Deliverable 3: Optimize the Model

```
# Import our dependencies
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,OneHotEncoder
import pandas as pd
import tensorflow as tf

import pandas as pd
application_df = pd.read_csv("./Resources/charity_data.csv")
application_df.head()
```

	EIN	NAME	APPLICATION_TYPE	AFFILIATION	CLASSIFICATIO
0	10520599	BLUE KNIGHTS MOTORCYCLE CLUB	T10	Independent	C100
1	10531628	AMERICAN CHESAPEAKE CLUB CHARITABLE TR	T3	Independent	C200
2	10547893	ST CLOUD PROFESSIONAL FIREFIGHTERS	T5	CompanySponsored	C300
3	10553066	SOUTHSIDE ATHLETIC ASSOCIATION	T3	CompanySponsored	C200

```
application_df.columns
```

```
Index(['EIN', 'NAME', 'APPLICATION_TYPE', 'AFFILIATION', 'CLASSIFICATION',
       'USE_CASE', 'ORGANIZATION', 'STATUS', 'INCOME_AMT',
       'SPECIAL_CONSIDERATIONS', 'ASK_AMT', 'IS_SUCCESSFUL'],
      dtype='object')
```

```
# Drop the non-beneficial ID columns
application_df = application_df.drop(["EIN"],1)
application_df
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarni
```

	NAME	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	U
0	BLUE KNIGHTS MOTORCYCLE CLUB	T10	Independent	C1000	Pr
1	AMERICAN CHESAPEAKE CLUB CHARITABLE TR	T3	Independent	C2000	Pre
2	ST CLOUD PROFESSIONAL FIREFIGHTERS	T5	CompanySponsored	C3000	Pr
3	SOUTHSIDE ATHLETIC ASSOCIATION	T3	CompanySponsored	C2000	Pre
4	GENETIC RESEARCH INSTITUTE OF THE DESERT	T3	Independent	C1000	H
...	...	...	...	...	
34294	THE LIONS CLUB OF HONOLULU KAMEHAMEHA	T4	Independent	C1000	Pr

```
application_df.nunique()
```

```
NAME                    19568
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
```

```
# Look at NAME value counts for binning
name_counts = application_df.NAME.value_counts()
#  How many name counts are greater than 5?
name_counts[name_counts>5]
```

```
PARENT BOOSTER USA INC                          1260
TOPS CLUB INC                                    765
UNITED STATES BOWLING CONGRESS INC               700
```

