Overview:

Explain the purpose of this analysis.

The nonprofit foundation Alphabet Soup wants a tool that can help it select the applicants for funding with the best chance of success in their ventures. With the help of machine learning and neural networks, the features in the provided dataset to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup.

Results: Using bulleted lists and images to support your answers, address the following questions:

- Data Preprocessing
    - 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?
- Answer : For Data Processing the irrelevant columns like EIN and NAME was removed and rest was considered. For the targets [IS_SUCESSFUL] column was taken where the values are 0 and 1.For binning purpose , APPLICATION_TYPE and CLASSIFICATION were taken .Converted all Categorical value to numeric using pd.get dummies.
-

```
# Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df = application_df.drop(columns = ['EIN', 'NAME'])
application_df
```

```
# Choose a cutoff value and create a list of application types to be replaced
# use the variable name `application_types_to_replace`
application_types_to_replace = list(value_counts[value_counts<500].index)
application_types_to_replace

# Replace in dataframe
for app in application_types_to_replace:
    application_df['APPLICATION_TYPE'] = application_df['APPLICATION_TYPE'].replace(app,"Other")

# Check to make sure binning was successful
application_df['APPLICATION_TYPE'].value_counts()
```

```
# Choose a cutoff value and create a list of classifications to be replaced
# use the variable name `classifications_to_replace`
classifications_to_replace = list (counts_binning[counts_binning<100].index)
classifications_to_replace

# Replace in dataframe
for cls in classifications_to_replace:
    application_df['CLASSIFICATION'] = application_df['CLASSIFICATION'].replace(cls,"Other")

# Check to make sure binning was successful
application_df['CLASSIFICATION'].value_counts()
```

```
# Convert categorical data to numeric with `pd.get_dummies`
application_df = pd.get_dummies(application_df,dtype=float)
application_df.head()
```

```
# Split our preprocessed data into our features and target arrays
y = application_df['IS_SUCCESSFUL'].values
y
X = application_df.drop('IS_SUCCESSFUL', axis=1).values
X
# Split the preprocessed data into a training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state = 42)
```

- **Compiling, Training, and Evaluating the Model**

- How many neurons, layers, and activation functions did you select for your neural network model, and why?
- Were you able to achieve the target model performance?
- What steps did you take in your attempts to increase model performance?

Three layers were seen after applying a neural network, and the hidden nodes were determined by number of features.

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features = len( X_train_scaled[0])
hidden_nodes_layer1=4
hidden_nodes_layer2=8
hidden_nodes_layer3=16

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=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()
```

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

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 4)                 200

 dense_4 (Dense)             (None, 8)                 40

 dense_5 (Dense)             (None, 1)                 9

=================================================================
Total params: 249
Trainable params: 249
Non-trainable params: 0
_________________________________________________________________
```

```
8] # Compile the model
nn.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics=['accuracy'])
```

```
# 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: 0.5542 - accuracy: 0.7290 - 354ms/epoch - 1ms/step
Loss: 0.5542369484901428, Accuracy: 0.7289795875549316
```

For Optimization:

The NAME column is added for binning and the accuracy has achieved near to 80%.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 4)                 1812

 dense_1 (Dense)             (None, 8)                 40

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 1,861
Trainable params: 1,861
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 - 0s - loss: 0.4583 - accuracy: 0.7922 - 330ms/epoch - 1ms/step
Loss: 0.4583020508289337, Accuracy: 0.7921866178512573
```

3. **Summary**:

   For deep learning adding multiple layers will allow the models to predict and filter.