ADTA 5410 - Application and Deployment Final Project

GROUP-14

1. Hemanth Sherla

2. Rohith Myana

3. Sushmitha Gangadari

4. Vivek Adharsh Veldi

CARDIO-VASCULAR DISEASES PREDICTION

predicting wheather a person is suffering from cardio vascuar diseases or not

INTRODUCTION

1. Our primary goals revolve around leveraging machine learning to forecast cardiovascular disease, pinpointing individuals at high risk for timely intervention. 2. Through comprehensive analysis of health data, our aim is to anticipate the onset of heart conditions, enabling swift intervention and tailored care delivery. 3. By identifying individuals susceptible to cardiovascular diseases, we can implement preventive measures and interventions, ultimately improving patient Our primary goals revolve around leveraging machine learning to forecast cardiovascular disease, pinpointing individuals at high risk for timely intervention. Through comprehensive analysis of health data, our aim is to anticipate the onset of heart conditions, enabling swift intervention and tailored care delivery. By identifying individuals susceptible to cardiovascular diseases, we can implement preventive measures and interventions, ultimately improving patient outcomes and reducing healthcare burdens. Additionally, our objective includes ensuring equitable distribution of resources, addressing disparities in healthcare access, and enhancing overall healthcare delivery efficiency. This holistic approach not only aids in early detection and intervention but also contributes to the broader goal of improving population health and well-being.

Data Description

The Dataset is collected from the Kaggle and it comprises various health-related attributes categorized into three types: Objective, Examination, and Subjective features. Objective features include factual information such as age, height, weight, and gender. Examination features encompass results from medical tests, including systolic and diastolic blood pressure, cholesterol levels, and glucose levels. Subjective features entail information provided by the patients themselves, such as smoking habits, alcohol intake, and physical activity levels. These attributes play a crucial role in predicting the presence or absence of cardiovascular disease, which serves as the target variable.

In [2]:
```
import pandas as pd
```

In [3]:
```
import numpy as np
```

In [4]:
```
import seaborn as sns
```

In [5]:
```
import os
```

In [6]:
```
os.getcwd()
```

Out[6]:
```
'C:\\Users\\sushm_8mc8bbk\\OneDrive\\Desktop\\SUSHMITHA\\IPYNB FIles\\ADTA 5410-Application and Deployment'
```

In [7]:
```
#pass the location of the csv file that is stored
#os.chdir("D:\SUSHMITHA\Dataframes_csvFiles")
```

In [8]:
```
os.getcwd()
```

Out[8]:
```
'C:\\Users\\sushm_8mc8bbk\\OneDrive\\Desktop\\SUSHMITHA\\IPYNB FIles\\ADTA 5410-Application and Deployment'
```

In [9]:
```
#importing dataset
df_cardio=pd.read_csv("cardiovascular_diseases.csv")
```

In [10]:
```
df_cardio.head()
```

Out[10]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

!!! here , age is in days not years

ap_hi is systolic blood pressure

ap_lo is diastolic blood pressure

gluc is the persons glucose level

height is in centimeters

weight is in kilograms

DATA UNDERSTANDING

In [11]:
```
df_cardio.shape
```

Out[11]:
```
(69999, 13)
```

In [17]:
```
len(df_cardio)
```

Out[17]:
```
69999
```

In [18]:
```
df_cardio.isnull().sum()
```

Out[18]:
```
id             0
 age           0
gender         0
height         0
weight         0
ap_hi          0
ap_lo          0
cholesterol    0
 gluc          0
smoke          0
alco           0
active         0
cardio         0
dtype: int64
```

In [19]:
```
df_cardio.isnull().values.any()
```

Out[19]:
```
False
```

In [20]:
```
df_cardio.isna().sum()
```

Out[20]:
```
id             0
 age           0
gender         0
height         0
weight         0
ap_hi          0
ap_lo          0
cholesterol    0
 gluc          0
smoke          0
alco           0
active         0
cardio         0
dtype: int64
```

In [21]:
```
df_cardio.columns
```

Out[21]:
```
Index(['id', ' age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo',
       'cholesterol', ' gluc', 'smoke', 'alco', 'active', 'cardio'],
      dtype='object')
```

In [22]:
```
df_cardio.columns.tolist()
```

Out[22]:
```
['id',
 ' age',
 'gender',
 'height',
 'weight',
 'ap_hi',
 'ap_lo',
 'cholesterol',
 ' gluc',
 'smoke',
 'alco',
 'active',
 'cardio']
```

DATA PREPARATION

In [23]:
```
df_cardio.dtypes
```

Out[23]:
```
id              int64
 age            int64
gender          int64
height          int64
weight        float64
ap_hi           int64
ap_lo           int64
cholesterol     int64
 gluc           int64
smoke           int64
alco            int64
active          int64
cardio          int64
dtype: object
```

In [24]:
```
#to view basic statistics
df_cardio.describe()
```

Out[24]:

	id	age	gender	height	weight	ap_hi	ap
count	69999.000000	69999.000000	69999.000000	69999.000000	69999.000000	69999.000000	69999.000
mean	49971.705224	19468.850512	1.349576	164.359148	74.205722	128.817412	96.630
std	28850.888785	2467.265969	0.476840	8.210157	14.395857	154.012516	188.473
min	0.000000	10798.000000	1.000000	55.000000	10.000000	-150.000000	-70.000
25%	25006.500000	17664.000000	1.000000	159.000000	65.000000	120.000000	80.000
50%	50001.000000	19703.000000	1.000000	165.000000	72.000000	120.000000	80.000
75%	74887.500000	21327.000000	2.000000	170.000000	82.000000	140.000000	90.000
max	99998.000000	23713.000000	2.000000	250.000000	200.000000	16020.000000	11000.000

In [25]:
```
#get a count of no of patients with cadiovascular diseases
df_cardio['cardio'].value_counts()
```

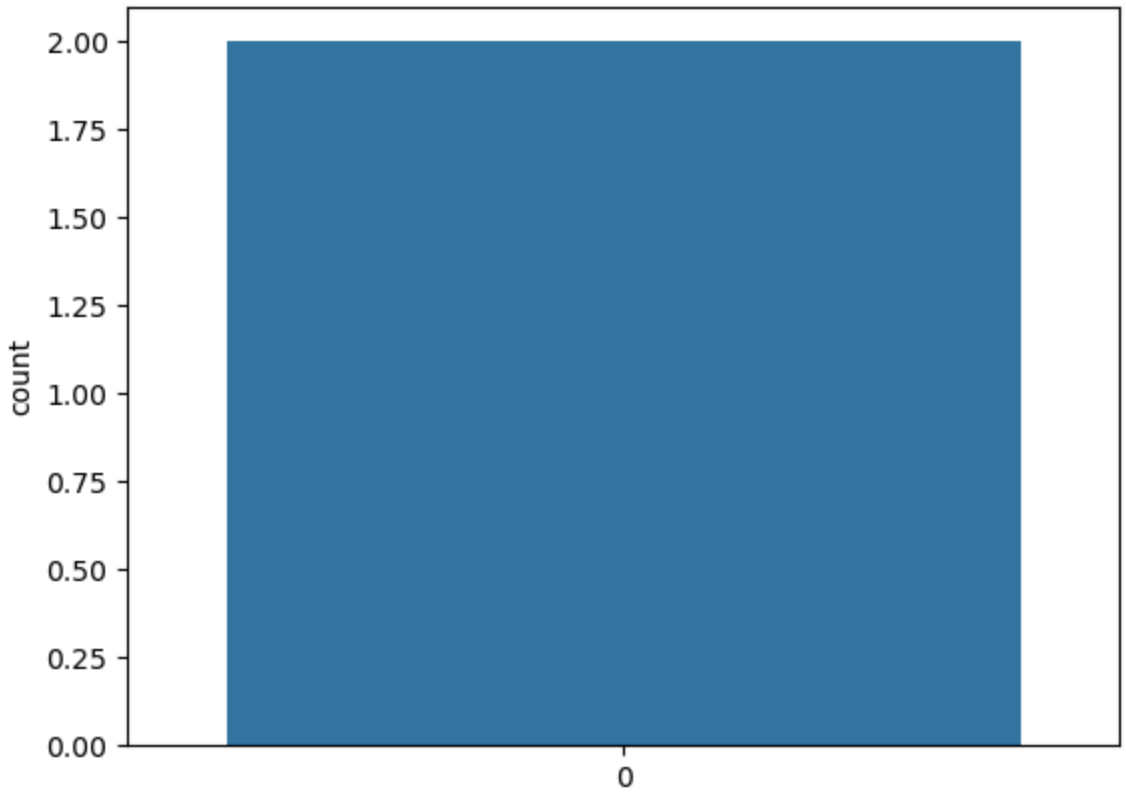
Out[25]:
```
cardio
0    35020
1    34979
Name: count, dtype: int64
```

0 indicates that the person is free from the diseases

1 indicates that the person is suffering from diseases

In [26]:
```
#visualize the count
sns.countplot(df_cardio['cardio'].value_counts())
```

Out[26]:
```
<Axes: ylabel='count'>
```



In [27]:
```
#look at number of people with cardiovascular diseases that exceed the number of peopl
#create a years column
df_cardio['years']=(df_cardio[' age']/365)
```

In [28]:
```
df_cardio['years']
```

Out[28]:
```
0        50.391781
1        55.419178
2        51.663014
3        48.282192
4        47.873973
           ...
69994    57.736986
69995    52.712329
69996    61.920548
69997    52.235616
69998    61.454795
Name: years, Length: 69999, dtype: float64
```

In [29]:
```
#rounding the vales of year column
df_cardio['years']=(df_cardio[' age']/365).round(0)
```

In [30]:
```
df_cardio['years']
```

Out[30]:
```
0        50.0
1        55.0
2        52.0
3        48.0
4        48.0
         ...
69994    58.0
69995    53.0
69996    62.0
69997    52.0
69998    61.0
Name: years, Length: 69999, dtype: float64
```

In [31]:
```
df_cardio['years'].dtypes
```

Out[31]:
```
dtype('float64')
```

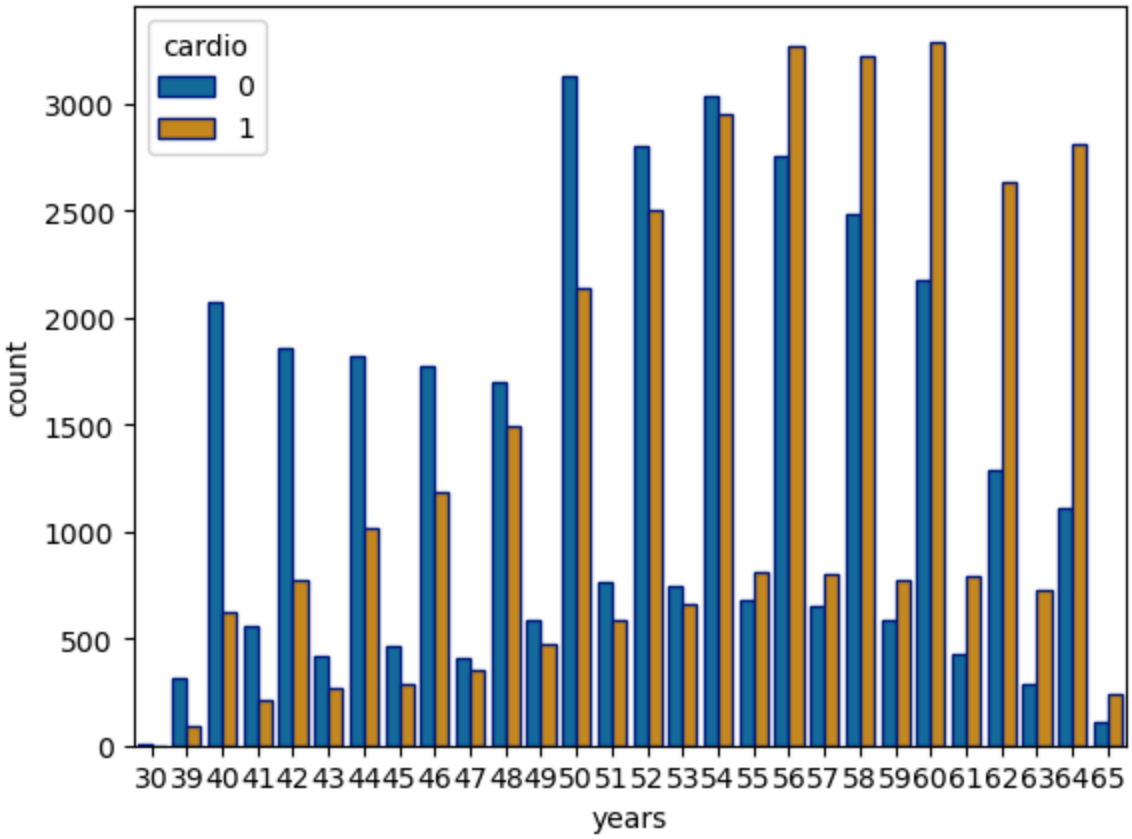
In [32]:
```
df_cardio['years']=pd.to_numeric(df_cardio['years'],downcast="integer")
```

In [33]:
```
df_cardio['years'].dtypes
```

Out[33]:
```
dtype('int8')
```

In [34]:
```
#visualize the data
sns.countplot(x="years" , hue="cardio",data=df_cardio,palette="colorblind",edgecolor=s
```

Out[34]:
```
<Axes: xlabel='years', ylabel='count'>
```



before age 55 we observed that the number of persons with cardiovascular diseases are less than number of persons without cardiovascular diseases

after age 55 we observed that the number of persons with cardiovascular diseases are more than number of persons without cardiovascular diseases

In [35]:
```
#get the corelation of columns
df_cardio.corr()
```

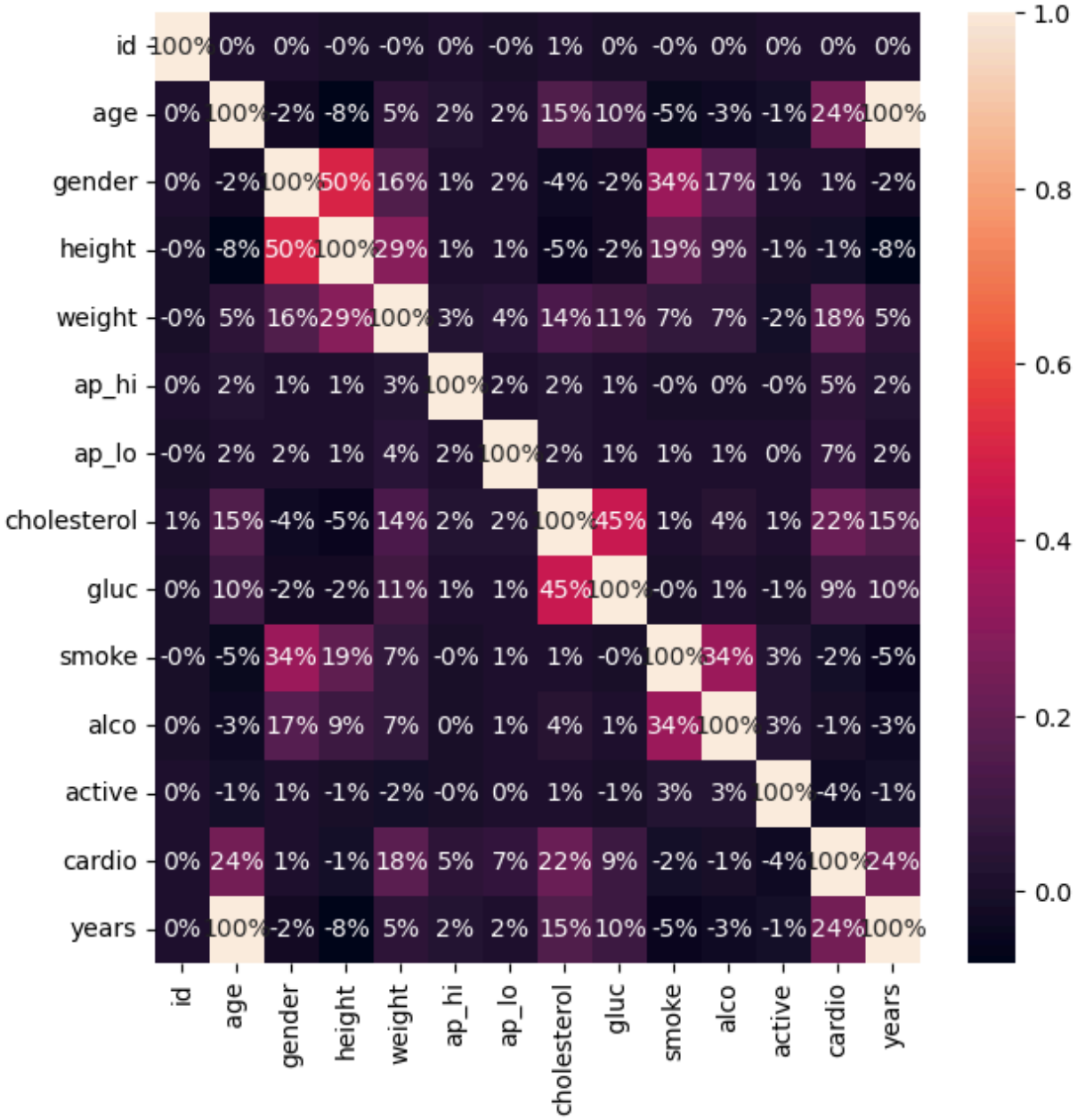
Out[35]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	
id	1.000000	0.003446	0.003520	-0.003055	-0.001826	0.003357	-0.002527	0.006083	0
age	0.003446	1.000000	-0.022806	-0.081520	0.053685	0.020765	0.017648	0.154419	0
gender	0.003520	-0.022806	1.000000	0.499044	0.155405	0.006005	0.015253	-0.035812	-0
height	-0.003055	-0.081520	0.499044	1.000000	0.290970	0.005488	0.006151	-0.050236	-0
weight	-0.001826	0.053685	0.155405	0.290970	1.000000	0.030702	0.043710	0.141771	0
ap_hi	0.003357	0.020765	0.006005	0.005488	0.030702	1.000000	0.016085	0.023779	0
ap_lo	-0.002527	0.017648	0.015253	0.006151	0.043710	0.016085	1.000000	0.024021	0
cholesterol	0.006083	0.154419	-0.035812	-0.050236	0.141771	0.023779	0.024021	1.000000	0
gluc	0.002477	0.098705	-0.020495	-0.018591	0.106857	0.011840	0.010805	0.451586	1
smoke	-0.003692	-0.047631	0.338133	0.187993	0.067780	-0.000922	0.005186	0.010359	-0
alco	0.001216	-0.029721	0.170965	0.094421	0.067112	0.001408	0.010601	0.035764	0
active	0.003743	-0.009930	0.005871	-0.006574	-0.016866	-0.000033	0.004781	0.009905	-0
cardio	0.003824	0.238167	0.008099	-0.010811	0.181659	0.054475	0.065718	0.221164	0
years	0.003041	0.999090	-0.023013	-0.081461	0.053662	0.020794	0.017755	0.154382	0

In [36]:
```
#visualize the data
import matplotlib.pyplot as plt
```

In [37]:
```
plt.figure(figsize=(7,7))
sns.heatmap(df_cardio.corr(),annot=True, fmt='.0%')
```

Out[37]:
```
<Axes: >
```



here,cardio column(x-label) has 24% positive correlation with age and years,1% positive correlation with gender,18% positive correlation with weight,22% positive correlation with cholesterol

In [38]:
```
#prepeare data for ML model
df_new=df_cardio.drop('years',axis=1)
```

In [39]:
```
#remove id column becoz it is 0% correlated with cardio
df_new=df_cardio.drop('id',axis=1)
```

In [ ]:
```

```

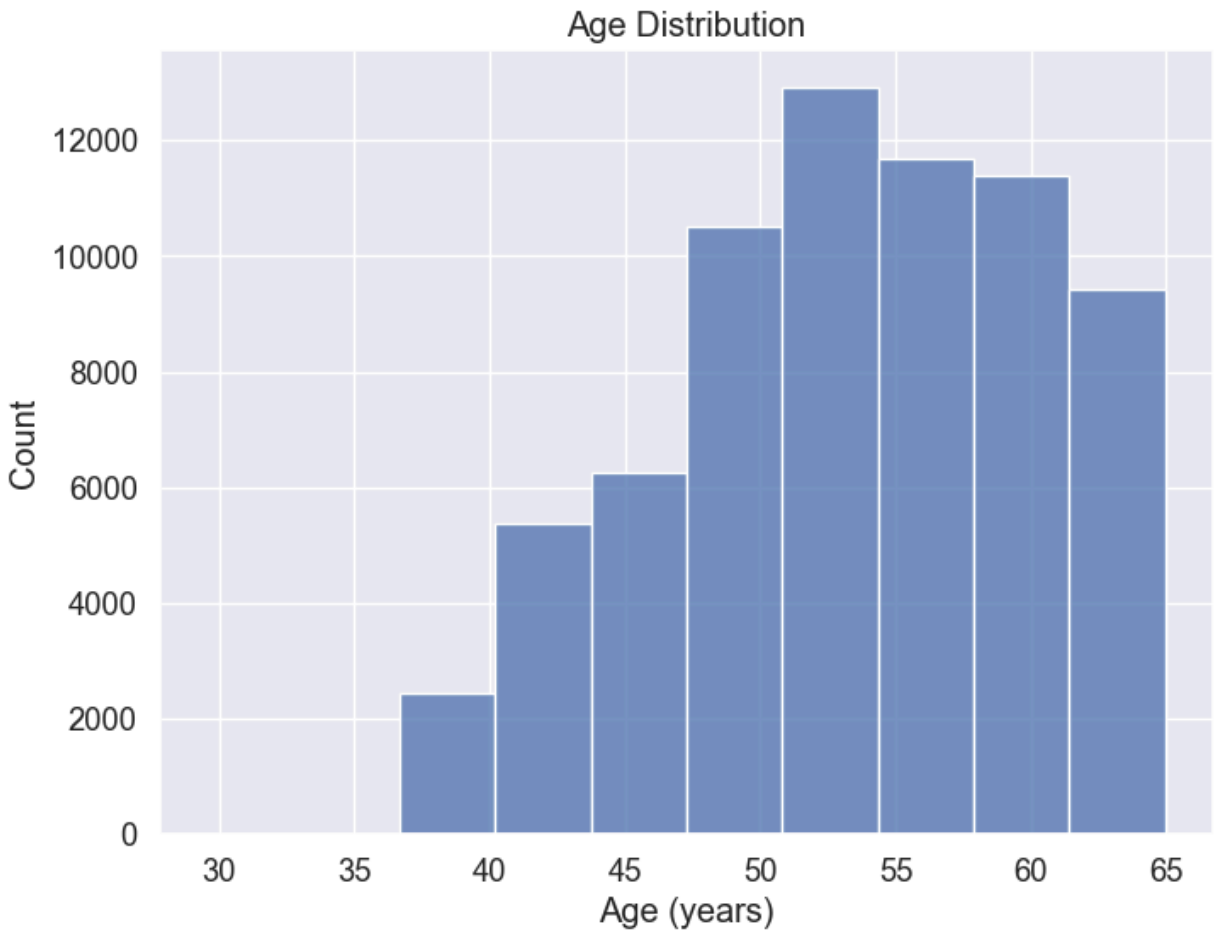
EXPLORATORY DATA ANALYSIS

Age Distribution

The goal of this analysis is to visualize the age distribution of individuals in the dataset to understand how age is spread across the population. We first convert the age data, which is given in days, into years by dividing the values by 365 (since there are 365 days in a year). Then, we use a histogram to visualize the distribution of ages, where the x-axis represents the age in years, and the y-axis shows the count of individuals in each age group.

In [135…

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

# Convert age from days to years (assuming 365 days per year)
df_cardio['age_years'] = df_cardio[' age'] / 365

# Age distribution
plt.figure(figsize=(8,6))
sns.histplot(df_cardio['age_years'], bins=10)
plt.title('Age Distribution')
plt.xlabel('Age (years)')
plt.ylabel('Count')
plt.show()
```



Gender distribution

This count plot shows the distribution of genders in the dataset. The x-axis represents gender (1 = Male, 2 = Female), while the y-axis shows the count of individuals in each category. It helps us understand whether the dataset is balanced in terms of gender, which is important for unbiased analysis of cardiovascular disease.

In [136…

```
plt.figure(figsize=(6,4))
sns.countplot(x='gender', data=df_cardio)
plt.title('Gender Distribution')
plt.xlabel('Gender (1 = Male, 2 = Female)')
plt.ylabel('Count')
plt.show()
```

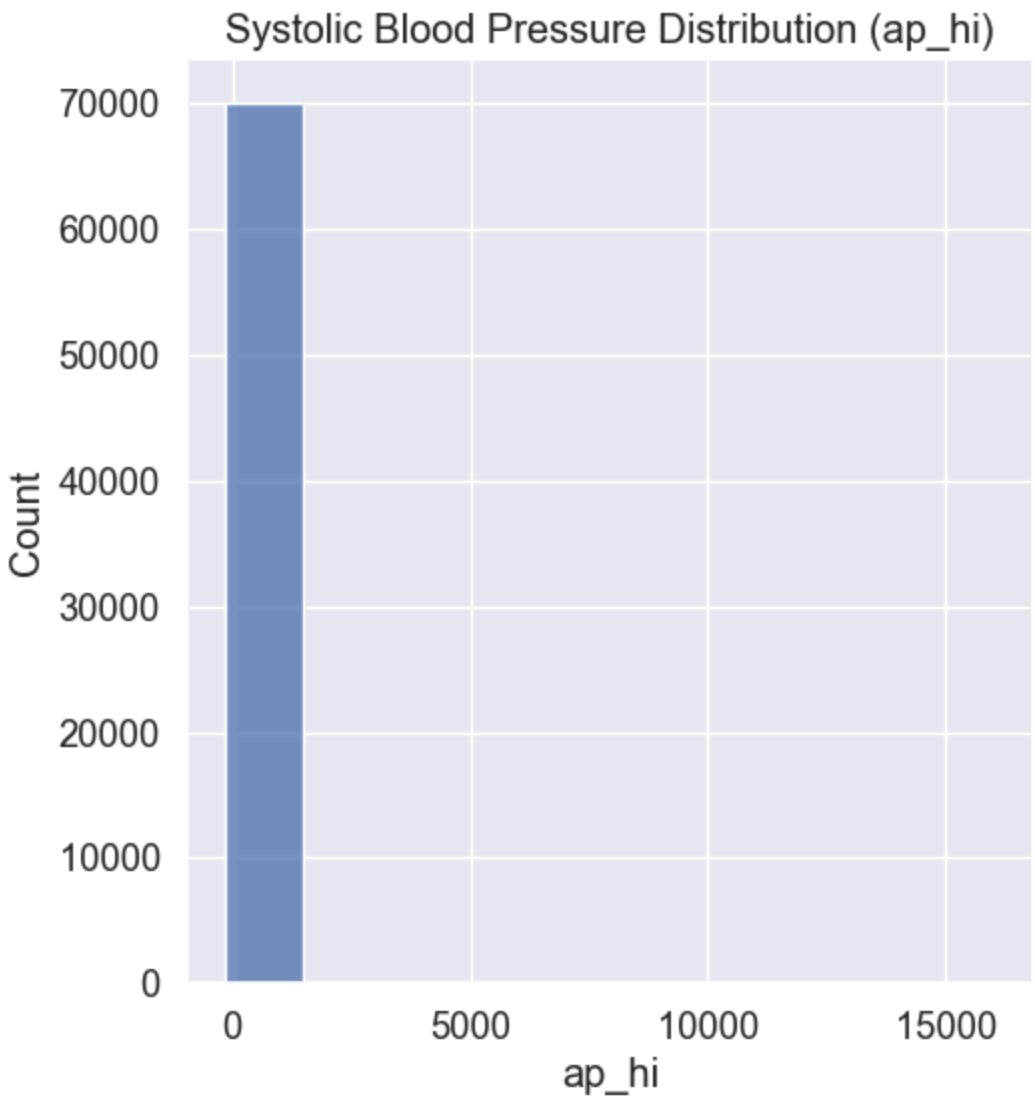
Blood pressure (ap_hi and ap_lo) distribution

The x-axis represents blood pressure values, while the y-axis shows the count of individuals in each range. Helps detect potential outliers or unusual values in blood pressure, which are key factors for cardiovascular disease risk.

In [137…

```
plt.figure(figsize=(12,6))
plt.subplot(1, 2, 1)
sns.histplot(df_cardio['ap_hi'], bins=10)
plt.title('Systolic Blood Pressure Distribution (ap_hi)')
plt.xlabel('ap_hi')
plt.ylabel('Count')
```

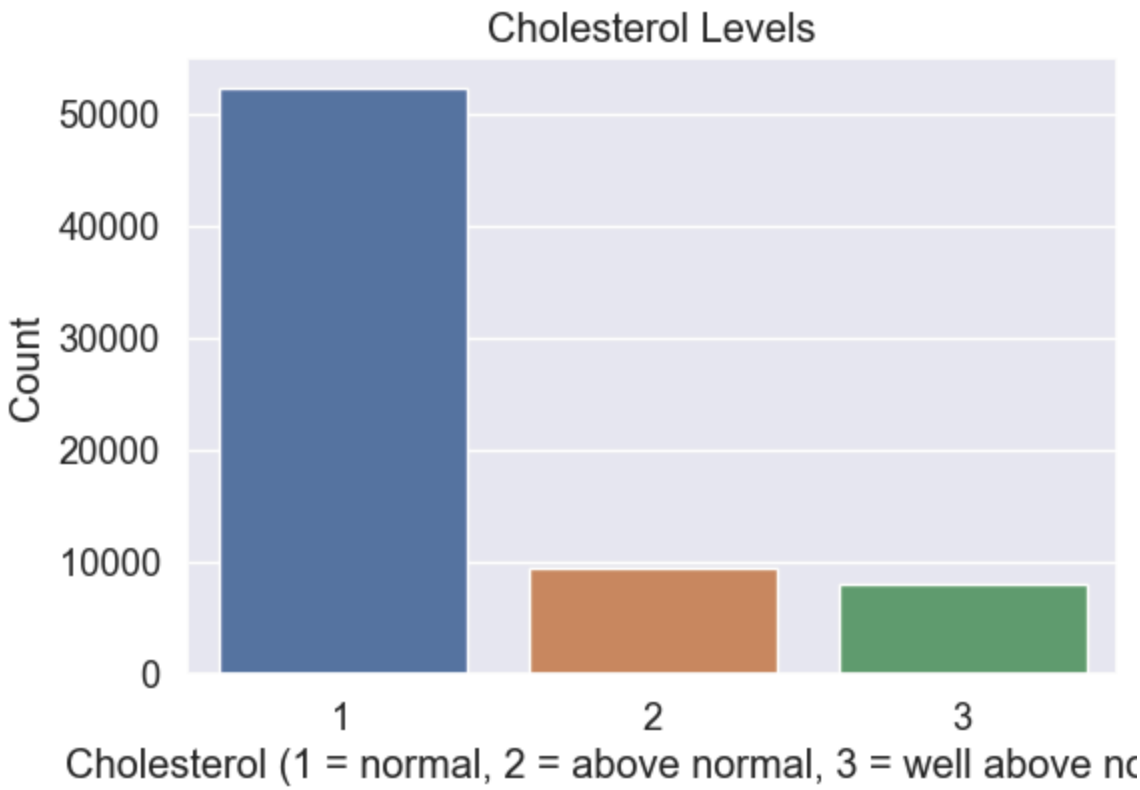
Out[137]:
```
Text(0, 0.5, 'Count')
```



Cholesterol distribution

The x-axis represents three categories of cholesterol: 1 = Normal 2 = Above normal 3 = Well above normal The y-axis shows the number of individuals in each category, giving a sense of how cholesterol is distributed within the group. This visualization helps assess whether cholesterol is a significant risk factor in the population.

In [138…
```
plt.figure(figsize=(6,4))
sns.countplot(x='cholesterol', data=df_cardio)
plt.title('Cholesterol Levels')
plt.xlabel('Cholesterol (1 = normal, 2 = above normal, 3 = well above normal)')
plt.ylabel('Count')
plt.show()
```

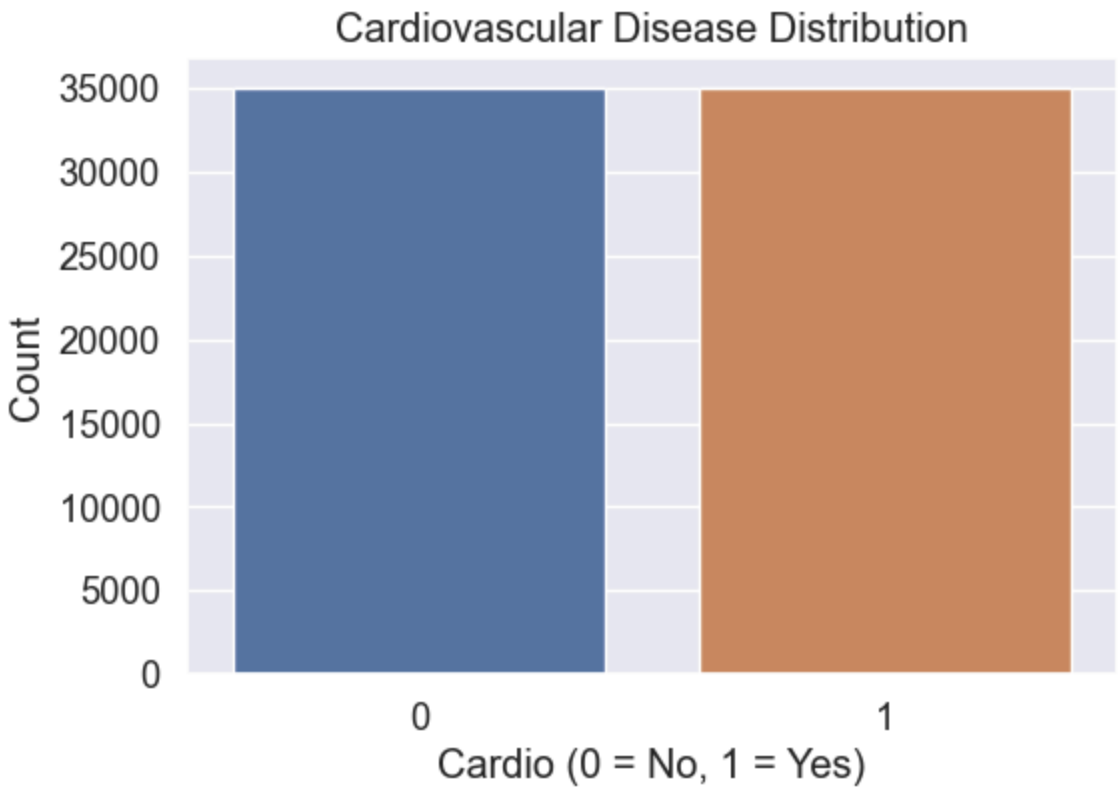


Cardiovascular disease distribution

The bar plot shows the distribution of individuals with and without cardiovascular disease. The x-axis represents the cardio variable: 0 = No Cardiovascular Disease 1 = Cardiovascular Disease The y-axis shows the number of individuals in each category. The plot shows nearly equal counts of individuals with and without cardiovascular disease, indicating a balanced dataset.

In [139…

