Alphabet Soup Charity Report

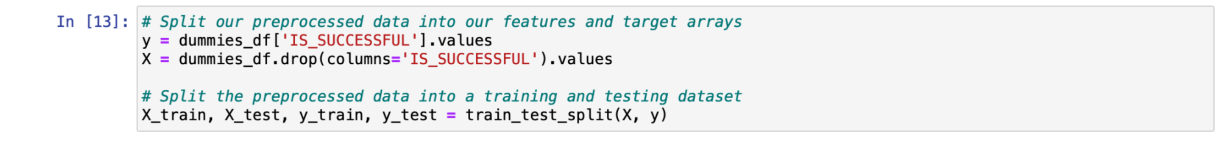
Overview of the analysis: Explain the purpose of this analysis.

The purpose of this analysis is to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup, a nonprofit foundation. The dataset used to create this classifier includes metadata (such as income classification, funding amount requested, and whether or not the money was used effectively, among other things) from over 34,000 organizations.

Data Preprocessing

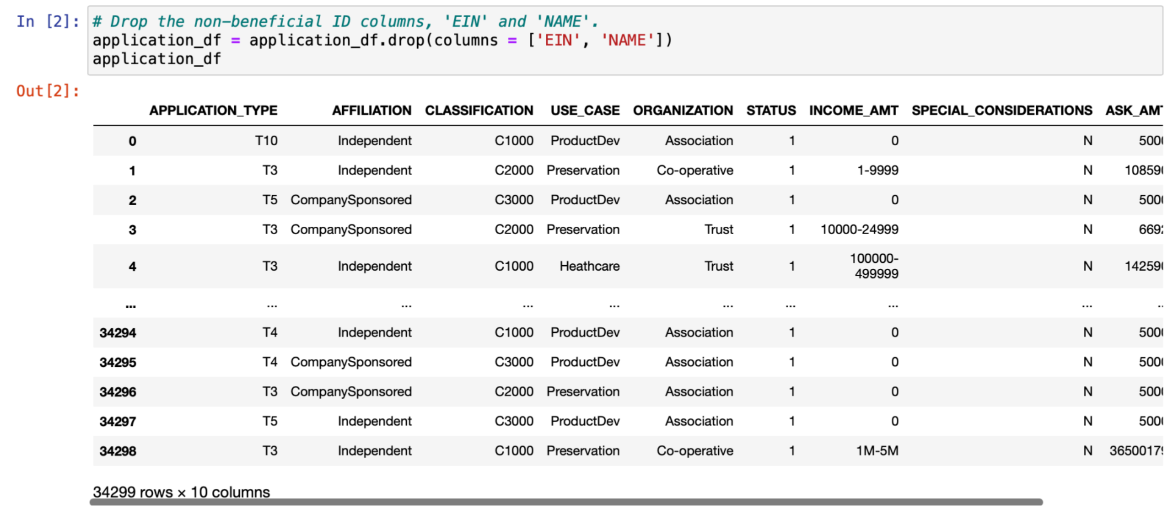
What variable(s) are the target(s) for your model?
The target for our model is X, which includes all of the columns in the dummies_df (modification of the application_df) except the "IS_SUCCESSFUL" column.

What variable(s) are the features for your model?
The feature for our model is y, which includes the "IS_SUCCESSFUL" column in the dummies_df.



```
In [13]: # Split our preprocessed data into our features and target arrays
         y = dummies_df['IS_SUCCESSFUL'].values
         X = dummies_df.drop(columns='IS_SUCCESSFUL').values

         # Split the preprocessed data into a training and testing dataset
         X_train, X_test, y_train, y_test = train_test_split(X, y)
```

What variable(s) should be removed from the input data because they are neither targets nor features?
We removed the 'EIN' and 'NAME' ID columns as they were non-beneficial.



```
In [2]: # Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
        application_df = application_df.drop(columns = ['EIN', 'NAME'])
        application_df
```

Out[2]:

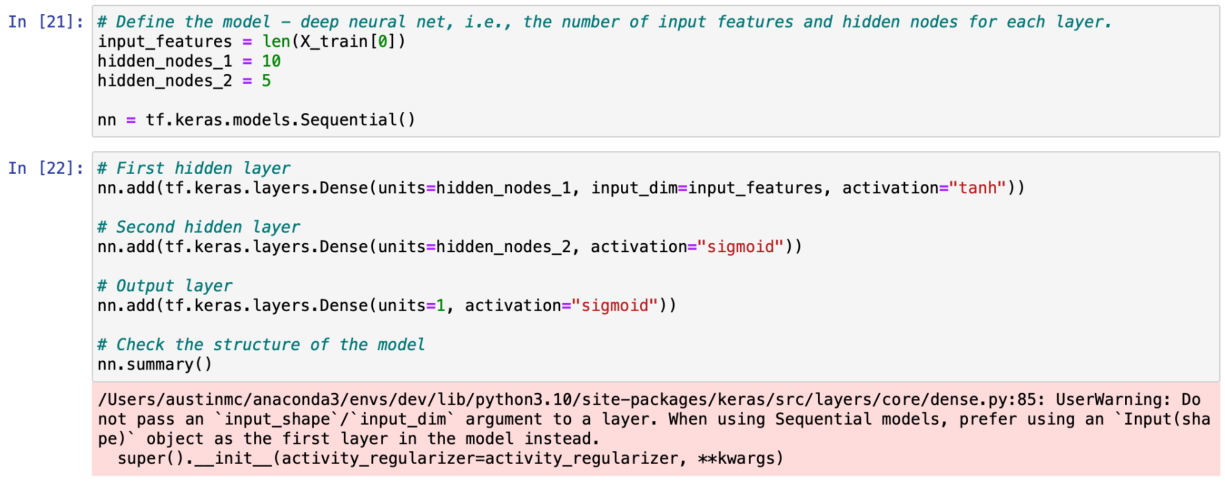
	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZATION	STATUS	INCOME_AMT	SPECIAL_CONSIDERATIONS	ASK_AM
0	T10	Independent	C1000	ProductDev	Association	1	0	N	500
1	T3	Independent	C2000	Preservation	Co-operative	1	1-9999	N	10859
2	T5	CompanySponsored	C3000	ProductDev	Association	1	0	N	500
3	T3	CompanySponsored	C2000	Preservation	Trust	1	10000-24999	N	669
4	T3	Independent	C1000	Heathcare	Trust	1	100000-499999	N	14259
...	...	...	...	...	...	...	...	...	..
34294	T4	Independent	C1000	ProductDev	Association	1	0	N	500
34295	T4	CompanySponsored	C3000	ProductDev	Association	1	0	N	500
34296	T3	CompanySponsored	C2000	Preservation	Association	1	0	N	500
34297	T5	Independent	C3000	ProductDev	Association	1	0	N	500
34298	T3	Independent	C1000	Preservation	Co-operative	1	1M-5M	N	365001

34299 rows × 10 columns

Compiling, Training, and Evaluating the Model

How many neurons, layers, and activation functions did you select for your neural network model, and why?

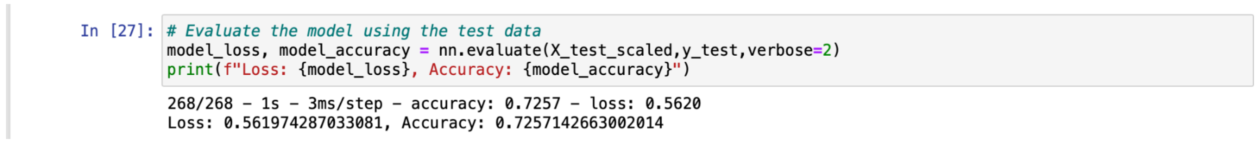
For my initial neural network model, I selected a sequential model with three layers (two hidden and one output) with “tanh” and “sigmoid” activation functions and 10, 5, and 1 units. I simply used this as a starting point to see what level of accuracy I would achieve before making tweaks.



```
In [21]: # Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
         input_features = len(X_train[0])
         hidden_nodes_1 = 10
         hidden_nodes_2 = 5

         nn = tf.keras.models.Sequential()

In [22]: # First hidden layer
         nn.add(tf.keras.layers.Dense(units=hidden_nodes_1, input_dim=input_features, activation="tanh"))

         # Second hidden layer
         nn.add(tf.keras.layers.Dense(units=hidden_nodes_2, activation="sigmoid"))

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

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

         /Users/austinmc/anaconda3/envs/dev/lib/python3.10/site-packages/keras/src/layers/core/dense.py:85: UserWarning: Do
         not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(sha
         pe)` object as the first layer in the model instead.
           super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Were you able to achieve the target model performance?

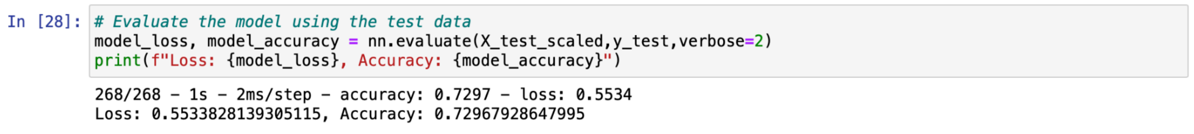
With an accuracy of around 72.5%, I was unable to achieve the target model performance.



```
In [27]: # 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 - 3ms/step - accuracy: 0.7257 - loss: 0.5620
         Loss: 0.561974287033081, Accuracy: 0.7257142663002014
```

What steps did you take in your attempt to increase model performance?

I tried various techniques, from dropping the ‘ASK_AMT’ column to changing the activation function to ‘relu’ to altering the number of units in each layer or number of epochs. My most successful was the second attempt, which garnered an accuracy of almost 73% and can be seen below.



```
In [28]: # 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 - 2ms/step - accuracy: 0.7297 - loss: 0.5534
         Loss: 0.5533828139305115, Accuracy: 0.72967928647995
```

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.

While the accuracy scores were fairly consistent, they could certainly use some improvement before any of these models would be reliable for Alphabet Soup. Perhaps a different type of neural network model other than sequential would be more fitting for this problem, which contains a very large dataset. We could also attempt to use a different machine learning model altogether to see if the results improved. However, at the end of the day, I believe additional attempts to increase the current model's performance would also be a surefire way to create a more reliable model.