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 14:53:05.761265: 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	F
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

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

	NAME	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE
0	BLUE KNIGHTS MOTORCYCLE CLUB	T10	Independent	C1000	ProductDev
1	AMERICAN CHESAPEAKE CLUB CHARITABLE TR	T3	Independent	C2000	Preservation
	ST CLOUD				

```
# Determine the number of unique values in each column.
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 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
```

```
# Look at NAME value counts for binning
name_counts = application_df['NAME'].value_counts()
name_counts
```

```
NAME
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
                                                                        ...
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: count, Length: 19568, dtype: int64
```

```
# Choose a cutoff value and create a list of names to be replaced
names_to_replace = list(name_counts[name_counts < 100].index)

# Replace in dataframe
for name in names_to_replace:
    application_df['NAME'] = application_df['NAME'].replace(
        name, "Other")

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

```
NAME
Other                                                              25987
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
PTA TEXAS CONGRESS                                                   368
SOROPTIMIST INTERNATIONAL OF THE AMERICAS INC                        331
ALPHA PHI SIGMA                                                      313
TOASTMASTERS INTERNATIONAL                                           293
MOST WORSHIPFUL STRINGER FREE AND ACCEPTED MASONS                    287
LITTLE LEAGUE BASEBALL INC                                           277
INTERNATIONAL ASSOCIATION OF LIONS CLUBS                             266
MOMS CLUB                                                            210
INTERNATIONAL ASSOCIATION OF SHEET METAL AIR RAIL & TRANSPORTATION   206
AMERICAN ASSOCIATION OF UNIVERSITY WOMEN                             197
FARMERS EDUCATIONAL AND COOPERATIVE UNION OF AMERICA                 166
KNIGHTS OF COLUMBUS                                                  158
HABITAT FOR HUMANITY INTERNATIONAL INC                               154
TENNESSEE ORDER OF THE EASTERN STAR                                  151
VETERANS OF FOREIGN WARS OF THE UNITED STATES AUXILIARY              144
PTA UTAH CONGRESS                                                    140
THE UNITED STATES PONY CLUBS INC                                     136
CIVITAN INTERNATIONAL                                                131
SIGMA BETA DELTA INC                                                 127
HONOR SOCIETY OF PHI KAPPA PHI                                       107
MONTANA 4-H FOUNDATION INC                                           107
WASHINGTON STATE GRANGE                                              106
UNIVERSITY OF WYOMING                                                105
DEMOLAY INTERNATIONAL                                                104
SERTOMA INC                                                          103
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).astype('int')
application_numeric.head()
```

	STATUS	ASK_AMT	IS_SUCCESSFUL	NAME_ALPHA PHI SIGMA	NAME_AMATEUR ATHLETIC UNION OF THE UNITED STATES INC	NAME_AMERICAN ASSOCIATION OF UNIVERSITY WOMEN	NA INT
0	1	5000	1	0	0	0	
1	1	108590	1	0	0	0	
2	1	5000	0	0	0	0	
3	1	6692	1	0	0	0	
4	1	142590	1	0	0	0	

5 rows × 75 columns

```
# 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)
```

RANDOM TESTING MODEL

```
# RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

# Define and train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=200, random_state=58)
rf_model.fit(X_train, y_train)

# Get feature importance from trained Random Forest model
feature_importance = rf_model.feature_importances_
feature_names = X_train.columns

# Ensure the lengths of feature_importance and feature_names match
if len(feature_importance) == len(feature_names):
    # Convert to DataFrame for sorting
    importance_df = pd.DataFrame({"Feature": feature_names, "Importance": feature
    importance_df = importance_df.sort_values(by="Importance", ascending=False)

    # Display ranked features
    print("Feature Importance Ranking:\n", importance_df)
else:
    print("Error: Mismatch in lengths of feature_importance and feature_names.")
    print(f"Length of feature_importance: {len(feature_importance)}")
    print(f"Length of feature_names: {len(feature_names)}")
```

```
Feature Importance Ranking:
                          Feature  Importance
1                        ASK_AMT    0.339761
42  AFFILIATION_CompanySponsored    0.090564
17                    NAME_Other    0.079711
44       AFFILIATION_Independent    0.073176
59      ORGANIZATION_Association    0.023193
..                           ...         ...
73      SPECIAL_CONSIDERATIONS_Y    0.000358
47          AFFILIATION_Regional    0.000265
0                         STATUS    0.000101
46             AFFILIATION_Other    0.000062
56                USE_CASE_Other    0.000049

[74 rows x 2 columns]
```

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 = 8
hidden_nodes_layer3= 6

nn = tf.keras.models.Sequential()

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

# 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()
```

```
/opt/anaconda3/envs/dev/lib/python3.10/site-packages/keras/src/layers/core/der
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	750
dense_1 (Dense)	(None, 8)	88
dense_2 (Dense)	(None, 6)	54
dense_3 (Dense)	(None, 1)	7

Total params: 899 (3.51 KB)
Trainable params: 899 (3.51 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 after Dropping low importance features
#fit_model = nn.fit(X_train_scaled, y_train, epochs=30)
fit_model = nn.fit(X_train_scaled, y_train, epochs=30, batch_size=16, verbose=2)
```

```
Epoch 1/30
1608/1608 - 6s - 3ms/step - Accuracy: 0.7128 - Precision: 0.6837 - Recall: 0
Epoch 2/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7513 - Precision: 0.7260 - Recall: 0
Epoch 3/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7530 - Precision: 0.7240 - Recall: 0
Epoch 4/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7551 - Precision: 0.7261 - Recall: 0
Epoch 5/30
1608/1608 - 4s - 2ms/step - Accuracy: 0.7547 - Precision: 0.7256 - Recall: 0
Epoch 6/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7565 - Precision: 0.7270 - Recall: 0
Epoch 7/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7565 - Precision: 0.7270 - Recall: 0
Epoch 8/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7575 - Precision: 0.7240 - Recall: 0
Epoch 9/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7584 - Precision: 0.7254 - Recall: 0
Epoch 10/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7578 - Precision: 0.7246 - Recall: 0
Epoch 11/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7572 - Precision: 0.7250 - Recall: 0
Epoch 12/30
1608/1608 - 4s - 2ms/step - Accuracy: 0.7574 - Precision: 0.7225 - Recall: 0
Epoch 13/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7587 - Precision: 0.7258 - Recall: 0
Epoch 14/30
1608/1608 - 4s - 2ms/step - Accuracy: 0.7583 - Precision: 0.7230 - Recall: 0
Epoch 15/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7584 - Precision: 0.7236 - Recall: 0
Epoch 16/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7593 - Precision: 0.7250 - Recall: 0
Epoch 17/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7591 - Precision: 0.7233 - Recall: 0
Epoch 18/30
1608/1608 - 4s - 3ms/step - Accuracy: 0.7599 - Precision: 0.7248 - Recall: 0
```

```
Epoch 19/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7589 - Precision: 0.7233 - Recall: 0
Epoch 20/30
1608/1608 - 4s - 2ms/step - Accuracy: 0.7601 - Precision: 0.7237 - Recall: 0
Epoch 21/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7597 - Precision: 0.7258 - Recall: 0
Epoch 22/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7601 - Precision: 0.7244 - Recall: 0
Epoch 23/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7601 - Precision: 0.7246 - Recall: 0
Epoch 24/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7607 - Precision: 0.7259 - Recall: 0
Epoch 25/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7595 - Precision: 0.7234 - Recall: 0
Epoch 26/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7604 - Precision: 0.7245 - Recall: 0
Epoch 27/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7605 - Precision: 0.7243 - Recall: 0
Epoch 28/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7608 - Precision: 0.7252 - Recall: 0
Epoch 29/30
1608/1608 - 3s - 2ms/step - Accuracy: 0.7594 - Precision: 0.7254 - Recall: 0
```

```
# 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: 0.7579 - Precision: 0.7158 - Recall: 0.905
```

```
# 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 `
```

Double-click (or enter) to edit