```
#using Scikit -learn for Iris  Classification
#importing libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score


#load Iris dataset
iris=load_iris()
X=iris.data
Y=iris.target

#split the dataset 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)


#feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#train a logistic regression model
model=LogisticRegression(max_iter=1000)
model.fit(X_train,Y_train)

# Predict on the test set
Y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy using Logistic Regression: {accuracy:.2f}")
```

```
Accuracy using Logistic Regression: 1.00
```

```
X.shape
```

```
(150, 4)
```

```
X.shape
```

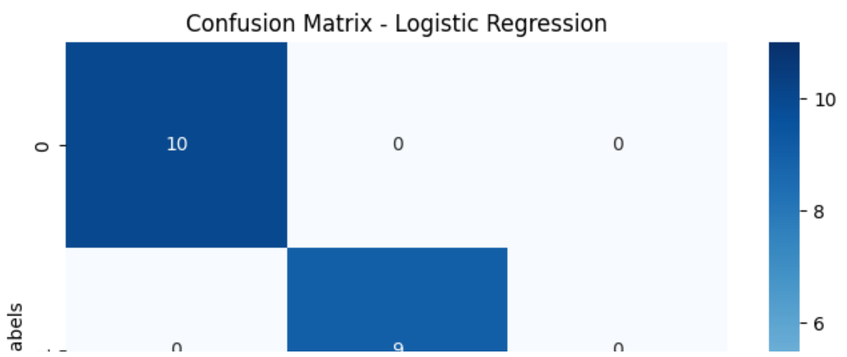
```
(150, 4)
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Generate predictions using the logistic regression model
Y_pred_lr = model.predict(X_test)

# Calculate the confusion matrix
cm = confusion_matrix(Y_test, Y_pred_lr)

# Create a heatmap for the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```



```
#Using TensorFlow for Iris Classification:
#importing libraries
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset 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)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build a simple neural network model using TensorFlow/Keras
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu', input_shape=(4,)),
    tf.keras.layers.Dense(20,activation="relu"),
    tf.keras.layers.Dense(10,activation="relu"),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
history=model.fit(X_train, y_train, epochs=50, batch_size=32,validation_split=0.2,verbose=1)

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, Y_test)
print(f"Accuracy using TensorFlow: {accuracy:.2f}")
```

