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("/content/charity_data.csv")
application_df.head()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-1-1d51bf91914b> in <module>
      7 #  Import and read the charity_data.csv.
      8 import pandas as pd
----> 9 application_df = pd.read_csv("/content/charity_data.csv")
     10 application_df.head()

                                  7 frames
/usr/local/lib/python3.7/dist-packages/pandas/io/common.py in
get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text,
errors, storage_options)
    705                 encoding=ioargs.encoding,
    706                 errors=errors,
--> 707                 newline="",
    708             )
    709         else:

FileNotFoundError: [Errno 2] No such file or directory:
'/content/charity_data.csv'
```

```
# from google.colab import drive
# drive.mount('/content/drive')
```

```
application_df.columns
```

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

```
application_df.dtypes
```

```
EIN                      int64
```

```
NAME                      object
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
```

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

# Print out the Country value counts
application_df.head()
```

	NAME	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	C
0	BLUE KNIGHTS MOTORCYCLE CLUB	T10	Independent	C1000	ProductDev	
1	AMERICAN CHESAPEAKE CLUB CHARITABLE TR	T3	Independent	C2000	Preservation	
2	ST CLOUD PROFESSIONAL FIREFIGHTERS	T5	CompanySponsored	C3000	ProductDev	
3	SOUTHSIDE ATHLETIC ASSOCIATION	T3	CompanySponsored	C2000	Preservation	
4	GENETIC RESEARCH INSTITUTE OF THE DESERT	T3	Independent	C1000	Heathcare	

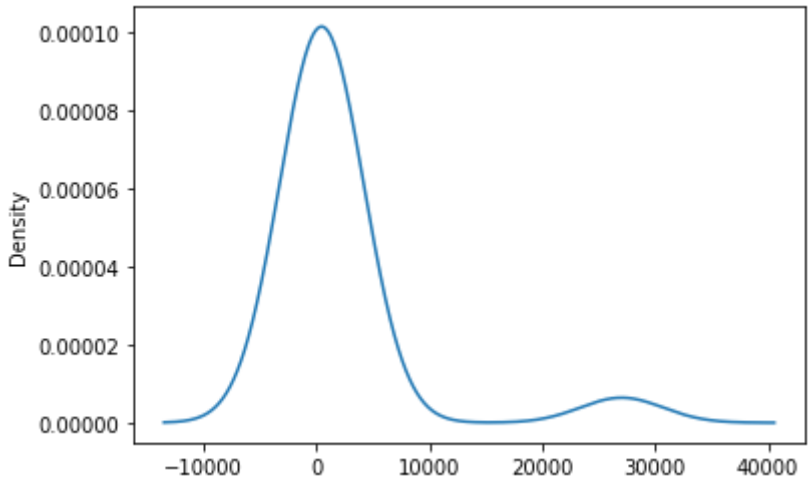
```
application_counts = application_df['APPLICATION_TYPE'].value_counts()
application_counts
```

```
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
```

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fde48054890>
```




```
# Determine which values to replace if counts are less than ...?
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,"Otl

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

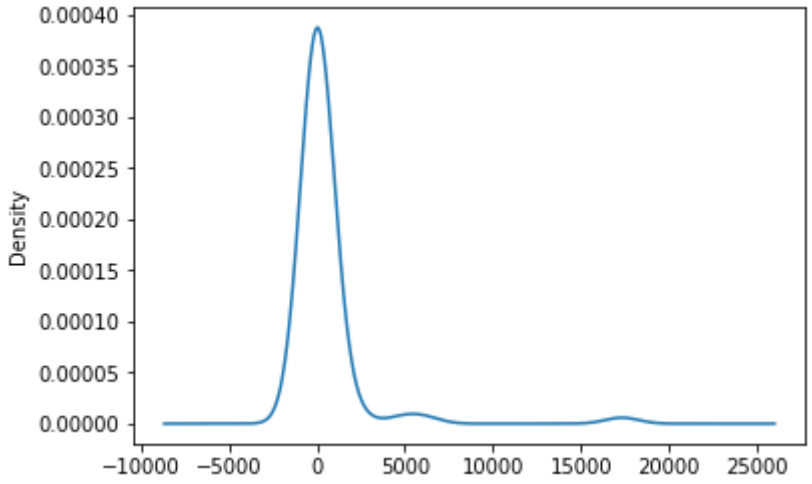
```
T3       27037
T4        1542
T6        1216
T5        1173
T19       1065
T8         737
T7         725
T10        528
Other      276
Name: APPLICATION_TYPE, dtype: int64
```

```
# Look at CLASSIFICATION value counts for binning
#  YOUR CODE GOES HERE
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
```

```
# Visualize the value counts of CLASSIFICATION
#  YOUR CODE GOES HERE
class_counts.plot.density()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fde47f7a4d0>
```




```
# Determine which values to replace if counts are less than ..?
#  YOUR CODE GOES HERE
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
```

```
application_df.dtypes
```

```
NAME                      object
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
```

```
application_cat=application_df.dtypes[application_df.dtypes == "object"].index.tolist(
application_cat
```

```
['NAME',
 'APPLICATION_TYPE',
 'AFFILIATION',
 'CLASSIFICATION',
 'USE_CASE',
 'ORGANIZATION',
 'INCOME_AMT',
 'SPECIAL_CONSIDERATIONS']
```

