```
import tensorflow as tf
```

```
a= tf.constant([[1,2,3],[4,5,6]])
b= tf.constant([[7,8, 9],[1,0,1]])
c= tf.constant([[7,8,9],[1,0,1],[1,1,1]])
```

```
tf.print(a.shape)
```

```
     TensorShape([2, 3])
```

```
tf.print(b.shape)
```

```
     TensorShape([2, 3])
```

```
tf.print(tf.rank(a))
```

```
     2
```

```
tf.print(tf.rank(b))
```

```
     2
```

```
tf.print(a)
tf.print(b)
tf.print(c)
```

```
     [[1 2 3]
      [4 5 6]]
     [[7 8 9]
      [1 0 1]]
     [[7 8 9]
      [1 0 1]
      [1 1 1]]
```

```
sum= tf.add(a,b)
tf.print(sum)
```

```
     [[8 10 12]
      [5 5 7]]
```

```
sub= tf.subtract(a,b)
tf.print(sub)
```

```
     [[-6 -6 -6]
      [3 5 5]]
```

```
div= tf.divide(b,a)
tf.print(div)
```

```
     [[7 4 3]
      [0.25 0 0.16666666666666666]]
```

```
mul =tf.matmul(a,c)
tf.print(mul)
```

```
     [[12 11 14]
      [39 38 47]]
```

```
tf.print(tf.rank(mul))
```

```
     2
```

```
tf.print(mul.shape)
```

```
     TensorShape([2, 3])
```

KNN on MNIST data

```
```

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
```

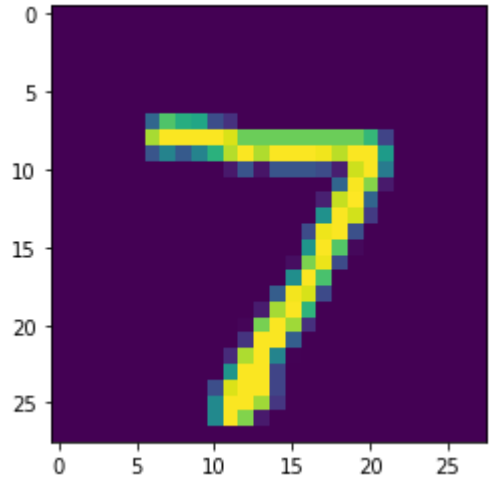
```
(X_train, y_train), (X_test, y_test)= mnist.load_data()
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

