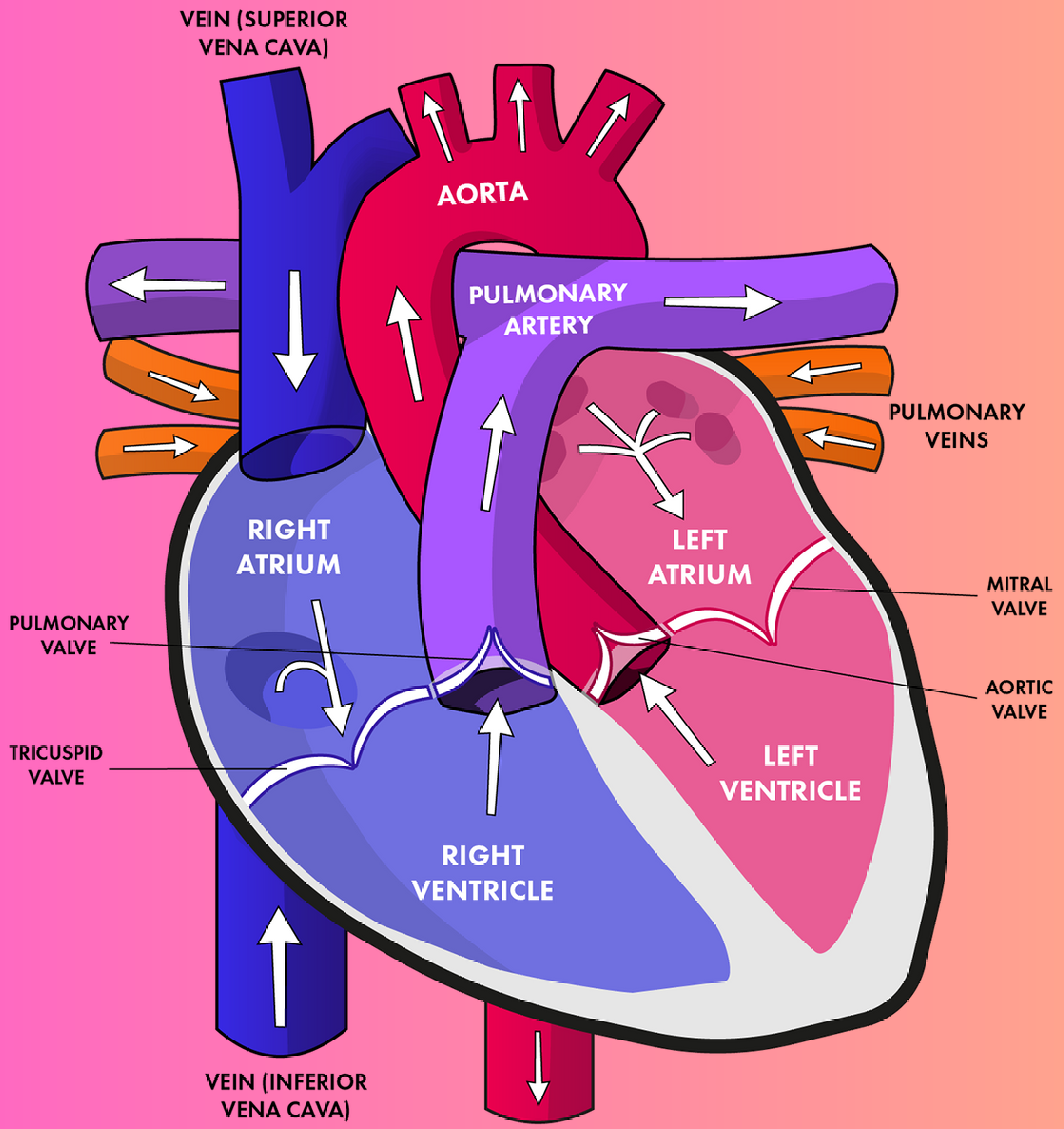


Team Project

Heart Disease Prediction

Context:

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worldwide. Four out of 5CVD deaths are due to heart attacks and strokes, and one-third of these deaths occur prematurely in people under 70 years of age. Heart failure is a common event caused by CVDs and this dataset contains 13 features that can be used to predict a possible heart disease.

People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidaemia or already established disease) need early detection and management wherein a machine learning model can be of great help.

Data Contains

- **age**- Age in Years
- **sex** - (1 = Male, 0 = Female)
- **cp** - chest pain type
- **trestbps** - resting blood pressure(in mm Hg on admission to the hospital)
- **chol** - serum cholestoral in mg/dl
- **restecg** - resting electrocardiographic results
- **thalach** - maximum heart rate achieved
- **exang** - exercise induced angina (1 = yes; 0 = no)
- **oldpeak** - ST depression induced by exercise relative to rest
- **slope** - the slope of the peak exercise ST segment
- **ca** - number of major vessels (0-3) colored by flourosopy
- **thal** - 3 = normal; 6 = fixed defect; 7 = reversable defect
- **target** - have disease or not (1=yes, 0=no)

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.cm import rainbow
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import scipy.stats as stats

#Algorithms
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, BaggingClassifier, ExtraTreesClassifier
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression

# metrics score
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, f1_score, recall_score
from sklearn.model_selection import cross_val_score

# PipeLine, GridSearch
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

# Discretization
from feature_engine.discretisation import EqualFrequencyDiscretiser, EqualWidthDiscretiser

# Scaling
from sklearn.preprocessing import StandardScaler, Normalizer, RobustScaler, OneHotEncoder, MinMaxScaler, MaxAbsScal

# Missing value imputation
from feature_engine.imputation import MeanMedianImputer, RandomSampleImputer, EndTailImputer, ArbitraryNumberImpute

# Transformation
import feature_engine.transformation as vt

# Feature-selection
from feature_engine.selection import (DropConstantFeatures,
                                      DropDuplicateFeatures,
                                      SmartCorrelatedSelection)
```

Data Information

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       294 non-null    float64
 1   sex       300 non-null    float64
 2   cp        297 non-null    float64
 3   trestbps  299 non-null    float64
 4   chol      297 non-null    float64
 5   fbs       299 non-null    float64
 6   restecg   300 non-null    float64
 7   thalach   291 non-null    float64
 8   exang     298 non-null    float64
 9   oldpeak   300 non-null    float64
 10  slope     297 non-null    float64
 11  ca        298 non-null    float64
 12  thal      300 non-null    float64
 13  target    303 non-null    int64  
dtypes: float64(13), int64(1)
memory usage: 33.3 KB
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63.0	1.0	3.0	145.0	233.0	1.0	0.0	150.0	0.0	2.3	0.0	0.0	1.0	1
1	37.0	1.0	2.0	130.0	250.0	0.0	1.0	187.0	0.0	3.5	0.0	0.0	2.0	1
2	41.0	0.0	1.0	130.0	204.0	0.0	0.0	NaN	0.0	1.4	2.0	0.0	2.0	1
3	56.0	1.0	1.0	120.0	236.0	0.0	1.0	178.0	0.0	0.8	2.0	NaN	2.0	1
4	57.0	0.0	0.0	120.0	354.0	0.0	1.0	163.0	1.0	0.6	2.0	0.0	2.0	1
5	57.0	NaN	0.0	140.0	192.0	0.0	1.0	148.0	0.0	0.4	1.0	0.0	1.0	1
6	56.0	0.0	1.0	140.0	294.0	0.0	0.0	NaN	0.0	1.3	1.0	0.0	2.0	1
7	44.0	1.0	1.0	120.0	263.0	0.0	1.0	173.0	0.0	0.0	2.0	0.0	3.0	1
8	52.0	1.0	2.0	172.0	199.0	1.0	1.0	162.0	0.0	0.5	2.0	0.0	3.0	1
9	57.0	1.0	2.0	150.0	168.0	0.0	1.0	174.0	0.0	1.6	2.0	0.0	2.0	1

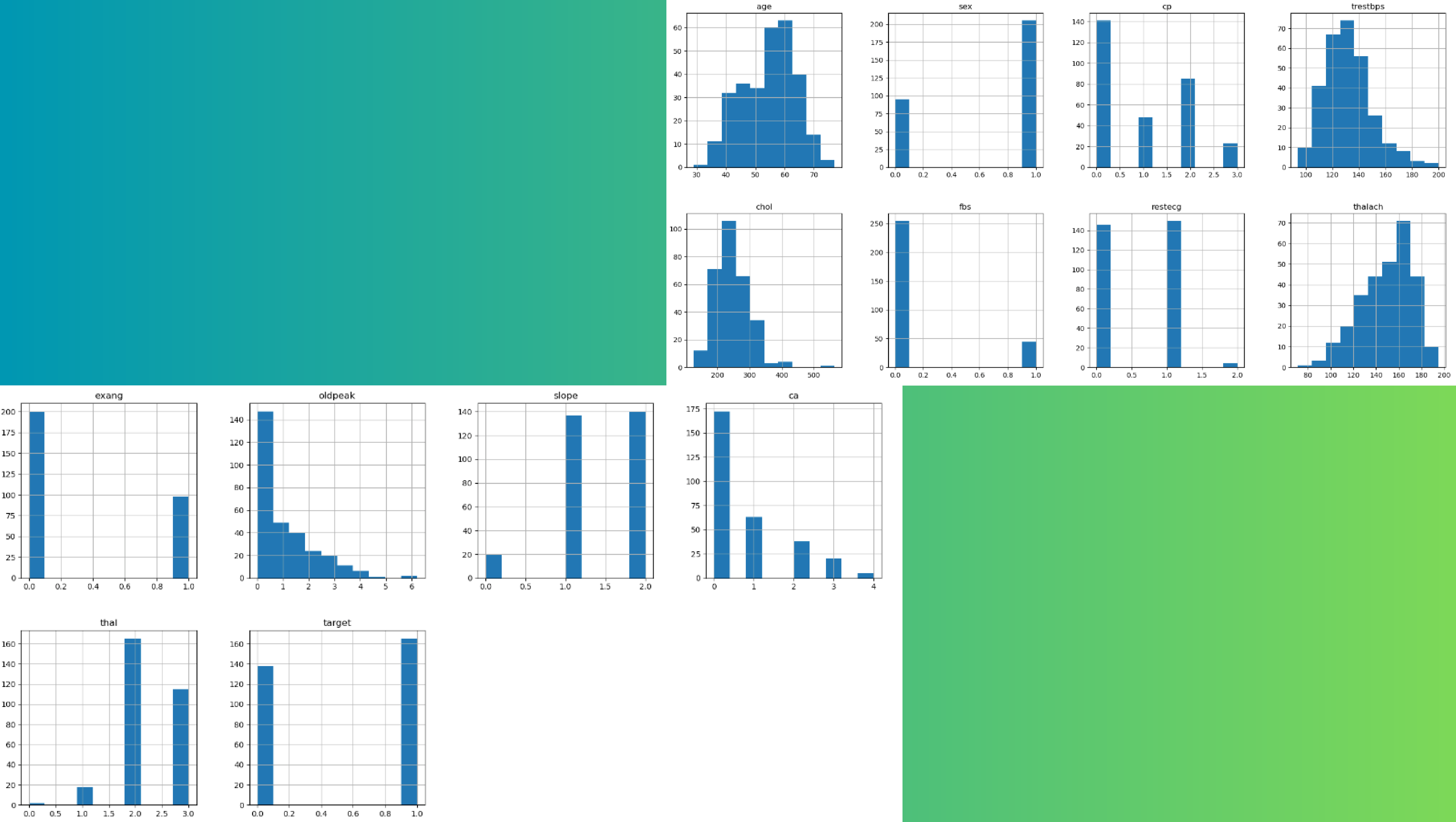
```
##### Types of Variable
dataset.dtypes
```

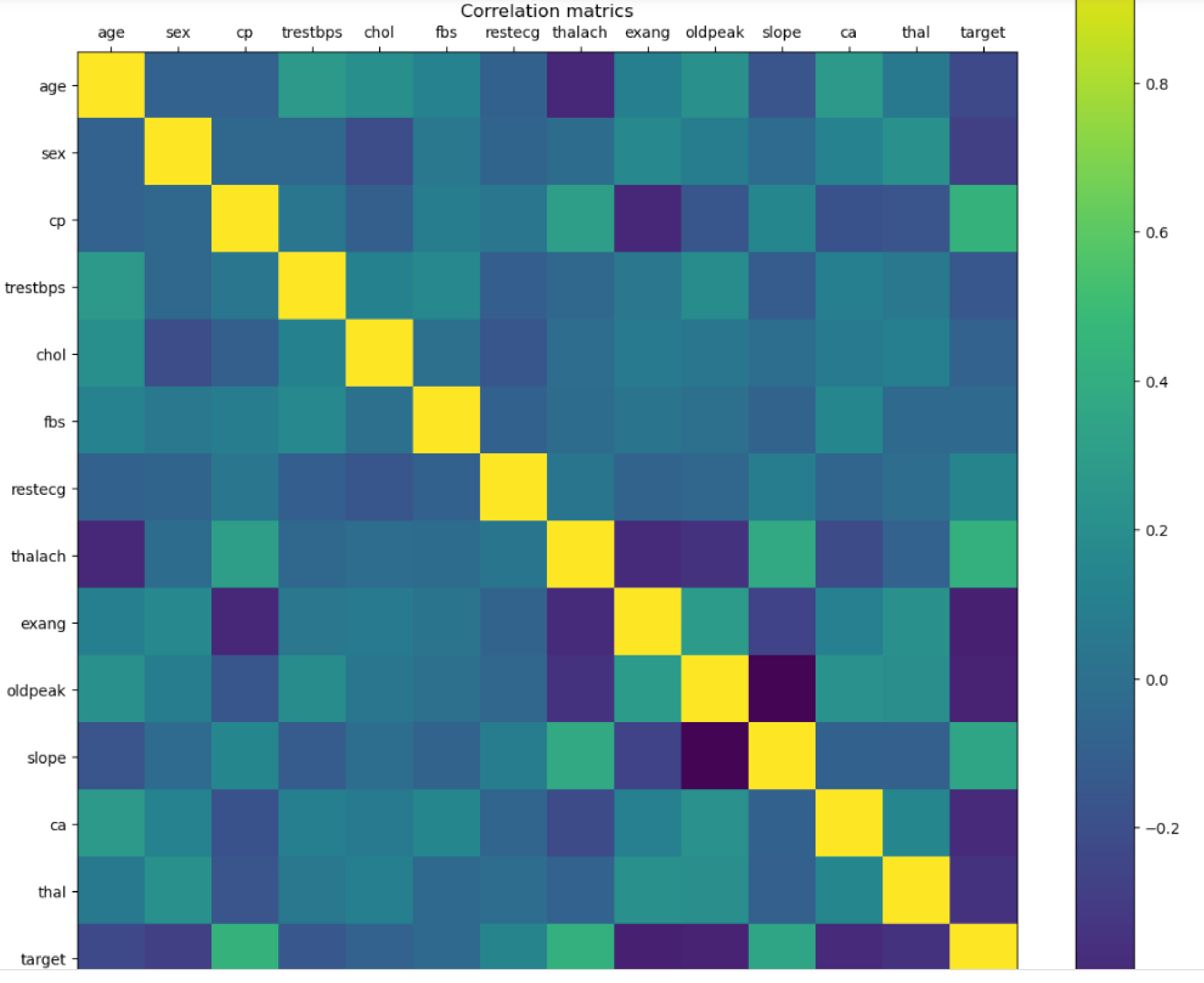
```
age         float64
sex         float64
cp          float64
trestbps    float64
chol        float64
fbs         float64
restecg     float64
thalach     float64
exang       float64
oldpeak     float64
slope       float64
ca          float64
thal        float64
target        int64
dtype: object
```

All variables in dataset numerical

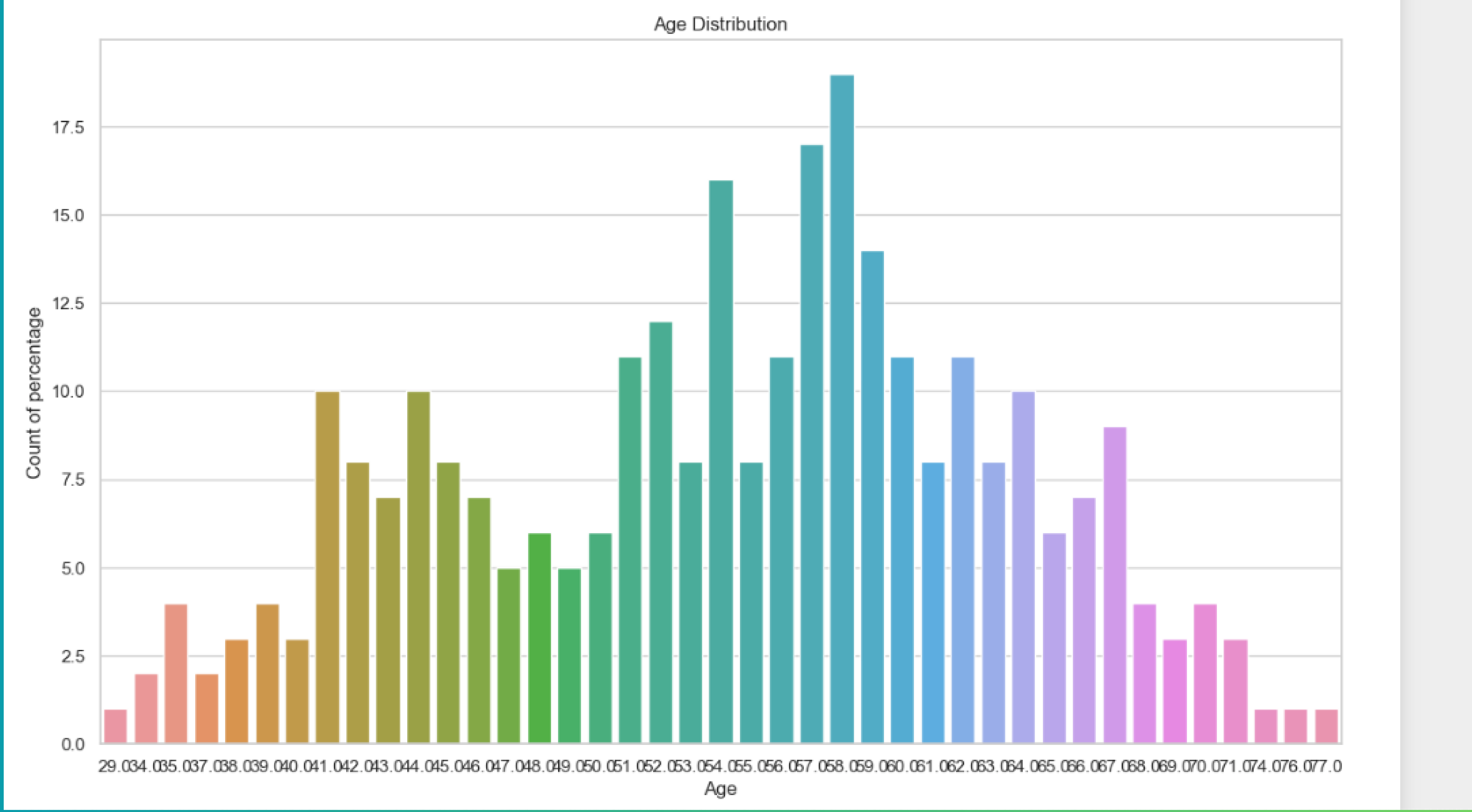
Data Describe

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	
ount	294.000000	300.000000	297.000000	299.000000	297.000000	299.000000	300.000000	291.000000	298.000000	300.000000	297.000000	298.000000	300.00
nean	54.394558	0.683333	0.966330	131.505017	246.084175	0.147157	0.526667	149.505155	0.328859	1.046000	1.404040	0.734899	2.31
std	9.106168	0.465953	1.035947	17.502516	52.016723	0.354856	0.526192	22.824574	0.470589	1.163729	0.613792	1.028315	0.61
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	47.250000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.00
50%	56.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.00
75%	61.000000	1.000000	2.000000	140.000000	274.000000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.00
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	195.000000	1.000000	6.200000	2.000000	4.000000	3.00

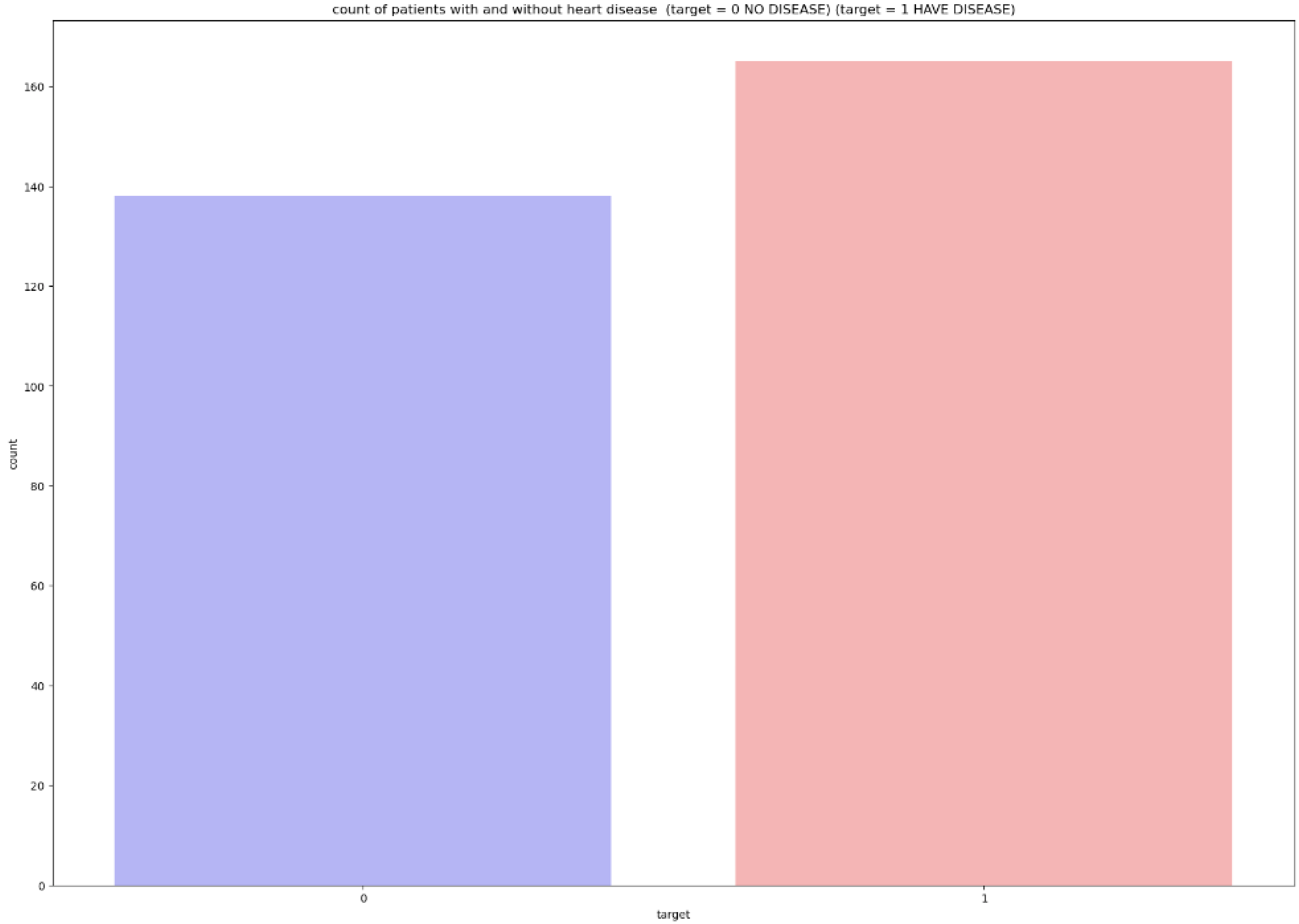

age
sex
cp
trestbps
chol
fbs
restecg
thalach
exang
oldpeak
slope
ca
thal
target



Correlation Matrics

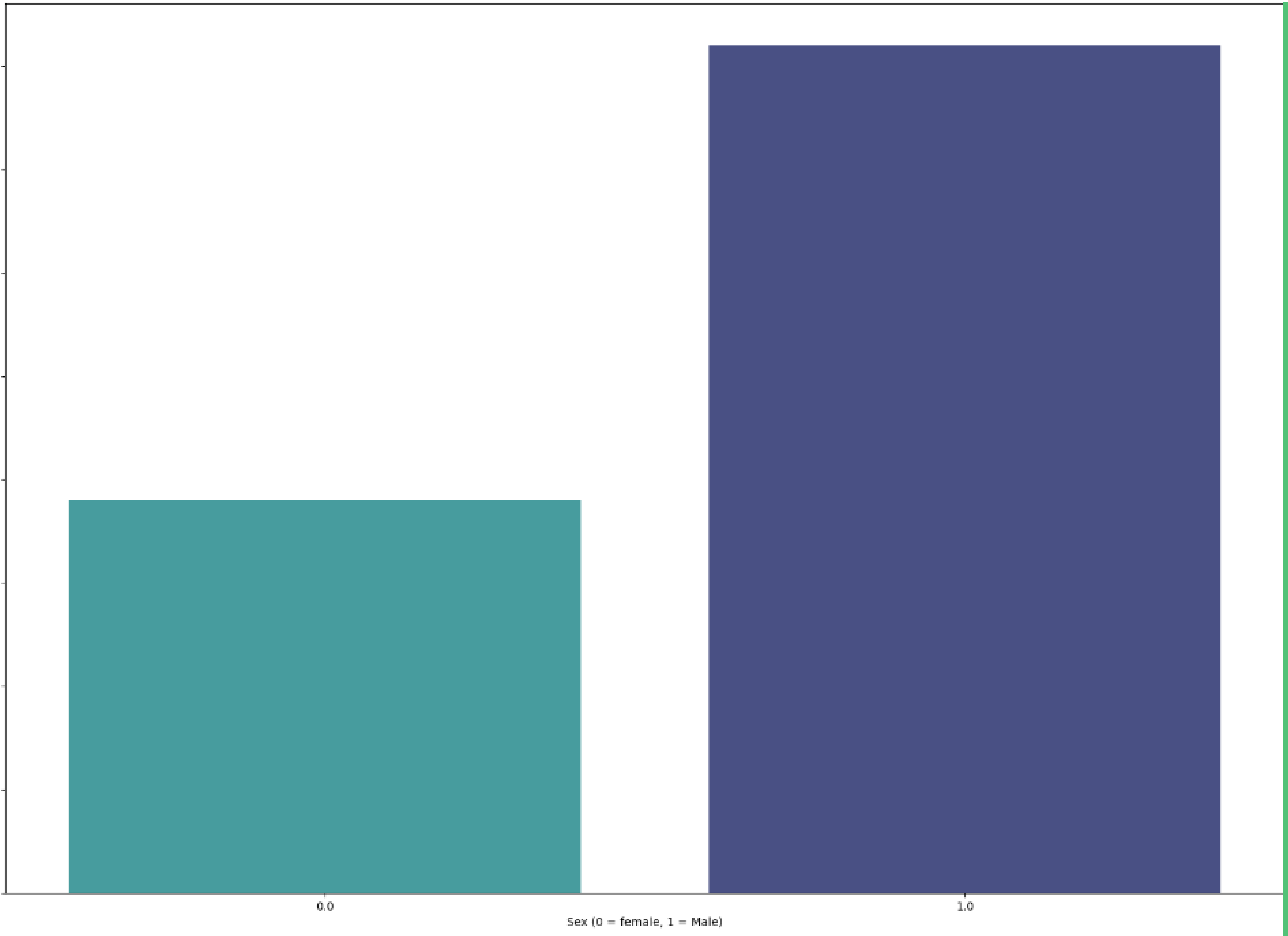


Count plot histogram for the feature "Age" to find out Age Distribution

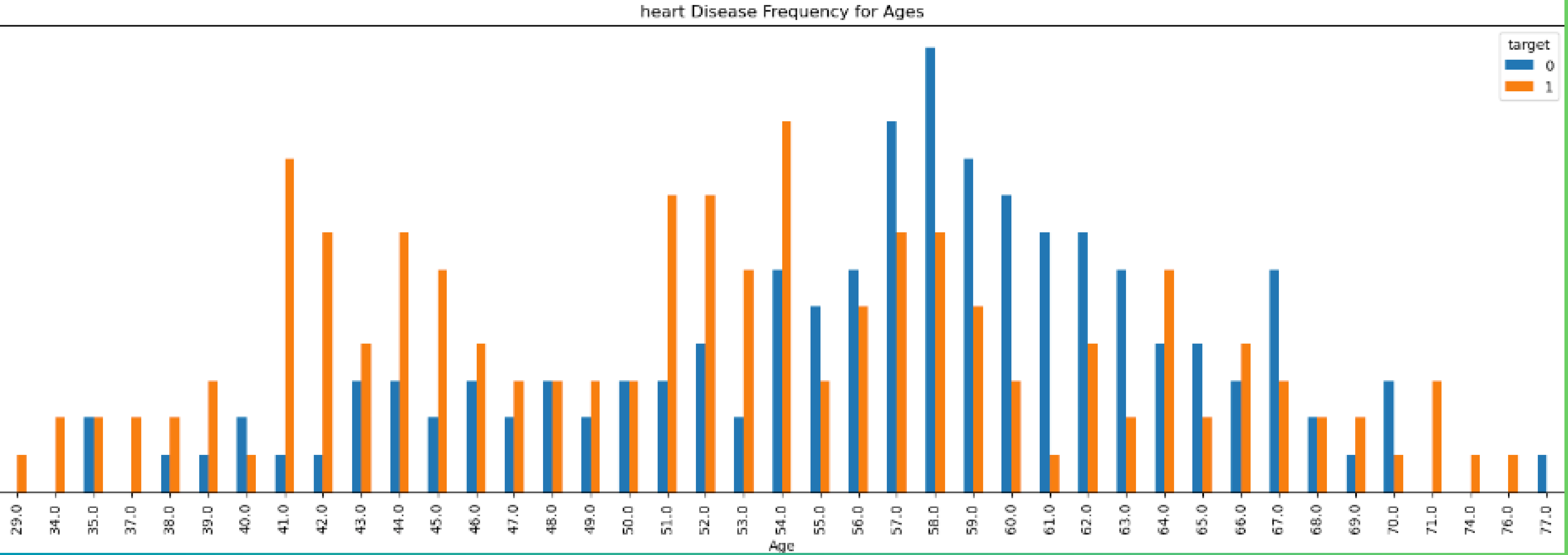


Target Count
0 = No Disease
1 = Have Disease

```
Percentage of Patients Haven't Heart Disease: 45.54%
Percentage of Patients Have Heart Disease: 54.46%
```

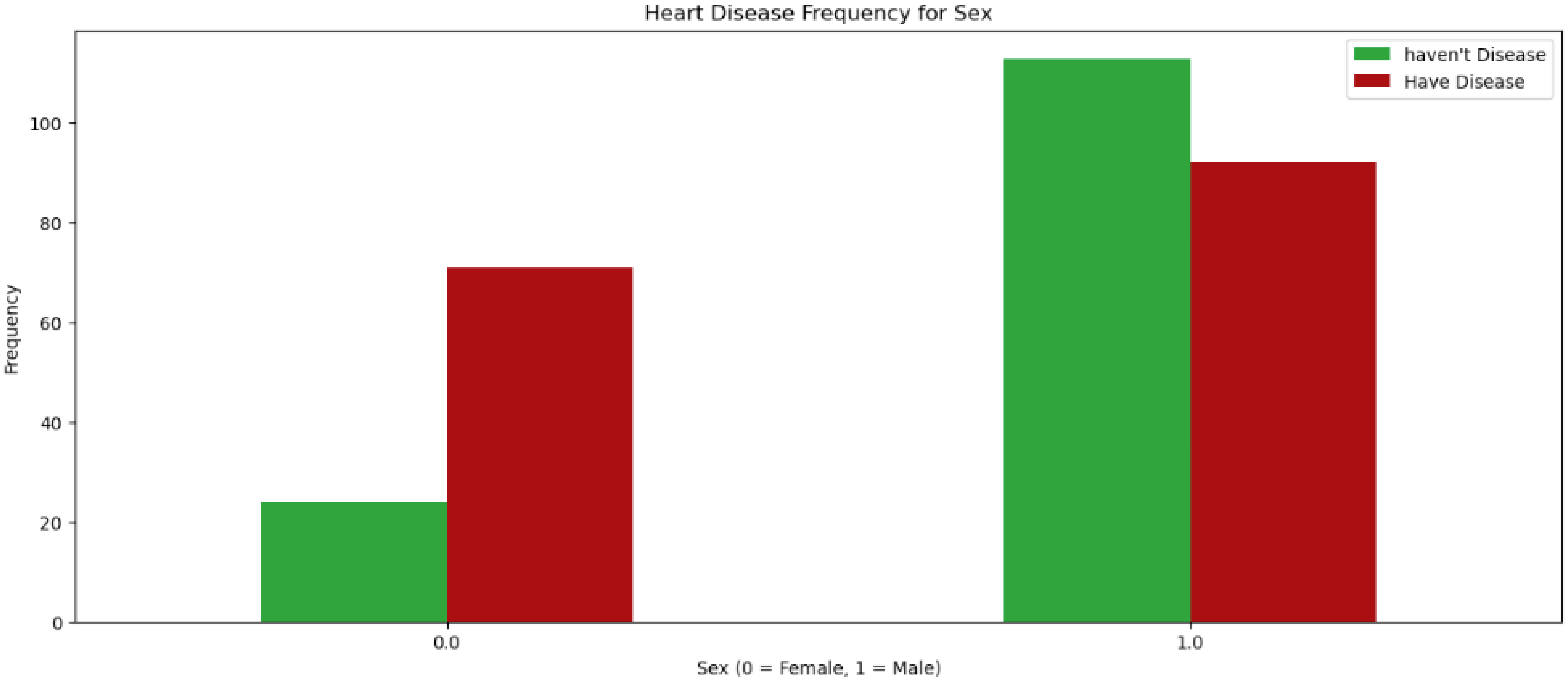


Count of male and female



In this graphs we can see the frequency of heart disease by Age

Frequency of heart by Sex



Impute Missing Values

Impute missing values:

EndTailImputer

```
endTailImpute = EndTailImputer(
    imputation_method = 'gaussian',
    tail = 'right'
)

endTailImpute.fit(X_train)
```

```
EndTailImputer()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

EndTail Imputer

```
endTailImpute.variables_
```

```
['age',
 'sex',
 'cp',
 'trestbps',
 'chol',
 'fbs',
 'restecg',
 'thalach',
 'exang',
 'oldpeak',
 'slope',
 'ca',
 'thal']
```

```
endTailImpute.imputer_dict_
```

```
{'age': 82.18659080810232,
 'sex': 2.0826960696630894,
 'cp': 4.009833273341641,
 'trestbps': 186.22514160368743,
 'chol': 402.3367006620612,
 'fbs': 1.228433700824096,
 'restecg': 2.082977941606012,
 'thalach': 216.35208813034708,
 'exang': 1.7843474629507259,
 'oldpeak': 4.542578580266698,
 'slope': 3.2423135429602743,
 'ca': 3.6070212966202324,
 'thal': 4.177213690998737}
```

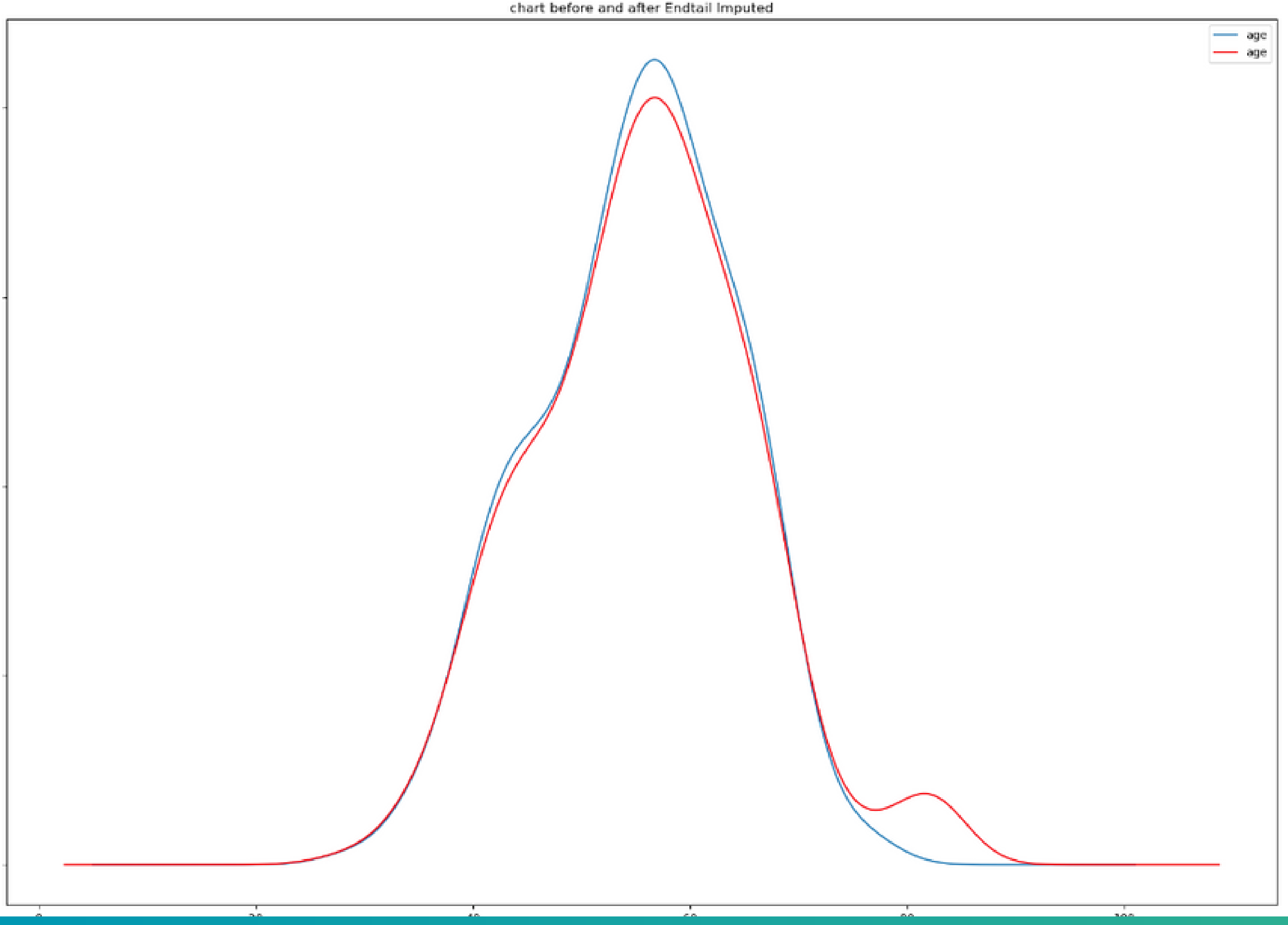
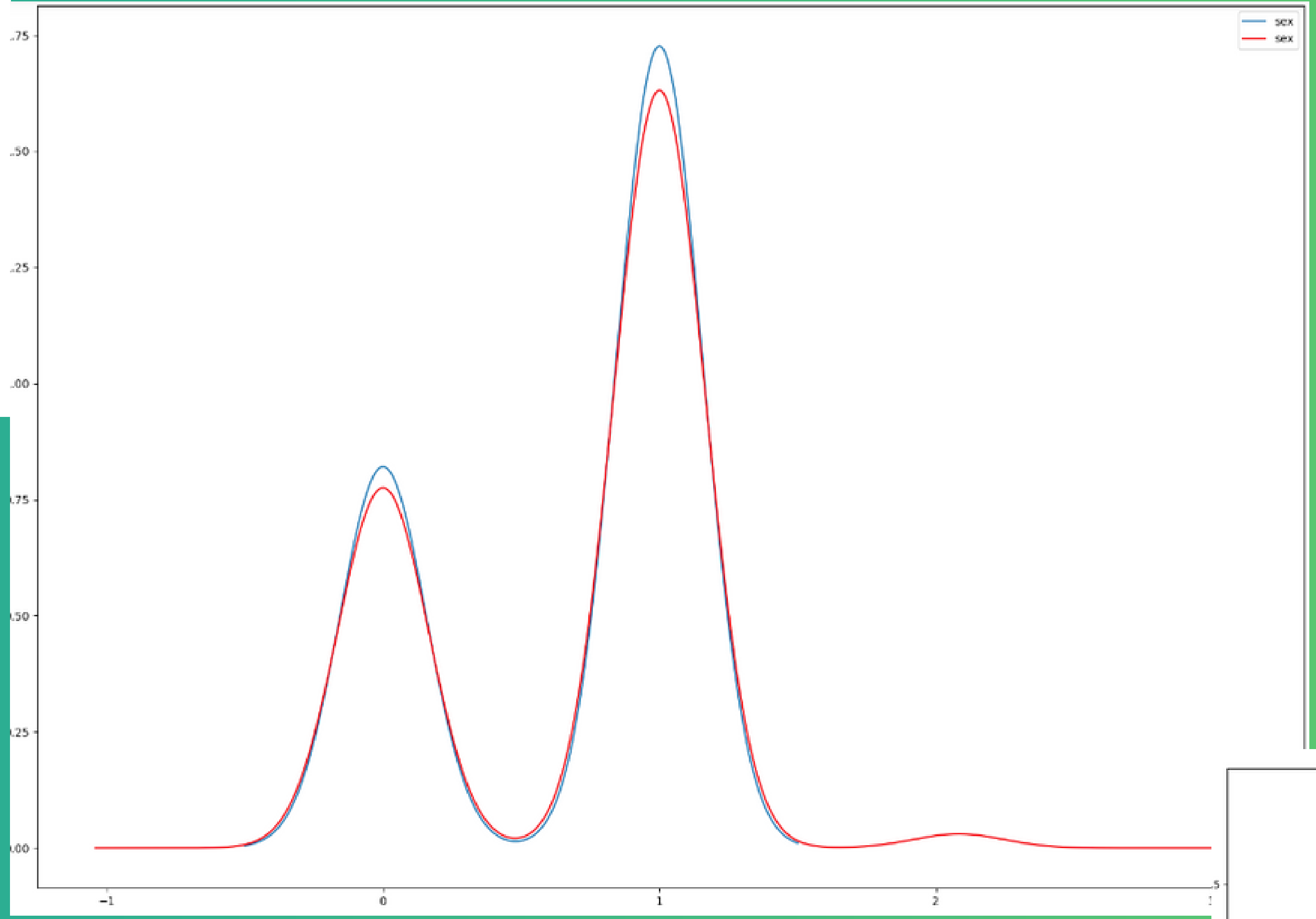


