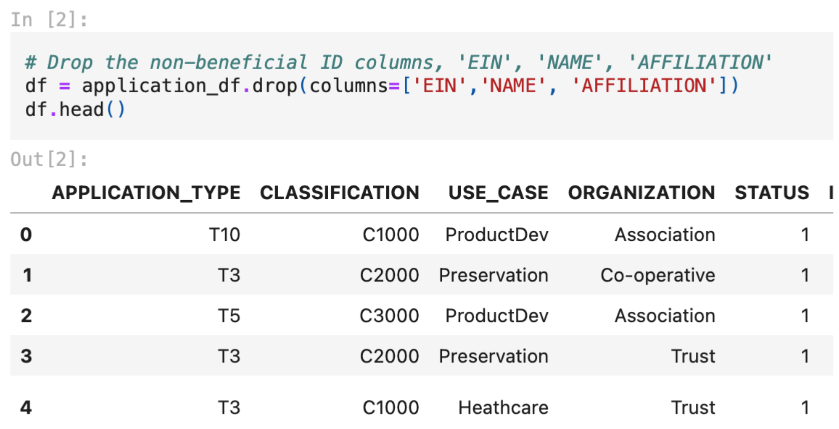
## OVERVIEW

This analysis is meant to predict the success of prospective applicants for funding from the nonprofit foundation Alphabet Soup.

## RESULTS

- What variable(s) are the target(s) for your model?
  - My model's target was the 'IS_SUCCESSFUL' column.
- What variable(s) are the features for your model?
  - My model's target was the rest of the dataframe omitting the 'IS_SUCCESSFUL' column.
- What variable(s) should be removed from the input data because they are neither targets nor features?
  - I decided to remove EIN and NAME columns, as they are identification columns and are thus neither targets nor features. I also decided to remove the AFFILIATION column as it has a similar categorical function as both the EIN and NAME columns.



```
In [2]:

# Drop the non-beneficial ID columns, 'EIN', 'NAME', 'AFFILIATION'
df = application_df.drop(columns=['EIN','NAME', 'AFFILIATION'])
df.head()

Out[2]:
```

| | APPLICATION_TYPE | CLASSIFICATION | USE_CASE | ORGANIZATION | STATUS | I |
|---|---|---|---|---|---|---|
| **0** | T10 | C1000 | ProductDev | Association | 1 | |
| **1** | T3 | C2000 | Preservation | Co-operative | 1 | |
| **2** | T5 | C3000 | ProductDev | Association | 1 | |
| **3** | T3 | C2000 | Preservation | Trust | 1 | |
| **4** | T3 | C1000 | Heathcare | Trust | 1 | |

- Compiling, Training, and Evaluating the Model
  - How many neurons, layers, and activation functions did you select for your neural network model, and why?
    - I tested 4 layers, 1 layer, and 3 layers while optimizing my model, and I found that it made no difference adding the layers. I used ReLu activation functions for my model so that my model could learn more complex representations of data, and I compiled it using binary cross entropy.
- Were you able to achieve the target model performance?

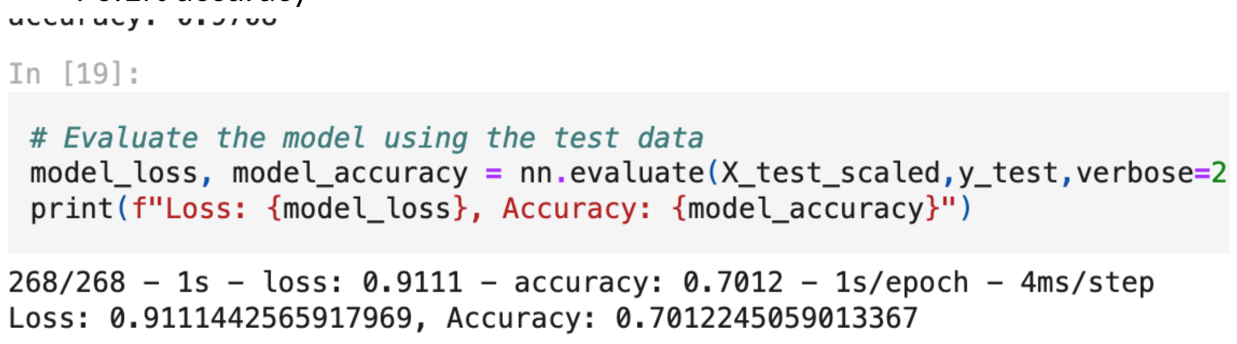
    - After three attempts, I was stuck at around a 60%-70% accuracy for my model, but I found that my highest accuracy score was the result in a decline of hidden layers. Using only one hidden layer, I was able to achieve a 72.1% accuracy, which isn't a significant difference from my other results, but it is an improvement.
- What steps did you take in your attempts to increase model performance?
    - I tried to change the number of layers and the number of neurons in my model, but these attempts all produced similar results of around a 70% accuracy rate.

## SUMMARY

The overall results of my model suggest a need for further optimization. At its current state, it is unfit to accurately predict the features right now, but I would recommend perhaps a further crack down on the number of columns, more layers, and a different activation function in order to have an optimal performance.

Attempt 1: 4 hidden nodes layers

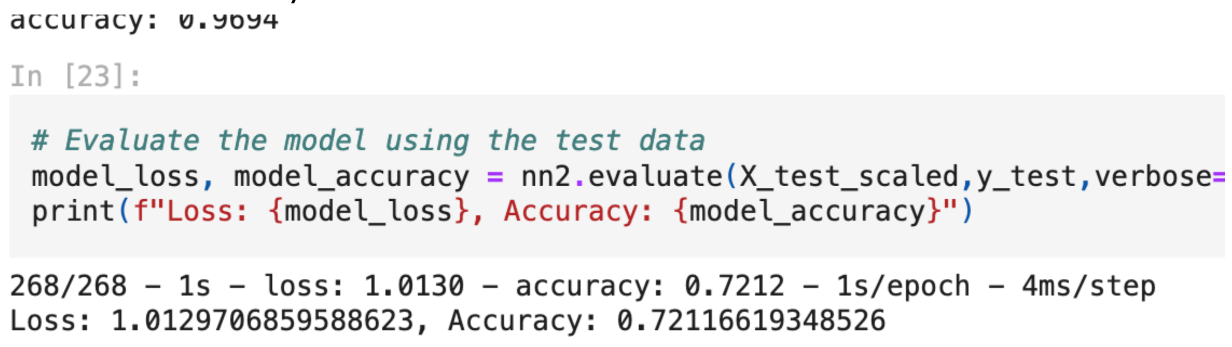
- 70.1% accuracy



```
accuracy: 0.9708

In [19]:

# 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 - 1s - loss: 0.9111 - accuracy: 0.7012 - 1s/epoch - 4ms/step
Loss: 0.9111442565917969, Accuracy: 0.7012245059013367
```

Attempt 2: 1 hidden nodes layer

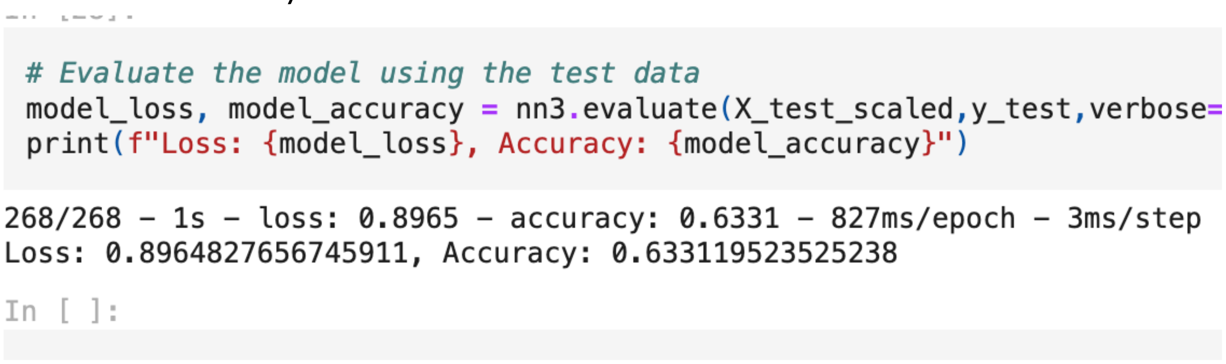
- 72.1% accuracy



```
accuracy: 0.9694

In [23]:

# Evaluate the model using the test data
model_loss, model_accuracy = nn2.evaluate(X_test_scaled,y_test,verbose=
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

268/268 - 1s - loss: 1.0130 - accuracy: 0.7212 - 1s/epoch - 4ms/step
Loss: 1.0129706859588623, Accuracy: 0.72116619348526
```

Attempt 3: 3 hidden nodes layers

- 63.3% accuracy



```
# Evaluate the model using the test data
model_loss, model_accuracy = nn3.evaluate(X_test_scaled,y_test,verbose=
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 1s - loss: 0.8965 - accuracy: 0.6331 - 827ms/epoch - 3ms/step
Loss: 0.8964827656745911, Accuracy: 0.633119523525238
```

```
In [ ]:
```