INDEX SHEET
PARTICULARS OF EXPERIMENTS PERFORMED

Serial No.	Name of the Experiment	Page No.	Date of Expt.	Date of Submission	Remarks
1.	Installation and working on Python, Jupyter and	1-7	07/12/23	28/12/23	
	its different libraries				
	for deep learning				
	(TensorFlow, Numpy,				
	Keras, Pandas, e.t.c.)				
2.	To implement a MLP	8-13	28/12/23	04/01/24	
	using keras with				
	Tensorflow, and finetune				
	neural network				
	hyperparameters for				
	regression problem				
	(houseprice prediction).				
3.	To implement a MLP	14-19	04/01/24	11/01/24	
	using keras with				
	Tensorflow for				
	classification problem				
	(Heart Disease				
	Prediction).				

1.

Experiment-No: ① :

Aim:

Installation and working on python, Jupyter, and its different libraries for deeplearning (TensorFlow, Numpy, Keras, Pandas, Matplotlib, etc).

Description:

① **TensorFlow:**

*. TensorFlow is an open-source machine learning framework developed by the Google Brain team.

*. It is used for Graph computation, Flexibility, High-level API's, TensorBoard.

② **Numpy:**

*. Numpy is a fundamental package for scientific computing with python.

*. It is used in Broadcasting, Lineal Algebra & Multi-Dimensional arrays.

③ **Pandas:**

*. Pandas is an open-source data manipulation and analysis library for python.

④ **matplotlib:**

*. matplotlib is a 2D plotting library that produces high quality charts & used for data visualization.

Program:

```
#Tensorflow
import tensorflow as tf
#Create a simple neural network
model = tf.keras.Sequential([tf.keras.layers.Dense(units=64,
                activation='relu', input_shape=(10,)),
                tf.keras.layers.Dense(units=1, activation='sigmoid')])
#Display the model summary
model.summary()
```

OUTPUT:

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	704
dense_1 (Dense)	(None, 1)	65

Total params: 769 (3.00 kB)

Trainable params: 769 (3.00 kB)

Non-trainable params: 0 (0.00 kB)

```
#NumPy
import numpy as np
#Create a NumPy array
array = np.array([[1,2,3],[4,5,6]])
#Perform basic operations
mean_value = np.mean(array)
sum_value = np.sum(array)
print("Array:")
print(array)
print("Mean:", mean_value)
print("Sum :", sum_value)
```

OUTPUT:

```
Array:
[[1 2 3]
 [4 5 6]]
Mean: 3.5
Sum : 21
```

```
#Pandas
import pandas as pd
data = {'Name': ['Ram', 'Sita'], 'Age': [21, 19]}
df = pd.DataFrame(data)
print(df)
```

OUTPUT:

	Name	Age
0	Ram	21
1	Sita	19

```
# Working with keras
import keras
from keras.models import Sequential
from keras.layers import Dense
# Create a simple neural network using keras
model = Sequential([Dense(units=64, activation='relu',
                    input_shape=(10,)),
                    Dense(units=1, activation='sigmoid')])
# Display the model summary
model.summary()
```

OUTPUT:

Model: "sequential_1"

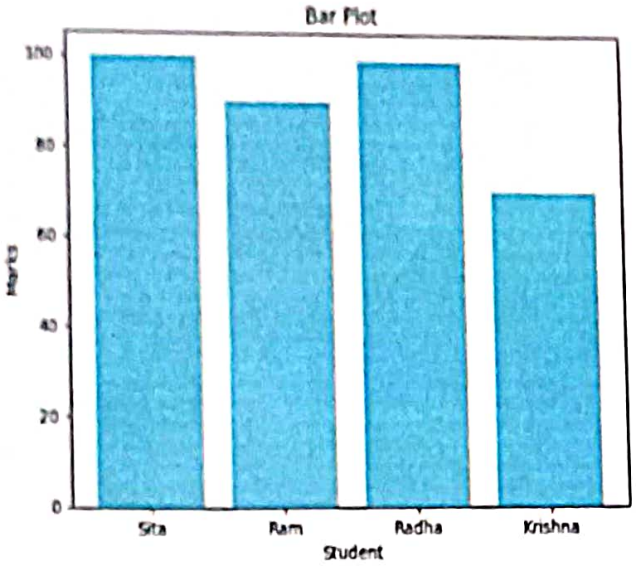
Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 64)	704
dense_3 (Dense)	(None, 1)	65

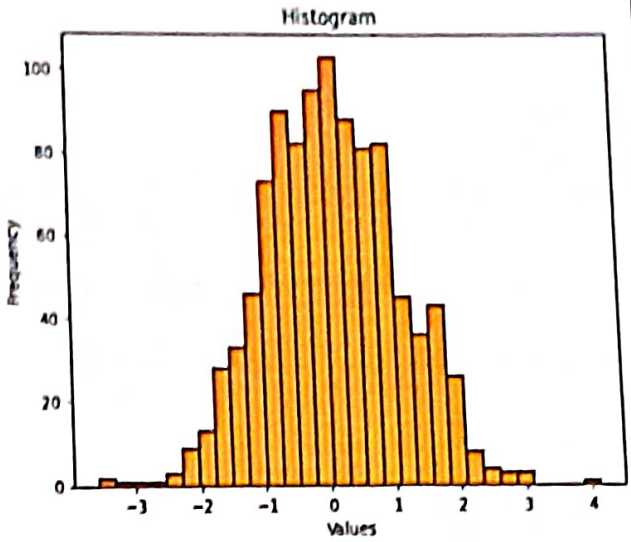
Total params: 769 (3.00 kB)
Trainable params: 769 (3.00 kB)
Non-Trainable params: 0 (0.00 kB)

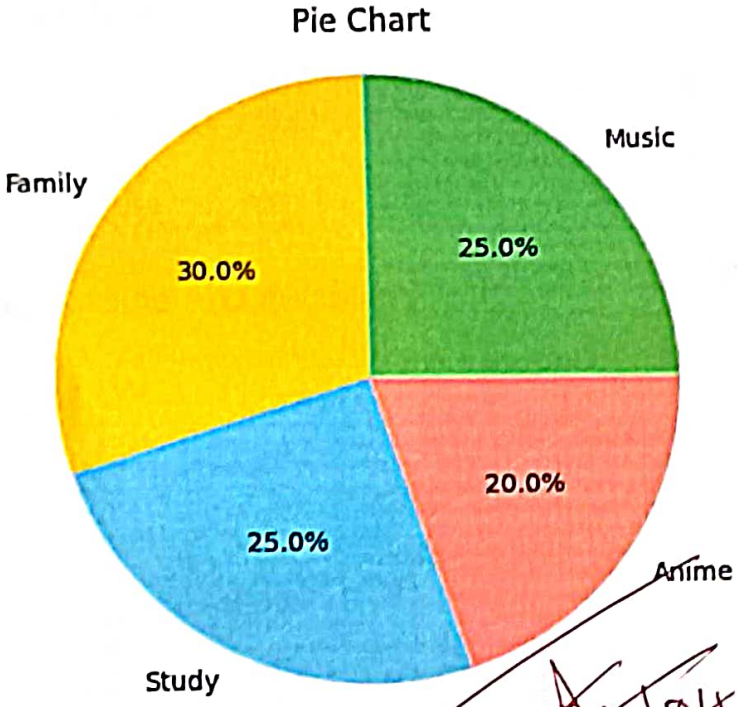
```
#Working with matplotlib
import matplotlib.pyplot as plt
import numpy as np
#Bar Plot
def bar_plot():
    names = ['Sita','Ram','Radha','Krishna']
    marks = [100,90,99,70]
    plt.bar(names, marks, color='skyblue')
    plt.xlabel('Students')
    plt.ylabel('Marks')
    plt.title('Bar Plot')
    plt.show()
#Histogram
def histogram():
    data=np.random.randn(1000) #Simple data for histogram
    plt.hist(data, bins=30, color='orange', edgecolor='black')
    plt.xlabel('Values')
    plt.ylabel('Freaquency')
    plt.show()
#Pie Chart
def pie_chart():
    labels = ['family','Study','Anime','Music']
    sizes = [30,25,20,25)]
    plt.pie(sizes, labels=labels, autopct='%1.1f%%',
            startangle=90, colors=['gold','lightskyblue',
            'lightcoral','lightgreen'])
```

```
    #Equal aspect ratio ensures that pie is
    #drawn as circle.
    plt.axis('equal')
    plt.title('Pie Chart')
    plt.show()

#Line Graph

def line_graph():
    x_values = np.linspace(0, 10, 100)
    y_values = np.sin(x_values)
    plt.plot(x_values, y_values, color='green', label='sin(x)')
    plt.xlabel('X-axis')
    plt.ylabel('Y_axis')
    plt.title('Line Graph')
    plt.legend()
    plt.show()

#Call the functions to display the plots.

bar_plot()
histogram()
pie_chart()
line_graph()
```

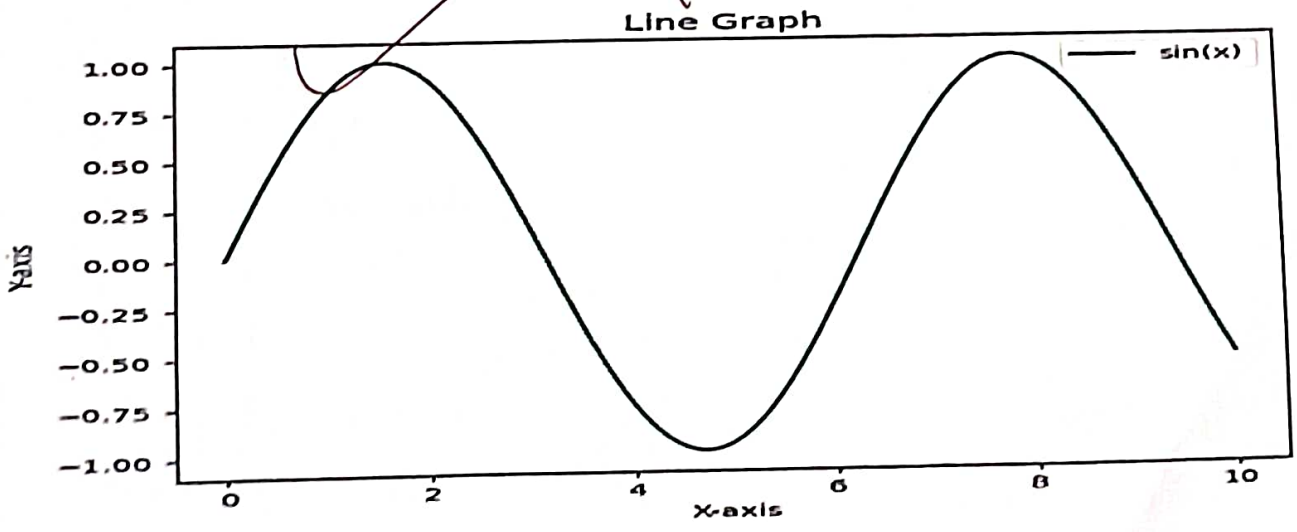
OUTPUT:







18/1/24



Experiment-NO:(2):

(2)

Aim:

TO implement a Multilayer Perceptron (MLP) using Keras with TensorFlow, and finetune neural network hyperparameters for regression problem (houseprice prediction).

Description:

Fine-tuning Neural Network Hyperparameters for House price prediction:

When we are working on a regression problem like predicting the house prices, the goal is to adjust Hyperparameters to improve the model's performance.

They are;

1. Data preparation
2. Feature Scaling
3. Model Architecture
4. Loss Function
5. Optimizer
6. Learning Rate
7. Batch Size
8. Epochs
9. Regularization
10. Early Stopping
11. Hyperparameter Search.

5) Program :

```
#Import all the required libraries.
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import columnTransformer
from tensorflow.keras.models import Seaquential
from tensorflow.keras.layers import Dense
from tensorflow.keras.Optimizers import Adam

#Loading the House price dataset including features.
data = pd.read_csv('Housing.csv')
data['date'] = pd.to_datetime(data['date'])
#Extract the features from date column
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month
data['day'] = data['date'].dt.day
#drop the original data column
data = data.drop(['date'], axis=1)
```

```
#Separate features and target variable
X = data.drop('price', axis=1)
y = data['price']
categorical_features = X.select_dtypes(include=['object']).columns
#Create a preprocessor using ColumnTransformer.
preprocessor = ColumnTransformer(transformer = [
                'num', StandardScaler(), X.select_dtypes
                (include=['number']).columns),
               ('cat', OneHotEncoder(), categorical_features)],
               remainder='passthrough')

#Transform the data
X_scaled = preprocessor.fit_transform(X)
#Split the data into training & testing sets.
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
                                   test_size = 0.2, random_state=42)

#Build the MLP model
model = Sequential()
#Add input layer and first hidden layer.
model.add(Dense(units=64, activation='relu', input_dim=
                                   X_scaled.shape[1]))

#Add additional hidden layers.
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=32, activation='relu'))
```

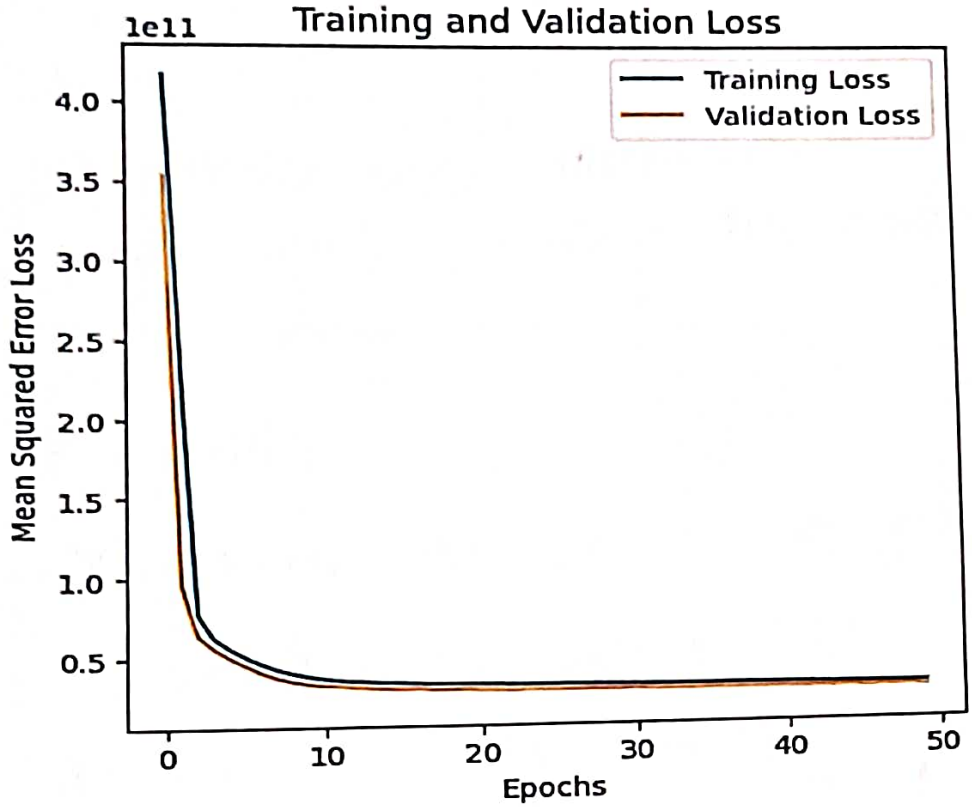
```
#Add Output layer (1 unit for regression, no activation function
model.add(Dense(units=1))
#Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='m_s_e')
#Train the model
history=model.fit(X_train, y_train, epochs=50, batch_size=32,
                  validation_split=0.2, verbose=0)
plt.figure(figsize=(5,5))
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('Mean Squared Error Loss')
plt.legend()
plt.show()
#Evaluate the model on the test set.
loss = model.evaluate(X_test, y_test)
print(f'Mean Squared Error: {loss}')
#Make Predictions
predictions = model.predict(X_test)
```

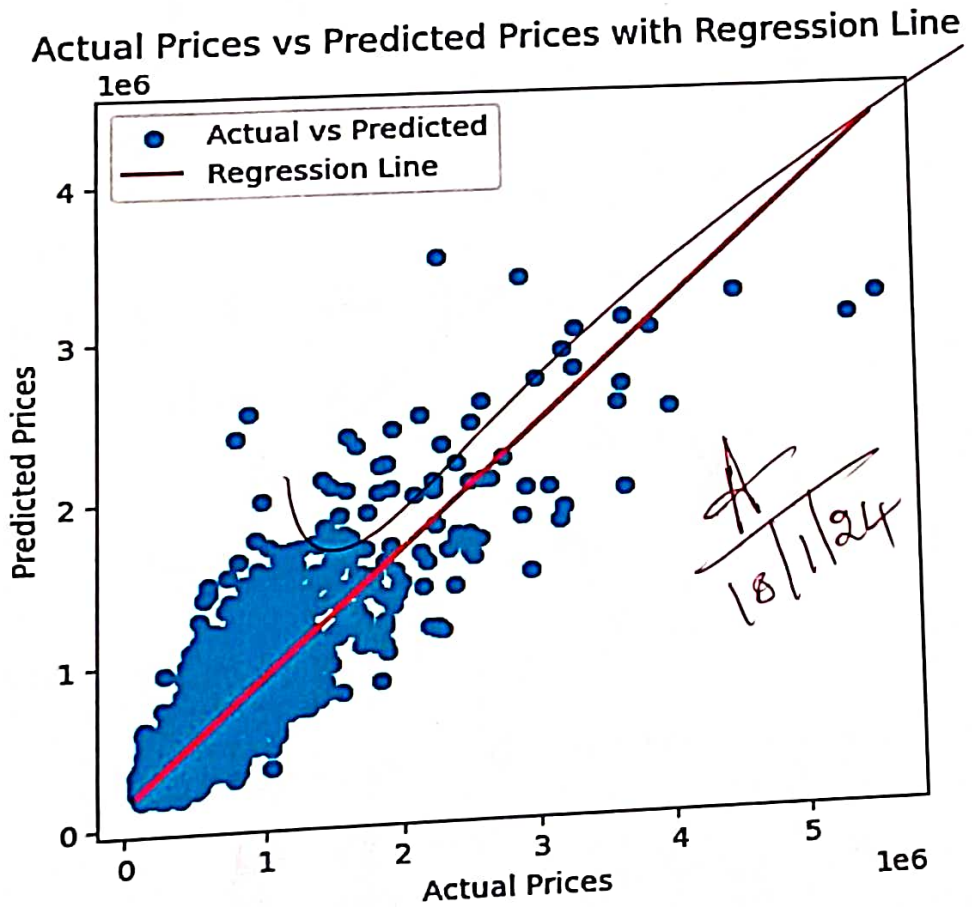
```
#Plot actual vis predicted values with regression_line.
plt.figure(figsize=(5,5))
plt.scatter(y_test, predictions, label='Actual vis Predicted')
plt.xlabel('Actual prices')
plt.ylabel('Predicted prices')
#Fit a linear Regression line.
regression_line = np.polyfit(y_test, predictions.flatten(), 1)
plt.plot(y_test, np.polyval(regression_line, y_test),
         color='red', label='Regression line')
plt.title('Actual prices vis Predicted prices with
          Regression line')
plt.legend()
PH.show()
```

OUTPUT:

```
1361136 [===] - 0s 2ms/step - loss: 33869586432.0000
Mean Squared Error on Test Set: 33869586432.0

1361136 [===] - 0s 1ms/step
```

OUTPUT:





18/1/24

3) Experiment-NO:(3):

Aim:-

To implement a Multilayer Perceptron (MLP) using keras with Tensorflow for classification problem. [Heart Disease Prediction].

Description:-

Steps involved to implement a MLP using keras with Tensorflow:

1. Import Necessary libraries.

i.e; import tensorflow as tf
from tensorflow import keras

2. Load and Preprocess the Data

3. Build the MLP model.

4. Compile the model using optimizer & loss function

5. Train the model.

where; Epchos: no.of training iterations.
Batch size : no.of samples per gradient update.

6. Evaluate the model.

Key features of Tensorflow:

a.) Graph Computation
b.) Flexibility
c.) Used in High-level API's
d.) TensorBoard.

(3) Program :

```
# import the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
import tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import confusion_matrix,
    accuracy_score, precision_score, recall_score, f1_score
import seaborn as sns

# Load the heart disease dataset
df = pd.read_csv('heart.csv')
X = df.drop('target', axis=1)
y = df['target']

# Split the the data into training & test sets.
X_train, y_train, X_test, y_test = train_test_split(X, y,
                test_size = 0.1, random_state = 42))

# Standardize the numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_train_scaled = scaler.transform(X_test)
```

```
# Convert labels to categorical format.
y_train_categorical = tf.keras.utils.to_categorical(y_train)
y_test_categorical = tf.keras.utils.to_categorical(y_test)

# Build the MLP model.
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=
                X_train_scaled.shape[1],))
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(128, activation='softmax')) # Binary classification

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

# Train the model.
history = model.fit(X_train_scaled, y_train_categorical,
                    epochs=50, batch_size=32, validation_split=0.1)

# Evaluate the model on the test set.
model.evaluate(X_test_scaled, y_test_categorical)
y_pred_probs = model.predict(X_test_scaled)
y_pred = np.argmax(y_pred_probs, axis=1)
y_test_int = np.argmax(y_test_categorical, axis=1)
```

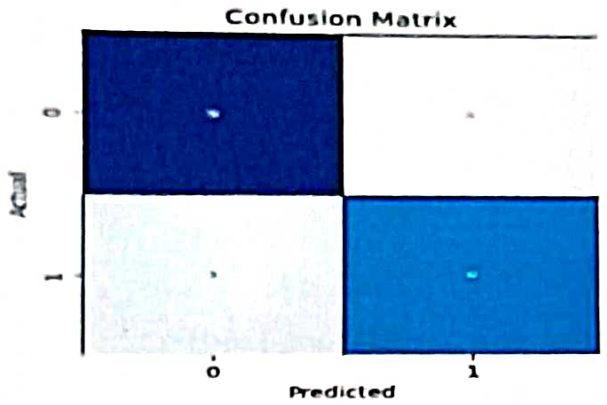
```
# Calculate Confusion matrix, accuracy, Presion, recall & f1_score.
cm = confusion_matrix (y_test_int, y_pred)
accuracy = accuracy_score (y_test_int, y_pred)
precision = precision_score (y_test_int, y_pred)
recall = recall_score (y_test_int, y_pred)
f1_score = f1_score (y_test, y_pred)

# Display the results
print ("Confusion Matrix: ")
print (cm)
print ("\n Accuracy : ", accuracy)
print (" Precision : ", precision)
print (" Recall : ", recall)
print (" F1 score : ", f1_scores)

# Plot the confusion matrix
plt.figure (figsize = (4, 4))
sns.heatmap (cm, annot = True, fmt = 'd', cmap = 'Blues', cbar = false,
             annot_kws = {'size' : 5}, linewidths = 0.5, linecolor = 'black')
plt.title ('Confusion Matrix')
plt.xlabel ('predicted')
plt.ylabel ('Actual')
plt.show ()
```

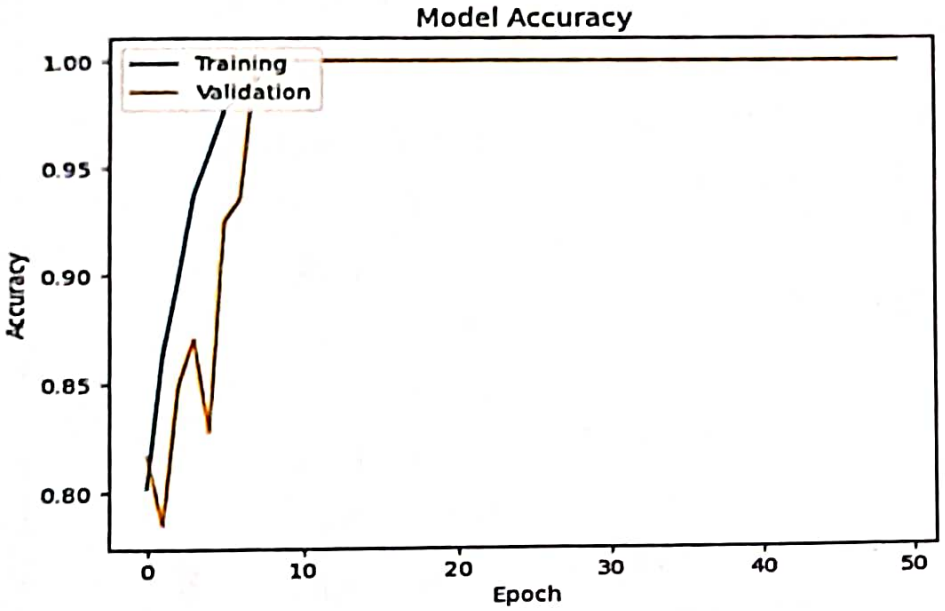
```
# Plot of training accuracy of & model loss.
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Training', 'Validation'], loc='upper left')
plt.show()

# Plot of Model loss.
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```

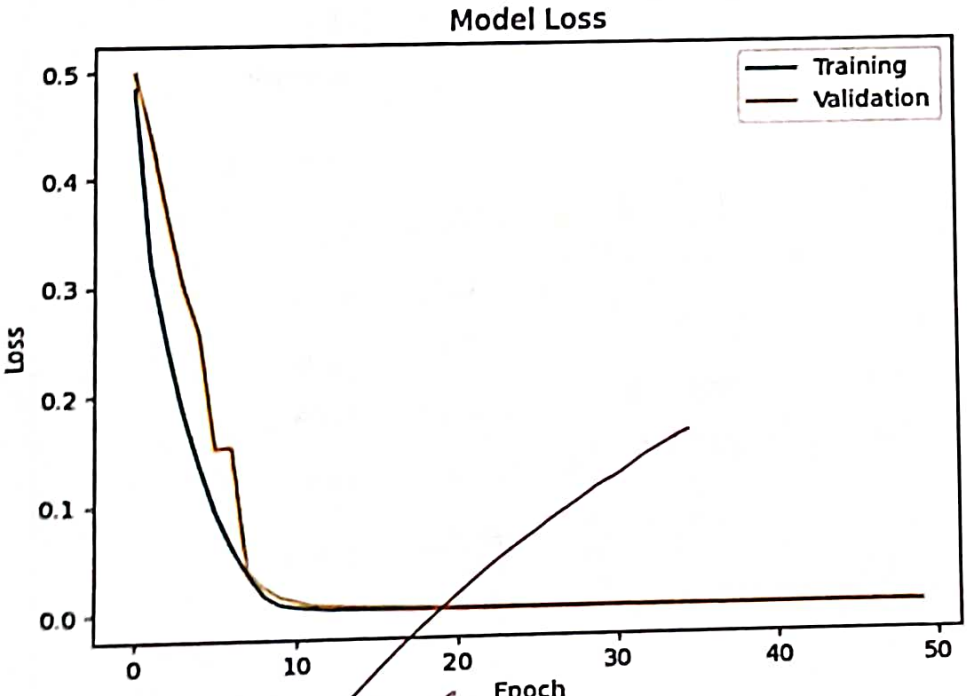
OUTPUT:

```
414 [===] - 0s 4ms/step - loss: 0.2793 - accuracy: 0.9709

Confusion Matrix:
 [[58  0
   3  42]]

Accuracy: 0.9708737864077767

Precision: 1.0

Recall   : 0.9333333333333333

F1_Score : 0.9655172413793101
```

OUTPUT:







18/1/24