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 of the model.

- What variable(s) are the target(s) for your model? The target variable is the "IS_SUCCESSFUL" column from application_df
- What variable(s) are the features for your model? The features variable is every other column from application_df, which was done by dropping the "IS_SUCCESSFUL" column from the original dataframe.
- What variable(s) should be removed from the input data because they are neither targets nor features? The "EIN" and "NAME" columns were dropped because they were neither targets nor features for the dataset.

Compiling, Training, and Evaluating the Model:

1. How many neurons, layers, and activation functions did you select for your neural network model, and why?
In the first attempt, I added two layers (layer 1: 8 neurons; layer 2: 5 neurons). Both layers were activated-by the rectified linear unit (ReLu).There were no particular selection criteria for the neural networks parameters and the neurons in each later were random guesses from which to iterate upon in the second try.

2. Were you able to achieve the target model performance? - I was not able to achieve the 75% model accuracy target

3. What steps did you take in your attempts to increase model performance?

- Removed more column (EIN, NAME, <u>STATUS, ASK_AMT</u> (newly removed))
- In all attempts, adjusted neurons in each of the LAYERS
- In the last two attempts, added an additional layer making three layers with in an attempt to achieve higher model accuracy.
- In the last attempt, changed the epoch from 100 to 150.

Despite my efforts, the accuracy attained was around 73%. Attached below are the three attempts at optimization from first to third attempts respectively.

Compile, Train and Evaluate the Model

```
# FIRST ATTEMPT- 73% ACCURACY
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.

input_features_total = len(X_train[0])
hidden_nodes_layer1 = 30
hidden_nodes_layer2 = 40


nn = tf.keras.models.Sequential()

# First hidden layer

nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim = input_features_total, 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, 30)                1260

 dense_1 (Dense)             (None, 40)                1240

 dense_2 (Dense)             (None, 1)                 41

=================================================================
Total params: 2541 (9.93 KB)
Trainable params: 2541 (9.93 KB)
```

```
# SECOND ATTEMPT- 73% ACCURACY
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.

input_features_total = len(X_train[0])
hidden_nodes_layer1 = 20
hidden_nodes_layer2 = 30
hidden_nodes_layer3=75
nn = tf.keras.models.Sequential()

# First hidden layer

nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim = input_features_total, activation = "relu"))

# Second hidden layer

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

# Third hidden layer

nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation = "relu"))

# Output layer

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

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

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 20)                840

 dense_4 (Dense)             (None, 30)                630

 dense_5 (Dense)             (None, 75)                2325

 dense_6 (Dense)             (None, 1)                 76

=================================================================
Total params: 3871 (15.12 KB)
Trainable params: 3871 (15.12 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# THIRD ATTEMPT- 73% ACCURACY
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.

input_features_total = len(X_train[0])
hidden_nodes_layer1 = 50
hidden_nodes_layer2 = 25
hidden_nodes_layer3 = 10

nn = tf.keras.models.Sequential()

# First hidden layer

nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim = input_features_total, activation = "relu"))

# Second hidden layer

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

# Third hidden layer

nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation = "relu"))

# Output layer

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

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

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 50)                2100

 dense_8 (Dense)             (None, 25)                1275

 dense_9 (Dense)             (None, 10)                260

 dense_10 (Dense)            (None, 1)                 11

=================================================================
Total params: 3646 (14.24 KB)
Trainable params: 3646 (14.24 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
Epoch 95/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5371 - accuracy: 0.7369
Epoch 96/100
804/804 [==============================] - 3s 3ms/step - loss: 0.5374 - accuracy: 0.7363
Epoch 97/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5369 - accuracy: 0.7378
Epoch 98/100
804/804 [==============================] - 3s 3ms/step - loss: 0.5374 - accuracy: 0.7371
Epoch 99/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5370 - accuracy: 0.7368
Epoch 100/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5370 - accuracy: 0.7362
```

```python
# Evaluate the model using the test data (FIRST ATTEMPT- 73% ACCURACY)
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: 0.5575 - accuracy: 0.7338 - 514ms/epoch - 2ms/step
Loss: 0.5575341582298279, Accuracy: 0.7337609529495239
```

```
Epoch 95/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5359 - accuracy: 0.7374
Epoch 96/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5359 - accuracy: 0.7386
Epoch 97/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5357 - accuracy: 0.7386
Epoch 98/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5363 - accuracy: 0.7372
Epoch 99/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5357 - accuracy: 0.7388
Epoch 100/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5361 - accuracy: 0.7371
```

```python
# Evaluate the model using the test data (SECOND ATTEMPT- 73% ACCURACY)
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: 0.5578 - accuracy: 0.7317 - 504ms/epoch - 2ms/step
Loss: 0.5577606558799744, Accuracy: 0.7316617965698242
```

```
Epoch 95/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5357 - accuracy: 0.7379
Epoch 96/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5358 - accuracy: 0.7379
Epoch 97/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5357 - accuracy: 0.7380
Epoch 98/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5361 - accuracy: 0.7383
Epoch 99/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5357 - accuracy: 0.7379
Epoch 100/100
804/804 [==============================] - 3s 3ms/step - loss: 0.5356 - accuracy: 0.7381
```

```python
# Evaluate the model using the test data (THIRD ATTEMPT- 73% ACCURACY)
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: 0.5594 - accuracy: 0.7328 - 485ms/epoch - 2ms/step
Loss: 0.5594484210014343, Accuracy: 0.7328279614448547
```

Summary:

Overall, the deep learning model was around 74% accurate in predicting the classification problem. Using a model with greater correlation between input and output would likely result in higher prediction accuracy. This could be achieved by performing additional data cleanup prior to importing the data with python, as well as by using a model with different activation functions and iterating until higher accuracy is reached. Furthermore, several layers should be considered to help improve the accuracy of the model.