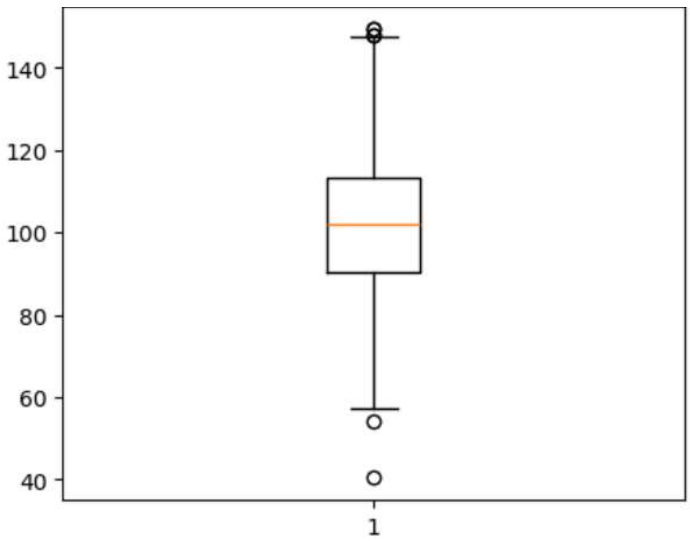
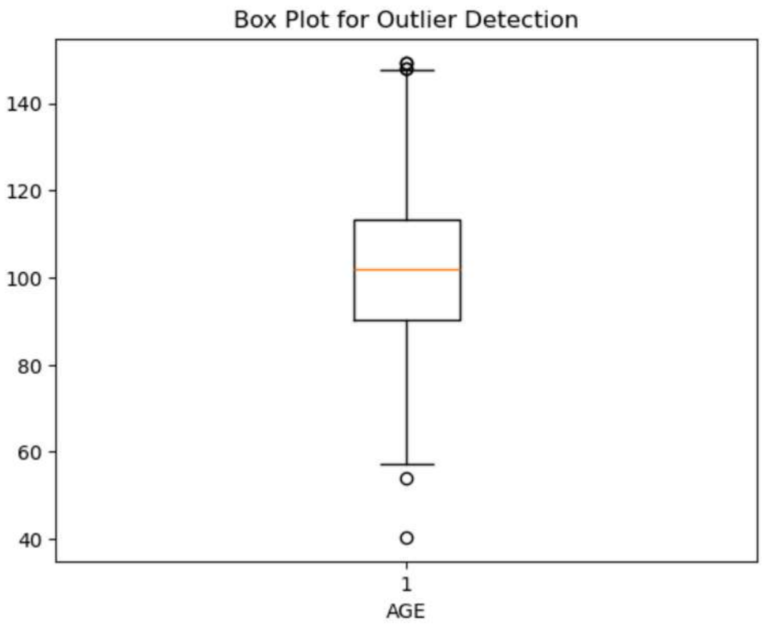
Exp 2.1




```
import pandas as pd


import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt


np.random.seed(10)

data = pd.read_excel('Data.xlsx')

fig = plt.figure(figsize =(5,4))

plt.boxplot(data)

plt.show()
```

Exp2.2




```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

data = pd.read_excel("data.xlsx")

plt.boxplot(data)

plt.xlabel('AGE')

plt.title('Box Plot for Outlier Detection')

plt.show()

Q1=np.percentile(data,25)

Q3=np.percentile(data,75)

IQR=Q3-Q1

k=1.5

lower_bound=Q1-k*IQR
```

```
upper_bound = Q3+k* IQR

array = data.to_numpy()

outliers = array[(array<lower_bound)|(array>upper_bound)]

print(outliers)
```

Output

[147.8940733 149.35302113 148.08651212 149.30650164 40.40806458

 54.09793342]

Exp 2.3

```
import numpy as np

import pandas as pd

def outlier_detection_zscore(data, k=3):

    mean = np.mean(data)

    std = np.std(data)

    zscores = (data - mean) / std

    outlier_indices = np.where(np.abs(zscores) > k)[0]

    outliers = data[outlier_indices]

    return outlier_indices.tolist(), outliers



data = data.to_numpy()

outlier_indices, outliers = outlier_detection_zscore(data)

print("Outlier Indices:", outlier_indices)

print("Outliers:", outliers)
```

