Preprocessing

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

# Import pandas and read the charity_data.csv from the provided cloud URL.
import pandas as pd
application_df = pd.read_csv("https://static.bc-edx.com/data/dl-1-2/m21/lms/starter
application_df.head()
```

```
2025-05-03 10:30:31.940678: I tensorflow/core/platform/cpu_feature_guard.cc:21
To enable the following instructions: AVX2 FMA, in other operations, rebuild T
```

	EIN	NAME	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION
0	10520599	BLUE KNIGHTS MOTORCYCLE CLUB	T10	Independent	C1000
1	10531628	AMERICAN CHESAPEAKE CLUB CHARITABLE TR	T3	Independent	C2000
2	10547893	ST CLOUD PROFESSIONAL FIREFIGHTERS	T5	CompanySponsored	C3000

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

	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZAT
0	T10	Independent	C1000	ProductDev	Associa
1	T3	Independent	C2000	Preservation	Co-opera
2	T5	CompanySponsored	C3000	ProductDev	Associa
3	T3	CompanySponsored	C2000	Preservation	T
4	T3	Independent	C1000	Heathcare	T
...	...	...	...	...	
34294	T4	Independent	C1000	ProductDev	Associa
34295	T4	CompanySponsored	C3000	ProductDev	Associa
34296	T3	CompanySponsored	C2000	Preservation	Associa
34297	T5	Independent	C3000	ProductDev	Associa
34298	T3	Independent	C1000	Preservation	Co-opera

```
# 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 to identify and replace with "Other"
type_counts = application_df['APPLICATION_TYPE'].value_counts()
type_counts
```

```
APPLICATION_TYPE
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: count, dtype: int64
```

```
# Choose a cutoff value and create a list of application types to be replaced
application_types_to_replace = list(type_counts[type_counts < 500].index)


# Replace in dataframe
for app in application_types_to_replace:
    application_df['APPLICATION_TYPE'] = application_df['APPLICATION_TYPE'].repla

# Check to make sure replacement was successful
application_df['APPLICATION_TYPE'].value_counts()
```

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

```
# Look at CLASSIFICATION value counts to identify and replace with "Other"
class_counts = application_df['CLASSIFICATION'].value_counts()
class_counts
```

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

```
# You may find it helpful to look at CLASSIFICATION value counts >1
class_counts_gt1 = class_counts.loc[class_counts > 1]
class_counts_gt1
```

```
CLASSIFICATION
C1000    17326
C2000     6074
C1200     4837
C3000     1918
C2100     1883
C7000      777
C1700      287
C4000      194
C5000      116
C1270      114
C2700      104
C2800       95
C7100       75
C1300       58
C1280       50
C1230       36
C1400       34
C7200       32
C2300       32
C1240       30
C8000       20
C7120       18
C1500       16
C1800       15
C6000       15
C1250       14
C8200       11
C1238       10
C1278       10
C1235        9
C1237        9
C7210        7
C2400        6
C1720        6
C4100        6
C1257        5
C1600        5
C1260        3
C2710        3
C0           3
C3200        2
C1234        2
C1246        2
C1267        2
C1256        2
Name: count, dtype: int64
```

```
# Choose a cutoff value and create a list of classifications to be replaced
# Use the variable name `classifications_to_replace`
classifications_to_replace = list(class_counts[class_counts < 1000].index)

# Replace in dataframe
for cls in classifications_to_replace:
    application_df['CLASSIFICATION'] = application_df['CLASSIFICATION'].replace(c

# Check to make sure replacement was successful
application_df['CLASSIFICATION'].value_counts()
```

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

```
# Convert categorical data to numeric with `pd.get_dummies`
#This is essential for preparing data for machine learning models, as most algori
#To convert alphanumeric values into numeric- use LabelEncoder:
application_numeric = pd.get_dummies(application_df)
```

```
# Split our preprocessed data into our features and target arrays
X = application_numeric.drop(['IS_SUCCESSFUL'], axis=1)
y = application_numeric['IS_SUCCESSFUL']
# Split the preprocessed data into a training and testing dataset
# To adjust the split ratio, you can specify test_size
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=58)
```

```
# Create a StandardScaler instances
scaler = StandardScaler() # Learn mean and standard deviation from the training d

# Fit the StandardScaler
# We only fit (.fit()) on training data, NOT testing data! This ensures that the
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)
```

Compile, Train and Evaluate the Model

```
# Define the model - deep neural net
number_input_features = len(X_train_scaled[0])
hidden_nodes_layer1 = 10
hidden_nodes_layer2 = 5

nn = tf.keras.models.Sequential()

# Explicit Input layer
nn.add(tf.keras.layers.Input(shape=(number_input_features,)))

# First hidden layer / "tanh" between -1 and 1 for balanced values
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, activation="tanh"))

# Second hidden layer / "relu" >=0 for efficiency
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))

# Output layer /"sigmoid" between 0 and 1 for binary classification.
nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the model
# Trainable Parameters: 501, means all are actively updated during training.
nn.summary()
```

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                   │ (None, 10)             │           440 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 5)              │            55 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 1)              │             6 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 501 (1.96 KB)
 Trainable params: 501 (1.96 KB)
 Non-trainable params: 0 (0.00 B)
```

```
# Compile the model
# Binary classification problems, where the output is either 0 or 1 (successful f
# "adam" automatically adjusts learning rates, making training more efficient.

nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['Accuracy','Pre
# Train the model
fit_model = nn.fit(X_train_scaled, y_train, epochs=20)
```

```
Epoch 1/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - Accuracy: 0.6471 - Precision: 0.684
Epoch 2/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7282 - Precision: 0.727
Epoch 3/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7262 - Precision: 0.727
Epoch 4/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - Accuracy: 0.7283 - Precision: 0.727
Epoch 5/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7291 - Precision: 0.728
Epoch 6/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7272 - Precision: 0.724
Epoch 7/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7317 - Precision: 0.727
Epoch 8/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7301 - Precision: 0.72
Epoch 9/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7332 - Precision: 0.725
Epoch 10/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7315 - Precision: 0.732
Epoch 11/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7348 - Precision: 0.73
Epoch 12/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7346 - Precision: 0.732
Epoch 13/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7316 - Precision: 0.73
Epoch 14/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7347 - Precision: 0.732
Epoch 15/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7344 - Precision: 0.733
Epoch 16/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7301 - Precision: 0.727
Epoch 17/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7312 - Precision: 0.729
Epoch 18/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7295 - Precision: 0.72
Epoch 19/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7308 - Precision: 0.72
Epoch 20/20
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Accuracy: 0.7294 - Precision: 0.728
```

```
# Evaluate the model using the test data
# Precision: Measures how many 'predicted successes were actually successful'.Hig
# Recall: Measures how many actual 'successes were correctly predicted'.High reca
model_loss, model_accuracy, model_precision, model_recall = nn.evaluate(X_test_sc
```

```
268/268 - 1s - 3ms/step - Accuracy: 0.7298 - Precision: 0.7287 - Recall: 0.78
```

```
# Export our model to HDF5 file
#The .h5 and .hdf5 extensions are actually interchangeable—they refer to the same
filepath = r"/Users/GURU/Desktop/deep-learning-challenge/Deep_Learning_Challenge/
nn.save(filepath)

# Export our model to the native Keras format(INCASE needed)
# filepath = r"/Users/GURU/Desktop/deep-learning-challenge/Deep_Learning_Challeng
# nn.save(filepath)
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
```