Alphabet Soup Funding Predictor

Overview:

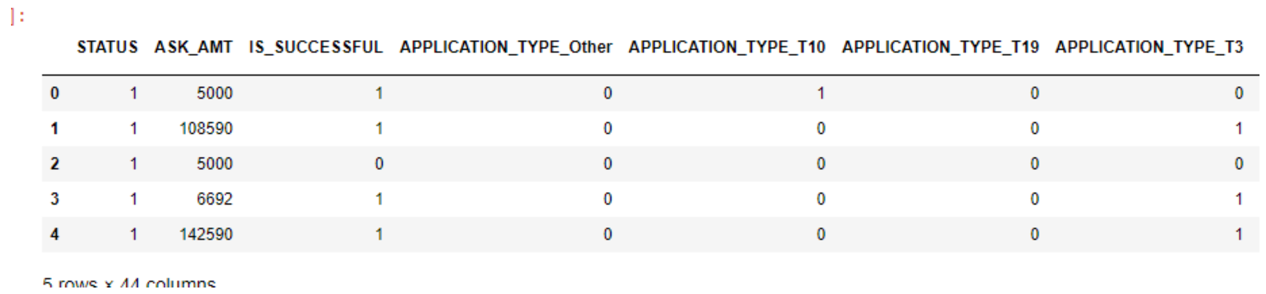
The aim of the analysis was to create a binary classifier model which would be able to predict the funding success outcome of a potential charitable project.

Constructing this model involved preprocessing the available dataset, and then using it to compile and train the model. Finally, this model was evaluated on the basis of its predictive accuracy.

Data Preprocessing:

A number of operations were performed on the initial dataset in order to make it more useful for training the model. These were:

- First, the identification columns EIN and NAME were dropped as they were not relevant as features that would assist in predicting outcomes.
- Bin application types and classification counts below a certain value to ensure a tractable number of items exist in each column. These figures are captured by a single 'Other' category.
- Convert categorical data to numeric binaries by creating dummy variables.
- A portion of the new dataframe is visible below for reference:



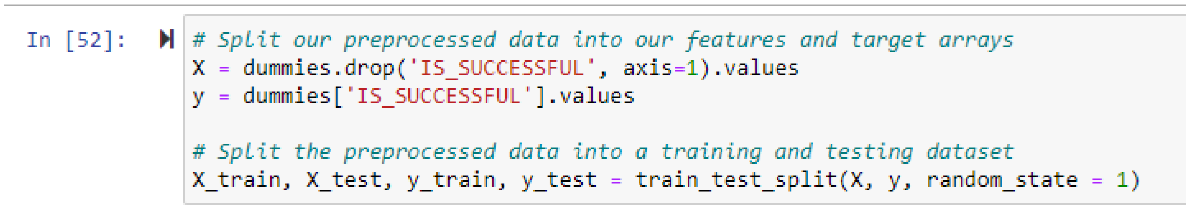
]:

	STATUS	ASK_AMT	IS_SUCCESSFUL	APPLICATION_TYPE_Other	APPLICATION_TYPE_T10	APPLICATION_TYPE_T19	APPLICATION_TYPE_T3
0	1	5000	1	0	1	0	0
1	1	108590	1	0	0	0	1
2	1	5000	0	0	0	0	0
3	1	6692	1	0	0	0	1
4	1	142590	1	0	0	0	1

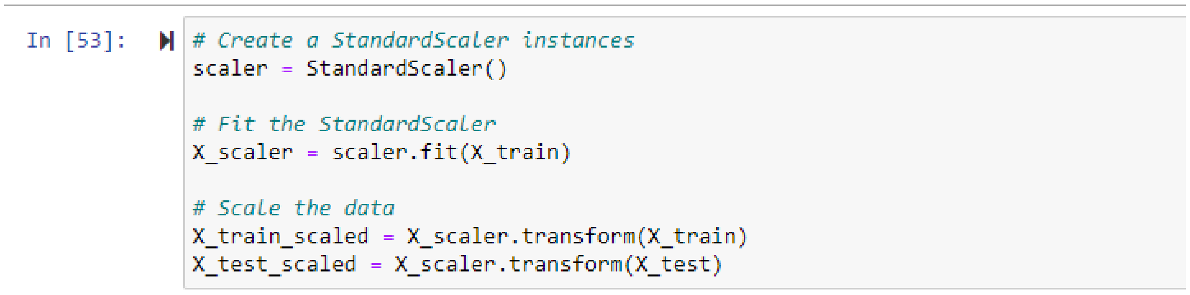
5 rows × 44 columns

- Our target variable is the binary 'IS_SUCCESFUL' values as the purpose of the model is to determine if a particular project will be funded or not.
- The features used to predict the target variable were the remaining variables apart from 'IS_SUCCESFUL': "APPLICATION_TYPE", "AFFILIATION", "CLASSIFICATION", "USE_CASE", "ORGANIZATION", "STATUS", "INCOME_AMT", "SPECIAL_CONSIDERATIONS", "ASK_AMT"
- Finally, numerical data was scaled using the standardscaler function.

- The pre-processed data was then split into training and testing sections; the training set would be used for training our model and the testing set would be used for evaluating its predictive accuracy:



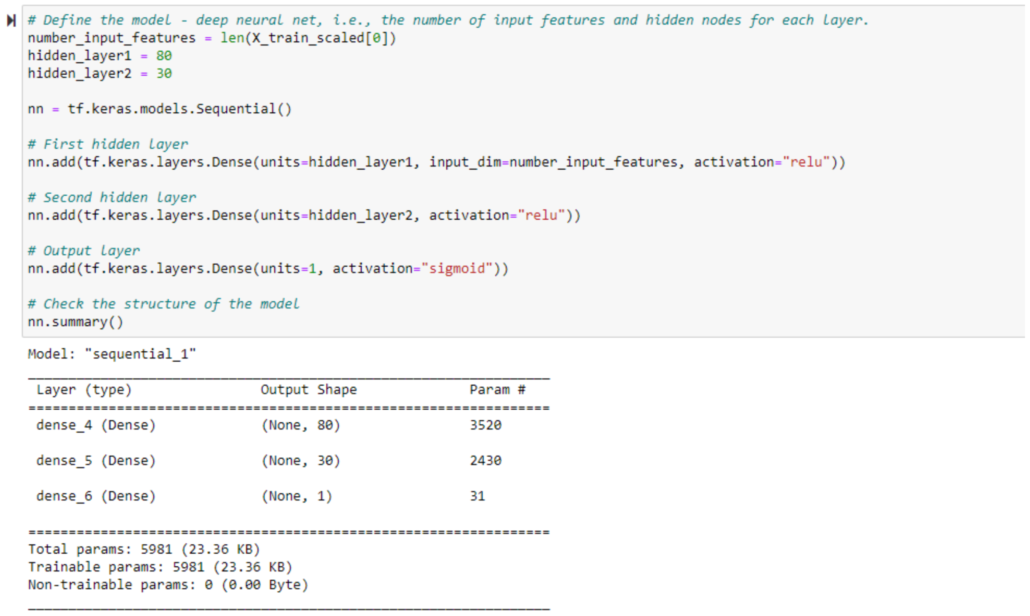
```
In [52]:  # Split our preprocessed data into our features and target arrays
          X = dummies.drop('IS_SUCCESSFUL', axis=1).values
          y = dummies['IS_SUCCESSFUL'].values

          # Split the preprocessed data into a training and testing dataset
          X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1)
```



```
In [53]:  # 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:

The benchmark accuracy to achieve for the model was 75%. A sequential model was implemented and the initial attempt at creating this model used an initial input layer, one hidden layer, and an output layer. The exact parameters of this model are described in the image below:



```
# 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_layer1 = 80
hidden_layer2 = 30

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_layer1, input_dim=number_input_features, activation="relu"))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_layer2, activation="relu"))

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

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

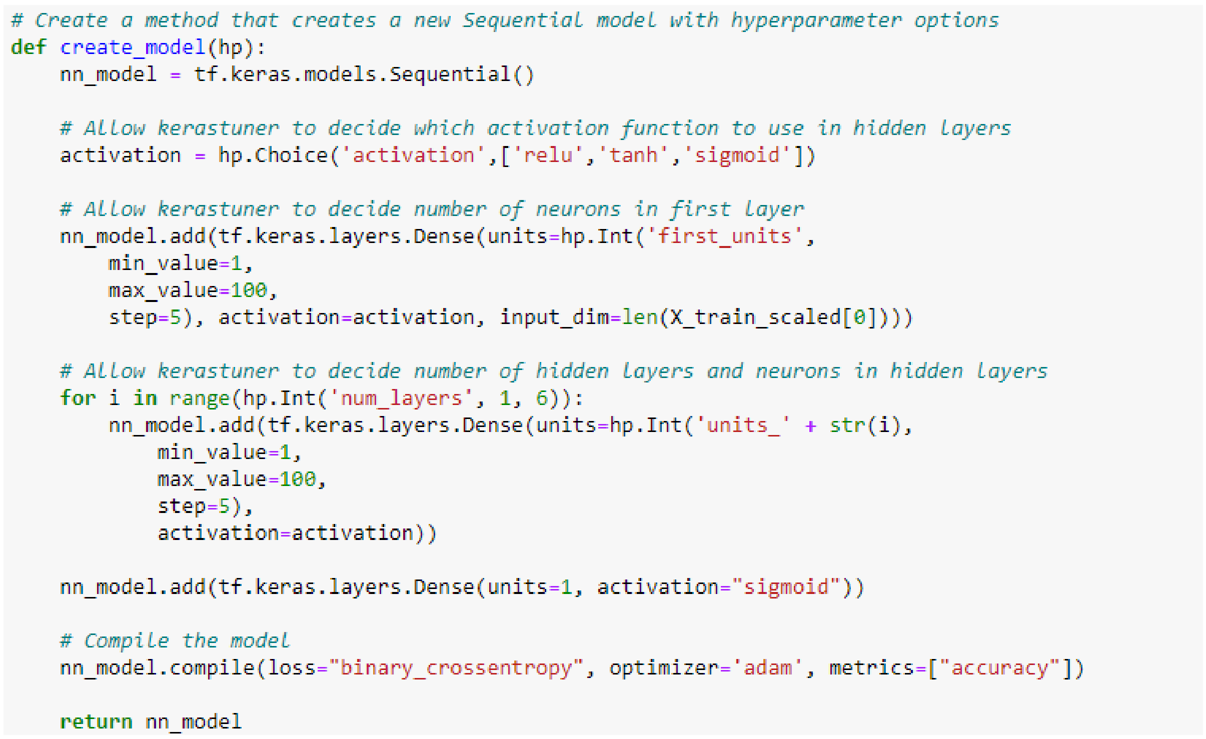
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 80)                3520

 dense_5 (Dense)             (None, 30)                2430

 dense_6 (Dense)             (None, 1)                 31

=================================================================
Total params: 5981 (23.36 KB)
Trainable params: 5981 (23.36 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

This model was trained on our scaled dataset and its accuracy was evaluated against our testing dataset. Our initial model achieved a predictive accuracy of approximately 73.08%.

Optimising the Model:

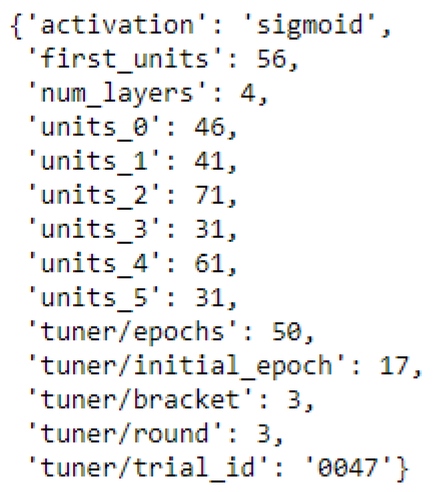
Two further attempts were made to improve the predictive accuracy of the model by adjusting its parameters. The first, Model Version 2, yielded the best results.

- **Model Version 2:**
    - An attempt to auto-optimise the model was made using the keras-tuner library to test different activation functions, numbers of hidden layers and the number of neurons in each layer. This can be seen below:



```python
# Create a method that creates a new Sequential model with hyperparameter options
def create_model(hp):
    nn_model = tf.keras.models.Sequential()

    # Allow kerastuner to decide which activation function to use in hidden layers
    activation = hp.Choice('activation',['relu','tanh','sigmoid'])

    # Allow kerastuner to decide number of neurons in first layer
    nn_model.add(tf.keras.layers.Dense(units=hp.Int('first_units',
        min_value=1,
        max_value=100,
        step=5), activation=activation, input_dim=len(X_train_scaled[0])))

    # Allow kerastuner to decide number of hidden layers and neurons in hidden layers
    for i in range(hp.Int('num_layers', 1, 6)):
        nn_model.add(tf.keras.layers.Dense(units=hp.Int('units_' + str(i),
            min_value=1,
            max_value=100,
            step=5),
            activation=activation))

    nn_model.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

    # Compile the model
    nn_model.compile(loss="binary_crossentropy", optimizer='adam', metrics=["accuracy"])

    return nn_model
```