Output

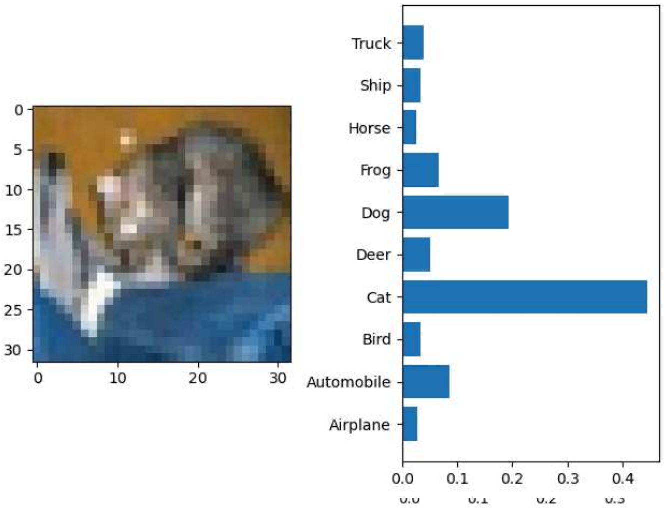
Outlier Indices: [159]

Outliers: [[40.40806458]]

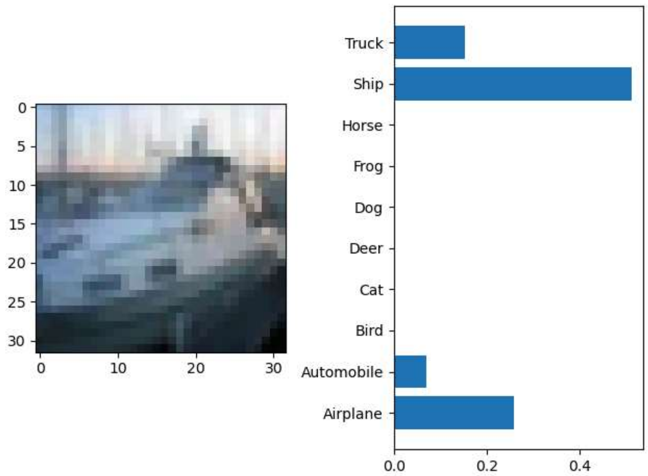
Exp 3

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Flatten
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

(X_train,y_train),(X_test,y_test)=cifar10.load_data()

X_train=X_train/255
X_test=X_test/255
y_train_cat=to_categorical(y_train,num_classes=10)
y_test_cat=to_categorical(y_test,num_classes=10)

model=Sequential()
model.add(Flatten(input_shape=(32,32,3)))
model.add(Dense(units=256,activation='relu'))
model.add(Dense(units=256,activation='relu'))
model.add(Dense(units=128,activation='relu'))
model.add(Dense(units=10,activation='softmax'))

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

model.fit(X_train,y_train_cat,epochs=10)

class_labels=['Airplane','Automobile','Bird','Cat','Deer','Dog','Frog','Horse','Ship','Truck']

for i in range(2):
  plt.subplot(1,2,1)
  plt.imshow(X_test[i])
  plt.subplot(1,2,2)
  pred=model.predict(X_test[i].reshape(1,32,32,3))
  plt.barh(class_labels,pred[0])
  plt.tight_layout()
  plt.show()
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.2755 - loss: 1.9784
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.3821 - loss: 1.7045
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4225 - loss: 1.6100
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4477 - loss: 1.5441
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4596 - loss: 1.5093
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4752 - loss: 1.4612
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4886 - loss: 1.4279
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4908 - loss: 1.4180
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.5004 - loss: 1.3900
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.5057 - loss: 1.3725

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
```



Exp 4

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Flatten
from tensorflow.keras import regularizers,initializers
from tensorflow.keras.datasets import cifar10

(X_train,y_train),(X_test,y_test)=cifar10.load_data()
X_train=X_train/255
X_test=X_test/255
y_train_cat=to_categorical(y_train,num_classes=10)
y_test_cat=to_categorical(y_test,num_classes=10)

def Create_Model(Units,Activation,Regularizer,Initializer):
    model=Sequential()
    model.add(Flatten(input_shape=(32,32,3)))
    for i in Units:

model.add(Dense(units=i,activation=Activation,kernel_regularizer=Regularizer,kernel_initializer=Initializer))
    model.add(Dense(units=10,activation='softmax'))
    return model

Xavier_Model=Create_Model([256,256,128,64,32],'relu',regularizers.L2(0.01),initializers.glorot_normal())
Kaiming_Model=Create_Model([256,256,128,64,32],'relu',regularizers.L2(0.01),initializers.he_normal())

Xavier_Model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

Kaiming_Model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

X_history=Xavier_Model.fit(X_train,y_train_cat,epochs=20,validation_split=0.2,batch_size=64,verbose=1)
K_history=Kaiming_Model.fit(X_train,y_train_cat,epochs=20,validation_split=0.2,batch_size=64,verbose=1)

plt.figure(figsize=(8,4))
plt.subplot(1,2,1)
plt.title('Xavier Model')
plt.plot(X_history.history['accuracy'],label='Train')
plt.plot(X_history.history['val_accuracy'],label='Val')
```

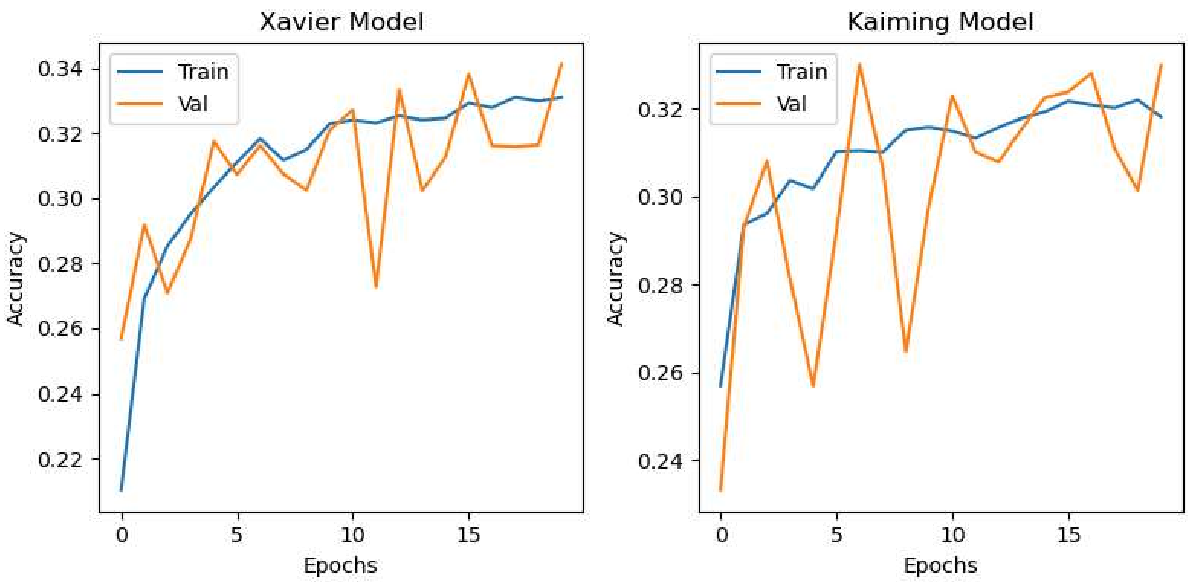
```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1,2,2)
plt.title("Kaiming Model")
plt.plot(K_history.history['accuracy'],label='Train')
plt.plot(K_history.history['val_accuracy'],label='Val')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.show()
```

