Deep Learning Challenge Report

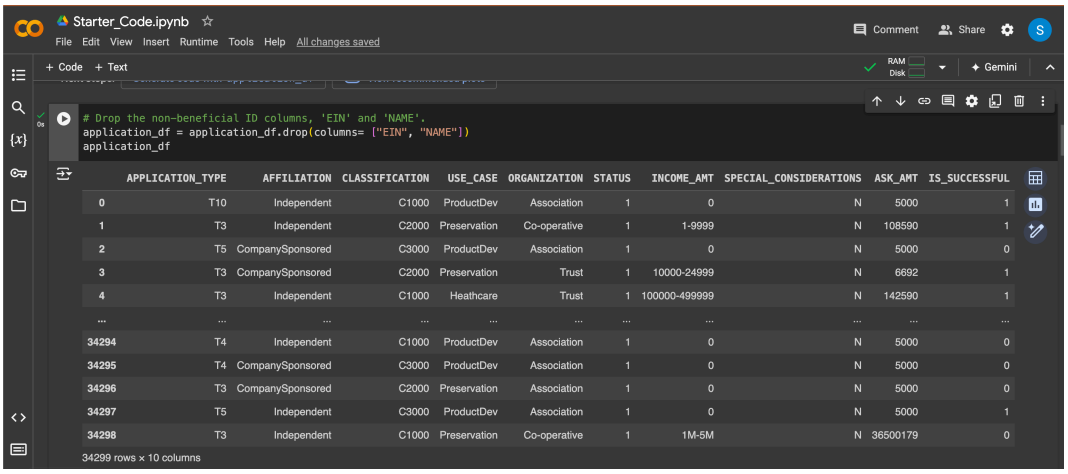
1. **Overview** of the analysis: Explain the purpose of this analysis.

The purpose of this analysis is to create a tool that can help the nonprofit organization, Alphabet Soup, select applicants with the highest chance of success and receive funding for their projects. By using machine learning and neural network background knowledge, we are able to create a binary classifier to determine whether applicants will be successful if funded by Alphabet Soup.

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

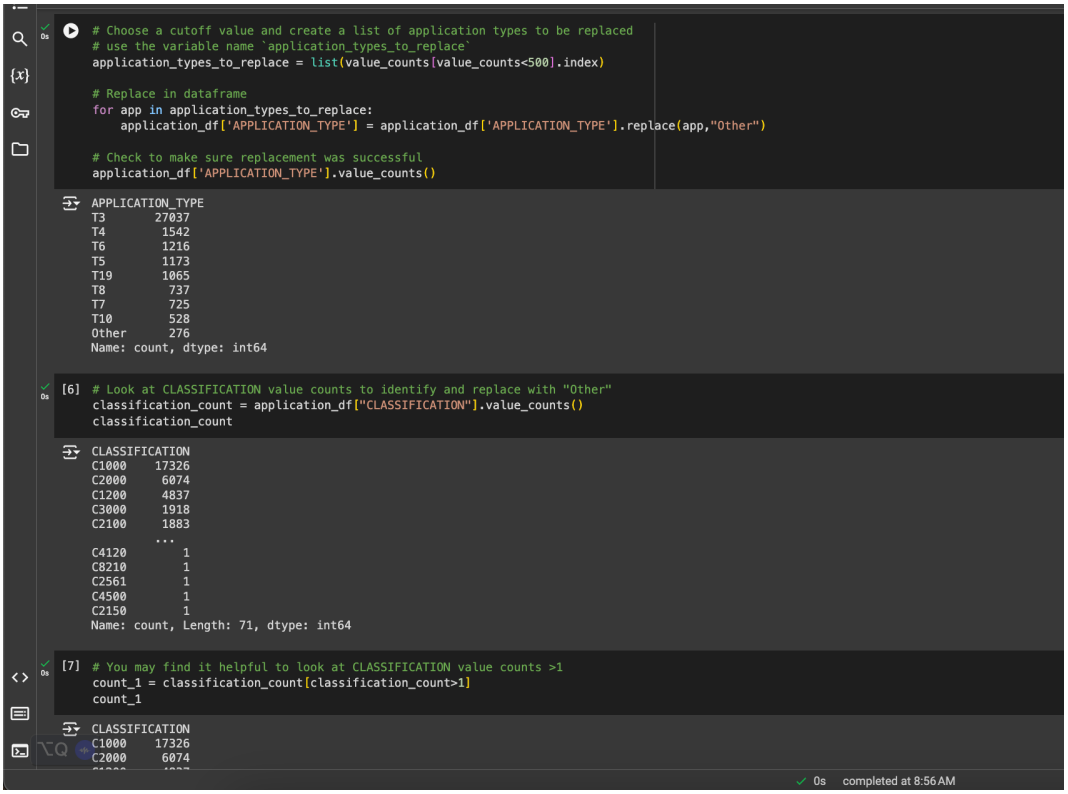
- Data Preprocessing
    - What variable(s) are the target(s) for your model?
        - The target variable that was evaluated was [‘IS_SUCCESSFUL] which represented the binary classifier values 0 and 1 were considered where 0 was ‘no’ and 1 was ‘yes’.
    - What variable(s) are the features for your model?
        - All the columns were features for the model especially ‘APPLICATION’ and ‘CLASSIFICATION’ which were used for binning and had certain cut offs in the preprocessing and optimization files.
        - The categories were checked for success after the preprocessing and optimization.
    - What variable(s) should be removed from the input data because they are neither targets nor features?
        - For preprocessing, variables such as ‘EIN’ and ‘NAME’ were dropped to remove identification of applicants.

■



	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZATION	STATUS	INCOME_AMT	SPECIAL_CONSIDERATIONS	ASK_AMT	IS_SUCCESSFUL
0	T10	Independent	C1000	ProductDev	Association	1	0	N	5000	1
1	T3	Independent	C2000	Preservation	Co-operative	1	1-9999	N	108590	1
2	T5	CompanySponsored	C3000	ProductDev	Association	1	0	N	5000	0
3	T3	CompanySponsored	C2000	Preservation	Trust	1	10000-24999	N	6692	1
4	T3	Independent	C1000	Heathcare	Trust	1	100000-499999	N	142590	1
...	...	...	...	...	...	...	...	...	...	...
34294	T4	Independent	C1000	ProductDev	Association	1	0	N	5000	0
34295	T4	CompanySponsored	C3000	ProductDev	Association	1	0	N	5000	0
34296	T3	CompanySponsored	C2000	Preservation	Association	1	0	N	5000	0
34297	T5	Independent	C3000	ProductDev	Association	1	0	N	5000	1
34298	T3	Independent	C1000	Preservation	Co-operative	1	1M-5M	N	36500179	0

34299 rows × 10 columns



```
# 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)

# 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 replacement was successful
application_df['APPLICATION_TYPE'].value_counts()
```

```
APPLICATION_TYPE
T3       27037
T4        1542
T6        1216
T5        1173
T19       1065
T8         737
T7         725
T10        528
Other      276
Name: count, dtype: int64
```

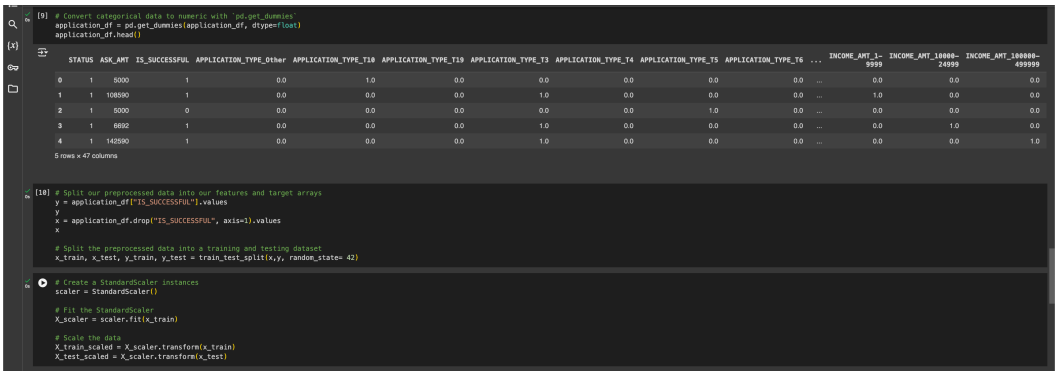
```
[6] # Look at CLASSIFICATION value counts to identify and replace with "Other"
classification_count = application_df["CLASSIFICATION"].value_counts()
classification_count
```

```
CLASSIFICATION
C1000    17326
C2000     6074
C1200     4837
C3000     1918
C2100     1883
         ...
C4120        1
C8210        1
C2561        1
C4500        1
C2150        1
Name: count, Length: 71, dtype: int64
```

```
[7] # You may find it helpful to look at CLASSIFICATION value counts >1
count_1 = classification_count[classification_count>1]
count_1
```

```
CLASSIFICATION
C1000    17326
C2000     6074
```

0s completed at 8:56 AM



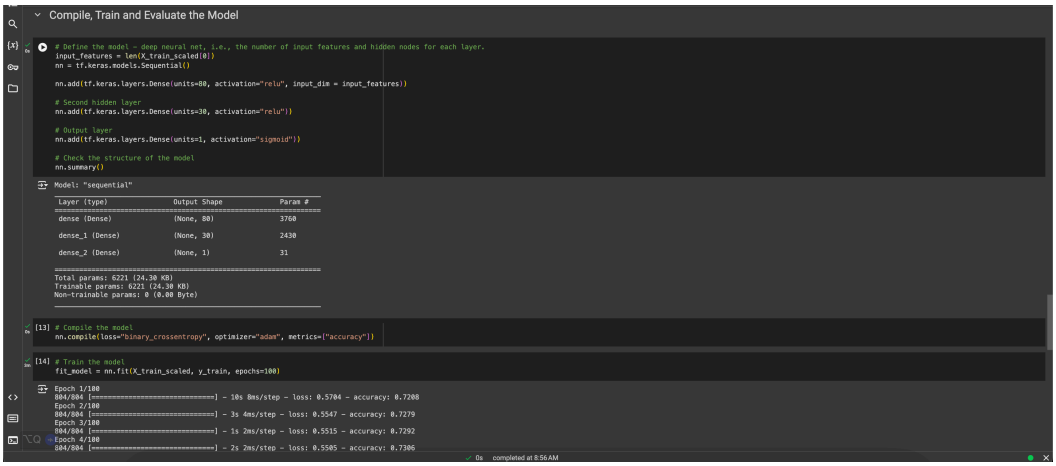
```
[9] # Convert categorical data to numeric with `pd.get_dummies`
    application_df = pd.get_dummies(application_df, dtype=float)
    application_df.head()
```

	STATUS	ASK_AMT	IS_SUCCESSFUL	APPLICATION_TYPE_Other	APPLICATION_TYPE_T10	APPLICATION_TYPE_T19	APPLICATION_TYPE_T3	APPLICATION_TYPE_T4	APPLICATION_TYPE_T5	APPLICATION_TYPE_T6	...	INCOME_AMT_1-9999	INCOME_AMT_10000-24999	INCOME_AMT_100000-499999
0	1	5000	1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
1	1	108590	1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0
2	1	5000	0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0
3	1	6692	1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	1.0	0.0
4	1	142590	1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	1.0

5 rows × 47 columns

```
[10] # 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)

     # 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)
```

- Compiling, Training, and Evaluating the Model
  - How many neurons, layers, and activation functions did you select for your neural network model, and why?
    - For both the preprocessing and optimization of the Alphabet Soup analysis, three layers were selected and the hidden nodes were dictated based on the number of features.



```
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 = len(X_train_scaled[0])
nn = tf.keras.models.Sequential()

nn.add(tf.keras.layers.Dense(units=80, activation="relu", input_dim = input_features))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=30, 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, 80)                3760

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 6221 (24.30 KB)
Trainable params: 6221 (24.30 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

[14] # Train the model
     fit_model = nn.fit(X_train_scaled, y_train, epochs=100)
```

```
Epoch 1/100
804/804 [==============================] - 10s 8ms/step - loss: 0.5704 - accuracy: 0.7208
Epoch 2/100
804/804 [==============================] - 3s 4ms/step - loss: 0.5547 - accuracy: 0.7279
Epoch 3/100
804/804 [==============================] - 1s 2ms/step - loss: 0.5515 - accuracy: 0.7292
Epoch 4/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5505 - accuracy: 0.7306
```

0s completed at 8:56 AM

  - Were you able to achieve the target model performance?
    - In the first attempt through preprocessing, there were 6221 total parameters which resulted in a 72.70% accuracy which was under the recommended 75%.

```
Epoch 87/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5329 - accuracy: 0.7403
Epoch 88/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5329 - accuracy: 0.7398
Epoch 89/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5327 - accuracy: 0.7407
Epoch 90/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5332 - accuracy: 0.7391
Epoch 91/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5331 - accuracy: 0.7397
Epoch 92/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5328 - accuracy: 0.7400
Epoch 93/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5328 - accuracy: 0.7402
Epoch 94/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5327 - accuracy: 0.7398
Epoch 95/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5325 - accuracy: 0.7406
Epoch 96/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5325 - accuracy: 0.7402
Epoch 97/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5320 - accuracy: 0.7405
Epoch 98/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5327 - accuracy: 0.7402
Epoch 99/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5321 - accuracy: 0.7393
Epoch 100/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5319 - accuracy: 0.7404
```

```
[15] # 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: 0.5646 - accuracy: 0.7270 - 508ms/epoch - 2ms/step
Loss: 0.5646084547042847, Accuracy: 0.7269970774650574
```

```
[16] # Export our model to HDF5 file
nn.save("./AlphabetSoupCharity.h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e
  saving_api.save_model(
```

