Purpose

This analysis aims to deploy deep-learning and Neural networks to assist the Alphabet Soup company in determining the best allocation of their funds. This analysis used features in the dataset to create binary classifiers to predict if the fund recipient will succeed. This report outlines the steps taken to preprocess the data, select and train a suitable model, evaluate its performance, and make a recommendation based on the results.

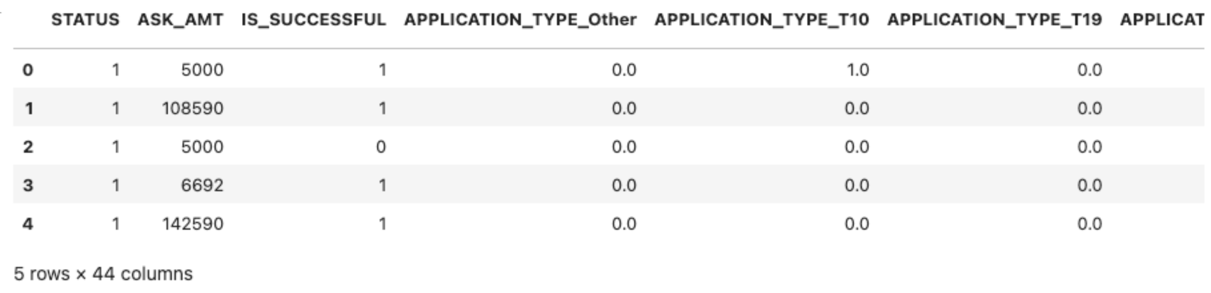
Results

Model 1: Alphabet Soup Charity Model

- **Data Preprocessing**

In this section, I explored the data to ensure it is processed and cleaned in a manner suitable for machine learning. Secondly, I used exploratory data analysis techniques to gain insights into the data distribution and identify patterns or correlations. This model removed the EIN and NAME columns; additionally, 'Other' replaced the CLASSIFICATION and APPLICATION_TYPE columns due to high fluctuations, as shown in Figure 1A below. Once the data was deemed suitable, I split the dataset into training and testing sets. The target variable for the model is "IS_SUCCESSFUL" and is verified by the value of 1 for YES and 0 for NO, as shown in Figure 1B below.

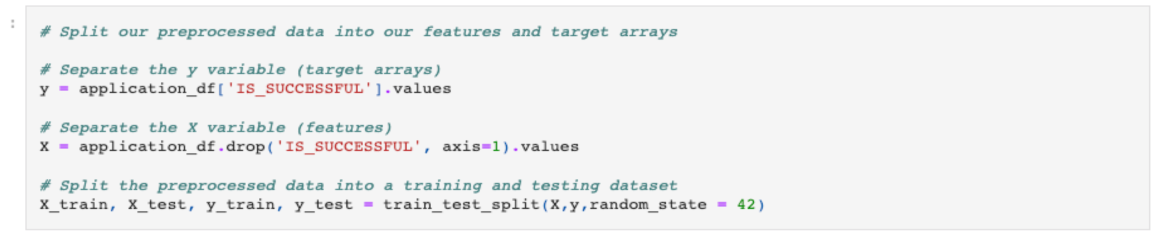
Figure 1A: Model 1 Processed Data for Machine Learning



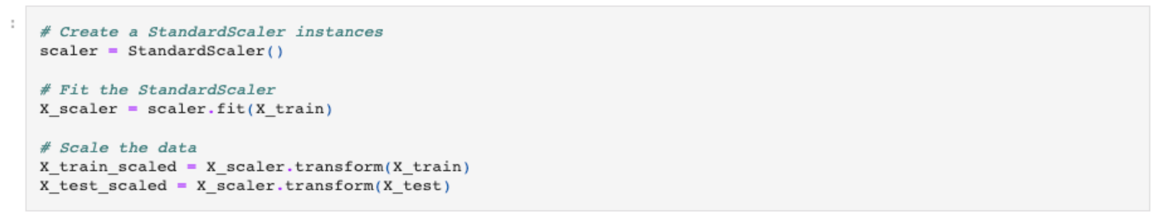
	STATUS	ASK_AMT	IS_SUCCESSFUL	APPLICATION_TYPE_Other	APPLICATION_TYPE_T10	APPLICATION_TYPE_T19	APPLICAT
0	1	5000	1	0.0	1.0	0.0	
1	1	108590	1	0.0	0.0	0.0	
2	1	5000	0	0.0	0.0	0.0	
3	1	6692	1	0.0	0.0	0.0	
4	1	142590	1	0.0	0.0	0.0	

5 rows × 44 columns

Figure 1B: Model 1 Data Fitting



```
# Split our preprocessed data into our features and target arrays

# Separate the y variable (target arrays)
y = application_df['IS_SUCCESSFUL'].values

# Separate the X variable (features)
X = application_df.drop('IS_SUCCESSFUL', axis=1).values

# 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)
```