Output

```
Epoch 1/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 12s 12ms/step - accuracy: 0.1753 - loss: 4.5799 - val_accuracy: 0.2568 - val_loss: 2.1965
Epoch 2/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.2683 - loss: 2.1490 - val_accuracy: 0.2918 - val_loss: 2.0886
Epoch 3/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.2785 - loss: 2.1010 - val_accuracy: 0.2708 - val_loss: 2.1645
Epoch 4/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.2930 - loss: 2.0703 - val_accuracy: 0.2875 - val_loss: 2.0605
Epoch 5/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3018 - loss: 2.0370 - val_accuracy: 0.3175 - val_loss: 2.0100
Epoch 6/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3138 - loss: 2.0193 - val_accuracy: 0.3072 - val_loss: 2.0152
Epoch 7/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.3181 - loss: 2.0030 - val_accuracy: 0.3162 - val_loss: 2.0363
Epoch 8/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3189 - loss: 1.9953 - val_accuracy: 0.3074 - val_loss: 2.0123
Epoch 9/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3095 - loss: 2.0008 - val_accuracy: 0.3024 - val_loss: 2.0085
Epoch 10/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3219 - loss: 1.9872 - val_accuracy: 0.3209 - val_loss: 1.9926
Epoch 11/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3267 - loss: 1.9668 - val_accuracy: 0.3271 - val_loss: 1.9793
Epoch 12/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3254 - loss: 1.9734 - val_accuracy: 0.2728 - val_loss: 2.0401
```

```
Epoch 13/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3173 - loss: 1.9727 - val_accuracy: 0.3333 - val_loss: 1.9782
Epoch 14/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3277 - loss: 1.9603 - val_accuracy: 0.3023 - val_loss: 1.9796
Epoch 15/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3278 - loss: 1.9560 - val_accuracy: 0.3127 - val_loss: 1.9992
Epoch 16/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3268 - loss: 1.9590 - val_accuracy: 0.3381 - val_loss: 1.9493
Epoch 17/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3346 - loss: 1.9453 - val_accuracy: 0.3161 - val_loss: 1.9742
Epoch 18/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3286 - loss: 1.9564 - val_accuracy: 0.3158 - val_loss: 1.9752
Epoch 19/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3303 - loss: 1.9441 - val_accuracy: 0.3163 - val_loss: 1.9863
Epoch 20/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3362 - loss: 1.9418 - val_accuracy: 0.3411 - val_loss: 1.9486
Epoch 1/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 9ms/step - accuracy: 0.2161 - loss: 6.8900 - val_accuracy: 0.2332 - val_loss: 2.5289
Epoch 2/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.2932 - loss: 2.3127 - val_accuracy: 0.2929 - val_loss: 2.1613
Epoch 3/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.2999 - loss: 2.1339 - val_accuracy: 0.3080 - val_loss: 2.0932
Epoch 4/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3014 - loss: 2.0805 - val_accuracy: 0.2812 - val_loss: 2.1769
Epoch 5/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.2982 - loss: 2.0538 - val_accuracy: 0.2568 - val_loss: 2.1206
Epoch 6/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 9ms/step - accuracy: 0.3108 - loss: 2.0204 - val_accuracy: 0.2920 - val_loss: 2.0560
Epoch 7/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3055 - loss: 2.0153 - val_accuracy: 0.3300 - val_loss: 1.9892
Epoch 8/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3094 - loss: 2.0109 - val_accuracy: 0.3065 - val_loss: 2.0200
Epoch 9/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.3175 - loss: 1.9938 - val_accuracy: 0.2647 - val_loss: 2.1042
Epoch 10/20
```

625/625 ━━━━━━━━━━━━━━━━━━━━ **5s** 8ms/step - accuracy: 0.3139 - loss: 2.0018 - val_accuracy: 0.2982 - val_loss: 2.0334
Epoch 11/20
625/625 ━━━━━━━━━━━━━━━━━━━━ **5s** 8ms/step - accuracy: 0.3118 - loss: 1.9817 - val_accuracy: 0.3229 - val_loss: 1.9753
Epoch 12/20
625/625 ━━━━━━━━━━━━━━━━━━━━ **5s** 8ms/step - accuracy: 0.3109 - loss: 1.9885 - val_accuracy: 0.3101 - val_loss: 1.9900
Epoch 13/20
625/625 ━━━━━━━━━━━━━━━━━━━━ **5s** 8ms/step - accuracy: 0.3145 - loss: 1.9757 - val_accuracy: 0.3078 - val_loss: 2.0006
Epoch 14/20
625/625 ━━━━━━━━━━━━━━━━━━━━ **5s** 8ms/step - accuracy: 0.3211 - loss: 1.9672 - val_accuracy: 0.3154 - val_loss: 1.9839
Epoch 15/20
625/625 ━━━━━━━━━━━━━━━━━━━━ **5s** 8ms/step - accuracy: 0.3202 - loss: 1.9609 - val_accuracy: 0.3225 - val_loss: 1.9691
Epoch 16/20
625/625 ━━━━━━━━━━━━━━━━━━━━ **5s** 8ms/step - accuracy: 0.3257 - loss: 1.9502 - val_accuracy: 0.3237 - val_loss: 1.9620
Epoch 17/20
625/625 ━━━━━━━━━━━━━━━━━━━━ **5s** 8ms/step - accuracy: 0.3237 - loss: 1.9451 - val_accuracy: 0.3280 - val_loss: 1.9464
Epoch 18/20
625/625 ━━━━━━━━━━━━━━━━━━━━ **5s** 8ms/step - accuracy: 0.3190 - loss: 1.9617 - val_accuracy: 0.3109 - val_loss: 1.9644
Epoch 19/20
625/625 ━━━━━━━━━━━━━━━━━━━━ **5s** 8ms/step - accuracy: 0.3213 - loss: 1.9592 - val_accuracy: 0.3013 - val_loss: 2.0361
Epoch 20/20
625/625 ━━━━━━━━━━━━━━━━━━━━ **5s** 8ms/step - accuracy: 0.3164 - loss: 1.9555 - val_accuracy: 0.3298 - val_loss: 1.9545



