In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report,confusion_matrix
```

In [2]:
```
# Import data set csv
#data_set = pd.read_csv('filtered_data_set.csv')
#data_set = pd.read_csv('expanded_filtered_data_set.csv')
data_set = pd.read_csv('expanded_pca_data_set.csv')
```

In [3]:
```
data_set
```

Out[3]:

	0	1	2	3	4	5	6	popularity
0	-0.452822	0.091309	0.244744	0.094262	0.012436	-0.114256	-0.151322	0.0
1	-0.555792	0.265329	0.456953	0.231703	0.331784	0.283464	-0.036913	0.0
2	-0.542268	0.395909	-0.334137	-0.152021	0.005078	0.884785	0.036422	0.0
3	-0.672898	0.436383	0.026988	0.073905	-0.298426	-0.225100	-0.069539	0.0
4	0.097621	1.028055	0.265734	0.001769	-0.277193	-0.329631	-0.083111	0.0
...	...	...	...	...	...	...	...	...
134548	0.035397	-0.527383	-0.201443	-0.090219	0.066222	-0.009346	-0.167107	1.0
134549	-0.021546	-0.459784	-0.219341	-0.166534	-0.010819	0.049580	-0.099907	1.0
134550	0.799701	0.011694	0.210383	-0.229390	0.080026	-0.088477	0.049927	1.0
134551	0.072498	-0.550209	0.383477	0.938321	-0.221875	-0.030451	0.292489	1.0
134552	0.181057	-0.615262	-0.109928	0.672678	-0.442818	0.309596	-0.054959	1.0

134553 rows × 8 columns

In [4]:
```
# Set X and y columns
#X = data_set[['valence','acousticness','danceability','duration_ms','energy','ex
#              'instrumentalness','key','liveness','loudness','mode','speechiness
#y = data_set['popularity'].values

X = data_set[['0','1','2','3','4','5',
             '6']].values
y = data_set['popularity'].values
```

In [5]:

```
# Create the X training and testing set, and Y training and testing set where 70%
# are for the training set and the rest to the testing set.
x_testing_set, x_training_set, y_testing_set, y_training_set = train_test_split(X
                                                                                 s
```

In [6]:

```
# Create model we still construct sequentially
model = Sequential()

# Add dense (every input connected to all units in hidden layer)
# Activation - sigmoid maps between 0 and 1. relu maps to 0 or 1
#model.add(Dense(15, input_dim=13, activation='relu'))
model.add(Dense(15, input_dim=7, activation='relu'))
model.add(Dense(18, activation='relu'))
model.add(Dense(13, activation='relu'))
model.add(Dense(10, activation='relu'))

# Output layer
model.add(Dense(1, activation='sigmoid'))
```

```
WARNING:tensorflow:From D:\Anaconda\lib\site-packages\tensorflow\python\ops\ini
t_ops.py:1251: calling VarianceScaling.__init__ (from tensorflow.python.ops.ini
t_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
```

In [7]:

```
# Compile the model.
# Optimizer - Adam is an efficient optimize to apply gradient descent to the mode
# Metrics - want the accuracy on how the model predicts
model.compile(optimizer='adam', loss='binary_crossentropy',metrics=['accuracy'])
```

```
WARNING:tensorflow:From D:\Anaconda\lib\site-packages\tensorflow\python\ops\nn_
impl.py:180: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.
array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

In [8]:

```
model.fit(x_training_set,y_training_set,epochs=100, batch_size=64)
```

```
Epoch 82/100
94188/94188 [==============================] - 1s 12us/sample - loss: 0.4611
- acc: 0.7892
Epoch 83/100
94188/94188 [==============================] - 1s 11us/sample - loss: 0.4607
- acc: 0.7900
Epoch 84/100
94188/94188 [==============================] - 1s 11us/sample - loss: 0.4607
- acc: 0.7898
Epoch 85/100
94188/94188 [==============================] - 1s 11us/sample - loss: 0.4608
- acc: 0.78950s - loss: 0
Epoch 86/100
94188/94188 [==============================] - 1s 11us/sample - loss: 0.4606
- acc: 0.7897
Epoch 87/100
94188/94188 [==============================] - 1s 11us/sample - loss: 0.4605
- acc: 0.7898
Epoch 88/100
94188/94188 [==============================] - 1s 11us/sample - loss: 0.4605
```

In [9]:

```
# Get the predicted values with the testing set
test_predictions = model.predict(x_testing_set)
```

In [10]:

```
# Resize to series
#test_predictions = pd.Series(test_predictions.reshape(2961,))
test_predictions = pd.Series(test_predictions.reshape(40365,))
```

In [11]:

```
training_score = model.evaluate(x_training_set,y_training_set)
test_score = model.evaluate(x_testing_set,y_testing_set)
print(training_score)
print(test_score)
```

```
94188/94188 [==============================] - 1s 10us/sample - loss: 0.4590 -
acc: 0.7904
40365/40365 [==============================] - 0s 10us/sample - loss: 0.4604 -
acc: 0.7864
[0.4590217236698562, 0.79036605]
[0.46043313260003416, 0.78637433]
```

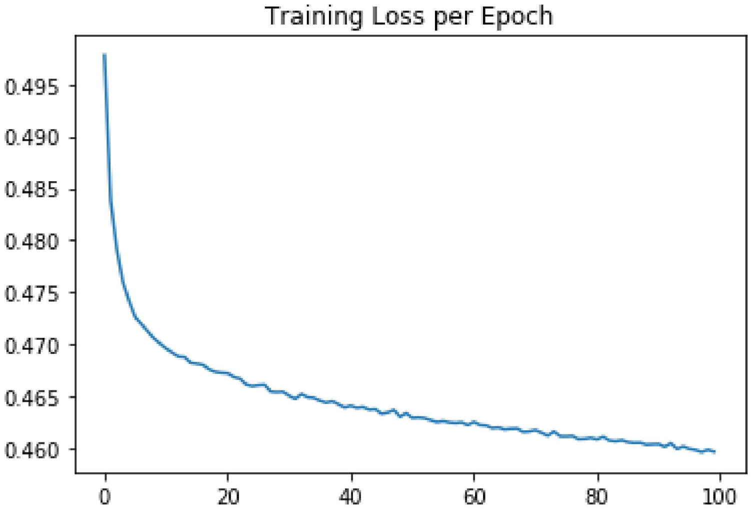
In [12]:

```
# Find predict y values with the x testing set and find accuracy
ynew = model.predict_classes(x_testing_set)
correct=0
for i in range(0,len(ynew)):
    if(ynew[i]==y_testing_set[i]):
        correct = correct + 1
print("Accuracy=", correct/len(test_predictions))
```

```
Accuracy= 0.7863743342004211
```

In [13]:

```
loss = model.history.history['loss']
sns.lineplot(x=range(len(loss)),y=loss)
plt.title("Training Loss per Epoch");
```



In [14]:

```
print(classification_report(y_testing_set, ynew))
```

```
              precision    recall  f1-score   support

         0.0       0.82      0.93      0.87     31087
         1.0       0.56      0.31      0.40      9278

    accuracy                           0.79     40365
   macro avg       0.69      0.62      0.64     40365
weighted avg       0.76      0.79      0.76     40365
```

In [ ]: