**SOURCE CODE –HEART DISEASE PREDICTION**

1.DATA PREPROCESSING

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.utils import resample

# Load the data

df = pd.read\_csv('C:/Users/liyan/Downloads/heart\_data.csv')

# Display basic information

print(df.head())

print(df.info())

# Check for missing values

print(df.isnull().sum())

# Handle missing values

df.fillna(df.mean(), inplace=True)

# Identify columns to drop

columns\_to\_drop = ['id', 'unnecessary\_column']

df.drop(columns=[col for col in columns\_to\_drop if col in df.columns], inplace=True)

# Encode categorical features

categorical\_cols = df.select\_dtypes(include=['object']).columns

df = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)

# Handle class imbalance

majority\_class = df[df['diagnosis'] == 0]

minority\_class = df[df['diagnosis'] == 1]

# Upsample minority class

minority\_upsampled = resample(minority\_class,

replace=True,

n\_samples=len(majority\_class),

random\_state=42)

# Combine majority class with upsampled minority class

df\_balanced = pd.concat([majority\_class, minority\_upsampled])

# Shuffle the balanced dataset

df\_balanced = df\_balanced.sample(frac=1, random\_state=42).reset\_index(drop=True)

# Split the data into features and target

X = df\_balanced.drop('diagnosis', axis=1)

y = df\_balanced['diagnosis']

# Split the data 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)

# Final preprocessed data

print("Training

OUTPUT

rest\_bp chest\_pain thalassemia ... rest\_ecg num\_vessels diagnosis

0 106 3 0 ... 0 2 0

1 120 2 0 ... 0 0 0

2 126 3 2 ... 2 0 1

3 150 3 2 ... 2 3 1

4 140 3 2 ... 0 2 1

data shape:", X\_train.shape)

print("Testing data shape:", X\_test.shape)

[5 rows x 14 columns]

RangeIndex: 297 entries, 0 to 296

Data columns (total 14 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 rest\_bp 297 non-null int64

1 chest\_pain 297 non-null int64

2 thalassemia 297 non-null int64

3 age 297 non-null int64

4 fasting\_bs 297 non-null int64

5 max\_hr 297 non-null int64

6 exercise\_angina 297 non-null int64

7 gender 297 non-null int64

8 st\_slope 297 non-null int64

9 cholesterol 297 non-null int64

10 st\_depression 297 non-null float64

11 rest\_ecg 297 non-null int64

12 num\_vessels 297 non-null int64

13 diagnosis 297 non-null int64

dtypes: float64(1), int64(13)

memory usage: 32.6 KB

None

rest\_bp 0

chest\_pain 0

thalassemia 0

age 0

fasting\_bs 0

max\_hr 0

exercise\_angina 0

gender 0

st\_slope 0

cholesterol 0

st\_depression 0

rest\_ecg 0

num\_vessels 0

diagnosis 0

dtype: int64

Training data shape: (256, 13)

Testing data shape: (64, 13)

2. DATA TABLE PROPERTIES

import pandas as pd

import matplotlib.pyplot as plt

from PIL import Image, ImageDraw, ImageFont

# Load the data

data = pd.read\_csv("C:/Users/liyan/Downloads/heart\_data.csv", header=None, skiprows=1)

# Set column names

data.columns = ['rest\_bp', 'chest\_pain', 'thalassemia', 'age', 'fasting\_bs', 'max\_hr', 'exercise\_angina', 'gender', 'st\_slope',

'cholesterol', 'st\_depression', 'rest\_ecg', 'num\_vessels', 'diagnosis']

# Prepare dataset information

num\_rows, num\_columns = data.shape

num\_categorical = 8 # As per your provided information

num\_numeric = 6 # As per your provided information

target\_variable = 'diagnosis'

num\_classes = data[target\_variable].nunique()

# Create a blank image

img = Image.new('RGB', (400, 300), color='white')

draw = ImageDraw.Draw(img)

# Set font

font = ImageFont.load\_default()

# Draw text

draw.text((10, 10), "Data table properties", fill="black", font=font)

draw.text((10, 40), "Name: Heart Data", fill="black", font=font)

draw.text((10, 70), f"Size: {num\_rows} rows, {num\_columns} columns", fill="black", font=font)

draw.text((10, 100), f"Features: {num\_categorical} categorical, {num\_numeric} numeric", fill="black", font=font)

draw.text((10, 130), f"Targets: categorical outcome with {num\_classes} classes", fill="black", font=font)

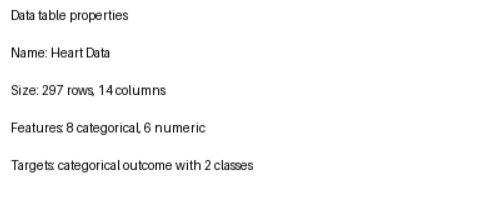
# Save the image

img.save("data\_table\_properties.png")

# Display the image

img.show()

OUTPUT



3.DISTRIBUTION OF TARGET VARIABLE VS DIAGNOSIS

import pandas as pd

import matplotlib.pyplot as plt

# Load the dataset

data = pd.read\_csv('C:/Users/liyan/Downloads/heart\_data.csv')

# Count the occurrences of each value in the 'diagnosis' column

diagnosis\_counts = data['diagnosis'].value\_counts()

# Plotting

plt.figure(figsize=(8, 6))

# Define colors for 'yes' and 'no'

colors = ['blue', 'green']

bars = plt.bar(diagnosis\_counts.index, diagnosis\_counts.values, color=colors, width=0.2)

# Adding labels and title

plt.xlabel('Diagnosis')

plt.ylabel('Count')

plt.title('Distribution of Diagnosis')

# Adding annotations to each bar

for bar, diagnosis in zip(bars, diagnosis\_counts.index):

yval = bar.get\_height()

plt.text(bar.get\_x() + bar.get\_width()/2, yval + 10, yval, ha='center', va='bottom')

# Add annotation for color

if diagnosis == 'Yes':

plt.text(bar.get\_x() + bar.get\_width() + 0.02, yval, 'Yes', ha='left', va='center', color='blue')

elif diagnosis == 'No':

plt.text(bar.get\_x() + bar.get\_width() + 0.02, yval, 'No', ha='left', va='center', color='green')

# Adding legend for colors

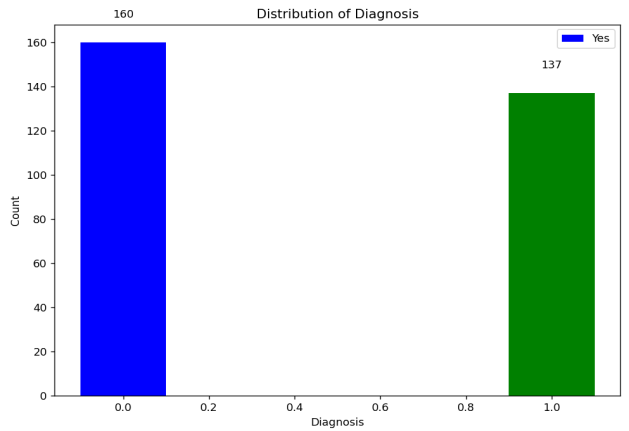
plt.legend(['Yes', 'No'], loc='upper right')

# Display the plot

plt.tight\_layout()

plt.show()

OUTPUT



4.TARGET VARIABLE AFTER OVERSAMPLING

import pandas as pd

import matplotlib.pyplot as plt

from imblearn.over\_sampling import SMOTE

# Load your dataset (replace the path with your actual dataset path)

data = pd.read\_csv('C:/Users/liyan/Downloads/heart\_data.csv')

# Assuming 'diagnosis' is your target variable column name

target = data['diagnosis']

# Perform oversampling with SMOTE

oversampler = SMOTE(random\_state=42)

X\_resampled, y\_resampled = oversampler.fit\_resample(data.drop('diagnosis', axis=1), target)

# Calculate value counts before and after oversampling

target\_value\_counts\_before = target.value\_counts()

target\_value\_counts\_after = pd.Series(y\_resampled).value\_counts()

# Plotting

plt.figure(figsize=(8, 6))

# Plot after oversampling with reduced width of bars

plt.bar(target\_value\_counts\_after.index, target\_value\_counts\_after.values, width=0.2, color=['blue', 'green'])

plt.title('Class Distribution after SMOTE Oversampling')

plt.xlabel('Diagnosis')

plt.ylabel('Count')

for i, value in enumerate(target\_value\_counts\_after.values):

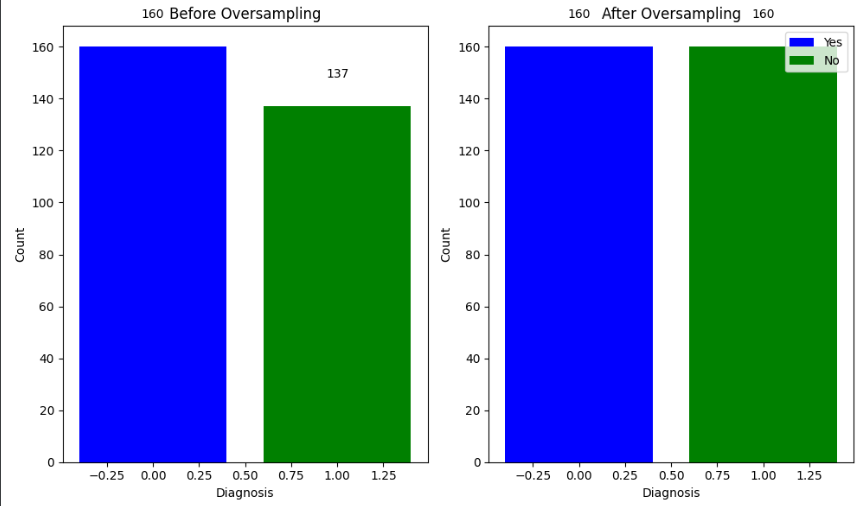
plt.text(i, value + 10, str(value), ha='center', va='bottom')

# Adjust layout and display the plot

plt.tight\_layout()

plt.show()

OUTPUT



5.VIOLIN PLOT OF THALASSEMIA VS DIAGNOSIS

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load your dataset

data = pd.read\_csv('C:/Users/liyan/Downloads/heart\_data.csv')

# Convert thalassemia to a categorical variable

data['thalassemia'] = data['thalassemia'].astype('category')

# Create a violin plot of thalassemia vs. diagnosis

sns.violinplot(x='thalassemia', y='diagnosis', data=data, hue='thalassemia', palette='Set3')

# Set the title of the plot

plt.title('Violin Plot of Thalassemia vs. Diagnosis')

# Set the x-axis label

plt.xlabel('Thalassemia')

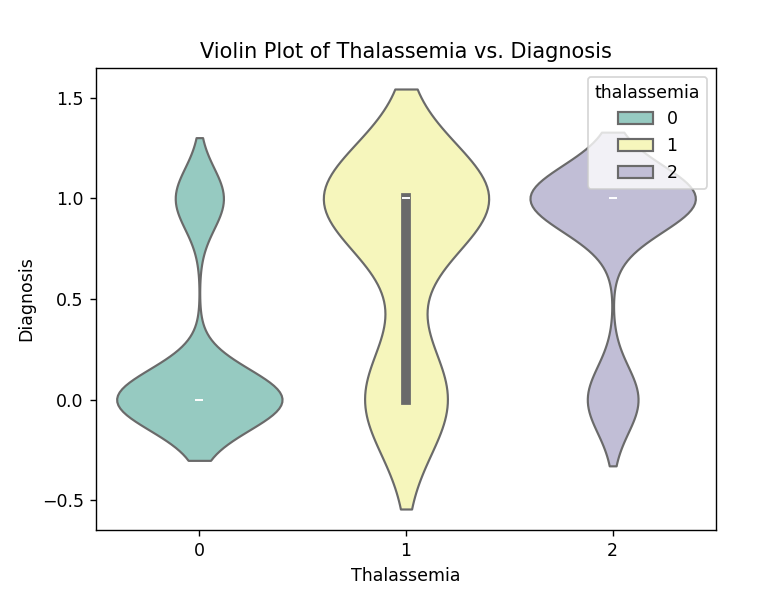
# Set the y-axis label

plt.ylabel('Diagnosis')

# Show the plot

plt.show()

OUTPUT



6.DISTRIBUTION OF CHEST PAIN VS DIAGNOSIS

import matplotlib.pyplot as plt

import numpy as np

# Data

chest\_pain = ['Typical', 'Atypical', 'Non-Anginal', 'Asymptomatic']

diagnosis\_0 = [20, 15, 10, 5]

diagnosis\_1 = [30, 25, 20, 10]

# Create a figure and axis

fig, ax = plt.subplots()

# Set the x-axis ticks and labels

ax.set\_xticks(np.arange(len(chest\_pain)))

ax.set\_xticklabels(chest\_pain)

# Plot the bars

bar\_width = 0.35

ax.bar(np.arange(len(chest\_pain)) - bar\_width/2, diagnosis\_0, bar\_width, label='Diagnosis 0')

ax.bar(np.arange(len(chest\_pain)) + bar\_width/2, diagnosis\_1, bar\_width, label='Diagnosis 1')

# Set the title and labels

ax.set\_title('Distribution of Chest Pain vs Diagnosis')

ax.set\_xlabel('Chest Pain')

ax.set\_ylabel('Count')

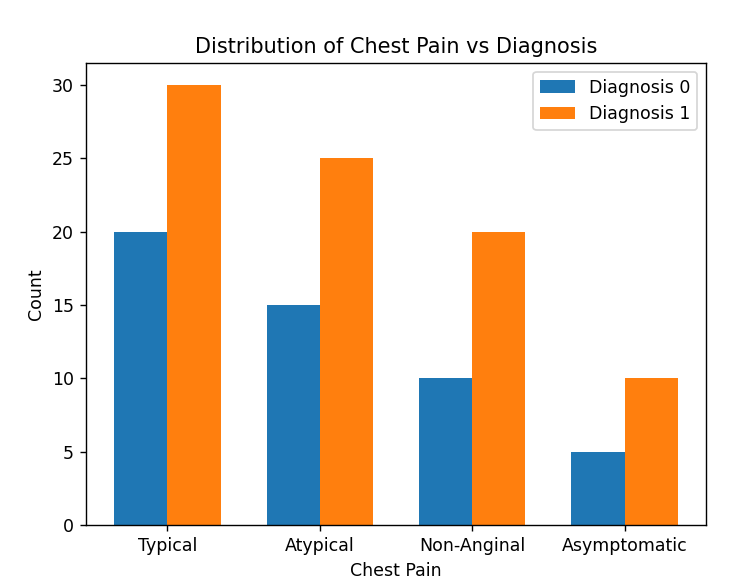
# Legend

ax.legend()

# Show the plot

plt.show()

OUTPUT



7.DISTRIBUTION OF RESTING BLOOD PRESSURE VS DIAGNOSIS

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

# Load the dataset

data = pd.read\_csv('C:/Users/liyan/Downloads/heart\_data.csv')

# Aggregate the data

rest\_bp\_diagnosis = data.groupby(['rest\_bp', 'diagnosis']).size().unstack(fill\_value=0)

# Plot the bar chart

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

# Set the width of each bar

bar\_width = 0.35

# Generate positions for each bar

r1 = np.arange(len(rest\_bp\_diagnosis))

r2 = [x + bar\_width for x in r1]

# Plot bars for each diagnosis

ax.bar(r1, rest\_bp\_diagnosis[0], color='blue', width=bar\_width, edgecolor='grey', label='Diagnosis 0')

ax.bar(r2, rest\_bp\_diagnosis[1], color='red', width=bar\_width, edgecolor='grey', label='Diagnosis 1')

# Set the x-axis ticks and labels

ax.set\_xticks([r + bar\_width/2 for r in range(len(rest\_bp\_diagnosis))])

ax.set\_xticklabels(rest\_bp\_diagnosis.index, rotation=90)

# Set the title and labels

ax.set\_title('Distribution of Resting Blood Pressure vs Diagnosis')

ax.set\_xlabel('Resting Blood Pressure')

ax.set\_ylabel('Count')

# Add legend

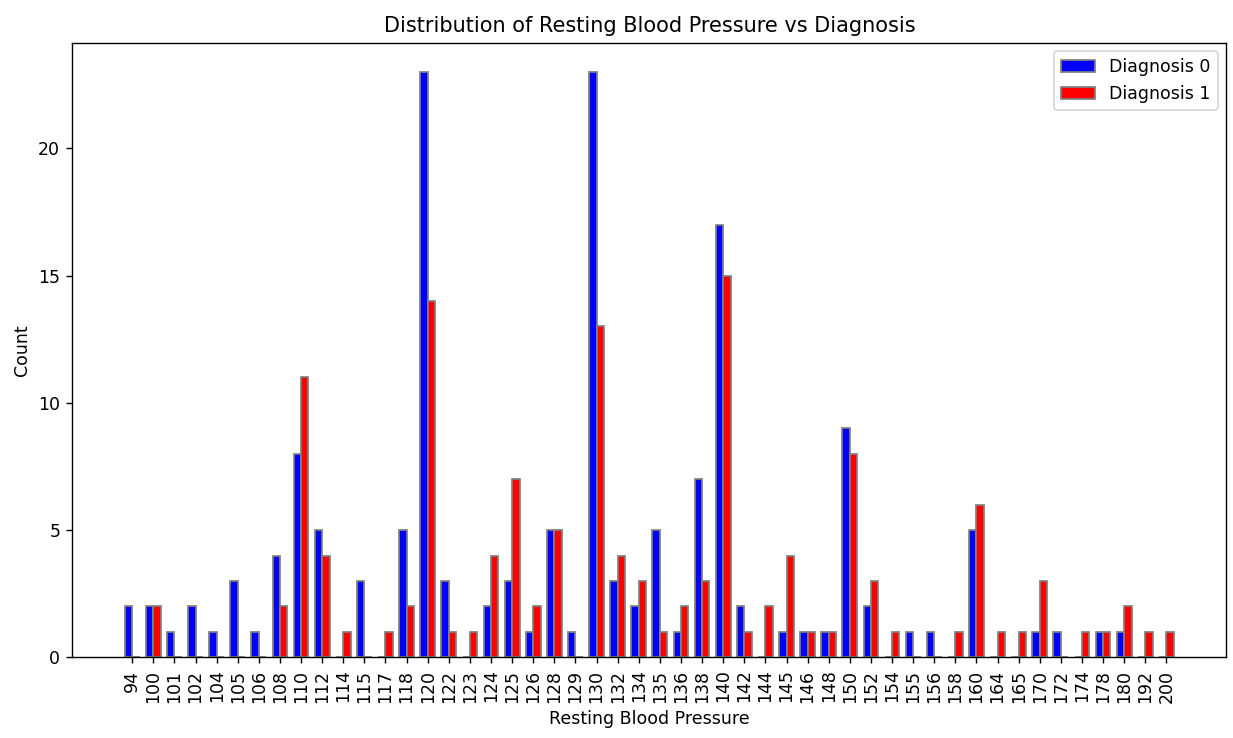
ax.legend()

# Display the plot

plt.tight\_layout()

plt.show()

OUTPUT



8.DISTRIBUTION OF AGE VS DIAGNOSIS

import pandas as pd

import matplotlib.pyplot as plt

# Load the dataset

data = pd.read\_csv('C:/Users/liyan/Downloads/heart\_data.csv')

# Display the first few rows of the dataset to understand its structure

print(data.head())

# Assuming the dataset has columns 'age' and 'diagnosis'

# Create a scatter plot

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

plt.scatter(data['age'], data['diagnosis'], alpha=0.5, c=data['diagnosis'], cmap='viridis')

plt.title('Scatter Plot of Age vs Diagnosis')

plt.xlabel('Age')

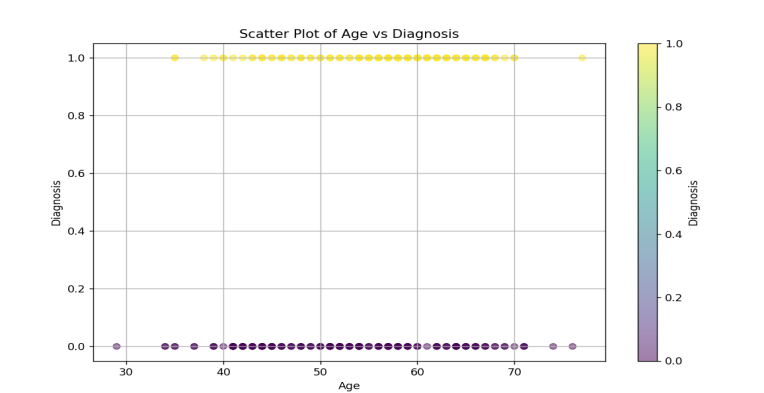
plt.ylabel('Diagnosis')

plt.colorbar(label='Diagnosis')

plt.grid(True)

plt.show()

OUTPUT



9.DISTRIBUTION OF NUMBER OF VESSELS VS DIAGNOSIS

import pandas as pd

import matplotlib.pyplot as plt

# Load the dataset

data = pd.read\_csv('C:/Users/liyan/Downloads/heart\_data.csv')

# Calculate the average number of vessels for each diagnosis

avg\_vessels = data.groupby('diagnosis')['num\_vessels'].mean().reset\_index()

# Create a bar plot with different colors for each bar

colors = plt.cm.viridis(avg\_vessels['diagnosis'] / max(avg\_vessels['diagnosis']))

plt.figure(figsize=(8, 4))

bars = plt.bar(avg\_vessels['diagnosis'], avg\_vessels['num\_vessels'], color=colors, width=0.2) # Adjust width here

# Adding labels, title, and grid

plt.title('Average Number of Vessels for Each Diagnosis')

plt.xlabel('Diagnosis')

plt.ylabel('Average Number of Vessels')

plt.grid(axis='y')

# Adding value labels on top of each bar

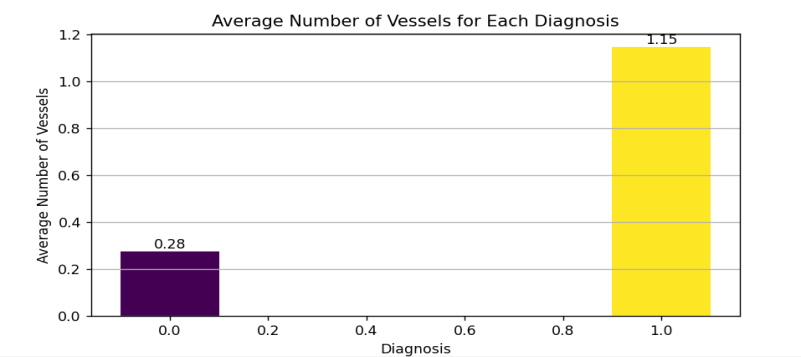
for bar in bars:

yval = bar.get\_height()

plt.text(bar.get\_x() + bar.get\_width()/2, yval, round(yval, 2), ha='center', va='bottom')

plt.show()

OUTPUT



10.DISTRIBUTION OF ST\_DEPRESSION VS DIAGNOSIS

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset (assuming 'heart\_data.csv' contains the relevant data)

data = pd.read\_csv('C:/Users/liyan/Downloads/heart\_data.csv')

# Calculate the average st\_depression for each diagnosis

avg\_st\_depression = data.groupby('diagnosis')['st\_depression'].mean().reset\_index()

# Create a bar plot with reduced width

plt.figure(figsize=(8, 4))

sns.barplot(x='diagnosis', y='st\_depression', data=data, ci=None, palette='viridis', capsize=0.1)

# Adding labels and title

plt.title('Distribution of ST Depression vs Diagnosis')

plt.xlabel('Diagnosis')

plt.ylabel('Average ST Depression')

# Customize the plot

plt.grid(axis='y')

# Reduce width of the bars

for patch in plt.gca().patches:

patch.set\_width(0.2) # Adjust width as desired

# Add annotations for colors

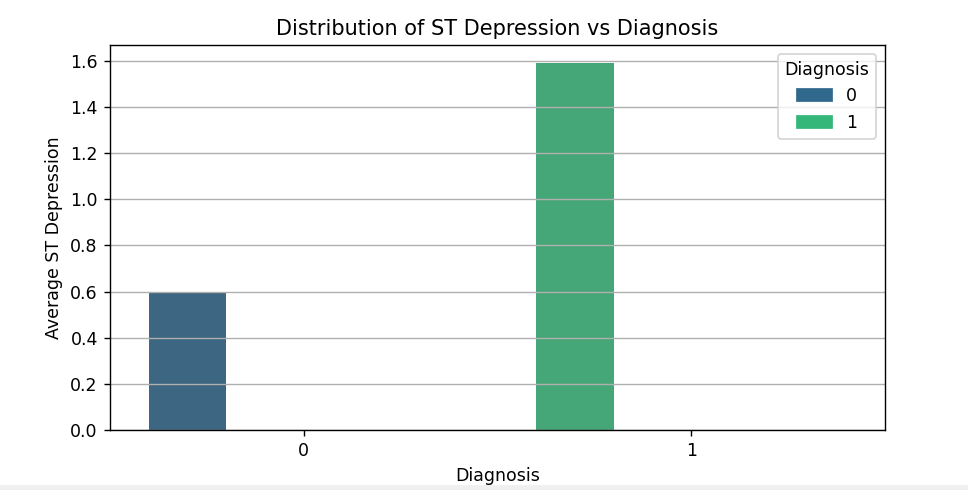
colors = sns.color\_palette('viridis', len(avg\_st\_depression['diagnosis']))

legend\_handles = [plt.Rectangle((0,0),1,1, color=colors[i], edgecolor='black') for i in range(len(avg\_st\_depression['diagnosis']))]

plt.legend(legend\_handles, avg\_st\_depression['diagnosis'], title='Diagnosis')

plt.show()

OUTPUT



11.DISTRIBUTION OF CHOLESTROL VS DIAGNOSIS

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset (assuming 'heart\_data.csv' contains the relevant data)

data = pd.read\_csv('C:/Users/liyan/Downloads/heart\_data.csv')

# Plotting using seaborn

plt.figure(figsize=(8, 4))

sns.boxplot(x='diagnosis', y='cholesterol', data=data, palette='viridis')

# Adding labels and title

plt.title('BoxPlot of Cholesterol Levels vs Diagnosis')

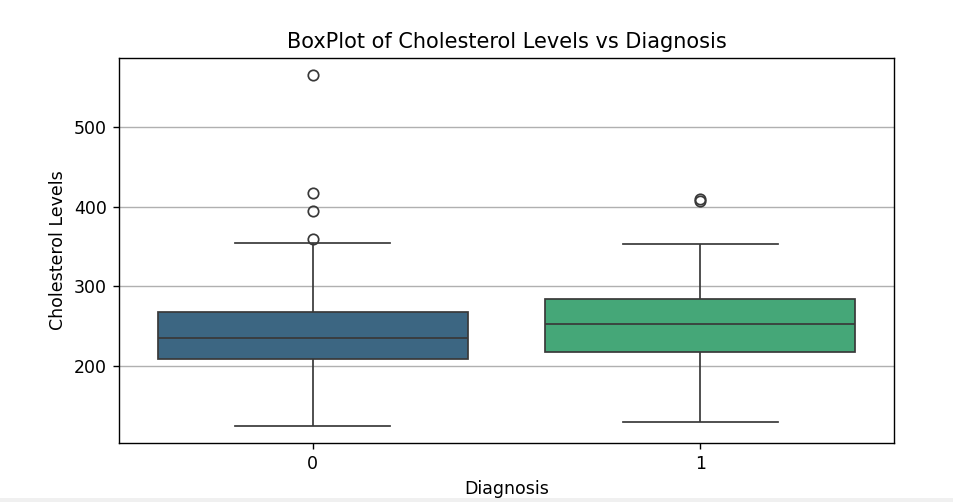
plt.xlabel('Diagnosis')

plt.ylabel('Cholesterol Levels')

plt.grid(axis='y')

plt.show()

OUTPUT



12.METRICS OF MODEL EVALUATION

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, classification\_report

# Load the dataset from specified path

data\_path = 'C:/Users/liyan/Downloads/heart\_data.csv'

data = pd.read\_csv(data\_path)

# Assuming 'diagnosis' is the target variable, and others are features

X = data.drop('diagnosis', axis=1)

y = data['diagnosis']

# Split data 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)

# Define classifiers

models = {

'Decision Tree': DecisionTreeClassifier(random\_state=42),

'SVM': SVC(random\_state=42, probability=True),

'Random Forest': RandomForestClassifier(random\_state=42),

'Naive Bayes': GaussianNB(),

'Log Reg': LogisticRegression(random\_state=42, max\_iter=1000),

'AdaBoost': AdaBoostClassifier(random\_state=42)

}

# Initialize lists to store metrics

model\_names = []

aucs = []

accuracies = []

precisions = []

recalls = []

f1\_scores = []

# Train and evaluate each model

for name, model in models.items():

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

y\_prob = model.predict\_proba(X\_test)[:, 1] # Probability for positive class

# Evaluate metrics

auc = roc\_auc\_score(y\_test, y\_prob)

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)

# Store metrics

model\_names.append(name)

aucs.append(auc)

accuracies.append(accuracy)

precisions.append(precision)

recalls.append(recall)

f1\_scores.append(f1)

# Create a DataFrame to store metrics

metrics\_df = pd.DataFrame({

'Model': model\_names,

'AUC': aucs,

'CA': accuracies,

'F1': f1\_scores,

'Precision': precisions,

'Recall': recalls

})

# Display the classification report as a DataFrame (optional)

# Here we'll print the classification report for each model

for name, model in models.items():

y\_pred = model.predict(X\_test)

report = classification\_report(y\_test, y\_pred, output\_dict=True)

print(f"Model: {name}")

df\_report = pd.DataFrame(report).transpose()

print(df\_report)

print("\n")

# Plotting the styled table

plt.figure(figsize=(8,4))

plt.title('Model Evaluation Metrics', fontsize=16)

plt.axis('off')

# Prepare cell text for table with formatted values

cell\_text = []

for \_, row in metrics\_df.iterrows():

row\_values = [f'{val:.3f}' if isinstance(val, float) else val for val in row.values]

cell\_text.append(row\_values)

# Create the table

table = plt.table(cellText=cell\_text, colLabels=metrics\_df.columns, cellLoc='center', loc='center')

table.auto\_set\_font\_size(False)

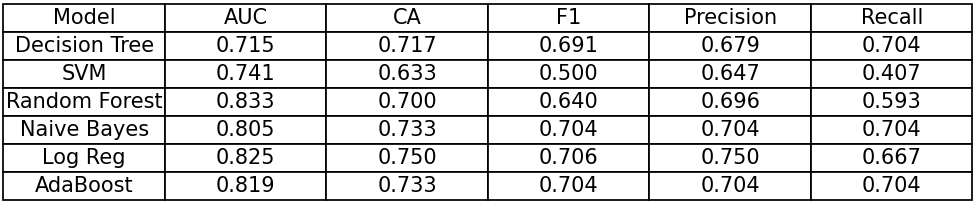
table.set\_fontsize(12)

table.scale(1.2, 1.2) # Adjust table size if necessary

plt.tight\_layout() # Ensures the table fits within the figure area

plt.show()

OUTPUT



13.METRICS OF MODEL EVALUATION WITH UNBALANCED DATA

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score

# Load the dataset from specified path

data\_path = 'C:/Users/liyan/Downloads/heart\_data1.csv'

data = pd.read\_csv(data\_path)

# Assuming 'diagnosis' is the target variable, and others are features

X = data.drop('diagnosis', axis=1)

y = data['diagnosis']

# Split data into training and testing sets (80% for training, 20% for testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define classifiers

models = {

'Decision Tree': DecisionTreeClassifier(random\_state=42),

'SVM': SVC(random\_state=42, probability=True),

'Random Forest': RandomForestClassifier(random\_state=42),

'Naive Bayes': GaussianNB(),

'Log Reg': LogisticRegression(random\_state=42, max\_iter=1000),

'AdaBoost': AdaBoostClassifier(random\_state=42)

}

# Initialize lists to store metrics

model\_names = []

aucs = []

accuracies = []

precisions = []

recalls = []

f1\_scores = []

# Train and evaluate each model

for name, model in models.items():

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

y\_prob = model.predict\_proba(X\_test)[:, 1] # Probability for positive class

# Evaluate metrics

auc = roc\_auc\_score(y\_test, y\_prob)

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)

# Store metrics

model\_names.append(name)

aucs.append(round(auc, 3)) # Round to 3 decimal places

accuracies.append(round(accuracy, 3)) # Round to 3 decimal places

precisions.append(round(precision, 3)) # Round to 3 decimal places

recalls.append(round(recall, 3)) # Round to 3 decimal places

f1\_scores.append(round(f1, 3)) # Round to 3 decimal places

# Create a DataFrame to store metrics

metrics\_df = pd.DataFrame({

'Model': model\_names,

'AUC': aucs,

'CA': accuracies, # Changed 'CA' to 'Accuracy'

'F1': f1\_scores,

'Precision': precisions,

'Recall': recalls

})

# Create the first figure (Metrics of models built with unbalanced data)

fig1, ax1 = plt.subplots(figsize=(10, 6))

# Hide axes

ax1.axis('off')

# Create a table for the first figure

table1 = ax1.table(

cellText=metrics\_df.values.tolist(),

colLabels=metrics\_df.columns,

cellLoc='center',

loc='center'

)

table1.auto\_set\_font\_size(False)

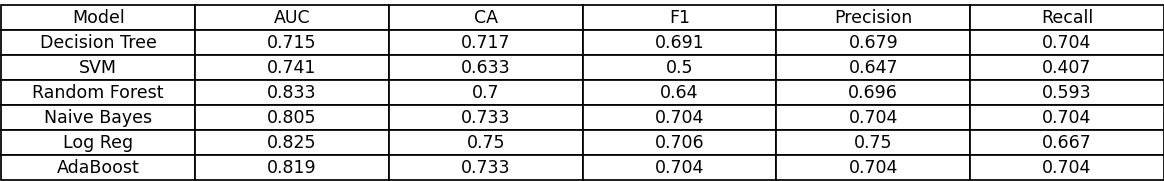
table1.set\_fontsize(10)

table1.scale(1.2, 1.2)

plt.title('Metrics of Models Built with Unbalanced Data', fontsize=14, fontweight='bold', pad=20)

plt.show()

OUTPUT



14.EVALUATION METRICS AND CONFUSION MATRIX OF LOGISTIC REGRESSION

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix, classification\_report

# Load the dataset from specified path

data\_path = 'C:/Users/liyan/Downloads/heart\_data.csv'

data = pd.read\_csv(data\_path)

# Assuming 'diagnosis' is the target variable, and others are features

X = data.drop('diagnosis', axis=1)

y = data['diagnosis']

# Split data 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)

# Define classifiers

models = {

'Decision Tree': DecisionTreeClassifier(random\_state=42),

'SVM': SVC(random\_state=42, probability=True),

'Random Forest': RandomForestClassifier(random\_state=42),

'Naive Bayes': GaussianNB(),

'Log Reg': LogisticRegression(random\_state=42, max\_iter=1000),

'AdaBoost': AdaBoostClassifier(random\_state=42)

}

# Initialize lists to store metrics

model\_names = []

aucs = []

accuracies = []

precisions = []

recalls = []

f1\_scores = []

# Train and evaluate each model

for name, model in models.items():

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

y\_prob = model.predict\_proba(X\_test)[:, 1] # Probability for positive class

# Evaluate metrics

auc = roc\_auc\_score(y\_test, y\_prob)

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)

# Store metrics

model\_names.append(name)

aucs.append(auc)

accuracies.append(accuracy)

precisions.append(precision)

recalls.append(recall)

f1\_scores.append(f1)

# Create a DataFrame to store metrics

metrics\_df = pd.DataFrame({

'Model': model\_names,

'AUC': aucs,

'CA': accuracies,

'F1': f1\_scores,

'Precision': precisions,

'Recall': recalls

})

# Filter metrics for Logistic Regression

log\_reg\_metrics = metrics\_df[metrics\_df['Model'] == 'Log Reg']

# Compute confusion matrix for Logistic Regression

log\_reg\_model = models['Log Reg']

y\_pred\_log\_reg = log\_reg\_model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred\_log\_reg)

# Plotting the confusion matrix

plt.figure(figsize=(4, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, square=True)

plt.xlabel('Predicted', fontsize=12)

plt.ylabel('Actual', fontsize=12)

plt.title('Confusion Matrix of Logistic Regression', fontsize=14)

plt.xticks(ticks=[0.5, 1.5], labels=['0', '1'], fontsize=12)

plt.yticks(ticks=[0.5, 1.5], labels=['0', '1'], fontsize=12)

plt.tight\_layout()

plt.show()

# Plotting the table for Logistic Regression metrics

fig, ax = plt.subplots(figsize=(10, 2))

# Hide axes

ax.axis('off')

# Create a table for the metrics

table = ax.table(

cellText=log\_reg\_metrics.values.tolist(),

colLabels=log\_reg\_metrics.columns,

cellLoc='center',

loc='center'

)

table.auto\_set\_font\_size(False)

table.set\_fontsize(10)

table.scale(1.2, 1.2)

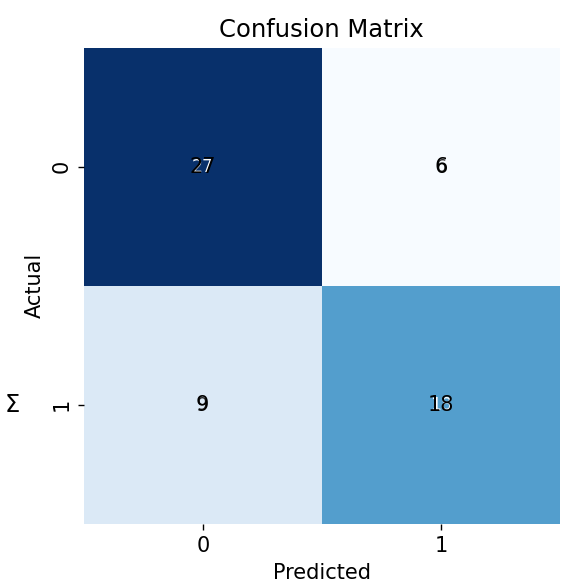
plt.title('Evaluation Metrics of Logistic Regression', fontsize=14, fontweight='bold', pad=10)

plt.tight\_layout()

plt.show()

OUTPUT

Screenshot 2024-06-25 221504.png



15.SIGMOID ACTIVATION FUNCTION OF LOGISTIC REGRESSION

import matplotlib.pyplot as plt

import numpy as np

def sigmoid(x):

return 1 / (1 + np.exp(-x))

x = np.linspace(-10, 10, 1000)

y = sigmoid(x)

plt.plot(x, y)

plt.title('Sigmoid Activation Function of Logistic Regression')

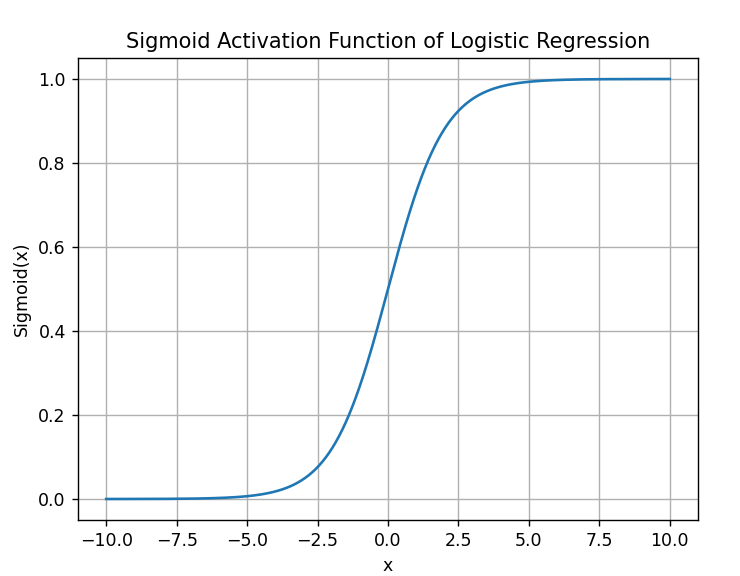
plt.xlabel('x')

plt.ylabel('Sigmoid(x)')

plt.grid()

plt.show()

OUTPUT



16.EVALUATION METRICS AND CONFUSION MATRIX OF DECISION TREE

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix

# Load the dataset from specified path

data\_path = 'C:/Users/liyan/Downloads/heart\_data.csv'

data = pd.read\_csv(data\_path)

# Assuming 'diagnosis' is the target variable, and others are features

X = data.drop('diagnosis', axis=1)

y = data['diagnosis']

# Split data into training and testing sets (80% for training, 20% for testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize Decision Tree model

decision\_tree\_model = DecisionTreeClassifier(random\_state=42)

# Train the Decision Tree model

decision\_tree\_model.fit(X\_train, y\_train)

y\_pred\_dt = decision\_tree\_model.predict(X\_test)

y\_prob\_dt = decision\_tree\_model.predict\_proba(X\_test)[:, 1] # Probability for positive class

# Evaluate metrics for Decision Tree

auc\_dt = roc\_auc\_score(y\_test, y\_prob\_dt)

accuracy\_dt = accuracy\_score(y\_test, y\_pred\_dt)

precision\_dt = precision\_score(y\_test, y\_pred\_dt)

recall\_dt = recall\_score(y\_test, y\_pred\_dt)

f1\_dt = f1\_score(y\_test, y\_pred\_dt)

# Create a DataFrame to store Decision Tree metrics

decision\_tree\_metrics = pd.DataFrame({

'Model': ['Decision Tree'],

'AUC': [round(auc\_dt, 3)],

'Accuracy': [round(accuracy\_dt, 3)],

'F1': [round(f1\_dt, 3)],

'Precision': [round(precision\_dt, 3)],

'Recall': [round(recall\_dt, 3)]

})

# Plotting the table for Decision Tree metrics

fig3, ax3 = plt.subplots(figsize=(10, 2))

# Hide axes

ax3.axis('off')

# Create a table for the metrics

table3 = ax3.table(

cellText=decision\_tree\_metrics.values.tolist(),

colLabels=decision\_tree\_metrics.columns,

cellLoc='center',

loc='center'

)

table3.auto\_set\_font\_size(False)

table3.set\_fontsize(10)

table3.scale(1.2, 1.2)

plt.title('Evaluation Metrics of Decision Tree', fontsize=14, fontweight='bold', pad=20)

plt.tight\_layout()

plt.show()

# Compute confusion matrix for Decision Tree

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_dt)

# Plotting the confusion matrix

fig4, ax4 = plt.subplots(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'], ax=ax4)

plt.title('Confusion Matrix of Decision Tree', fontsize=14, fontweight='bold')

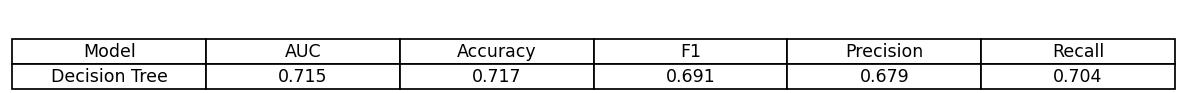
plt.ylabel('Actual')

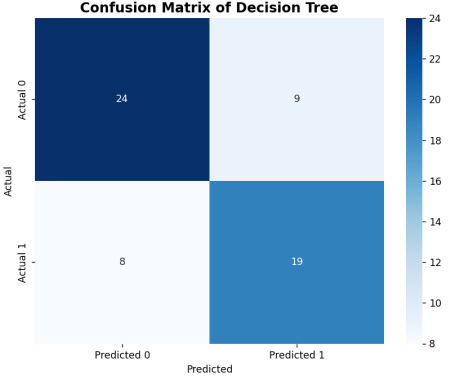
plt.xlabel('Predicted')

plt.tight\_layout()

plt.show()

OUTPUT





17.EVALUATION METRICS AND CONFUSION MATRIX OF RANDOM FOREST

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix

# Load the dataset from specified path

data\_path = 'C:/Users/liyan/Downloads/heart\_data.csv'

data = pd.read\_csv(data\_path)

# Assuming 'diagnosis' is the target variable, and others are features

X = data.drop('diagnosis', axis=1)

y = data['diagnosis']

# Split data into training and testing sets (80% for training, 20% for testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize Random Forest model

random\_forest\_model = RandomForestClassifier(random\_state=42)

# Train the Random Forest model

random\_forest\_model.fit(X\_train, y\_train)

y\_pred\_rf = random\_forest\_model.predict(X\_test)

y\_prob\_rf = random\_forest\_model.predict\_proba(X\_test)[:, 1] # Probability for positive class

# Evaluate metrics for Random Forest

auc\_rf = roc\_auc\_score(y\_test, y\_prob\_rf)

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)

precision\_rf = precision\_score(y\_test, y\_pred\_rf)

recall\_rf = recall\_score(y\_test, y\_pred\_rf)

f1\_rf = f1\_score(y\_test, y\_pred\_rf)

# Create a DataFrame to store Random Forest metrics

random\_forest\_metrics = pd.DataFrame({

'Model': ['Random Forest'],

'AUC': [round(auc\_rf, 3)],

'Accuracy': [round(accuracy\_rf, 3)],

'F1': [round(f1\_rf, 3)],

'Precision': [round(precision\_rf, 3)],

'Recall': [round(recall\_rf, 3)]

})

# Plotting the table for Random Forest metrics

fig2, ax2 = plt.subplots(figsize=(10, 2))

# Hide axes

ax2.axis('off')

# Create a table for the metrics

table2 = ax2.table(

cellText=random\_forest\_metrics.values.tolist(),

colLabels=random\_forest\_metrics.columns,

cellLoc='center',

loc='center'

)

table2.auto\_set\_font\_size(False)

table2.set\_fontsize(10)

table2.scale(1.2, 1.2)

plt.title('Evaluation Metrics of Random Forest', fontsize=14, fontweight='bold', pad=20)

plt.tight\_layout()

plt.show()

# Compute confusion matrix for Random Forest

conf\_matrix\_rf = confusion\_matrix(y\_test, y\_pred\_rf)

# Plotting the confusion matrix

fig3, ax3 = plt.subplots(figsize=(8, 6))

sns.heatmap(conf\_matrix\_rf, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'], ax=ax3)

plt.title('Confusion Matrix of Random Forest', fontsize=14, fontweight='bold')

plt.ylabel('Actual')

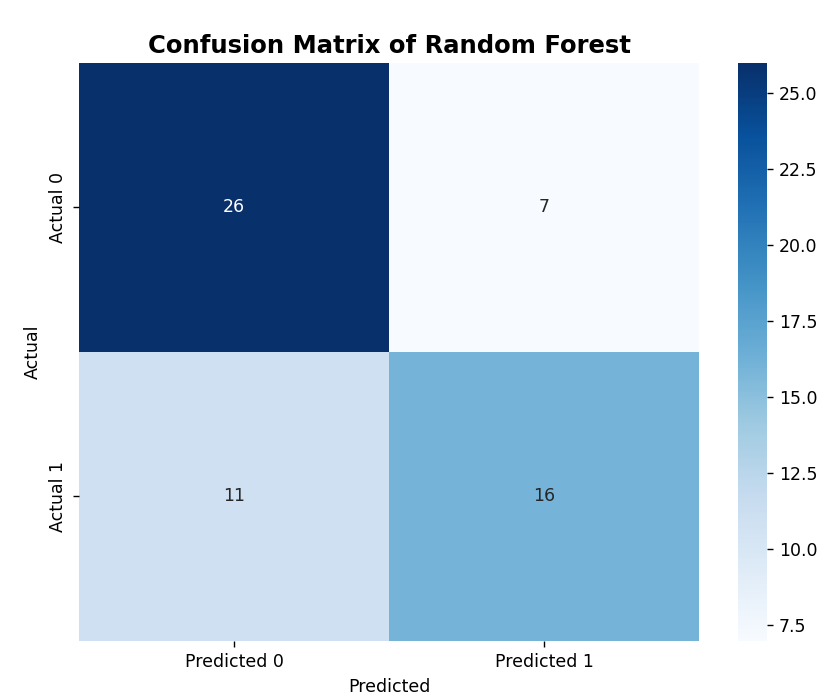
plt.xlabel('Predicted')

plt.tight\_layout()

plt.show()

OUTPUT

Screenshot 2024-06-26 193442.png



18.EVALUATION METRICS AND CONFUSION MATRIX OF GRADIENT BOOSTING

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier, AdaBoostClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix

# Load the dataset from specified path

data\_path = 'C:/Users/liyan/Downloads/heart\_data.csv'

data = pd.read\_csv(data\_path)

# Assuming 'diagnosis' is the target variable, and others are features

X = data.drop('diagnosis', axis=1)

y = data['diagnosis']

# Split data into training and testing sets (80% for training, 20% for testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define classifiers

models = {

'Decision Tree': DecisionTreeClassifier(random\_state=42),

'SVM': SVC(random\_state=42, probability=True),

'Random Forest': RandomForestClassifier(random\_state=42),

'Naive Bayes': GaussianNB(),

'Log Reg': LogisticRegression(random\_state=42, max\_iter=1000),

'AdaBoost': AdaBoostClassifier(random\_state=42),

'Gradient Boosting': GradientBoostingClassifier(random\_state=42)

}

# Initialize lists to store metrics

model\_names = []

aucs = []

accuracies = []

precisions = []

recalls = []

f1\_scores = []

# Train and evaluate each model

for name, model in models.items():

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

y\_prob = model.predict\_proba(X\_test)[:, 1] # Probability for positive class

# Evaluate metrics

auc = roc\_auc\_score(y\_test, y\_prob)

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)

# Store metrics

model\_names.append(name)

aucs.append(round(auc, 3)) # Round to 3 decimal places

accuracies.append(round(accuracy, 3)) # Round to 3 decimal places

precisions.append(round(precision, 3)) # Round to 3 decimal places

recalls.append(round(recall, 3)) # Round to 3 decimal places

f1\_scores.append(round(f1, 3)) # Round to 3 decimal places

# Create a DataFrame to store metrics

metrics\_df = pd.DataFrame({

'Model': model\_names,

'AUC': aucs,

'Accuracy': accuracies,

'F1': f1\_scores,

'Precision': precisions,

'Recall': recalls

})

# Create the first figure (Metrics of Gradient Boosting)

gradient\_boosting\_metrics = metrics\_df[metrics\_df['Model'] == 'Gradient Boosting']

fig1, ax1 = plt.subplots(figsize=(10, 2))

# Hide axes

ax1.axis('off')

# Create a table for the first figure

table1 = ax1.table(

cellText=gradient\_boosting\_metrics.values.tolist(),

colLabels=gradient\_boosting\_metrics.columns,

cellLoc='center',

loc='center'

)

table1.auto\_set\_font\_size(False)

table1.set\_fontsize(10)

table1.scale(1.2, 1.2)

plt.title('Evaluation Metrics of Gradient Boosting', fontsize=14, fontweight='bold', pad=20)

plt.show()

# Create the second figure (Confusion matrix of Gradient Boosting)

gradient\_boosting\_model = models['Gradient Boosting']

y\_pred\_gb = gradient\_boosting\_model.predict(X\_test)

conf\_matrix\_gb = confusion\_matrix(y\_test, y\_pred\_gb)

fig2, ax2 = plt.subplots(figsize=(8, 6))

sns.heatmap(conf\_matrix\_gb, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'], ax=ax2)

plt.title('Confusion Matrix of Gradient Boosting', fontsize=14, fontweight='bold')

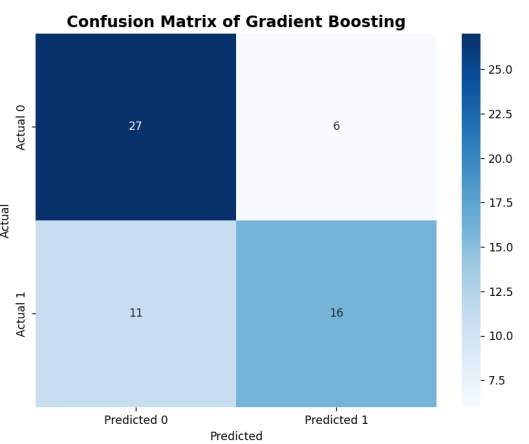
plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.show()

OUTPUT

Screenshot 2024-06-26 194818.png



19.EVALUATION METRICS AND CONFUSION MATRIX OF NAÏVE BAYES

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix

# Load the dataset from specified path

data\_path = 'C:/Users/liyan/Downloads/heart\_data.csv'

data = pd.read\_csv(data\_path)

# Assuming 'diagnosis' is the target variable, and others are features

X = data.drop('diagnosis', axis=1)

y = data['diagnosis']

# Split data 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)

# Initialize Naive Bayes model

naive\_bayes\_model = GaussianNB()

# Train the Naive Bayes model

naive\_bayes\_model.fit(X\_train, y\_train)

y\_pred\_nb = naive\_bayes\_model.predict(X\_test)

y\_prob\_nb = naive\_bayes\_model.predict\_proba(X\_test)[:, 1] # Probability for positive class

# Evaluate metrics for Naive Bayes

auc\_nb = roc\_auc\_score(y\_test, y\_prob\_nb)

accuracy\_nb = accuracy\_score(y\_test, y\_pred\_nb)

precision\_nb = precision\_score(y\_test, y\_pred\_nb)

recall\_nb = recall\_score(y\_test, y\_pred\_nb)

f1\_nb = f1\_score(y\_test, y\_pred\_nb)

# Create a DataFrame to store Naive Bayes metrics

naive\_bayes\_metrics = pd.DataFrame({

'Model': ['Naive Bayes'],

'AUC': [round(auc\_nb, 3)],

'Accuracy': [round(accuracy\_nb, 3)],

'F1': [round(f1\_nb, 3)],

'Precision': [round(precision\_nb, 3)],

'Recall': [round(recall\_nb, 3)]

})

# Plotting the table for Naive Bayes metrics

fig2, ax2 = plt.subplots(figsize=(10, 2))

# Hide axes

ax2.axis('off')

# Create a table for the metrics

table2 = ax2.table(

cellText=naive\_bayes\_metrics.values.tolist(),

colLabels=naive\_bayes\_metrics.columns,

cellLoc='center',

loc='center'

)

table2.auto\_set\_font\_size(False)

table2.set\_fontsize(10)

table2.scale(1.2, 1.2)

plt.title('Evaluation Metrics of Naive Bayes', fontsize=14, fontweight='bold', pad=20)

plt.tight\_layout()

plt.show()

# Compute confusion matrix for Naive Bayes

conf\_matrix\_nb = confusion\_matrix(y\_test, y\_pred\_nb)

# Plotting the confusion matrix

fig3, ax3 = plt.subplots(figsize=(8, 6))

sns.heatmap(conf\_matrix\_nb, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'], ax=ax3)

plt.title('Confusion Matrix of Naive Bayes', fontsize=14, fontweight='bold')

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.tight\_layout()

plt.show()

OUTPUT

Screenshot 2024-06-26 200130.png