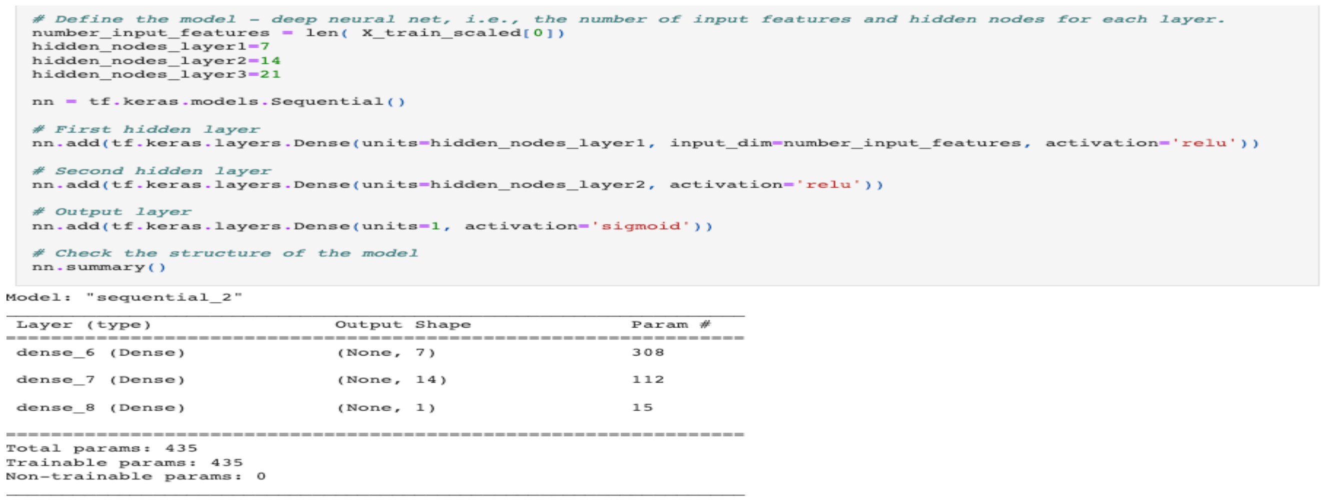


```
# Create a StandardScaler instances
scaler = StandardScaler()

# Fit the StandardScaler
X_scaler = scaler.fit(X_train)

# Scale the data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

- <u>**Compiling, Training, and Evaluating the Model**</u>

This model used the application of a neural network. The number of features dictated the number of hidden nodes. The number of features dictated the number of hidden nodes; a three-layer training model generated 435 parameters, as shown in Figure 2A. The target accuracy for this exercise is 75%; however, the model achieved only 72% accuracy, as shown in Figure 2B below.

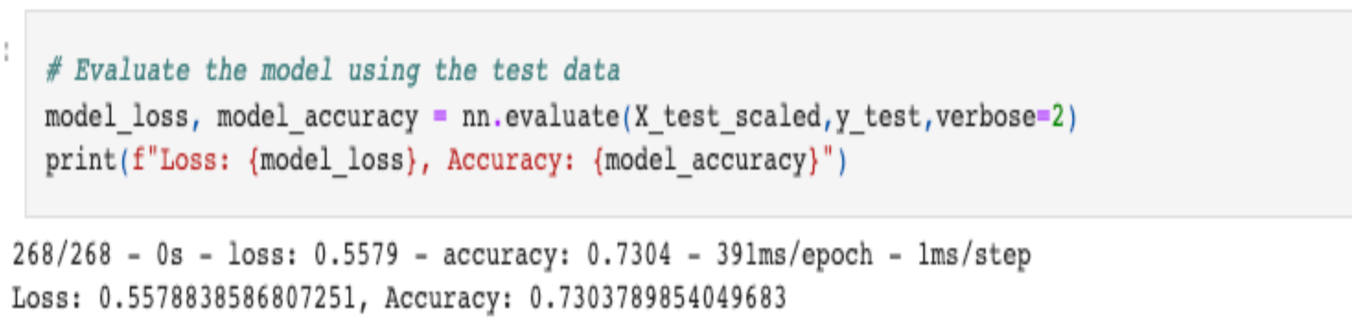
Figure 2A: Model 1 Input Features and Hidden Nodes and Parameters



```
# 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()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 7)                 308

 dense_7 (Dense)             (None, 14)                112

 dense_8 (Dense)             (None, 1)                 15

=================================================================
Total params: 435
Trainable params: 435
Non-trainable params: 0
_________________________________________________________________
```

Figure 2B: Model 1 Accuracy Score