```
plt.figure(figsize=(6,4))
sns.countplot(x='cardio', data=df_cardio)
plt.title('Cardiovascular Disease Distribution')
plt.xlabel('Cardio (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()
```



Correlation matrix

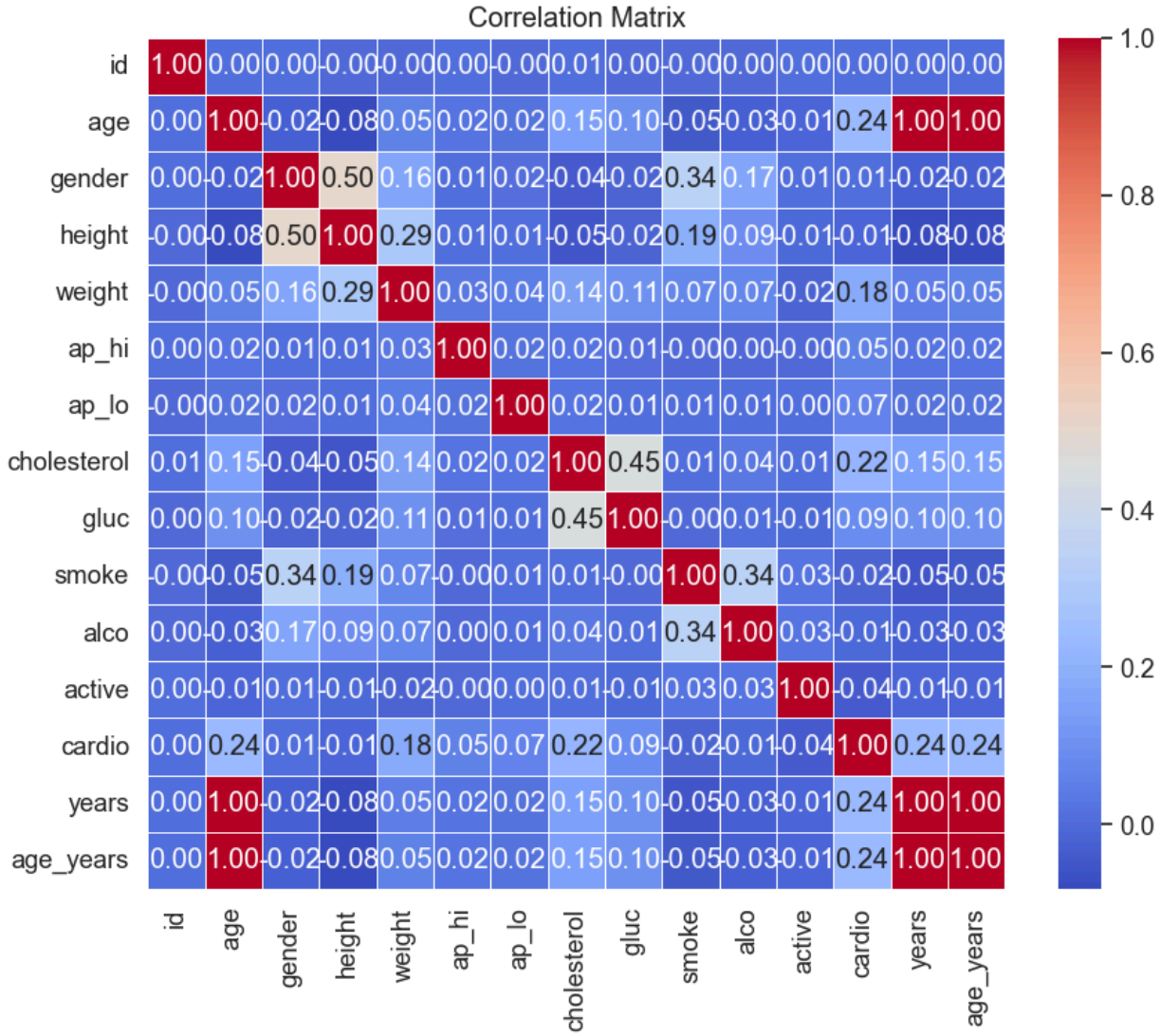
The correlation matrix shows relationships between variables. Positive correlations indicate both variables increase together. Moderate correlations are seen between cardiovascular disease and factors like age, cholesterol, and systolic blood pressure, highlighting their importance for predicting disease risk.

In [141…

```
# Correlation matrix
corr = df_cardio.corr()

# Heatmap for correlation
plt.figure(figsize=(10,8))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix')
```

Out[141]:

```
Text(0.5, 1.0, 'Correlation Matrix')
```



DATA SPLITTING

In [100…

```
df_new.head(15)
```

Out[100]:

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	year
0	18393	2	168	62.0	110	80	1	1	0	0	1	0	5
1	20228	1	156	85.0	140	90	3	1	0	0	1	1	5
2	18857	1	165	64.0	130	70	3	1	0	0	0	1	5
3	17623	2	169	82.0	150	100	1	1	0	0	1	1	4
4	17474	1	156	56.0	100	60	1	1	0	0	0	0	4
5	21914	1	151	67.0	120	80	2	2	0	0	0	0	6
6	22113	1	157	93.0	130	80	3	1	0	0	1	0	6
7	22584	2	178	95.0	130	90	3	3	0	0	1	1	6
8	17668	1	158	71.0	110	70	1	1	0	0	1	0	4
9	19834	1	164	68.0	110	60	1	1	0	0	0	0	5
10	22530	1	169	80.0	120	80	1	1	0	0	1	0	6
11	18815	2	173	60.0	120	80	1	1	0	0	1	0	5
12	14791	2	165	60.0	120	80	1	1	0	0	0	0	4
13	19809	1	158	78.0	110	70	1	1	0	0	1	0	5
14	14532	2	181	95.0	130	90	1	1	1	1	1	0	4

In [101…
```
#splitting data into feature data and target data
X=df_new.iloc[:,:-1].values
```

In [102…
```
X
```

Out[102]:
```
array([[1.8393e+04, 2.0000e+00, 1.6800e+02, ..., 0.0000e+00, 1.0000e+00,
        0.0000e+00],
       [2.0228e+04, 1.0000e+00, 1.5600e+02, ..., 0.0000e+00, 1.0000e+00,
        1.0000e+00],
       [1.8857e+04, 1.0000e+00, 1.6500e+02, ..., 0.0000e+00, 0.0000e+00,
        1.0000e+00],
       ...,
       [2.2601e+04, 1.0000e+00, 1.5800e+02, ..., 0.0000e+00, 1.0000e+00,
        1.0000e+00],
       [1.9066e+04, 2.0000e+00, 1.8300e+02, ..., 1.0000e+00, 0.0000e+00,
        1.0000e+00],
       [2.2431e+04, 1.0000e+00, 1.6300e+02, ..., 0.0000e+00, 0.0000e+00,
        1.0000e+00]])
```

In [103…
```
y=df_new.iloc[:,-1].values
```

In [104…
```
y
```

Out[104]:
```
array([50, 55, 52, ..., 62, 52, 61], dtype=int8)
```

In [105…
```
#splitting the datset as training and testing ddtaset
from sklearn.model_selection import train_test_split
```

In [106…
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=1)
```

In [107…
```
X_train
```

Out[107]:
```
array([[2.1913e+04, 1.0000e+00, 1.5700e+02, ..., 0.0000e+00, 0.0000e+00,
        1.0000e+00],
       [1.8787e+04, 1.0000e+00, 1.5600e+02, ..., 0.0000e+00, 1.0000e+00,
        0.0000e+00],
       [1.6093e+04, 2.0000e+00, 1.6800e+02, ..., 0.0000e+00, 1.0000e+00,
        0.0000e+00],
       ...,
       [1.8049e+04, 2.0000e+00, 1.7800e+02, ..., 0.0000e+00, 1.0000e+00,
        0.0000e+00],
       [2.1957e+04, 1.0000e+00, 1.6900e+02, ..., 0.0000e+00, 0.0000e+00,
        1.0000e+00],
       [2.0671e+04, 1.0000e+00, 1.7400e+02, ..., 0.0000e+00, 1.0000e+00,
        1.0000e+00]])
```

In [108…
```
X_test
```

Out[108]:
```
array([[1.8857e+04, 2.0000e+00, 1.7100e+02, ..., 1.0000e+00, 1.0000e+00,
        1.0000e+00],
       [2.2696e+04, 2.0000e+00, 1.8500e+02, ..., 0.0000e+00, 1.0000e+00,
        1.0000e+00],
       [1.5052e+04, 2.0000e+00, 1.5900e+02, ..., 0.0000e+00, 0.0000e+00,
        0.0000e+00],
       ...,
       [1.6005e+04, 2.0000e+00, 1.7000e+02, ..., 0.0000e+00, 1.0000e+00,
        1.0000e+00],
       [1.7469e+04, 2.0000e+00, 1.6500e+02, ..., 0.0000e+00, 1.0000e+00,
        0.0000e+00],
       [1.9589e+04, 1.0000e+00, 1.6400e+02, ..., 0.0000e+00, 0.0000e+00,
        0.0000e+00]])
```

In [109…
```
y_train
```

Out[109]:
```
array([60, 51, 44, ..., 49, 60, 57], dtype=int8)
```

In [110…
```
y_test
```

Out[110]:
```
array([52, 62, 41, ..., 44, 48, 54], dtype=int8)
```

In [111…
```
#feature scaling
#scale the values in the data to be between 0 nd 1
from sklearn.preprocessing import StandardScaler
```

In [112…
```
sc=StandardScaler()
```

In [113…
```
X_train=sc.fit_transform(X_train)
X_test=sc.fit_transform(X_test)
```

MODEL CONSTRUCTION

MODEL: 1 RANDON FOREST

In [114…
```
#use random forest classifier
from sklearn.ensemble import RandomForestClassifier
```

In [115…
```
forest=RandomForestClassifier(n_estimators=10,criterion='entropy',random_state=1)
```

In [116…
```
forest
```

Out[116]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=1)
```

In [117…
```
forest.fit(X_train,y_train)
```

Out[117]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=1)
```

In [118…
```
#Test the model accuracy on the training data set
model_rf=forest
model_rf.score(X_train,y_train)
```

Out[118]:
```
0.999580944398941
```

In [119…
```
#test the model accuracy on the test data set
from sklearn.metrics import confusion_matrix
cm_rf=confusion_matrix(y_test,model_rf.predict(X_test))
```

In [120…
```
#true negative
TN=cm_rf[0][0]
#true positive
TP=cm_rf[1][1]
#FN=false negative
FN=cm_rf[1][0]
#FP=false positive
FP=cm_rf[0][1]
```

In [121…
```
#print the confusion matrix
cm_rf
```

Out[121]:

```
array([[   0,    1,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,   58,   35,    1,    2,    0,    1,    0,    0,    0,    0,
          1,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,   58,  590,    8,    5,    0,    1,    0,    1,    0,    1,
          0,    0,    1,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    2,   73,   93,   10,    0,    3,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    2,   17,   87,  522,    0,    1,    0,    4,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    5,    4,   51,   95,   12,    0,    1,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    2,    5,    1,   12,   81,  603,    0,    4,    0,    0,
          0,    2,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    2,    0,   47,  117,   19,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,   11,   81,  608,    1,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,   45,  104,   15,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,   15,   79,  736,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    4,    2,   52,
        183,   19,    0,    2,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    1,    0,    1,    0,    1,    0,    4,
         80, 1238,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1,
          0,   87,  247,    5,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    8,   86, 1217,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,   97,  269,    5,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    3,   87, 1411,    1,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    1,   83,  304,   11,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    1,   72, 1441,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,   61,  272,    5,    0,
          0,    0,    0,    0,    0,    0],
```

```
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    1,   43, 1420,    0,
    0,    0,    0,    0,    0,    0],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    1,    0,   44,  259,
   12,    0,    0,    0,    0,    0],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,    2,   29,
 1380,    1,    0,    0,    0,    0],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
   68,  236,   12,    0,    0,    0],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,    1,    0,
    9,   12,  988,    1,    2,    0],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    2,    1,   50,  170,   36,    1],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    2,    0,   12,    5,  917,    1],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    1,    0,    2,    0,   50,   31]], dtype=int64)
```

In [122…

```
#print the model accuracy on the test data
accuracy1=(TP+TN)/(TP+TN+FN+FP)
accuracy1



#print("model accuracy ={}".format((TP+TN)/(TP+TN+FN+FP)))
```

Out[122]:

```
0.9830508474576272
```

MODEL : 2 LOGISTIC REGRESSION

In [123…

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

In [142…

```
model_lg = LogisticRegression()
model_lg.fit(X_train, y_train)

# Step 5: Model Evaluation
y_pred_lg = model_lg.predict(X_test)

accuracy_lg = accuracy_score(y_test, y_pred_lg)
print("Accuracy:", accuracy_lg)

print("Classification Report:")
Logistic_regg_report=classification_report(y_test, y_pred_lg)
print(Logistic_regg_report)

print("Confusion Matrix:")
```

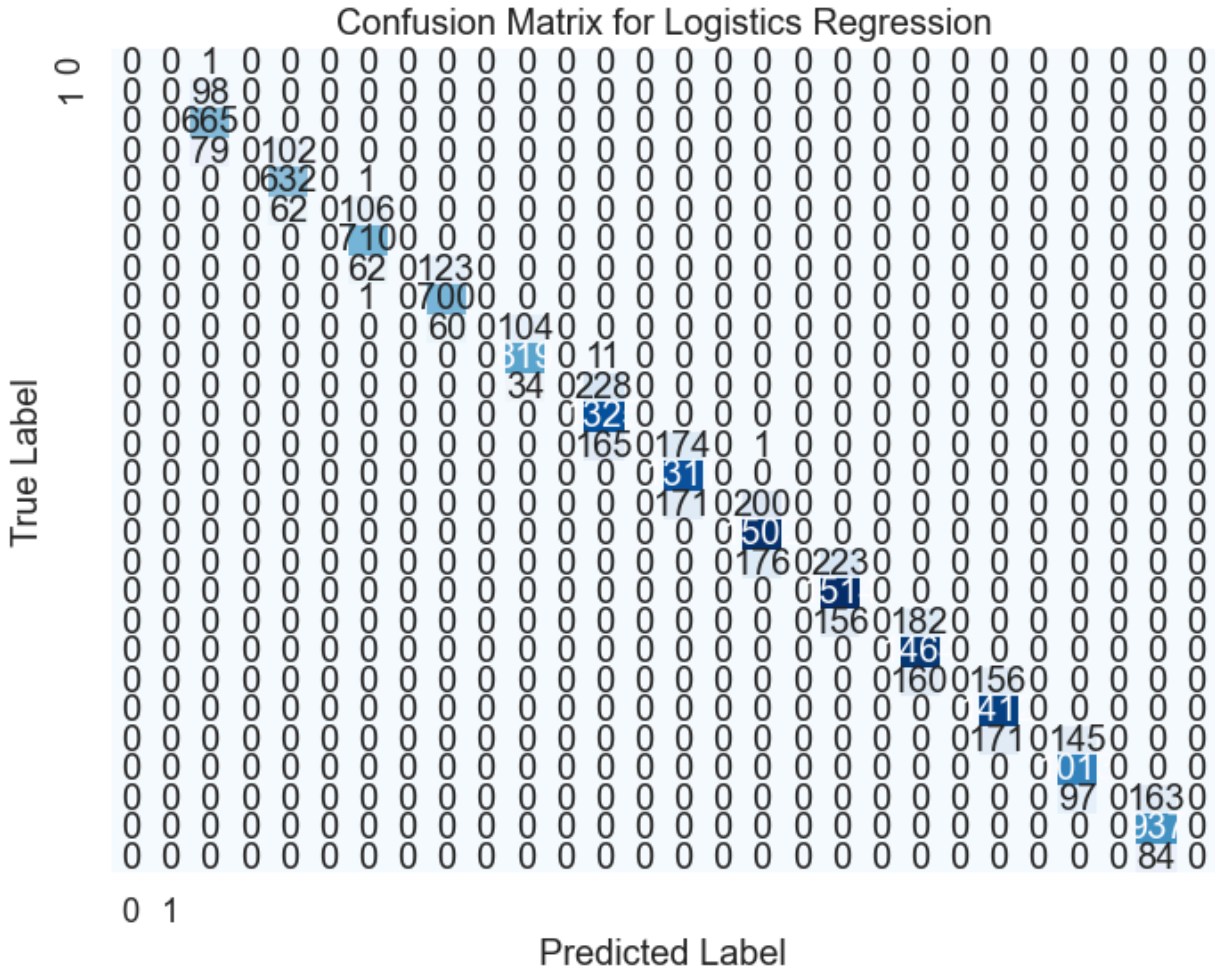
```
print(confusion_matrix(y_test, y_pred_lg))
#conf_mat_logis=confusion_matrix(y_test, y_pred_lg)
```

```
Accuracy: 0.8002285714285714
Classification Report:
              precision    recall  f1-score   support

          30       0.00      0.00      0.00         1
          39       0.00      0.00      0.00        98
          40       0.79      1.00      0.88       665
          41       0.00      0.00      0.00       181
          42       0.79      1.00      0.88       633
          43       0.00      0.00      0.00       168
          44       0.81      1.00      0.89       710
          45       0.00      0.00      0.00       185
          46       0.79      1.00      0.88       701
          47       0.00      0.00      0.00       164
          48       0.86      0.99      0.92       830
          49       0.00      0.00      0.00       262
          50       0.77      1.00      0.87      1325
          51       0.00      0.00      0.00       340
          52       0.79      1.00      0.88      1311
          53       0.00      0.00      0.00       371
          54       0.80      1.00      0.89      1502
          55       0.00      0.00      0.00       399
          56       0.80      1.00      0.89      1514
          57       0.00      0.00      0.00       338
          58       0.81      1.00      0.90      1464
          59       0.00      0.00      0.00       316
          60       0.81      1.00      0.90      1412
          61       0.00      0.00      0.00       316
          62       0.81      1.00      0.89      1013
          63       0.00      0.00      0.00       260
          64       0.79      1.00      0.88       937
          65       0.00      0.00      0.00        84

    accuracy                           0.80     17500
   macro avg       0.37      0.46      0.41     17500
weighted avg       0.64      0.80      0.71     17500

Confusion Matrix:
[[   0    0    1    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0   98    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0  665    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0   79    0  102    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0  632    0    1    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0   62    0  106    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0  710    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0   62    0  123    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    1    0  700    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0   60    0  104    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0  819    0   11    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
```

```
 [   0    0    0    0    0    0    0    0    0    0   34    0  228    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0 1325    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0  165    0
   174    0    1    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
  1311    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
   171    0  200    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0 1502    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0  176    0  223    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0 1514    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0  156    0  182    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0 1464    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0  160    0  156    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0 1412    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0  171    0  145    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0 1013    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0   97    0  163    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0  937    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0   84    0]]
```

```
C:\Users\sushm_8mc8bbk\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:
460: ConvergenceWarning: lbfgs failed to converge (status=1):
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(
C:\Users\sushm_8mc8bbk\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter to control t
his behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\sushm_8mc8bbk\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter to control t
his behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\sushm_8mc8bbk\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter to control t
his behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [125…
```
# Create confusion matrix heatmap
class_labels = ['0', '1']
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
sns.heatmap(conf_mat_logis, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Logistics Regression')
plt.show()
```



In [126…
```
#print the model accuracy on the test data
print("model accuracy ={}".format(accuracy_lg))
```

```
model accuracy =0.8002285714285714
```

In [127…
```
model_lg.score(X_train,y_train)
```

Out[127]:
```
0.7963580258671594
```

MODEL : 3 DECISION TREE CLASSIFIER

In [128…
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
# Create and train decision tree classifier
clf = DecisionTreeClassifier(random_state=92)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy4= accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy4)
```

```
Accuracy: 0.9158857142857143
```

In [129…

```
print("Classification Report:")
print(classification_report(y_test, y_pred))

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

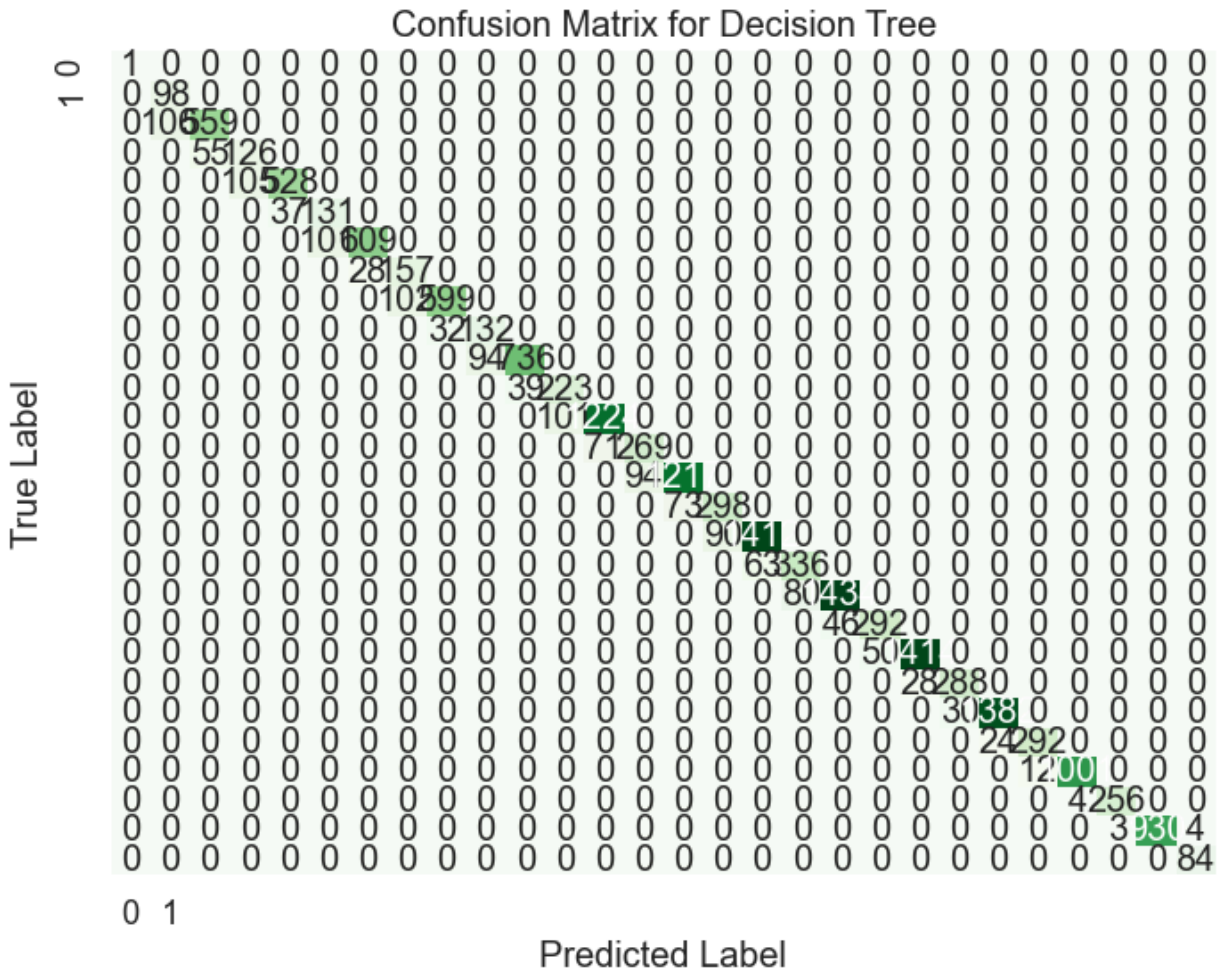
```
Classification Report:
              precision    recall  f1-score   support

          30       1.00      1.00      1.00         1
          39       0.48      1.00      0.65        98
          40       0.91      0.84      0.87       665
          41       0.55      0.70      0.61       181
          42       0.93      0.83      0.88       633
          43       0.56      0.78      0.66       168
          44       0.96      0.86      0.90       710
          45       0.61      0.85      0.71       185
          46       0.95      0.85      0.90       701
          47       0.58      0.80      0.68       164
          48       0.95      0.89      0.92       830
          49       0.69      0.85      0.76       262
          50       0.95      0.92      0.93      1325
          51       0.74      0.79      0.77       340
          52       0.94      0.93      0.94      1311
          53       0.77      0.80      0.79       371
          54       0.96      0.94      0.95      1502
          55       0.81      0.84      0.82       399
          56       0.97      0.95      0.96      1514
          57       0.85      0.86      0.86       338
          58       0.98      0.97      0.97      1464
          59       0.91      0.91      0.91       316
          60       0.98      0.98      0.98      1412
          61       0.96      0.92      0.94       316
          62       1.00      0.99      0.99      1013
          63       0.99      0.98      0.99       260
          64       1.00      0.99      1.00       937
          65       0.95      1.00      0.98        84

    accuracy                           0.92     17500
   macro avg       0.85      0.89      0.87     17500
weighted avg       0.93      0.92      0.92     17500

Confusion Matrix:
```

Out[129]:
```
array([[   1,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,   98,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,  106,  559,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,   55,  126,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,  105,  528,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,   37,  131,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,  101,  609,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,   28,  157,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,  102,  599,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,   32,  132,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,   94,  736,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   39,
        223,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        101, 1224,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,   71,  269,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,   94, 1217,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,   73,  298,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,   90, 1412,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,   63,  336,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,   80, 1434,    0,    0,    0,
          0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,   46,  292,    0,    0,
          0,    0,    0,    0,    0,    0],
```

```
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,   50, 1414,    0,
    0,    0,    0,    0,    0,    0],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,   28,  288,
    0,    0,    0,    0,    0,    0],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,   30,
 1382,    0,    0,    0,    0,    0],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
   24,  292,    0,    0,    0,    0],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,   12, 1001,    0,    0,    0],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    4,  256,    0,    0],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    3,  930,    4],
[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,   84]], dtype=int64)
```

In [130…

```python
# Create confusion matrix heatmap
class_labels = ['0', '1']
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
sns.heatmap(conf_mat_DT, annot=True, fmt="d", cmap="Greens", cbar=False,
            xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Decision Tree')
plt.show()
```



In [131…

```
import pandas as pd

# List of accuracy values for four models
accuracies = [accuracy1,accuracy_rf,accuracy4]

# Define model names
model_names = ['Logistic Regression', 'Random Forest',  'Decision Tree']

# Create DataFrame
accuracy_df = pd.DataFrame({'Model': model_names, 'Accuracy': accuracies})

# Print the DataFrame
print(accuracy_df)
```

```
                Model  Accuracy
0  Logistic Regression  0.983051
1        Random Forest  0.889600
2        Decision Tree  0.915886
```

ASSUMPTION VALIDATION

Independence of Observations: Each observation should be independent. The data points should not influence each other, ensuring that the model learns from individual cases without bias from related subjects.

Feature Independence: While not strictly necessary for Random Forest, it's assumed that features do not have perfect multicollinearity. High correlations among predictors can affect model interpretability and performance.

Sufficient Sample Size: The dataset should be large enough to provide reliable estimates of model parameters. A small sample size may lead to overfitting and unreliable predictions.

Relevance of Features: The selected features (age, gender, blood pressure, etc.) should have a logical and statistically significant relationship with the target variable (cardio). Irrelevant features can introduce noise and reduce model performance.

No Significant Outliers: The data should be free of significant outliers that could disproportionately influence the model's predictions. Outliers can skew results and affect the stability of the model.

Balanced Class Distribution: For classification tasks, it's important that the classes in the target variable (cardio) are reasonably balanced. Imbalanced classes can lead to biased predictions, where the model favors the majority class.

RESULTS

In [132…

```
from tabulate import tabulate

# Display DataFrame using tabulate
print(tabulate(accuracy_df, headers='keys', tablefmt='grid'))
```

```
+----+---------------------+------------+
|    | Model               |   Accuracy |
+====+=====================+============+
|  0 | Logistic Regression |   0.983051 |
+----+---------------------+------------+
|  1 | Random Forest       |   0.8896   |
+----+---------------------+------------+
|  2 | Decision Tree       |   0.915886 |
+----+---------------------+------------+
```

CONCLUSION

Based on the accuracy results of the three models for predicting cardiovascular disease, the following conclusions can be drawn:

Logistic Regression as the Top Performer: With an accuracy of 98.31%, Logistic Regression is the most effective model for predicting cardiovascular disease in this dataset. Its high accuracy indicates that it reliably differentiates between the presence and absence of cardiovascular conditions.

Strong Performance of Decision Tree: The Decision Tree model achieved an accuracy of 91.59%. While it is not as accurate as Logistic Regression, it still performs well and can provide insights

into feature importance, making it useful for understanding which factors contribute to cardiovascular risk.

Random Forest Underperformance: The Random Forest model recorded the lowest accuracy at 88.96%. This is unexpected, given its typical robustness in handling complex datasets. The lower performance may suggest that further tuning of hyperparameters or addressing potential issues such as feature selection is necessary.

Recommendation for Model Selection: Given the results, Logistic Regression should be prioritized for deployment due to its superior accuracy. However, it may be beneficial to explore the Decision Tree for interpretability and feature insights, while further investigating the Random Forest model to improve its performance. Overall, Logistic Regression emerges as the best choice for predicting cardiovascular disease in this scenario, with the Decision Tree providing a complementary perspective on feature significance.

FUTURE STEPS

Hyperparameter Tuning: Optimize model performance by using techniques like Grid Search or Random Search for tuning hyperparameters, especially for the Random Forest and Decision Tree models.

Feature Engineering: Create new features (e.g., BMI) and select significant features through techniques like Recursive Feature Elimination (RFE) to improve model accuracy and interpretability.

Address Class Imbalance: Implement strategies like SMOTE or other resampling techniques to balance the target variable's classes, enhancing model reliability.

Model Interpretation: Use tools like SHAP or LIME to explain model predictions, helping stakeholders understand the key factors influencing cardiovascular risk.

Deployment Planning: Prepare for integrating the best-performing model into healthcare systems, ensuring it can provide real-time predictions in a user-friendly interface.

Performance Monitoring: Establish a system for continuously monitoring the model's performance post-deployment, with plans for periodic retraining to maintain accuracy with new data.

In [ ]:

In [ ]:

In [ ]: