Overview and Purpose: This analysis aims to develop a predictive tool for Alphabet Soup, a nonprofit foundation, to identify funding applicants with the highest likelihood of success in their ventures. By leveraging machine learning and neural networks, the provided dataset will be analyzed to build a binary classifier capable of predicting whether an applicant is likely to succeed if funded by Alphabet Soup.

Results: Data Preprocessing

1. Target Variable

- The target variable for the model is:
    - IS_SUCCESSFUL: A binary variable indicating whether the applicant was successful (1) or not (0).

2. Feature Variables

- The following variables were selected as features for the model:
    - APPLICATION_TYPE
    - AFFILIATION
    - CLASSIFICATION
    - USE_CASE
    - ORGANIZATION
    - STATUS
    - INCOME_AMT
    - SPECIAL_CONSIDERATIONS
    - ASK_AMT

3. Irrelevant Variables Removed

- The following variables were removed from the input data as they are neither targets nor meaningful features:
    - EIN: A unique identifier for each applicant that doesn't contribute to the prediction.
    - NAME: Often highly unique and unlikely to be predictive of success.

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

4. Preprocessing Steps

- Binning:
    - The columns APPLICATION_TYPE and CLASSIFICATION were binned to reduce the number of unique categories, ensuring meaningful groupings and avoiding overfitting.

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

    # 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	count
T3	27037
T4	1542
T6	1216
T5	1173
T19	1065
T8	737
T7	725
T10	528
Other	276

dtype: int64

```
[7] # Look at CLASSIFICATION value counts to identify and replace with "Other"
    classification_type_counts = application_df['CLASSIFICATION'].value_counts()
    print(classification_type_counts)
```

```
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
```

```
[9] # You may find it helpful to look at CLASSIFICATION value counts >1
    common_classifications = classification_type_counts[classification_type_counts > 1]
    print("\nCommon classifications (counts > 1):")
    print(common_classifications)
```

✓ 0s completed at 8:57 PM

- Encoding Categorical Data:
    - All categorical columns were converted into numeric representations using pd.get_dummies, enabling the model to process them.

```
columns_to_encode = ['APPLICATION_TYPE', 'AFFILIATION', 'CLASSIFICATION',
                     'USE_CASE', 'ORGANIZATION', 'INCOME_AMT', 'SPECIAL_CONSIDERATIONS']

# Verify the columns exist in the DataFrame
missing_columns = [col for col in columns_to_encode if col not in application_df.columns]
if missing_columns:
    print(f"Warning: These columns are missing from the DataFrame: {missing_columns}")
else:
    # Apply get_dummies to the specified columns
    application_df_dummies = pd.get_dummies(application_df, columns=columns_to_encode)
    print(application_df_dummies.head())
```

```
   STATUS  ASK_AMT  IS_SUCCESSFUL  APPLICATION_TYPE_Other  \
0       1     5000              1                   False
1       1   108590              1                   False
2       1     5000              0                   False
```

✓ 0s completed at 8:57 PM

Compiling, Training, and Evaluating the Model

Neural Network Architecture

- **Number of Input Features**:
    - Determined dynamically using len(X_train_scaled[0]) to match the dataset's features.
- **Hidden Layers**:
    - **First Hidden Layer**:
        - **Neurons**: 80
        - **Activation Function**: ReLU
    - **Second Hidden Layer**:
        - **Neurons**: 30

        - **Activation Function**: ReLU
- **Output Layer**:
    - **Neurons**: 1
    - **Activation Function**: Sigmoid (for binary classification)
- **Model Summary**:
    - The model consists of two hidden layers and one output layer. The ReLU activation function was chosen for hidden layers to capture non-linear relationships, while the Sigmoid function was used for the binary classification output.

Model Performance

- **Initial Model Results**:
    - **Accuracy**: 72.57%
    - **Loss**: 0.5617
    - This performance was achieved without including the NAME column as a feature.
- **Optimized Model Results**:
    - **Accuracy**: 77.91%
    - **Loss**: 0.4704
    - The model performance improved by including the NAME column, which provided additional relevant information, and refining the architecture with the same two-layer structure.

Steps to Improve Model Performance

- **Feature Engineering**:
    - Added the NAME column to the categorical features encoded with pd.get_dummies, ensuring all meaningful data was included in the model.
    - Verified the existence of all specified columns before encoding to prevent errors or data loss.
- **Hyperparameter Tuning**:
    - Experimented with the number of neurons in hidden layers (80 and 30) to balance model complexity and generalization.
    - Used the ReLU activation function for efficient training in hidden layers and Sigmoid for output.

- **Preprocessing Improvements**:
    - Standardized features with X_train_scaled to optimize model convergence.

Summary

- **Overall Results**:
    - The optimized model achieved an accuracy of 77.91% and a loss of 0.4704 on the test dataset, showing significant improvement compared to the initial implementation.