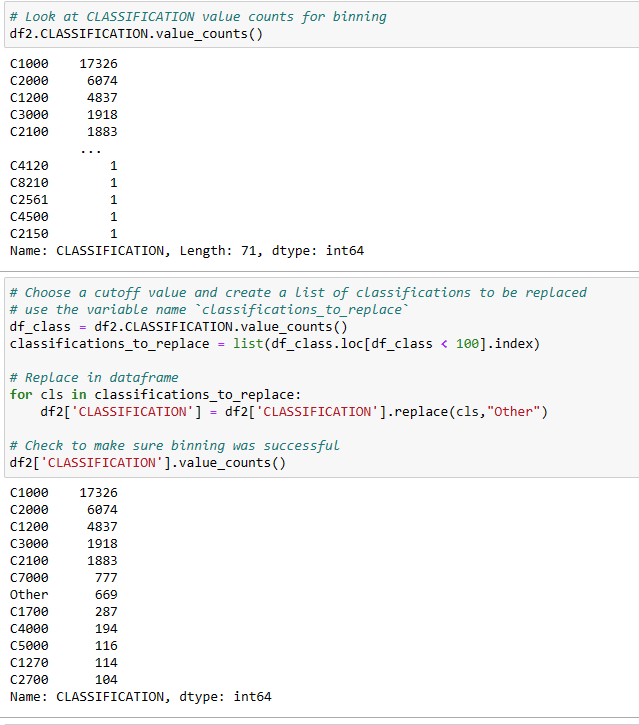
* Look at “NAME” value counts for binning, then pick a cutoff point to bin “rare” categorical values together to a new value “Other”.



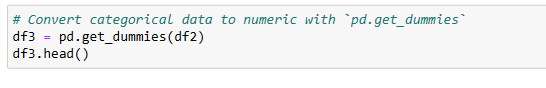
* Look at “APPLICATION” value counts for binning, and then pick a cutoff point to bin “rare” categorical values together to a new value “Other”.



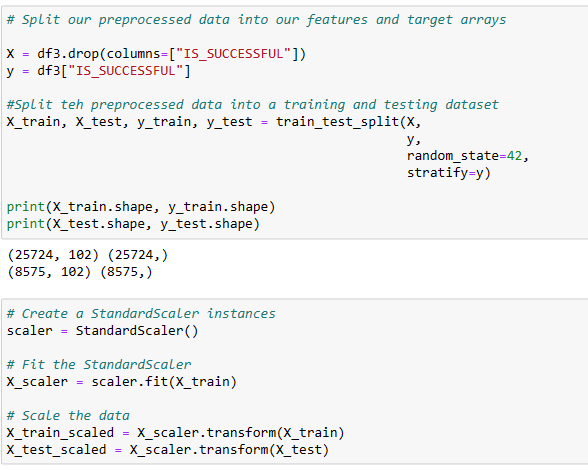
* Look at “CLASSIFICATION” value counts for binning, and then pick a cutoff point to bin “rare” categorical values together to a new value “Other”.

****

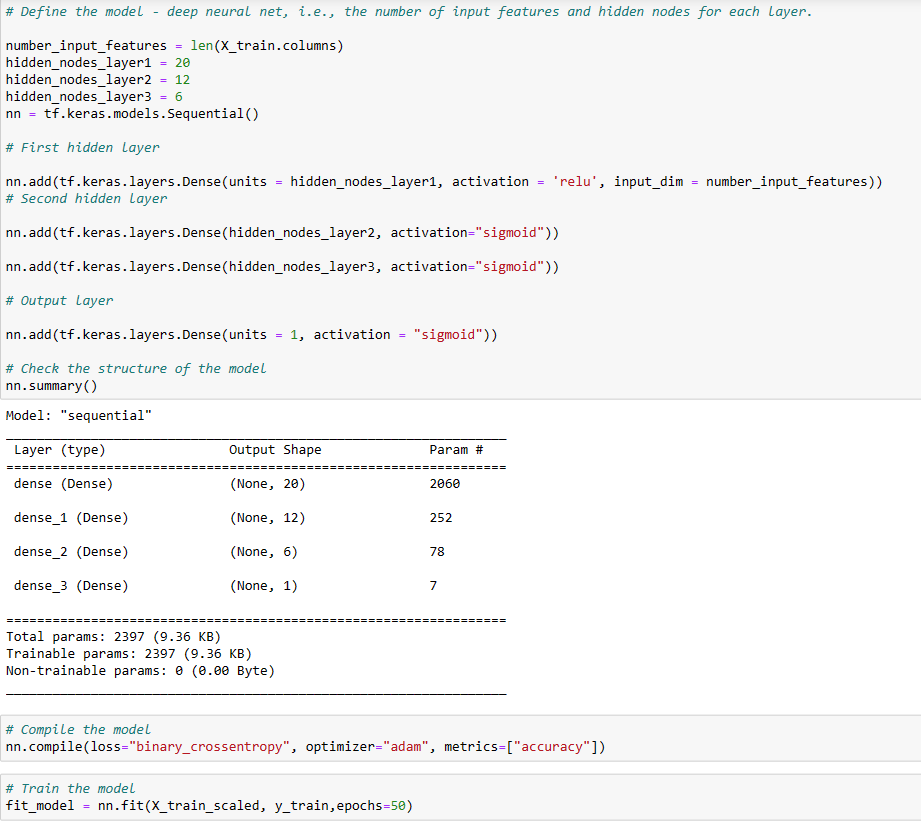
* Use pd.get\_dummies() to encode categorical values

****

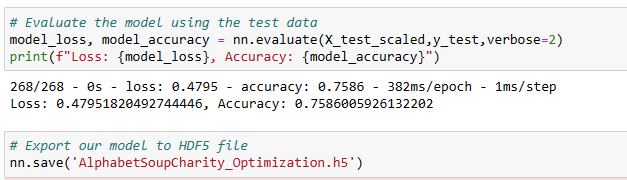
* Split our preprocessed data into our features and target arrays/Split the preprocessed data into a training and testing dataset/Create a StandardScaler instances/Fit the StandardScaler/Scale the data.

****

* Compile/Train/and Evaluate the Optimized model

****

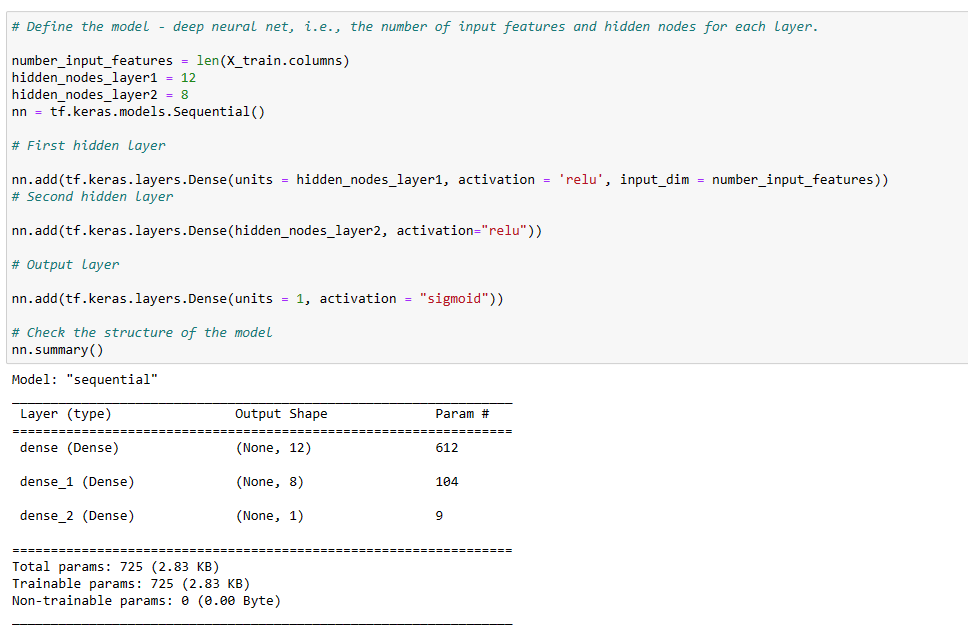
* Evaluate the model using the test data/Export our model to HDF5 file

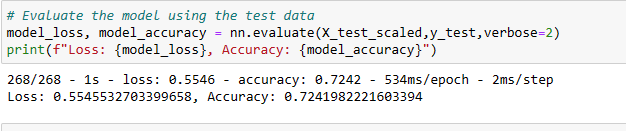
****

1. **Results**: Using bulleted lists and images to support your answers, address the following questions:

* Data Preprocessing
  + What variable(s) are the target(s) for your model? IS\_SUCCESSFUL

What variable(s) are the features for your model? APPLICATION\_TYPE, AFFILIATION, CLASSIFICATION, USE\_CASE, ORGANIZATION, STATUS, INCOME\_AMT, SPECIAL\_CONSIDERATIONS, ASK\_AMT

* + What variable(s) should be removed from the input data because they are neither targets nor features? EIN, NAME
* Compiling, Training, and Evaluating the Model
  + How many neurons, layers, and activation functions did you select for your neural network model, and why? 
  + Were you able to achieve the target model performance? The goal was 75%, but only 72% were reached with this model.



* + What steps did you take in your attempts to increase model performance?
  + I didn’t remove the Name column, added another hidden nodes layer3. I also increase the number of input and change the epochs from 20 to 50. The accuracy turned out to be 75.9%, achieved the goal of 75%. I would recommend to using Keras Tuner library to try to further optimize your model and get better accuracy score.

