Overview:
The nonprofit foundation Alphabet Soup wanted a tool to help them select applicants for funding. They need a binary classifier to predict whether applicants will be successful if funded by Alphabet Soup.

Data Preprocessing:

Unnecessary metrics such as EIN and Name were removed from the dataset and all remaining metrics were considered in the model. Both Classification and Application Type were features for the model.

- What variable(s) are the target(s) for your model?
- What variable(s) are the features for your model?
- What variable(s) should be removed from the input data because they are neither targets nor features?

Compiling, Training, and Evaluating the Model:

Neural Network was used on each model and originally set with 2. For the final model, 3 layers were added that helped achieve an accuracy of over 75%.

Compile, Train and Evaluate the Model

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features = X_train_scaled.shape[1]
hidden_nodes1=7
hidden_nodes2=14
hidden_nodes3=21

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes1, input_dim=input_features, activation='relu'))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes2, activation='relu'))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes3, 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)                 1967

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 21)                315

 dense_3 (Dense)             (None, 1)                 22

=================================================================
Total params: 2,416
Trainable params: 2,416
Non-trainable params: 0
```

In order to achieve the model performance, I kept Name in the model and applied Name as a feature and binned the values. I kept classification as a feature in the model as well. In addition to the changes previously mentioned, I also added a third layer and changed the epochs to 200 instead of 100.

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features = X_train_scaled.shape[1]
hidden_nodes1=7
hidden_nodes2=14
hidden_nodes3=21

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes1, input_dim=input_features, activation='relu'))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes2, activation='relu'))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes3, 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)                 1967

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 21)                315

 dense_3 (Dense)             (None, 1)                 22

=================================================================
Total params: 2,416
Trainable params: 2,416
Non-trainable params: 0
```

```
# Train the model
fit_model = nn.fit(X_train_scaled,y_train,validation_split=0.15, epochs=200)
```

```
Epoch 1/200
684/684 [==============================] - 1s 653us/step - loss: 0.5338 - accuracy: 0.7437 - val_loss: 0.4533 - val
_accuracy: 0.7883
Epoch 2/200
684/684 [==============================] - 0s 542us/step - loss: 0.4558 - accuracy: 0.7811 - val_loss: 0.4444 - val
_accuracy: 0.7896
Epoch 3/200
684/684 [==============================] - 0s 537us/step - loss: 0.4472 - accuracy: 0.7823 - val_loss: 0.4414 - val
_accuracy: 0.7932
Epoch 4/200
684/684 [==============================] - 0s 540us/step - loss: 0.4435 - accuracy: 0.7841 - val_loss: 0.4413 - val
_accuracy: 0.7937
Epoch 5/200
684/684 [==============================] - 0s 539us/step - loss: 0.4420 - accuracy: 0.7852 - val_loss: 0.4417 - val
_accuracy: 0.7927
Epoch 6/200
684/684 [==============================] - 0s 541us/step - loss: 0.4401 - accuracy: 0.7867 - val_loss: 0.4419 - val
_accuracy: 0.7860
Epoch 7/200
```

```
# 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: 0.4721 - accuracy: 0.7801 - 89ms/epoch - 330us/step
Loss: 0.47208479046821594, Accuracy: 0.7800583243370056
```

Summary:

Several layers should be considered, so that it can continue to predict and classify information based on the model.