```
Epoch 1/50
3/3 [==============================] - 3s 176ms/step - loss: 1.2025 - accuracy: 0.2292 - val_loss: 1.1401 - val_accuracy: 0.2917
Epoch 2/50
3/3 [==============================] - 0s 44ms/step - loss: 1.1815 - accuracy: 0.2396 - val_loss: 1.1302 - val_accuracy: 0.2917
Epoch 3/50
3/3 [==============================] - 0s 38ms/step - loss: 1.1644 - accuracy: 0.2604 - val_loss: 1.1200 - val_accuracy: 0.2917
Epoch 4/50
3/3 [==============================] - 0s 58ms/step - loss: 1.1460 - accuracy: 0.2708 - val_loss: 1.1096 - val_accuracy: 0.2917
Epoch 5/50
3/3 [==============================] - 0s 52ms/step - loss: 1.1284 - accuracy: 0.2917 - val_loss: 1.0991 - val_accuracy: 0.2917
Epoch 6/50
3/3 [==============================] - 0s 28ms/step - loss: 1.1113 - accuracy: 0.3021 - val_loss: 1.0887 - val_accuracy: 0.3333
Epoch 7/50
3/3 [==============================] - 0s 38ms/step - loss: 1.0934 - accuracy: 0.3125 - val_loss: 1.0783 - val_accuracy: 0.3750
Epoch 8/50
3/3 [==============================] - 0s 37ms/step - loss: 1.0760 - accuracy: 0.3542 - val_loss: 1.0677 - val_accuracy: 0.4167
Epoch 9/50
3/3 [==============================] - 0s 40ms/step - loss: 1.0593 - accuracy: 0.3958 - val_loss: 1.0572 - val_accuracy: 0.4167
Epoch 10/50
3/3 [==============================] - 0s 57ms/step - loss: 1.0404 - accuracy: 0.4583 - val_loss: 1.0469 - val_accuracy: 0.5000
Epoch 11/50
3/3 [==============================] - 0s 40ms/step - loss: 1.0238 - accuracy: 0.5000 - val_loss: 1.0365 - val_accuracy: 0.5000
Epoch 12/50
3/3 [==============================] - 0s 38ms/step - loss: 1.0064 - accuracy: 0.5625 - val_loss: 1.0258 - val_accuracy: 0.5833
Epoch 13/50
3/3 [==============================] - 0s 49ms/step - loss: 0.9883 - accuracy: 0.6250 - val_loss: 1.0141 - val_accuracy: 0.6250
Epoch 14/50
3/3 [==============================] - 0s 55ms/step - loss: 0.9690 - accuracy: 0.6875 - val_loss: 1.0023 - val_accuracy: 0.6667
Epoch 15/50
3/3 [==============================] - 0s 36ms/step - loss: 0.9495 - accuracy: 0.7292 - val_loss: 0.9902 - val_accuracy: 0.6667
Epoch 16/50
```

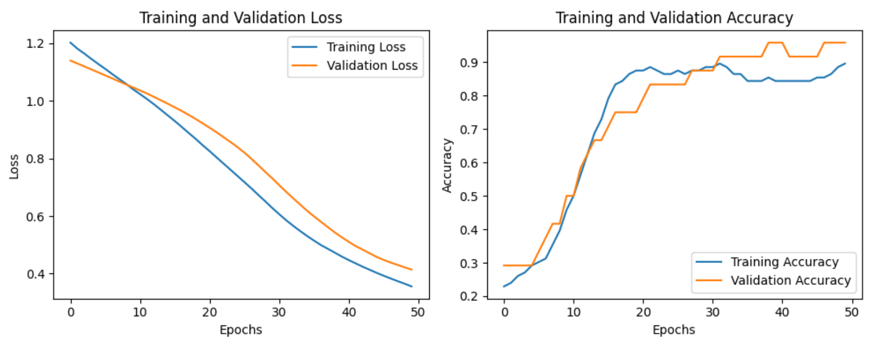
```
3/3 [==============================] - 0s 42ms/step - loss: 0.9299 - accuracy: 0.7917 - val_loss: 0.9778 - val_accuracy: 0.7083
Epoch 17/50
3/3 [==============================] - 0s 79ms/step - loss: 0.9094 - accuracy: 0.8333 - val_loss: 0.9651 - val_accuracy: 0.7500
Epoch 18/50
3/3 [==============================] - 0s 44ms/step - loss: 0.8884 - accuracy: 0.8438 - val_loss: 0.9514 - val_accuracy: 0.7500
Epoch 19/50
3/3 [==============================] - 0s 38ms/step - loss: 0.8680 - accuracy: 0.8646 - val_loss: 0.9372 - val_accuracy: 0.7500
Epoch 20/50
3/3 [==============================] - 0s 50ms/step - loss: 0.8461 - accuracy: 0.8750 - val_loss: 0.9222 - val_accuracy: 0.7500
Epoch 21/50
3/3 [==============================] - 0s 50ms/step - loss: 0.8255 - accuracy: 0.8750 - val_loss: 0.9069 - val_accuracy: 0.7917
Epoch 22/50
3/3 [==============================] - 0s 58ms/step - loss: 0.8037 - accuracy: 0.8854 - val_loss: 0.8909 - val_accuracy: 0.8333
Epoch 23/50
3/3 [==============================] - 0s 37ms/step - loss: 0.7822 - accuracy: 0.8750 - val_loss: 0.8742 - val_accuracy: 0.8333
Epoch 24/50
3/3 [==============================] - 0s 37ms/step - loss: 0.7610 - accuracy: 0.8646 - val_loss: 0.8572 - val_accuracy: 0.8333
Epoch 25/50
3/3 [==============================] - 0s 38ms/step - loss: 0.7393 - accuracy: 0.8646 - val_loss: 0.8397 - val_accuracy: 0.8333
Epoch 26/50
3/3 [==============================] - 0s 38ms/step - loss: 0.7178 - accuracy: 0.8750 - val_loss: 0.8203 - val_accuracy: 0.8333
Epoch 27/50
3/3 [==============================] - 0s 37ms/step - loss: 0.6961 - accuracy: 0.8646 - val_loss: 0.7989 - val_accuracy: 0.8333
Epoch 28/50
3/3 [==============================] - 0s 36ms/step - loss: 0.6732 - accuracy: 0.8750 - val_loss: 0.7764 - val_accuracy: 0.8750
Epoch 29/50
3/3 [==============================] - 0s 38ms/step - loss: 0.6510 - accuracy: 0.8750 - val_loss: 0.7534 - val_accuracy: 0.8750
```

```
# Plotting training and validation loss
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

# Plotting training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

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


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epochs
Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs