KNN_HEARTATTACK_PREDICTION

December 8, 2024

HEART ATTCK PREDICTION

Heart Attack Prediction using machine learning involves using various factors (features) related to a person's health and medical history to predict whether they are at risk of a heart attack. The key goal is to create a model that can make predictions based on available data, which can help in preventive care and early diagnosis.

[504]:
```
!unzip /content/archive (88).zip
```

```
/bin/bash: -c: line 1: syntax error near unexpected token `('
/bin/bash: -c: line 1: `unzip /content/archive (88).zip'
```

[505]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[506]:
```
df=pd.read_csv('/content/archive (88).zip')
```

UNDERSTANDING THE DATA

[507]:
```
df
```

[507]:
```
     Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  \
0     40   M           ATA        140          289          0     Normal
1     49   F           NAP        160          180          0     Normal
2     37   M           ATA        130          283          0         ST
3     48   F           ASY        138          214          0     Normal
4     54   M           NAP        150          195          0     Normal
..   …  ..           …        …          …        …        …
913   45   M            TA        110          264          0     Normal
914   68   M           ASY        144          193          1     Normal
915   57   M           ASY        130          131          0     Normal
916   57   F           ATA        130          236          0        LVH
917   38   M           NAP        138          175          0     Normal

     MaxHR ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0      172              N      0.0       Up             0
1      156              N      1.0     Flat             1
```

```
2        98               N      0.0       Up           0
3       108               Y      1.5     Flat           1
4       122               N      0.0       Up           0
..      …            …      …      …            …
913     132               N      1.2     Flat           1
914     141               N      3.4     Flat           1
915     115               Y      1.2     Flat           1
916     174               N      0.0     Flat           1
917     173               N      0.0       Up           0

[918 rows x 12 columns]
```

[508]:
```
df.shape
```

[508]:
```
(918, 12)
```

[509]:
```
df.dtypes
```

[509]:
```
Age                 int64
Sex                object
ChestPainType      object
RestingBP           int64
Cholesterol         int64
FastingBS           int64
RestingECG         object
MaxHR               int64
ExerciseAngina     object
Oldpeak           float64
ST_Slope           object
HeartDisease        int64
dtype: object
```

[510]:
```
df.describe()
```

[510]:
```
              Age   RestingBP  Cholesterol   FastingBS       MaxHR  \
count  918.000000  918.000000   918.000000  918.000000  918.000000
mean    53.510893  132.396514   198.799564    0.233115  136.809368
std      9.432617   18.514154   109.384145    0.423046   25.460334
min     28.000000    0.000000     0.000000    0.000000   60.000000
25%     47.000000  120.000000   173.250000    0.000000  120.000000
50%     54.000000  130.000000   223.000000    0.000000  138.000000
75%     60.000000  140.000000   267.000000    0.000000  156.000000
max     77.000000  200.000000   603.000000    1.000000  202.000000

          Oldpeak  HeartDisease
count  918.000000    918.000000
mean     0.887364      0.553377
```

```
std     1.066570    0.497414
min    -2.600000    0.000000
25%     0.000000    0.000000
50%     0.600000    1.000000
75%     1.500000    1.000000
max     6.200000    1.000000
```

[511]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

[512]:
```
df.corr(numeric_only=True)
```

[512]:
```
                   Age  RestingBP  Cholesterol  FastingBS     MaxHR   Oldpeak  \
Age           1.000000   0.254399    -0.095282   0.198039 -0.382045  0.258612
RestingBP     0.254399   1.000000     0.100893   0.070193 -0.112135  0.164803
Cholesterol  -0.095282   0.100893     1.000000  -0.260974  0.235792  0.050148
FastingBS     0.198039   0.070193    -0.260974   1.000000 -0.131438  0.052698
MaxHR        -0.382045  -0.112135     0.235792  -0.131438  1.000000 -0.160691
Oldpeak       0.258612   0.164803     0.050148   0.052698 -0.160691  1.000000
HeartDisease  0.282039   0.107589    -0.232741   0.267291 -0.400421  0.403951

              HeartDisease
Age               0.282039
RestingBP         0.107589
Cholesterol      -0.232741
FastingBS         0.267291
MaxHR            -0.400421
Oldpeak           0.403951
```

```
HeartDisease      1.000000
```