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

# Fit and transform the OneHotEncoder using the categorical variable list
#  YOUR CODE GOES HERE
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: FutureWa
  warnings.warn(msg, category=FutureWarning)
```

	NAME_1 DAY RANCH RESCUE AND RURAL OKLAHOMA ANIMAL RESOURCE INC	NAME_100 BLACK MEN OF AMERICA	NAME_100 BLACK MEN OF MEMPHIS INC	NAME_100 BLACK MEN OF WEST GEORGIA INC	NAME_1150 WEBSTER STREET INC	NAME_116TH CAVALRY REGIMENT CHAPTER OF THE US CAVALRY & ARMOR ASSOCIATION	NAME_13TH BOMB SQUADRON ASSOCIATION	N AS
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

```
# 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()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-acb515938c55> in <module>
      1 # Merge one-hot encoded features and drop the originals
----> 2
application_df=application_df.merge(encode_df,left_index=True,right_index=True)
      3 application_df=application_df.drop(application_cat,1)
      4 application_df.head()

NameError: name 'application_df' is not defined
```

SEARCH STACK OVERFLOW

```
# 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=42)
```

```
/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 until
```

```
# 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)
```

Deliverable 2: Compile, Train and Evaluate the Model

```
# Define the model - deep neural net, i.e., the number of input features and hidden nc
#  YOUR CODE GOES HERE

number_input_features = len(X_train[0])
hidden_nodes_layer1 = 8
hidden_nodes_layer2 = 5
nn = tf.keras.models.Sequential()

# First hidden layer
#  YOUR CODE GOES HERE
nn.add(
    tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features,
)
# Second hidden layer
#  YOUR CODE GOES HERE
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))

# Output layer
#  YOUR CODE GOES HERE
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, 8)                 352

 dense_1 (Dense)             (None, 5)                 45

 dense_2 (Dense)             (None, 1)                 6

=================================================================
Total params: 403
Trainable params: 403
Non-trainable params: 0
_________________________________________________________________
```

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

# Train the model
#  YOUR CODE GOES HERE
fit_model = nn.fit(X_train,y_train,epochs=100)
```

```
Epoch 1/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 2/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 3/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 4/200
804/804 [==============================] - 1s 2ms/step - loss: 0.6912 - accur
Epoch 5/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 6/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 7/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 8/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 9/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 10/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 11/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 12/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 13/200
804/804 [==============================] - 1s 2ms/step - loss: 0.6912 - accur
Epoch 14/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 15/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 16/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 17/200
804/804 [==============================] - 1s 2ms/step - loss: 0.6912 - accur
Epoch 18/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 19/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 20/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 21/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 22/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 23/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6911 - accur
Epoch 24/200
```

```
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 25/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 26/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 27/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
Epoch 28/200
804/804 [==============================] - 2s 2ms/step - loss: 0.6912 - accur
```

```
# 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: 1.2187 - accuracy: 0.5345 - 500ms/epoch - 2ms/step
Loss: 1.2186620235443115, Accuracy: 0.5344606637954712
```

```
nn.save("AlphabetSoupCharity.h5")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-3f97cdb2be7e> in <module>
----> 1 nn.save("AlphabetSoupCharity.h5")

NameError: name 'nn' is not defined
```

SEARCH STACK OVERFLOW