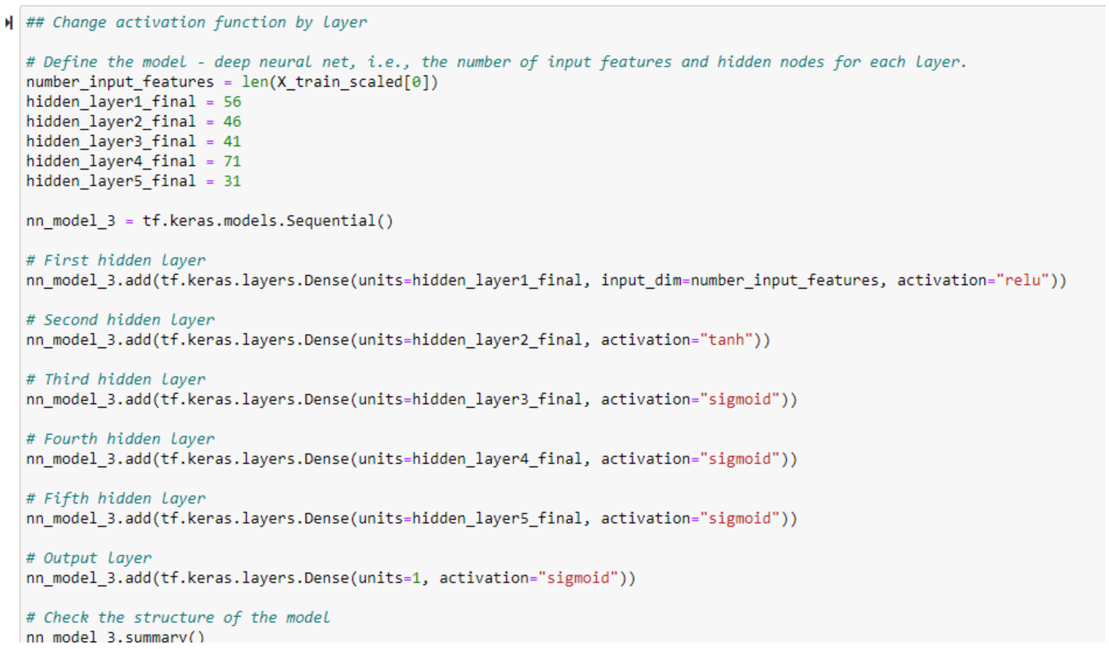
- The optimal suggested parameters are detailed in the image below:



```
{'activation': 'sigmoid',
 'first_units': 56,
 'num_layers': 4,
 'units_0': 46,
 'units_1': 41,
 'units_2': 71,
 'units_3': 31,
 'units_4': 61,
 'units_5': 31,
 'tuner/epochs': 50,
 'tuner/initial_epoch': 17,
 'tuner/bracket': 3,
 'tuner/round': 3,
 'tuner/trial_id': '0047'}
```

- The revised model, as explained by the characteristics above, achieved a predictive accuracy score of approximately 73.34%.

- **Model Version 3:**
  - As the auto-optimisation outlined above only considered one activation function for all hidden layers, it was decided to revise the model to consider different activation functions for each layer to see if predictive accuracy could be further improved. This model is described in the image below:



```
## Change activation function by layer

# 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_layer1_final = 56
hidden_layer2_final = 46
hidden_layer3_final = 41
hidden_layer4_final = 71
hidden_layer5_final = 31

nn_model_3 = tf.keras.models.Sequential()

# First hidden layer
nn_model_3.add(tf.keras.layers.Dense(units=hidden_layer1_final, input_dim=number_input_features, activation="relu"))

# Second hidden layer
nn_model_3.add(tf.keras.layers.Dense(units=hidden_layer2_final, activation="tanh"))

# Third hidden layer
nn_model_3.add(tf.keras.layers.Dense(units=hidden_layer3_final, activation="sigmoid"))

# Fourth hidden layer
nn_model_3.add(tf.keras.layers.Dense(units=hidden_layer4_final, activation="sigmoid"))

# Fifth hidden layer
nn_model_3.add(tf.keras.layers.Dense(units=hidden_layer5_final, activation="sigmoid"))

# Output layer
nn_model_3.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the model
nn_model_3.summary()
```

    - This model achieved a predictive accuracy score of 72.81%, the lowest of the three models tested.

Results and Conclusions:

It should be noted that our models fell just shy of achieving the benchmark accuracy of 75%. Model Version 2, which utilised the parameters suggested by the auto-optimisation, performed the best of the three tested.

Alternatives:

The auto-optimisation was limited by processing power constraints. It may be possible to improve the predictive accuracy of the model by developing a more detailed optimisation process that allows for different combinations of activation functions to be used for each hidden layer. Other parameters could also be given wider ranges, for example the number of neurons used in each layer could expanded to between 1 and 1,000. The practicability of this will depend on the processing power available.