Overview:

The purpose of this analysis is to utilize machine learning and deep learning neural networks to form predictions using a binary classifier on whether applicants funded by nonprofit foundation Alphabet Soup will be successful. A CSV containing 34,000 organizations that have received Alphabet Soup funding will be utilized to train a model to make these predictions. The final objective of this project is to develop a model that will be able to effectively determine applicants that will have the highest likelihood of success if they were to receive funding from Alphabet Soup.

Results:

Data Preprocessing

- The **"IS_SUCCESSFUL**" column will serve as the target variable for this model to perform binary classification. This column indicates (1) as successful and (0) as not successful.
- The following columns will serve as features for the model:
    - **APPLICATION TYPE:** Alphabet Soup application type
    - **AFFILIATION:** Affiliated sector of industry
    - **CLASSIFICATION:** Government organization classification
    - **USE_CASE:** Use case for funding
    - **ORGANIZATION:** Organization type
    - **INCOME_AMT:** Income classification
    - **ASK_AMT:** Funding amount requested
- The following variables were removed, as they did not serve as contributors to the success of the organizations:
    - **EIN**
    - **NAME**

Compiling, Training, and Evaluating the Model

(3) attempts were made to optimize a model:

- **Attempt 1:** *Starter_Code.ipynb*

```
# 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 = 80
hidden_nodes_layer2 = 30

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()
```

Python

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 80)	6,320
dense_3 (Dense)	(None, 30)	2,430
dense_4 (Dense)	(None, 1)	31

```
Total params: 8,781 (34.30 KB)

Trainable params: 8,781 (34.30 KB)

Non-trainable params: 0 (0.00 B)
```

```
268/268 - 1s - 3ms/step - accuracy: 0.7284 - loss: 0.6052
Loss: 0.605224072933197, Accuracy: 0.728396475315094
```

- **<u>Attempt 2</u>:** *AlphabetSoupCharity_Optimization.ipynb*

```python
# 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 = 80
hidden_nodes_layer2 = 30
hidden_nodes_layer3 = 15

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'))

# 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()
```

Python

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 80)	6,720
dense_5 (Dense)	(None, 30)	2,430
dense_6 (Dense)	(None, 15)	465
dense_7 (Dense)	(None, 1)	16

```
Total params: 9,631 (37.62 KB)

Trainable params: 9,631 (37.62 KB)

Non-trainable params: 0 (0.00 B)
```

```
268/268 - 1s - 4ms/step - accuracy: 0.7305 - loss: 0.5692
Loss: 0.5691813230514526, Accuracy: 0.7304956316947937
```

- **<u>Attempt 3</u>:** *AlphabetSoupCharity_Optimization2.ipynb*

```python
# 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 = 160
hidden_nodes_layer2 = 60
hidden_nodes_layer3 = 30
hidden_nodes_layer4 = 15

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'))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation='relu'))

# Fourth hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer4, activation='relu'))

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

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

Python

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 160)	13,440
dense_1 (Dense)	(None, 60)	9,660
dense_2 (Dense)	(None, 30)	1,830
dense_3 (Dense)	(None, 15)	465
dense_4 (Dense)	(None, 1)	16

```
Total params: 25,411 (99.26 KB)

Trainable params: 25,411 (99.26 KB)

Non-trainable params: 0 (0.00 B)
```

```
268/268 - 1s - 4ms/step - accuracy: 0.7270 - loss: 0.5824
Loss: 0.5824322700500488, Accuracy: 0.7269970774650574
```

Were you able to achieve the target model performance?
I was not able to achieve the target model performance across 3 attempts. My second attempt was the closest to achieving the target model performance at 73.8% accuracy.

What steps did you take in your attempts to increase model performance?
Steps that I took to increase model performance were the following:

- Adjusting number of input features
- Adjusting number of hidden layers
- Adjusting number of epochs

From attempt 1 through attempt 3, I experimented with *addition* of input features, input layers, and epochs. I did not take considerations to *subtract* the quantity of these features. I additionally utilized the reLU and sigmoid functions to build this model.

Summary
Three attempts were made to optimize a deep learning model that would effectively predict the success outcome of a company based on certain features. The target accuracy that was desired for this model was 75%, and the highest accuracy score between the three attempts was 73.8% in attempt 2.

Generating a decision tree could be used to potentially solve this classification problem, as it is able to visually model decisions and the process of decision making. This can be useful as a developer who wants to see the entire decision making process in order to draw their own conclusions based on the decision tree output of the model. This type of model can decide on what features to include in the decision making process and under what certain conditions would a decision to be made, ultimately having the capability to make a binary classification prediction.