Deliverable 1: Preprocessing the Data for a Neural Network

```
# 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 and read the charity_data.csv.
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

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

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

	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANI
0	T10	Independent	C1000	ProductDev	Ass
1	T3	Independent	C2000	Preservation	Co-o
2	T5	CompanySponsored	C3000	ProductDev	Ass
3	T3	CompanySponsored	C2000	Preservation	
4	T3	Independent	C1000	Heathcare	

```
# Determine the number of unique values in each column.
application_df.nunique()
```

```
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
```

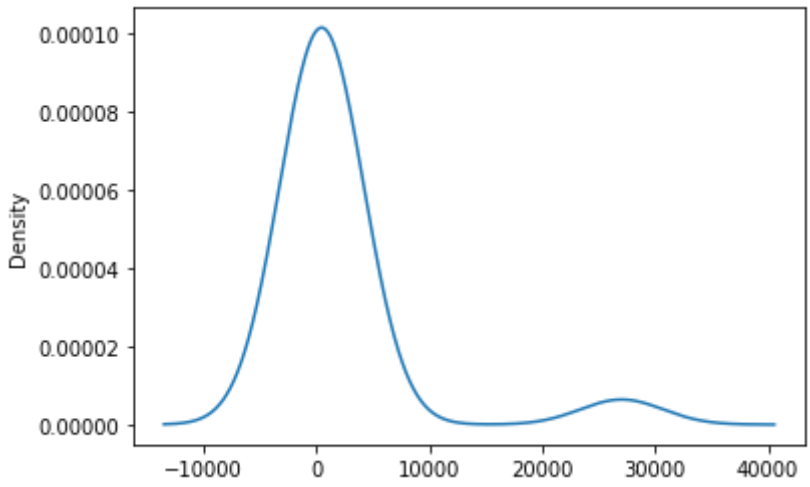
```
# Look at APPLICATION_TYPE value counts for binning
application_counts = application_df.APPLICATION_TYPE.value_counts()
application_counts
```

```
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
```

```
# Visualize the value counts of APPLICATION_TYPE
```

```
application_counts.plot.density()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff29ce69a10>
```




```
# Determine which values to replace if counts are less than ...?
replace_application = list(application_counts[application_counts < 200].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
```

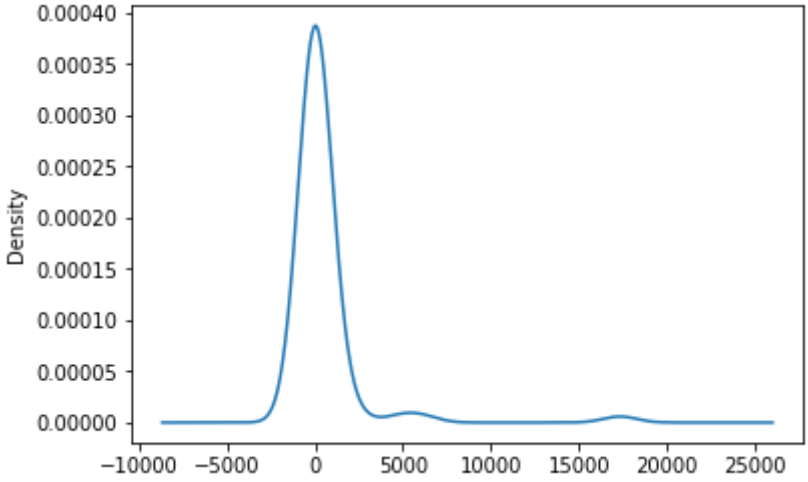
```
# Look at CLASSIFICATION value counts for binning
classification_counts = application_df.CLASSIFICATION.value_counts()
classification_counts
```

```
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
```

```
# Visualize the value counts of CLASSIFICATION
classification_counts.plot.density()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff29cd4aa90>
```




```
# Determine which values to replace if counts are less than ..?
replace_class = list(classification_counts[classification_counts < 1800].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
```

```
application_df.dtypes
```

```
APPLICATION_TYPE          object
AFFILIATION               object
CLASSIFICATION            object
USE_CASE                  object
ORGANIZATION              object
STATUS                     int64
INCOME_AMT                object
SPECIAL_CONSIDERATIONS    object
ASK_AMT                    int64
```

```
IS_SUCCESSFUL              int64
dtype: object
```

```
# 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)
```

	APPLICATION_TYPE_Other	APPLICATION_TYPE_T10	APPLICATION_TYPE_T19	APPL
0	0.0	1.0	0.0	
1	0.0	0.0	0.0	
2	0.0	0.0	0.0	
3	0.0	0.0	0.0	
4	0.0	0.0	0.0	

5 rows × 41 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
```

