Given historical data on loans given out with information on whether or not the borrower defaulted (charge-off), can we build a model that can predict wether or nor a borrower will pay back their loan? This way in the future when we get a new potential customer we can assess whether or not they are likely to pay back the loan. Keep in mind classification metrics when evaluating the performance of your model! source: https://www.kaggle.com/harlfoxem/housesalesprediction

In [1]:

```
import pandas as pd
```

In [2]:

```
df=pd.read_csv((r'C:\Users\chumj\Downloads\Data.csv'))
```

In [3]:

```
df.tail(3)
```

Out[3]:

	Unnamed: 0	loan_amnt	term	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	...	sub_grade_F1	sub_gr
395216	396027	5000.0	36	9.99	161.32	56500.0	17.56	15.0	0.0	32704.0	...	0	
395217	396028	21000.0	60	15.31	503.02	64000.0	15.88	9.0	0.0	15704.0	...	0	
395218	396029	2000.0	36	13.61	67.98	42996.0	8.32	3.0	0.0	4292.0	...	0	

3 rows × 80 columns

In [4]:

```
from sklearn.model_selection import train_test_split
```

In [5]:

```
X=df.drop('repaid_loans',axis=1).values
y=df['repaid_loans'].values
```

In [6]:

```
len(df)
```

Out[6]:

```
395219
```

In [7]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
```

In [8]:

```
# Data normalizing
from sklearn.preprocessing import MinMaxScaler
```

In [9]:

```
scaler=MinMaxScaler()
```

In [10]:

```
X_train=scaler.fit_transform(X_train)
```

In [11]:

```
X_test=scaler.transform(X_test)
```

In [42]:

```
#Creating the model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Activation,Dropout
import warnings
warnings .filterwarnings('ignore')
```

In [13]:

```
X_train.shape
```

Out[13]:

```
(276653, 79)
```

In [14]:

```
model=Sequential()
```

In [15]:

```
model.add(Dense(79,activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(40,activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(20,activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(units=1,activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam')
```

```
WARNING:tensorflow:From C:\Users\chumj\Anaconda3\Ben\lib\site-
packages\tensorflow\python\ops\init_ops.py:1251: calling VarianceScaling.__init__ (from
tensorflow.python.ops.init_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 [16]:

```
model.fit(x=X_train,y=y_train,epochs=40,batch_size=300,validation_data=(X_test,y_test),verbose=1)
```

```
WARNING:tensorflow:From C:\Users\chumj\Anaconda3\Ben\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
Train on 276653 samples, validate on 118566 samples
Epoch 1/40
276653/276653 [==============================] - 8s 28us/sample - loss: 0.3155 - val_loss: 0.2657
Epoch 2/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2666 - val_loss: 0.2628
Epoch 3/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2631 - val_loss: 0.2621
Epoch 4/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2615 - val_loss: 0.2618
Epoch 5/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2609 - val_loss: 0.2617
Epoch 6/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2600 - val_loss: 0.2617
Epoch 7/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2596 - val_loss: 0.2617
```

```
Epoch 8/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2592 - val_loss: 0.2618
Epoch 9/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2591 - val_loss: 0.2619
Epoch 10/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2585 - val_loss: 0.2613
Epoch 11/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2581 - val_loss: 0.2616
Epoch 12/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2581 - val_loss: 0.2617
Epoch 13/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2576 - val_loss: 0.2616
Epoch 14/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2574 - val_loss: 0.2611
Epoch 15/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2574 - val_loss: 0.2615
Epoch 16/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2570 - val_loss: 0.2614
Epoch 17/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2568 - val_loss: 0.2617
Epoch 18/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2566 - val_loss: 0.2613
Epoch 19/40
276653/276653 [==============================] - 7s 26us/sample - loss: 0.2564 - val_loss: 0.2615
Epoch 20/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2561 - val_loss: 0.2616
Epoch 21/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2560 - val_loss: 0.2621
Epoch 22/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2560 - val_loss: 0.2615
Epoch 23/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2558 - val_loss: 0.2617
Epoch 24/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2556 - val_loss: 0.2620
Epoch 25/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2554 - val_loss: 0.2617
Epoch 26/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2550 - val_loss: 0.2616
Epoch 27/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2549 - val_loss: 0.2616
Epoch 28/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2551 - val_loss: 0.2616
Epoch 29/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2546 - val_loss: 0.2616
Epoch 30/40
276653/276653 [==============================] - 8s 27us/sample - loss: 0.2544 - val_loss: 0.2616
Epoch 31/40
276653/276653 [==============================] - 8s 29us/sample - loss: 0.2542 - val_loss: 0.2628
Epoch 32/40
276653/276653 [==============================] - 7s 26us/sample - loss: 0.2540 - val_loss: 0.2627
Epoch 33/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2538 - val_loss: 0.2623
Epoch 34/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2541 - val_loss: 0.2620
Epoch 35/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2535 - val_loss: 0.2632
Epoch 36/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2538 - val_loss: 0.2619
Epoch 37/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2537 - val_loss: 0.2619
Epoch 38/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2534 - val_loss: 0.2622
Epoch 39/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2532 - val_loss: 0.2628
Epoch 40/40
276653/276653 [==============================] - 7s 25us/sample - loss: 0.2531 - val_loss: 0.2624
```