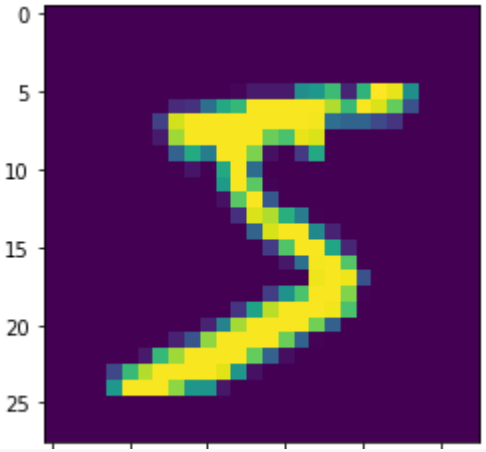
```
plt.imshow(X_test[0])
```

```
<matplotlib.image.AxesImage at 0x7f581ef604a8>
```




```
plt.imshow(X_train[0])
```

```
<matplotlib.image.AxesImage at 0x7f581ef48400>
```




```
X_train= X_train.reshape(-1, 28*28)
X_test= X_test.reshape(-1, 28*28)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(60000, 784)
(10000, 784)
(60000,)
(10000,)
```

```
train_digits= tf.Variable(X_train, dtype="float", shape=[None, 784])
test_digits= tf.Variable(X_test[0], dtype="float", shape=[784])

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(60000, 784)
(10000, 784)
```

```
(60000,)
(10000,)
```

```
print(train_digits.shape)
print(test_digits.shape)
```

```
(None, 784)
(784,)
```

```
def kNearestNeighbour(test_digits):
  l1_distance = tf.abs(tf.add(train_digits,tf.negative(test_digits)))
  dist= tf.reduce_sum(l1_distance,axis=1)
  return np.array(tf.argsort(dist))
```

```
classes=[0,1,2,3,4,5,6,7,8,9]
```

```
test_digits.assign(X_test[0])
sort_indices= kNearestNeighbour(test_digits)

tf.print(sort_indices)
print(y_test[0])
print(len(sort_indices))
```

```
array([53843, 27059, 38620, ..., 25321, 59439, 41358], dtype=int32)
7
60000
```

```
k=int(input("Enter the value of k:"))
correct=0
for i in range(100): #for 100 test digits
  ind=[0,0,0,0,0,0,0,0,0,0]
  test_digits.assign(X_test[i,:])
  indices = kNearestNeighbour(test_digits)
  labels=[]
  for j in range(k):
    key= indices[j]
    labels.append(y_train[key])
  for l in labels:
```

```
  if l==0:
    ind[0]=ind[0]+1
  elif l==1:
    ind[1]=ind[1]+1
  elif l==2:
    ind[2]=ind[2]+1
  elif l==3:
    ind[3]=ind[3]+1
  elif l==4:
    ind[4]=ind[4]+1
  elif l==5:
    ind[5]=ind[5]+1
  elif l==6:
    ind[6]=ind[6]+1
  elif l==7:
    ind[7]=ind[7]+1
  elif l==8:
    ind[8]=ind[8]+1
  else:
    ind[9]=ind[9]+1
print("Epoch: ",(i+1)," Predicted: ", classes[np.argmax(ind)], " actual: ",y_test[i])
if classes[np.argmax(ind)]==y_test[i]:
  correct+= 1
print("correctly predicted: ",correct)
```

```
 Enter the value of k:4
 Epoch:  1  Predicted:  7  actual:  7
 correctly predicted:  1
 Epoch:  2  Predicted:  2  actual:  2
 correctly predicted:  2
 Epoch:  3  Predicted:  1  actual:  1
 correctly predicted:  3
 Epoch:  4  Predicted:  0  actual:  0
 correctly predicted:  4
 Epoch:  5  Predicted:  4  actual:  4
 correctly predicted:  5
 Epoch:  6  Predicted:  1  actual:  1
 correctly predicted:  6
 Epoch:  7  Predicted:  4  actual:  4
 correctly predicted:  7
```

```
Epoch:  8  Predicted:  9  actual:  9
correctly predicted:  8
Epoch:  9  Predicted:  5  actual:  5
correctly predicted:  9
Epoch:  10  Predicted:  9  actual:  9
correctly predicted:  10
Epoch:  11  Predicted:  0  actual:  0
correctly predicted:  11
Epoch:  12  Predicted:  6  actual:  6
correctly predicted:  12
Epoch:  13  Predicted:  9  actual:  9
correctly predicted:  13
Epoch:  14  Predicted:  0  actual:  0
correctly predicted:  14
Epoch:  15  Predicted:  1  actual:  1
correctly predicted:  15
Epoch:  16  Predicted:  5  actual:  5
correctly predicted:  16
Epoch:  17  Predicted:  9  actual:  9
correctly predicted:  17
Epoch:  18  Predicted:  7  actual:  7
correctly predicted:  18
Epoch:  19  Predicted:  3  actual:  3
correctly predicted:  19
Epoch:  20  Predicted:  4  actual:  4
correctly predicted:  20
Epoch:  21  Predicted:  9  actual:  9
correctly predicted:  21
Epoch:  22  Predicted:  6  actual:  6
correctly predicted:  22
Epoch:  23  Predicted:  6  actual:  6
correctly predicted:  23
Epoch:  24  Predicted:  5  actual:  5
correctly predicted:  24
Epoch:  25  Predicted:  9  actual:  4
correctly predicted:  24
Epoch:  26  Predicted:  0  actual:  0
correctly predicted:  25
Epoch:  27  Predicted:  7  actual:  7
correctly predicted:  26
Epoch:  28  Predicted:  4  actual:  4
correctly predicted:  27
Epoch:  29  Predicted:  0  actual:  0
```

KNN on IRIS Dataset

```
import pandas as pd
import numpy as np
```

```
data = pd.read_csv('/content/drive/MyDrive/iris (1).csv')
```

```
data.head()
```

	Unnamed: 0	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	1	5.1	3.5	1.4	0.2	setosa
1	2	4.9	3.0	1.4	0.2	setosa
2	3	4.7	3.2	1.3	0.2	setosa
3	4	4.6	3.1	1.5	0.2	setosa
4	5	5.0	3.6	1.4	0.2	setosa

```
data = data.drop("Unnamed: 0",axis=1)
```

```
data["Species"].unique()
```

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

```
X= data.drop("Species", axis=1)
y= data["Species"]
```

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

```
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.2, random_state = 42)
```

```
X_train.head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
22	4.6	3.6	1.0	0.2
15	5.7	4.4	1.5	0.4
65	6.7	3.1	4.4	1.4
11	4.8	3.4	1.6	0.2
42	4.4	3.2	1.3	0.2

```
X_train.shape
```

```
(120, 4)
```

```
y_test.shape
```

```
(30,)
```

```
X_test.shape
```

```
(30, 4)
```

```
y_train.shape
```

```
(120,)
```

```
import tensorflow as tf
```

```
X_train= np.array(X_train)
X_test= np.array(X_test)
y_train= np.array(y_train)
```

```
y_train= np.array(y_train)
y_test= np.array(y_test)
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(120, 4)
(120,)
(30, 4)
(30,)
```

```
def kNearestNeighbour(train_data,new_entry):
  l1_distance = tf.abs(tf.add(train_data,tf.negative(new_entry)))
  dist= tf.reduce_sum(l1_distance,axis=1)
  return np.array(tf.argsort(dist))
```

```
classes=['setosa', 'versicolor', 'virginica']
```

```
sort_indices= kNearestNeighbour(X_train, X_test[0])
tf.print(sort_indices)
print(y_test[0])
print(len(sort_indices))
print(len(sort_indices[5:]))
```

```
array([ 79,  90,  39,  81, 115,  92,  80,  16, 111,  62,  42,  73,  93,
        20,  36,  30, 108, 110,  86,  34, 118,  18,  25,  22,  12,   6,
        63,  68,  59,  11, 113,  45,   2,  95,  10,  43,   5, 112,  82,
        44,  54, 105,  65,  50,  40, 107,  56,  47,  87,  88,  89,  53,
       109,  17,  15, 116,  83,  46,  76,  85, 101,  74, 103,  97,  60,
        99,  49,  29,  77,  61, 119, 100, 106,  69,  96,  64,  24,  21,
        19,  33, 114,  32,  58,  28,  48,  37,  31,  55, 104,  26,   7,
        14,  35,  51,  57,  71,  94,  98,   1,   3,  13,  52,  70, 117,
        23,  27,  66,  84,  78,   8,  38,  72,  41,  75,  91, 102,   9,
         4,  67,   0], dtype=int32)
versicolor
```

```
     120
     115
```

```
for i in range(5):
  print(y_train[sort_indices[i]])
```

```
     versicolor
     versicolor
     versicolor
     virginica
     versicolor
```

```
k=int(input("Enter the value of k:"))
correct=0
for i in range(X_test.shape[0]):
  ind=[0,0,0] #classes=['setosa', 'versicolor', 'virginica']
  indices= kNearestNeighbour(X_train, X_test[i])
  labels=[]
  for j in range(k):
    key= indices[j]
    labels.append(y_train[key])
  for l in labels:
    if l=="setosa":
      ind[0]=ind[0]+1
    elif l=="versicolor":
      ind[1]=ind[1]+1
    else:
      ind[2]=ind[2]+1
  print("Predicted: ", classes[np.argmax(ind)], " actual: ",y_test[i])

  if classes[np.argmax(ind)]==y_test[i]:
    correct+= 1

print("correctly predicted: ",correct)
```

```
     Enter the value of k:3
     Predicted:  versicolor  actual:  versicolor
     Predicted:  setosa  actual:  setosa
     Predicted:  virginica  actual:  virginica
     Predicted:  versicolor  actual:  versicolor
```

```
Predicted:  versicolor  actual:  versicolor
Predicted:  setosa  actual:  setosa
Predicted:  versicolor  actual:  versicolor
Predicted:  virginica  actual:  virginica
Predicted:  versicolor  actual:  versicolor
Predicted:  versicolor  actual:  versicolor
Predicted:  virginica  actual:  virginica
Predicted:  setosa  actual:  setosa
Predicted:  setosa  actual:  setosa
Predicted:  setosa  actual:  setosa
Predicted:  setosa  actual:  setosa
Predicted:  versicolor  actual:  versicolor
Predicted:  virginica  actual:  virginica
Predicted:  versicolor  actual:  versicolor
Predicted:  versicolor  actual:  versicolor
Predicted:  virginica  actual:  virginica
Predicted:  setosa  actual:  setosa
Predicted:  virginica  actual:  virginica
Predicted:  setosa  actual:  setosa
Predicted:  virginica  actual:  virginica
Predicted:  virginica  actual:  virginica
Predicted:  virginica  actual:  virginica
Predicted:  virginica  actual:  virginica
Predicted:  virginica  actual:  virginica
Predicted:  setosa  actual:  setosa
Predicted:  setosa  actual:  setosa
correctly predicted:  30
```

Using inbuilt model

```
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.2, random_state = 42)
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
kVals= np.arange(1,6)

for k in kVals:

    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train,y_train)
```

```
    # evaluate the model and update the accuracies list
    score = model.score(X_test, y_test)
    print("k: ",k,", Accuracy: ",(score*100),"%")
```

```
 k:  1 , Accuracy:  100.0 %
 k:  2 , Accuracy:  100.0 %
 k:  3 , Accuracy:  100.0 %
 k:  4 , Accuracy:  100.0 %
 k:  5 , Accuracy:  100.0 %
```

```
pred= model.predict([[4.4,  3.2,  1.3,  0.2]])
```

```
print(pred)
```

```
 ['setosa']
```

```
data.iloc[42]
```

```
 Sepal.Length       4.4
 Sepal.Width        3.2
 Petal.Length       1.3
 Petal.Width        0.2
 Species         setosa
 Name: 42, dtype: object
```

```

```