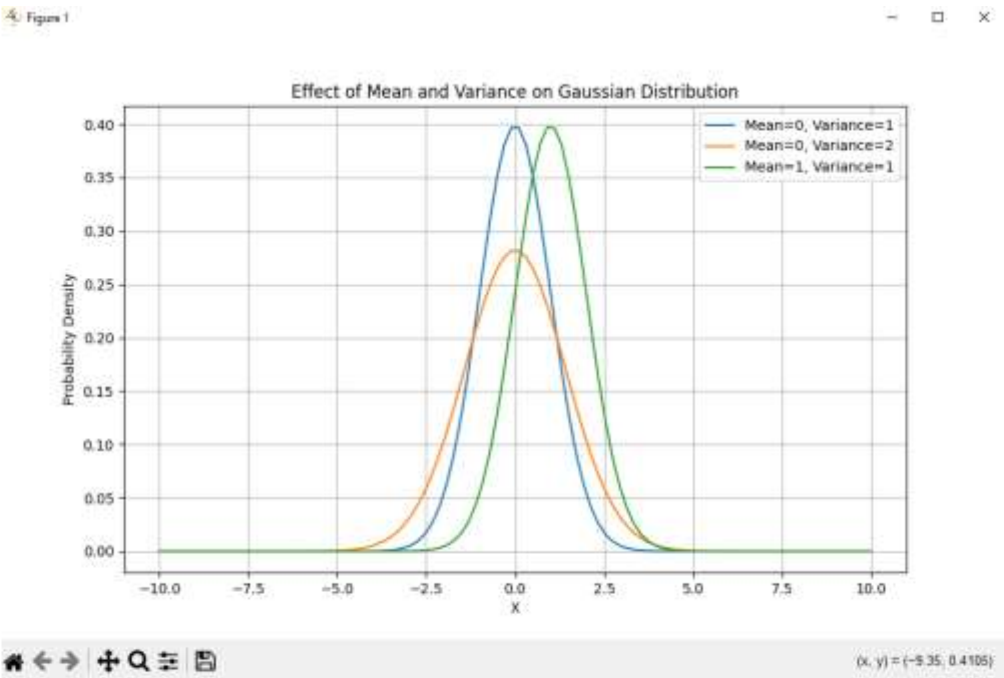
Lab 7

Statistics, Machine Learning, Deep Learning

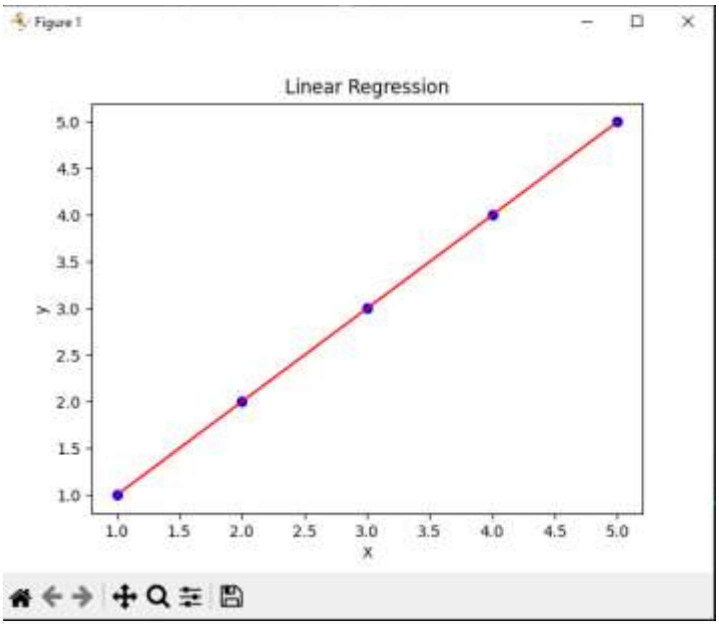
1. **Write a Python program that computes the value of the Gaussian distribution at a given vector X. Hence, plot the effect of varying mean and variance to the normal distribution.**

```
import numpy as np

import matplotlib.pyplot as plt

def gaussian_distribution(x, mean, variance):

    return (1 / (np.sqrt(2 * np.pi * variance))) * np.exp(-((x - mean) ** 2) / (2 * variance))

x_values = np.linspace(-10, 10, 100)

means = [0, 0, 1]

variances = [1, 2, 1]

plt.figure(figsize=(10, 6))

for mean, variance in zip(means, variances):

    plt.plot(x_values, gaussian_distribution(x_values, mean, variance), label=f'Mean={mean}, Variance={variance}')

plt.title('Effect of Mean and Variance on Gaussian Distribution')

plt.xlabel('X')

plt.ylabel('Probability Density')

plt.legend()

plt.grid(True)

plt.show()
```



2. Write a python program to implement linear regression.

```
import numpy as np

from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt

X = np.array([[1], [2], [3], [4], [5]])

y = np.array([1, 2, 3, 4, 5])

model = LinearRegression()

model.fit(X, y)

y_pred = model.predict(X)

plt.scatter(X, y, color='blue')

plt.plot(X, y_pred, color='red')

plt.title('Linear Regression')

plt.xlabel('X')

plt.ylabel('y')

plt.show()
```

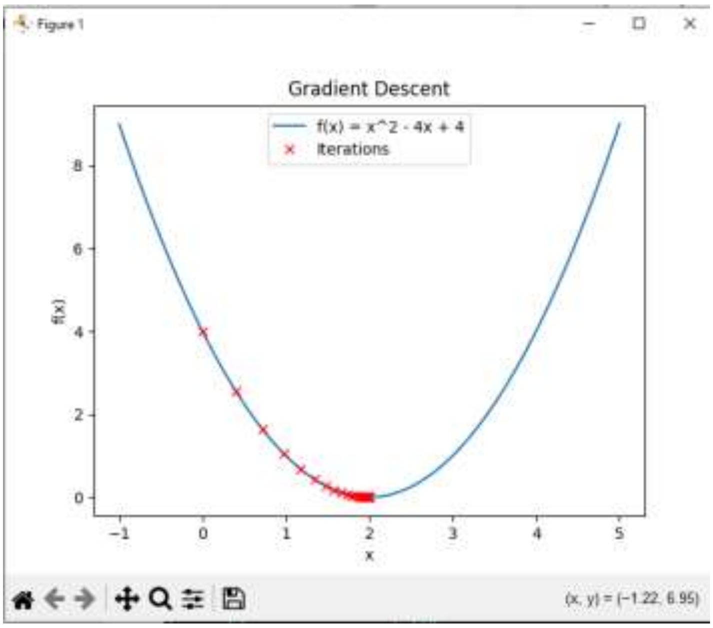


3. Write a python program to implement gradient descent.

```
import numpy as np

import matplotlib.pyplot as plt

def f(x):

    return x**2 - 4*x + 4

def df(x):

    return 2*x - 4
```

```
def gradient_descent(initial_x, learning_rate, num_iterations):
    x = initial_x
    x_history = [x]
    for i in range(num_iterations):
        gradient = df(x)
        x = x - learning_rate * gradient
        x_history.append(x)
    return x, x_history
initial_x = 0
learning_rate = 0.1
num_iterations = 50
x, x_history = gradient_descent(initial_x, learning_rate, num_iterations)
print("Local minimum: {:.2f}".format(x))
x_vals = np.linspace(-1, 5, 100)
plt.plot(x_vals, f(x_vals), label='f(x) = x^2 - 4x + 4')
plt.plot(x_history, f(np.array(x_history)), 'rx', label='Iterations')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Gradient Descent')
plt.legend()
plt.show()
```

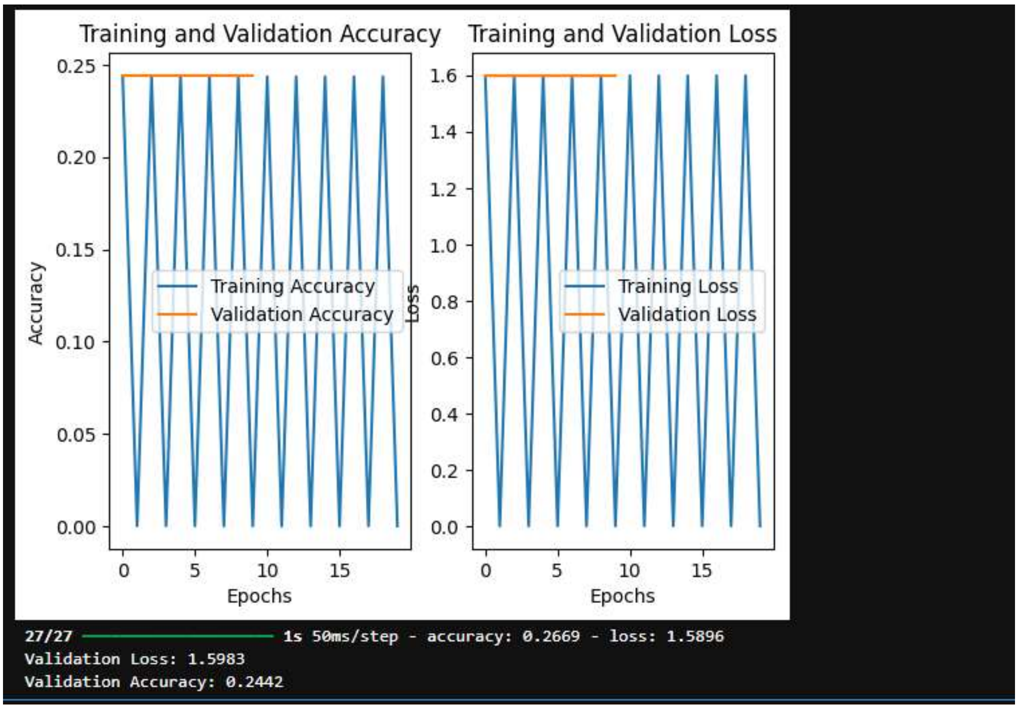
```
PS C:\Users\Rudradeep\Desktop\Python\pythonCollage> python -u
Local minimum: 2.00
```



4. **Write a python program to classify different flower images using MLP.**

```
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.layers import Flatten, Dense, Dropout, Input

dataset_path = r"C:\Users\Rudradeep\Desktop\Python\pythonCollage\Collage_python\Lab 7\flowers"

data_gen = ImageDataGenerator(rescale=1.0/255, validation_split=0.2)
train_data = data_gen.flow_from_directory(
    directory=dataset_path,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_data = data_gen.flow_from_directory(
    directory=dataset_path,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
model = Sequential([
    Input(shape=(64, 64, 3)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(train_data.num_classes, activation='softmax')
])
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
steps_per_epoch = len(train_data)
validation_steps = len(val_data)

history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=20,
    verbose=1,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps
)
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

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

plt.show()

# Final Evaluation
loss, accuracy = model.evaluate(val_data)
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy:.4f}")
```



5. **Write a python program to classify different flower images using the SVM classifier.**

```
import os

import numpy as np

import cv2

from skimage.feature import hog

from skimage import exposure

from sklearn import svm

from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.image import ImageDataGenerator


dataset_path = r"C:\Users\Rudradeep\Desktop\Python\pythonCollage\Collage_python\Lab 7\flowers"
image_size = (64, 64)
batch_size = 32
data_gen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
train_data_gen = data_gen.flow_from_directory(
    directory=dataset_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)
val_data_gen = data_gen.flow_from_directory(
    directory=dataset_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)
class_names = list(train_data_gen.class_indices.keys())
def extract_features_and_labels(data_gen):
    features = []
    labels = []
    for batch_images, batch_labels in data_gen:
        for image, label in zip(batch_images, batch_labels):
            gray_image = cv2.cvtColor((image * 255).astype(np.uint8), cv2.COLOR_RGB2GRAY)
            hog_features = hog(gray_image, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=False)
            features.append(hog_features)
            labels.append(np.argmax(label))
        if len(features) >= data_gen.samples:
            break
```

```
    return np.array(features), np.array(labels)

X_train, y_train = extract_features_and_labels(train_data_gen)

X_val, y_val = extract_features_and_labels(val_data_gen)

le = LabelEncoder()

y_train = le.fit_transform(y_train)

y_val = le.transform(y_val)

clf = svm.SVC(kernel='linear')

clf.fit(X_train, y_train)

y_pred = clf.predict(X_val)

print("Classification Report:")

print(classification_report(y_val, y_pred, target_names=class_names))

print("Accuracy Score:")

print(accuracy_score(y_val, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

       daisy       0.34      0.40      0.37       152
   dandelion       0.42      0.45      0.44       210
        rose       0.31      0.29      0.30       156
   sunflower       0.32      0.34      0.33       146
       tulip       0.42      0.34      0.38       196

    accuracy                           0.37       860
   macro avg       0.36      0.36      0.36       860
weighted avg       0.37      0.37      0.37       860

Accuracy Score:
0.3686046511627907
```

6. Write a python program to classify different flower images using CNN.

```
import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os

dataset_path = r'C:\Users\Rudradeep\Desktop\Python\pythonCollage\Collage_python\Lab 7\flowers'

IMG_SIZE = (128, 128)

BATCH_SIZE = 32

train_datagen = ImageDataGenerator(

    rescale=1./255,

    rotation_range=20,
```

```
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2
)
train_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training'
)
validation_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation'
)
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(train_generator.num_classes, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
model.summary()

history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)

val_loss, val_acc = model.evaluate(validation_generator)
print(f"Validation Accuracy: {val_acc * 100:.2f}%")
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_3 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_4 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_5 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_2 (Dense)	(None, 128)	3,211,392
dense_3 (Dense)	(None, 5)	645

Total params: 3,305,285 (12.61 MB)
Trainable params: 3,305,285 (12.61 MB)
Non-trainable params: 0 (0.00 B)

```
109/109 ━━━━━━━━━━━━━━━━━━━━ 162s 1s/step - accuracy: 0.3281 - loss: 1.5571 - val_accuracy: 0.5384 - val_loss: 1.1444
Epoch 2/10
109/109 ━━━━━━━━━━━━━━━━━━━━ 42s 376ms/step - accuracy: 0.5521 - loss: 1.1087 - val_accuracy: 0.5930 - val_loss: 1.0031
Epoch 3/10
109/109 ━━━━━━━━━━━━━━━━━━━━ 44s 391ms/step - accuracy: 0.5972 - loss: 0.9858 - val_accuracy: 0.6291 - val_loss: 0.9427
Epoch 4/10
109/109 ━━━━━━━━━━━━━━━━━━━━ 43s 388ms/step - accuracy: 0.6118 - loss: 0.9346 - val_accuracy: 0.6326 - val_loss: 0.9773
Epoch 5/10
109/109 ━━━━━━━━━━━━━━━━━━━━ 44s 393ms/step - accuracy: 0.6698 - loss: 0.8480 - val_accuracy: 0.6291 - val_loss: 0.9171
Epoch 6/10
109/109 ━━━━━━━━━━━━━━━━━━━━ 45s 403ms/step - accuracy: 0.6522 - loss: 0.8871 - val_accuracy: 0.6500 - val_loss: 0.9528
Epoch 7/10
109/109 ━━━━━━━━━━━━━━━━━━━━ 44s 394ms/step - accuracy: 0.6679 - loss: 0.8862 - val_accuracy: 0.6860 - val_loss: 0.8209
Epoch 8/10
109/109 ━━━━━━━━━━━━━━━━━━━━ 44s 393ms/step - accuracy: 0.6885 - loss: 0.7936 - val_accuracy: 0.6570 - val_loss: 0.9096
Epoch 9/10
109/109 ━━━━━━━━━━━━━━━━━━━━ 45s 409ms/step - accuracy: 0.6962 - loss: 0.7959 - val_accuracy: 0.6826 - val_loss: 0.7842
Epoch 10/10
109/109 ━━━━━━━━━━━━━━━━━━━━ 44s 399ms/step - accuracy: 0.7088 - loss: 0.7572 - val_accuracy: 0.6837 - val_loss: 0.8248
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 205ms/step - accuracy: 0.6887 - loss: 0.7750
Validation Accuracy: 68.02%
```

7. Write a python program to classify different handwritten character images using the SVM classifier.

```
import os
import cv2
import numpy as np
from sklearn import svm
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```python
dataset_path = r'C:\Users\Rudradeep\Desktop\Python\pythonCollage\Collage_python\Lab 7\flowers'
IMG_SIZE = (128, 128)


def load_images_from_folder(folder):
    images = []
    labels = []
    label_names = os.listdir(folder)
    for label_index, label_name in enumerate(label_names):
        label_folder = os.path.join(folder, label_name)
        if os.path.isdir(label_folder):
            for filename in os.listdir(label_folder):
                img_path = os.path.join(label_folder, filename)
                img = cv2.imread(img_path)
                if img is not None:
                    img = cv2.resize(img, IMG_SIZE)
                    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                    images.append(img.flatten())
                    labels.append(label_index)
    return np.array(images), np.array(labels), label_names
print("Loading dataset...")
x, y, label_names = load_images_from_folder(dataset_path)
x = x / 255.0
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
clf = svm.SVC(kernel='linear')
print("Training the SVM model...")
clf.fit(x_train, y_train)
print("Testing the model...")
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=label_names))
```

```
Training the SVM model...
Testing the model...
Test Accuracy: 28.70%

Classification Report:
              precision    recall  f1-score   support

       daisy       0.28      0.29      0.29       162
   dandelion       0.34      0.44      0.38       223
        rose       0.21      0.22      0.21       155
   sunflower       0.27      0.19      0.22       135
       tulip       0.29      0.22      0.25       189

    accuracy                           0.29       864
   macro avg       0.28      0.27      0.27       864
weighted avg       0.28      0.29      0.28       864
```

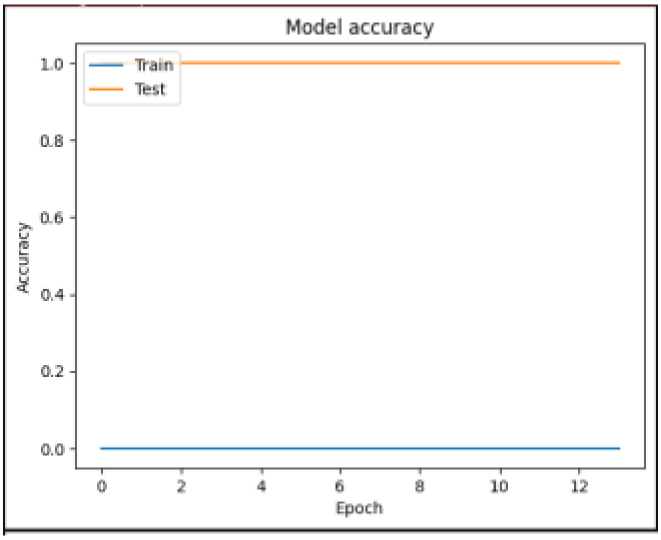
10. Write a python program to classify breast cancer from histopathological images using VGG-16 and DenseNet-201 CNN architectures

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.layers import Input


csv_file = r'C:\Users\Rudradeep\Desktop\Python\pythonCollage\Collage_python\Lab 7\breast-cancer.csv'

df = pd.read_csv(csv_file)

print(df.info())

categorical_cols = df.select_dtypes(include=['object']).columns

for col in categorical_cols:

    if df[col].nunique() == 2:

        le = LabelEncoder()

        df[col] = le.fit_transform(df[col])

    else:

        df = pd.get_dummies(df, columns=[col])
```

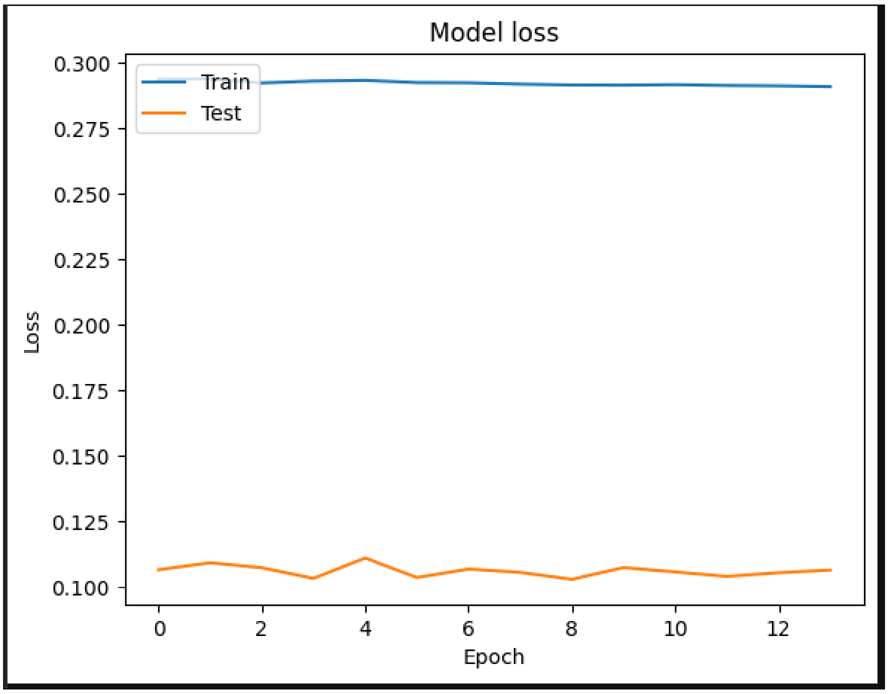
```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
def build_model(input_shape):
    model = Sequential([
        Input(shape=(input_shape,)),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    return model

input_shape = X_train.shape[1]
model = build_model(input_shape)
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=50,
    batch_size=32,
    callbacks=[early_stopping]
)
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)
y_test = y_test.astype(int)
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f'Test Accuracy: {accuracy * 100:.2f}%')

conf_matrix = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")

print(conf_matrix)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()
```

```
Epoch 1/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0000e+00 - loss: 0.2930 - val_accuracy: 1.0000 - val_loss: 0.1064
Epoch 2/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.0000e+00 - loss: 0.2949 - val_accuracy: 1.0000 - val_loss: 0.1091
Epoch 3/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.0000e+00 - loss: 0.2929 - val_accuracy: 1.0000 - val_loss: 0.1072
Epoch 4/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.0000e+00 - loss: 0.2921 - val_accuracy: 1.0000 - val_loss: 0.1031
Epoch 5/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.0000e+00 - loss: 0.2927 - val_accuracy: 1.0000 - val_loss: 0.1109
Epoch 6/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.0000e+00 - loss: 0.2926 - val_accuracy: 1.0000 - val_loss: 0.1035
Epoch 7/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.0000e+00 - loss: 0.2924 - val_accuracy: 1.0000 - val_loss: 0.1067
Epoch 8/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.0000e+00 - loss: 0.2915 - val_accuracy: 1.0000 - val_loss: 0.1054
Epoch 9/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.0000e+00 - loss: 0.2891 - val_accuracy: 1.0000 - val_loss: 0.1028
Epoch 10/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.0000e+00 - loss: 0.2901 - val_accuracy: 1.0000 - val_loss: 0.1073
Epoch 11/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.0000e+00 - loss: 0.2916 - val_accuracy: 1.0000 - val_loss: 0.1056
Epoch 12/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.0000e+00 - loss: 0.2902 - val_accuracy: 1.0000 - val_loss: 0.1039
Epoch 13/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.0000e+00 - loss: 0.2918 - val_accuracy: 1.0000 - val_loss: 0.1053
Epoch 14/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.0000e+00 - loss: 0.2892 - val_accuracy: 1.0000 - val_loss: 0.1063
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```



Model accuracy
Train
Test
1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
0
2
4
6
8
10
12
Epoch



Model loss
Train
Test
0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.100
Loss
0
2
4
6
8
10
12
Epoch