DATA CLEANING

[513]:
```
df.isna().sum()
```

[513]:
```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

[514]:
```
df.duplicated().sum()
```

[514]:
```
0
```

[515]:
```
df['Sex'].value_counts()
```

[515]:
```
Sex
M    725
F    193
Name: count, dtype: int64
```

[516]:
```
df['ChestPainType'].value_counts()
```

[516]:
```
ChestPainType
ASY    496
NAP    203
ATA    173
TA      46
Name: count, dtype: int64
```

[517]:
```
df['RestingECG'].value_counts()
```

[517]:
```
RestingECG
Normal    552
LVH       188
ST        178
Name: count, dtype: int64
```

[518]:
```
df['ExerciseAngina'].value_counts()
```

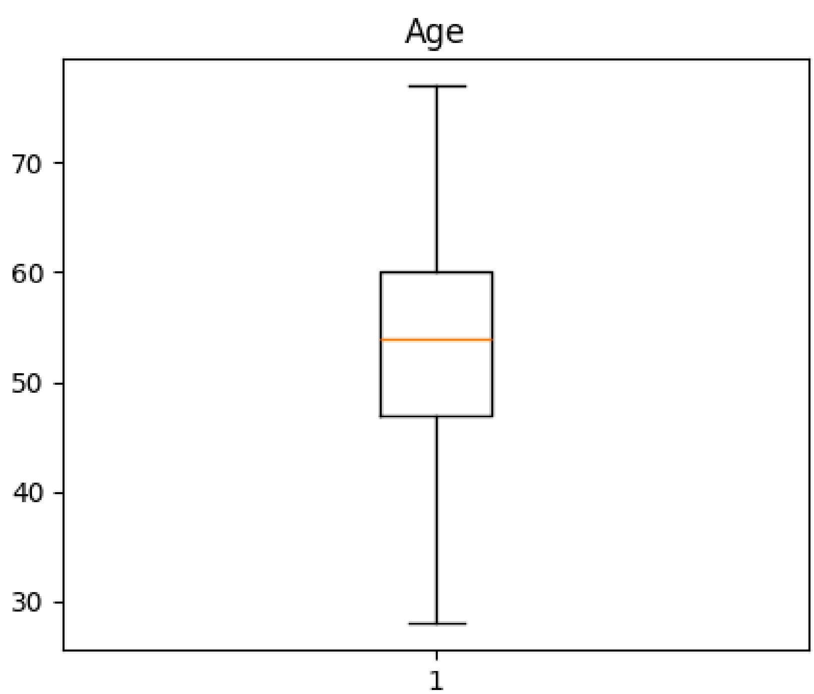
[518]:
```
ExerciseAngina
N    547
Y    371
Name: count, dtype: int64
```

[519]:
```
df['ST_Slope'].value_counts()
```

[519]:
```
ST_Slope
Flat    460
Up      395
Down     63
Name: count, dtype: int64
```

OUTLIER DETECTION

[520]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(5,4))
plt.boxplot(df['Age'])
plt.title('Age')
plt.show()
```



[521]:
```
q1=df['RestingBP'].quantile(0.25)
q1
```

[521]: 120.0

[522]:
```
q3=df['RestingBP'].quantile(0.75)
q3
```

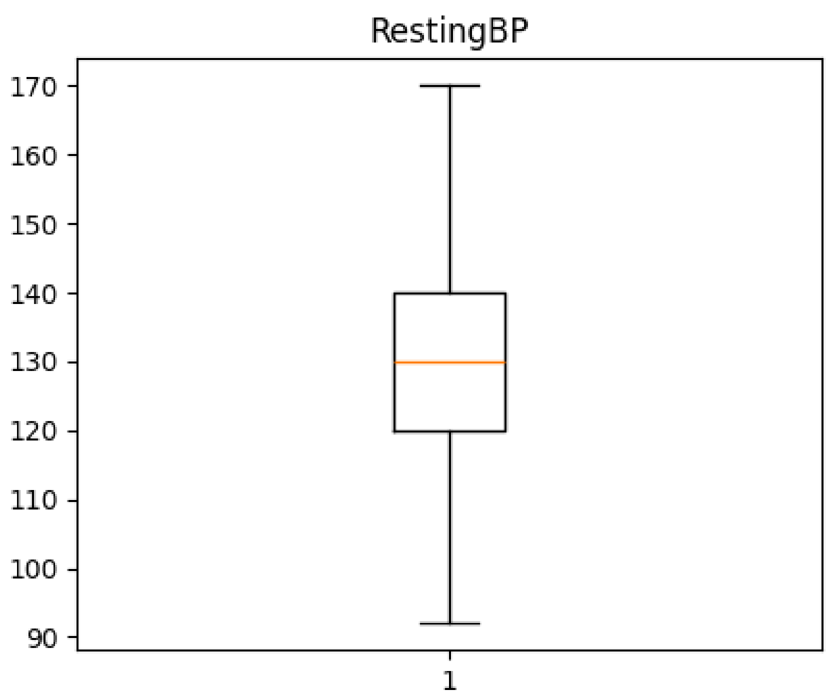
[522]: 140.0

[523]:
```
iqr=q3-q1
min_range=q1-1.5*iqr
print(min_range)
max_range=q3+1.5*iqr
print(max_range)
```

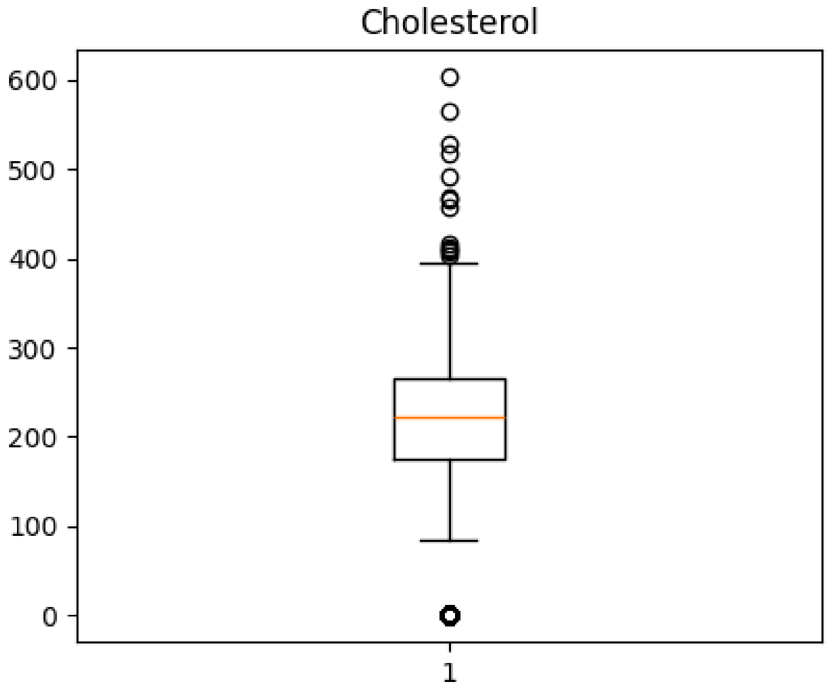
```
90.0
170.0
```

[524]:
```
df=df[(df['RestingBP'] <= max_range) & (df['RestingBP'] >= min_range)]
```

[525]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(5,4))
plt.boxplot(df['RestingBP'])
plt.title('RestingBP')
plt.show()
```

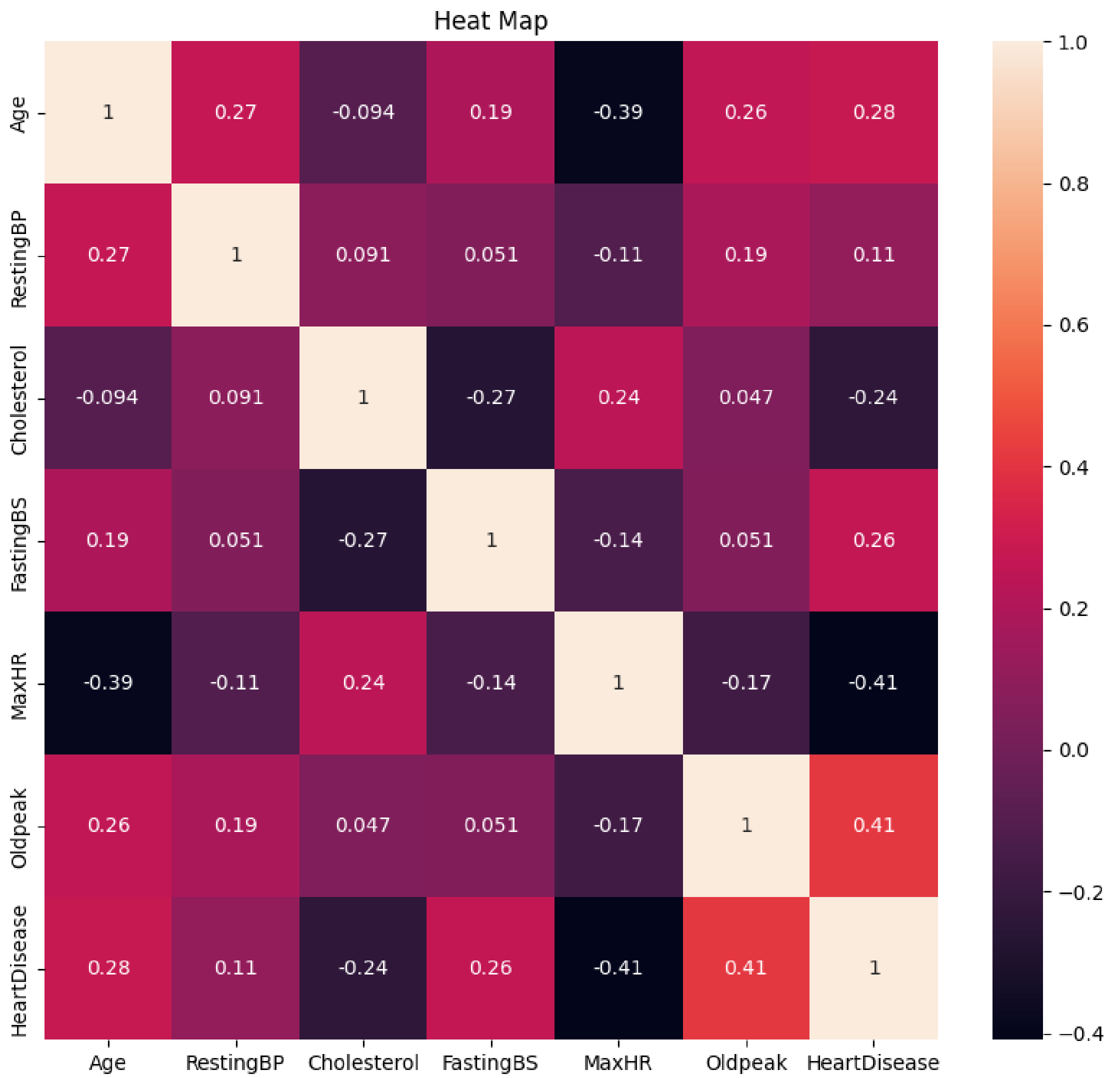


[526]:
```python
import matplotlib.pyplot as plt
plt.figure(figsize=(5,4))
plt.boxplot(df['Cholesterol'])
plt.title('Cholesterol')
plt.show()
```



HEAT MAP

[527]:
```python
plt.figure(figsize=(10,9))
var1=df.corr(numeric_only=True)
sns.heatmap(var1,annot=True)
plt.title('Heat Map')
plt.show()
```



CATEGORICAL VALUE ENCODING

[528]:
```
obj1=[]
for i in df:
  if df[i].dtype=='object':
    obj1.append(i)
```

[529]:
```
obj1
```

[529]: ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']

ORDINAL ENCODING

[530]:
```
from sklearn.preprocessing import OrdinalEncoder
obj=OrdinalEncoder(categories=[['Normal','ST','LVH']])
df['RestingECG']=obj.fit_transform(df[['RestingECG']])
```

```
<ipython-input-530-e43b1fad3ba5>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['RestingECG']=obj.fit_transform(df[['RestingECG']])
```

[531]:
```
from sklearn.preprocessing import OrdinalEncoder
obj=OrdinalEncoder(categories=[['Up','Flat','Down']])
df['ST_Slope']=obj.fit_transform(df[['ST_Slope']])
```

```
<ipython-input-531-7b6285265aa4>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['ST_Slope']=obj.fit_transform(df[['ST_Slope']])
```

LABEL ENCODING

[532]:
```
label_Enc=[]
Onehot_Enc=[]
for i in df:
  if df[i].dtype=='object' and df[i].nunique()>2:
    Onehot_Enc.append(i)
  elif df[i].dtype=='object' and df[i].nunique()<=2:
    label_Enc.append(i)
```

[533]:
```
label_Enc
```

[533]:
```
['Sex', 'ExerciseAngina']
```

[534]:
```
df['Sex']
```

[534]:
```
0      M
1      F
2      M
3      F
4      M
      ..
913    M
914    M
```

```
915    M
916    F
917    M
Name: Sex, Length: 890, dtype: object
```

[535]:
```python
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['Sex']=le.fit_transform(df['Sex'])
```

```
<ipython-input-535-5473b90f9496>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Sex']=le.fit_transform(df['Sex'])
```

[536]:
```python
le.inverse_transform(df['Sex'])
```

[536]:
```
array(['M', 'F', 'M', 'F', 'M', 'M', 'F', 'M', 'M', 'F', 'F', 'M', 'M',
       'M', 'F', 'F', 'M', 'F', 'M', 'M', 'F', 'M', 'F', 'M', 'M', 'M',
       'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F', 'F',
       'F', 'F', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F',
       'M', 'F', 'F', 'F', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F',
       'F', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'F',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F',
       'M', 'F', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M',
       'M', 'M', 'F', 'M', 'M', 'F', 'M', 'M', 'M', 'F', 'F', 'M', 'F',
       'F', 'M', 'F', 'F', 'M', 'M', 'M', 'F', 'F', 'F', 'M', 'M', 'M',
       'M', 'M', 'F', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F',
       'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M',
       'F', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'F', 'F', 'M', 'M', 'M',
       'F', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F',
       'M', 'F', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'F', 'F', 'M', 'M',
       'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F', 'M', 'F', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'F', 'F', 'M', 'F', 'F', 'M', 'F', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'F', 'M', 'F', 'M', 'M', 'M', 'M', 'F', 'M',
       'M', 'F', 'F', 'M', 'F', 'M', 'M', 'F', 'M', 'M', 'F', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'F', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
```

