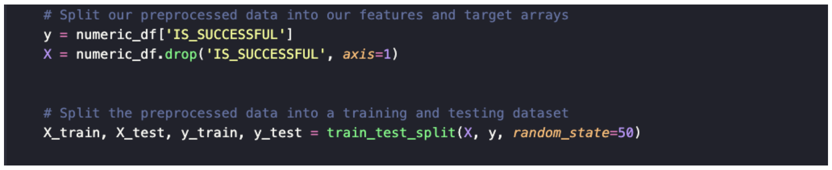
Overview:

The main purpose of this assignment is to use a deep learning model, TensorFlow/Keras, which aids a charity in selecting applicants for funding. This neural network model is to predict whether or not applicants are going to be successful or unsuccessful.

Data Preprocessing:

What variable(s) are the target(s) for your model?

- Target variable for our model was ‘IS_SUCCESSFUL’ which indicated whether an applicant was successful(1) or not(0).



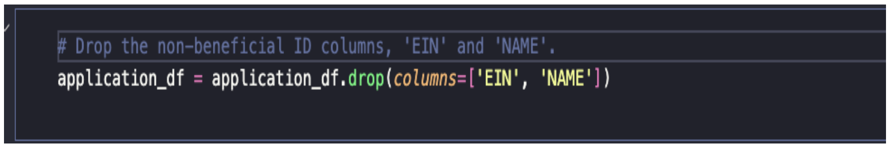
```
# Split our preprocessed data into our features and target arrays
y = numeric_df['IS_SUCCESSFUL']
X = numeric_df.drop('IS_SUCCESSFUL', axis=1)


# 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=50)
```

What variable(s) are the features for your model?

- Features included: ‘APPLICATION_TYPE’, ‘AFFILIATION’, ‘CLASSIFICATION’, ‘USE_CASE’, ‘ORGANIZATION’, ‘STATUS’, ‘INCOME_AMT’, ‘SPECIAL_CONSIDERATIONS’.

What variable(s) should be removed from the input data because they are neither targets nor features?

- EIN and Name were both removed from the data

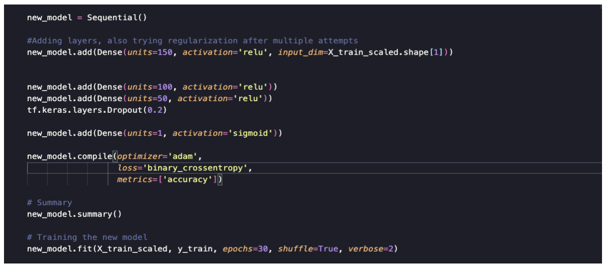


```
# Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df = application_df.drop(columns=['EIN', 'NAME'])
```

Compiling, Training, and Evaluating the Model:

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

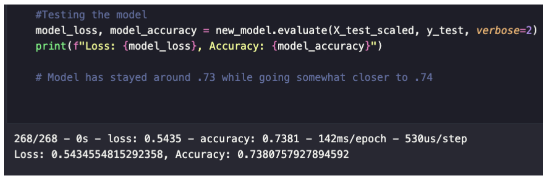
- This model contains 150 neurons for the input, while having hidden layers with 100 and 50. Also a 1 neuron output. Based on 3 I separated by 50.



```
new_model = Sequential()

#Adding layers, also trying regularization after multiple attempts
new_model.add(Dense(units=150, activation='relu', input_dim=X_train_scaled.shape[1]))


new_model.add(Dense(units=100, activation='relu'))
new_model.add(Dense(units=50, activation='relu'))
tf.keras.layers.Dropout(0.2)

new_model.add(Dense(units=1, activation='sigmoid'))

new_model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

# Summary
new_model.summary()

# Training the new model
new_model.fit(X_train_scaled, y_train, epochs=30, shuffle=True, verbose=2)
```

Were you able to achieve the target model performance?

- No, I was not able to get passed 74% after multiple attempts. I unfortunately got even lower after I looked online for other ways of aiding.

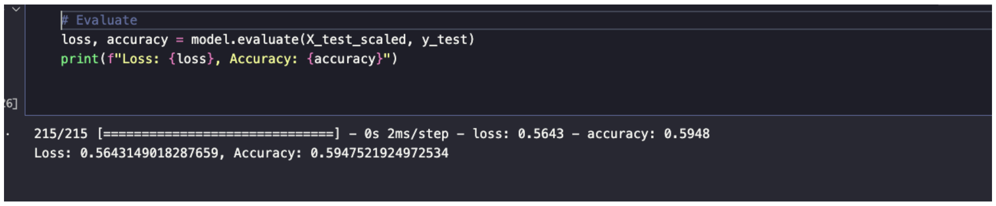


```
#Testing the model
model_loss, model_accuracy = new_model.evaluate(X_test_scaled, y_test, verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

# Model has stayed around .73 while going somewhat closer to .74
```

```
268/268 - 0s - loss: 0.5435 - accuracy: 0.7381 - 142ms/epoch - 530us/step
Loss: 0.5434554815292358, Accuracy: 0.7380757927894592
```

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

I changed layers as well as even added in callbacks and early stopping. Which was recommended from the Keras Website. However, this made the model perform worse at just about 60% accuracy.



```
# Evaluate
loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f"Loss: {loss}, Accuracy: {accuracy}")
```

```
215/215 [==============================] - 0s 2ms/step - loss: 0.5643 - accuracy: 0.5948
Loss: 0.5643149018287659, Accuracy: 0.5947521924972534
```

Summary:

In all, the model stayed around 73-74%. Adding more layers is better for making predictions. Using different commands and further knowledge would most likely be beneficial as when I attempted to find on my own without prior knowledge, I was actually left worse than before.