Out[16]:

```
<tensorflow.python.keras.callbacks.History at 0x2878650bcc8>
```

In [17]:

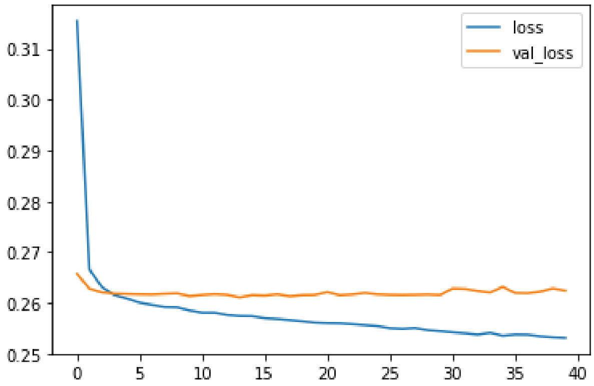
```
loss=pd.DataFrame(model.history.history)
```

In [18]:

```
loss.plot()
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2879a77f888>
```



In [24]:

```
predictions3=model.predict_classes(X_test)
```

In [25]:

```
# Implementing callbacks
#from tensorflow.keras.callbacks import EarlyStopping
```

In [26]:

```
#early=EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=26)
```

In [27]:

```
#model.fit(x=X_train,y=y_train,epochs=400,validation_data=(X_test,y_test),verbose=1,callbacks=[ear
ly])
```

In [23]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

In [28]:

```
print(classification_report(y_test,predictions3))
print(confusion_matrix(y_test,predictions3))
print(accuracy_score(y_test,predictions3))
```

```
              precision    recall  f1-score   support

           0       1.00      0.43      0.60     23363
           1       0.88      1.00      0.93     95203

    accuracy                           0.89    118566
   macro avg       0.94      0.71      0.77    118566
weighted avg       0.90      0.89      0.87    118566

[[ 9958 13405]
 [    0 95203]]
0.886940606919353
```

In [37]:

```
#Given a customer below we you offer loan to the person or not
import random
```

```
random_ind=random.randint(0,len(df))
```

In [38]:

```
new_person=df.drop('repaid_loans',axis=1).iloc[random_ind]
```

In [39]:

```
new_person
```

Out[39]:

```
Unnamed: 0      348599.00
loan_amnt        22400.00
term                36.00
int_rate            12.49
installment        749.26
                  ...
sub_grade_G1         0.00
sub_grade_G2         0.00
sub_grade_G3         0.00
sub_grade_G4         0.00
sub_grade_G5         0.00
Name: 347877, Length: 79, dtype: float64
```

In [40]:

```
model.predict_classes(new_person.values.reshape(1,79))
```

Out[40]:

```
array([[1]])
```

In [41]:

```
#check if this person end up paying the loan
df.iloc[random_ind]['repaid_loans']
```

Out[41]:

```
1.0
```

In [ ]: