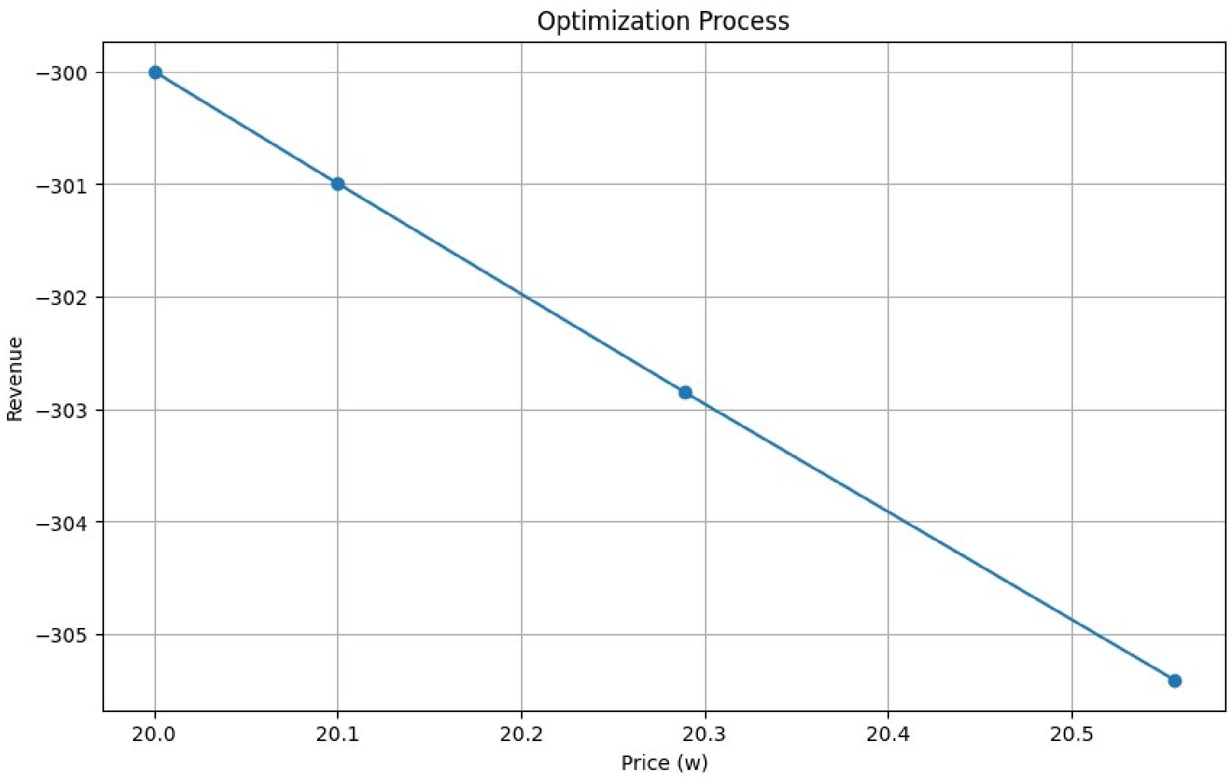
Question 01

```
import math
import matplotlib.pyplot as plt

# Objective function
def revenue(w):
    return 0.5 * w**2 - 30 * w + 100

# Gradient of the objective function
def gradient(w):
    return w - 30

# Hyperparameters
learning_rate = 0.1
momentum = 0.9
iterations = 3

# Initial price
w = 20

# Initialize velocity
v = 0

# Lists to store optimization history for visualization
w_history = [w]
revenue_history = [revenue(w)]

for i in range(iterations):
    v = momentum * v + (1 - momentum) * gradient(w)
    w -= learning_rate * v
    w_history.append(w)
    revenue_history.append(revenue(w))

print("Optimal price:", w)

# Visualization
plt.figure(figsize=(10, 6))
plt.plot(w_history, revenue_history, marker='o', linestyle='-')
plt.title('Optimization Process')
plt.xlabel('Price (w)')
plt.ylabel('Revenue')
plt.grid(True)
plt.show()
```

```
Optimal price: 20.55621
```



Question - 2

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("/content/diabetes.csv")
df.head().T
```

{"summary":"{\n \"name\": \"df\",\n \"rows\": 768,\n \"fields\": [\n   {\n   \"column\": \"Pregnancies\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3,\n   \"min\": 0,\n   \"max\": 17,\n   \"num_unique_values\": 17,\n   \"samples\": [\n   6,\n   1,\n   3\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Glucose\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 31,\n   \"min\": 0,\n   \"max\": 199,\n   \"num_unique_values\": 136,\n   \"samples\": [\n   151,\n   101,\n   112\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"BloodPressure\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 19,\n   \"min\": 0,\n   \"max\": 122,\n   \"num_unique_values\": 47,\n

\"samples\": [\n   86,\n   46,\n   85\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"SkinThickness\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 15,\n   \"min\": 0,\n
\"max\": 99,\n   \"num_unique_values\": 51,\n   \"samples\":
[\n   7,\n   12,\n   48\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Insulin\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": 115,\n
\"min\": 0,\n   \"max\": 846,\n   \"num_unique_values\":
186,\n   \"samples\": [\n   52,\n   41,\n
183\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"BMI\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 7.884160320375446,\n   \"min\": 0.0,\n   \"max\":
67.1,\n   \"num_unique_values\": 248,\n   \"samples\": [\n
19.9,\n   31.0,\n   38.1\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"DiabetesPedigreeFunction\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.3313285950127749,\n   \"min\": 0.078,\n   \"max\": 2.42,\n
\"num_unique_values\": 517,\n   \"samples\": [\n   1.731,\
n   0.426,\n   0.138\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Age\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 11,\n   \"min\": 21,\n
\"max\": 81,\n   \"num_unique_values\": 52,\n   \"samples\":
[\n   60,\n   47,\n   72\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Outcome\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": 0,\n
\"min\": 0,\n   \"max\": 1,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   0,\n   1\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"df"}

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
```

```
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
df.isna().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

No null values.

```
df.describe()
```

{"summary":"{\n \"name\": \"df\",\n \"rows\": 8,\n \"fields\": [\n   {\n   \"column\": \"Pregnancies\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 269.85223453356366,\n   \"min\": 0.0,\n   \"max\": 768.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   3.8450520833333335,\n   3.0,\n   768.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Glucose\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 243.73802348295857,\n   \"min\": 0.0,\n   \"max\": 768.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   120.89453125,\n   117.0,\n   768.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"BloodPressure\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 252.8525053581062,\n   \"min\": 0.0,\n   \"max\": 768.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   69.10546875,\n   72.0,\n   768.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"SkinThickness\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 263.7684730531098,\n   \"min\": 0.0,\n   \"max\": 768.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   768.0,\n   20.536458333333332,\n   32.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Insulin\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 350.26059167945886,\n   \"min\": 0.0,\n

\"max\": 846.0,\n   \"num_unique_values\": 7,\n
\"samples\": [\n   768.0,\n   79.79947916666667,\n
127.25\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"BMI\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 262.05117817552093,\n   \"min\": 0.0,\n   \"max\":
768.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n
31.992578124999998,\n   32.0,\n   768.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"DiabetesPedigreeFunction\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
271.3005221658502,\n   \"min\": 0.078,\n   \"max\": 768.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
0.47187630208333325,\n   0.3725,\n   768.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Age\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 260.1941178528413,\n
\"min\": 11.760231540678685,\n   \"max\": 768.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
33.240885416666664,\n   29.0,\n   768.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Outcome\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
271.3865920388932,\n   \"min\": 0.0,\n   \"max\": 768.0,\n
\"num_unique_values\": 5,\n   \"samples\": [\n
0.3489583333333333,\n   1.0,\n   0.47695137724279896\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}


```
df.shape
```

```
(768, 9)
```

```
# Numerical features

df.describe(exclude=['O'])
```


{"summary":"{\n \"name\": \"df\",\n \"rows\": 8,\n \"fields\": [\n
{\n   \"column\": \"Pregnancies\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 269.85223453356366,\n
\"min\": 0.0,\n   \"max\": 768.0,\n   \"num_unique_values\":
8,\n   \"samples\": [\n   3.8450520833333335,\n
3.0,\n   768.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Glucose\",\n   \"properties\": {\n   \"dtype\": \"number\",\
n   \"std\": 243.73802348295857,\n   \"min\": 0.0,\n
\"max\": 768.0,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   120.89453125,\n   117.0,\n
768.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":

```
\"BloodPressure\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 252.8525053581062,\n        \"min\":
0.0,\n        \"max\": 768.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          69.10546875,\n          72.0,\n
768.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"SkinThickness\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 263.7684730531098,\n        \"min\":
0.0,\n        \"max\": 768.0,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          768.0,\n          20.536458333333332,\n
32.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Insulin\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 350.26059167945886,\n        \"min\": 0.0,\n
\"max\": 846.0,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          768.0,\n          79.79947916666667,\n
127.25\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"BMI\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 262.05117817552093,\n        \"min\": 0.0,\n        \"max\":
768.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
31.992578124999998,\n          32.0,\n          768.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"DiabetesPedigreeFunction\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
271.3005221658502,\n        \"min\": 0.078,\n        \"max\": 768.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
0.47187630208333325,\n          0.3725,\n          768.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Age\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 260.1941178528413,\n
\"min\": 11.760231540678685,\n        \"max\": 768.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
33.240885416666664,\n          29.0,\n          768.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Outcome\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
271.3865920388932,\n        \"min\": 0.0,\n        \"max\": 768.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.3489583333333333,\n          1.0,\n          0.4769513772427989\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


No categorical columns, so no need for hot encoding

```
#lib for model building & eval
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score,
```

```
recall_score, f1_score, confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt

# split data into features and target
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Base Model

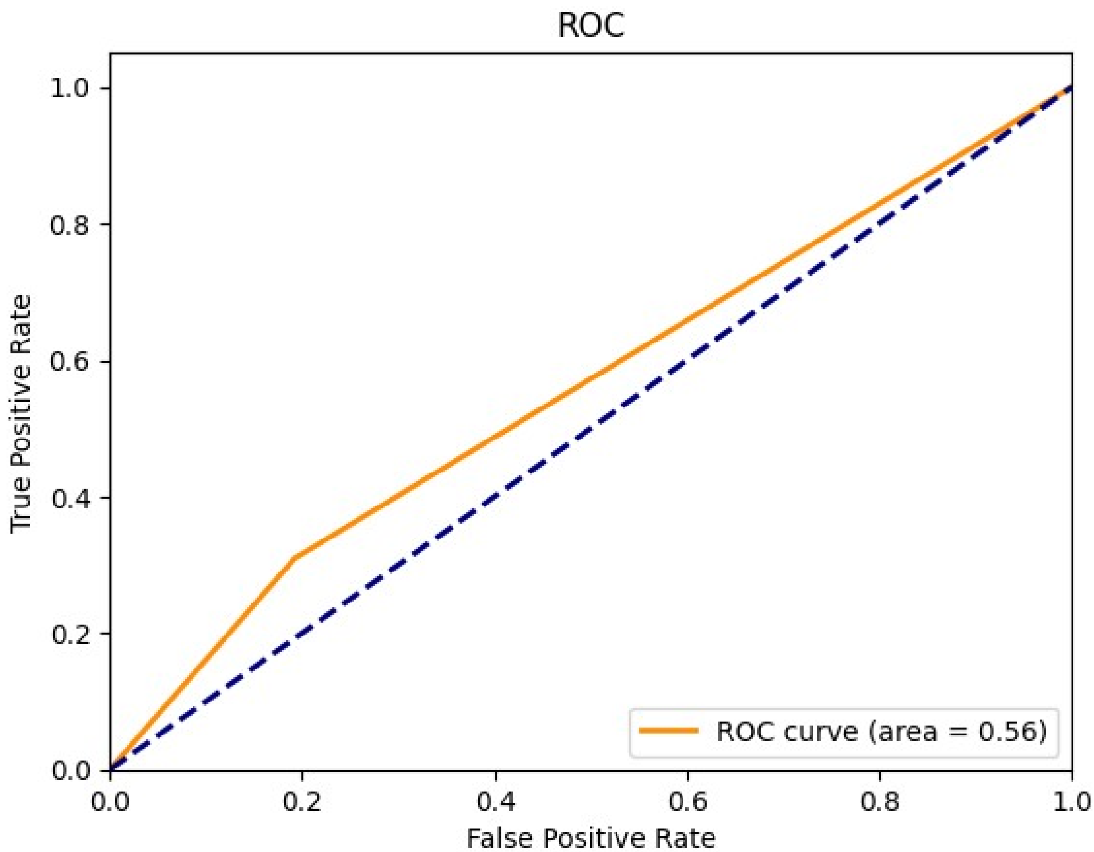
```
# neural network model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# model.complie
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# model train
hist = model.fit(X_train, y_train, epochs=50, batch_size=32,
validation_split=0.2, verbose=0)

# model eval
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", acc)
print("Precision:", prec)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
5/5 [==============================] - 0s 4ms/step
Accuracy: 0.6298701298701299
Precision: 0.4722222222222222
Recall: 0.3090909090909091
F1 Score: 0.37362637362637363
```

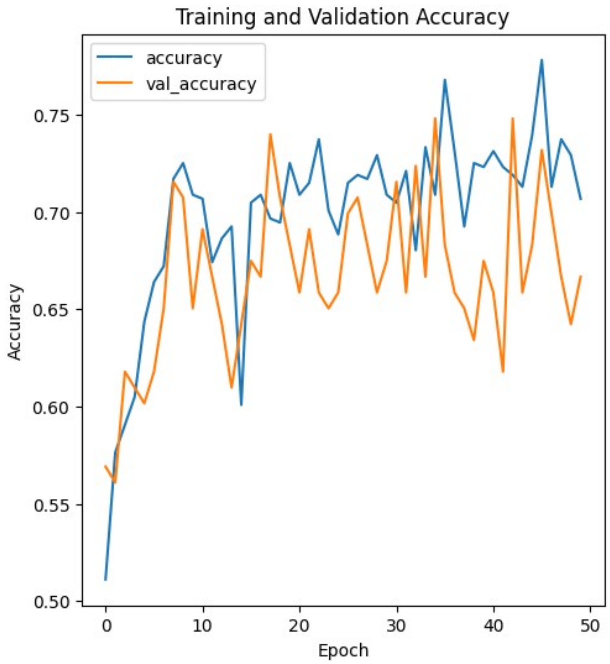
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

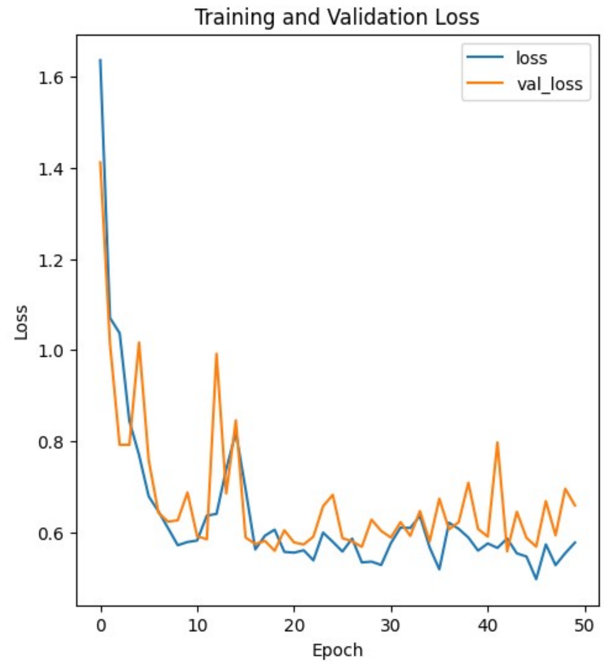
```
Confusion Matrix:
[[80 19]
 [38 17]]
```

```
# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(loc="lower right")
plt.show()
```




```
# Learning Curves
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(hist.history['accuracy'], label='accuracy')
plt.plot(hist.history['val_accuracy'], label='val_accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

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

plt.show()
```





User defined code for evaluation, Conf matrix, ROC curve, Learning curve for reusability.

```
# Evaluate the performance of models
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    y_pred = (y_pred > 0.5).astype(int)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    return accuracy, precision, recall, f1

# Confusion Matrix
def plot_confusion_matrix(model, X_test, y_test):
    y_pred = model.predict(X_test)
    y_pred = (y_pred > 0.5).astype(int)
    cm = confusion_matrix(y_test, y_pred)
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.colorbar()
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.xticks([0, 1], ['Negative', 'Positive'])
    plt.yticks([0, 1], ['Negative', 'Positive'])

    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
```

```
            plt.text(j, i, format(cm[i, j], 'd'),
horizontalalignment="center", color="white" if cm[i, j] > thresh else
"black")

    plt.show()

# ROC Curve
def plot_roc_curve(model, X_test, y_test):
    y_pred = model.predict(X_test)
    fpr, tpr, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve
(area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC')
    plt.legend(loc="lower right")
    plt.show()

# Learning Curves
def plot_learning_curves(history, title):
    plt.plot(history.history['accuracy'], label='accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.title(title)
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```

L1 Model

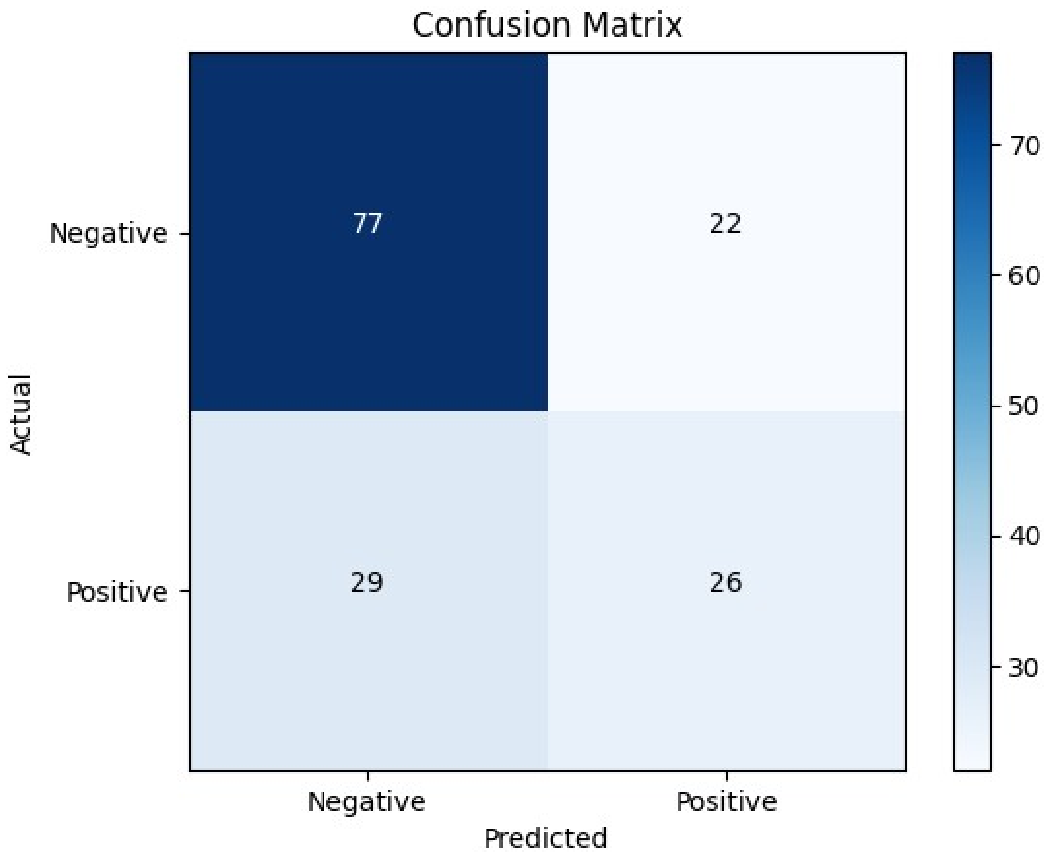
```
# L1 regularization
model_l1 = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu',
kernel_regularizer=tf.keras.regularizers.l1(0.01),
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu',
kernel_regularizer=tf.keras.regularizers.l1(0.01)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# model
model_l1.compile(optimizer='adam',
                 loss='binary_crossentropy',
                 metrics=['accuracy'])
```

```python
# Train
hist_l1 = model_l1.fit(X_train, y_train, epochs=50, batch_size=32,
validation_split=0.2, verbose=0)

#eval
accuracy_l1, precision_l1, recall_l1, f1_l1 = evaluate_model(model_l1,
X_test, y_test)
print("L1 Regularization:")
print("Accuracy:", accuracy_l1)
print("Precision:", precision_l1)
print("Recall:", recall_l1)
print("F1 Score:", f1_l1)
```

```
5/5 [==============================] - 0s 4ms/step
L1 Regularization:
Accuracy: 0.6688311688311688
Precision: 0.5416666666666666
Recall: 0.4727272727272727
F1 Score: 0.5048543689320388
```

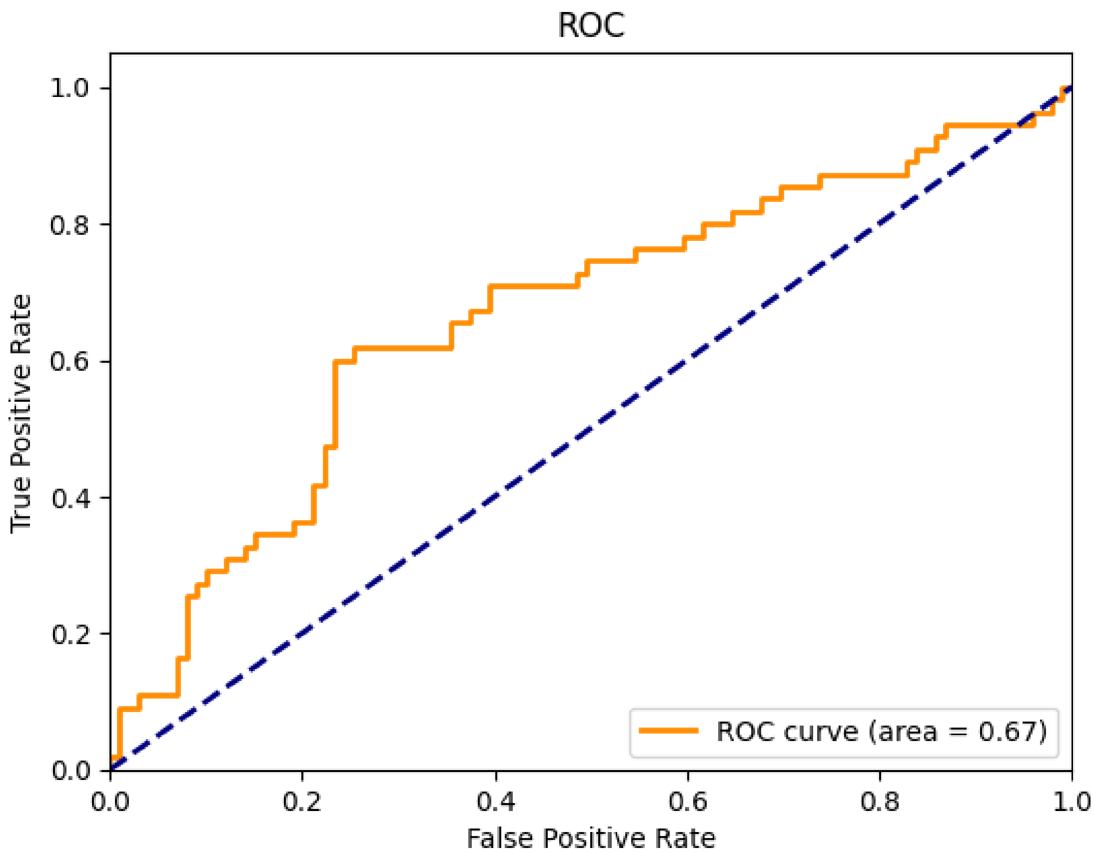
```python
print("\nConfusion Matrix - L1 Regularization:")
plot_confusion_matrix(model_l1, X_test, y_test)
```

```
Confusion Matrix - L1 Regularization:
5/5 [==============================] - 0s 2ms/step
```



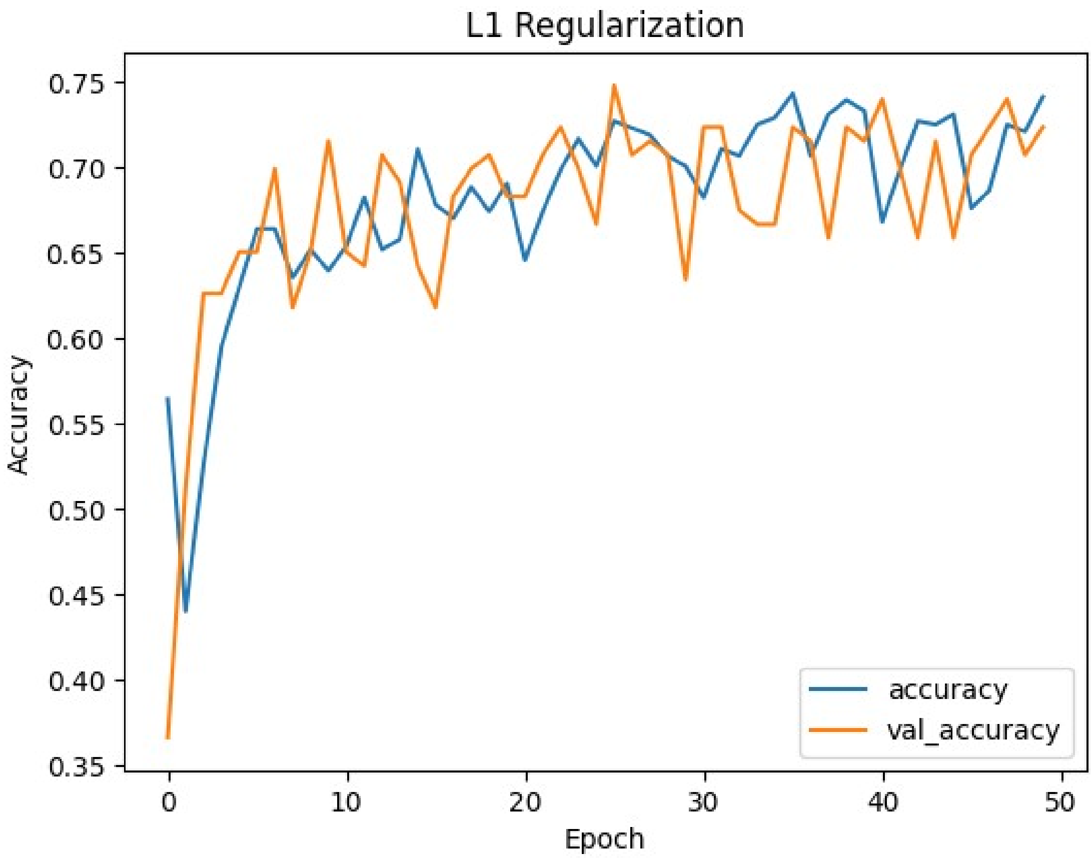

```
print("\nROC Curve - L1 Regularization:")
plot_roc_curve(model_l1, X_test, y_test)
```

```
ROC Curve - L1 Regularization:
5/5 [==============================] - 0s 3ms/step
```




```
print("\nLearning Curves - L1 Regularization:")
plot_learning_curves(hist_l1, "L1 Regularization")
```

```
Learning Curves - L1 Regularization:
```



L2

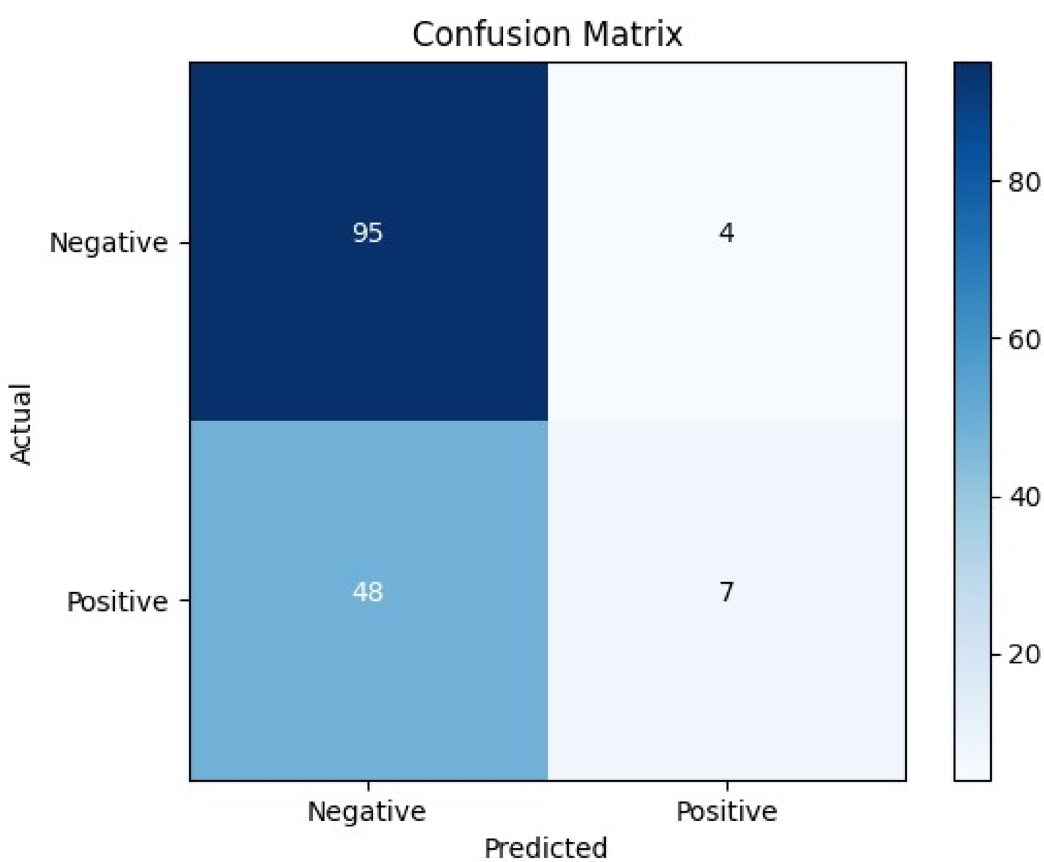
```
# L2 regularization
model_l2 = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.01),
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# model
model_l2.compile(optimizer='adam',
                 loss='binary_crossentropy',
                 metrics=['accuracy'])

# Train
hist_l2 = model_l2.fit(X_train, y_train, epochs=50, batch_size=32,
validation_split=0.2, verbose=0)

#eval
accuracy_l2, precision_l2, recall_l2, f1_l2 = evaluate_model(model_l2,
X_test, y_test)
```

```
print("\nL2 Regularization:")
print("Accuracy:", accuracy_l2)
print("Precision:", precision_l2)
print("Recall:", recall_l2)
print("F1 Score:", f1_l2)
```

```
5/5 [==============================] - 0s 3ms/step

L2 Regularization:
Accuracy: 0.6623376623376623
Precision: 0.6363636363636364
Recall: 0.12727272727272726
F1 Score: 0.2121212121212121
```

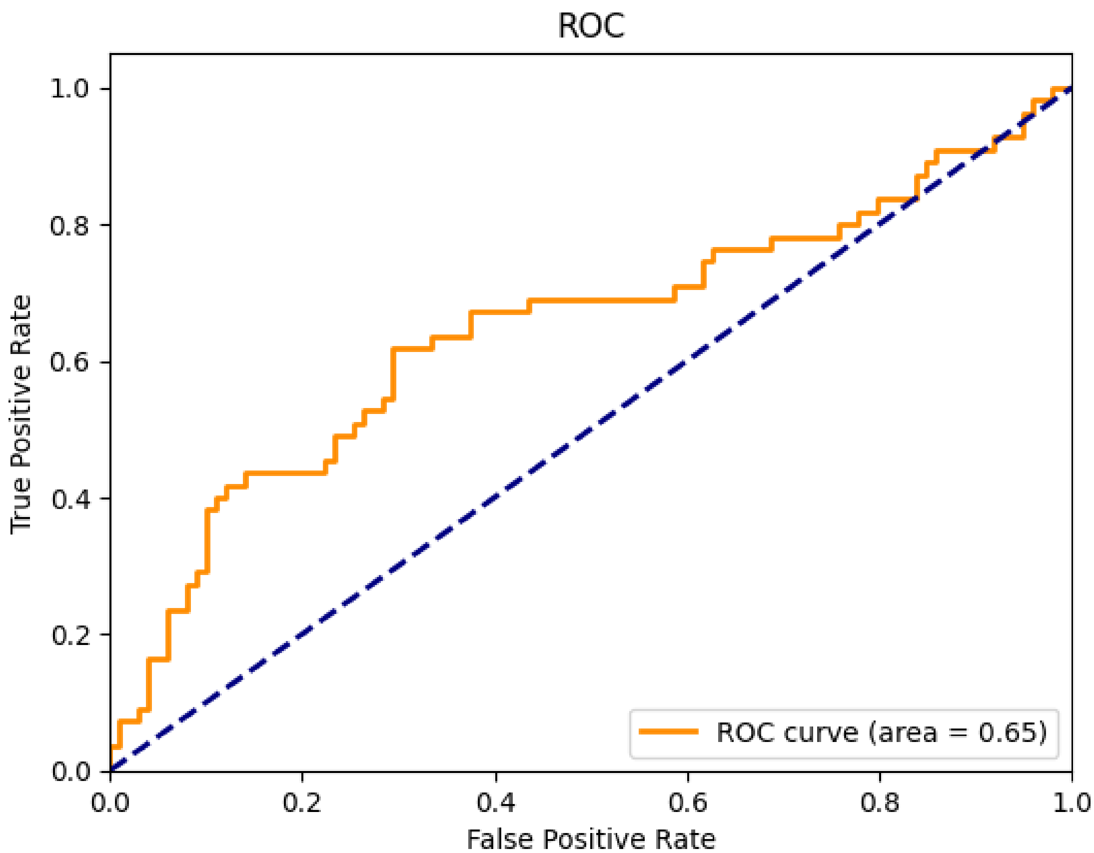
```
print("\nConfusion Matrix - L2 Regularization:")
plot_confusion_matrix(model_l2, X_test, y_test)
```

```
Confusion Matrix - L2 Regularization:
5/5 [==============================] - 0s 3ms/step
```



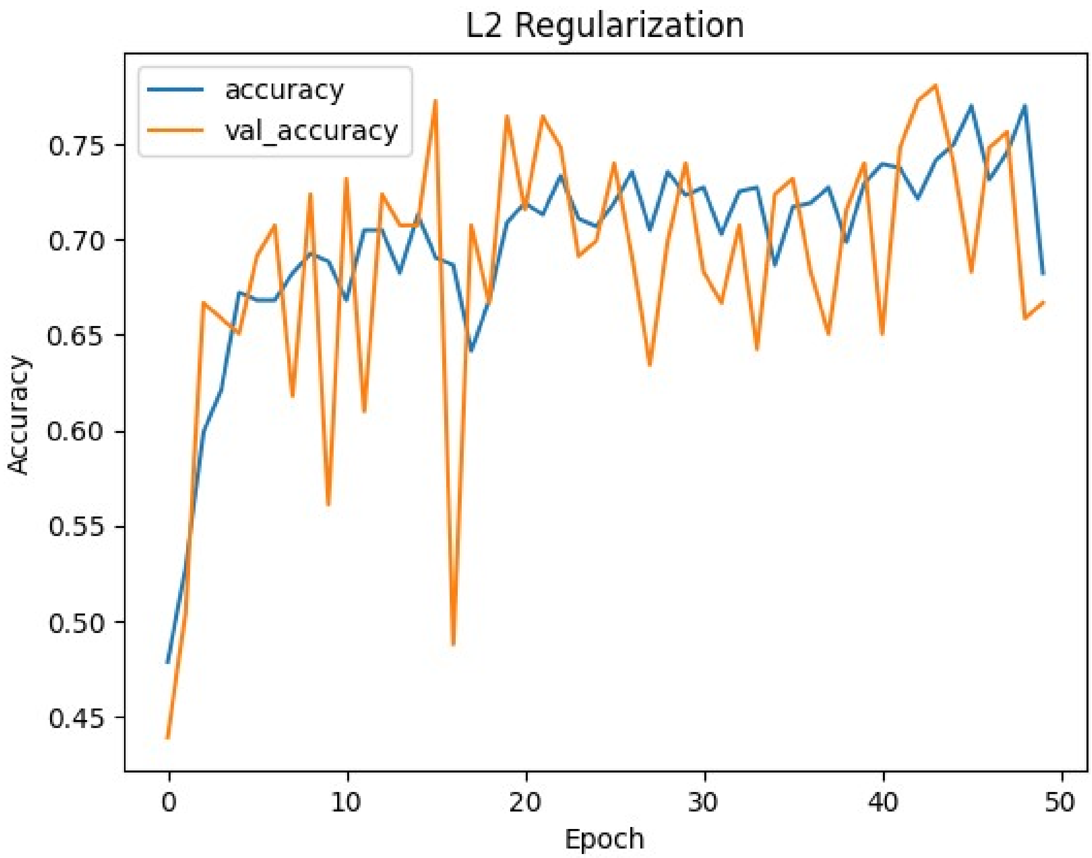

```
print("\nROC Curve - L2 Regularization:")
plot_roc_curve(model_l2, X_test, y_test)
```

```
ROC Curve - L2 Regularization:
5/5 [==============================] - 0s 6ms/step
```




```
print("\nLearning Curves - L2 Regularization:")
plot_learning_curves(hist_l2, "L2 Regularization")
```

```
Learning Curves - L2 Regularization:
```



Elastic net

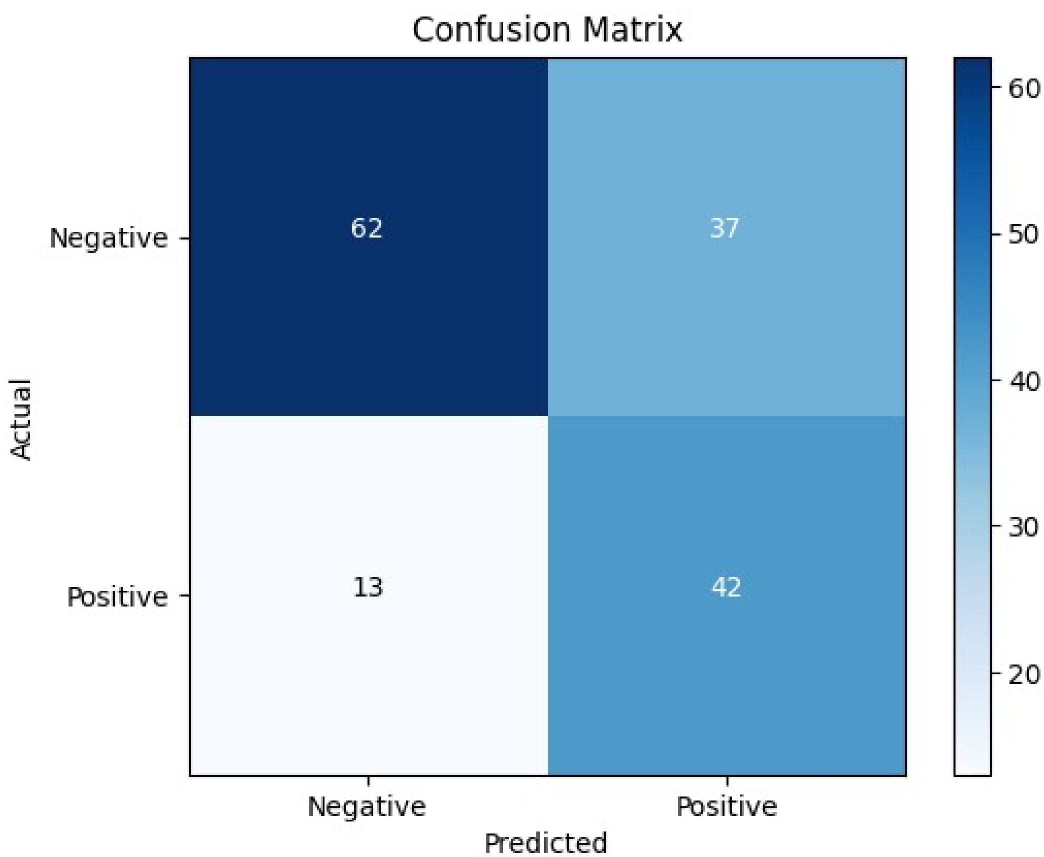
```
# Elastic Net regularization
model_elastic_net = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu',
kernel_regularizer=tf.keras.regularizers.l1_l2(l1=0.01, l2=0.01),
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu',
kernel_regularizer=tf.keras.regularizers.l1_l2(l1=0.01, l2=0.01)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# model
model_elastic_net.compile(optimizer='adam',
                           loss='binary_crossentropy',
                           metrics=['accuracy'])

# Train the model
hist_elastic_net = model_elastic_net.fit(X_train, y_train, epochs=50,
batch_size=32, validation_split=0.2, verbose=0)

#eval
accuracy_elastic_net, precision_elastic_net, recall_elastic_net,
```

```
f1_elastic_net = evaluate_model(model_elastic_net, X_test, y_test)
print("\nElastic Net Regularization:")
print("Accuracy:", accuracy_elastic_net)
print("Precision:", precision_elastic_net)
print("Recall:", recall_elastic_net)
print("F1 Score:", f1_elastic_net)
```

```
5/5 [==============================] - 0s 3ms/step

Elastic Net Regularization:
Accuracy: 0.6753246753246753
Precision: 0.5316455696202531
Recall: 0.7636363636363637
F1 Score: 0.6268656716417911
```

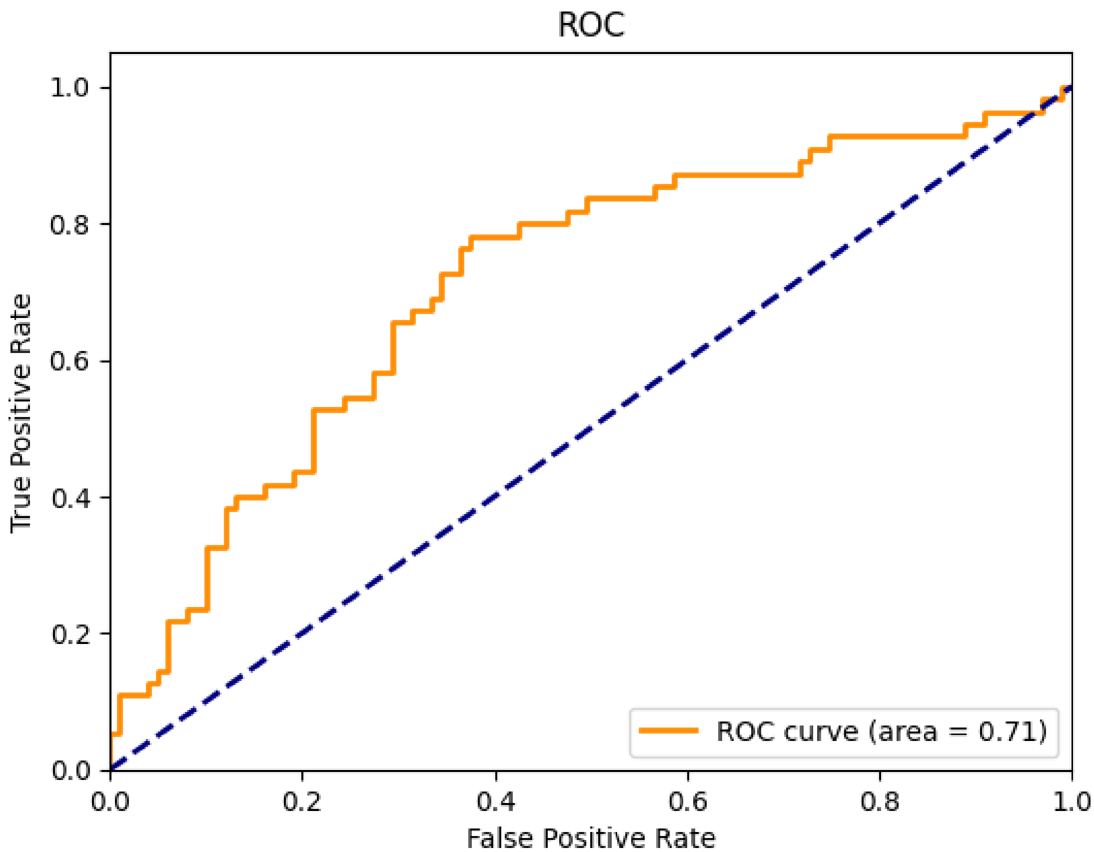
```
print("\nConfusion Matrix - Elastic Net Regularization:")
plot_confusion_matrix(model_elastic_net, X_test, y_test)
```

```
Confusion Matrix - Elastic Net Regularization:
5/5 [==============================] - 0s 3ms/step
```



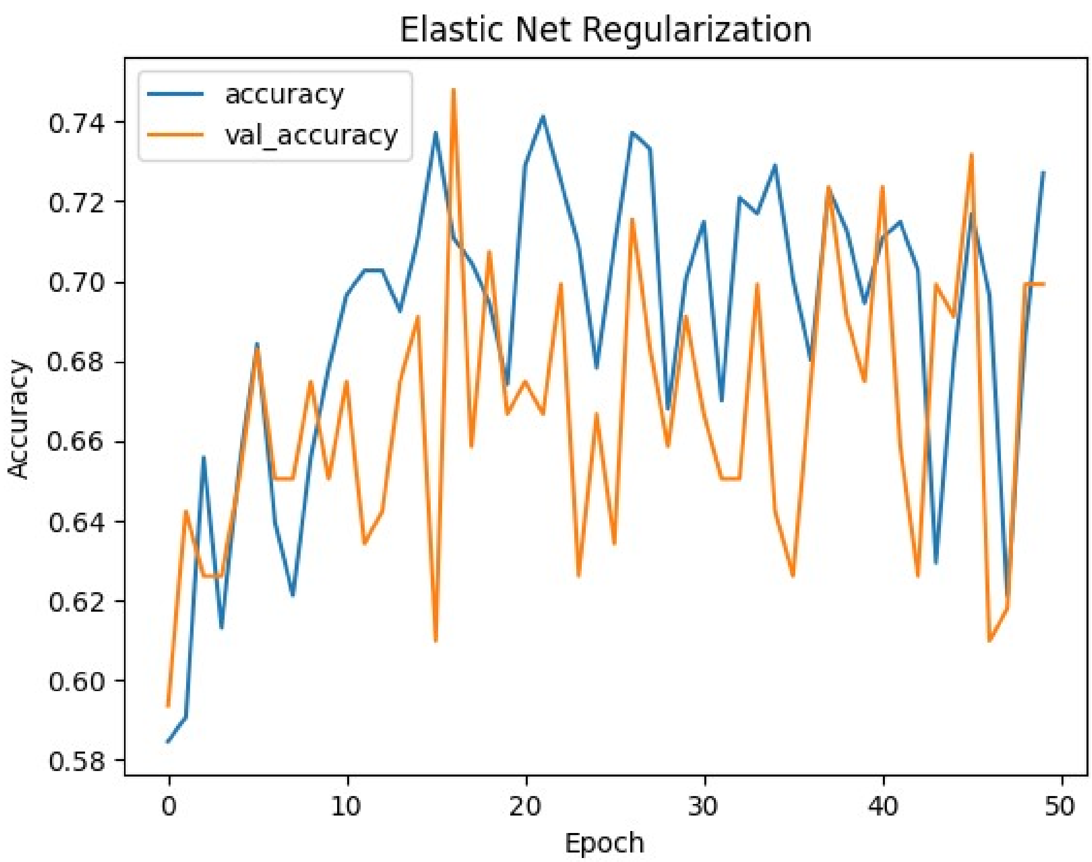
```
print("\nROC Curve - Elastic Net Regularization:")
plot_roc_curve(model_elastic_net, X_test, y_test)
```

```
ROC Curve - Elastic Net Regularization:
5/5 [==============================] - 0s 3ms/step
```




```
print("\nLearning Curves - Elastic Net Regularization:")
plot_learning_curves(hist_elastic_net, "Elastic Net Regularization")
```

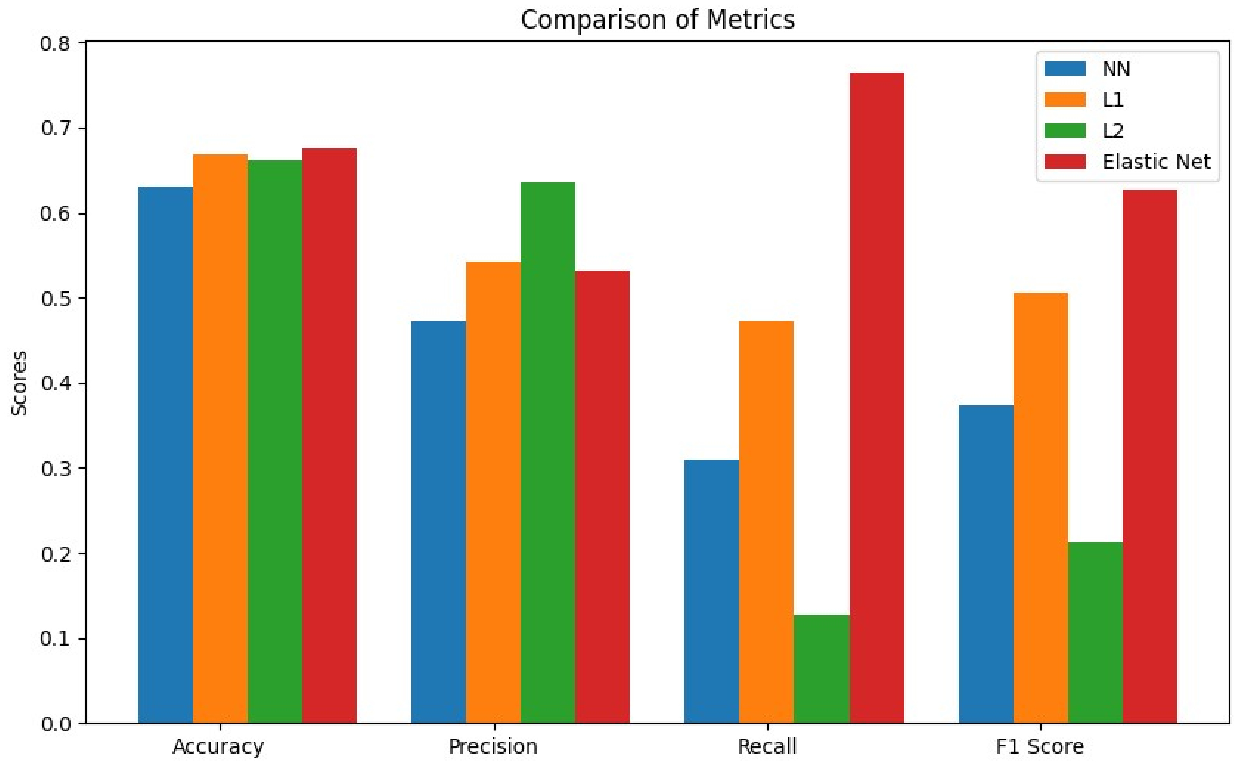
```
Learning Curves - Elastic Net Regularization:
```



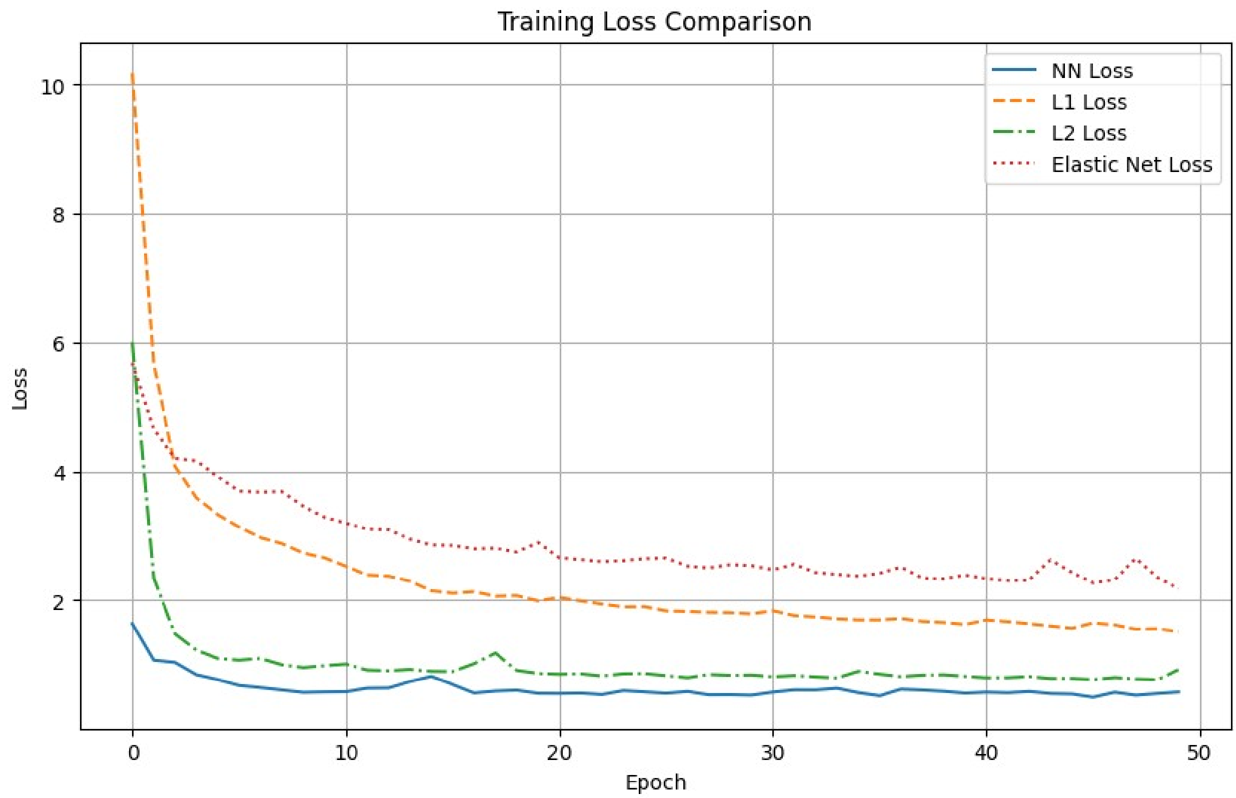
Comparison

```
# Comparison of metrics
labels = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
metrics_nn = [accuracy_score(y_test, y_pred), precision_score(y_test,
y_pred), recall_score(y_test, y_pred), f1_score(y_test, y_pred)]
metrics_l1 = [accuracy_l1, precision_l1, recall_l1, f1_l1]
metrics_l2 = [accuracy_l2, precision_l2, recall_l2, f1_l2]
metrics_elastic_net = [accuracy_elastic_net, precision_elastic_net,
recall_elastic_net, f1_elastic_net]

x = np.arange(len(labels))
width = 0.2

fig, ax = plt.subplots(figsize=(10, 6))

rects1 = ax.bar(x - width, metrics_nn, width, label='NN')
rects2 = ax.bar(x, metrics_l1, width, label='L1')
rects3 = ax.bar(x + width, metrics_l2, width, label='L2')
rects4 = ax.bar(x + 2*width, metrics_elastic_net, width,
label='Elastic Net')

ax.set_ylabel('Scores')
```

```python
ax.set_title('Comparison of Metrics')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

plt.show()
```




```python
# Training loss comparison
plt.figure(figsize=(10, 6))
plt.plot(hist.history['loss'], label='NN Loss', linestyle='-')
plt.plot(hist_l1.history['loss'], label='L1 Loss', linestyle='--')
plt.plot(hist_l2.history['loss'], label='L2 Loss', linestyle='-.')
plt.plot(hist_elastic_net.history['loss'], label='Elastic Net Loss',
linestyle=':')
plt.title('Training Loss Comparison')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```


Training Loss Comparison
NN Loss
L1 Loss
L2 Loss
Elastic Net Loss
Loss
Epoch
0
10
20
30
40
50
2
4
6
8
10