```
# 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: 0.5579 - accuracy: 0.7304 - 391ms/epoch - 1ms/step
Loss: 0.5578838586807251, Accuracy: 0.7303789854049683
```

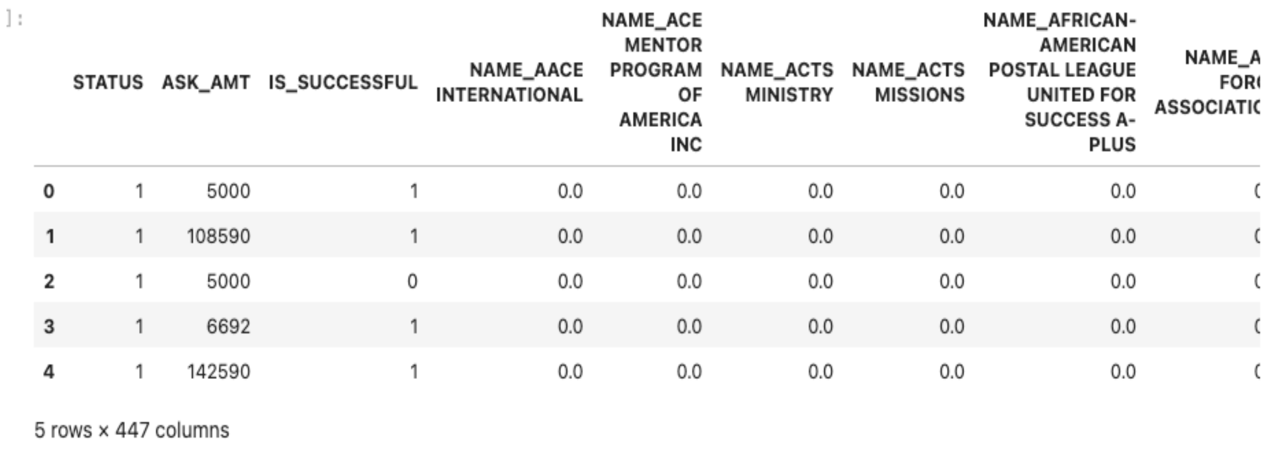
Model 2: Alphabet Soup Charity Optimization Model

- **Data Preprocessing**

Model 2 followed the same data preprocessing steps as Model except for the columns deleted.

Only the EIN column was deleted in this model, as shown in Figure 3 below.

Figure 3: Model 3 Input Features and Hidden Nodes



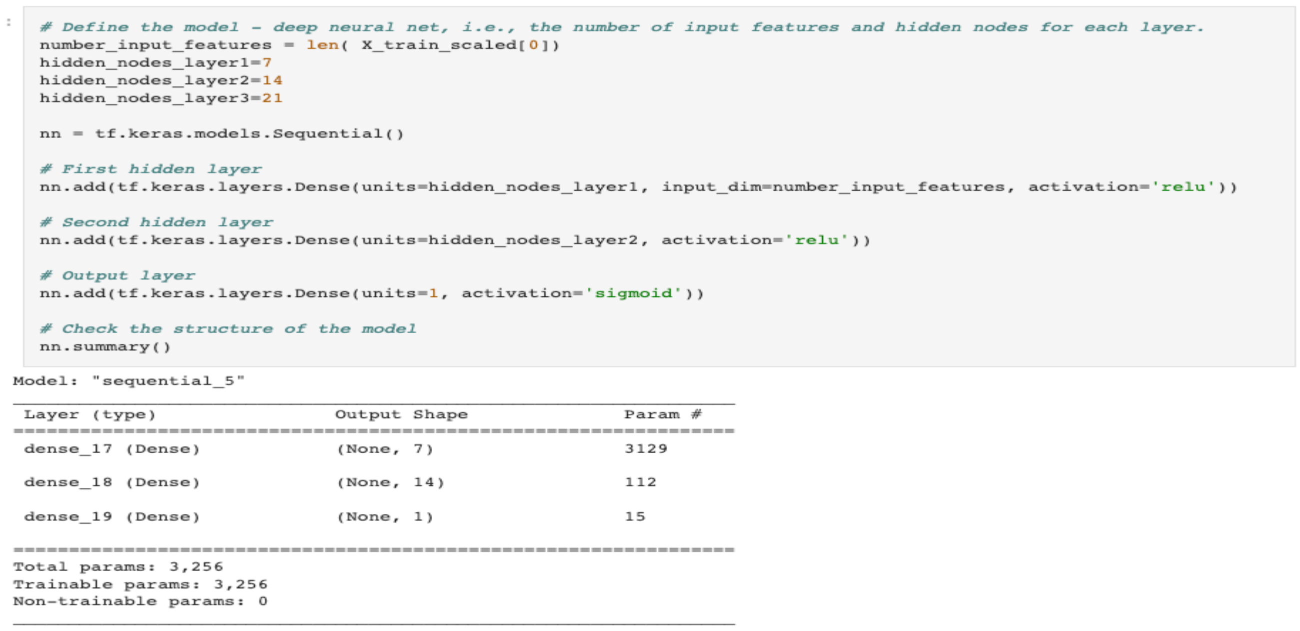
	STATUS	ASK_AMT	IS_SUCCESSFUL	NAME_AACE INTERNATIONAL	NAME_ACE MENTOR PROGRAM OF AMERICA INC	NAME_ACTS MINISTRY	NAME_ACTS MISSIONS	NAME_AFRICAN-AMERICAN POSTAL LEAGUE UNITED FOR SUCCESS A-PLUS	NAME_A FORI ASSOCIATIC
0	1	5000	1	0.0	0.0	0.0	0.0	0.0	(
1	1	108590	1	0.0	0.0	0.0	0.0	0.0	(
2	1	5000	0	0.0	0.0	0.0	0.0	0.0	(
3	1	6692	1	0.0	0.0	0.0	0.0	0.0	(
4	1	142590	1	0.0	0.0	0.0	0.0	0.0	(

5 rows × 447 columns

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

Like Model 1, this model used the application of a neural network. With the inclusion of the name column, a three-layer training model generated the number of parameters increases from 435 to 3,256, as shown in Figure 4A below. This optimized model achieved almost 79% accuracy, 4% above the target accuracy of 75%, as shown in Figure 4B below.

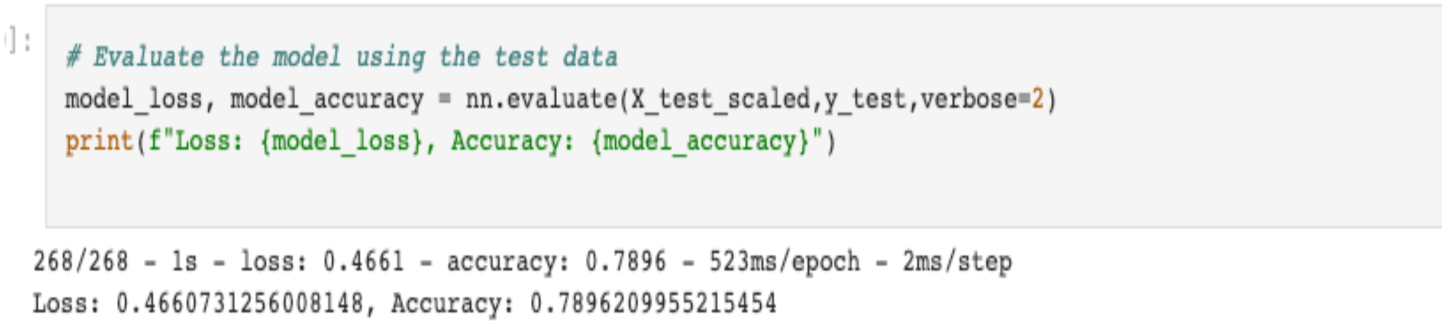
Figure 4A: Model 2 Input Features and Hidden Nodes and Parameters



```
# 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()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_17 (Dense)            (None, 7)                 3129

 dense_18 (Dense)            (None, 14)                112

 dense_19 (Dense)            (None, 1)                 15

=================================================================
Total params: 3,256
Trainable params: 3,256
Non-trainable params: 0
_________________________________________________________________
```

Figure 4B: Model 2 Accuracy Score



```
# 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: 0.4661 - accuracy: 0.7896 - 523ms/epoch - 2ms/step
Loss: 0.4660731256008148, Accuracy: 0.7896209955215454
```

Conclusion/Recommendation

In deep learning and neural network, multiple layers are fundamental. The model places layers hierarchically, each processing and transforming the input data to extract higher-level features.

The deeper the network, the more abstract and complex the features it can learn and represent.

Using multiple layers in deep learning offers several advantages:

- Representation power: Deep architectures can capture intricate patterns and relationships in the data, avoiding bias and making results fairer.
- Feature hierarchy: Each layer in a deep network can learn to represent different levels of abstraction, creating a hierarchical representation of the input data and allowing the model to understand the data more meaningfully.
- End-to-end learning: Multiple layers allow the model to directly map the raw input data to the desired output, avoiding the need for manual feature engineering.

When choosing the number of layers for a model, it is essential to consider that deeper models are not always better. Very deep networks can suffer from vanishing or exploding gradients, making training challenging. Additionally, deeper models require more computational resources and may be prone to overfitting when insufficient training data is unavailable. One should choose a neural network based on the complexity of the problem and the availability of data and computational resources. For some tasks, shallow networks might be sufficient; as seen in the optimized model, three layers with an additional column yield a desirable result of an accuracy score of 75% or higher.