Machine Learning Presentation

HANDWRITTEN CHARACTER RECOGNITION

AVANI PAL
APURVA PANCHAL
PRAGATI VISHWAKARMA

OUTLINE

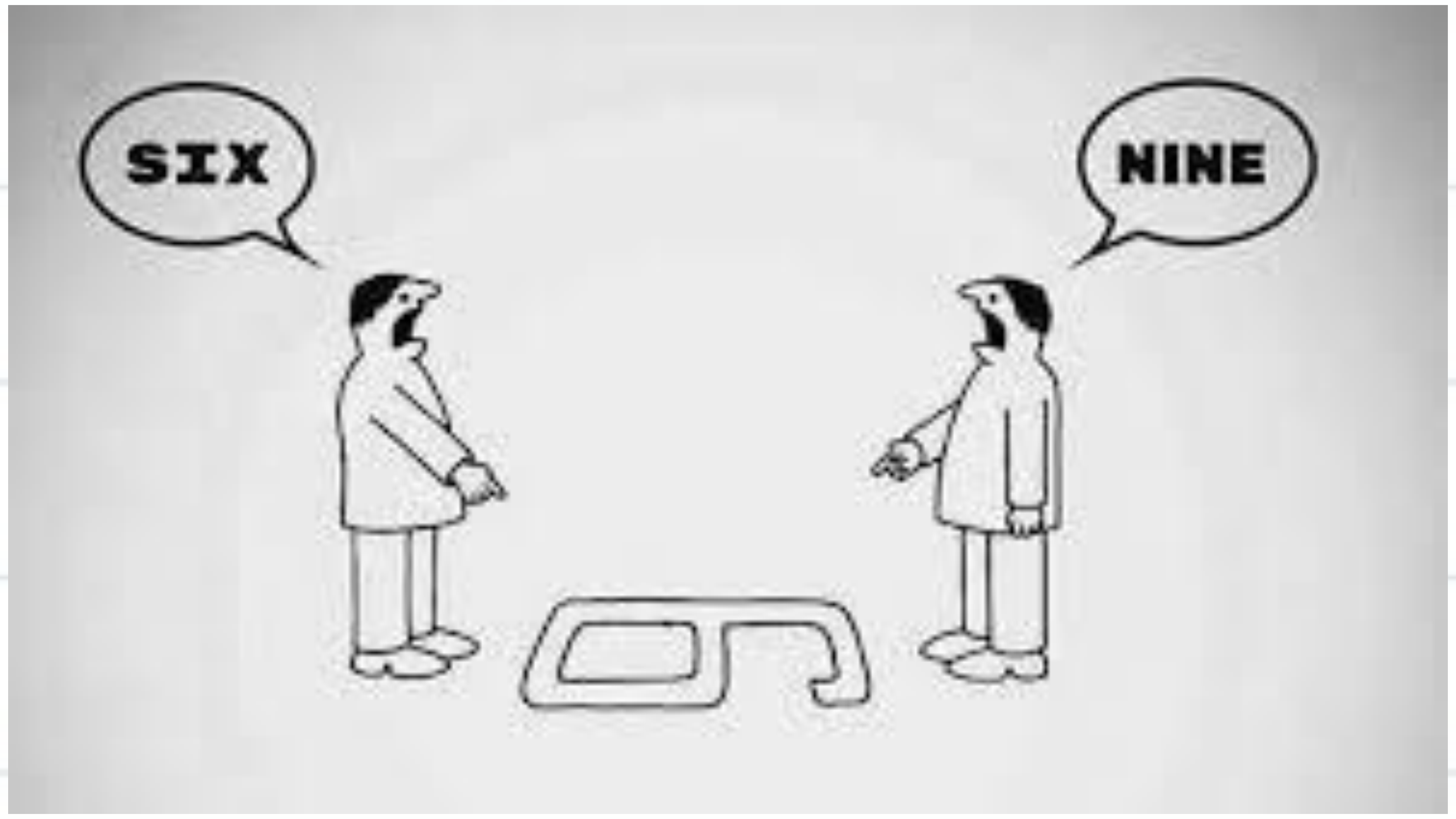
WHY HANDWRITTEN DIGIT RECOGNITION?

- Handwritten digit recognition is the ability of a computer to recognise the human handwritten digits from different sources like images, papers, touch screens, etc., and classify them into 10 predefined classes (0-9).

- Digit recognition has many applications like number plate recognition, postal mail sorting, bank check processing, etc

- In Handwritten digit recognition, we face many challenges because of different styles of writing of different peoples as it is not an Optical character recognition.

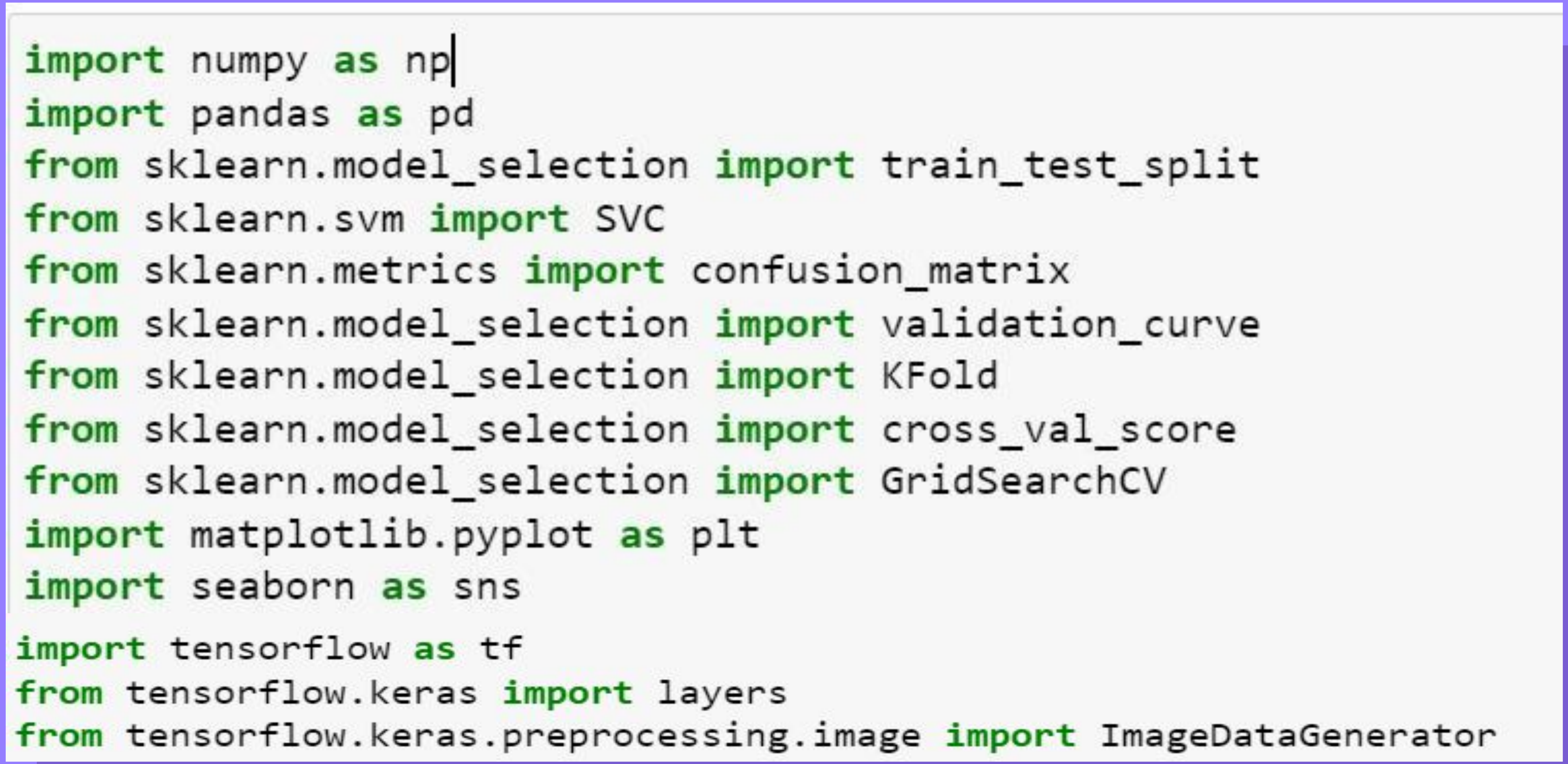
PROBLEM STATEMENT



It is easy for the human to perform a task accurately by practicing it repeatedly and memorising it for the next time. Human brain can process and analyse images easily. Also, recognise the different elements present in the images.

The goal is to correctly identify digits from a dataset of tens of thousands of handwritten images and experiment with different algorithms to learn first-hand what works well and how techniques compare.

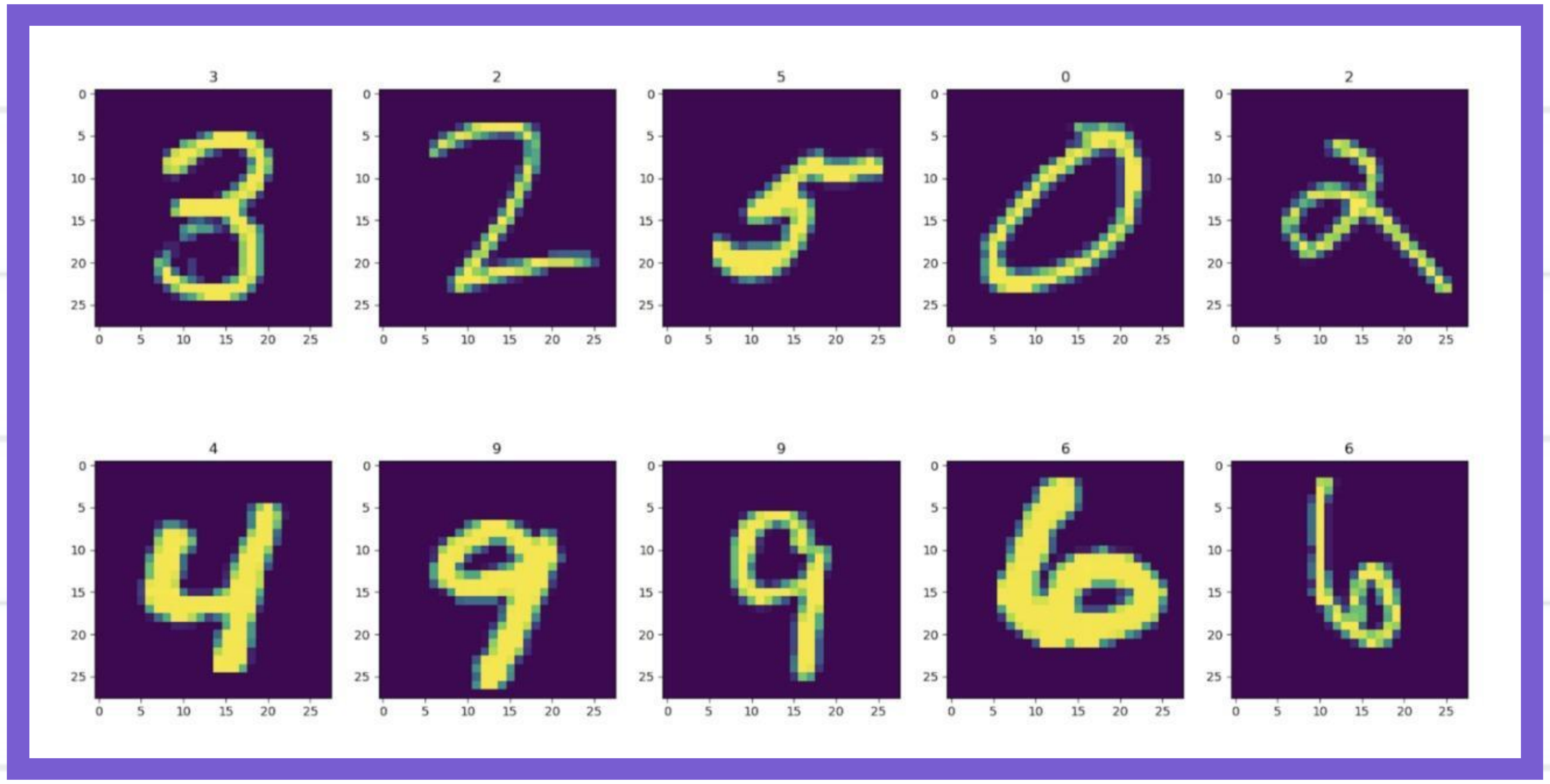
LIBRARIES USED



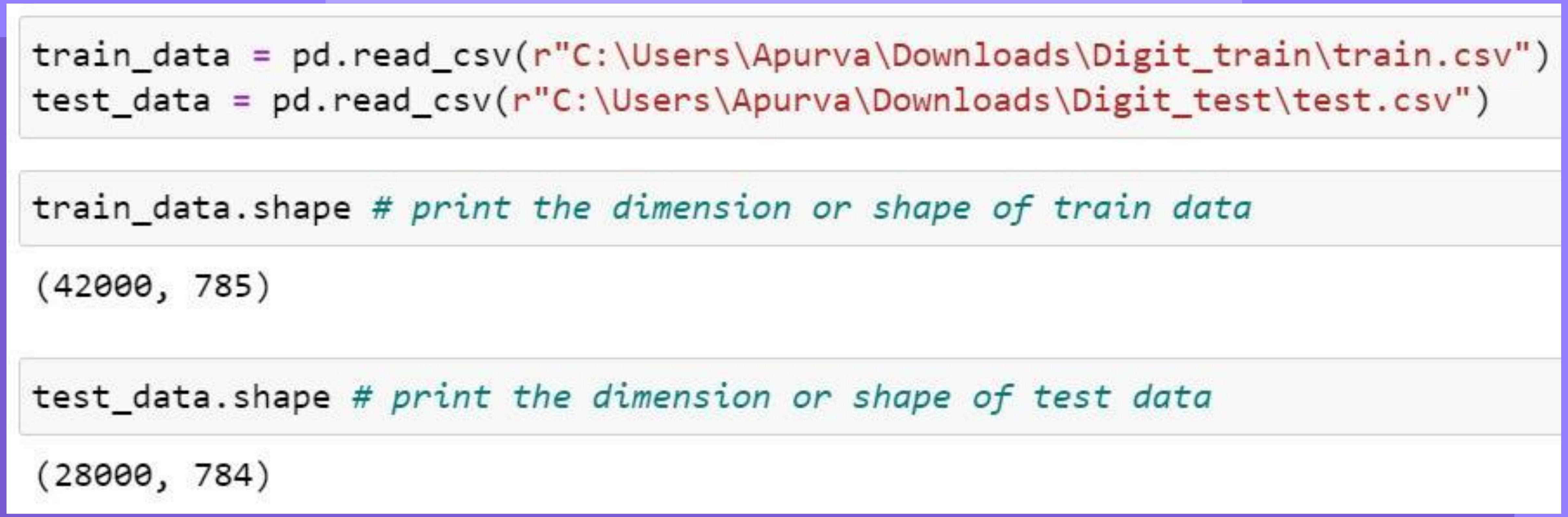
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import validation_curve
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

DATASET DESCRIPTION

- Data Description For this problem, we use the MNIST data which is a large database of handwritten digits. The 'pixel values' of each digit (image) comprise the features, and the actual number between 0-9 is the label

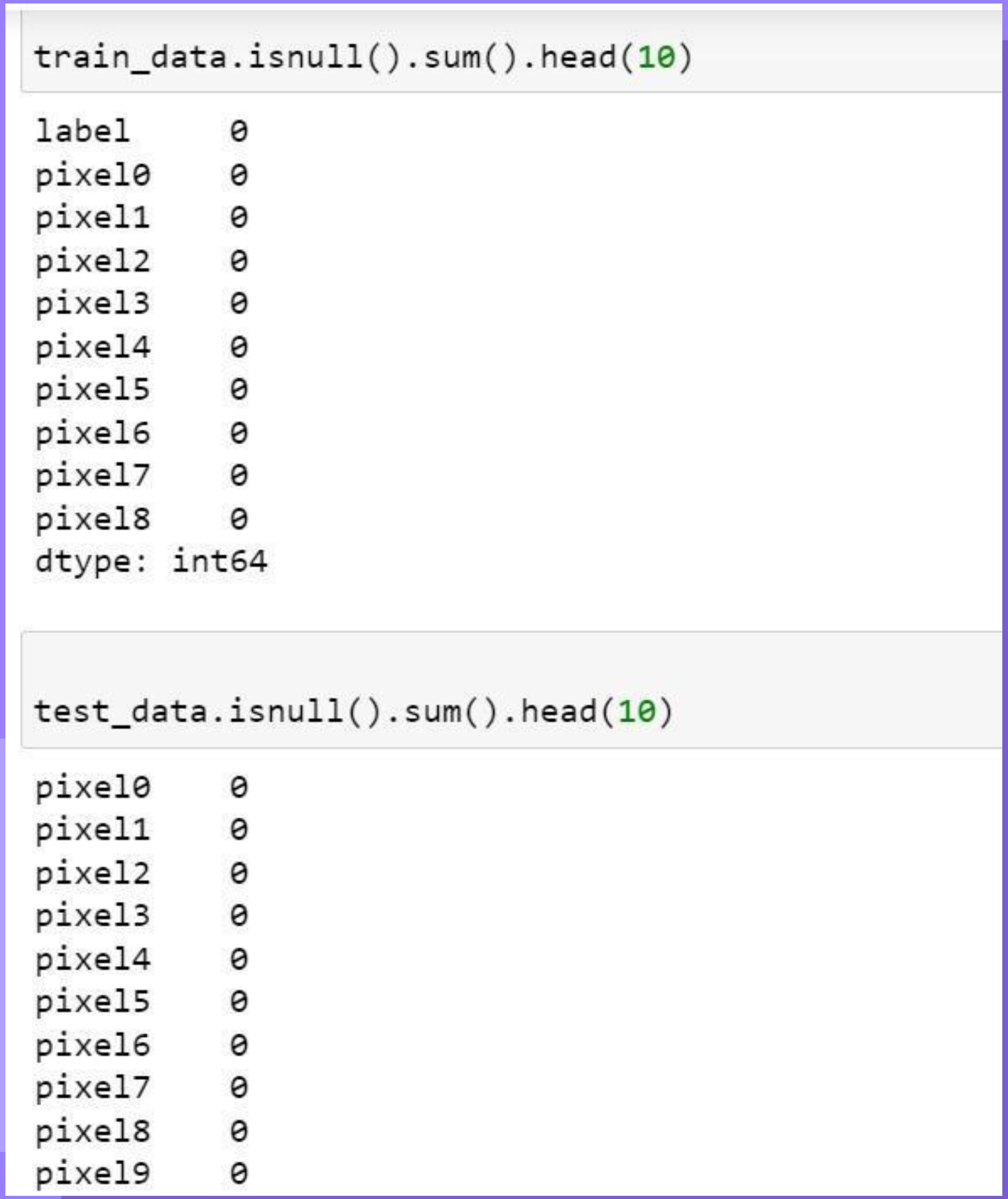


The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine. The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image



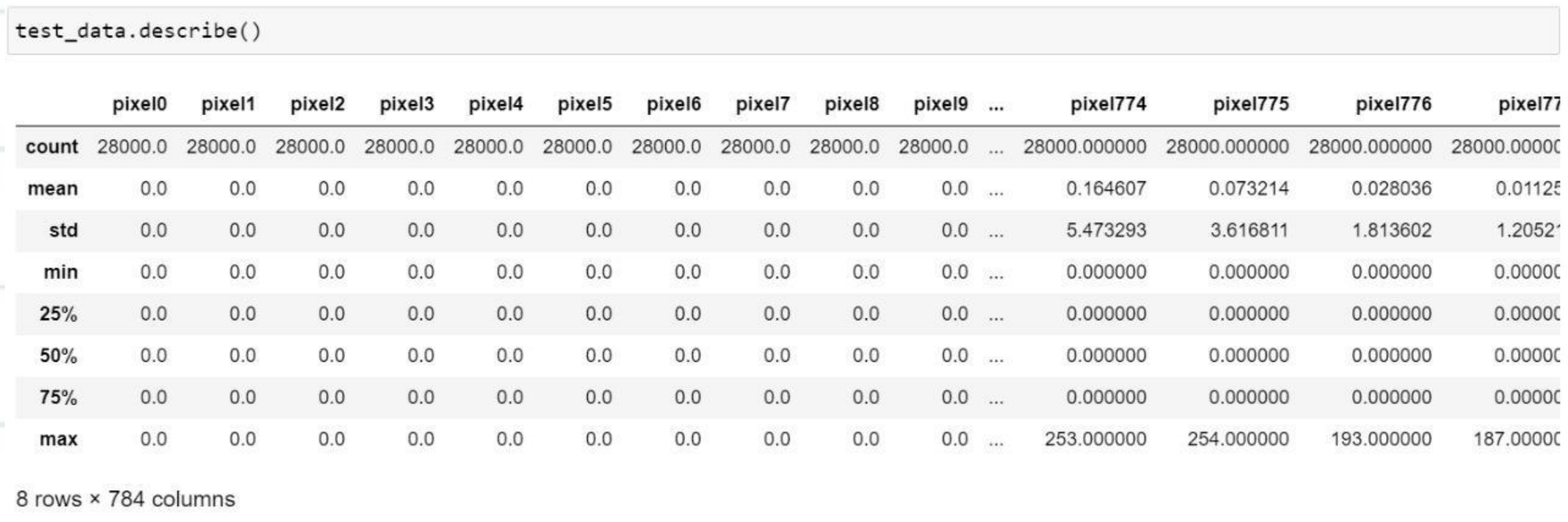
```
train_data = pd.read_csv(r"C:\Users\Apurva\Downloads\Digit_train\train.csv")
test_data = pd.read_csv(r"C:\Users\Apurva\Downloads\Digit_test\test.csv")

train_data.shape # print the dimension or shape of train data

(42000, 785)

test_data.shape # print the dimension or shape of test data

(28000, 784)
```

Shape of dataset



```
train_data.isnull().sum().head(10)

label     0
pixel0    0
pixel1    0
pixel2    0
pixel3    0
pixel4    0
pixel5    0
pixel6    0
pixel7    0
pixel8    0
dtype: int64

test_data.isnull().sum().head(10)

pixel0    0
pixel1    0
pixel2    0
pixel3    0
pixel4    0
pixel5    0
pixel6    0
pixel7    0
pixel8    0
pixel9    0
```

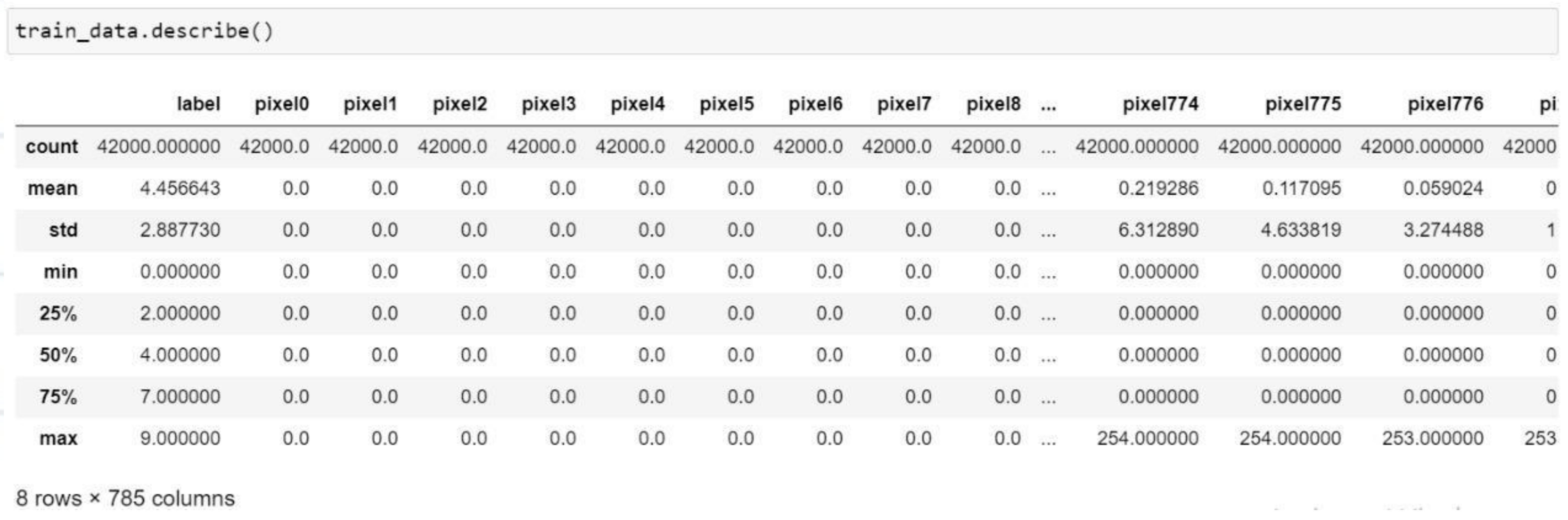
There are no missing values



```
test_data.describe()
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel77
count	28000.0	28000.0	28000.0	28000.0	28000.0	28000.0	28000.0	28000.0	28000.0	28000.0	...	28000.000000	28000.000000	28000.000000	28000.0000
mean	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.164607	0.073214	0.028036	0.01125
std	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	5.473293	3.616811	1.813602	1.2052
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.0000
25%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.0000
50%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.0000
75%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.0000
max	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	253.000000	254.000000	193.000000	187.0000

8 rows × 784 columns



```
train_data.describe()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pi
count	42000.000000	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	...	42000.000000	42000.000000	42000.000000	42000
mean	4.456643	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.219286	0.117095	0.059024	0
std	2.887730	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	6.312890	4.633819	3.274488	1
min	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0
25%	2.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0
50%	4.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0
75%	7.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0
max	9.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	254.000000	254.000000	253.000000	253

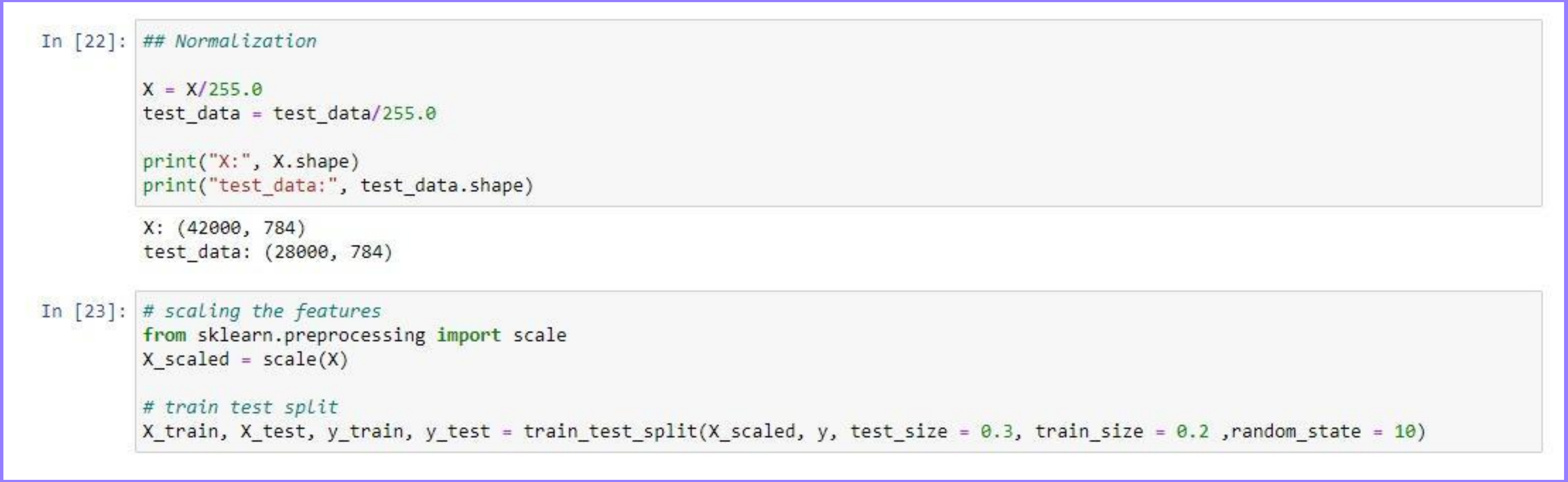
8 rows × 785 columns

The describe() method returns description of the data in the DataFrame.

If the DataFrame contains numerical data, the description contains these information for each column:

count - The number of not-empty values.
mean - The average (mean) value.
std - The standard deviation.
min - the minimum value.
25% - The 25% percentile.
50% - The 50% percentile*.
75% - The 75% percentile.
max - the maximum value.

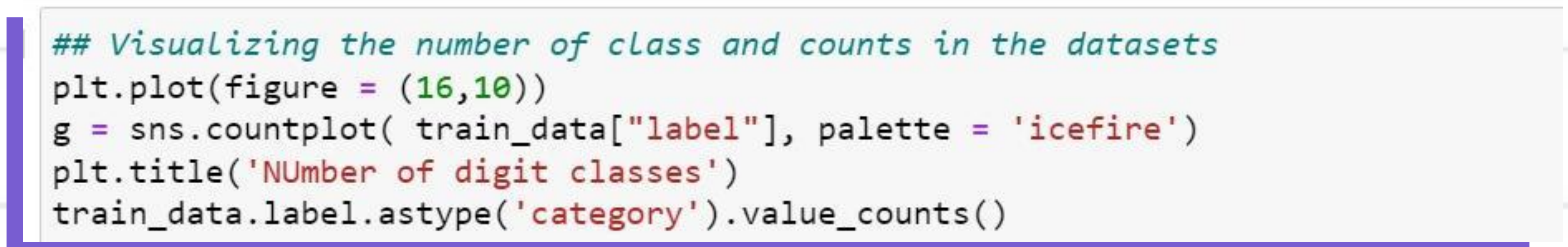
DATA CLEANING AND TRANSFORMATION



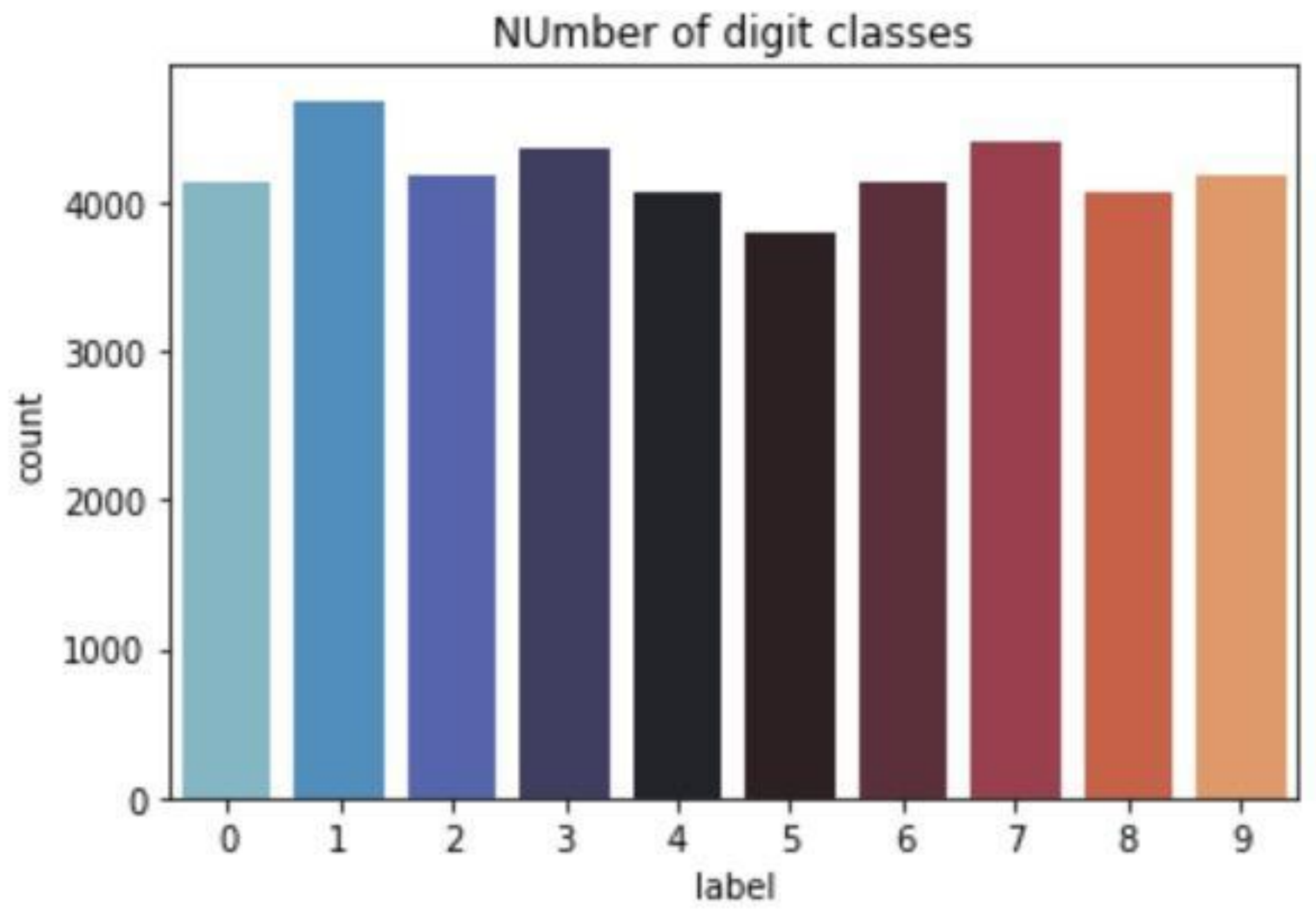
```
In [22]: ## Normalization

         X = X/255.0
         test_data = test_data/255.0

         print("X:", X.shape)
         print("test_data:", test_data.shape)

         X: (42000, 784)
         test_data: (28000, 784)

In [23]: # scaling the features
         from sklearn.preprocessing import scale
         X_scaled = scale(X)

         # train test split
         X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.3, train_size = 0.2 ,random_state = 10)
```

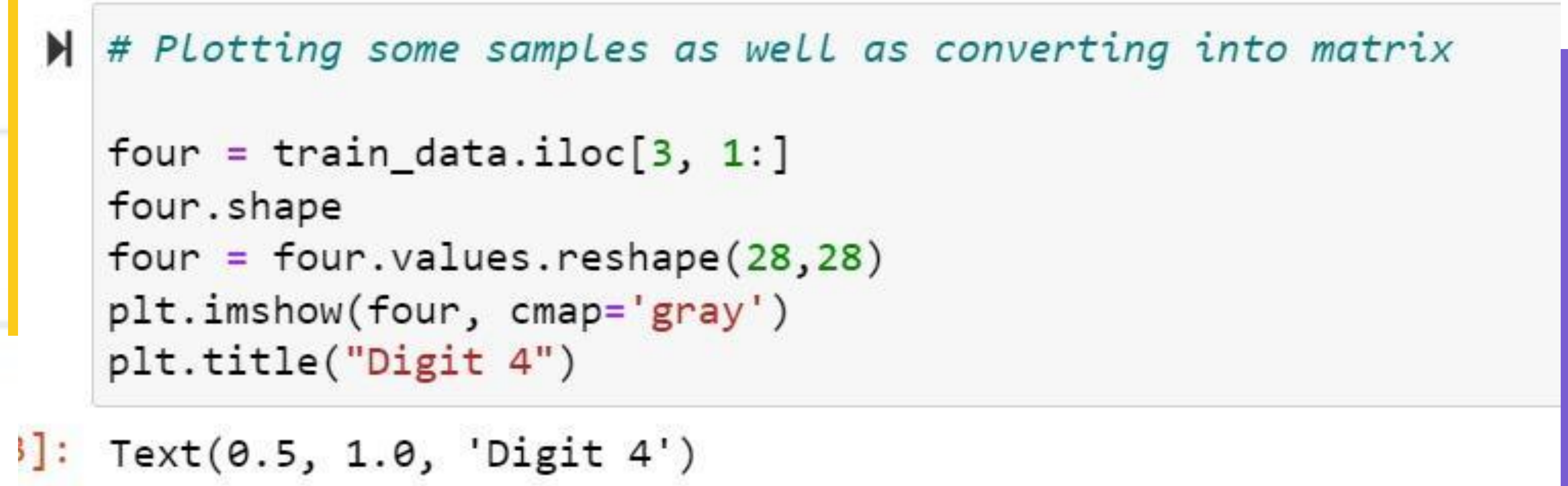
plotting of some samples as well as converting into matrix followed by normalisation and scaling of features have been done.

DATA VISUALISATION



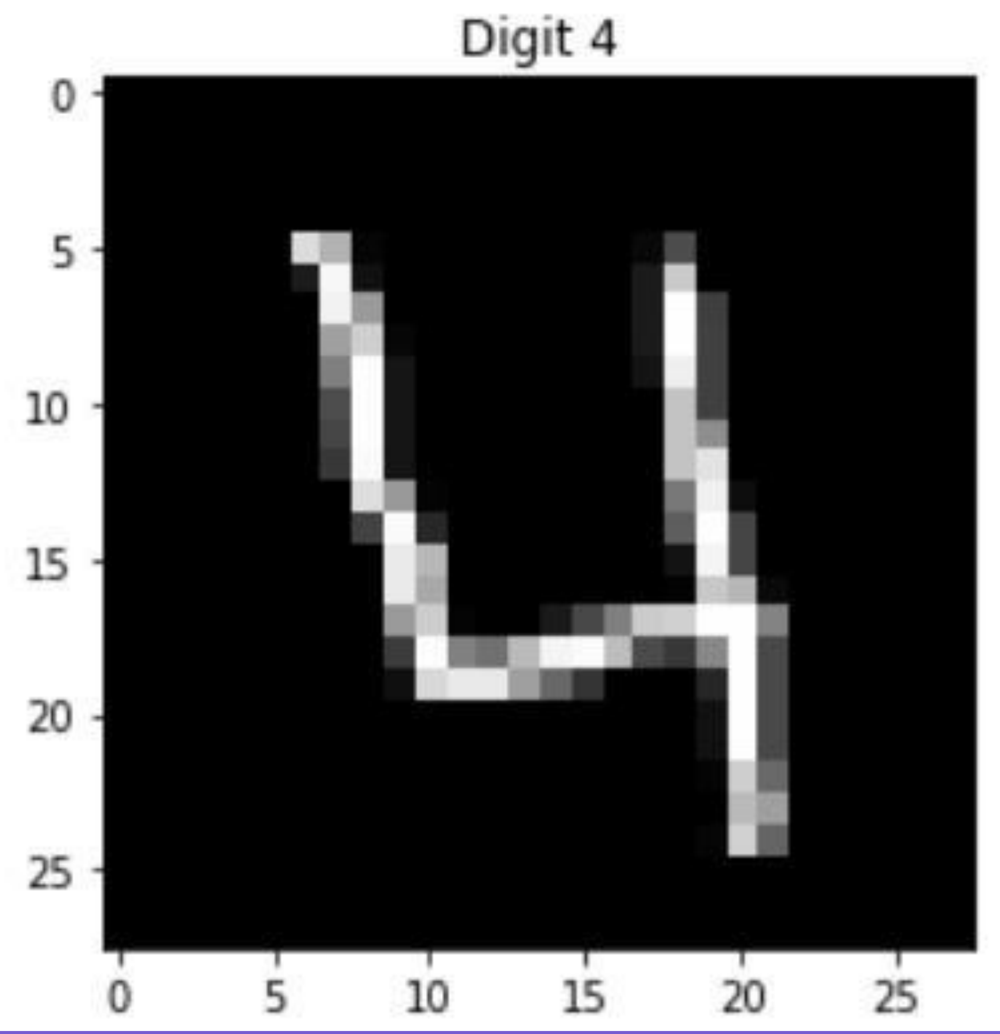
```
## Visualizing the number of class and counts in the datasets
plt.plot(figure = (16,10))
g = sns.countplot( train_data["label"], palette = 'icefire')
plt.title('NUmber of digit classes')
train_data.label.astype('category').value_counts()
```





```
# Plotting some samples as well as converting into matrix

four = train_data.iloc[3, 1:]
four.shape
four = four.values.reshape(28,28)
plt.imshow(four, cmap='gray')
plt.title("Digit 4")
```

```
]: Text(0.5, 1.0, 'Digit 4')
```



APPLICATION OF HCR

HEALTHCARE AND PHARMACEUTICALS

In the healthcare/pharmaceutical industry, patient medication digitisation is a serious issue. Roche, for example, processes millions of petabytes of medical PDFs every day. Patient enrolment and form digitalisation are other areas where handwritten character detection has a significant influence. Hospitals and pharmaceutical companies can greatly improve customer experience by adding handwriting analysis to their toolbox of services.

INSURANCE

A huge insurance firm receives over 20 million documents per day, and a claim processing delay can have a significant impact on the organisation. The claims document may contain a variety of handwriting styles, and relying solely on human processing to handle claims can significantly slow down the pipeline.

BANKING

People write checks on a regular basis, and they continue to play a significant part in the majority of non-cash transactions. The current check processing technique in many developing nations involves a bank staff to read and manually put the information on a cheque, as well as verify the data such as signature and date. Because a bank processes a huge number of cheques every day, a handwriting textual recognition system can save money and hours of human labour.

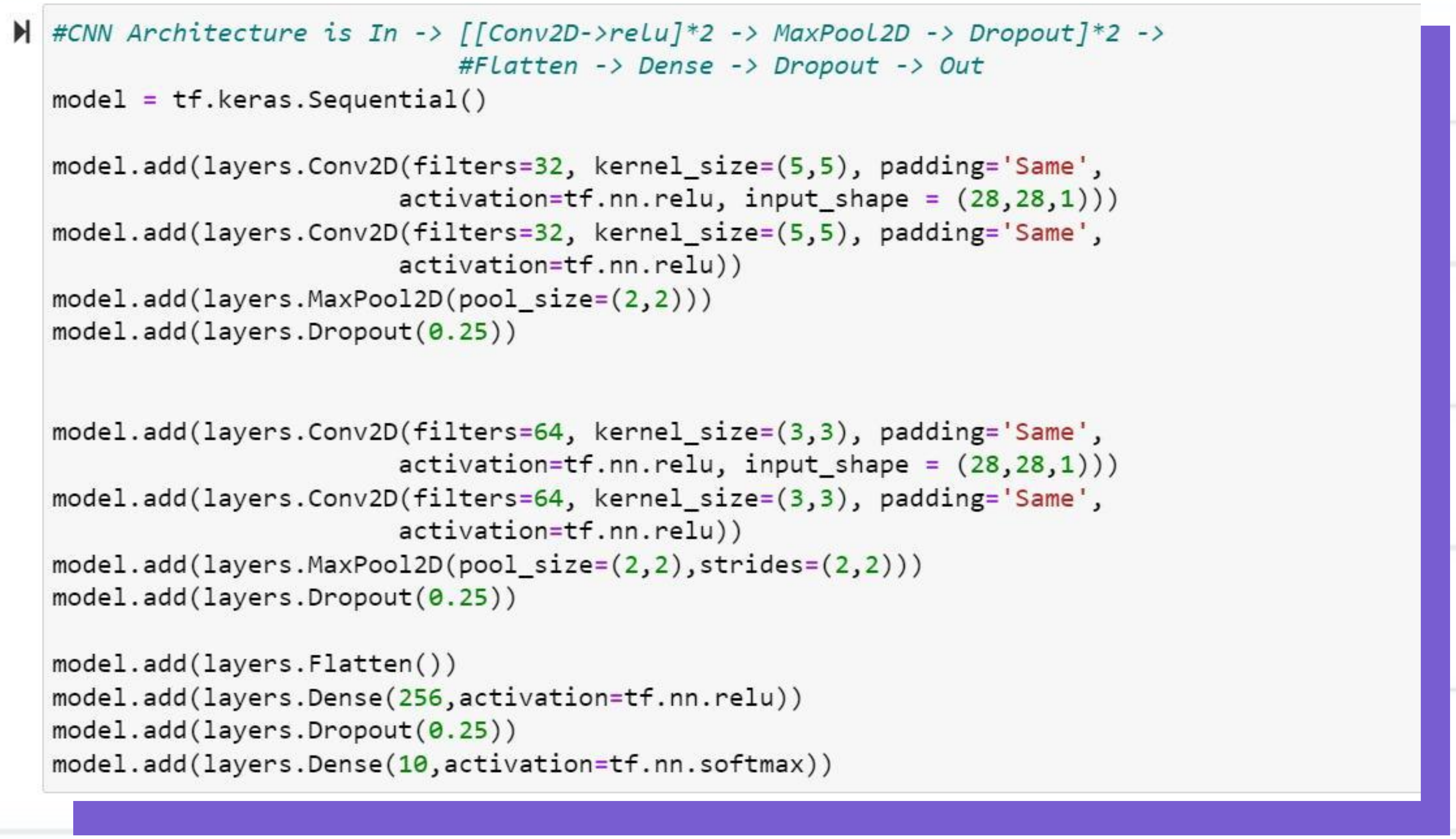
ONLINE LIBRARIES

Huge volumes of historical knowledge are being digitised and made available to the world by uploading image scans. However, this endeavour will be ineffective unless the text in the photographs can be detected and indexed, queried, and browsed. Handwriting identification is essential for bringing mediaeval and twentieth-century papers, postcards, and research works to life.

ALGORITHMS USED

- CNN (Convolutional Neural Network)
- SVM (Support Vector Machine)
  - Linear Model
  - Non-Linear Model

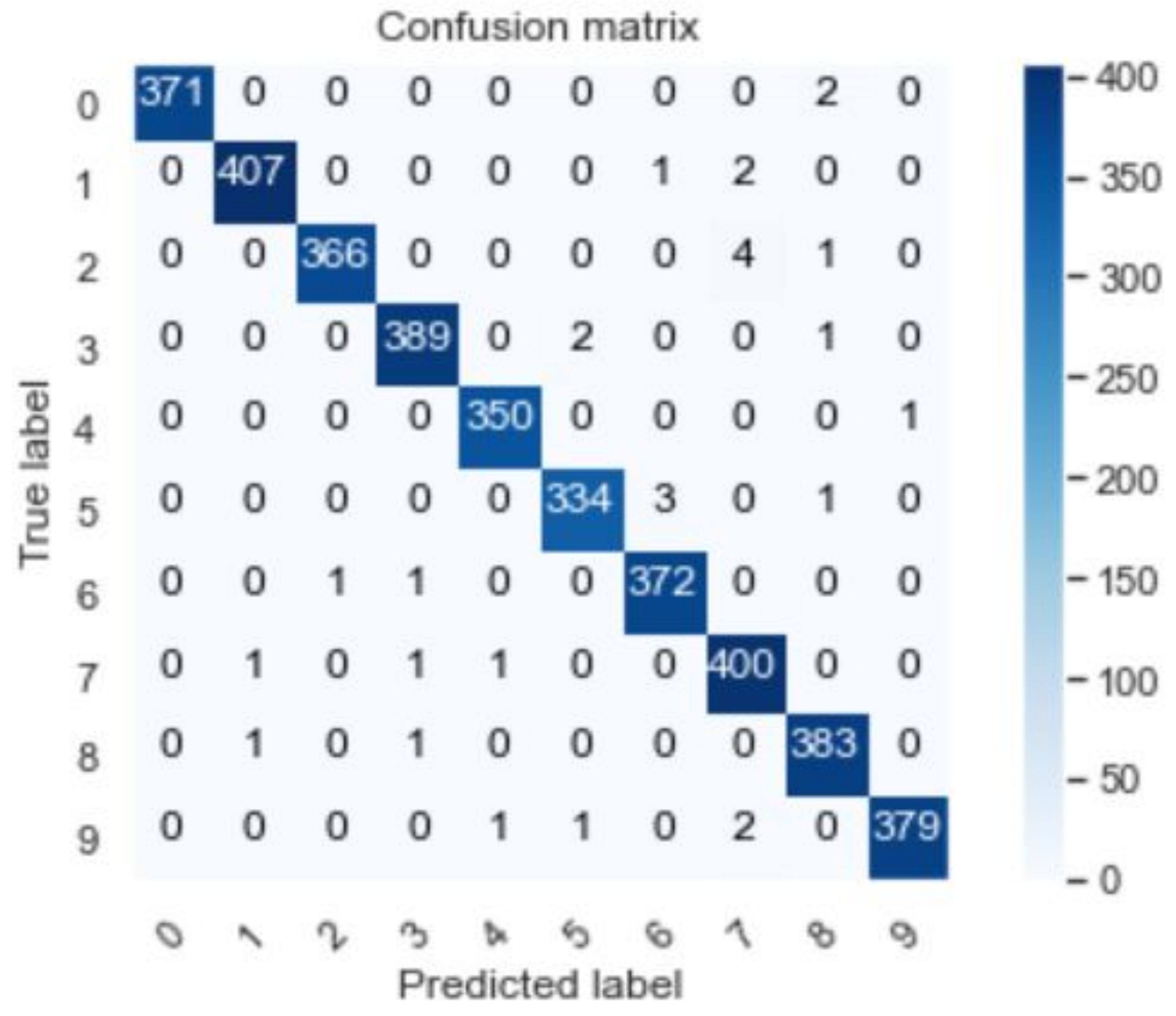
CONVOLUTIONAL NEURAL NETWORK



```
#CNN Architecture is In -> [[Conv2D->relu]*2 -> MaxPool2D -> Dropout]*2 ->
                           #Flatten -> Dense -> Dropout -> Out
model = tf.keras.Sequential()

model.add(layers.Conv2D(filters=32, kernel_size=(5,5), padding='Same',
                       activation=tf.nn.relu, input_shape = (28,28,1)))
model.add(layers.Conv2D(filters=32, kernel_size=(5,5), padding='Same',
                       activation=tf.nn.relu))
model.add(layers.MaxPool2D(pool_size=(2,2)))
model.add(layers.Dropout(0.25))


model.add(layers.Conv2D(filters=64, kernel_size=(3,3), padding='Same',
                       activation=tf.nn.relu, input_shape = (28,28,1)))
model.add(layers.Conv2D(filters=64, kernel_size=(3,3), padding='Same',
                       activation=tf.nn.relu))
model.add(layers.MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(layers.Dropout(0.25))

model.add(layers.Flatten())
model.add(layers.Dense(256,activation=tf.nn.relu))
model.add(layers.Dropout(0.25))
model.add(layers.Dense(10,activation=tf.nn.softmax))
```

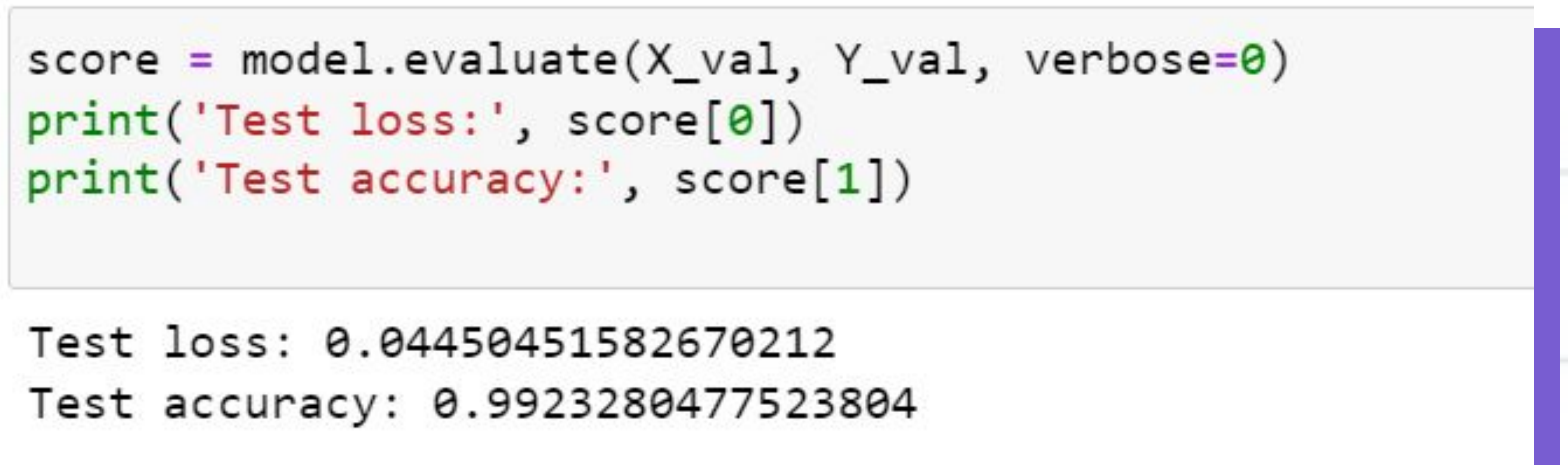
A simple convolutional neural network is a sequence of layers, and every layer transforms one volume of activations to another through a differentiable function. We use three main types of layers to build the network. These are convolutional layers, pooling layers and fully connected layers. We will stack these layers to form our network architecture. We will go into more details below

The image data cannot be fed directly into the model so we need to perform some operations and process the data to make it ready for our neural network. The dimension of the training data is (60000,28,28). The CNN model will require one more dimension so we reshape the matrix to shape (60000,28,28,1).



	0	1	2	3	4	5	6	7	8	9
0	371	0	0	0	0	0	0	0	2	0
1	0	407	0	0	0	0	1	2	0	0
2	0	0	366	0	0	0	0	4	1	0
3	0	0	0	389	0	2	0	0	1	0
4	0	0	0	0	350	0	0	0	0	1
5	0	0	0	0	0	334	3	0	1	0
6	0	0	1	1	0	0	372	0	0	0
7	0	1	0	1	1	0	0	400	0	0
8	0	1	0	1	0	0	0	0	383	0
9	0	0	0	0	1	1	0	2	0	379

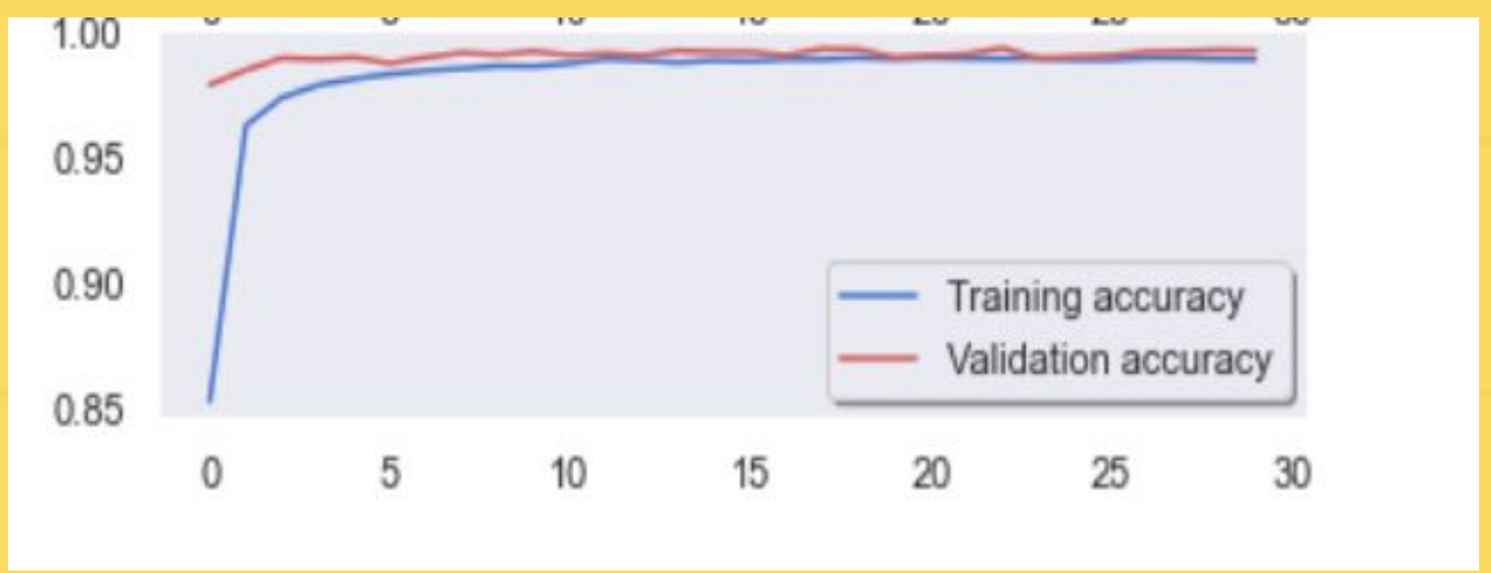
A **Confusion** matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.



```
score = model.evaluate(X_val, Y_val, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

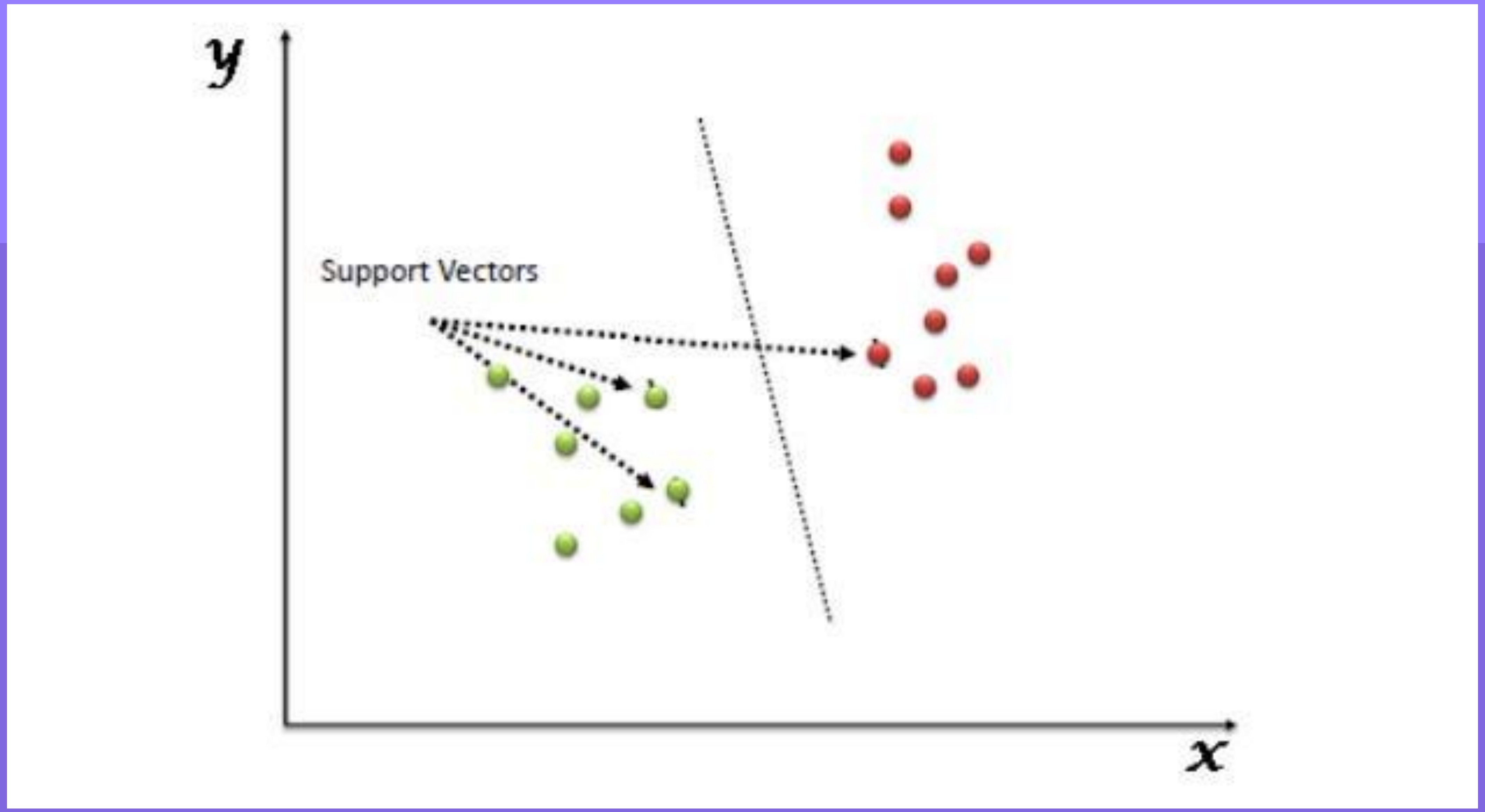
```
Test loss: 0.04450451582670212
Test accuracy: 0.9923280477523804
```

The CNN model gives approx. 99% accuracy



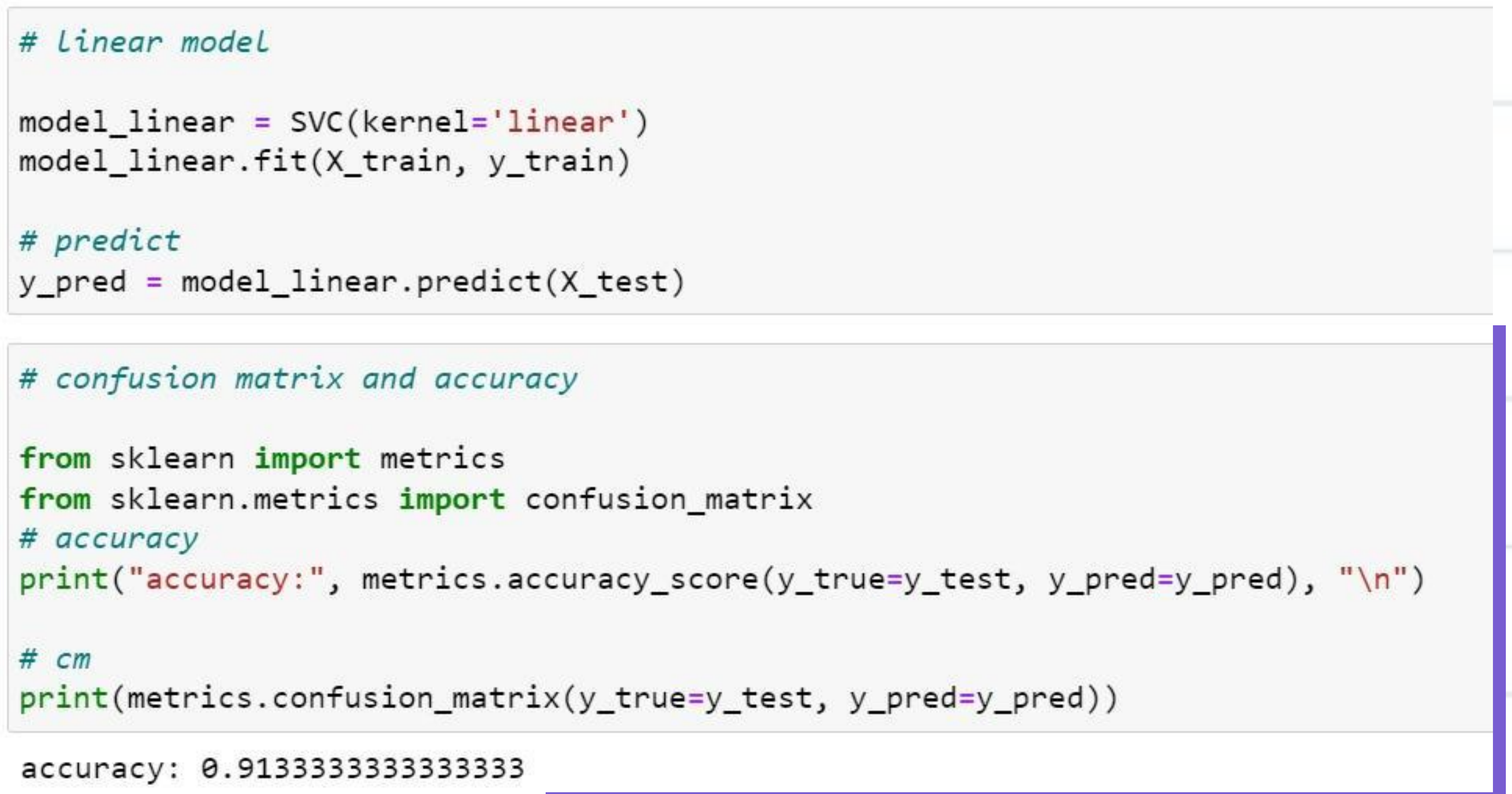
SUPPORT VECTOR MACHINE

- Support Vector Machine” (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well



LINEAR MODEL

Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.



```
# linear model

model_linear = SVC(kernel='linear')
model_linear.fit(X_train, y_train)

# predict
y_pred = model_linear.predict(X_test)
```

```
# confusion matrix and accuracy

from sklearn import metrics
from sklearn.metrics import confusion_matrix
# accuracy
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")

# cm
print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))
```

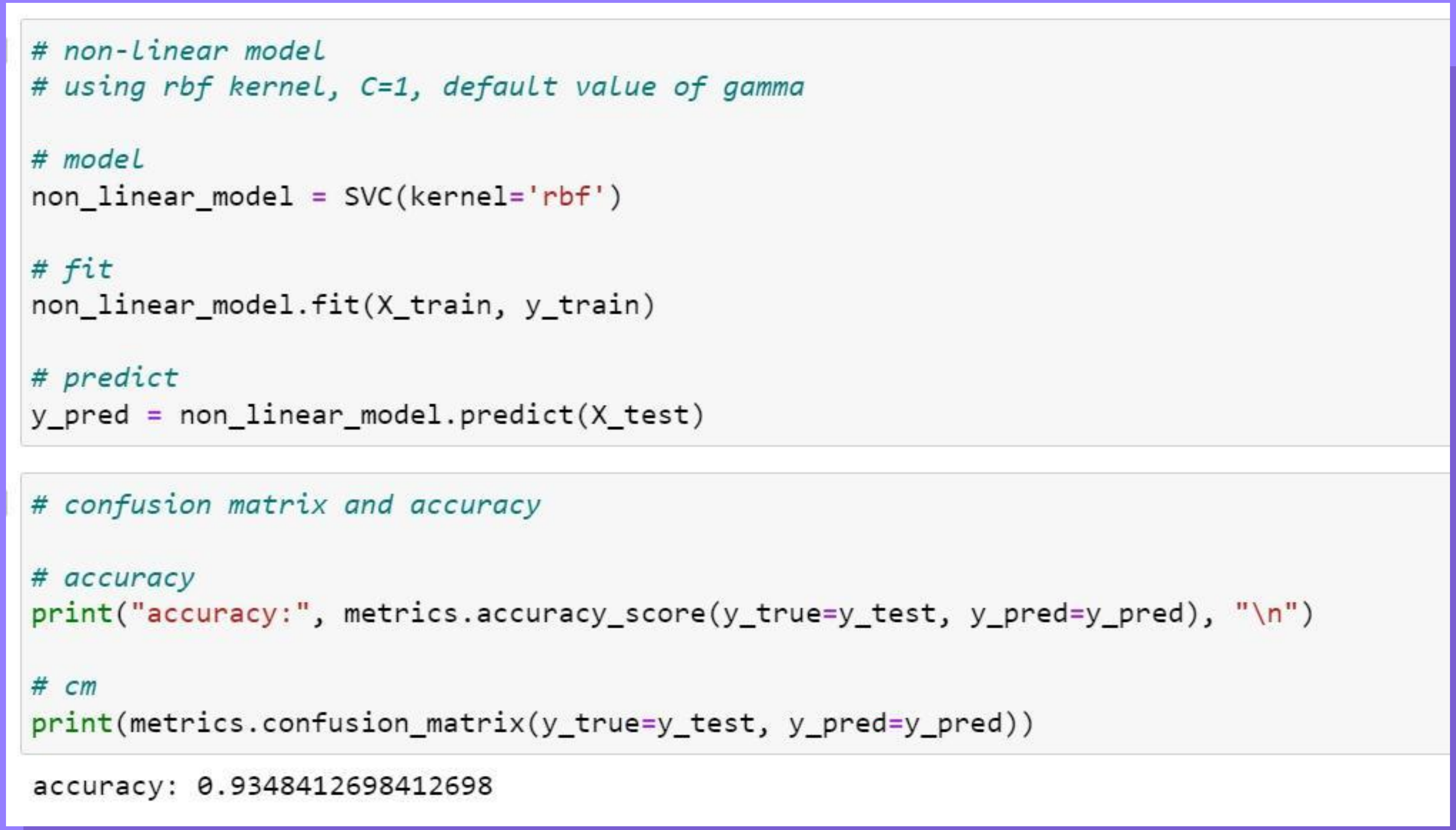
```
accuracy: 0.9133333333333333
```

The linear model gives approx. 91% accuracy

Machine Learning Presentation

NON-LINEAR MODEL

Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.



```
# non-linear model
# using rbf kernel, C=1, default value of gamma

# model
non_linear_model = SVC(kernel='rbf')

# fit
non_linear_model.fit(X_train, y_train)

# predict
y_pred = non_linear_model.predict(X_test)
```

```
# confusion matrix and accuracy

# accuracy
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")

# cm
print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))
```

```
accuracy: 0.9348412698412698
```

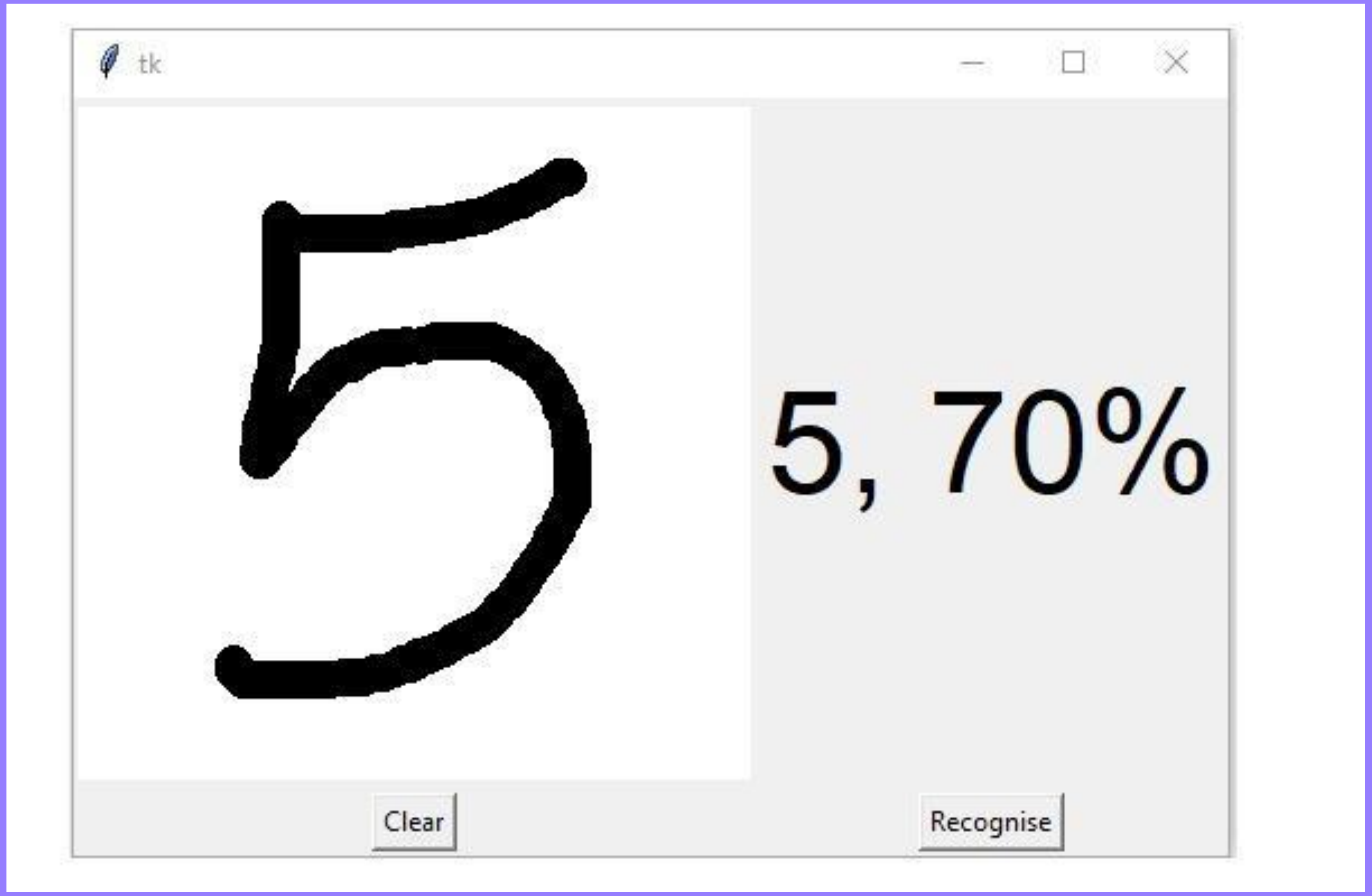
The linear model gives approx. 93% accuracy

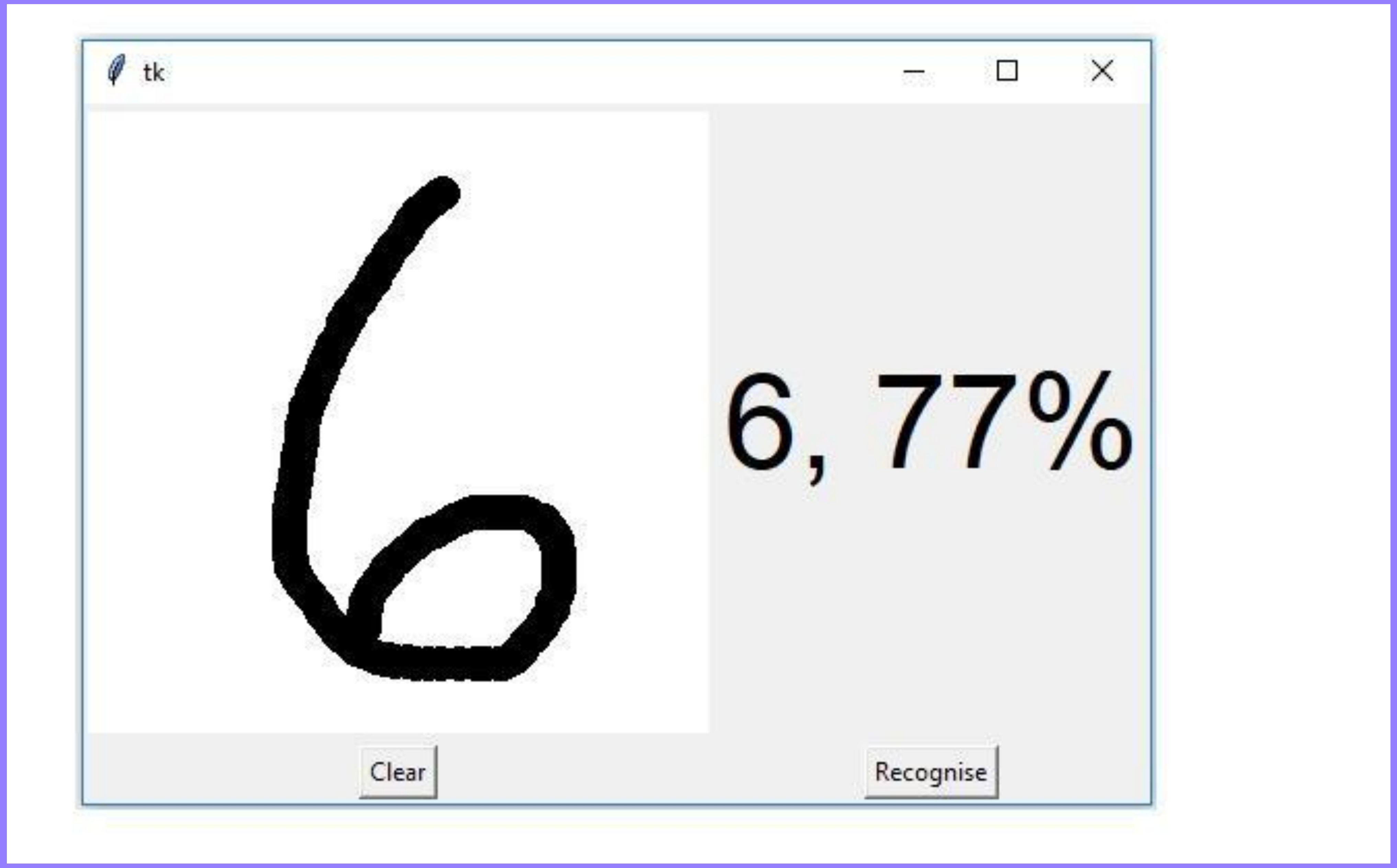
CONCLUSION

After executing all the models, we found that SVM has the highest accuracy on training data while on testing dataset CNN accomplishes the utmost accuracy. Additionally, we have compared the execution time to gain more insight into the working of the algorithms. Generally, the running time of an algorithm depends on the number of operations it has performed. So, we have trained our deep learning model up to 30 epochs and SVM models according to norms to get the apt outcome. SVM took the minimum time for execution while CNN accounts for the maximum running time.

PREDICTIONS

As it is shown below, the model has successfully predicted the handwritten digits given to it.





THANK YOU