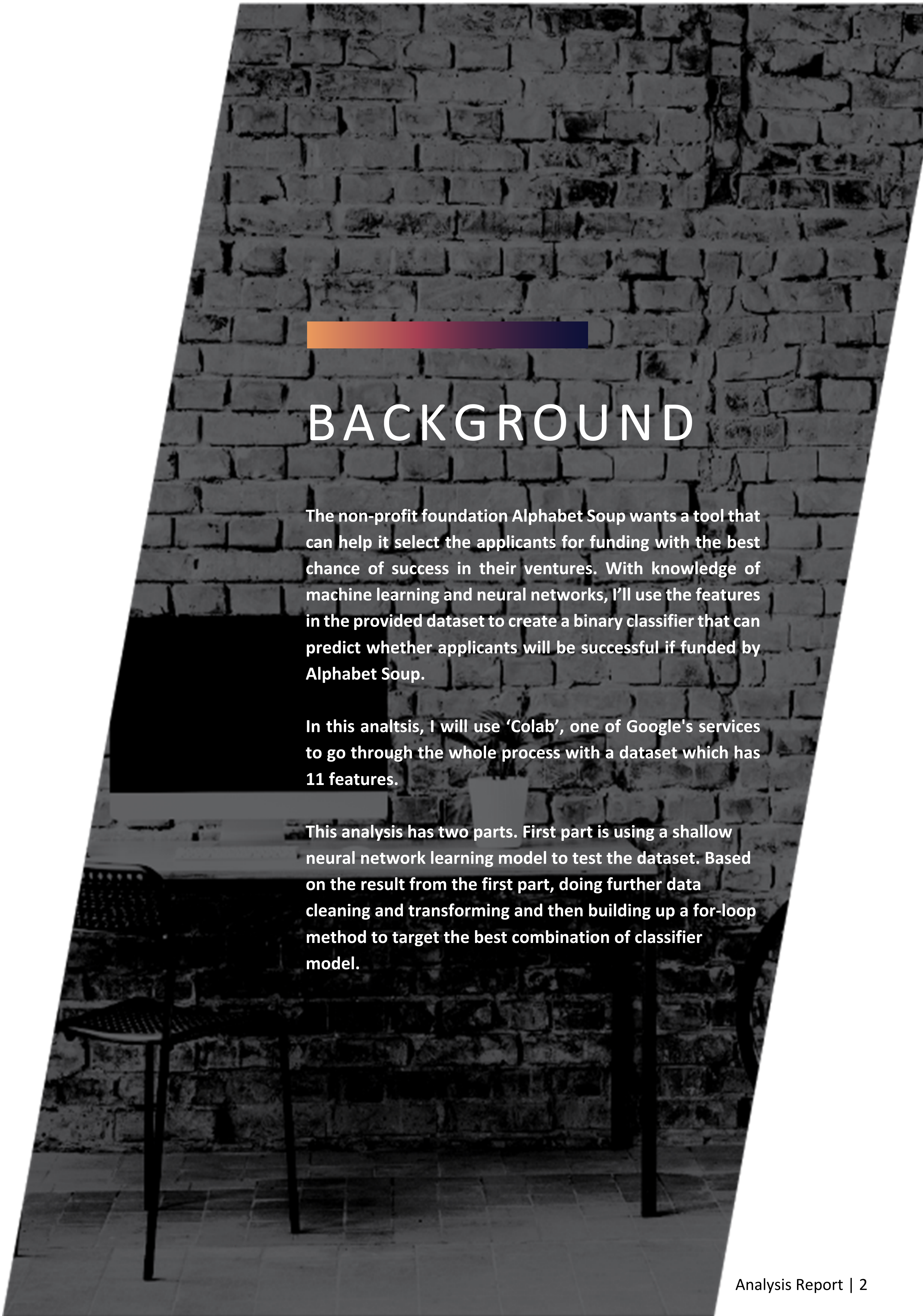
ANALYSIS REPORT

NEURAL NETWORKS M.L

LIN HUAN JHE | lin.linhuanjhe92@gmail.com



BACKGROUND

The non-profit foundation Alphabet Soup wants a tool that can help it select the applicants for funding with the best chance of success in their ventures. With knowledge of machine learning and neural networks, I'll use the features in the provided dataset to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup.

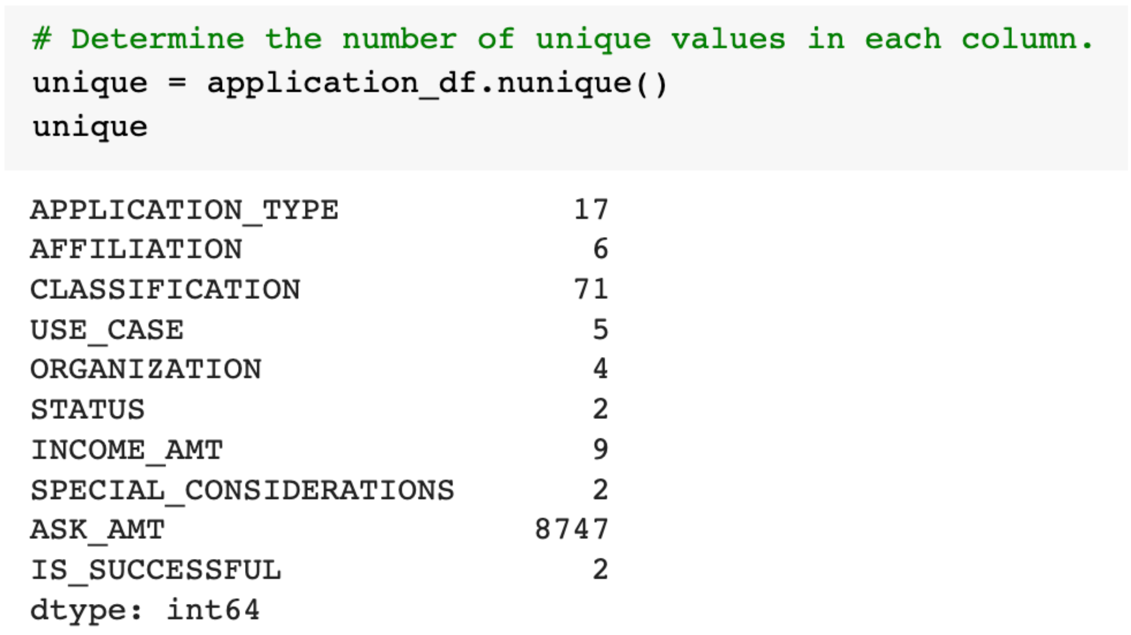
In this analtsis, I will use 'Colab', one of Google's services to go through the whole process with a dataset which has 11 features.

This analysis has two parts. First part is using a shallow neural network learning model to test the dataset. Based on the result from the first part, doing further data cleaning and transforming and then building up a for-loop method to target the best combination of classifier model.

INTRODUCTION

11FEATURES

- EIN and NAME—Identification columns
- APPLICATION_TYPE—Alphabet Soup application type
- AFFILIATION—Affiliated sector of industry
- CLASSIFICATION—Government organisation classification
- USE_CASE—Use case for funding
- ORGANIZATION—Organisation type
- STATUS—Active status
- INCOME_AMT—Income classification
- SPECIAL_CONSIDERATIONS—Special consideration for application
- ASK_AMT—Funding amount requested
- IS_SUCCESSFUL—Was the money used effectively



```
# Determine the number of unique values in each column.
unique = application_df.nunique()
unique
```

```
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
```

DATA TANSFORMATION

DATA PREPROCESSING

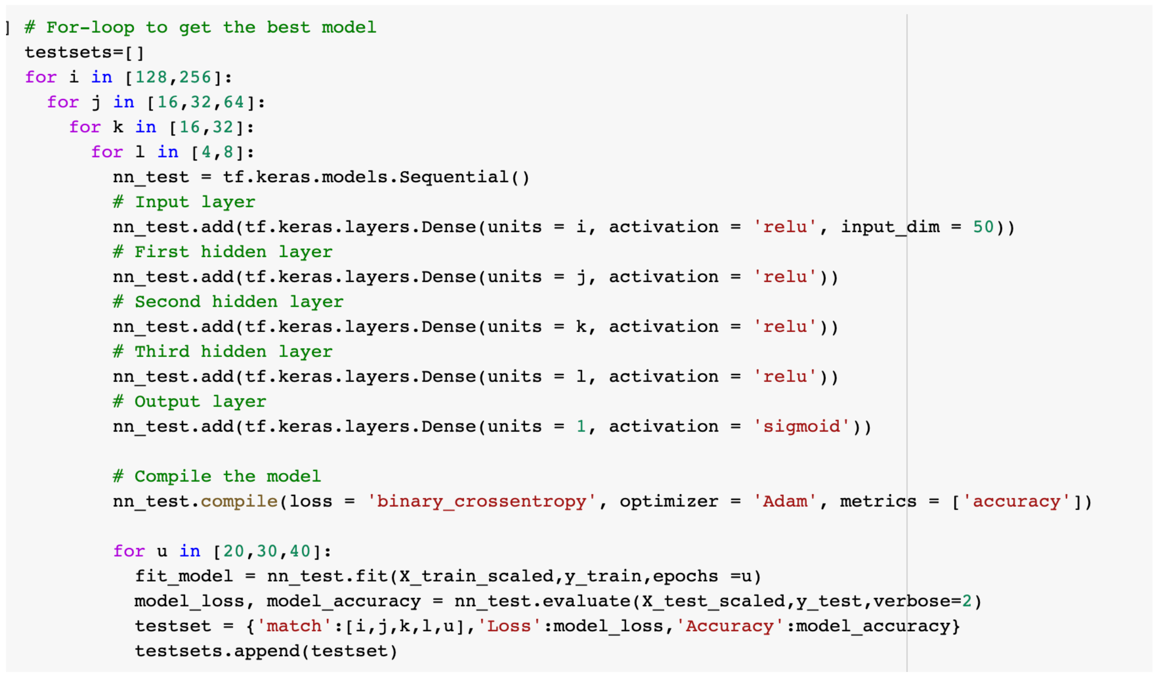
- **What variable(s) are the target(s) for my model?**
  “IS_SUCCESSFUL” is my target.

- **What variable(s) are the features for my model?**
  “APPLICATION_TYPE”,” AFFILIATION**”, “**CLASSIFICATION”,” USE_CASE”,” ORGANIZATION”,” INCOME_AMT”,” SPECIAL_CONSIDERATIONS”,” ASK_AMT” are the features for my model.

- **What variable(s) should be removed from the input data because they are neither targets nor features?**
  EIN and NAME: Drop
  STATUS: Filter “STATUs==1 ’ then Drop whole column

COMPILING, TRAINING, AND EVALUATING THE MODEL

- **How many neurons, layers, and activation functions did I select for my neural network model, and why?**
  I used for-loop method to get a better model instead of blindly changing variables.



```
] # For-loop to get the best model
  testsets=[]
  for i in [128,256]:
    for j in [16,32,64]:
      for k in [16,32]:
        for l in [4,8]:
          nn_test = tf.keras.models.Sequential()
          # Input layer
          nn_test.add(tf.keras.layers.Dense(units = i, activation = 'relu', input_dim = 50))
          # First hidden layer
          nn_test.add(tf.keras.layers.Dense(units = j, activation = 'relu'))
          # Second hidden layer
          nn_test.add(tf.keras.layers.Dense(units = k, activation = 'relu'))
          # Third hidden layer
          nn_test.add(tf.keras.layers.Dense(units = l, activation = 'relu'))
          # Output layer
          nn_test.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))

          # Compile the model
          nn_test.compile(loss = 'binary_crossentropy', optimizer = 'Adam', metrics = ['accuracy'])

          for u in [20,30,40]:
            fit_model = nn_test.fit(X_train_scaled,y_train,epochs =u)
            model_loss, model_accuracy = nn_test.evaluate(X_test_scaled,y_test,verbose=2)
            testset = {'match':[i,j,k,l,u],'Loss':model_loss,'Accuracy':model_accuracy}
            testsets.append(testset)
```

DATA TANSFORMATION

COMPILING, TRAINING, AND EVALUATING THE MODEL

- **Was I able to achieve the target model performance (0.75)?**

  Unfortunately, I couldn't optimize my model to achieve the target accuracy.
  The second model I got is:

```
# 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}")
```

```
215/215 - 0s - loss: 0.5493 - accuracy: 0.7312 - 448ms/epoch - 2ms/step
Loss: 0.5492578744888306, Accuracy: 0.7311561703681946
```

- **What steps did you take in your attempts to increase model performance?**

  1. Removing T12, T2, T25, T14, T29, T15, T17 in the 'APPLICATION_TYPE' column as the number of each record is too less and it may cause confusion in the model.
  2. In the ' CLASSIFICATION' column, dropping where the total record of CLASSIFICATION is less than 20 for the same reason from above.
  3. In the ' STATUS' column, filter where 'STATUS'==1 then drop the column as the number of record which 'STATUS'==0 is 5 while 34294 records of 'STATUS'==0
  4. In the 'INCOME_AMT' column, based on bin range, using numbers to emphasize the different grades between each record.

```
# INCOME_AMT has difference in grades
for i in range (0, len(application_df)):
  if application_df.loc[i,'INCOME_AMT']==0:
    application_df.loc[i,'INCOME_AMT']= 1
  elif application_df.loc[i,'INCOME_AMT']=='1-9999':
    application_df.loc[i,'INCOME_AMT']= 2
  elif application_df.loc[i,'INCOME_AMT']=='10000-24999':
    application_df.loc[i,'INCOME_AMT']= 3
  elif application_df.loc[i,'INCOME_AMT']=='25000-99999':
    application_df.loc[i,'INCOME_AMT']= 4
  elif application_df.loc[i,'INCOME_AMT']=='100000-499999':
    application_df.loc[i,'INCOME_AMT']= 5
  elif application_df.loc[i,'INCOME_AMT']=='1M-5M':
    application_df.loc[i,'INCOME_AMT']= 6
  elif application_df.loc[i,'INCOME_AMT']=='5M-10M':
    application_df.loc[i,'INCOME_AMT']= 7
  elif application_df.loc[i,'INCOME_AMT']=='10M-50M':
    application_df.loc[i,'INCOME_AMT']= 8
  elif application_df.loc[i,'INCOME_AMT']=='50M+':
    application_df.loc[i,'INCOME_AMT']= 9
```

DATA TANSFORMATION

COMPILING, TRAINING, AND EVALUATING THE MODEL

5. In the 'ASK_AMT' column, based on bin range, using numbers to emphasize the different grades between each record.

```
# Bin ASK_AMT value based on 'INCOME_AMT'

for i in range (0, len(application_df)):
  if (application_df.loc[i,'ASK_AMT']>=1)&(application_df.loc[i,'ASK_AMT']<=9999):
    application_df.loc[i,'ASK_AMT']=1
  elif (application_df.loc[i,'ASK_AMT']>=10000)&(application_df.loc[i,'ASK_AMT']<=24999):
    application_df.loc[i,'ASK_AMT']=2
  elif (application_df.loc[i,'ASK_AMT']>=25000)&(application_df.loc[i,'ASK_AMT']<=99999):
    application_df.loc[i,'ASK_AMT']=3
  elif (application_df.loc[i,'ASK_AMT']>=100000)&(application_df.loc[i,'ASK_AMT']<=499999):
    application_df.loc[i,'ASK_AMT']=4
  elif (application_df.loc[i,'ASK_AMT']>=500000)&(application_df.loc[i,'ASK_AMT']<=999999):
    application_df.loc[i,'ASK_AMT']=5
  elif (application_df.loc[i,'ASK_AMT']>=1000000)&(application_df.loc[i,'ASK_AMT']<=4999999):
    application_df.loc[i,'ASK_AMT']=6
  elif (application_df.loc[i,'ASK_AMT']>=5000000)&(application_df.loc[i,'ASK_AMT']<=9999999):
    application_df.loc[i,'ASK_AMT']=7
  elif (application_df.loc[i,'ASK_AMT']>=10000000)&(application_df.loc[i,'ASK_AMT']<=49999999):
    application_df.loc[i,'ASK_AMT']=8
  elif (application_df.loc[i,'ASK_AMT']>=50000000):
    application_df.loc[i,'ASK_AMT']=9
```

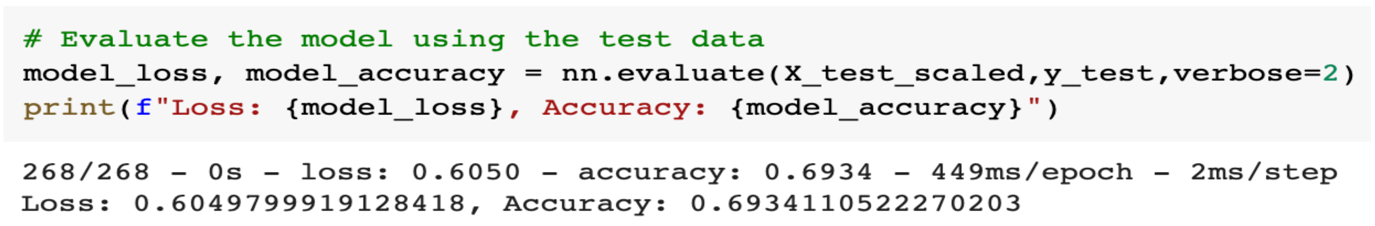
6. Dummied the data to transform non-numeric data to numeric data.
7. I used a for-loop method to go through 72 types of combinations.

```
] # For-loop to get the best model
  testsets=[]
  for i in [128,256]:
    for j in [16,32,64]:
      for k in [16,32]:
        for l in [4,8]:
          nn_test = tf.keras.models.Sequential()
          # Input layer
          nn_test.add(tf.keras.layers.Dense(units = i, activation = 'relu', input_dim = 50))
          # First hidden layer
          nn_test.add(tf.keras.layers.Dense(units = j, activation = 'relu'))
          # Second hidden layer
          nn_test.add(tf.keras.layers.Dense(units = k, activation = 'relu'))
          # Third hidden layer
          nn_test.add(tf.keras.layers.Dense(units = l, activation = 'relu'))
          # Output layer
          nn_test.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))

          # Compile the model
          nn_test.compile(loss = 'binary_crossentropy', optimizer = 'Adam', metrics = ['accuracy'])

          for u in [20,30,40]:
            fit_model = nn_test.fit(X_train_scaled,y_train,epochs =u)
            model_loss, model_accuracy = nn_test.evaluate(X_test_scaled,y_test,verbose=2)
            testset = {'match':[i,j,k,l,u],'Loss':model_loss,'Accuracy':model_accuracy}
            testsets.append(testset)
```

8. The best model is [256, 32, 16, 8] with 20 epochs. Loss:0.544602 and Accuracy:0.735093

SUMMARY

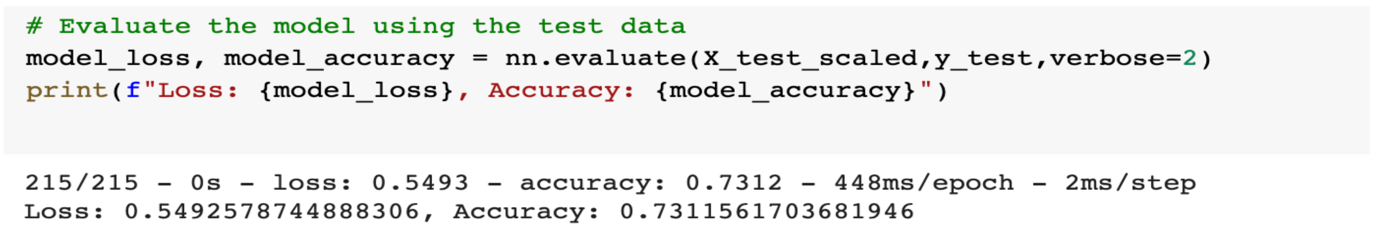
First model:



```
# 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: 0.6050 - accuracy: 0.6934 - 449ms/epoch - 2ms/step
Loss: 0.6049799919128418, Accuracy: 0.6934110522270203
```

Optimized model:



```
# 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}")
```

```
215/215 - 0s - loss: 0.5493 - accuracy: 0.7312 - 448ms/epoch - 2ms/step
Loss: 0.5492578744888306, Accuracy: 0.7311561703681946
```

In general, compared to the result in the first classifier model, the optimized model has higher accuracy rate and lower loss rate. It indicates that the way I optimize the model is on the right track.

When I designed the first model, I was trying to simplify the data and apply a shallow deep learning model. For each non-numeric-data column, I categorized them into two categories, a category which has the highest count numbers and the other. The result was the first picture above.

After I had executed the whole process, I realized that the way I binned data might confuse the model as the organization did not simply use dichotomy to assess each applicant. Therefore, I demonstrated how I optimized my model on page 4~page 6. The final result is shown in the second picture above.

This dataset is the records about an organization following their existing procedures and established requirements to assess the applicants who can get support or not. However, the deep-learning model is based on supervised machine learning and normally using unstructured data. For example, to classify cats and dogs.

In my opinion, supervised machine learning might be a better way to create a classifier. As this data set has 10 features and 50 columns after executing the dummy method, applying the “Random Forest Classifier Model” might be a suitable option to build up a classifier.