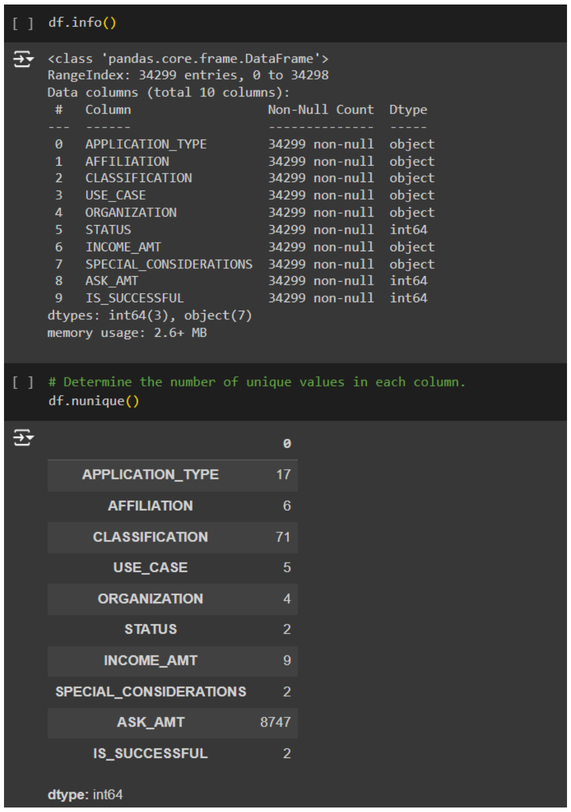
The purpose of this neural network was to help Alphabet Soup determine whether or not applicants who request funding will be successful or not based on a number of different features.

For example, the features of this dataset are application type, affiliation, classification, use_case, organization, status, income amount, special considerations, and ask amount.



```
[ ]  df.info()
```

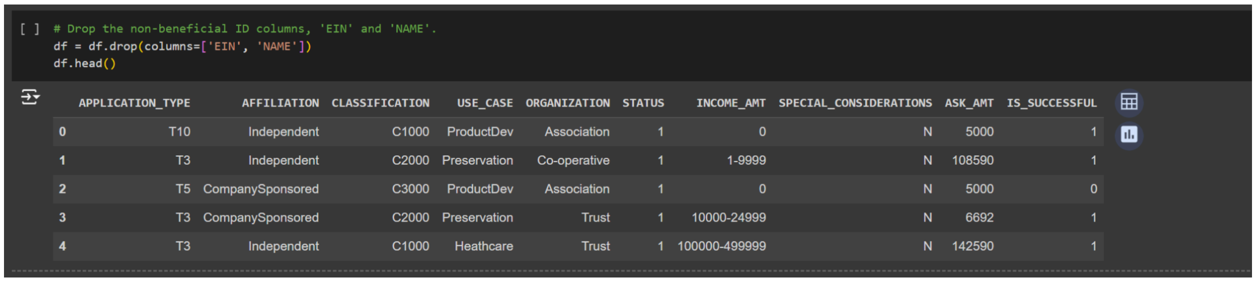
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34299 entries, 0 to 34298
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   APPLICATION_TYPE        34299 non-null  object
 1   AFFILIATION             34299 non-null  object
 2   CLASSIFICATION          34299 non-null  object
 3   USE_CASE                34299 non-null  object
 4   ORGANIZATION            34299 non-null  object
 5   STATUS                  34299 non-null  int64
 6   INCOME_AMT              34299 non-null  object
 7   SPECIAL_CONSIDERATIONS  34299 non-null  object
 8   ASK_AMT                 34299 non-null  int64
 9   IS_SUCCESSFUL           34299 non-null  int64
dtypes: int64(3), object(7)
memory usage: 2.6+ MB
```

