```
This is a Glass Identification Data Set from UCI. It contains 10 attributes including id.
The response is glass type(discrete 7 values)



Attribute Information:

Id number: 1 to 214 (removed from CSV file)
RI: refractive index
Na: Sodium (unit measurement: weight percent in the corresponding oxide, as attributes 4-
10)
Mg: Magnesium
Al: Aluminum
Si: Silicon
K: Potassium
Ca: Calcium
Ba: Barium
Fe: Iron
Type of glass: (class attribute)
-- 1 buildingwindowsfloatprocessed

-- 2 buildingwindowsnonfloatprocessed

-- 3 vehiclewindowsfloatprocessed
-- 4 vehiclewindowsnonfloatprocessed (none in this database)
-- 5 containers
-- 6 tableware
-- 7 headlamps



Perform multi-class classification using neural networks to predict glass type.
```

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [2]:

```
df = pd.read_csv("glass.csv")
```

In [3]:

```
df
```

Out[3]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...	...	...	...	...	...	...	...	...	...	...
209	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7
213	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows × 10 columns

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
RI      0
Na      0
Mg      0
Al      0
Si      0
K       0
Ca      0
Ba      0
Fe      0
Type    0
dtype: int64
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   RI      214 non-null    float64
 1   Na      214 non-null    float64
 2   Mg      214 non-null    float64
 3   Al      214 non-null    float64
 4   Si      214 non-null    float64
 5   K       214 non-null    float64
 6   Ca      214 non-null    float64
 7   Ba      214 non-null    float64
 8   Fe      214 non-null    float64
 9   Type    214 non-null    int64  
dtypes: float64(9), int64(1)
memory usage: 16.8 KB
```

In [6]:

```
df['Type'].value_counts()
```

Out[6]:

```
2    76
1    70
7    29
3    17
5    13
6     9
Name: Type, dtype: int64
```

In [7]:

```
x = df.drop("Type", axis=1)
y = df["Type"]
```

In [8]:

```
x.shape[1]
```

Out[8]:

```
9
```

In [9]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, stratify=y) #strat
```

In [10]:

```
y_train.value_counts()
```

Out[10]:

```
2    53
1    49
7    20
3    12
5     9
6     6
Name: Type, dtype: int64
```

In [11]:

```
y_test.value_counts()
```

Out[11]:

```
2    23
1    21
7     9
3     5
5     4
6     3
Name: Type, dtype: int64
```

In [12]:

```
model = tf.keras.Sequential(
    [tf.keras.layers.Dense(2, activation="relu", input_shape=(x.shape[1],)),
     tf.keras.layers.Dense(5, activation="relu"),
     tf.keras.layers.Dense(8, activation="softmax")])
```

In [13]:

```
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy")
```

In [14]:

```
model.fit(x_train, y_train, epochs=50, batch_size=10)
```

```
Epoch 1/50
15/15 [==============================] - 0s 641us/step - loss: 1.9145
Epoch 2/50
15/15 [==============================] - 0s 712us/step - loss: 1.8589
Epoch 3/50
15/15 [==============================] - 0s 926us/step - loss: 1.8607
Epoch 4/50
15/15 [==============================] - 0s 1ms/step - loss: 1.8750
Epoch 5/50
15/15 [==============================] - 0s 997us/step - loss: 1.8636
Epoch 6/50
15/15 [==============================] - 0s 997us/step - loss: 1.7554
Epoch 7/50
15/15 [==============================] - 0s 712us/step - loss: 1.6817
Epoch 8/50
15/15 [==============================] - 0s 855us/step - loss: 1.7441
Epoch 9/50
15/15 [==============================] - 0s 926us/step - loss: 1.6032
Epoch 10/50
15/15 [==============================] - 0s 1ms/step - loss: 1.5926
Epoch 11/50
15/15 [==============================] - 0s 855us/step - loss: 1.7620
Epoch 12/50
15/15 [==============================] - 0s 855us/step - loss: 1.6673
Epoch 13/50
15/15 [==============================] - 0s 1ms/step - loss: 1.6209
Epoch 14/50
15/15 [==============================] - 0s 855us/step - loss: 1.7562
Epoch 15/50
15/15 [==============================] - 0s 926us/step - loss: 1.6507
Epoch 16/50
15/15 [==============================] - 0s 855us/step - loss: 1.6781
Epoch 17/50
15/15 [==============================] - 0s 642us/step - loss: 1.6095
Epoch 18/50
15/15 [==============================] - 0s 641us/step - loss: 1.7569
Epoch 19/50
15/15 [==============================] - 0s 712us/step - loss: 1.6316
Epoch 20/50
15/15 [==============================] - 0s 712us/step - loss: 1.6577
Epoch 21/50
15/15 [==============================] - 0s 926us/step - loss: 1.6368
Epoch 22/50
15/15 [==============================] - 0s 855us/step - loss: 1.6903
Epoch 23/50
15/15 [==============================] - 0s 926us/step - loss: 1.6575
Epoch 24/50
15/15 [==============================] - 0s 997us/step - loss: 1.6210
Epoch 25/50
15/15 [==============================] - 0s 926us/step - loss: 1.5100
Epoch 26/50
15/15 [==============================] - 0s 926us/step - loss: 1.5592
Epoch 27/50
15/15 [==============================] - 0s 926us/step - loss: 1.6997
Epoch 28/50
15/15 [==============================] - 0s 784us/step - loss: 1.5758
```

```
Epoch 29/50
15/15 [==============================] - 0s 641us/step - loss: 1.7311
Epoch 30/50
15/15 [==============================] - 0s 641us/step - loss: 1.6039
Epoch 31/50
15/15 [==============================] - 0s 570us/step - loss: 1.5009
Epoch 32/50
15/15 [==============================] - 0s 997us/step - loss: 1.5330
Epoch 33/50
15/15 [==============================] - 0s 927us/step - loss: 1.5112
Epoch 34/50
15/15 [==============================] - 0s 784us/step - loss: 1.5507
Epoch 35/50
15/15 [==============================] - 0s 926us/step - loss: 1.6563
Epoch 36/50
15/15 [==============================] - 0s 926us/step - loss: 1.6524
Epoch 37/50
15/15 [==============================] - 0s 926us/step - loss: 1.6098
Epoch 38/50
15/15 [==============================] - 0s 926us/step - loss: 1.6178
Epoch 39/50
15/15 [==============================] - 0s 855us/step - loss: 1.5158
Epoch 40/50
15/15 [==============================] - 0s 926us/step - loss: 1.5082
Epoch 41/50
15/15 [==============================] - 0s 926us/step - loss: 1.5936
Epoch 42/50
15/15 [==============================] - ETA: 0s - loss: 1.673 - 0s 712us/
step - loss: 1.5526
Epoch 43/50
15/15 [==============================] - 0s 784us/step - loss: 1.5418
Epoch 44/50
15/15 [==============================] - 0s 641us/step - loss: 1.6275
Epoch 45/50
15/15 [==============================] - 0s 784us/step - loss: 1.4866
Epoch 46/50
15/15 [==============================] - 0s 1ms/step - loss: 1.6128
Epoch 47/50
15/15 [==============================] - 0s 1ms/step - loss: 1.6386
Epoch 48/50
15/15 [==============================] - 0s 1ms/step - loss: 1.4447
Epoch 49/50
15/15 [==============================] - 0s 1ms/step - loss: 1.4740
Epoch 50/50
15/15 [==============================] - 0s 997us/step - loss: 1.5939
```

Out[14]:

```
<tensorflow.python.keras.callbacks.History at 0x270f1290040>
```

In [15]:

```
y_hat = model.predict(x_test)
```

In [16]:

```
y_hat
```

Out[16]:

```
array([[0.01783166, 0.3342196 , 0.3650139 , 0.06897898, 0.00692977,
        0.04720223, 0.02509435, 0.13472947],
       [0.02087367, 0.32366467, 0.3497539 , 0.07579865, 0.00873555,
        0.05342462, 0.02988615, 0.13786294],
       [0.01931141, 0.32904357, 0.35744807, 0.07236549, 0.00778955,
        0.05026086, 0.02741337, 0.13636765],
       [0.01900035, 0.33012486, 0.35901487, 0.07166486, 0.00760587,
        0.04962305, 0.02692398, 0.13604213],
       [0.01747189, 0.33549076, 0.36689854, 0.06813447, 0.00672613,
        0.04644896, 0.02453405, 0.13429528],
       [0.02261776, 0.31775552, 0.34147674, 0.07947318, 0.00983739,
        0.05688262, 0.03267514, 0.13928184],
       [0.03060175, 0.29179484, 0.30684727, 0.09448954, 0.01548534,
        0.07184162, 0.04579498, 0.14314467],
       [0.01984082, 0.32721123, 0.35480878, 0.07354465, 0.00810576,
        0.05134029, 0.02824858, 0.13689984],
       [0.0193777 , 0.32881352, 0.35711572, 0.07251405, 0.0078289 ,
        0.05039645, 0.0275178 , 0.13643578],
```

In [17]:

```
# return index number of the maximum value
y_hat1 = y_hat.argmax(axis=1)
```

In [18]:

```
y_hat1
```

Out[18]:

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
      dtype=int64)
```

In [19]:

```
from sklearn.metrics import classification_report
```

In [20]:

```
print(classification_report(y_test, y_hat1))
```

```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        21
           2       0.35      1.00      0.52        23
           3       0.00      0.00      0.00         5
           5       0.00      0.00      0.00         4
           6       0.00      0.00      0.00         3
           7       0.00      0.00      0.00         9

    accuracy                           0.35        65
   macro avg       0.06      0.17      0.09        65
weighted avg       0.13      0.35      0.18        65
```

In [ ]:

```

```