Overview:

Alphabet Soup is interested in creating an algorithm which predicts whether or not applicant for funding will be successful. Using my knowledge of machine learning and neural networks, I will use the features in the dataset to create a binary classifier that is capable of predicting whether applicants will be successful if funded by Alphabet Soup. In my analysis I will use python's TensorFlow library to create, train, and evaluate data gathered from previous loans.

Results:

Data Preprocessing

I interrogated the data and established that the target variable is “IS_SUCCESSFUL”, with a value of 1 for yes and 0 for no. It was necessary to remove irrelevant information. In this I dropped “EIN” and “NAME” columns as they were not helpful in the model. The remaining columns became features for the model. Furthermore, I analyzed “APPLICATION” data and used the “CLASSIFICSTION” value for binning.

Compiling, Training, and Evaluating the Model

After applying the neural networks, each model had three-layer totals with hidden nodes dictated by the number of features

In [ ]:
```
# 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()
```

As a result, 447 parameters were created by this three-layer model with the first attempt producing about 73% accuracy, slightly below the ideal 75%.

```
[12] # 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)                 350

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 1)                 15

=================================================================
Total params: 477
Trainable params: 477
Non-trainable params: 0
_________________________________________________________________
```

```
# 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: 0.5525 - accuracy: 0.7303 - 656ms/epoch - 2ms/step
Loss: 0.5525117516517639, Accuracy: 0.7302623987197876
```

- **Optimization**

  For the second attempt, I maintained the “NAME” in the dataset and achieved a higher accuracy at about 79%, thus surpassing the target of 75%, albeit with 3,298 parameters.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 3171

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 1)                 15

=================================================================
Total params: 3,298
Trainable params: 3,298
Non-trainable params: 0
_________________________________________________________________
```

It would follow that multiple layers are applicable in deep learning models, allowing the prediction and classification of information with the computer filtering inputs via layers.

```
# 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: 0.4607 - accuracy: 0.7859 - 422ms/epoch - 2ms/step
Loss: 0.4606952369213104, Accuracy: 0.785889208316803
```