- What steps did you take in your attempts to increase model performance?
    - To optimize the data, only the 'EIN' category was dropped and the 'NAME' category was added back in and the cutoff value was narrowed to <25 which helped narrow the analysis. The resulting parameters were then 1331 and the accuracy became 76.51% which is above the required 75% threshold.

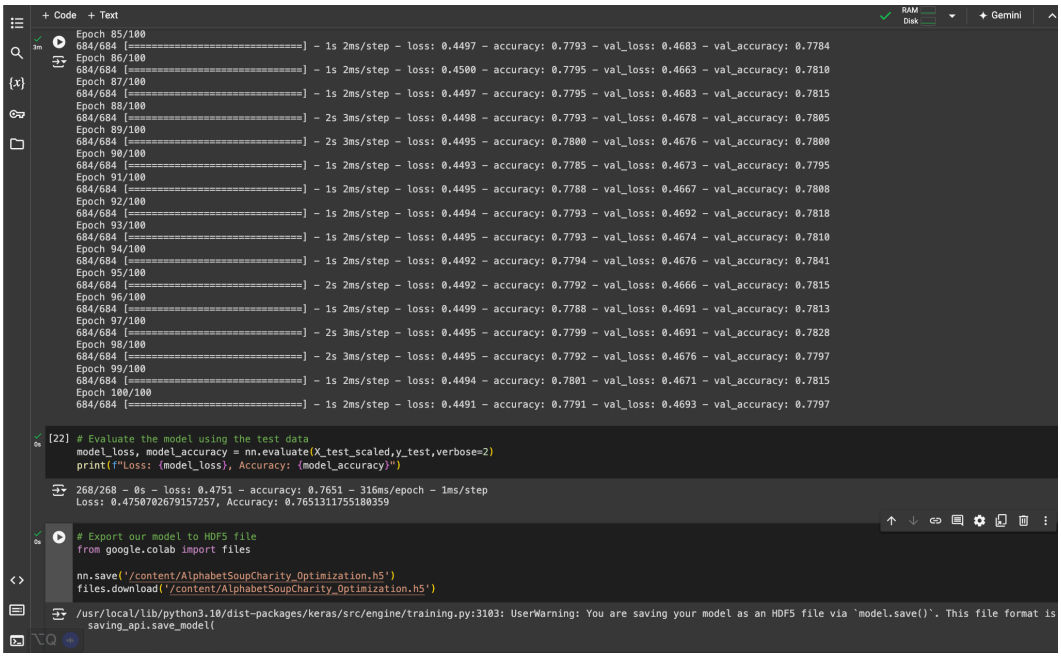
```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len( X_train_scaled[0])
hidden_nodes_layer1=7
hidden_nodes_layer2=14
hidden_nodes_layer3=21
nn = tf.keras.models.Sequential()

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_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 1204

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 1)                 15

=================================================================
Total params: 1331 (5.20 KB)
Trainable params: 1331 (5.20 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

```
[21] # Train the model
fit_model = nn.fit(X_train_scaled,y_train,validation_split=0.15, epochs=100)
```

```
Epoch 1/100
684/684 [==============================] - 5s 5ms/step - loss: 0.5478 - accuracy: 0.7220 - val_loss: 0.4769 - val_accuracy: 0.7632
Epoch 2/100
684/684 [==============================] - 2s 3ms/step - loss: 0.4901 - accuracy: 0.7555 - val_loss: 0.4701 - val_accuracy: 0.7722
```



3. **Summary**: Summarize the overall results of the deep learning model. Include a recommendation for how a different model could solve this classification problem, and then explain your recommendation.

    - To summarize, adding multiple layers to a deep learning model help optimize the data and allows the model to filter and predict the data better. This was proven by the optimization of the layers that were added after the preprocessing as well as the narrowing of the filter to less than 25 names. The deep learning, machine learning model is a good model to use for classification and prediction of various datasets.
    - If one were to look for other methods of solving this problem, other alternatives could be to use simulation modeling to mimic the behavior of the desired system over time or rule-based systems as there are specific rules and logic to follow which would result in defined patterns.