Exp 5

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report,confusion_matrix

(X_train,y_train),(X_test,y_test)=mnist.load_data()
X_train=X_train/255
X_test=X_test/255
y_test_cat=to_categorical(y_test,10)
y_train_cat=to_categorical(y_train,10)

model=Sequential()
model.add(Conv2D(28,(3,3),activation='relu',input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

model.fit(X_train,y_train_cat,epochs=10)

loss,acc=model.evaluate(X_test,y_test_cat)
print(f"Loss : {loss}")
print(f"Accuracy : {acc}")

pred=[np.argmax(i)for i in model.predict(X_test)]
sns.heatmap(confusion_matrix(y_test,pred),cmap='viridis',annot=True)
print("\nClassification Report\n",classification_report(y_test,pred))

test=X_test[36]
plt.imshow(test)
pred=np.argmax(model.predict(test.reshape(1,28,28)))
print(f"Actual value : {y_test[36]}")
print(f"Predicted value : {pred}")
```

Output

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 6ms/step - accuracy: 0.9033 - loss: 0.3217
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9802 - loss: 0.0651
```

Epoch 3/10
1875/1875 **11s** 6ms/step - accuracy: 0.9866 - loss: 0.0430
Epoch 4/10
1875/1875 **12s** 6ms/step - accuracy: 0.9898 - loss: 0.0321
Epoch 5/10
1875/1875 **12s** 6ms/step - accuracy: 0.9925 - loss: 0.0230
Epoch 6/10
1875/1875 **11s** 6ms/step - accuracy: 0.9940 - loss: 0.0194
Epoch 7/10
1875/1875 **11s** 6ms/step - accuracy: 0.9945 - loss: 0.0153
Epoch 8/10
1875/1875 **12s** 6ms/step - accuracy: 0.9961 - loss: 0.0116
Epoch 9/10
1875/1875 **11s** 6ms/step - accuracy: 0.9967 - loss: 0.0105
Epoch 10/10
1875/1875 **11s** 6ms/step - accuracy: 0.9962 - loss: 0.0104

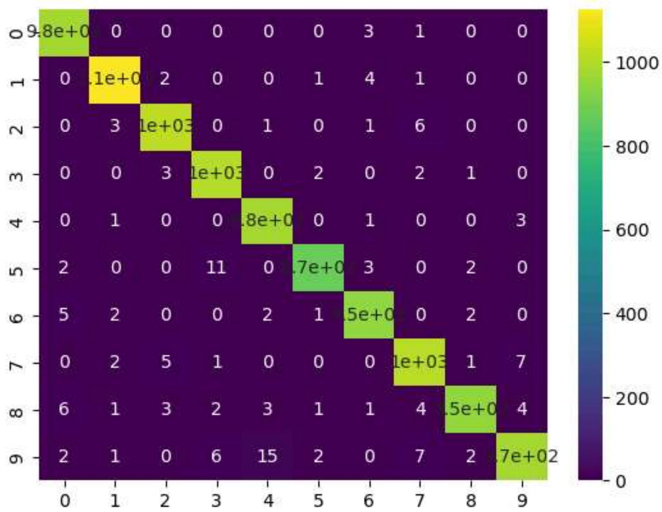
313/313 **1s** 3ms/step - accuracy: 0.9822 - loss: 0.0649
Loss : 0.049730781465768814
Accuracy : 0.98580002784729

[28]:

313/313 **1s** 2ms/step

Classification Report

	precision	recall	f1-score	support
0	0.98	1.00	0.99	980
1	0.99	0.99	0.99	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.98	0.99	0.99	982
5	0.99	0.98	0.99	892
6	0.99	0.99	0.99	958
7	0.98	0.98	0.98	1028
8	0.99	0.97	0.98	974
9	0.99	0.97	0.98	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000



1/1 **0s** 31ms/step

Actual value : 7

Predicted value : 7