Importing the Dependencies

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

Data collection and Processing

```
 #loading the csv data to Pandas
 heart_data = pd.read_csv('heart_disease_dataset.csv')
```

```
#print first 5 rows
heart_data.head()
```

	Age	Gender	Cholesterol	Blood Pressure	Heart Rate	Smoking	Alcohol Intake	Exercise Hours	Family History	Dia
0	75	Female	228	119	66	Current	Heavy	1	No	
1	48	Male	204	165	62	Current	NaN	5	No	
2	53	Male	234	91	67	Never	Heavy	3	Yes	
3	69	Female	192	90	72	Current	NaN	4	No	
4	62	Female	172	163	93	Never	NaN	6	No	

```
#print last 5 rows
heart_data.tail()
```

	Age	Gender	Cholesterol	Blood Pressure	Heart Rate	Smoking	Alcohol Intake	Exercise Hours	Family History
995	56	Female	269	111	86	Never	Heavy	5	No
996	78	Female	334	145	76	Never	NaN	6	No
997	79	Male	151	179	81	Never	Moderate	4	Yes
998	60	Female	326	151	68	Former	NaN	8	Yes
999	53	Male	226	116	82	Current	NaN	6	No

```
# no of rows and columns in the dataset
heart_data.shape
```

```
(1000, 16)
```

```
# getting some info about the data
heart_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Age                      1000 non-null   int64 
 1   Gender                   1000 non-null   object
 2   Cholesterol              1000 non-null   int64 
 3   Blood Pressure           1000 non-null   int64 
 4   Heart Rate               1000 non-null   int64 
 5   Smoking                  1000 non-null   object
 6   Alcohol Intake           660 non-null    object
 7   Exercise Hours           1000 non-null   int64 
 8   Family History           1000 non-null   object
 9   Diabetes                 1000 non-null   object
 10  Obesity                  1000 non-null   object
 11  Stress Level             1000 non-null   int64 
 12  Blood Sugar              1000 non-null   int64 
 13  Exercise Induced Angina  1000 non-null   object
 14  Chest Pain Type          1000 non-null   object
 15  Heart Disease            1000 non-null   int64 
dtypes: int64(8), object(8)
memory usage: 125.1+ KB
```

```
# checking for missing values
heart_data.isnull().sum()
```

	0
Age	0
Gender	0
Cholesterol	0
Blood Pressure	0
Heart Rate	0
Smoking	0
Alcohol Intake	340
Exercise Hours	0
Family History	0
Diabetes	0
Obesity	0
Stress Level	0
Blood Sugar	0
Exercise Induced Angina	0
Chest Pain Type	0
Heart Disease	0

dtype: int64

```
# statistical measures about the data
heart_data.describe()
```

	Age	Cholesterol	Blood Pressure	Heart Rate	Exercise Hours	Stress Level	
count	1000.000000	1000.000000	1000.0000	1000.000000	1000.000000	1000.000000	1000
mean	52.293000	249.939000	135.2810	79.204000	4.529000	5.646000	134
std	15.727126	57.914673	26.3883	11.486092	2.934241	2.831024	36
min	25.000000	150.000000	90.0000	60.000000	0.000000	1.000000	70
25%	39.000000	200.000000	112.7500	70.000000	2.000000	3.000000	104
50%	52.000000	248.000000	136.0000	79.000000	4.500000	6.000000	135
75%	66.000000	299.000000	159.0000	89.000000	7.000000	8.000000	167
max	79.000000	349.000000	179.0000	99.000000	9.000000	10.000000	199

```
# checking the distribution of the target value
heart_data['Heart Disease'].value_counts()
```

Heart Disease	count
0	608
1	392

dtype: int64

```
# prompt: check for duplicate values from the code

from google.colab import drive
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Importing the Dependencies
drive.mount('/content/drive')

# Data collection and Processing
# loading the csv data to Pandas
heart_data = pd.read_csv('heart_disease_dataset.csv')

# print first 5 rows
print(heart_data.head())

# print last 5 rows
print(heart_data.tail())

# number of rows and columns in the dataset
print(heart_data.shape)

# getting some info about the data
print(heart_data.info())

# checking for missing values
print(heart_data.isnull().sum())

# statistical measures about the data
print(heart_data.describe())

# checking the distribution of the target value
print(heart_data['Heart Disease'].value_counts())

# Check for duplicate rows
duplicate_rows = heart_data[heart_data.duplicated()]
print("Number of duplicate rows:", len(duplicate_rows))

# Display duplicate rows (optional)
duplicate_rows
```

```
# Remove duplicate rows (optional - uncomment to remove)
# heart_data = heart_data.drop_duplicates()
# print("Number of rows after removing duplicates:", len(heart_data))
```

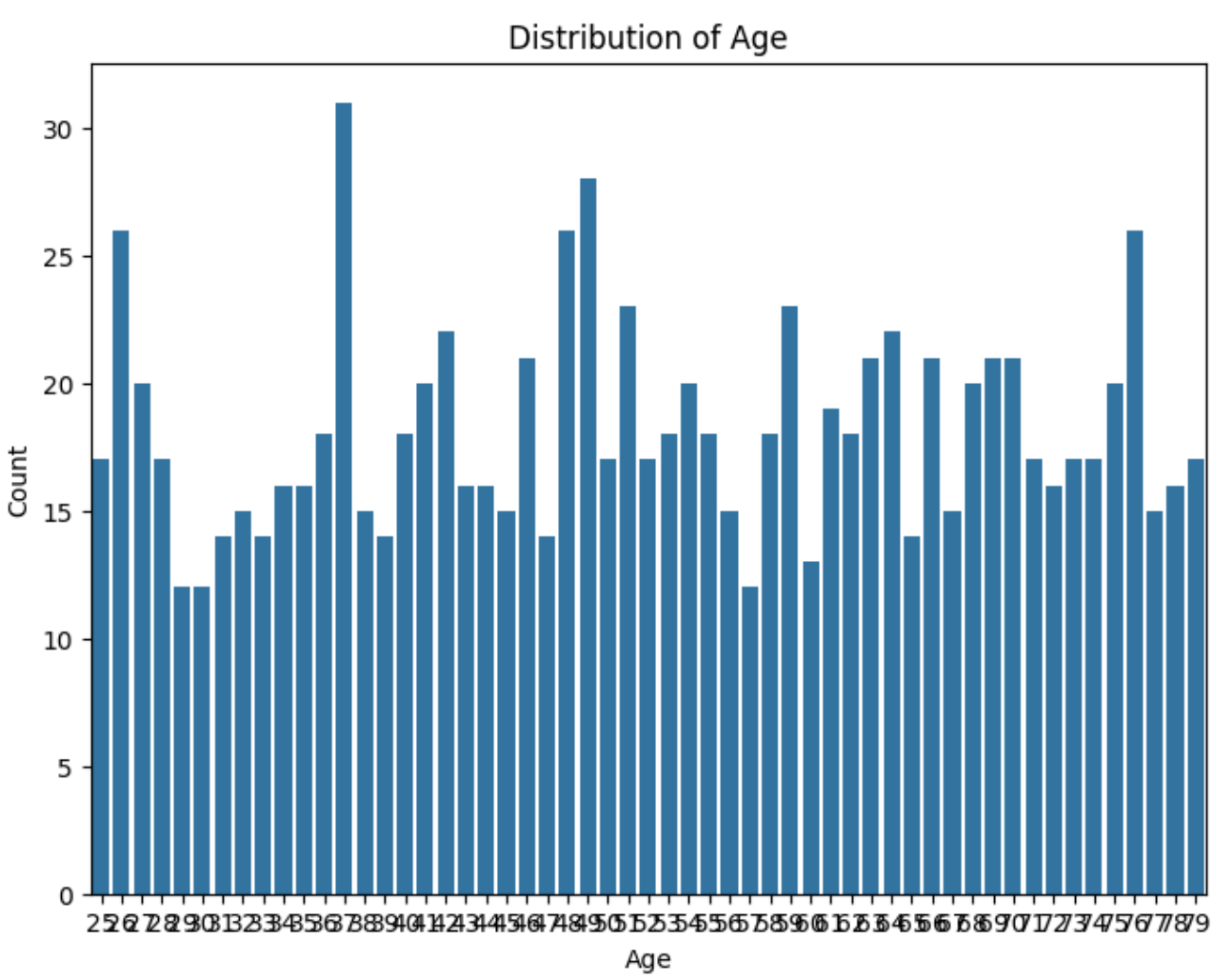
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive
   Age  Gender  Cholesterol  Blood Pressure  Heart Rate  Smoking  \
0   75  Female          228             119          66  Current
1   48    Male          204             165          62  Current
2   53    Male          234              91          67    Never
3   69  Female          192              90          72  Current
4   62  Female          172             163          93    Never

  Alcohol Intake  Exercise Hours Family History Diabetes Obesity  \
0          Heavy               1             No       No     Yes
1            NaN               5             No       No      No
2          Heavy               3            Yes       No     Yes
3            NaN               4             No      Yes      No
4            NaN               6             No      Yes      No

   Stress Level  Blood Sugar Exercise Induced Angina   Chest Pain Type  \
0             8          119                     Yes   Atypical Angina
1             9           70                     Yes    Typical Angina
2             5          196                     Yes   Atypical Angina
3             7          107                     Yes  Non-anginal Pain
4             2          183                     Yes      Asymptomatic

   Heart Disease
0              1
1              0
2              1
3              0
4              0
     Age  Gender  Cholesterol  Blood Pressure  Heart Rate  Smoking  \
995   56  Female          269             111          86    Never
996   78  Female          334             145          76    Never
997   79    Male          151             179          81    Never
998   60  Female          326             151          68   Former
999   53    Male          226             116          82  Current

    Alcohol Intake  Exercise Hours Family History Diabetes Obesity  \
995          Heavy               5             No      Yes     Yes
996            NaN               6             No       No      No
997       Moderate               4            Yes       No     Yes
998            NaN               8            Yes      Yes      No
999            NaN               6             No       No     Yes

     Stress Level  Blood Sugar Exercise Induced Angina   Chest Pain Type  \
995            10          120                     No  Non-anginal Pain
996            10          196                    Yes    Typical Angina
997             8          189                    Yes      Asymptomatic
998             5          174                    Yes   Atypical Angina
999             5          161                    Yes      Asymptomatic

     Heart Disease
995              1
996              1
997              0
998              1
999              1
(1000, 16)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
```

```
---  ------                    --------------  -----
 0   Age                       1000 non-null   int64
 1   Gender                    1000 non-null   object
 2   Cholesterol               1000 non-null   int64
 3   Blood Pressure            1000 non-null   int64
 4   Heart Rate                1000 non-null   int64
 5   Smoking                   1000 non-null   object
 6   Alcohol Intake            660 non-null    object
 7   Exercise Hours            1000 non-null   int64
 8   Family History            1000 non-null   object
 9   Diabetes                  1000 non-null   object
 10  Obesity                   1000 non-null   object
 11  Stress Level              1000 non-null   int64
 12  Blood Sugar               1000 non-null   int64
 13  Exercise Induced Angina   1000 non-null   object
 14  Chest Pain Type           1000 non-null   object
 15  Heart Disease             1000 non-null   int64
dtypes: int64(8), object(8)
memory usage: 125.1+ KB
None
Age                          0
Gender                       0
Cholesterol                  0
Blood Pressure               0
Heart Rate                   0
Smoking                      0
Alcohol Intake             340
Exercise Hours               0
Family History               0
Diabetes                     0
Obesity                      0
Stress Level                 0
Blood Sugar                  0
Exercise Induced Angina      0
Chest Pain Type              0
Heart Disease                0
dtype: int64
                Age  Cholesterol  Blood Pressure   Heart Rate  Exercise Hours  \
count  1000.000000  1000.000000       1000.0000  1000.000000     1000.000000
mean     52.293000   249.939000        135.2810    79.204000        4.529000
std      15.727126    57.914673         26.3883    11.486092        2.934241
min      25.000000   150.000000         90.0000    60.000000        0.000000
25%      39.000000   200.000000        112.7500    70.000000        2.000000
50%      52.000000   248.000000        136.0000    79.000000        4.500000
75%      66.000000   299.000000        159.0000    89.000000        7.000000
max      79.000000   349.000000        179.0000    99.000000        9.000000

       Stress Level  Blood Sugar  Heart Disease
count   1000.000000  1000.000000    1000.000000
mean       5.646000   134.941000       0.392000
std        2.831024    36.699624       0.488441
min        1.000000    70.000000       0.000000
25%        3.000000   104.000000       0.000000
50%        6.000000   135.000000       0.000000
75%        8.000000   167.000000       1.000000
max       10.000000   199.000000       1.000000
Heart Disease
0    608
1    392
Name: count, dtype: int64
```

Number of duplicate rows: 0

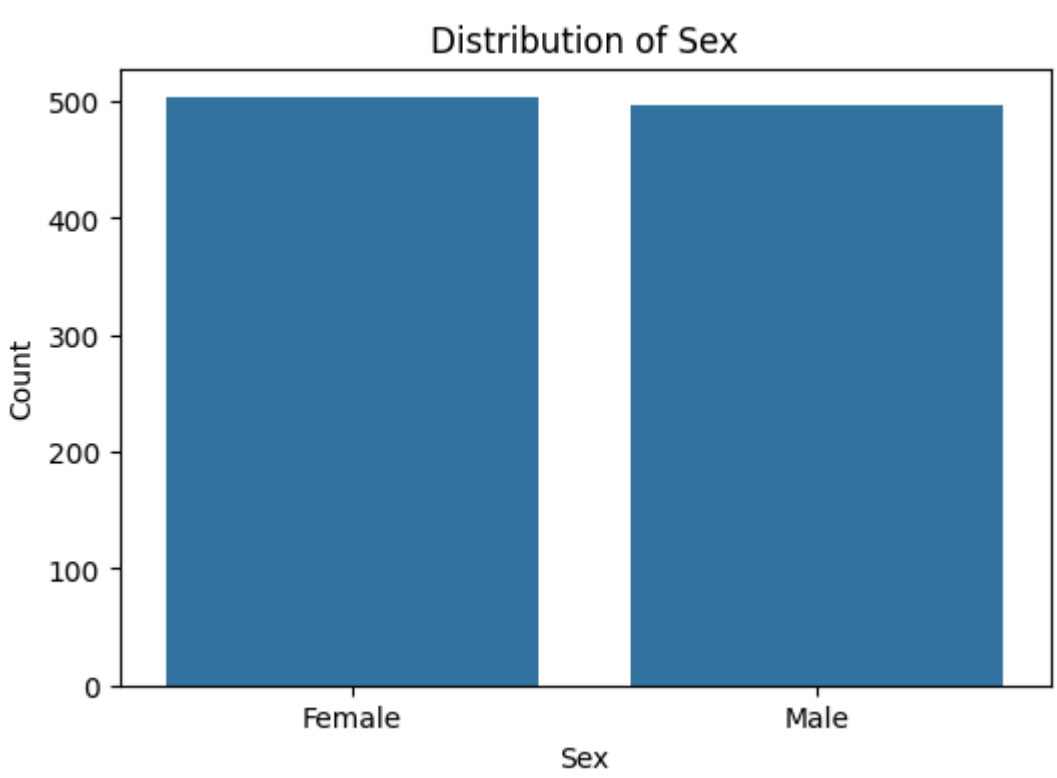
	Age	Gender	Cholesterol	Blood Pressure	Heart Rate	Smoking	Alcohol Intake	Exercise Hours	Family History	D

```
# prompt: create barchart for age in the dataset

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'heart_data' DataFrame is already loaded as in your provided code

plt.figure(figsize=(8, 6))
sns.countplot(x='Age', data=heart_data)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```

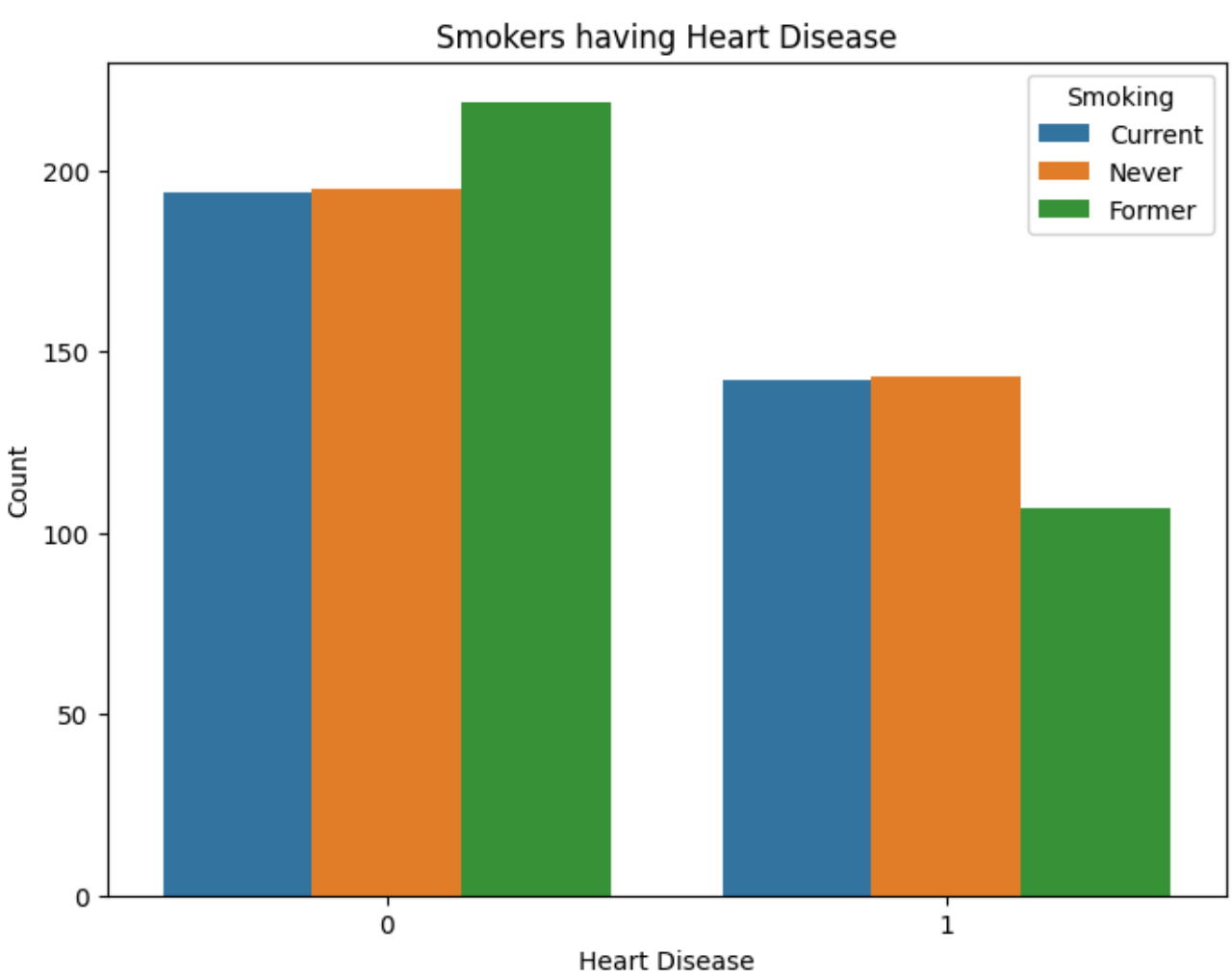



```
# prompt: create barchart for sex in dataset

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'heart_data' DataFrame is already loaded as in your provided code

# Assuming 'Sex' column exists. If not, replace 'Sex' with the actual column name.
# Changed 'Sex' to 'Gender' based on the global variables
plt.figure(figsize=(6, 4))
```

```
sns.countplot(x='Gender', data=heart_data) # Using 'Gender' instead of 'Sex'
plt.title('Distribution of Sex')
plt.xlabel('Sex')
plt.ylabel('Count')
plt.show()
```

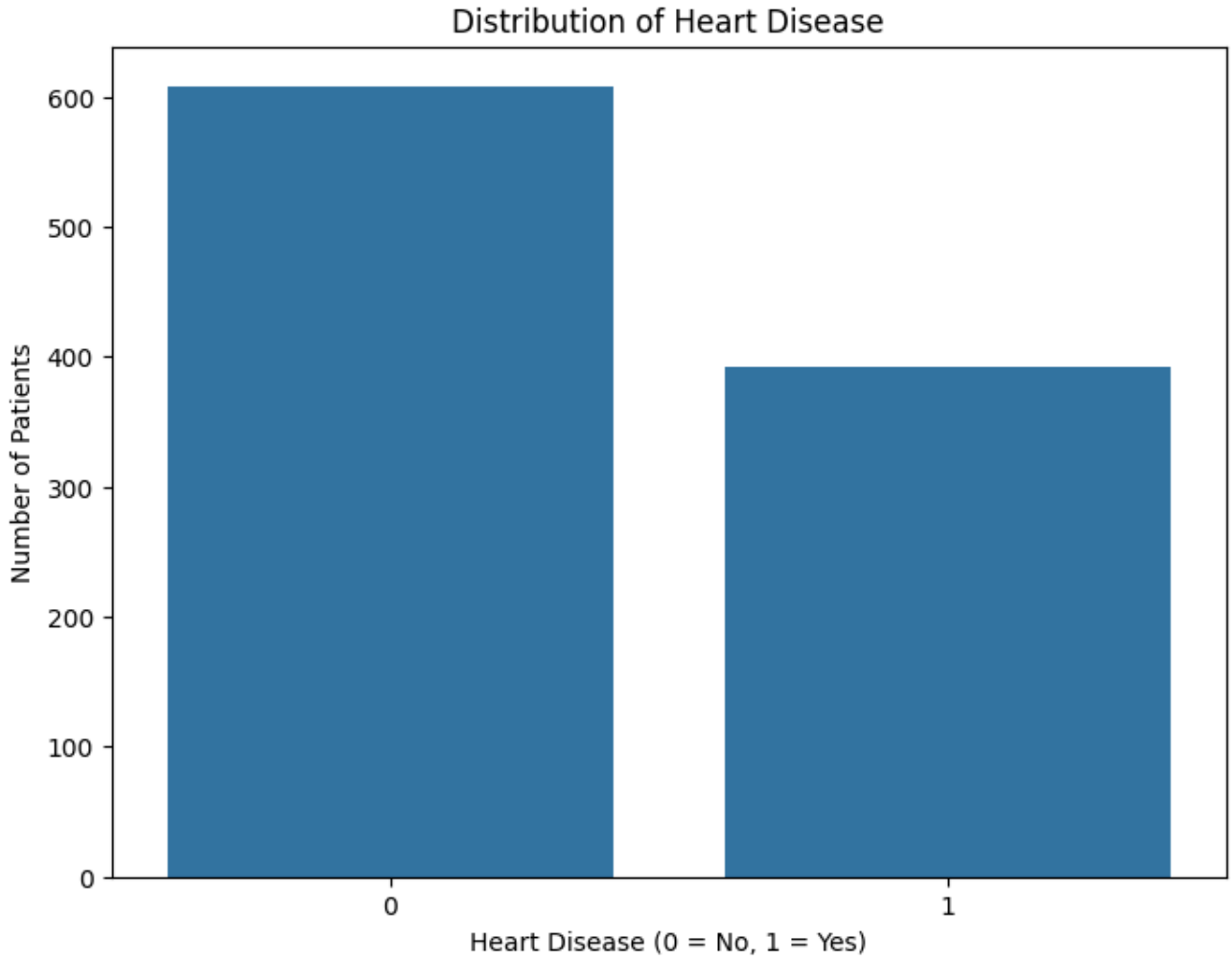



```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'heart_data' DataFrame is already loaded as in your provided code

plt.figure(figsize=(8, 6))
# Replace 'Smoker' with the actual column name in your dataset, for example, 'Smoking
sns.countplot(x='Heart Disease', hue='Smoking', data=heart_data)
plt.title('Smokers having Heart Disease')
plt.xlabel('Heart Disease')
plt.ylabel('Count')
plt.show()
```



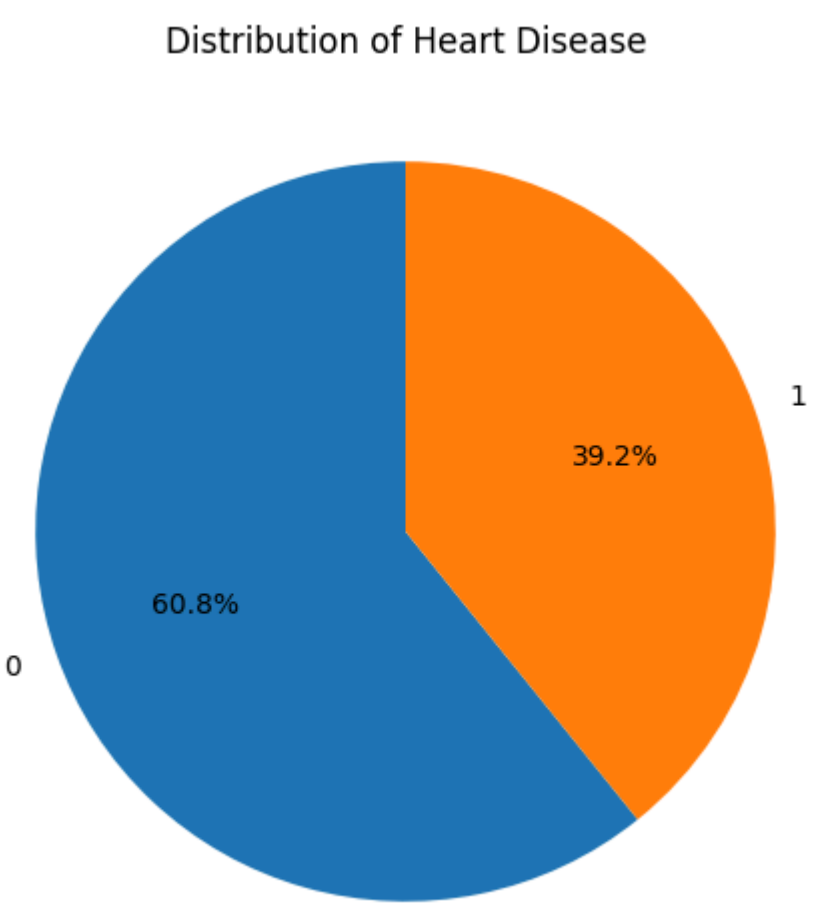
Bar chart for visualizing the person does not have heart disease and person having heart disease

```
# prompt: create an bar chart for visualizing the person does not have heart disease

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'heart_data' DataFrame is already loaded as in your previous code

# Count the occurrences of each target value
heart_disease_counts = heart_data['Heart Disease'].value_counts()

# Create a bar chart
plt.figure(figsize=(8, 6))
sns.barplot(x=heart_disease_counts.index, y=heart_disease_counts.values)
plt.title('Distribution of Heart Disease')
plt.xlabel('Heart Disease (0 = No, 1 = Yes)')
plt.ylabel('Number of Patients')
plt.show()
```



create an pie chart for visualizing the person does not have heart disease and person having heart disease

```
# prompt: create an pie chart for visualizing the person does not have heart disease

# Assuming 'heart_data' DataFrame is already loaded as in your previous code

# Count the occurrences of each target value
heart_disease_counts = heart_data['Heart Disease'].value_counts()

# Create a pie chart
plt.figure(figsize=(8, 6))
plt.pie(heart_disease_counts, labels=heart_disease_counts.index, autopct='%1.1f%%', s
plt.title('Distribution of Heart Disease')
plt.show()
```



Start coding or generate with AI.

1--> Defective heart 0--> Healthy heart

Splitting the features and data

```
X = heart_data.drop(columns='Heart Disease', axis=1)
Y = heart_data['Heart Disease']
```

```
print(X)
```

```
      Age  Gender  Cholesterol  Blood Pressure  Heart Rate  Smoking  \
0      75  Female          228             119          66  Current
1      48    Male          204             165          62  Current
2      53    Male          234              91          67    Never
3      69  Female          192              90          72  Current
4      62  Female          172             163          93    Never
..    ...     ...          ...             ...         ...      ...
995    56  Female          269             111          86    Never
996    78  Female          334             145          76    Never
997    79    Male          151             179          81    Never
998    60  Female          326             151          68   Former
999    53    Male          226             116          82  Current
```

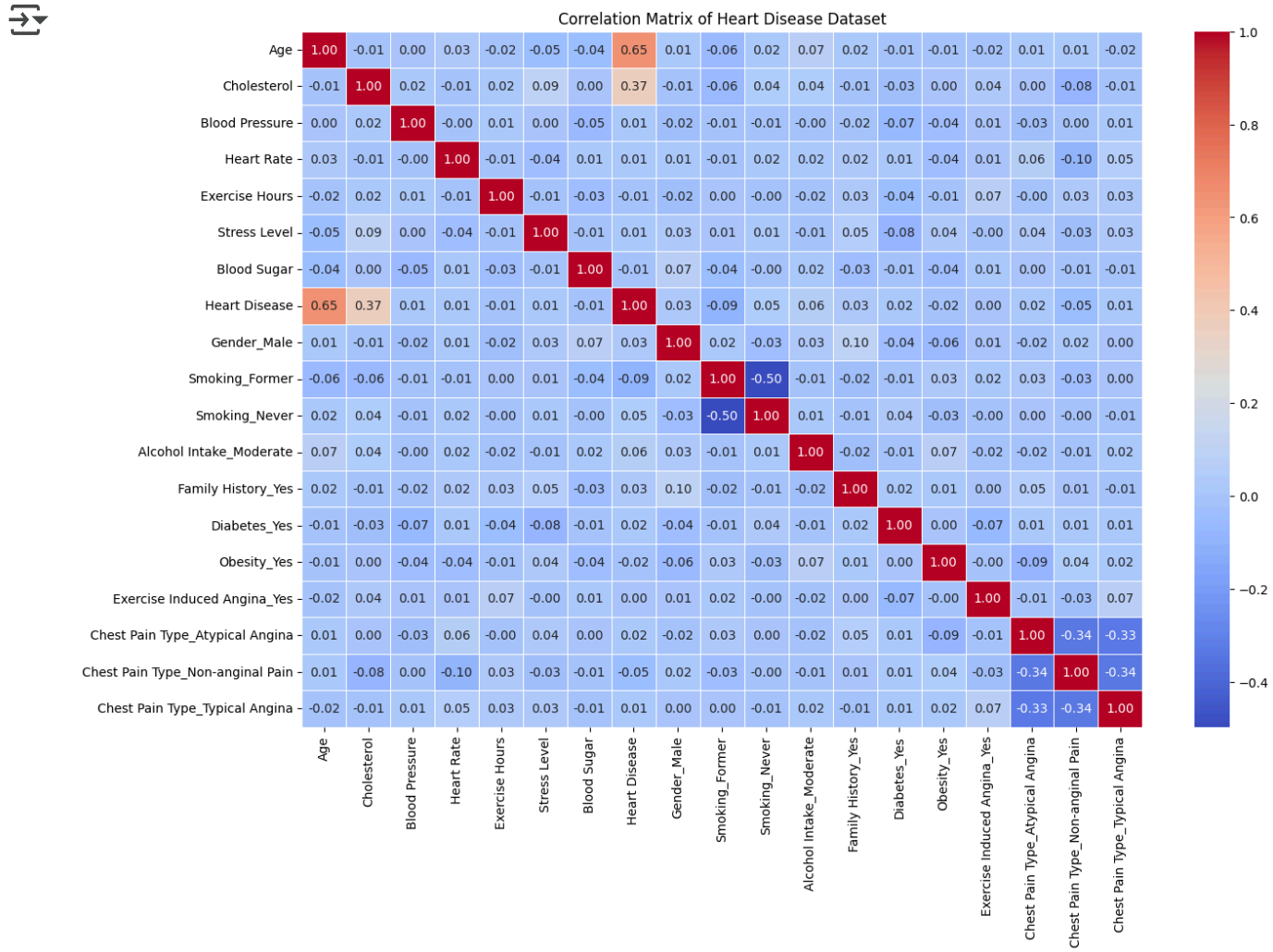
```
     Alcohol Intake  Exercise Hours Family History Diabetes Obesity  \
0             Heavy               1             No       No     Yes
1               NaN               5             No       No      No
2             Heavy               3            Yes       No     Yes
3               NaN               4             No      Yes      No
4               NaN               6             No      Yes      No
..              ...             ...            ...      ...     ...
995           Heavy               5             No      Yes     Yes
996             NaN               6             No       No      No
997        Moderate               4            Yes       No     Yes
998             NaN               8            Yes      Yes      No
999             NaN               6             No       No     Yes

     Stress Level  Blood Sugar Exercise Induced Angina   Chest Pain Type
0               8          119                      Yes   Atypical Angina
1               9           70                      Yes    Typical Angina
2               5          196                      Yes   Atypical Angina
3               7          107                      Yes  Non-anginal Pain
4               2          183                      Yes      Asymptomatic
..            ...          ...                      ...               ...
995            10          120                       No  Non-anginal Pain
996            10          196                      Yes    Typical Angina
997             8          189                      Yes      Asymptomatic
998             5          174                      Yes   Atypical Angina
999             5          161                      Yes      Asymptomatic

[1000 rows x 15 columns]
```

```
print(Y)
```

```
0      1
1      0
2      1
3      0
4      0
      ..
995    1
996    1
997    0
998    1
999    1
Name: Heart Disease, Length: 1000, dtype: int64
```

```
# prompt: create an heatmap for the given data

# Importing necessary libraries
import pandas as pd  # Import pandas for data manipulation
import matplotlib.pyplot as plt  # Import matplotlib for plotting
import seaborn as sns  # Import seaborn for heatmap visualization

# Assuming 'heart_data' DataFrame is already loaded as in your previous code
# and X, Y are defined as in the previous code.

# Load the dataset (if not already loaded)
heart_data = pd.read_csv('heart_disease_dataset.csv')  # Load the data into heart_dat

# Convert categorical columns to numerical using one-hot encoding
```

```
# Select categorical features (object type)
categorical_features = heart_data.select_dtypes(include=['object']).columns
# Perform one-hot encoding
heart_data_encoded = pd.get_dummies(heart_data, columns=categorical_features, drop_fi

# Calculate the correlation matrix using the encoded data
correlation_matrix = heart_data_encoded.corr()

# Create a heatmap
plt.figure(figsize=(15, 10))  # Adjust figure size as needed
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.
plt.title("Correlation Matrix of Heart Disease Dataset")
plt.show()
```



Splitting the data into Training data and Testing data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, rand
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(1000, 15) (800, 15) (200, 15)
```

Model Training

Logistic Regression

```
model = LogisticRegression()
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
# Assuming 'heart_data' is your original DataFrame

# ... (Your previous code for loading and inspecting the data) ...

# Splitting the features and data
X = heart_data.drop(columns='Heart Disease', axis=1)
Y = heart_data['Heart Disease']

# Get the actual column names from the DataFrame
print(X.columns)

# Perform one-hot encoding on categorical features
# Update the column names to reflect the actual column names
categorical_features = X.select_dtypes(include=['object']).columns.tolist() #Selects
# Alternatively, manually adjust the column names to match those printed above
# categorical_features = ['Sex', 'ChestPainType', 'FastingBS', 'RestingECG', 'Exercis
print(categorical_features)
X = pd.get_dummies(X, columns=categorical_features, drop_first=True)  # drop_first to

# Splitting the data into Training data and Testing data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y,

# Model Training
model = LogisticRegression()

# training the logistic regression model with training data
model.fit(X_train, Y_train)
```

```
Index(['Age', 'Gender', 'Cholesterol', 'Blood Pressure', 'Heart Rate',
       'Smoking', 'Alcohol Intake', 'Exercise Hours', 'Family History',
       'Diabetes', 'Obesity', 'Stress Level', 'Blood Sugar',
       'Exercise Induced Angina', 'Chest Pain Type'],
      dtype='object')
['Gender', 'Smoking', 'Alcohol Intake', 'Family History', 'Diabetes', 'Obesity', 'Exe
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

LogisticRegression

```
LogisticRegression()
```

Model Evalution

Accuracy Score

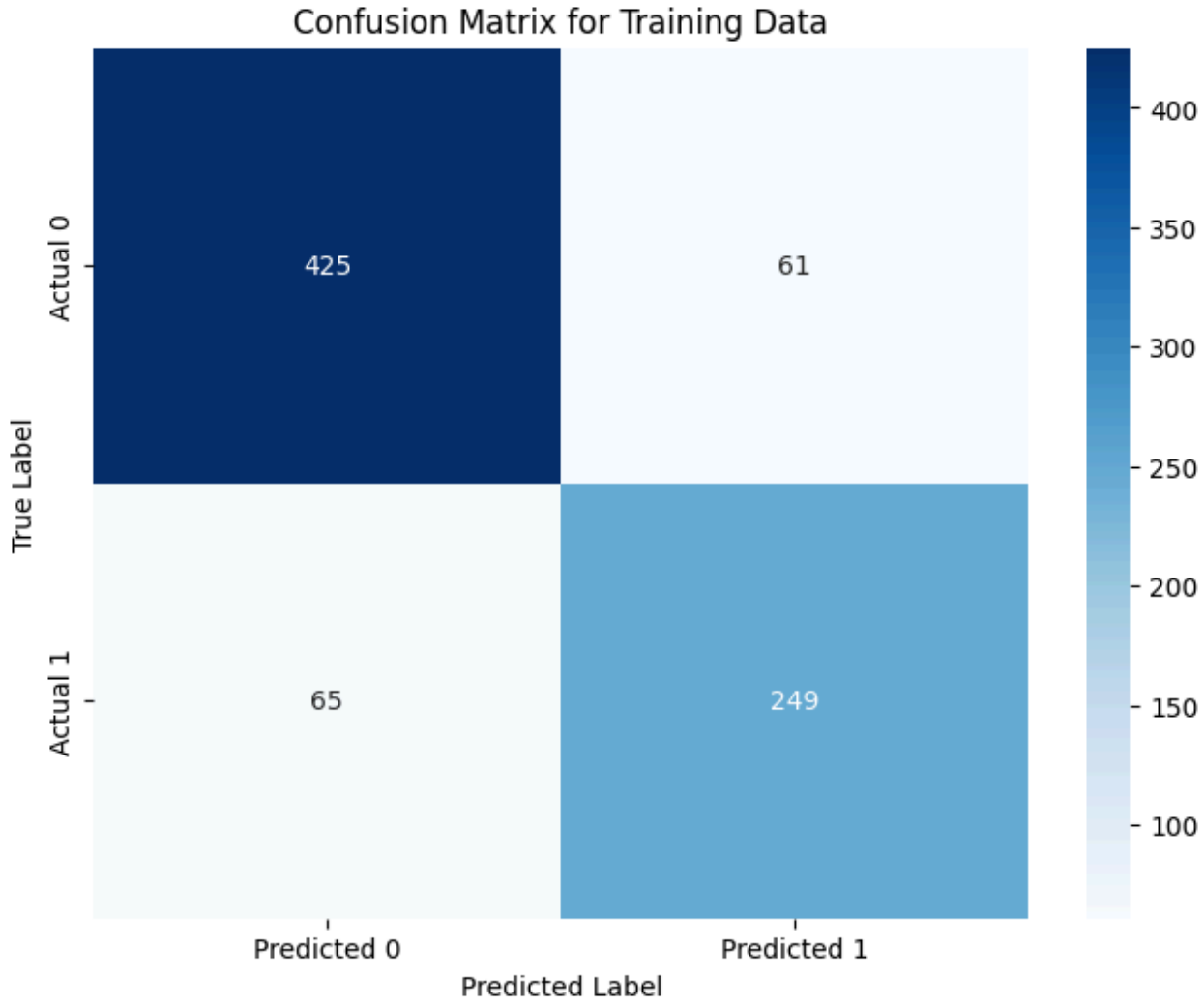
```
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)


print('Accuracy on Training data : ', training_data_accuracy)


# accuracy on training data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)


print('Accuracy on Test data : ', test_data_accuracy)
```

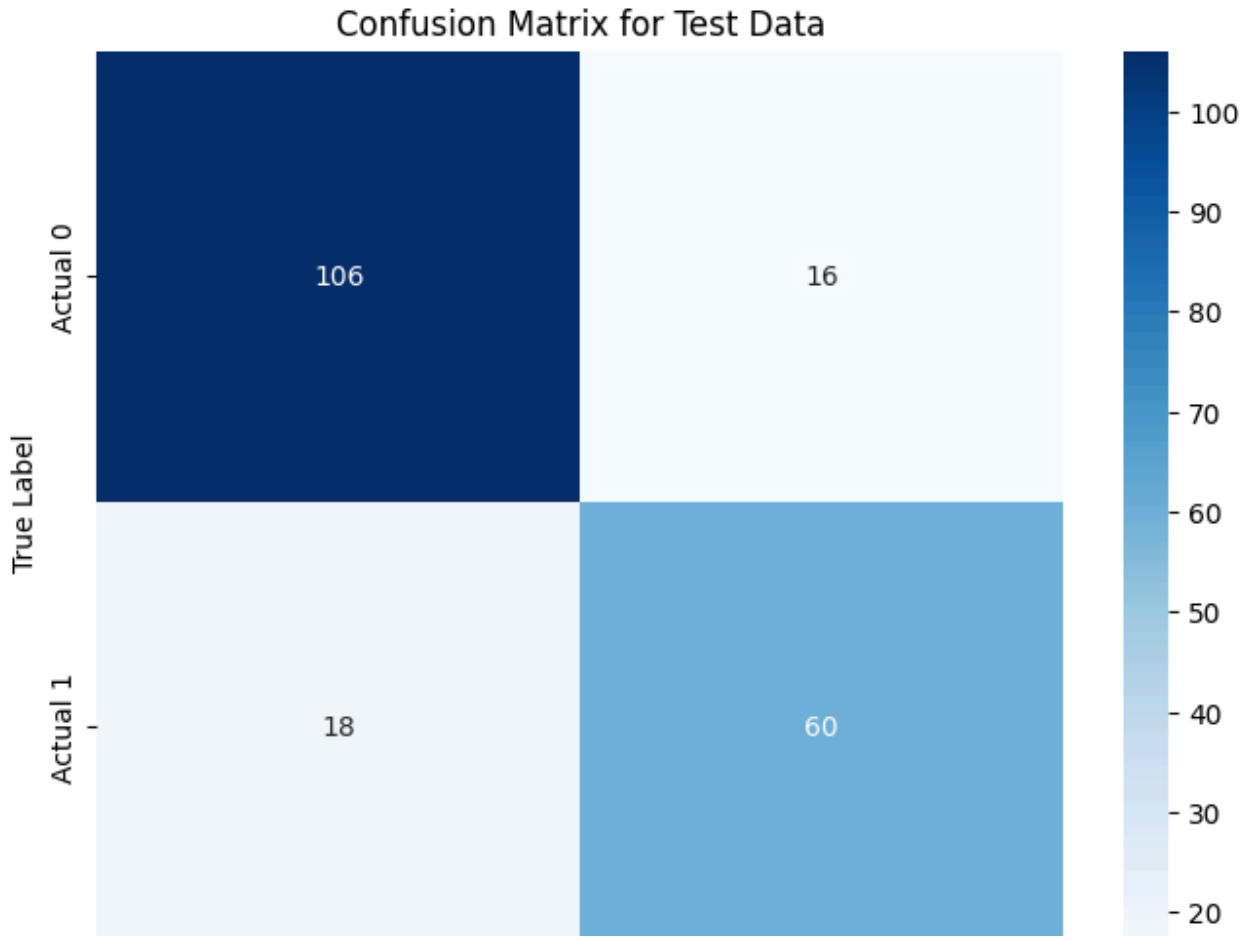
```
Accuracy on Test data :  0.845
```

```
# prompt: create confusion create for all the methods used in the program

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# ... (Your existing code) ...

# Model Evalution
# ### Accuracy Score
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on Training data : ', training_data_accuracy)

# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy on Test data : ', test_data_accuracy)

# Confusion Matrix for Training Data
cm_train = confusion_matrix(Y_train, X_train_prediction)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_train, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix for Training Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# Confusion Matrix for Test Data
cm_test = confusion_matrix(Y_test, X_test_prediction)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_test, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix for Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
Accuracy on Training data :  0.8425
Accuracy on Test data :  0.83
```





Building Predictive System

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Assuming 'heart_data' is your original DataFrame and you've loaded it

# ... (Your previous code for loading and inspecting the data) ...

# Get the actual column names from the DataFrame
# ... (Your code for splitting and printing X.columns) ...

# Perform one-hot encoding on categorical features during training
# ... (Your code for one-hot encoding during training) ...


### Building Predictive System

input_data = (50,'Male',308,166,97,'Current','None',7,'No','No','No',2,116,'Yes','Typical

# Convert input_data to DataFrame for one-hot encoding
# creates a single row dataframe with the same columns as the ORIGINAL heart_data
input_data_df = pd.DataFrame([input_data], columns=heart_data.columns[:-1])
#print(input_data_df) # check what the columns are

# Perform one-hot encoding on the input data using the same columns as in training data
input_data_encoded = pd.get_dummies(input_data_df, columns=categorical_features, drop_fir

# Ensure that the input data has the same columns as the training data after encoding
missing_cols = set(X_train.columns) - set(input_data_encoded.columns) #checks if any colu
for c in missing_cols:
    input_data_encoded[c] = 0 #Add any missing column and set its value to 0 to ensure co
input_data_encoded = input_data_encoded[X_train.columns] # Reorder columns to match train

# Convert the encoded DataFrame back to a NumPy array for prediction
input_data_reshaped = input_data_encoded.to_numpy()


prediction = model.predict(input_data_reshaped)
print(prediction)


if (prediction[0]== 0):
  print('The Person does not have a Heart Disease')
```

```
else:
  print('The Person has Heart Disease')
```

```
[0]
The Person does not have a Heart Disease
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning
  warnings.warn(
```

SVM METHOD

```
# prompt: produce prediction by using support vector machine

from sklearn.svm import SVC

# ... (Your existing code for data loading, preprocessing, and one-hot encoding) ...

# Model Training using Support Vector Machine
model = SVC(kernel='linear')  # You can experiment with different kernels (e.g., 'rbf', '
model.fit(X_train, Y_train)

# Model Evaluation
# ... (Your existing code for model evaluation using accuracy_score) ...

# Building Predictive System
# ... (Your existing code for creating input_data and encoding it) ...

prediction = model.predict(input_data_reshaped)
print(prediction)

if (prediction[0]== 0):
  print('The Person does not have a Heart Disease')
else:
  print('The Person has Heart Disease')
```

```
[0]
The Person does not have a Heart Disease
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning
  warnings.warn(
```

Accuracy for this svm method

```
# prompt: accuracy for this svm method

# ... (Your existing code for data loading, preprocessing, and one-hot encoding) ...

# Model Training using Support Vector Machine
model = SVC(kernel='linear')  # You can experiment with different kernels (e.g., 'rbf', '
model.fit(X_train, Y_train)
```

```
# Model Evaluation
# Accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on Training data : ', training_data_accuracy)

# Accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy on Test data : ', test_data_accuracy)
# Building Predictive System
# ... (Your existing code for creating input_data and encoding it) ...

prediction = model.predict(input_data_reshaped)
print(prediction)

if (prediction[0]== 0):
  print('The Person does not have a Heart Disease')
else:
  print('The Person has Heart Disease')
```

```
Accuracy on Training data :  0.85875
Accuracy on Test data :  0.885
[0]
The Person does not have a Heart Disease
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning
  warnings.warn(
```

create prediction in random forest method with prediction accuracy

```
# prompt: create prediction in random forest method with prediction accuracy

from sklearn.ensemble import RandomForestClassifier

# ... (Your existing code for data loading, preprocessing, and one-hot encoding) ...

# Model Training using Random Forest
model = RandomForestClassifier(random_state=42)  # You can adjust hyperparameters
model.fit(X_train, Y_train)

# Model Evaluation
# Accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on Training data : ', training_data_accuracy)

# Accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy on Test data : ', test_data_accuracy)
```

```
# Building Predictive System
# ... (Your existing code for creating input_data and encoding it) ...

prediction = model.predict(input_data_reshaped)
print(prediction)

if (prediction[0]== 0):
  print('The Person does not have a Heart Disease')
else:
  print('The Person has Heart Disease')
```

```
Accuracy on Training data :  1.0
Accuracy on Test data :  1.0
[0]
The Person does not have a Heart Disease
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning
  warnings.warn(
```

```
# prompt: create accuracy,precision, recall, r2 score for the random forest method train

# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, r2_score, accuracy_score
import pandas as pd
from sklearn.model_selection import train_test_split

# ... (Your existing code for data loading, preprocessing, and model training) ...
# Assuming 'heart_data' is your original DataFrame and you've loaded it

# ... (Your previous code for loading and inspecting the data) ...

# Splitting the features and data
X = heart_data.drop(columns='Heart Disease', axis=1)
Y = heart_data['Heart Disease']

# Perform one-hot encoding on categorical features
categorical_features = X.select_dtypes(include=['object']).columns.tolist()
X = pd.get_dummies(X, columns=categorical_features, drop_first=True)

# Splitting the data into Training data and Testing data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, rand

# Model Training using Random Forest
model = RandomForestClassifier(random_state=42)
model.fit(X_train, Y_train)

# Predictions
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Evaluate the model
print("Random Forest Performance:")
```

```
print("Train Set:")
print(f"Accuracy: {accuracy_score(Y_train, y_train_pred)}")
print(f"Precision: {precision_score(Y_train, y_train_pred)}")
print(f"Recall: {recall_score(Y_train, y_train_pred)}")
print(f"R2 Score: {r2_score(Y_train, y_train_pred)}") # Note: R2 score might not be ideal

print("\nTest Set:")
print(f"Accuracy: {accuracy_score(Y_test, y_test_pred)}")
print(f"Precision: {precision_score(Y_test, y_test_pred)}")
print(f"Recall: {recall_score(Y_test, y_test_pred)}")
print(f"R2 Score: {r2_score(Y_test, y_test_pred)}") # Note: R2 score might not be ideal f
```

```
Random Forest Performance:
Train Set:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
R2 Score: 1.0

Test Set:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
R2 Score: 1.0
```

create prediction in knn method with prediction accuracy

```
# prompt: create prediction in knn method with prediction accuracy

from sklearn.neighbors import KNeighborsClassifier

# ... (Your existing code for data loading, preprocessing, and one-hot encoding) ...

# Model Training using K-Nearest Neighbors
model = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of neighbors
model.fit(X_train, Y_train)

# Model Evaluation
# Accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on Training data : ', training_data_accuracy)

# Accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy on Test data : ', test_data_accuracy)

# Building Predictive System
# ... (Your existing code for creating input_data and encoding it) ...

prediction = model.predict(input_data_reshaped)
print(prediction)
```

```
if (prediction[0]== 0):
  print('The Person does not have a Heart Disease')
else:
  print('The Person has Heart Disease')
```

```
Accuracy on Training data :  0.95125
Accuracy on Test data :  0.92
[0]
The Person does not have a Heart Disease
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning
  warnings.warn(
```

create prediction in gradient boosting method with prediction accuracy

```
# prompt: create prediction in gradient boosting method with prediction accuracy

from sklearn.ensemble import GradientBoostingClassifier

# ... (Your existing code for data loading, preprocessing, and one-hot encoding) ...

# Model Training using Gradient Boosting
model = GradientBoostingClassifier(random_state=42)  # You can adjust hyperparameters
model.fit(X_train, Y_train)

# Model Evaluation
# Accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on Training data : ', training_data_accuracy)

# Accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy on Test data : ', test_data_accuracy)

# Building Predictive System
# ... (Your existing code for creating input_data and encoding it) ...

prediction = model.predict(input_data_reshaped)
print(prediction)

if (prediction[0]== 0):
  print('The Person does not have a Heart Disease')
else:
  print('The Person has Heart Disease')
```

```
Accuracy on Training data :  1.0
Accuracy on Test data :  1.0
[0]
The Person does not have a Heart Disease
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning
```

```
      warnings.warn(
```

create prediction in naive bayes method with prediction accuracy

```
# prompt: create prediction in naive bayes method with prediction accuracy

from sklearn.naive_bayes import GaussianNB

# ... (Your existing code for data loading, preprocessing, and one-hot encoding) ...

# Model Training using Naive Bayes
model = GaussianNB()
model.fit(X_train, Y_train)

# Model Evaluation
# Accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on Training data : ', training_data_accuracy)

# Accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy on Test data : ', test_data_accuracy)

# Building Predictive System
# ... (Your existing code for creating input_data and encoding it) ...

prediction = model.predict(input_data_reshaped)
print(prediction)

if (prediction[0]== 0):
  print('The Person does not have a Heart Disease')
else:
  print('The Person has Heart Disease')
```

```
Accuracy on Training data :  0.90875
Accuracy on Test data :  0.91
[0]
The Person does not have a Heart Disease
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning
  warnings.warn(
```

create prediction in clustering method with prediction accuracy

```
# prompt: create prediction in clustering method with prediction accuracy

from sklearn.cluster import KMeans
```

```
from sklearn.metrics import accuracy_score

# Assuming X_train, X_test, Y_train, Y_test are already defined from your previous code

# Model Training using KMeans (Clustering)
kmeans = KMeans(n_clusters=2, random_state=42)  # Assuming 2 clusters for binary classifi
kmeans.fit(X_train)

# Predict cluster labels for the training and testing data
X_train_cluster_labels = kmeans.predict(X_train)
X_test_cluster_labels = kmeans.predict(X_test)

# Map cluster labels to target labels (0 or 1) based on majority class in each cluster
cluster_mapping = {}  # Dictionary to store the mapping
for cluster_label in range(2):
  cluster_data = Y_train[X_train_cluster_labels == cluster_label]
  majority_class = cluster_data.mode()[0]
  cluster_mapping[cluster_label] = majority_class

# Convert predicted cluster labels to target labels using the mapping
Y_train_predicted = np.array([cluster_mapping[cluster_label] for cluster_label in X_train
Y_test_predicted = np.array([cluster_mapping[cluster_label] for cluster_label in X_test_c


# Model Evaluation
training_accuracy = accuracy_score(Y_train, Y_train_predicted)
testing_accuracy = accuracy_score(Y_test, Y_test_predicted)

print('Accuracy on Training data : ', training_accuracy)
print('Accuracy on Test data : ', testing_accuracy)

# Building Predictive System (for the same input_data as before)
input_data_cluster_label = kmeans.predict(input_data_reshaped)
predicted_target_label = cluster_mapping[input_data_cluster_label[0]]


print(predicted_target_label)

if predicted_target_label == 0:
  print('The Person does not have a Heart Disease')
else:
  print('The Person has Heart Disease')
```

```
Accuracy on Training data :  0.635
Accuracy on Test data :  0.675
1
The Person has Heart Disease
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning
  warnings.warn(
```

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