Overview of the Analysis

The objective of this analysis was to develop an algorithm for Alphabet Soup, a non-profit foundation, to predict funding success for applicants. Leveraging machine learning and neural networks, the task was to create a binary classifier using the provided dataset's features. The goal was to determine whether applicants would be successful if funded by Alphabet Soup.

Results

Data Preprocessing

- **Target Variable(s):** The target variable for the model was defined as "IS_SUCCESSFUL," having values 1 for 'yes' and 0 for 'no'.
- **Feature Variable(s):** Following the removal of 'EIN' and 'NAME,' the remaining columns were considered as features for the model. 'NAME' was temporarily added back for binning purposes in a secondary analysis.
- **Variables Removed:** 'EIN' and 'NAME' were removed as they did not contribute as either target or feature variables.

Compiling, Training, and Evaluating the Model

- **Neural Network Configuration:** Each model utilized a total of three layers following the application of Neural Networks. The number of hidden nodes was aligned with the number of features available.
- **Model Parameters:** The three-layer model resulted in 435 parameters. The initial attempt achieved accuracy slightly above 73%, slightly below the desired 75% threshold.

Compile, Train and Evaluate the Model

In [57]:
```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len(X_train_scaled[0])

hidden_nodes_layer1 =  7
hidden_nodes_layer2 = 14
hidden_nodes_layer3 = 21

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu"))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))

# Output layer
nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the model
nn.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 7)                 308

 dense_7 (Dense)             (None, 14)                112

 dense_8 (Dense)             (None, 1)                 15

=================================================================
Total params: 435 (1.70 KB)
Trainable params: 435 (1.70 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [60]:
```
# 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: 0.5534 - accuracy: 0.7303 - 237ms/epoch - 883us/step
Loss: 0.5534491539001465, Accuracy: 0.7302623987197876
```

Optimization

- **Improved Accuracy:** The second attempt, incorporating the "NAME" column in the dataset, achieved an accuracy of nearly 79%. This surpassed the target by 6%, utilizing 3,417 parameters.
- **Utilizing Multiple Layers:** Emphasizing the use of multiple layers in deep learning models was found beneficial, allowing the model to learn and classify information by filtering inputs through several layers.

In [21]:
```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len(X_train_scaled[0])

hidden_nodes_layer1 =  7
hidden_nodes_layer2 = 14
hidden_nodes_layer3 = 21

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu"))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))

# Output layer
nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the model
nn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 3290

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 1)                 15

=================================================================
Total params: 3417 (13.35 KB)
Trainable params: 3417 (13.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [24]:
```
# 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: 0.4589 - accuracy: 0.7893 - 243ms/epoch - 908us/step
Loss: 0.4589340090751648, Accuracy: 0.7892711162567139
```

Summary

The deep learning model exhibited promising results, achieving close to 79% accuracy after optimization. The addition of certain columns and the use of multiple layers significantly enhanced the model's predictive performance.

For further enhancement, exploring alternative models might be beneficial. Considering models like Random Forest or Gradient Boosting could provide a different perspective on this classification problem. These models can capture complex relationships between features differently, potentially offering more accuracy or insights.