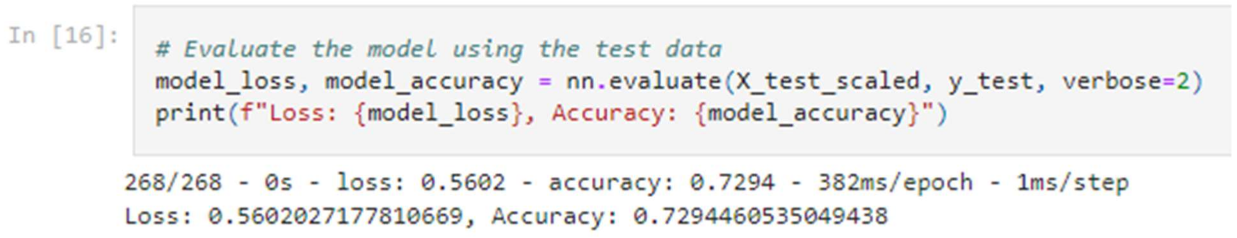
Alphabet Soup Neural Network Report

The purpose of this report is to analyze the performance of the neural network model created for the Alphabet Soup dataset with the given initial settings, discuss the optimization process, and see how significant of an improvement was achieved. The target of this model is accuracy, ideally achieving an accuracy above 75%. The most important variable in the dataset is the self-explanatory 'IS_SUCCESSFUL' column, which the model is attempting to predict. The 'EIN' and 'NAME' columns were removed from the dataset, as they were deemed non-beneficial.

The initial settings for the model consisted of two hidden layers with 80 and 30 nodes, respectively, each with 'relu' activation. The model ran for 100 epochs with the 'adam' optimizer and 'binary_crossentropy' for loss, which resulted in an accuracy of 72.94%.

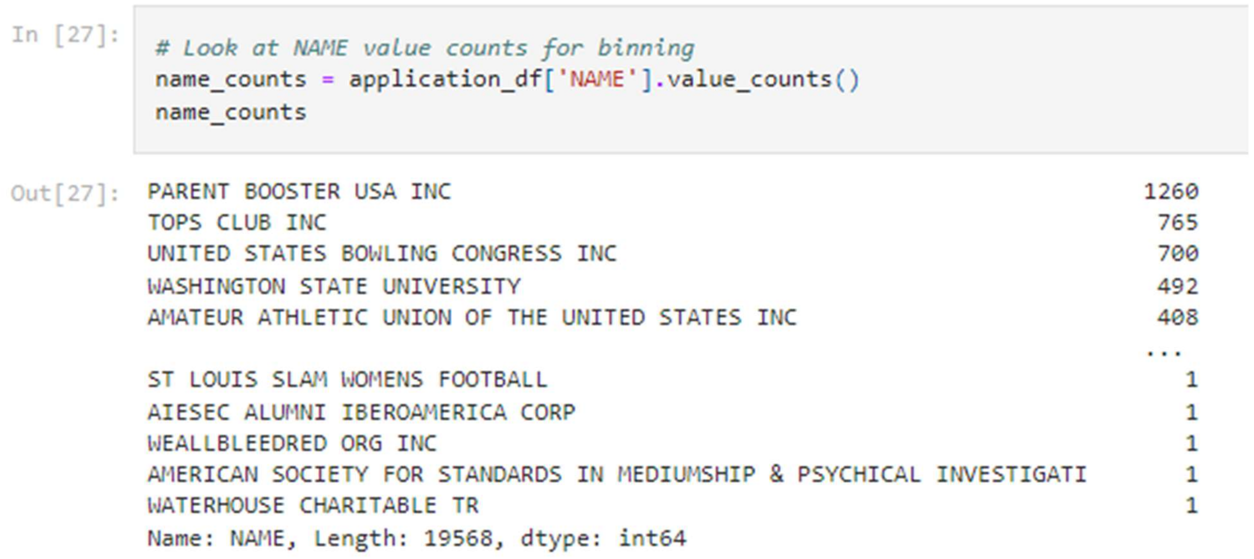


```
In [16]:
# Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test_scaled, y_test, verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

268/268 - 0s - loss: 0.5602 - accuracy: 0.7294 - 382ms/epoch - 1ms/step
Loss: 0.5602027177810669, Accuracy: 0.7294460535049438
```

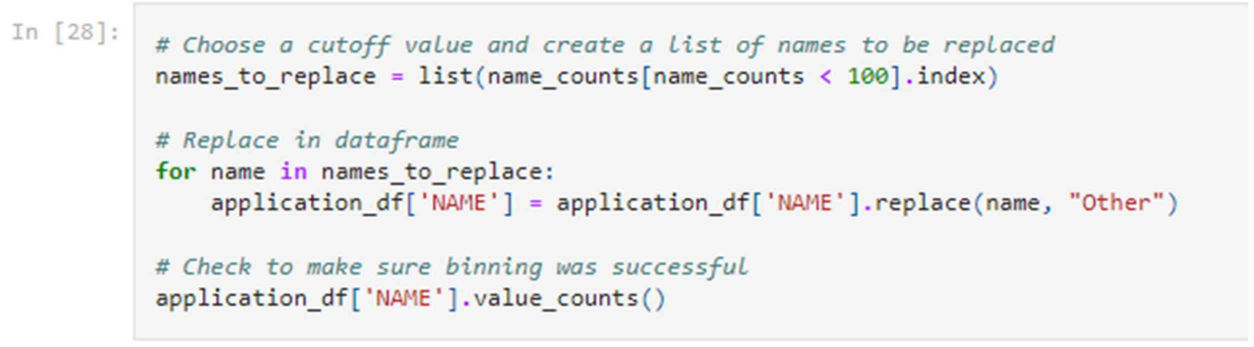
The first few attempts to increase the accuracy were relatively unsuccessful. First the number of nodes in the second layer was increased from 30 to 40, resulting in a minor accuracy increase to about 74%. The number of epochs was reduced from 100 to 50, as in most cases the model had essentially converged by only 10-20 epochs. Adding a third hidden layer was unhelpful, as the accuracy remained near 74% regardless of the number of nodes. Next the cutoff values for application type and classification binning were tweaked, but the accuracy remained at 74%. Removing additional columns from the dataset that seemed inconsequential, like 'SPECIAL_CONSIDERATIONS', actually decreased the accuracy.

The solution was to reintroduce the 'NAME' column; rather than removing it, it was binned like the application type and classification.



```
In [27]:  # Look at NAME value counts for binning
          name_counts = application_df['NAME'].value_counts()
          name_counts

Out[27]:  PARENT BOOSTER USA INC                                                  1260
          TOPS CLUB INC                                                            765
          UNITED STATES BOWLING CONGRESS INC                                       700
          WASHINGTON STATE UNIVERSITY                                              492
          AMATEUR ATHLETIC UNION OF THE UNITED STATES INC                          408
                                                                                  ...
          ST LOUIS SLAM WOMENS FOOTBALL                                              1
          AIESEC ALUMNI IBEROAMERICA CORP                                            1
          WEALLBLEEDRED ORG INC                                                      1
          AMERICAN SOCIETY FOR STANDARDS IN MEDIUMSHIP & PSYCHICAL INVESTIGATI       1
          WATERHOUSE CHARITABLE TR                                                   1
          Name: NAME, Length: 19568, dtype: int64
```

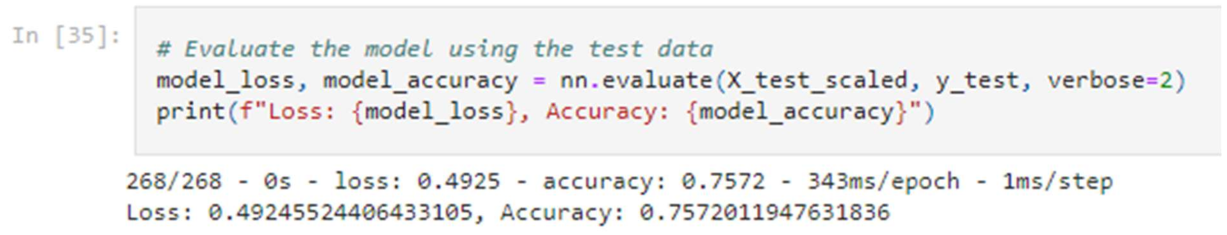


```
In [28]:  # Choose a cutoff value and create a list of names to be replaced
          names_to_replace = list(name_counts[name_counts < 100].index)

          # Replace in dataframe
          for name in names_to_replace:
              application_df['NAME'] = application_df['NAME'].replace(name, "Other")

          # Check to make sure binning was successful
          application_df['NAME'].value_counts()
```

This alone increased the trainable parameters from 6381 to 8701. The model was then run using the same settings as the unoptimized model, resulting in an accuracy of 75.72%. The model could likely be optimized further, but with this we have met our accuracy goal.



```
In [35]:  # Evaluate the model using the test data
          model_loss, model_accuracy = nn.evaluate(X_test_scaled, y_test, verbose=2)
          print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

          268/268 - 0s - loss: 0.4925 - accuracy: 0.7572 - 343ms/epoch - 1ms/step
          Loss: 0.49245524406433105, Accuracy: 0.7572011947631836
```