	STATUS	ASK_AMT	IS_SUCCESSFUL	APPLICATION_TYPE_Other	APPLICATION_TYPE_
0	1	5000	1	0.0	
1	1	108590	1	0.0	
2	1	5000	0	0.0	

```
# 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)
```

```
X_train
```

```
array([[1.00000e+00, 5.00000e+03, 0.00000e+00, ..., 0.00000e+00,
        1.00000e+00, 0.00000e+00],
       [1.00000e+00, 5.00000e+03, 0.00000e+00, ..., 0.00000e+00,
        1.00000e+00, 0.00000e+00],
       [1.00000e+00, 5.00000e+03, 0.00000e+00, ..., 0.00000e+00,
        1.00000e+00, 0.00000e+00],
       ...,
       [1.00000e+00, 1.03405e+05, 0.00000e+00, ..., 0.00000e+00,
        1.00000e+00, 0.00000e+00],
       [1.00000e+00, 5.00000e+03, 0.00000e+00, ..., 0.00000e+00,
        1.00000e+00, 0.00000e+00],
       [1.00000e+00, 1.86924e+06, 0.00000e+00, ..., 0.00000e+00,
        1.00000e+00, 0.00000e+00]])
```

```
len(X_train)
```

```
25724
```

Deliverable 2: Compile, Train and Evaluate the Model

```
# Define the model - deep neural net, i.e., the number of input features and hidden n
number_input_features = len(X_train[0])
hidden_nodes_layer1 = 80
hidden_nodes_layer2 = 30

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="relu"))


# 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, 80)                3520

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 5,981
Trainable params: 5,981
Non-trainable params: 0
_________________________________________________________________
```

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

# Define the checkpoint path and filenames
os.makedirs("checkpoints/",exist_ok=True)
checkpoint_path = "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])
```

```
783/804 [============================>.] - ETA: 0s - loss: 0.5361 - accuracy: 0.
Epoch 79: saving model to checkpoints/weights.79.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5369 - accuracy
Epoch 80/100
776/804 [============================>..] - ETA: 0s - loss: 0.5364 - accuracy: 0.
Epoch 80: saving model to checkpoints/weights.80.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5365 - accuracy
Epoch 81/100
797/804 [============================>.] - ETA: 0s - loss: 0.5369 - accuracy: 0.
Epoch 81: saving model to checkpoints/weights.81.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5372 - accuracy
Epoch 82/100
777/804 [============================>..] - ETA: 0s - loss: 0.5355 - accuracy: 0.
Epoch 82: saving model to checkpoints/weights.82.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5367 - accuracy
Epoch 83/100
793/804 [============================>.] - ETA: 0s - loss: 0.5369 - accuracy: 0.
Epoch 83: saving model to checkpoints/weights.83.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5368 - accuracy
Epoch 84/100
780/804 [============================>.] - ETA: 0s - loss: 0.5371 - accuracy: 0.
Epoch 84: saving model to checkpoints/weights.84.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5367 - accuracy
Epoch 85/100
800/804 [============================>.] - ETA: 0s - loss: 0.5365 - accuracy: 0.
Epoch 85: saving model to checkpoints/weights.85.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5369 - accuracy
Epoch 86/100
802/804 [============================>.] - ETA: 0s - loss: 0.5370 - accuracy: 0.
Epoch 86: saving model to checkpoints/weights.86.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5370 - accuracy
Epoch 87/100
780/804 [============================>.] - ETA: 0s - loss: 0.5365 - accuracy: 0.
Epoch 87: saving model to checkpoints/weights.87.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5364 - accuracy
Epoch 88/100
788/804 [============================>.] - ETA: 0s - loss: 0.5369 - accuracy: 0.
Epoch 88: saving model to checkpoints/weights.88.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5364 - accuracy
Epoch 89/100
```

```
Epoch 89/100
792/804 [============================>.] - ETA: 0s - loss: 0.5361 - accuracy: 0.
Epoch 89: saving model to checkpoints/weights.89.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5362 - accuracy
Epoch 90/100
802/804 [============================>.] - ETA: 0s - loss: 0.5368 - accuracy: 0.
Epoch 90: saving model to checkpoints/weights.90.hdf5
804/804 [==============================] - 2s 2ms/step - loss: 0.5366 - accuracy
Epoch 91/100
795/804 [============================>.] - ETA: 0s - loss: 0.5361 - accuracy: 0.
Epoch 91: saving model to checkpoints/weights.91.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5365 - accuracy
Epoch 92/100
799/804 [============================>.] - ETA: 0s - loss: 0.5361 - accuracy: 0.
Epoch 92: saving model to checkpoints/weights.92.hdf5
804/804 [==============================] - 1s 2ms/step - loss: 0.5360 - accuracy
Epoch 93/100
781/804 [============================>.] - ETA: 0s - loss: 0.5355 - accuracy: 0.
Epoch 93: saving model to checkpoints/weights.93.hdf5
```

```
# 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 - 1s - loss: 0.5593 - accuracy: 0.7268 - 590ms/epoch - 2ms/step
Loss: 0.5592514872550964, Accuracy: 0.7267638444900513
```

```
#Save and export your results to an HDF5 file, and name it AlphabetSoupCharity.h5
nn.save("./Resources/AlphabetSoupCharity.h5")
```