```
'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'F', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'F', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'F',
'M', 'M', 'F', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'F',
'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'F', 'F', 'F',
'M', 'F', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F', 'F',
'F', 'F', 'M', 'F', 'M', 'M', 'F', 'M', 'F', 'M', 'M', 'M', 'F',
'F', 'M', 'M', 'F', 'M', 'F', 'F', 'M', 'M', 'F', 'F', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F', 'M', 'M', 'M',
'M', 'M', 'M', 'F', 'F', 'F', 'M', 'F', 'M', 'M', 'M', 'M', 'F',
'M', 'F', 'M', 'F', 'F', 'M', 'F', 'M', 'F', 'M', 'M', 'M', 'M',
'F', 'M', 'F', 'M', 'F', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'F',
'F', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F',
'M', 'F', 'F', 'F', 'F', 'F', 'M', 'M', 'M', 'M', 'F', 'M', 'M',
'M', 'F', 'M', 'F', 'M', 'M', 'M', 'M', 'F', 'M', 'F', 'M', 'M',
'M', 'M', 'M', 'F', 'M', 'F', 'M', 'M', 'M', 'M', 'F', 'M', 'M',
'M', 'M', 'M', 'M', 'F', 'F', 'F', 'F', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F', 'F',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'F', 'F', 'F', 'M',
'F', 'M', 'M', 'M', 'M', 'F', 'F', 'M', 'M', 'M', 'M', 'F', 'F',
'F', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M',
'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F',
'M', 'M', 'M', 'M', 'M', 'F', 'F', 'M', 'F', 'M', 'M', 'F', 'M',
'M', 'M', 'F', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F', 'M', 'M',
'F', 'M', 'M', 'M', 'F', 'M'], dtype=object)
```

[537]:
```
from sklearn.preprocessing import LabelEncoder
le_ex=LabelEncoder()
df['ExerciseAngina']=le_ex.fit_transform(df['ExerciseAngina'])
```

```
<ipython-input-537-d4e7f18dc591>:3: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['ExerciseAngina']=le_ex.fit_transform(df['ExerciseAngina'])
```

[538]:
```python
le_ex.inverse_transform(df['Sex'])
```

[538]:
```
array(['Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'Y',
       'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N',
       'N', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'N', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N',
       'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N',
       'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N',
       'N', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y',
       'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
```

```
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N',
       'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'N',
       'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N',
       'N', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'N',
       'N', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'N', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N',
       'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'N', 'N', 'N', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y',
       'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N', 'Y',
       'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N',
       'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'N', 'Y'], dtype=object)
```

ONE HOT ENCODING

[539]:
```
from sklearn.preprocessing import OneHotEncoder
onehot=OneHotEncoder(sparse_output=False,drop='first')
result=onehot.fit_transform(df[Onehot_Enc])
```

[540]:
```
onehot.get_feature_names_out()
```

[540]:
```
array(['ChestPainType_ATA', 'ChestPainType_NAP', 'ChestPainType_TA'],
      dtype=object)
```

[541]:
```
result=pd.DataFrame(result,columns=onehot.get_feature_names_out())
```

[542]:
```
result
```

[542]:
```
     ChestPainType_ATA  ChestPainType_NAP  ChestPainType_TA
0                  1.0                0.0               0.0
1                  0.0                1.0               0.0
2                  1.0                0.0               0.0
3                  0.0                0.0               0.0
4                  0.0                1.0               0.0
..                 ...                ...               ...
885                0.0                0.0               1.0
886                0.0                0.0               0.0
887                0.0                0.0               0.0
888                1.0                0.0               0.0
889                0.0                1.0               0.0

[890 rows x 3 columns]
```

[543]:
```
df=df.drop(columns=Onehot_Enc)
```

[544]:
```
df.reset_index(drop=True, inplace=True)
```

[545]:
```
df=df.join(result)
```

[546]:
```
x=df.drop(columns=['HeartDisease'])
y=df['HeartDisease']
```

SAMPLING

[547]:
```
from imblearn.over_sampling import SMOTE
sampler=SMOTE()
x_resample,y_resample=sampler.fit_resample(x,y)
y_resample.value_counts()
```

[547]:
```
HeartDisease
0    490
1    490
Name: count, dtype: int64
```

SCALING

[548]:
```
from sklearn.preprocessing import MinMaxScaler
minmax=MinMaxScaler()
x_scaled=minmax.fit_transform(x_resample)
```

DATA SPLITTING

[549]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_scaled,y_resample,test_size=0.
↪2,random_state=50)
```

[550]:
```
x_train.shape
```

[550]: (784, 13)

[551]:
```
x_test.shape
```

[551]: (196, 13)

[552]:
```
y_train.shape
```

[552]: (784,)

[553]:
```
y_test.shape
```

[553]: (196,)

MODEL TRAINING

K NEAREST NEIGHBOR

K-Nearest Neighbors (KNN) is a simple, intuitive, and versatile machine learning algorithm used for both classification and regression tasks. It is based on the idea that similar data points tend to exist near each other in the feature space.

[554]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model=KNeighborsClassifier()
knn_model.fit(x_test,y_test)
```

[554]: KNeighborsClassifier()

[555]:
```
y_pred=knn_model.predict(x_test)
```

MODEL EVALUATION

[556]:
```
from sklearn.metrics import␣
↪accuracy_score,classification_report,confusion_matrix
print(accuracy_score(y_test,y_pred))
```

```
0.8979591836734694
```

[557]:
```
models=[KNeighborsClassifier()]
for model in models:
  model.fit(x_train,y_train)
  y_pred=model.predict(x_test)
  print(f"Accuracy :{accuracy_score(y_test,y_pred)}")
  print("------------------------------")
  print(f"Confusion_matrix:\n {confusion_matrix(y_test,y_pred)}")
  print()
  print(f"classification_report:\n{classification_report(y_test,y_pred)}")
```

```
print("\n\n\n")
```

```
Accuracy :0.8520408163265306
-----------------------------
Confusion_matrix:
 [[84 15]
 [14 83]]

classification_report:
              precision    recall  f1-score   support

           0       0.86      0.85      0.85        99
           1       0.85      0.86      0.85        97

    accuracy                           0.85       196
   macro avg       0.85      0.85      0.85       196
weighted avg       0.85      0.85      0.85       196
```

[558]:
```
!jupyter nbconvert --to pdf /content/KNN_HEARTATTACK_PREDICTION (2).ipynb
```

```
/bin/bash: -c: line 1: syntax error near unexpected token `('
/bin/bash: -c: line 1: `jupyter nbconvert --to pdf
/content/KNN_HEARTATTACK_PREDICTION (2).ipynb'
```

The heart attack prediction model, implemented using the K-Nearest Neighbors (KNN) algorithm, has achieved an 86% accuracy in predicting the likelihood of heart attacks. This is a promising result, especially considering the model's ability to effectively distinguish between the two classes (heart attack or no heart attack). The high accuracy can be attributed to the balanced nature of the dataset, which was achieved through the use of the SMOTE (Synthetic Minority Over-sampling Technique) technique. By applying SMOTE, the dataset was resampled to ensure that both classes (positive and negative heart attack cases) were equally represented, thereby preventing the model from being biased towards predicting the majority class.

KNN, being a distance-based algorithm, calculates the proximity of new data points to the training examples. It works well for this problem due to its simplicity and the fact that heart attack predictions are often influenced by a combination of numerical and categorical features, such as age, cholesterol levels, blood pressure, and chest pain type. These features, when properly preprocessed, allow the KNN model to classify a new patient's data based on similarities to known examples.

The performance of the model can be further evaluated using a classification report, which provides important metrics such as precision, recall, and F1-score. These metrics give a more detailed insight into how well the model performs in detecting both heart attack and non-heart attack cases. A high precision indicates that when the model predicts a heart attack, it is correct most of the time. On the other hand, a high recall suggests that the model is good at identifying most of the true heart attack cases, reducing the chances of false negatives.

Overall, this model provides a reliable approach for heart attack prediction, which can be useful in clinical decision-making, potentially aiding healthcare professionals in making faster, data-driven decisions. While 86% accuracy is impressive, future improvements could involve fine-tuning the model's hyperparameters, exploring other algorithms, or incorporating additional features such as family history or lifestyle factors to further enhance prediction accuracy.