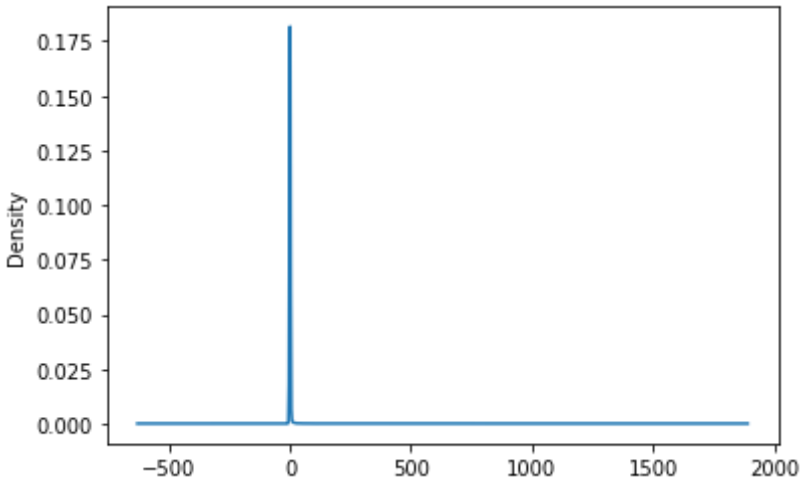
```
WASHINGTON STATE UNIVERSITY                              492
AMATEUR ATHLETIC UNION OF THE UNITED STATES INC          408
                                                        ...
OLD OAK CLIFF CONSERVATION LEAGUE INC                      6
AMERICAN NEPHROLOGY NURSES ASSOCIATION                     6
HUMBLE ISD EDUCATIONAL SUPPORT GROUPS INC                  6
PROFESSIONAL LOADMASTER ASSOCIATION                        6
CBMC INC                                                   6
Name: NAME, Length: 354, dtype: int64
```

```
#  How many name counts are less than or equal to 5?
name_counts[name_counts <= 5]
```

```
FLORIDA FAMILY CHILD CARE HOME ASSOCIATION INC                                5
GERONTOLOGICAL ADVANCED PRACTICE NURSES ASSOCIATION                           5
INTERNATIONL TRANSPLANT NURSES SOCIETY                                        5
NATIONAL ORGANIZATION FOR WOMEN INC                                           5
PTA HAWAII CONGRESS                                                           5
                                                                             ..
ST LOUIS SLAM WOMENS FOOTBALL                                                 1
AIESEC ALUMNI IBEROAMERICA CORP                                               1
WEALLBLEEDRED ORG INC                                                         1
AMERICAN SOCIETY FOR STANDARDS IN MEDIUMSHIP & PSYCHICAL INVESTIGATI          1
WATERHOUSE CHARITABLE TR                                                      1
Name: NAME, Length: 19214, dtype: int64
```

```
# Visualize the value counts of NAME
name_counts.plot.density()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f875526f710>
```




```
# Determine which values to replace if counts are less than or equal to 5.
replace_application = list(name_counts[name_counts <= 5].index)

# Replace in dataframe
for app in replace_application:
    application_df.NAME = application_df.NAME.replace(app,"Other")
```

```
# Check to make sure binning was successful
application_df.NAME.value_counts()
```

```
Other                                          20043
PARENT BOOSTER USA INC                          1260
TOPS CLUB INC                                    765
UNITED STATES BOWLING CONGRESS INC               700
WASHINGTON STATE UNIVERSITY                      492
                                                 ...
HABITAT FOR HUMANITY INTERNATIONAL                 6
DAMAGE PREVENTION COUNCIL OF TEXAS                 6
FLEET RESERVE ASSOCIATION                          6
HUGH OBRIAN YOUTH LEADERSHIP                       6
INTERNATIONAL CONGRESS OF CHURCHES MINISTERS       6
Name: NAME, Length: 355, dtype: int64
```

```
# Look at APPLICATION_TYPE value counts for binning
application_counts = application_df.APPLICATION_TYPE.value_counts()
application_counts
```

```
T3     27037
T4      1542
T6      1216
T5      1173
T19     1065
T8       737
T7       725
T10      528
T9       156
T13       66
T12       27
T2        16
T25        3
T14        3
T29        2
T15        2
T17        1
Name: APPLICATION_TYPE, dtype: int64
```

```
# Determine which values to replace if counts are less than 500
replace_application = list(application_counts[application_counts < 500].index)

# Replace in dataframe
for app in replace_application:
    application_df.APPLICATION_TYPE = application_df.APPLICATION_TYPE.replace(app,"Ot

# Check to make sure binning was successful
application_df.APPLICATION_TYPE.value_counts()
```

```
T3       27037
T4        1542
```

```
T6        1216
T5        1173
T19       1065
T8         737
T7         725
T10        528
Other      276
Name: APPLICATION_TYPE, dtype: int64
```

```
# Look at CLASSIFICATION value counts for binning
class_counts = application_df.CLASSIFICATION.value_counts()
class_counts
```

```
C1000    17326
C2000     6074
C1200     4837
C3000     1918
C2100     1883
          ...
C4120        1
C8210        1
C2561        1
C4500        1
C2150        1
Name: CLASSIFICATION, Length: 71, dtype: int64
```

```
# Determine which values to replace if counts are less than 1000
replace_class = list(class_counts[class_counts < 1000].index)

# Replace in dataframe
for cls in replace_class:
    application_df.CLASSIFICATION = application_df.CLASSIFICATION.replace(cls,"Other"

# Check to make sure binning was successful
application_df.CLASSIFICATION.value_counts()
```

```
C1000    17326
C2000     6074
C1200     4837
Other     2261
C3000     1918
C2100     1883
Name: CLASSIFICATION, dtype: int64
```

```
# Generate our categorical variable lists
application_cat = application_df.dtypes[application_df.dtypes == "object"].index.toli


# Create a OneHotEncoder instance
enc = OneHotEncoder(sparse=False)

# Fit and transform the OneHotEncoder using the categorical variable list
```

```
encode_df = pd.DataFrame(enc.fit_transform(application_df[application_cat]))

# Add the encoded variable names to the dataframe
encode_df.columns = enc.get_feature_names(application_cat)
encode_df.head()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: Fut
  warnings.warn(msg, category=FutureWarning)
```

	NAME_AACE INTERNATIONAL	NAME_ACE MENTOR PROGRAM OF AMERICA INC	NAME_AFRICAN-AMERICAN POSTAL LEAGUE UNITED FOR SUCCESS A-PLUS	NAME_AIR FORCE ASSOCIATION	NAME_ALABAMA FEDERATION OF WOMENS CLUBS	NAME_A TF ASSOC
0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	

5 rows × 396 columns

```
# Merge one-hot encoded features and drop the originals
application_df = application_df.merge(encode_df,left_index=True, right_index=True)
application_df = application_df.drop(application_cat,1)
application_df.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: FutureWarni
  This is separate from the ipykernel package so we can avoid doing imports
```

NAME AACE	NAME_ACE MENTOR PROGRAM	NAME_AFRICAN-AMERICAN POSTAL LEAGUE

```
# Split our preprocessed data into our features and target arrays
y = application_df["IS_SUCCESSFUL"].values
X = application_df.drop(["IS_SUCCESSFUL"],1).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=78)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: FutureWarning: I
  This is separate from the ipykernel package so we can avoid doing imports unti
```

```
# 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)


# Define the model - deep neural net
number_input_features = len(X_train[0])
hidden_nodes_layer1 =  100
hidden_nodes_layer2 = 30
hidden_nodes_layer3 = 10

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(
    tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features,
)

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="sigmoid"))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation="sigmoid"))

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

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               39900

 dense_1 (Dense)             (None, 30)                3030

 dense_2 (Dense)             (None, 10)                310

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 43,251
Trainable params: 43,251
Non-trainable params: 0
_________________________________________________________________
```

```
# Import checkpoint dependencies
import os
from tensorflow.keras.callbacks import ModelCheckpoint

# Define the checkpoint path and filenames
os.makedirs("challenge_checkpoints/",exist_ok=True)
checkpoint_path = "challenge_checkpoints/weights.{epoch:02d}.hdf5"


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


# Create a callback that saves the model's weights every epoch
cp_callback = ModelCheckpoint(
    filepath=checkpoint_path,
    verbose=1,
    save_weights_only=True,
    save_freq='epoch')


# Train the model
fit_model = nn.fit(X_train_scaled,y_train,epochs=100, callbacks=[cp_callback])
```

```
804/804 [==============================] - 2s 2ms/step - loss: 0.4017 - accuracy
Epoch 86/100
792/804 [============================>.] - ETA: 0s - loss: 0.4011 - accuracy: 0.
Epoch 86: saving model to challenge_checkpoints/weights.86.hdf5
804/804 [==============================] - 2s 2ms/step - loss: 0.4014 - accuracy
Epoch 87/100
791/804 [============================>.] - ETA: 0s - loss: 0.4017 - accuracy: 0.
Epoch 87: saving model to challenge_checkpoints/weights.87.hdf5
804/804 [==============================] - 2s 3ms/step - loss: 0.4013 - accuracy
Epoch 88/100
785/804 [============================>.] - ETA: 0s - loss: 0.4006 - accuracy: 0.
```

```
Epoch 88: saving model to challenge_checkpoints/weights.88.hdf5
804/804 [==============================] - 2s 2ms/step - loss: 0.4010 - accuracy
Epoch 89/100
794/804 [============================>.] - ETA: 0s - loss: 0.4016 - accuracy: 0.
Epoch 89: saving model to challenge_checkpoints/weights.89.hdf5
804/804 [==============================] - 2s 2ms/step - loss: 0.4015 - accuracy
Epoch 90/100
787/804 [============================>.] - ETA: 0s - loss: 0.4013 - accuracy: 0.
Epoch 90: saving model to challenge_checkpoints/weights.90.hdf5
804/804 [==============================] - 2s 2ms/step - loss: 0.4015 - accuracy
Epoch 91/100
791/804 [============================>.] - ETA: 0s - loss: 0.4012 - accuracy: 0.
Epoch 91: saving model to challenge_checkpoints/weights.91.hdf5
804/804 [==============================] - 2s 2ms/step - loss: 0.4011 - accuracy
Epoch 92/100
802/804 [============================>.] - ETA: 0s - loss: 0.4007 - accuracy: 0.
Epoch 92: saving model to challenge_checkpoints/weights.92.hdf5
804/804 [==============================] - 2s 2ms/step - loss: 0.4009 - accuracy
Epoch 93/100
798/804 [============================>.] - ETA: 0s - loss: 0.4022 - accuracy: 0.
Epoch 93: saving model to challenge_checkpoints/weights.93.hdf5
804/804 [==============================] - 2s 2ms/step - loss: 0.4019 - accuracy
Epoch 94/100
802/804 [============================>.] - ETA: 0s - loss: 0.4013 - accuracy: 0.
Epoch 94: saving model to challenge_checkpoints/weights.94.hdf5
804/804 [==============================] - 2s 2ms/step - loss: 0.4015 - accuracy
Epoch 95/100
802/804 [============================>.] - ETA: 0s - loss: 0.4011 - accuracy: 0.
Epoch 95: saving model to challenge_checkpoints/weights.95.hdf5
804/804 [==============================] - 2s 2ms/step - loss: 0.4010 - accuracy
Epoch 96/100
799/804 [============================>.] - ETA: 0s - loss: 0.4014 - accuracy: 0.
Epoch 96: saving model to challenge_checkpoints/weights.96.hdf5
804/804 [==============================] - 2s 3ms/step - loss: 0.4011 - accuracy
Epoch 97/100
798/804 [============================>.] - ETA: 0s - loss: 0.4017 - accuracy: 0.
Epoch 97: saving model to challenge_checkpoints/weights.97.hdf5
804/804 [==============================] - 3s 4ms/step - loss: 0.4015 - accuracy
Epoch 98/100
796/804 [============================>.] - ETA: 0s - loss: 0.4012 - accuracy: 0.
Epoch 98: saving model to challenge_checkpoints/weights.98.hdf5
804/804 [==============================] - 2s 2ms/step - loss: 0.4013 - accuracy
Epoch 99/100
799/804 [============================>.] - ETA: 0s - loss: 0.4014 - accuracy: 0.
Epoch 99: saving model to challenge_checkpoints/weights.99.hdf5
804/804 [==============================] - 2s 2ms/step - loss: 0.4012 - accuracy
Epoch 100/100
```

```
# 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 - 0s - loss: 0.4523 - accuracy: 0.7879 - 451ms/epoch - 2ms/step
Loss: 0.4523466229438782, Accuracy: 0.7878717184066772
```

```
# Export our model to HDF5 file
```

```
# Export our model to HDF5 file
nn.save("./Resources/AlphabetSoupCharity_Optimization.h5")
```

Random Forest model

```
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier


# Create a random forest classifier.
rf_model = RandomForestClassifier(n_estimators=128, random_state=78)

# Fitting the model
rf_model = rf_model.fit(X_train_scaled, y_train)

# Evaluate the model
y_pred = rf_model.predict(X_test_scaled)
print(f" Random forest model accuracy: {accuracy_score(y_test,y_pred):.3f}")
```

```
 Random forest model accuracy: 0.776
```

✓ 0s completed at 6:21 PM