```
[ ]  # Determine the number of unique values in each column.
     df.nunique()
```

	0
APPLICATION_TYPE	17
AFFILIATION	6
CLASSIFICATION	71
USE_CASE	5
ORGANIZATION	4
STATUS	2
INCOME_AMT	9
SPECIAL_CONSIDERATIONS	2
ASK_AMT	8747
IS_SUCCESSFUL	2

dtype: int64

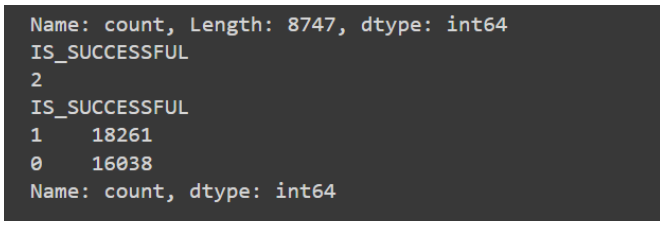
While the EIN and Name features were not necessary to the model because the EIN number and Name features were just classifying features for each individual instance. The numbers have no correlation to whether or not the loan would be successful as well as the Name.



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

	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

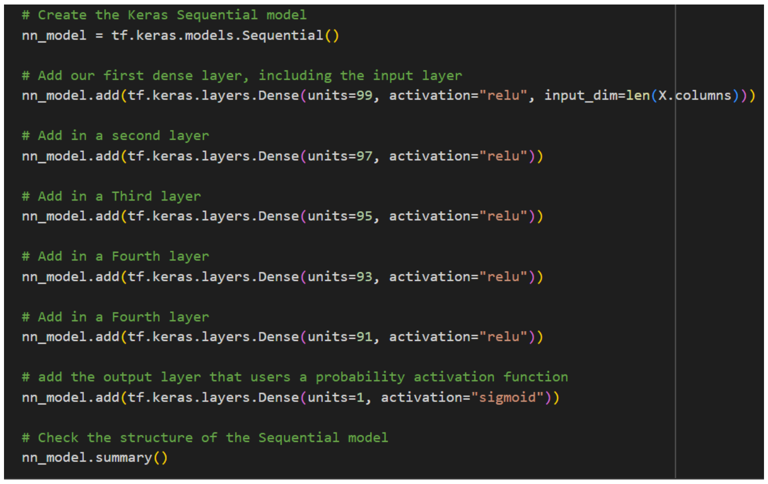
Our target feature was of course if the designated instance was a success or was not and if the money was used effectively.



```
Name: count, Length: 8747, dtype: int64
IS_SUCCESSFUL
2
IS_SUCCESSFUL
1    18261
0    16038
Name: count, dtype: int64
```

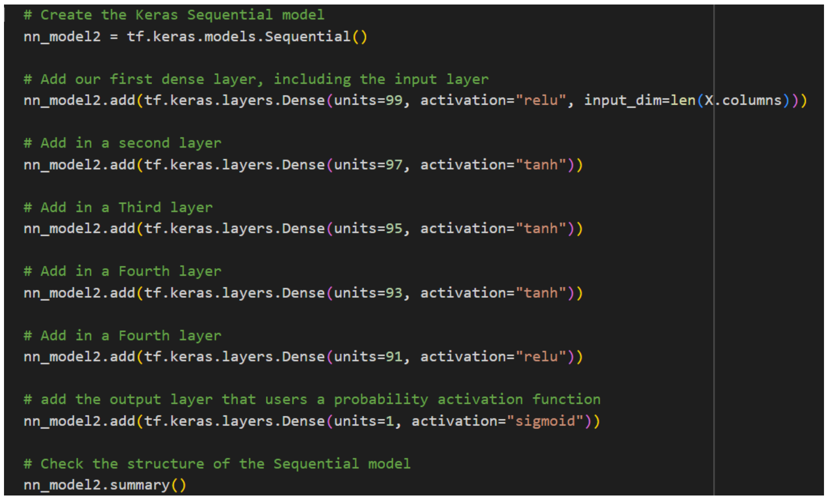
In my first model I bucketed the classification feature as well as the application type feature and used 6 layers with a total of 476 neurons and it only gave me an accuracy of 53.22%. This clearly missed the targeted results so I next bucketed the affiliation feature as well because of the range of smaller types with little amounts in them. I did the same thing to the organization feature as well. This only increased my accuracy to 53.52% in the end. So clearly bucketing wasn't working as I hoped. My next model I decided to increase the number of neurons from the 476 to 751, but even still only got a final accuracy of 53.15%. So finally I decided to change up the activation of some of the layers as well as add some more. I changed a few to tanh and others to elu. However, this resulted in the lowest accuracy of 52.93%.

Model 1:



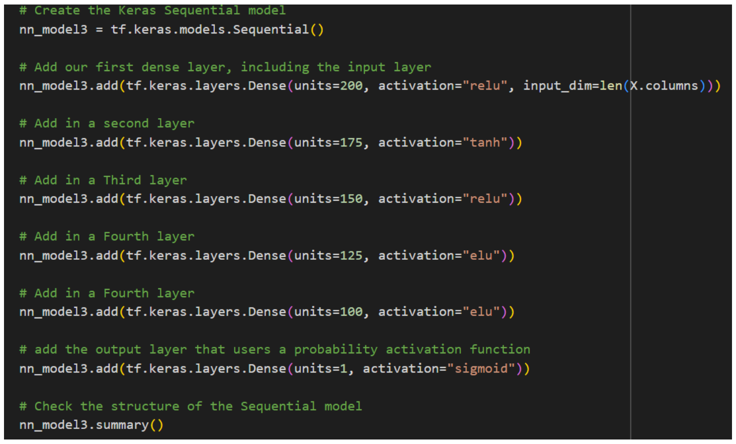
```
# Create the Keras Sequential model
nn_model = tf.keras.models.Sequential()

# Add our first dense layer, including the input layer
nn_model.add(tf.keras.layers.Dense(units=99, activation="relu", input_dim=len(X.columns)))

