Imports

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc, precision_recall_curve, avera
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
```

Question - 2

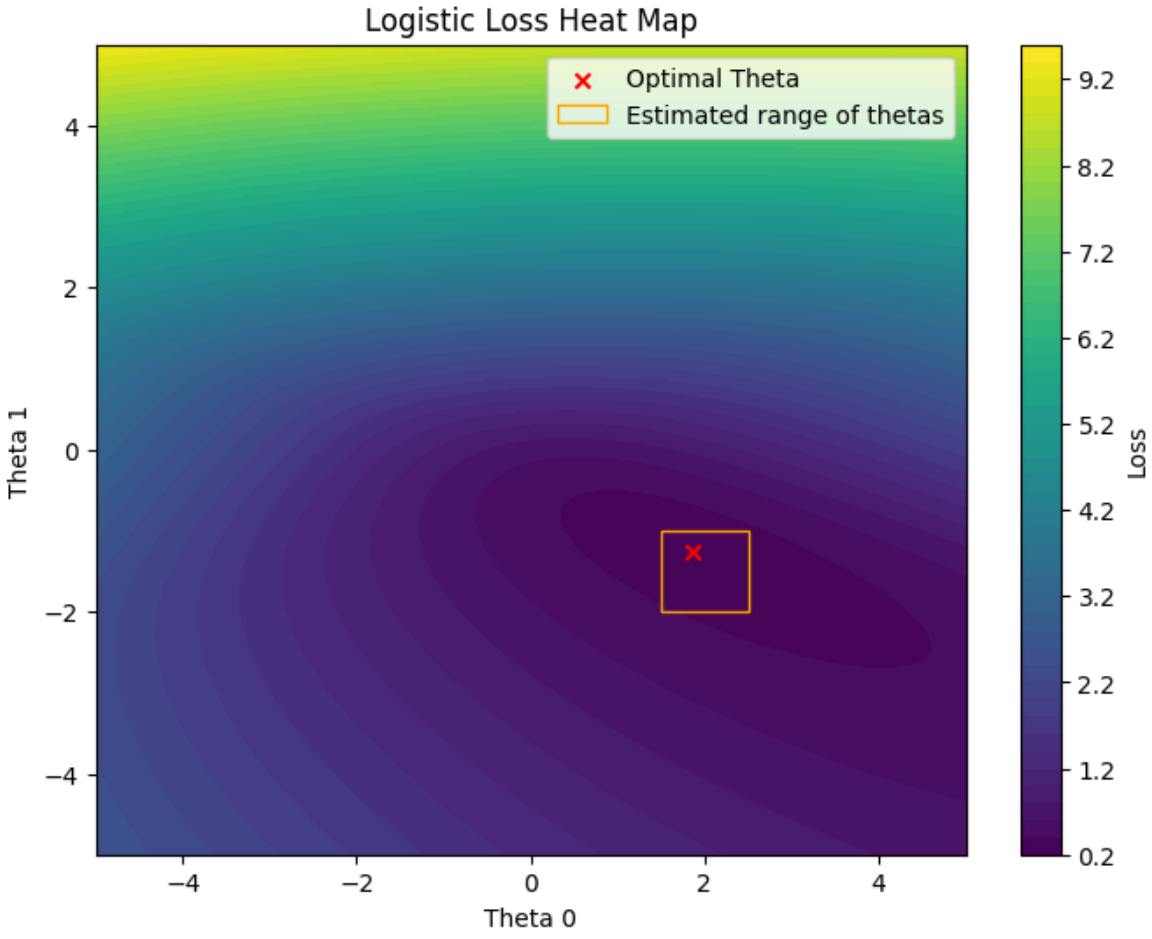
A) 1 D Logistic regression on dummy data to visually inspect theta values

- Calculate log loss for each value of theta in the assumed linspace.
- Plot the heatmap for visual estimation.
- Find the actual optimal theta values for verification.

In [ ]:
```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def log_loss(theta, x, y):
    z = theta[0] + theta[1] * x
    y_hat = sigmoid(z)
    loss = -np.mean(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
    return loss

data = np.array([[-3, 1],
                 [-2, 1],
                 [-1, 1],
                 [0, 1],
                 [1, 0],
                 [2, 1],
                 [3, 0],
                 [4, 0]])

theta0_values = np.linspace(-5, 5, 100)
theta1_values = np.linspace(-5, 5, 100)
loss_values = np.zeros((len(theta0_values), len(theta1_values)))

for i, theta0 in enumerate(theta0_values):
    for j, theta1 in enumerate(theta1_values):
        loss_values[i, j] = log_loss([theta0, theta1], data[:, 0], data[:

min_index = np.unravel_index(np.argmin(loss_values, axis=None), loss_valu
```

```
estimated_theta0 = theta0_values[min_index[0]]
estimated_theta1 = theta1_values[min_index[1]]

print("Estimated optimal values:")
print("Theta 0:", estimated_theta0)
print("Theta 1:", estimated_theta1)

plt.figure(figsize=(8, 6))
plt.contourf(theta0_values, theta1_values, loss_values.T, levels=50, cmap
plt.colorbar(label='Loss')
plt.scatter(estimated_theta0, estimated_theta1, marker='x', color='red',
plt.xlabel('Theta 0')
plt.ylabel('Theta 1')
plt.title('Logistic Loss Heat Map')

rect = patches.Rectangle((1.5, -1), 1, -1, linewidth=1, edgecolor='orange
plt.gca().add_patch(rect)

plt.legend()
plt.show()
```

```
Estimated optimal values:
Theta 0: 1.8686868686868685
Theta 1: -1.2626262626262625
```



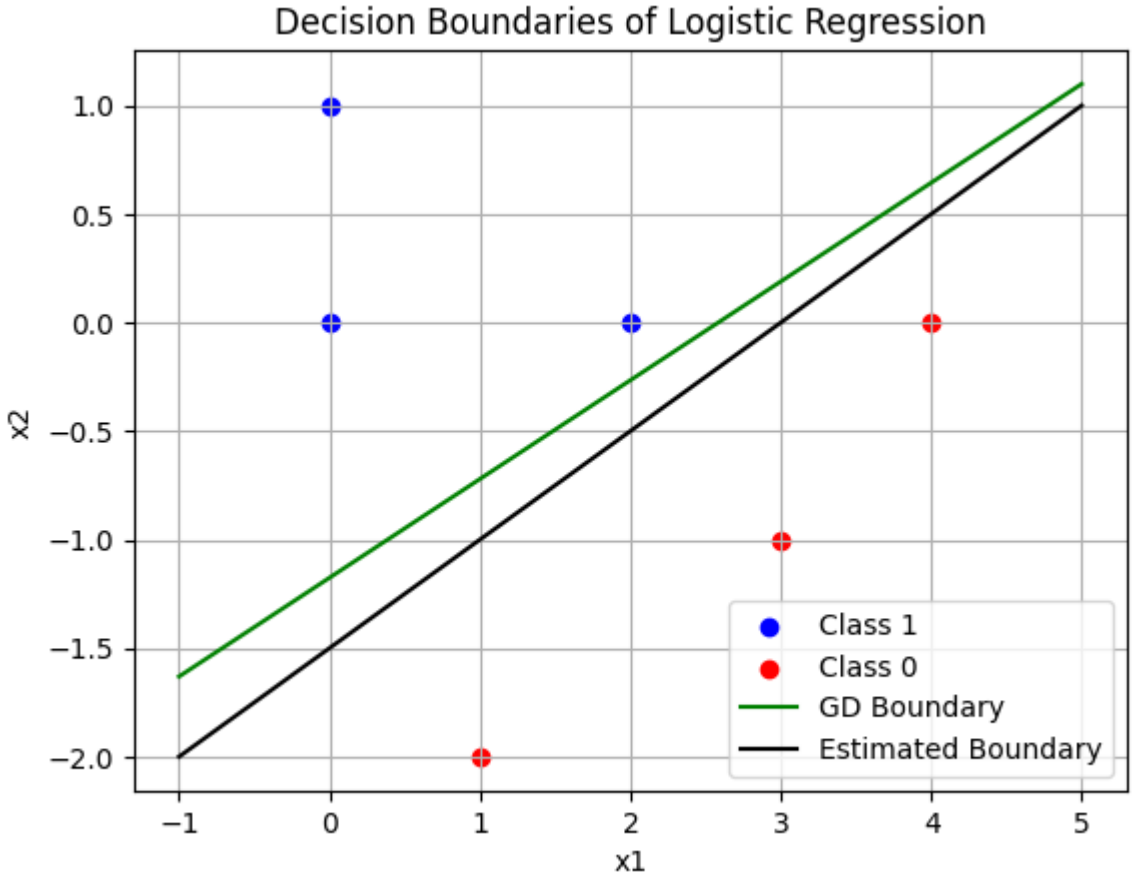
B) 2 D Logistic Regression on dummy data to acheive 100 percent accuracy

- Generate a scatter plot of the given data to visually see the decision boundary.

- Plot the decision boundary of the estimated theta values from the equation solved by hand.
- Compute optimal theta values using gradient descent.

In [ ]:
```
def decision_boundary(x1, theta):
    return (-theta[0] - theta[1]*x1) / theta[2]

theta = np.random.rand(3)

X = np.array([[1, 0, 0],
              [1, 2, 0],
              [1, 0, 1],
              [1, 1, -2],
              [1, 4, 0],
              [1, 3, -1]])

y = np.array([1, 1, 1, 0, 0, 0])

alpha = 0.01
num_iterations = 1000

for _ in range(num_iterations):

    y_pred = sigmoid(np.dot(X, theta))

    gradient = np.dot(X.T, (y_pred - y))

    theta -= alpha * gradient

theta_estimated = [3, -1, 2]

print(f'Estimated Theta :',theta_estimated)
print(f'GD Theta :',theta)

x1_values = np.linspace(-1, 5, 100)
theta = np.array([theta[0], theta[1], theta[2]])
x2_values = decision_boundary(x1_values, theta)
x2_values_estimated = decision_boundary(x1_values, theta_estimated)

plt.scatter(X[y == 1, 1], X[y == 1, 2], color='blue', label='Class 1')
plt.scatter(X[y == 0, 1], X[y == 0, 2], color='red', label='Class 0')

plt.plot(x1_values, x2_values, color='green', label='GD Boundary')
plt.plot(x1_values, x2_values_estimated, color='black', label='Estimated
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Decision Boundaries of Logistic Regression')
plt.legend()
plt.grid(True)
plt.show()
```

```
Estimated Theta : [3, -1, 2]
GD Theta : [ 3.30155484 -1.27768575  2.80802699]
```



Question - 3

A) Train test split based on given criteria and feature distribution histograms for train and test dataset.

- Drop the features supposed to be ignored and fill missing values by mode.
- Create seperate dataframes from each value of num and select 20 percent for test dataset.
- Plot histograms for each feature in both train and test data set.

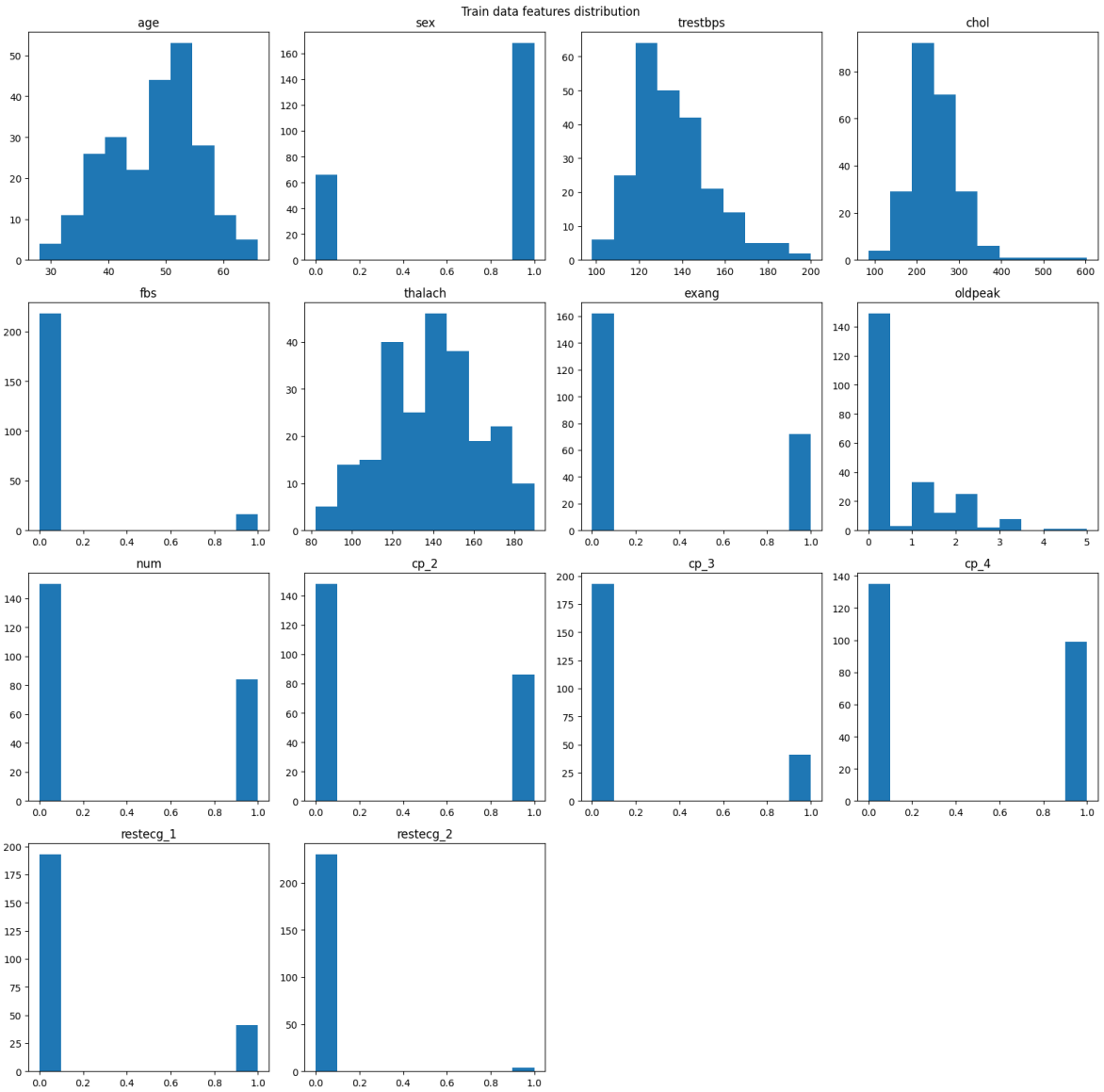
In [ ]:
```
df = pd.read_csv('/content/data.csv')

df.rename(columns ={'num       ': 'num'},inplace=True)
df.drop(['slope','ca','thal'],axis=1, inplace=True)
df.replace('?', np.nan, inplace=True)

for i in ['trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang']:
    df[i].fillna(df[i].mode()[0], inplace=True)

df = pd.get_dummies(df, columns=['cp', 'restecg'], drop_first=True)
```

In [ ]:
```
num_0 = df[df['num'] == 0]
num_1 = df[df['num'] == 1]

train_num_0, test_num_0 = train_test_split(num_0, test_size=0.2)
train_num_1, test_num_1 = train_test_split(num_1, test_size=0.2)
```

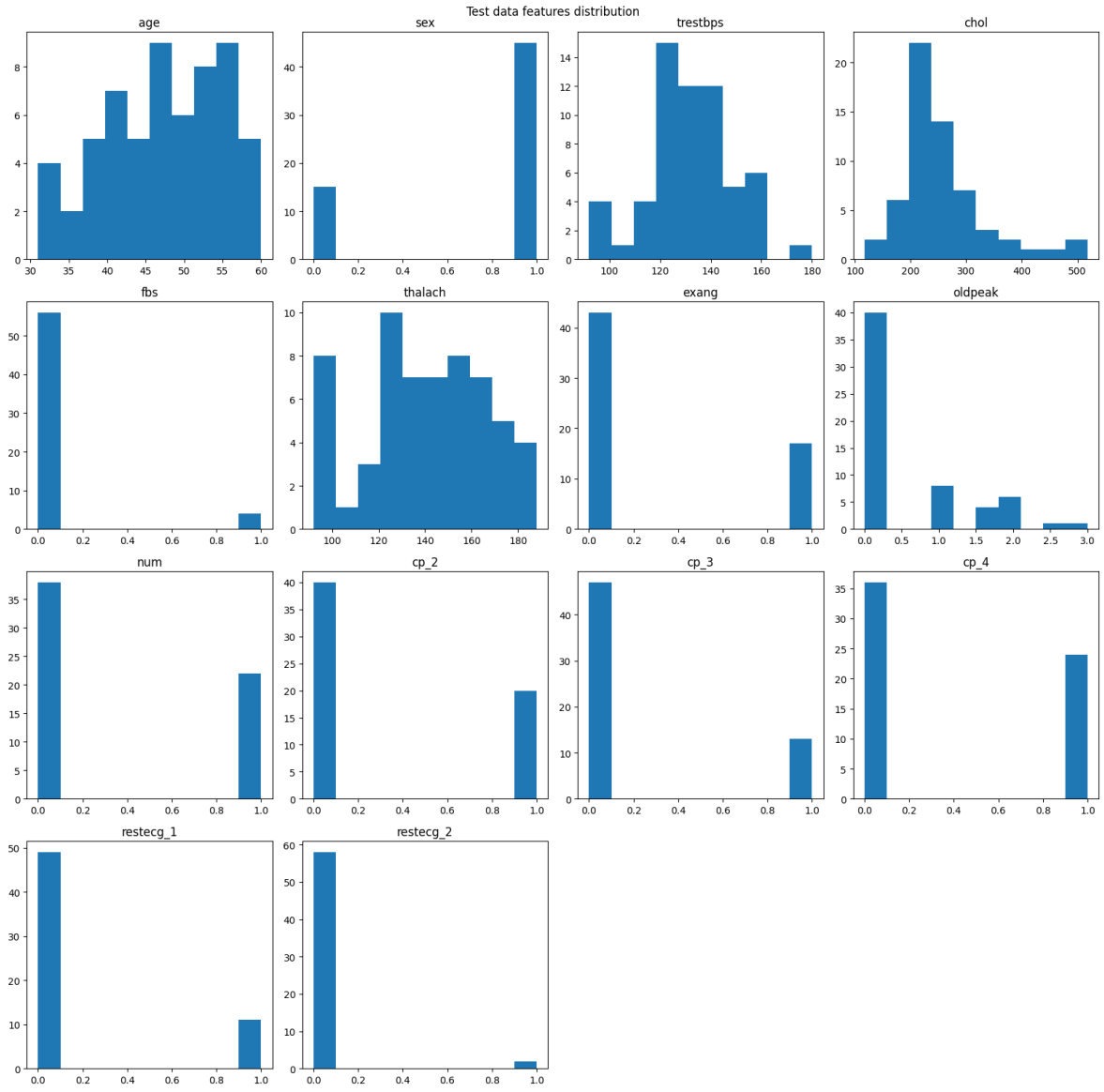
```
test_data = pd.concat([test_num_0, test_num_1])
train_data = pd.concat([train_num_0, train_num_1])

print(test_data.shape)
print(train_data.shape)
```

```
(60, 14)
(234, 14)
```

In [ ]:
```
train_data.head()
feature = 0
fig, axs = plt.subplots(4,4,figsize=(16,16))
fig.suptitle('Train data features distribution')
for i in range(4):
  for j in range(4):
    axs[i][j].set_title(f'{train_data.columns[feature]}')
    axs[i][j].hist(np.asarray(train_data[train_data.columns[feature]] ,fl
    feature += 1
    if feature > 13:
      break
fig.delaxes(axs[3][2])
fig.delaxes(axs[3][3])
fig.tight_layout()
```



In [ ]:
```
test_data.head()
feature = 0
fig, axs = plt.subplots(4,4,figsize=(16,16))
fig.suptitle('Test data features distribution')
for i in range(4):
  for j in range(4):
    axs[i][j].set_title(f'{test_data.columns[feature]}')
    axs[i][j].hist(np.asarray(test_data[test_data.columns[feature]] ,floa
    feature += 1
    if feature > 13:
      break
fig.delaxes(axs[3][2])
fig.delaxes(axs[3][3])
fig.tight_layout()
```



In [ ]:
```
numerical_features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']

scaler = StandardScaler()

train_data[numerical_features] = scaler.fit_transform(train_data[numerica
test_data[numerical_features] = scaler.fit_transform(test_data[numerical_

train_data.head()
```

Out[ ]:

	age	sex	trestbps	chol	fbs	thalach	exang	oldpeak	num	cp_
172	1.159710	0	2.681880	1.621759	0	-0.552383	1	0.213462	0	
182	1.416570	1	0.404399	-0.269073	0	0.052634	0	-0.647748	0	
164	0.902851	1	-0.734342	0.377365	0	0.052634	0	-0.647748	0	
102	0.132272	0	-1.303712	-0.269073	0	0.916944	0	-0.647748	0	
56	-0.766737	1	-0.734342	-0.301395	0	0.571220	1	0.967021	0	

B) Binary classifier using Logistic Regression using only the features: age, sex, cp, chol.

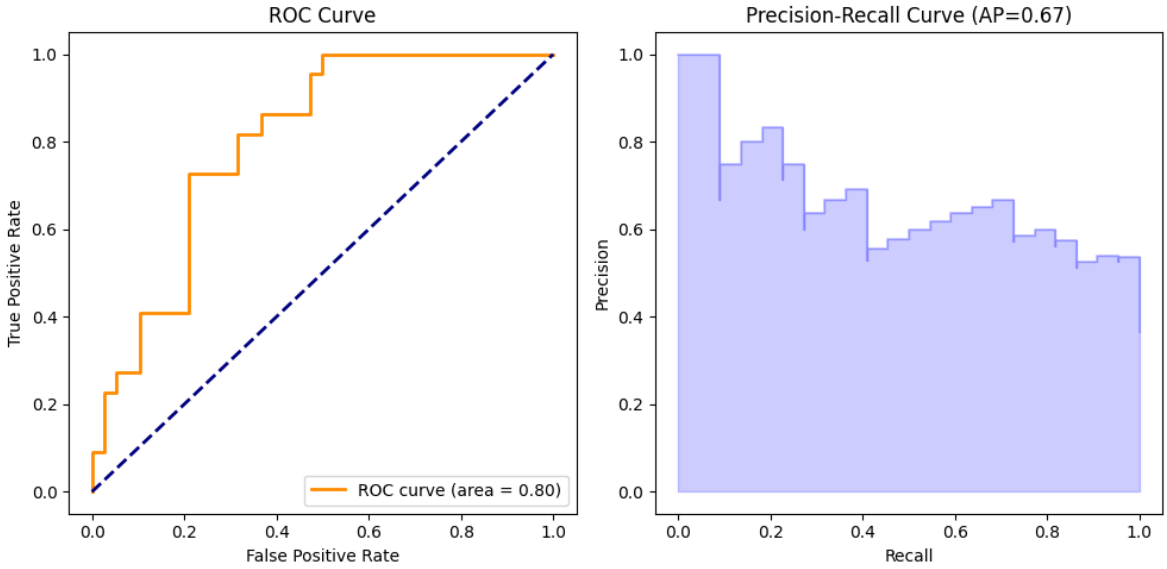
- Generate train and test splits of features and labels for given features.
- Fit the logistic regression model.
- Calculate fpr, tpr, precision and recall.
- Plot ROC and precision-recall curve.

In [ ]:
```
def generate_plots(fpr, tpr, recall, precision, roc_auc, average_precisio
  fig, axes = plt.subplots(1, 2, figsize=(10, 5))

  axes[0].plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (are
  axes[0].plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
  axes[0].set_xlabel('False Positive Rate')
  axes[0].set_ylabel('True Positive Rate')
  axes[0].set_title('ROC Curve')
  axes[0].legend(loc="lower right")

  axes[1].step(recall, precision, color='b', alpha=0.2, where='post')
  axes[1].fill_between(recall, precision, step='post', alpha=0.2, color='
  axes[1].set_xlabel('Recall')
  axes[1].set_ylabel('Precision')
  axes[1].set_title('Precision-Recall Curve (AP={:.2f})'.format(average_p

  plt.tight_layout()

  return plt
```

In [ ]:
```
selected = ['age','sex','cp_2', 'cp_3', 'cp_4','chol']
target = 'num'

train_1 = train_data[selected + [target]].dropna()
test_1 = test_data[selected + [target]].dropna()

x_train_1 = train_1[selected]
x_test_1 = test_1[selected]
y_train_1 = train_1[target]
y_test_1 = test_1[target]

model = make_pipeline(SimpleImputer(strategy='mean'), LogisticRegression(
model.fit(x_train_1, y_train_1)

y_score = model.predict_proba(x_test_1)

fpr, tpr, _ = roc_curve(y_test_1.ravel(), y_score[:, 1].ravel())
```

```
roc_auc = auc(fpr, tpr)

precision, recall, _ = precision_recall_curve(y_test_1.ravel(), y_score[:
average_precision = average_precision_score(y_test_1.ravel(), y_score[:,

generate_plots(fpr,tpr,recall,precision,roc_auc,average_precision)
```

Out[ ]:
```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-package
s/matplotlib/pyplot.py'>
```



C) Binary classifier using Logistic Regression using only the features: age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak.

- Generate train and test splits of features and labels for given features.
- Fit the logistic regression model.
- Calculate fpr, tpr, precision and recall.
- Plot ROC and precision-recall curve.

In [ ]:
```
selected = ['age','sex','cp_2', 'cp_3', 'cp_4','trestbps','chol','fbs','r
target = 'num'

train_2 = train_data[selected + [target]].dropna()
test_2 = test_data[selected + [target]].dropna()

x_train_2 = train_2[selected]
x_test_2 = test_2[selected]
y_train_2 = train_2[target]
y_test_2 = test_2[target]

model = make_pipeline(SimpleImputer(strategy='mean'), LogisticRegression(
model.fit(x_train_2, y_train_2)

y_score = model.predict_proba(x_test_2)

fpr, tpr, _ = roc_curve(y_test_2.ravel(), y_score[:, 1].ravel())
roc_auc = auc(fpr, tpr)

precision, recall, _ = precision_recall_curve(y_test_2.ravel(), y_score[:
average_precision = average_precision_score(y_test_2.ravel(), y_score[:,
```

```
generate_plots(fpr,tpr,recall,precision,roc_auc,average_precision)
```

Out[ ]: <module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>