DA1

August 30, 2024

0.1 NAME: SURESH BABU DHANUSH

0.2 REG NO: 21BIT0623

[31]:
```
from ucimlrepo import fetch_ucirepo
import pandas as pd
```

[32]:
```
dataset_id=53 # 46
dataset=fetch_ucirepo(id=dataset_id)
```

[105]:
```
dataset.data.keys()
```

[105]:
```
dict_keys(['ids', 'features', 'targets', 'original', 'headers'])
```

[140]:
```
X=pd.DataFrame(dataset.data.features)
X.head()
```

[140]:

	sepal length	sepal width	petal length	petal width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

[141]:
```
Y=pd.DataFrame(dataset.data.targets)
Y.head()
```

[141]:

	class
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa
4	Iris-setosa

[142]:
```
Y["class"].unique()
```

[142]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

1 Pre Processing

[143]:
```
Y.isnull().sum()
# no null values in output
# print(Y.size)->155
```

[143]:
```
class    0
dtype: int64
```

[144]:
```
# count for null values in columns
na_cols=[]
cols=X.columns

for col in cols:
    current_col=X[col]
    na_count=current_col.isna().sum()
    if na_count>0:
        na_cols.append(col)


print(f"columns with null values:\n{na_cols}")
```

```
columns with null values:
[]
```

[145]:
```
# null values before replacing with median
X.isna().any()
```

[145]:
```
sepal length    False
sepal width     False
petal length    False
petal width     False
dtype: bool
```

[146]:
```
for na_col in na_cols:
    na_mask=X[na_col].isna()==True
    X.loc[na_mask,na_col]=X[na_col].median()
```

[147]:
```
# after replace
X.isna().any()
```

[147]:
```
sepal length    False
sepal width     False
```

```
petal length    False
petal width     False
dtype: bool
```

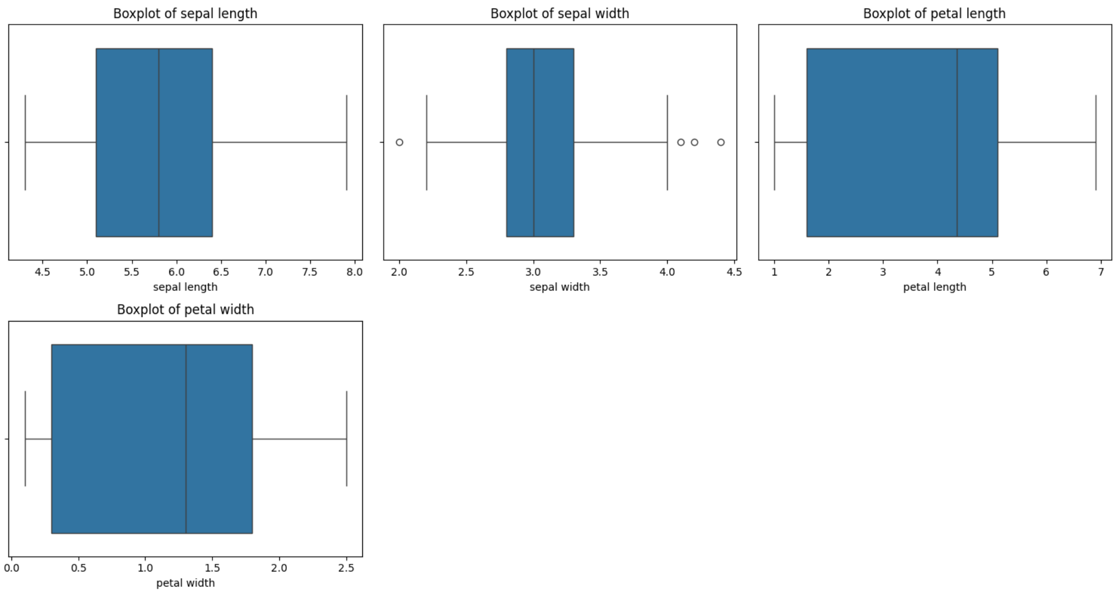
[148]:
```
X.shape
```

[148]:
```
(150, 4)
```

[ ]:
```

```

[149]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math

def plot_outliers(X: pd.DataFrame):
    num_cols = len(X.columns)
    num_rows = math.ceil(num_cols / 3)

    plt.figure(figsize=(15, num_rows * 4))

    for i, col in enumerate(X.columns, 1):
        if X[col].dtype in ['int64', 'float64']:
            plt.subplot(num_rows, 3, i)
            sns.boxplot(x=X[col])
            plt.title(f'Boxplot of {col}')
            plt.xlabel(col)

    plt.tight_layout()
    plt.show()
```

[150]:
```
plot_outliers(X)
```



[ ]:

```
```

[ ]:

```
```

[151]:

```
df.head()
```

[151]:

	sepal length	sepal width	petal length	petal width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

[ ]:

```
```

2 MODEL BUILDING

[152]:

```
# classes=Y["class"].unique()
Y.shape, X.shape
```

[152]: ((150, 1), (150, 4))

3 Encode stuff

[153]:
```
from sklearn.preprocessing import LabelEncoder
```

[154]:
```
label_encoder=LabelEncoder()
Y["class"]=label_encoder.fit_transform(Y["class"])
Y["class"].unique()
```

[154]:
```
array([0, 1, 2])
```

[155]:
```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

[156]:
```
# Assuming X and Y are already defined as DataFrames
# Convert Y["class"] to numpy array
Y = Y["class"].values

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
↪random_state=42)


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[157]:
```
# Define the model
model = Sequential()

# Add input layer (with 4 input features) and first hidden layer
model.add(Dense(16, input_shape=(4,), activation='relu'))

# Add second hidden layer
model.add(Dense(8, activation='relu'))

# Add output layer (3 output classes, corresponding to unique classes in Y)
model.add(Dense(3, activation='softmax'))
```

```
/home/munke/.venvs/pokedex/lib/python3.11/site-
packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
2024-08-30 23:17:02.197198: E
external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:282] failed call to
cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detected
```

[158]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 16)	80
dense_1 (Dense)	(None, 8)	136
dense_2 (Dense)	(None, 3)	27

Total params: 243 (972.00 B)

Trainable params: 243 (972.00 B)

Non-trainable params: 0 (0.00 B)

[159]:
```
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

[160]:
```
# Train the model
history = model.fit(X_train, Y_train, epochs=100, batch_size=8,␣
↪validation_split=0.2, verbose=1)
```

```
Epoch 1/100
12/12              1s 8ms/step -
accuracy: 0.2402 - loss: 1.0614 - val_accuracy: 0.5417 - val_loss: 0.9560
Epoch 2/100
12/12              0s 2ms/step -
accuracy: 0.4586 - loss: 0.9883 - val_accuracy: 0.6667 - val_loss: 0.8877
Epoch 3/100
12/12              0s 2ms/step -
accuracy: 0.6200 - loss: 0.8944 - val_accuracy: 0.7083 - val_loss: 0.8249
Epoch 4/100
12/12              0s 2ms/step -
accuracy: 0.6475 - loss: 0.7975 - val_accuracy: 0.7083 - val_loss: 0.7696
Epoch 5/100
12/12              0s 2ms/step -
```

```
accuracy: 0.7040 - loss: 0.7783 - val_accuracy: 0.7917 - val_loss: 0.7225
Epoch 6/100
12/12            0s 2ms/step -
accuracy: 0.7078 - loss: 0.7648 - val_accuracy: 0.7917 - val_loss: 0.6856
Epoch 7/100
12/12            0s 2ms/step -
accuracy: 0.7780 - loss: 0.6141 - val_accuracy: 0.7917 - val_loss: 0.6489
Epoch 8/100
12/12            0s 2ms/step -
accuracy: 0.7742 - loss: 0.6519 - val_accuracy: 0.8333 - val_loss: 0.6207
Epoch 9/100
12/12            0s 2ms/step -
accuracy: 0.7204 - loss: 0.6696 - val_accuracy: 0.8333 - val_loss: 0.5965
Epoch 10/100
12/12            0s 2ms/step -
accuracy: 0.7984 - loss: 0.5416 - val_accuracy: 0.8333 - val_loss: 0.5743
Epoch 11/100
12/12            0s 2ms/step -
accuracy: 0.7753 - loss: 0.5908 - val_accuracy: 0.8333 - val_loss: 0.5558
Epoch 12/100
12/12            0s 2ms/step -
accuracy: 0.7065 - loss: 0.5720 - val_accuracy: 0.8333 - val_loss: 0.5407
Epoch 13/100
12/12            0s 2ms/step -
accuracy: 0.7835 - loss: 0.5672 - val_accuracy: 0.8333 - val_loss: 0.5258
Epoch 14/100
12/12            0s 2ms/step -
accuracy: 0.8194 - loss: 0.4725 - val_accuracy: 0.8333 - val_loss: 0.5131
Epoch 15/100
12/12            0s 2ms/step -
accuracy: 0.7365 - loss: 0.5237 - val_accuracy: 0.8333 - val_loss: 0.5020
Epoch 16/100
12/12            0s 2ms/step -
accuracy: 0.7917 - loss: 0.4592 - val_accuracy: 0.8333 - val_loss: 0.4911
Epoch 17/100
12/12            0s 2ms/step -
accuracy: 0.7532 - loss: 0.5802 - val_accuracy: 0.8333 - val_loss: 0.4831
Epoch 18/100
12/12            0s 2ms/step -
accuracy: 0.7518 - loss: 0.5658 - val_accuracy: 0.8333 - val_loss: 0.4737
Epoch 19/100
12/12            0s 2ms/step -
accuracy: 0.7831 - loss: 0.4692 - val_accuracy: 0.8333 - val_loss: 0.4663
Epoch 20/100
12/12            0s 2ms/step -
accuracy: 0.7929 - loss: 0.4448 - val_accuracy: 0.8333 - val_loss: 0.4581
Epoch 21/100
12/12            0s 2ms/step -
```

```
accuracy: 0.8323 - loss: 0.4531 - val_accuracy: 0.8750 - val_loss: 0.4503
Epoch 22/100
12/12            0s 2ms/step -
accuracy: 0.8325 - loss: 0.4391 - val_accuracy: 0.8750 - val_loss: 0.4426
Epoch 23/100
12/12            0s 2ms/step -
accuracy: 0.7994 - loss: 0.4496 - val_accuracy: 0.8750 - val_loss: 0.4369
Epoch 24/100
12/12            0s 2ms/step -
accuracy: 0.8201 - loss: 0.4579 - val_accuracy: 0.8750 - val_loss: 0.4297
Epoch 25/100
12/12            0s 2ms/step -
accuracy: 0.8496 - loss: 0.3683 - val_accuracy: 0.8750 - val_loss: 0.4217
Epoch 26/100
12/12            0s 2ms/step -
accuracy: 0.8298 - loss: 0.4037 - val_accuracy: 0.8333 - val_loss: 0.4158
Epoch 27/100
12/12            0s 2ms/step -
accuracy: 0.7625 - loss: 0.4475 - val_accuracy: 0.8333 - val_loss: 0.4089
Epoch 28/100
12/12            0s 2ms/step -
accuracy: 0.8491 - loss: 0.4037 - val_accuracy: 0.8333 - val_loss: 0.4021
Epoch 29/100
12/12            0s 2ms/step -
accuracy: 0.8632 - loss: 0.3342 - val_accuracy: 0.8333 - val_loss: 0.3952
Epoch 30/100
12/12            0s 2ms/step -
accuracy: 0.8271 - loss: 0.4013 - val_accuracy: 0.8333 - val_loss: 0.3877
Epoch 31/100
12/12            0s 2ms/step -
accuracy: 0.8721 - loss: 0.3874 - val_accuracy: 0.8333 - val_loss: 0.3807
Epoch 32/100
12/12            0s 2ms/step -
accuracy: 0.8295 - loss: 0.4281 - val_accuracy: 0.8750 - val_loss: 0.3750
Epoch 33/100
12/12            0s 2ms/step -
accuracy: 0.8835 - loss: 0.3419 - val_accuracy: 0.8750 - val_loss: 0.3654
Epoch 34/100
12/12            0s 2ms/step -
accuracy: 0.8287 - loss: 0.3278 - val_accuracy: 0.8750 - val_loss: 0.3598
Epoch 35/100
12/12            0s 2ms/step -
accuracy: 0.8789 - loss: 0.3281 - val_accuracy: 0.8750 - val_loss: 0.3536
Epoch 36/100
12/12            0s 2ms/step -
accuracy: 0.9273 - loss: 0.3058 - val_accuracy: 0.8750 - val_loss: 0.3441
Epoch 37/100
12/12            0s 2ms/step -
```

```
accuracy: 0.8478 - loss: 0.3227 - val_accuracy: 0.8750 - val_loss: 0.3421
Epoch 38/100
12/12            0s 2ms/step -
accuracy: 0.9068 - loss: 0.2971 - val_accuracy: 0.8750 - val_loss: 0.3335
Epoch 39/100
12/12            0s 2ms/step -
accuracy: 0.9444 - loss: 0.2841 - val_accuracy: 0.8750 - val_loss: 0.3238
Epoch 40/100
12/12            0s 2ms/step -
accuracy: 0.9093 - loss: 0.2756 - val_accuracy: 0.9167 - val_loss: 0.3185
Epoch 41/100
12/12            0s 2ms/step -
accuracy: 0.8593 - loss: 0.3597 - val_accuracy: 0.9167 - val_loss: 0.3146
Epoch 42/100
12/12            0s 2ms/step -
accuracy: 0.9331 - loss: 0.2741 - val_accuracy: 0.9167 - val_loss: 0.3067
Epoch 43/100
12/12            0s 2ms/step -
accuracy: 0.9382 - loss: 0.2116 - val_accuracy: 0.9167 - val_loss: 0.2968
Epoch 44/100
12/12            0s 2ms/step -
accuracy: 0.9553 - loss: 0.2194 - val_accuracy: 0.9167 - val_loss: 0.2895
Epoch 45/100
12/12            0s 2ms/step -
accuracy: 0.9254 - loss: 0.2141 - val_accuracy: 0.9167 - val_loss: 0.2870
Epoch 46/100
12/12            0s 2ms/step -
accuracy: 0.9273 - loss: 0.2450 - val_accuracy: 0.9167 - val_loss: 0.2859
Epoch 47/100
12/12            0s 2ms/step -
accuracy: 0.9606 - loss: 0.1868 - val_accuracy: 0.9167 - val_loss: 0.2722
Epoch 48/100
12/12            0s 2ms/step -
accuracy: 0.9447 - loss: 0.2350 - val_accuracy: 0.9167 - val_loss: 0.2683
Epoch 49/100
12/12            0s 2ms/step -
accuracy: 0.9721 - loss: 0.1845 - val_accuracy: 0.9583 - val_loss: 0.2604
Epoch 50/100
12/12            0s 2ms/step -
accuracy: 0.9253 - loss: 0.2374 - val_accuracy: 0.9583 - val_loss: 0.2635
Epoch 51/100
12/12            0s 2ms/step -
accuracy: 0.9452 - loss: 0.1924 - val_accuracy: 0.9583 - val_loss: 0.2568
Epoch 52/100
12/12            0s 2ms/step -
accuracy: 0.9495 - loss: 0.1807 - val_accuracy: 0.9583 - val_loss: 0.2461
Epoch 53/100
12/12            0s 2ms/step -
```

```
accuracy: 0.9444 - loss: 0.1845 - val_accuracy: 0.9583 - val_loss: 0.2457
Epoch 54/100
12/12            0s 2ms/step -
accuracy: 0.9149 - loss: 0.2070 - val_accuracy: 0.9583 - val_loss: 0.2410
Epoch 55/100
12/12            0s 2ms/step -
accuracy: 0.9352 - loss: 0.1779 - val_accuracy: 0.9583 - val_loss: 0.2428
Epoch 56/100
12/12            0s 2ms/step -
accuracy: 0.9591 - loss: 0.1571 - val_accuracy: 0.9583 - val_loss: 0.2304
Epoch 57/100
12/12            0s 2ms/step -
accuracy: 0.9618 - loss: 0.1537 - val_accuracy: 0.9583 - val_loss: 0.2300
Epoch 58/100
12/12            0s 2ms/step -
accuracy: 0.9524 - loss: 0.1583 - val_accuracy: 0.9583 - val_loss: 0.2295
Epoch 59/100
12/12            0s 2ms/step -
accuracy: 0.9787 - loss: 0.1391 - val_accuracy: 0.9583 - val_loss: 0.2252
Epoch 60/100
12/12            0s 2ms/step -
accuracy: 0.9363 - loss: 0.1733 - val_accuracy: 0.9583 - val_loss: 0.2198
Epoch 61/100
12/12            0s 2ms/step -
accuracy: 0.9371 - loss: 0.1542 - val_accuracy: 0.9583 - val_loss: 0.2173
Epoch 62/100
12/12            0s 2ms/step -
accuracy: 0.9177 - loss: 0.1824 - val_accuracy: 0.9583 - val_loss: 0.2188
Epoch 63/100
12/12            0s 2ms/step -
accuracy: 0.9397 - loss: 0.1484 - val_accuracy: 0.9583 - val_loss: 0.2083
Epoch 64/100
12/12            0s 2ms/step -
accuracy: 0.9822 - loss: 0.1092 - val_accuracy: 0.9583 - val_loss: 0.2096
Epoch 65/100
12/12            0s 2ms/step -
accuracy: 0.9759 - loss: 0.1246 - val_accuracy: 0.9583 - val_loss: 0.2049
Epoch 66/100
12/12            0s 2ms/step -
accuracy: 0.9630 - loss: 0.1325 - val_accuracy: 0.9583 - val_loss: 0.2116
Epoch 67/100
12/12            0s 2ms/step -
accuracy: 0.9732 - loss: 0.1076 - val_accuracy: 0.9583 - val_loss: 0.2071
Epoch 68/100
12/12            0s 2ms/step -
accuracy: 0.9313 - loss: 0.1451 - val_accuracy: 0.9583 - val_loss: 0.2107
Epoch 69/100
12/12            0s 2ms/step -
```

```
accuracy: 0.9656 - loss: 0.1035 - val_accuracy: 0.9583 - val_loss: 0.2110
Epoch 70/100
12/12            0s 2ms/step -
accuracy: 0.9432 - loss: 0.1381 - val_accuracy: 0.9583 - val_loss: 0.2039
Epoch 71/100
12/12            0s 2ms/step -
accuracy: 0.8887 - loss: 0.1743 - val_accuracy: 0.9583 - val_loss: 0.2077
Epoch 72/100
12/12            0s 2ms/step -
accuracy: 0.9526 - loss: 0.1175 - val_accuracy: 0.9583 - val_loss: 0.1920
Epoch 73/100
12/12            0s 2ms/step -
accuracy: 0.9420 - loss: 0.1372 - val_accuracy: 0.9583 - val_loss: 0.1953
Epoch 74/100
12/12            0s 2ms/step -
accuracy: 0.9508 - loss: 0.1096 - val_accuracy: 0.9583 - val_loss: 0.1993
Epoch 75/100
12/12            0s 2ms/step -
accuracy: 0.9655 - loss: 0.1139 - val_accuracy: 0.9583 - val_loss: 0.1931
Epoch 76/100
12/12            0s 2ms/step -
accuracy: 0.9567 - loss: 0.1130 - val_accuracy: 0.9583 - val_loss: 0.1935
Epoch 77/100
12/12            0s 2ms/step -
accuracy: 0.9414 - loss: 0.1290 - val_accuracy: 0.9583 - val_loss: 0.1926
Epoch 78/100
12/12            0s 2ms/step -
accuracy: 0.9672 - loss: 0.0994 - val_accuracy: 0.9583 - val_loss: 0.1919
Epoch 79/100
12/12            0s 2ms/step -
accuracy: 0.9248 - loss: 0.1214 - val_accuracy: 0.9583 - val_loss: 0.1974
Epoch 80/100
12/12            0s 2ms/step -
accuracy: 0.9753 - loss: 0.0911 - val_accuracy: 0.9583 - val_loss: 0.1919
Epoch 81/100
12/12            0s 2ms/step -
accuracy: 0.9422 - loss: 0.1151 - val_accuracy: 0.9583 - val_loss: 0.1921
Epoch 82/100
12/12            0s 2ms/step -
accuracy: 0.9406 - loss: 0.1140 - val_accuracy: 0.9583 - val_loss: 0.1982
Epoch 83/100
12/12            0s 2ms/step -
accuracy: 0.9400 - loss: 0.1175 - val_accuracy: 0.9583 - val_loss: 0.1965
Epoch 84/100
12/12            0s 2ms/step -
accuracy: 0.9807 - loss: 0.1023 - val_accuracy: 0.9583 - val_loss: 0.1877
Epoch 85/100
12/12            0s 2ms/step -
```

```
accuracy: 0.9671 - loss: 0.0961 - val_accuracy: 0.9583 - val_loss: 0.1963
Epoch 86/100
12/12            0s 2ms/step -
accuracy: 0.9282 - loss: 0.1167 - val_accuracy: 0.9583 - val_loss: 0.2019
Epoch 87/100
12/12            0s 2ms/step -
accuracy: 0.9732 - loss: 0.0728 - val_accuracy: 0.9583 - val_loss: 0.1947
Epoch 88/100
12/12            0s 2ms/step -
accuracy: 0.9591 - loss: 0.1061 - val_accuracy: 0.9583 - val_loss: 0.1915
Epoch 89/100
12/12            0s 2ms/step -
accuracy: 0.9802 - loss: 0.1030 - val_accuracy: 0.9583 - val_loss: 0.1892
Epoch 90/100
12/12            0s 2ms/step -
accuracy: 0.9905 - loss: 0.0821 - val_accuracy: 0.9583 - val_loss: 0.1908
Epoch 91/100
12/12            0s 2ms/step -
accuracy: 0.9617 - loss: 0.0896 - val_accuracy: 0.9583 - val_loss: 0.1998
Epoch 92/100
12/12            0s 2ms/step -
accuracy: 0.9519 - loss: 0.0895 - val_accuracy: 0.9583 - val_loss: 0.1972
Epoch 93/100
12/12            0s 2ms/step -
accuracy: 0.9705 - loss: 0.0780 - val_accuracy: 0.9583 - val_loss: 0.1955
Epoch 94/100
12/12            0s 2ms/step -
accuracy: 0.9726 - loss: 0.0947 - val_accuracy: 0.9583 - val_loss: 0.1923
Epoch 95/100
12/12            0s 2ms/step -
accuracy: 0.9742 - loss: 0.0821 - val_accuracy: 0.9583 - val_loss: 0.1954
Epoch 96/100
12/12            0s 2ms/step -
accuracy: 0.9717 - loss: 0.0731 - val_accuracy: 0.9583 - val_loss: 0.1940
Epoch 97/100
12/12            0s 2ms/step -
accuracy: 0.9599 - loss: 0.0905 - val_accuracy: 0.9583 - val_loss: 0.1983
Epoch 98/100
12/12            0s 2ms/step -
accuracy: 0.9633 - loss: 0.0879 - val_accuracy: 0.9583 - val_loss: 0.1981
Epoch 99/100
12/12            0s 2ms/step -
accuracy: 0.9484 - loss: 0.0941 - val_accuracy: 0.9583 - val_loss: 0.1977
Epoch 100/100
12/12            0s 2ms/step -
accuracy: 0.9566 - loss: 0.0741 - val_accuracy: 0.9583 - val_loss: 0.1974
```

4 MODEL EVALUATION

[163]:
```
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, Y_test, verbose=0)
print(f"Test accuracy using tf keras default evaluation: {test_accuracy:.2f}")
```

```
Test accuracy using tf keras default evaluation: 0.97
```

[164]:
```
from sklearn.metrics import precision_score, recall_score, f1_score,␣
↪confusion_matrix
```

[165]:
```
# Predict classes for the test data
Y_pred = model.predict(X_test)
Y_pred_classes = np.argmax(Y_pred, axis=1)  # Convert predictions to class␣
↪labels
```

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

[166]:
```
# Calculate precision
precision = precision_score(Y_test, Y_pred_classes, average='weighted')
print(f"Precision: {precision:.2f}")

# Calculate recall
recall = recall_score(Y_test, Y_pred_classes, average='weighted')
print(f"Recall: {recall:.2f}")

# Calculate F1-score
f1 = f1_score(Y_test, Y_pred_classes, average='weighted')
print(f"F1-score: {f1:.2f}")
```

```
Precision: 0.97
Recall: 0.97
F1-score: 0.97
```

[ ]:
```
```