# Add in a second layer
nn_model.add(tf.keras.layers.Dense(units=97, activation="relu"))

# Add in a Third layer
nn_model.add(tf.keras.layers.Dense(units=95, activation="relu"))

# Add in a Fourth layer
nn_model.add(tf.keras.layers.Dense(units=93, activation="relu"))

# Add in a Fourth layer
nn_model.add(tf.keras.layers.Dense(units=91, activation="relu"))

# add the output layer that users a probability activation function
nn_model.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the Sequential model
nn_model.summary()
```

Model 2:



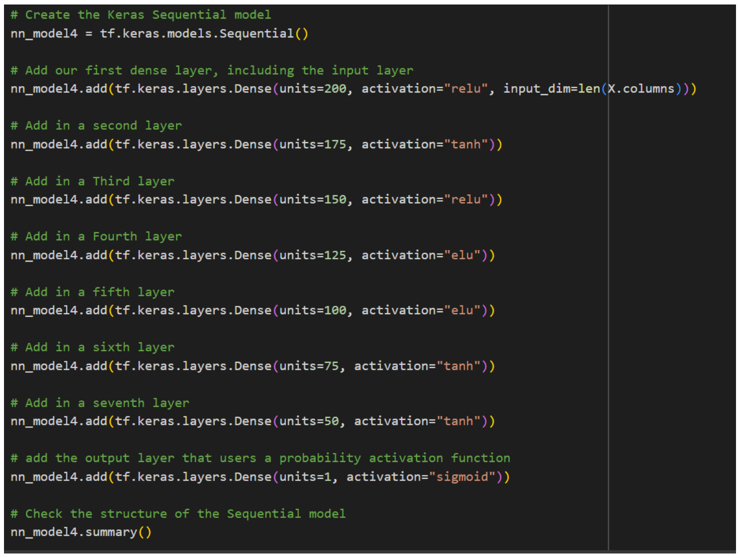
```
# Create the Keras Sequential model
nn_model2 = tf.keras.models.Sequential()

# Add our first dense layer, including the input layer
nn_model2.add(tf.keras.layers.Dense(units=99, activation="relu", input_dim=len(X.columns)))

# Add in a second layer
nn_model2.add(tf.keras.layers.Dense(units=97, activation="tanh"))

# Add in a Third layer
nn_model2.add(tf.keras.layers.Dense(units=95, activation="tanh"))

# Add in a Fourth layer
nn_model2.add(tf.keras.layers.Dense(units=93, activation="tanh"))

# Add in a Fourth layer
nn_model2.add(tf.keras.layers.Dense(units=91, activation="relu"))

# add the output layer that users a probability activation function
nn_model2.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the Sequential model
nn_model2.summary()
```

Model 3:



```
# Create the Keras Sequential model
nn_model3 = tf.keras.models.Sequential()

# Add our first dense layer, including the input layer
nn_model3.add(tf.keras.layers.Dense(units=200, activation="relu", input_dim=len(X.columns)))

# Add in a second layer
nn_model3.add(tf.keras.layers.Dense(units=175, activation="tanh"))

# Add in a Third layer
nn_model3.add(tf.keras.layers.Dense(units=150, activation="relu"))

# Add in a Fourth layer
nn_model3.add(tf.keras.layers.Dense(units=125, activation="elu"))

# Add in a Fourth layer
nn_model3.add(tf.keras.layers.Dense(units=100, activation="elu"))

# add the output layer that users a probability activation function
nn_model3.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the Sequential model
nn_model3.summary()
```

Model 4:



```
# Create the Keras Sequential model
nn_model4 = tf.keras.models.Sequential()

# Add our first dense layer, including the input layer
nn_model4.add(tf.keras.layers.Dense(units=200, activation="relu", input_dim=len(X.columns)))

# Add in a second layer
nn_model4.add(tf.keras.layers.Dense(units=175, activation="tanh"))

# Add in a Third layer
nn_model4.add(tf.keras.layers.Dense(units=150, activation="relu"))

# Add in a Fourth layer
nn_model4.add(tf.keras.layers.Dense(units=125, activation="elu"))

# Add in a fifth layer
nn_model4.add(tf.keras.layers.Dense(units=100, activation="elu"))

# Add in a sixth layer
nn_model4.add(tf.keras.layers.Dense(units=75, activation="tanh"))

# Add in a seventh layer
nn_model4.add(tf.keras.layers.Dense(units=50, activation="tanh"))

# add the output layer that users a probability activation function
nn_model4.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the Sequential model
nn_model4.summary()
```

Overall these models did not perform very well and were far off of the targeted accuracy of 75%. A way to possibly create a better model could have been to use a different type of encoding. I could have used one hot encoding instead of get dummies. I could of also possibly tried a few different things such as dropping additional features to see if certain features might not have as much weight as I thought.