chart before and after Endtail Imputed
age
age


sex
sex

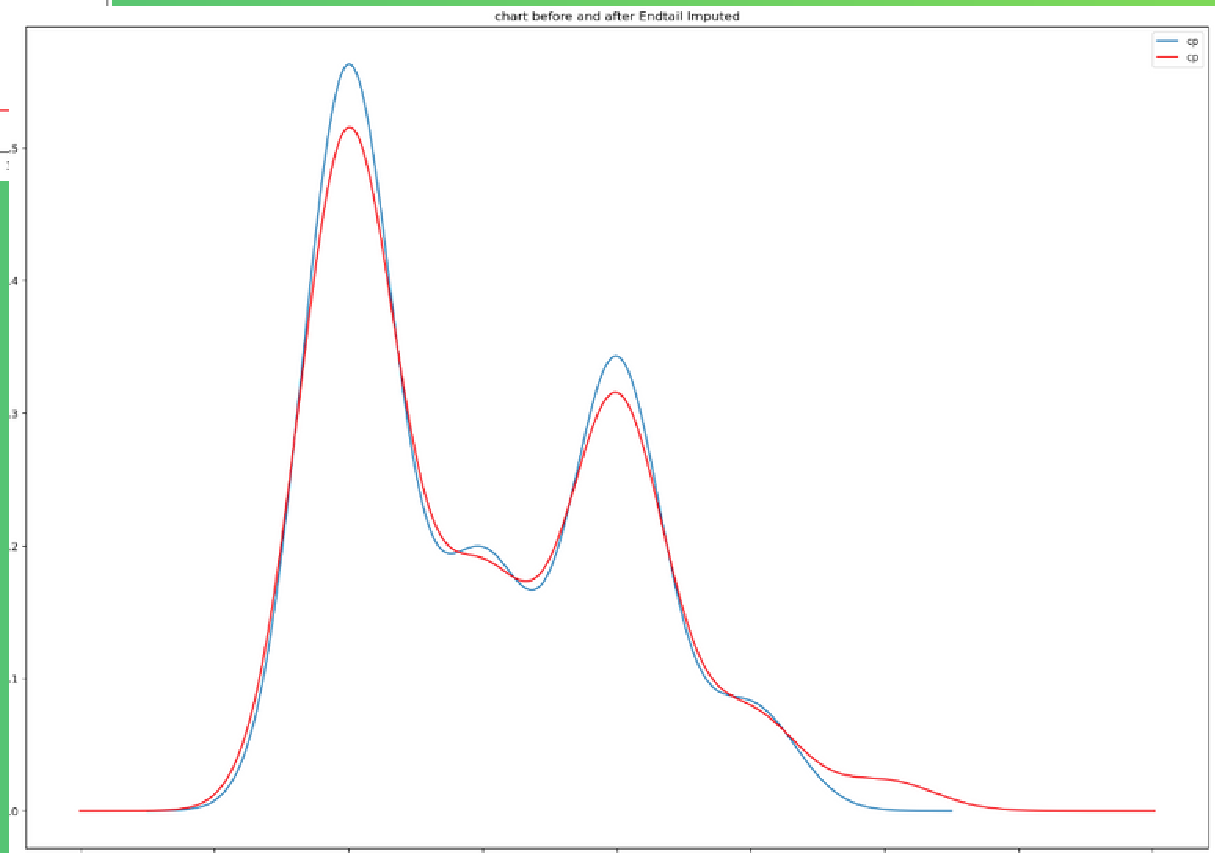


chart before and after Endtail Imputed
cp
cp

Arbitrary Number Imputer

```python
imputer = ArbitraryNumberImputer(imputer_dict = {
    'age': 99,
    'sex': -1,
    'cp': -1,
    'trestbps': 99,
    'chol': 222,
    'fbs': -1,
    'restecg': -1,
    'thalach': 99,
    'exang': -1,
    'oldpeak': -1,
    'slope': -1,
    'ca': 2,
    'thal': 3
})
imputer.fit(X_train)
```

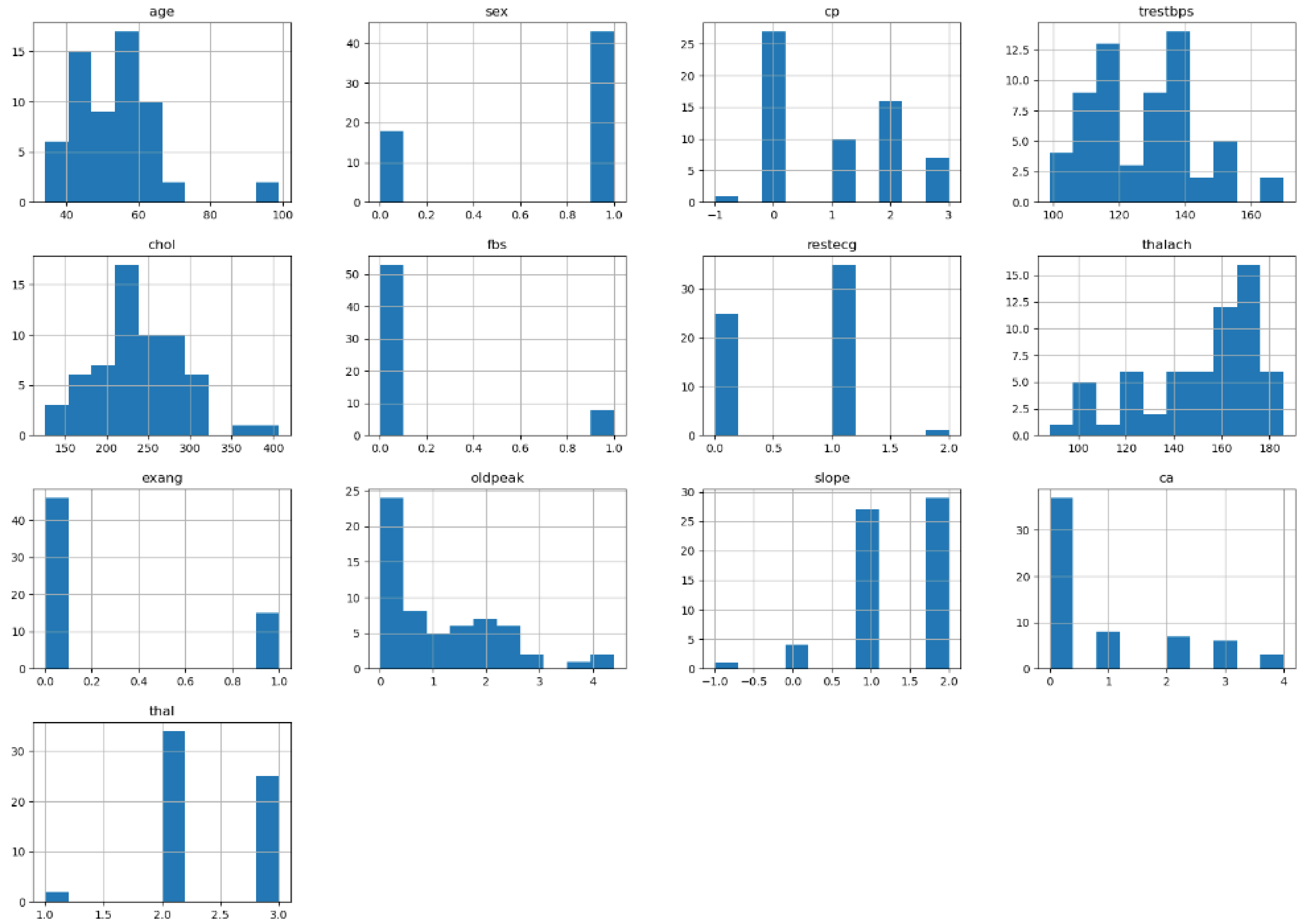
Arbitrary Number Imputer

```python
imputer.imputer_dict_
```

```
{'age': 99,
 'sex': -1,
 'cp': -1,
 'trestbps': 99,
 'chol': 222,
 'fbs': -1,
 'restecg': -1,
 'thalach': 99,
 'exang': -1,
 'oldpeak': -1,
 'slope': -1,
 'ca': 2,
 'thal': 3}
```

```python
tmp_arbTrain = imputer.transform(X_train)
tmp_arbTest = imputer.transform(X_test)
tmp_arbTrain[imputer.variables_].isnull().mean()
```

```
age         0.0
sex         0.0
cp          0.0
trestbps    0.0
chol        0.0
fbs         0.0
restecg     0.0
thalach     0.0
exang       0.0
oldpeak     0.0
slope       0.0
ca          0.0
thal        0.0
dtype: float64
```



Histogram after Arbitrary Number Imputer

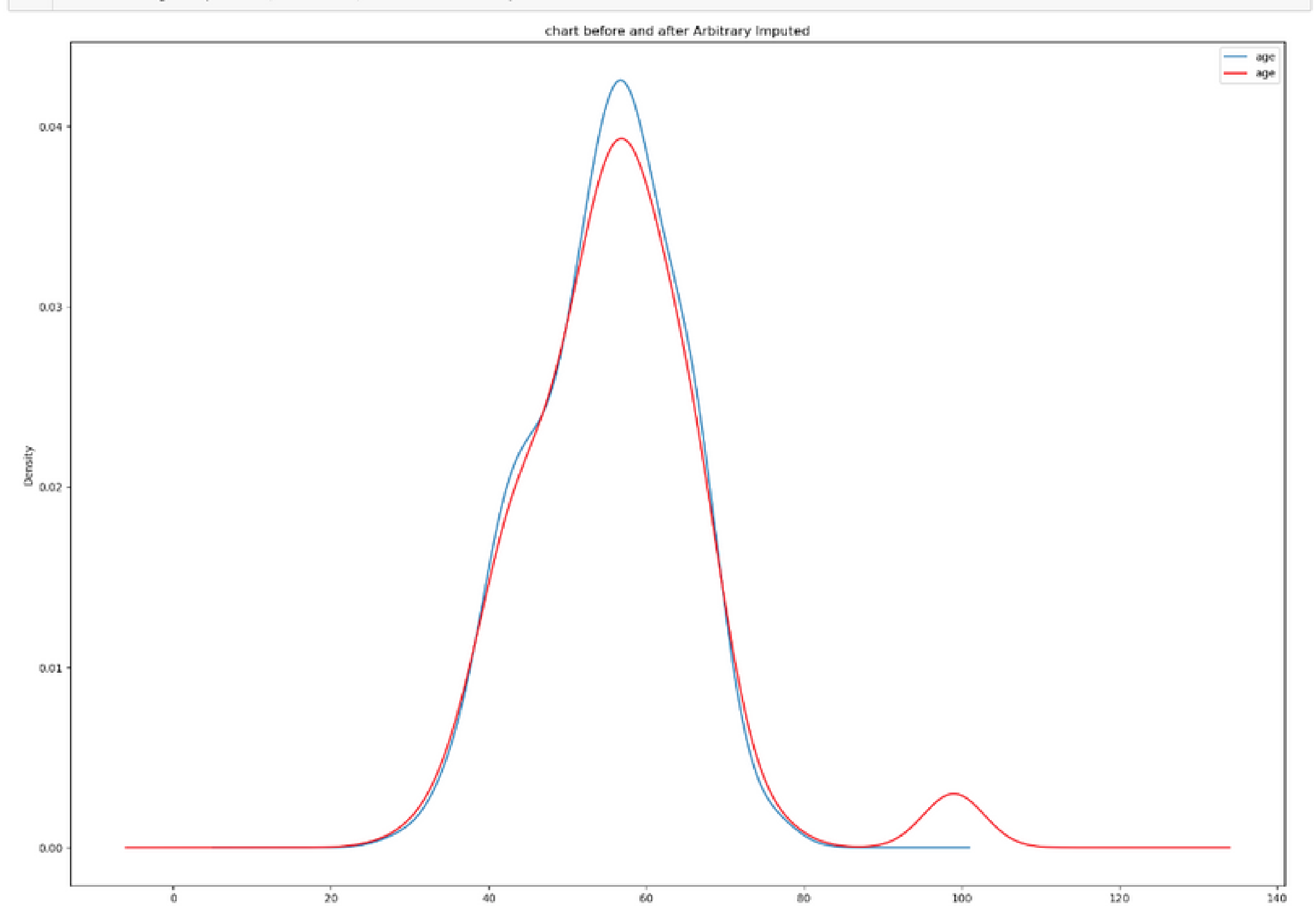


chart before and after Arbitrary Imputed
age
age
Density
0.00
0.01
0.02
0.03
0.04
0
20
40
60
80
100
120
140

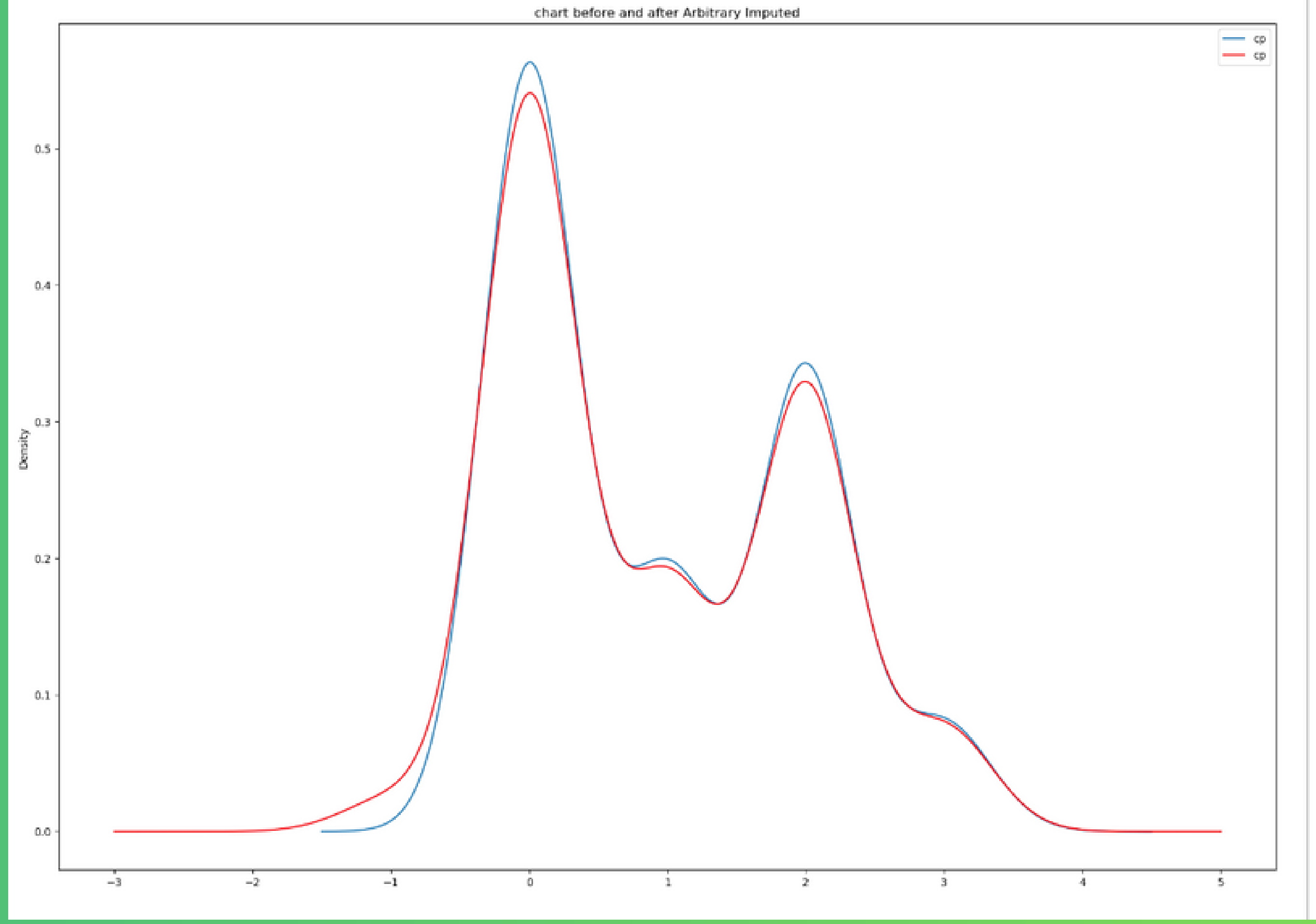
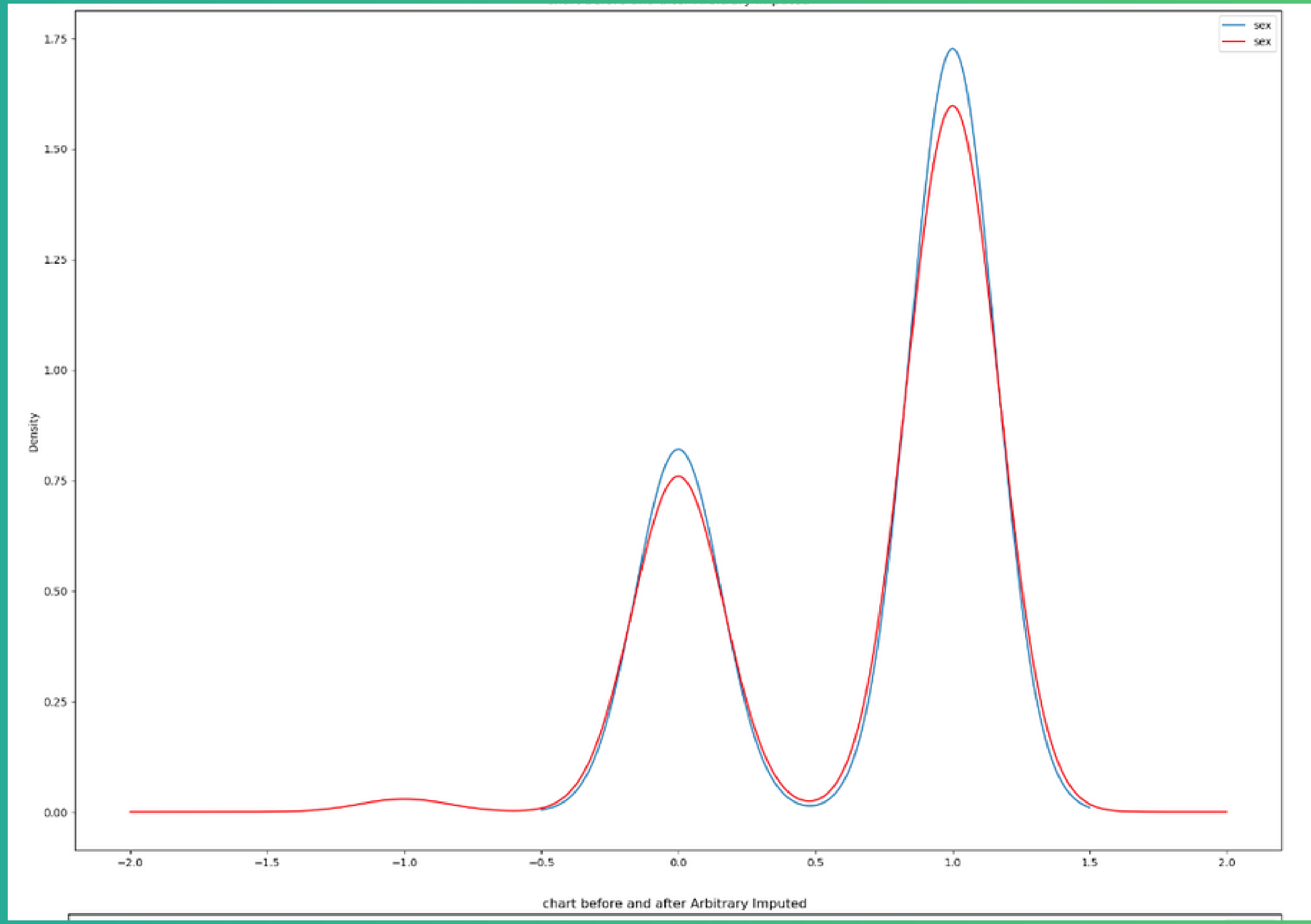


chart before and after Arbitrary Imputed
cp
cp
Density
0.0
0.1
0.2
0.3
0.4
0.5
−3
−2
−1
0
1
2
3
4
5


sex
sex
Density
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
chart before and after Arbitrary Imputed

Transformation with:

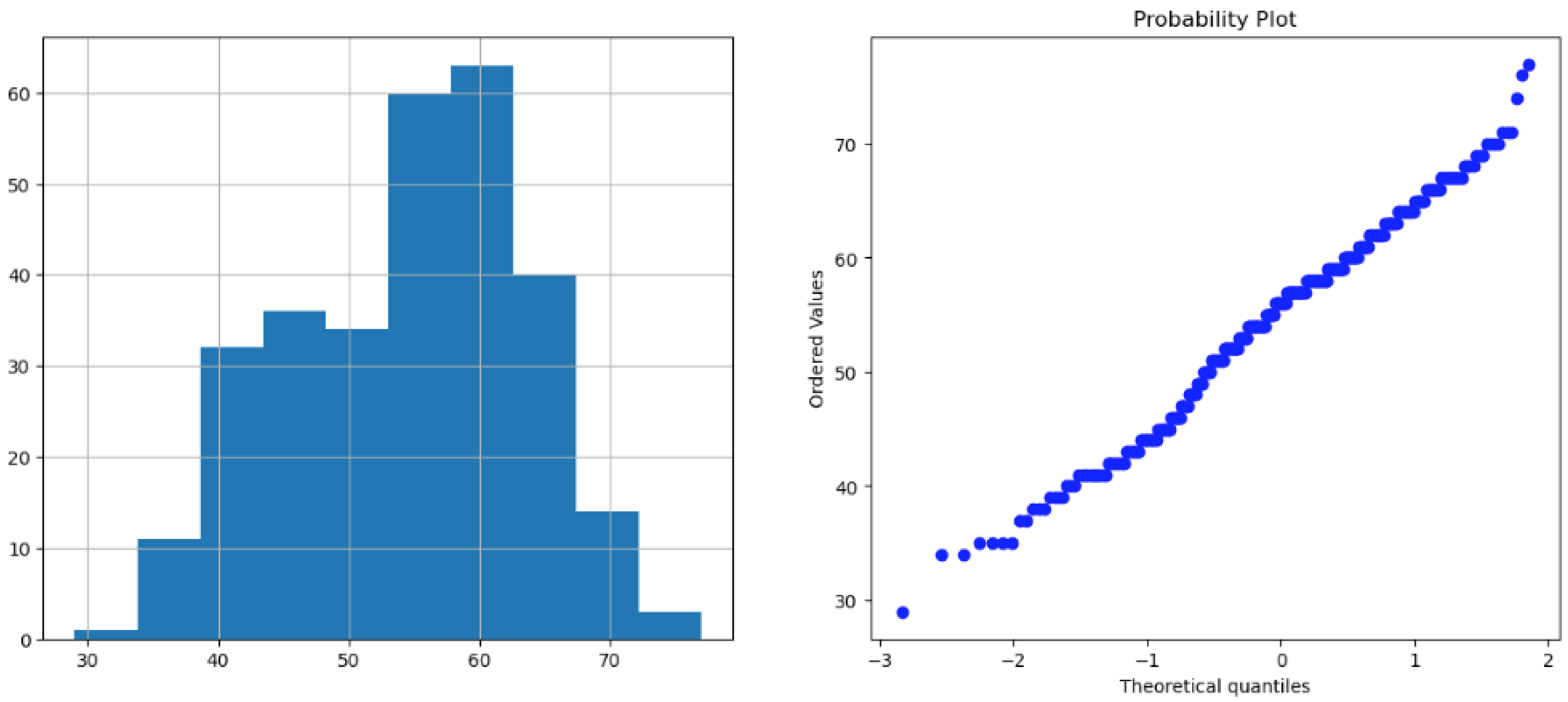
Log Transformer

Exponential Transformer

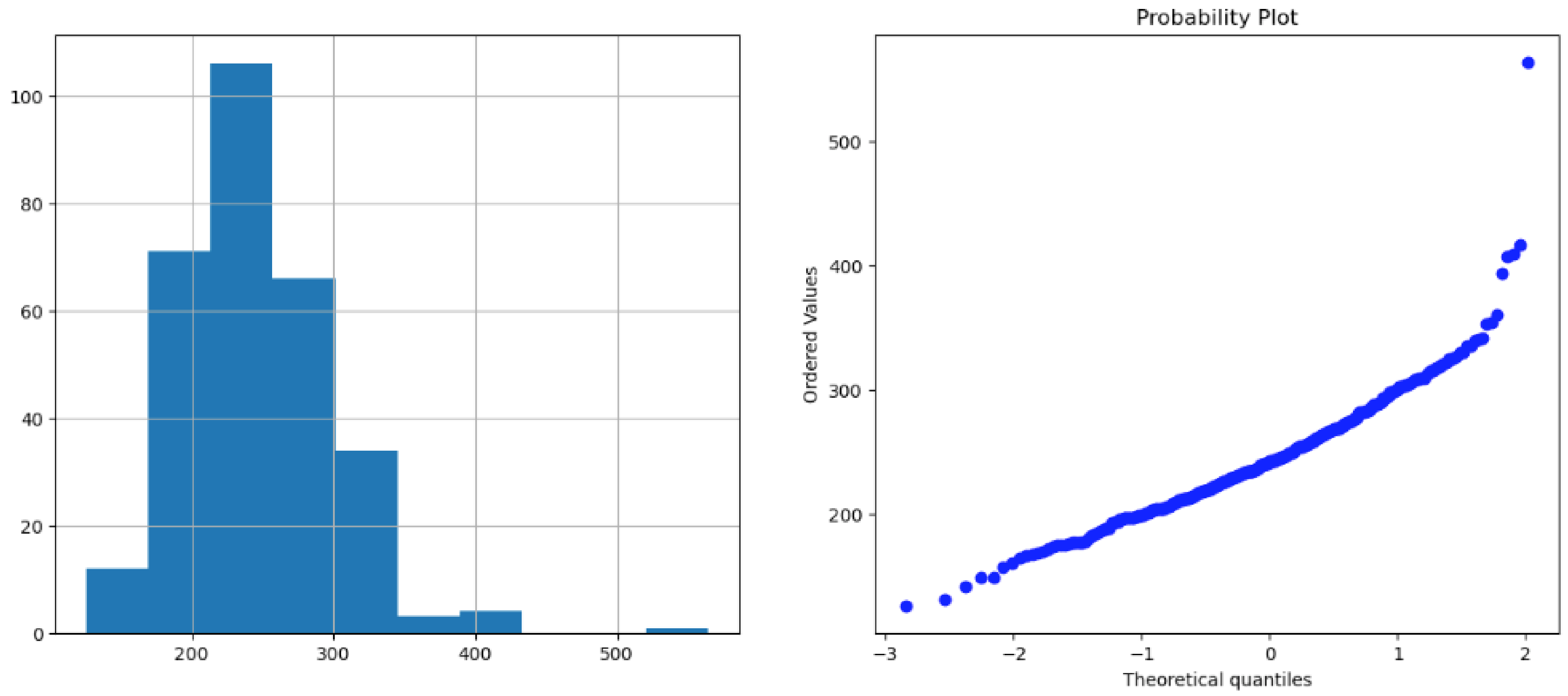
Yeo-Johnson Transformer

```python
def diagnostics_plots(dataset, variable):


    plt.figure(figsize = (15, 6))

    plt.subplot(1, 2, 1)
    dataset[variable].hist()

    plt.subplot(1, 2, 2)
    stats.probplot(dataset[variable], dist = 'norm', plot = plt)

    plt.show()
```

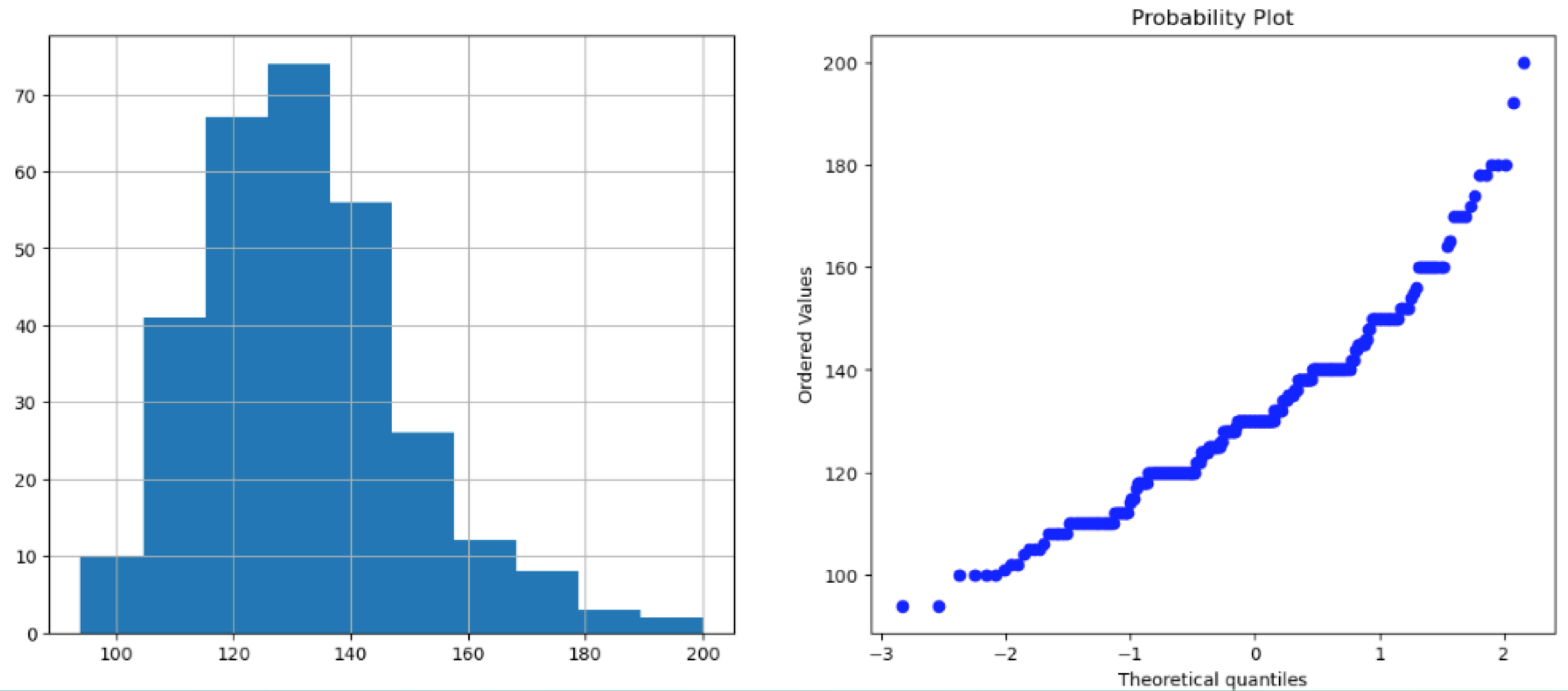
```python
diagnostics_plots(dataset, 'age')
```




```python
diagnostics_plots(dataset, 'chol')
```




```python
diagnostics_plots(dataset, 'trestbps')
```



Before Transformation

Log Transfomer

```
# Log Transform
lt = vt.LogCpTransformer(variables = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak'])
lt.fit(imputer_MeanMedian)
```

```
LogCpTransformer(variables=['age', 'trestbps', 'chol', 'thalach', 'oldpeak'])
```

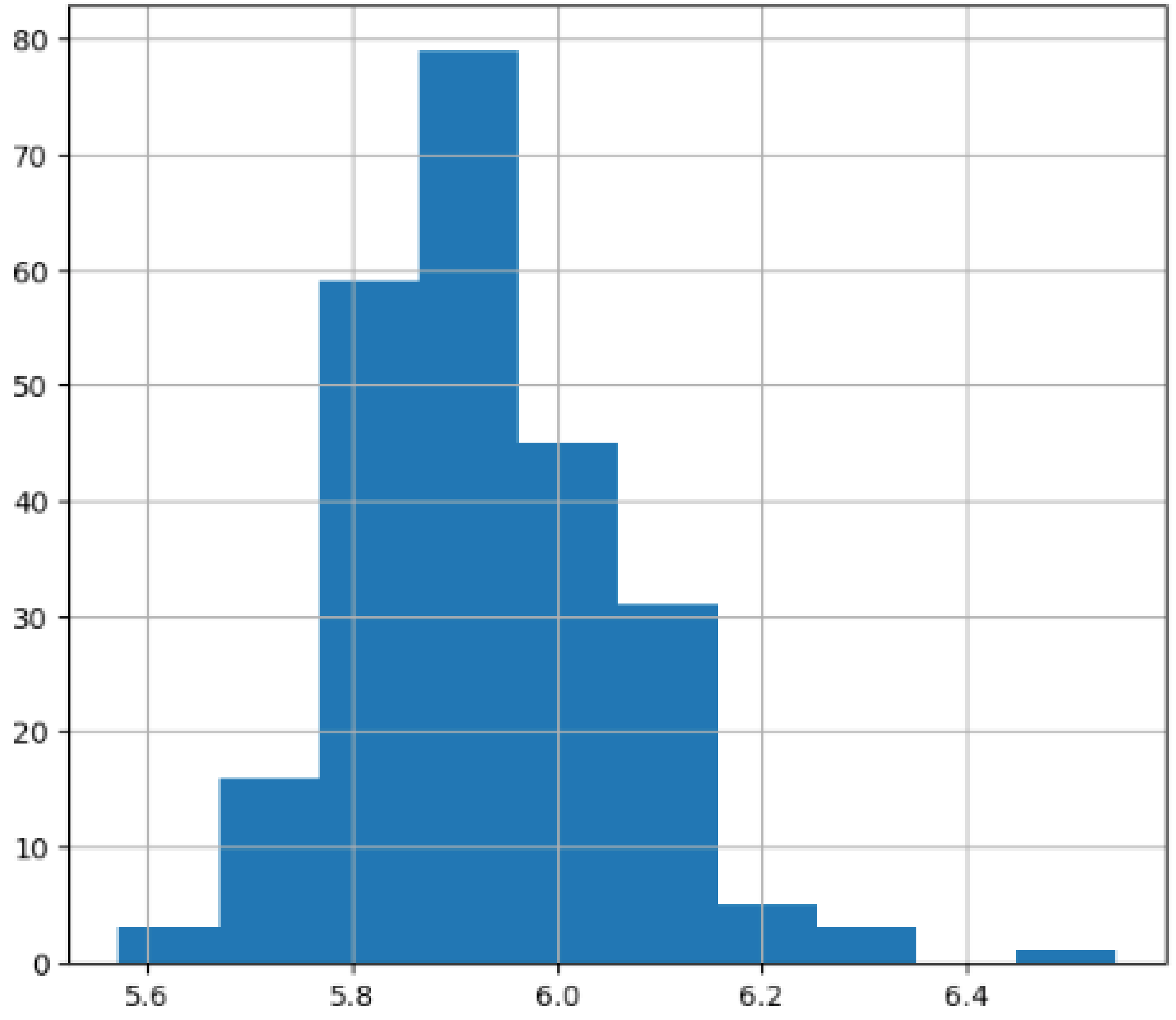
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

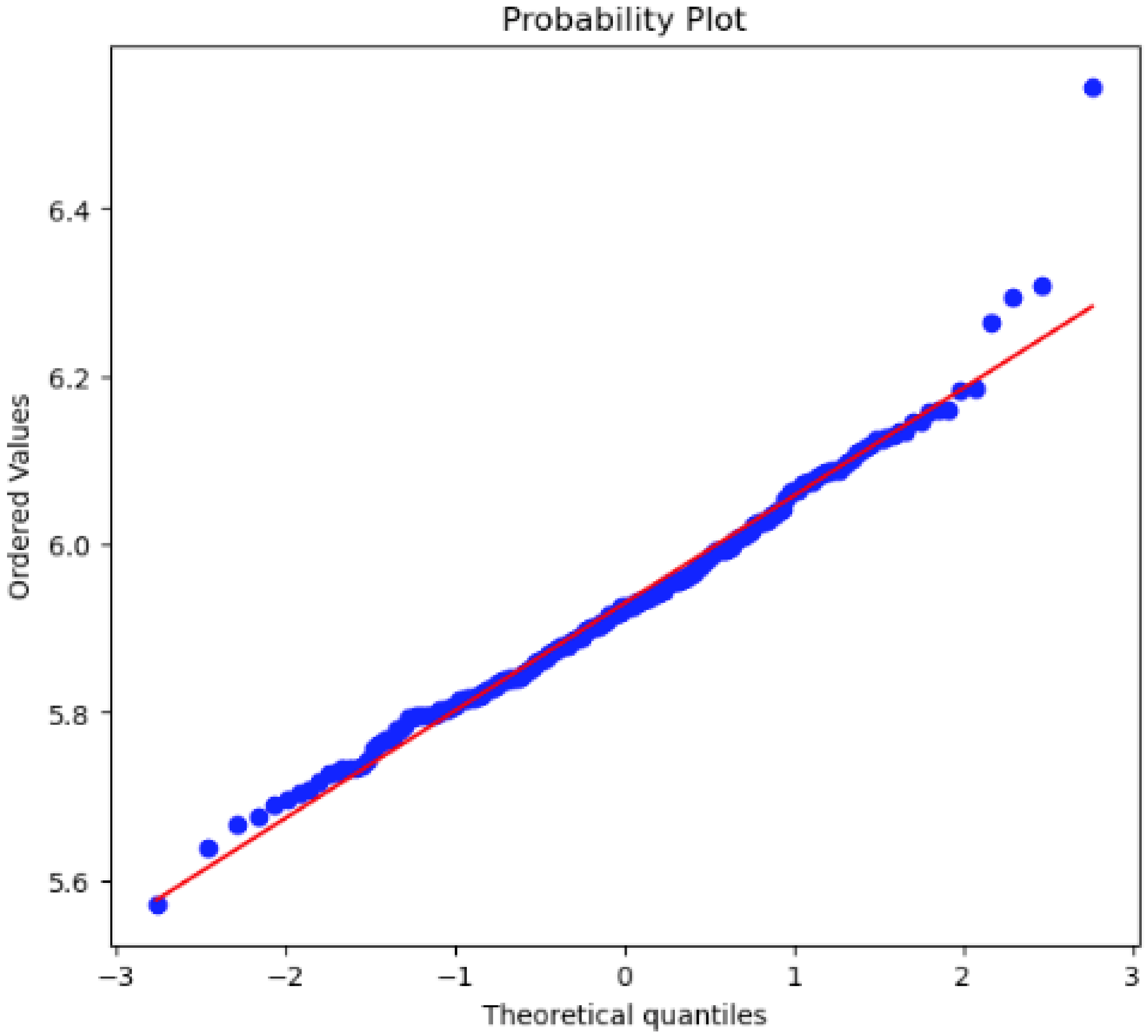
```
lt.variables_
```

```
['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
```

```
data_tf = lt.transform(imputer_MeanMedian)
```

```
diagnostics_plots(data_tf, 'chol')
```





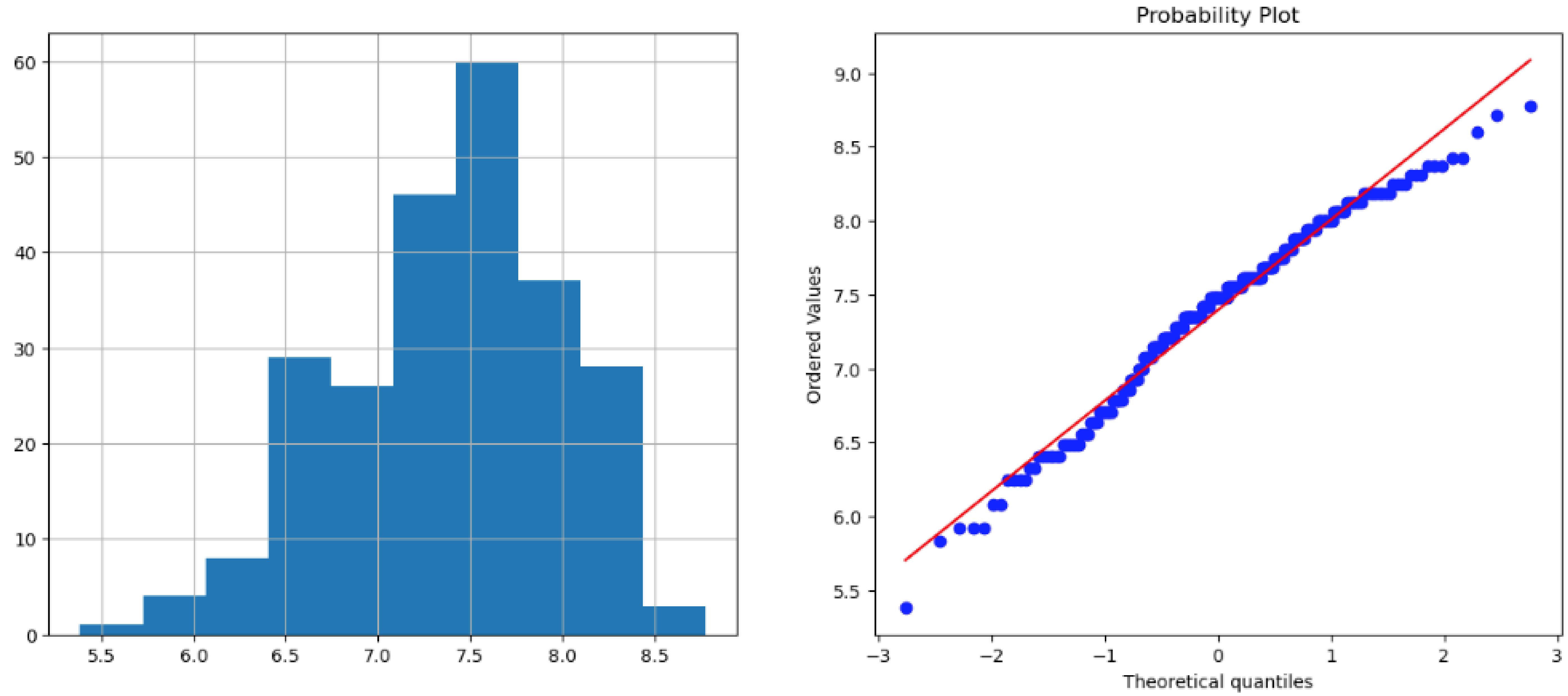
```
#Exponential Transofrmer
et = vt.PowerTransformer(variables = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak'])
et.fit(imputer_MeanMedian)
```

```
PowerTransformer(variables=['age', 'trestbps', 'chol', 'thalach', 'oldpeak'])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
data_tf = et.transform(imputer_MeanMedian)
```

```
diagnostics_plots(data_tf, 'age')
```



Exponential Transformer

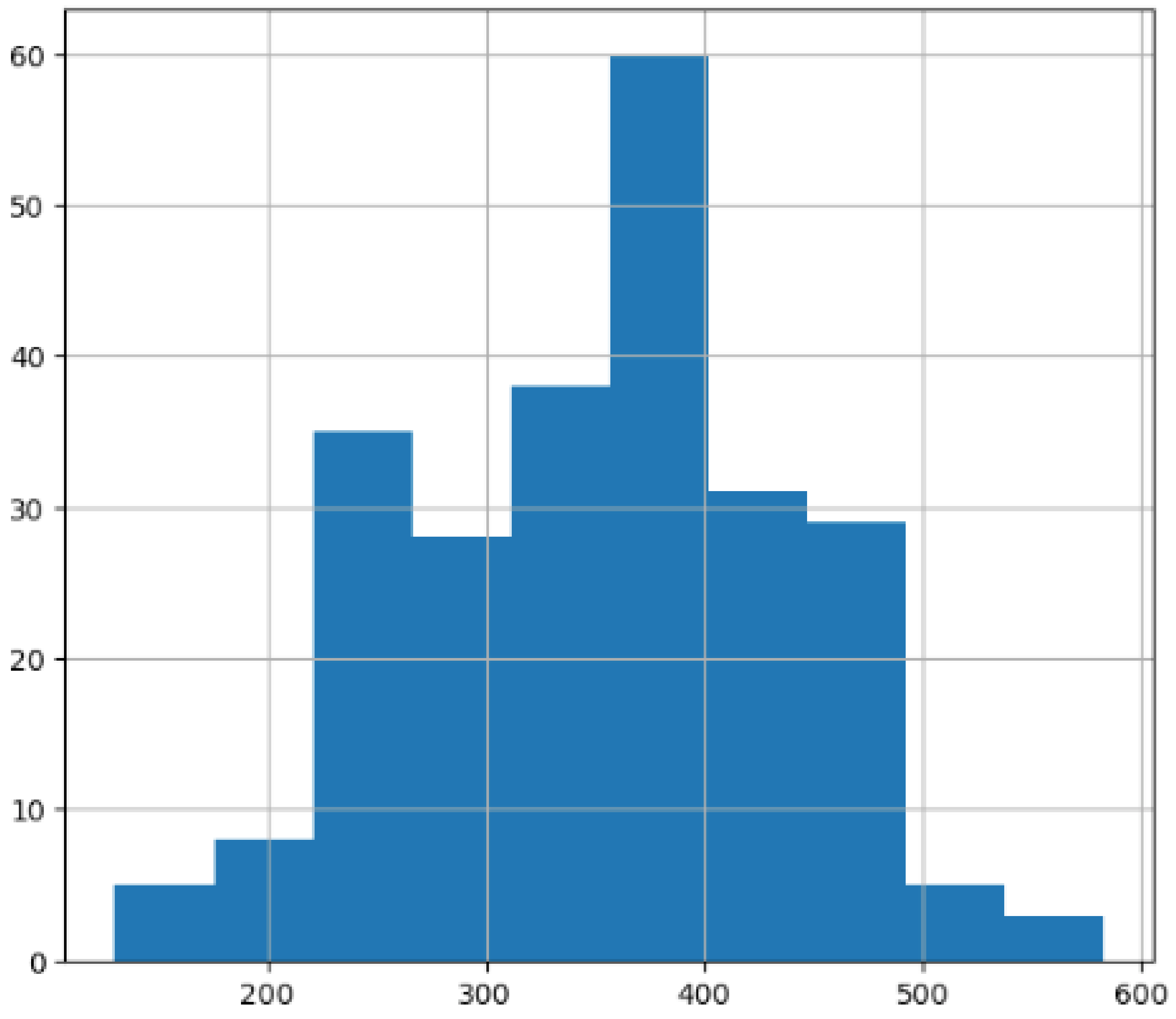
```
#Yeo-Johnson
yjt = vt.YeoJohnsonTransformer(variables = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak'])
yjt.fit(imputer_MeanMedian)
```

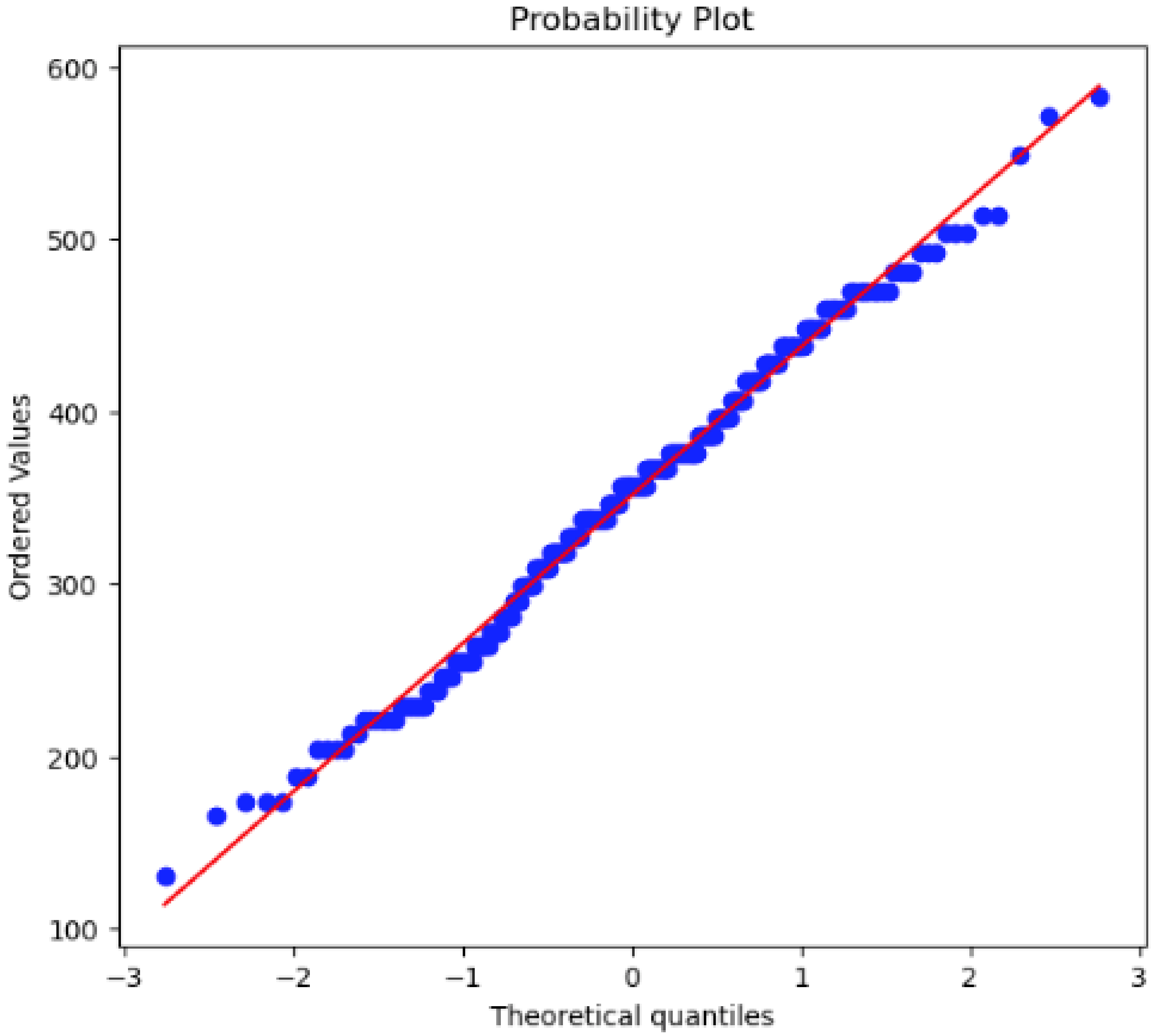
```
YeoJohnsonTransformer(variables=['age', 'trestbps', 'chol', 'thalach',
                                 'oldpeak'])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
data_tf = yjt.transform(imputer_MeanMedian)
```

```
diagnostics_plots(data_tf, 'age')
```





Yeo-Johnson Transformer

Scaling with Unit Norm, Robust Scaler

```
scaler = Normalizer(norm = 'l2')

scaler.fit(imputer_MeanMedian)

X_train_scaled = scaler.transform(imputer_MeanMedian)
X_test_scaled = scaler.transform(imputer_MeanMedianTest)
```

```
np.round(np.linalg.norm(imputer_MeanMedian, ord = 2, axis = 1), 1)
```

```
ray([332. , 406.3, 323. , 470.4, 323.9, 296.7, 271.8, 364.8, 411.8,
     296.6, 333.4, 315.1, 308.4, 364.9, 325.2, 395.4, 291.6, 255.8,
     309.9, 301.5, 317.4, 359.3, 363.8, 323.1, 297.1, 224.8, 295.8,
     364.4, 304.8, 289.1, 333.5, 331.2, 304.6, 288.5, 325.1, 366. ,
     379.2, 338.3, 321.4, 324.8, 319.6, 283. , 363.6, 383.1, 328.6,
     450.9, 288.5, 290.5, 262.9, 281.2, 291.2, 341.4, 339.1, 325.8,
     284.9, 339.5, 317.9, 367.6, 392.5, 374.5, 345.3, 329.5, 299.1,
     281.3, 307.6, 370.5, 324.4, 383.7, 255.8, 351.5, 336.3, 320.5,
     317.8, 322.8, 359.8, 324.4, 331.2, 373.3, 297. , 337.3, 261.4,
     295.5, 364.5, 338.3, 327.5, 298.9, 340.2, 340.9, 334.1, 331.5,
     296.9, 303.4, 281.6, 280.3, 299.7, 397.8, 281.5, 309.8, 327.4,
     325.3, 384.8, 263.6, 326.9, 304.3, 339.8, 296.1, 283.5, 320.9,
     274.7, 322.3, 324.9, 298.9, 267.4, 331.7, 334.1, 347.7, 279.9,
     256.2, 309.4, 300.6, 396.6, 340.9, 375.2, 278.2, 306.2, 317.3,
     339. , 322.1, 281.6, 320.6, 395.1, 403.1, 377.1, 346.5, 328.4,
     326.3, 318.6, 327.1, 333.3, 261.3, 299.6, 316.9, 308.3, 401.1,
     329.8, 363.5, 309.2, 331.1, 291.9, 371.2, 287.9, 242.7, 326.6,
     406.2, 355.1, 267.6, 459.2, 373.2, 233.3, 308.9, 358.6, 319.8,
     304.2, 345.7, 376.2, 304. , 302.5, 382.5, 347.8, 284. , 331.6,
```

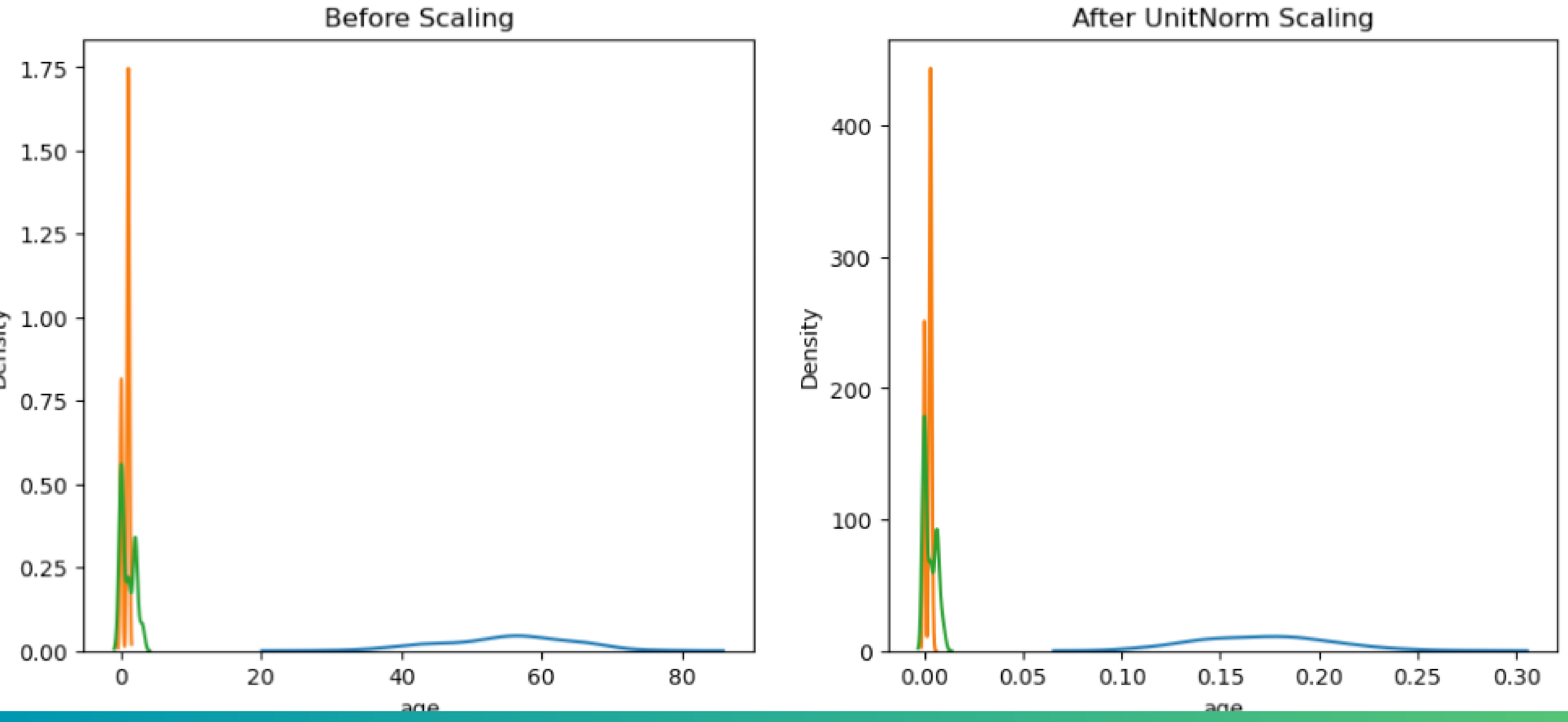
```
X_train_scaled = pd.DataFrame(X_train_scaled, columns = imputer_MeanMedian.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns = imputer_MeanMedian.columns)
```

```
X_train_scaled
```

UNIT NORM SCALER

```
fig, (ax1, ax2) = plt.subplots(ncols = 2, figsize = (12, 5))
#before scaling
ax1.set_title('Before Scaling')
sns.kdeplot(imputer_MeanMedian['age'], ax = ax1)
sns.kdeplot(imputer_MeanMedian['sex'], ax = ax1)
sns.kdeplot(imputer_MeanMedian['cp'], ax = ax1)

#after scaling
ax2.set_title('After UnitNorm Scaling')
sns.kdeplot(X_train_scaled['age'], ax = ax2)
sns.kdeplot(X_train_scaled['sex'], ax = ax2)
sns.kdeplot(X_train_scaled['cp'], ax = ax2)
```

```
xesSubplot: title={'center': 'After UnitNorm Scaling'}, xlabel='age', ylabel='Density'>
```



Robust Scaler

```
scaler = RobustScaler()

scaler.fit(imputer_MeanMedian)

X_train_scaled = scaler.transform(imputer_MeanMedian)
X_test_scaled = scaler.transform(imputer_MeanMedianTest)
```

```
X_train_scaled = pd.DataFrame(X_train_scaled, columns = imputer_MeanMedian.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns = imputer_MeanMedianTest.columns)
```

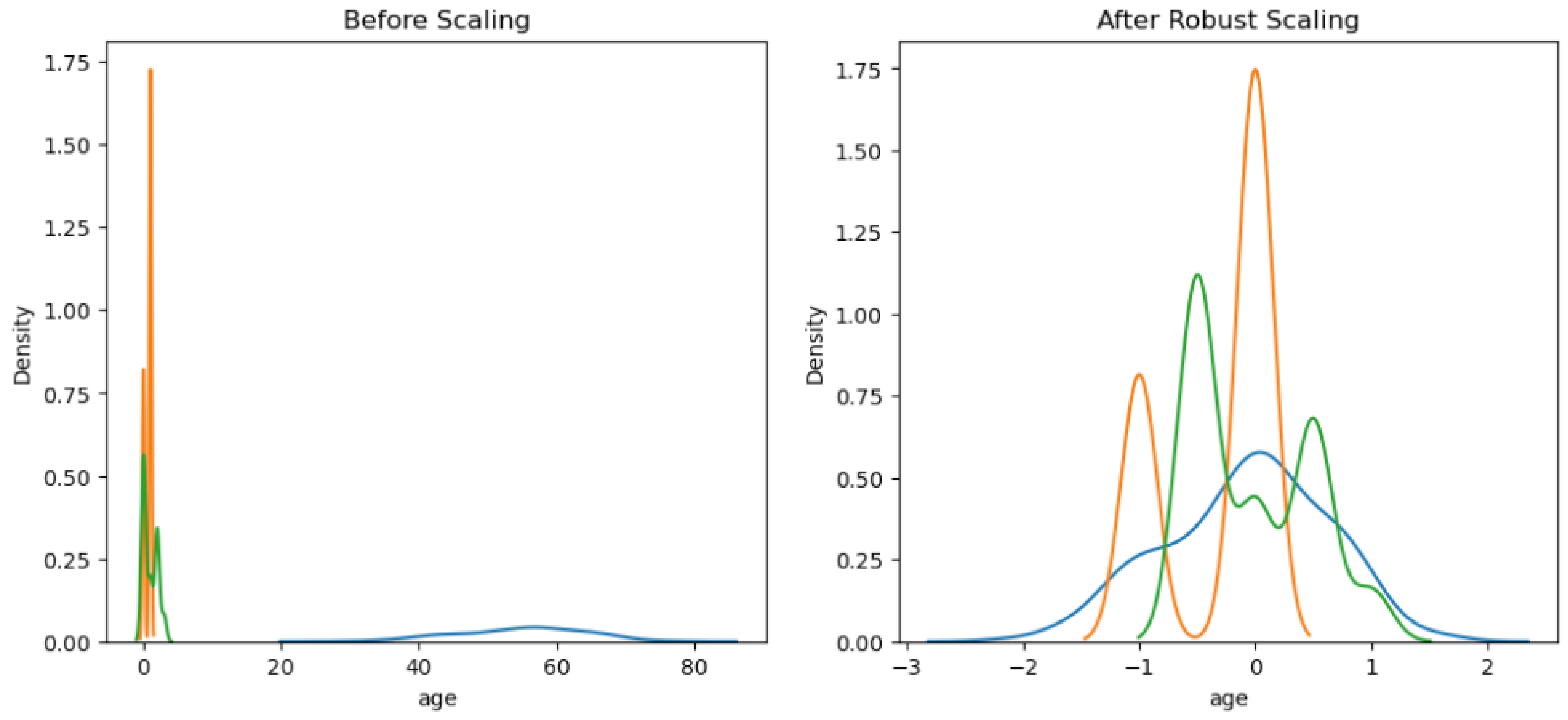
```
X_train_scaled
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	1.019608	0.0	0.5	0.5	0.178138	0.0	0.0	-0.188034	0.0	0.7500	0.0	3.0	1.0
1	0.627451	-1.0	-0.5	2.5	1.327935	0.0	1.0	0.000000	1.0	-0.5000	1.0	0.0	0.0
2	-0.862745	-1.0	0.0	0.0	-0.145749	0.0	0.0	0.803419	0.0	-0.1250	0.0	0.0	0.0
3	0.000000	-1.0	0.5	0.5	2.817814	1.0	0.0	0.188034	0.0	0.0000	0.0	1.0	0.0
4	0.784314	0.0	0.0	1.5	0.048583	0.0	1.0	-1.076923	1.0	-0.5000	0.0	3.0	-1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
237	0.392157	0.0	0.5	0.0	0.000000	1.0	1.0	-0.495726	1.0	0.1250	0.0	0.0	0.0
238	0.941176	0.0	0.5	-0.6	0.550607	0.0	1.0	-0.017094	0.0	0.1250	1.0	1.0	1.0
239	-0.862745	-1.0	0.0	-0.9	-1.344130	0.0	1.0	-0.461538	0.0	0.0000	0.0	0.0	0.0
240	-0.078431	0.0	-0.5	0.0	-0.421053	0.0	1.0	-1.384615	1.0	3.0000	-1.0	0.0	1.0
241	0.156863	0.0	-0.5	-1.5	-0.145749	0.0	1.0	0.153846	0.0	-0.4375	1.0	1.0	1.0

242 rows × 13 columns

Robust Scaler

```
fig, (ax1, ax2) = plt.subplots(ncols = 2, figsize =(12, 5))

#before scaling

ax1.set_title('Before Scaling')
sns.kdeplot(X_train['age'], ax = ax1)
sns.kdeplot(X_train['sex'], ax = ax1)
sns.kdeplot(X_train['cp'], ax = ax1)

#after Scaling
ax2.set_title('After Robust Scaling')
sns.kdeplot(X_train_scaled['age'], ax = ax2)
sns.kdeplot(X_train_scaled['sex'], ax = ax2)
sns.kdeplot(X_train_scaled['cp'], ax = ax2)

plt.show()
```



Discretization with EqualFrequency, EqualWidthDisc

Equal Frequency

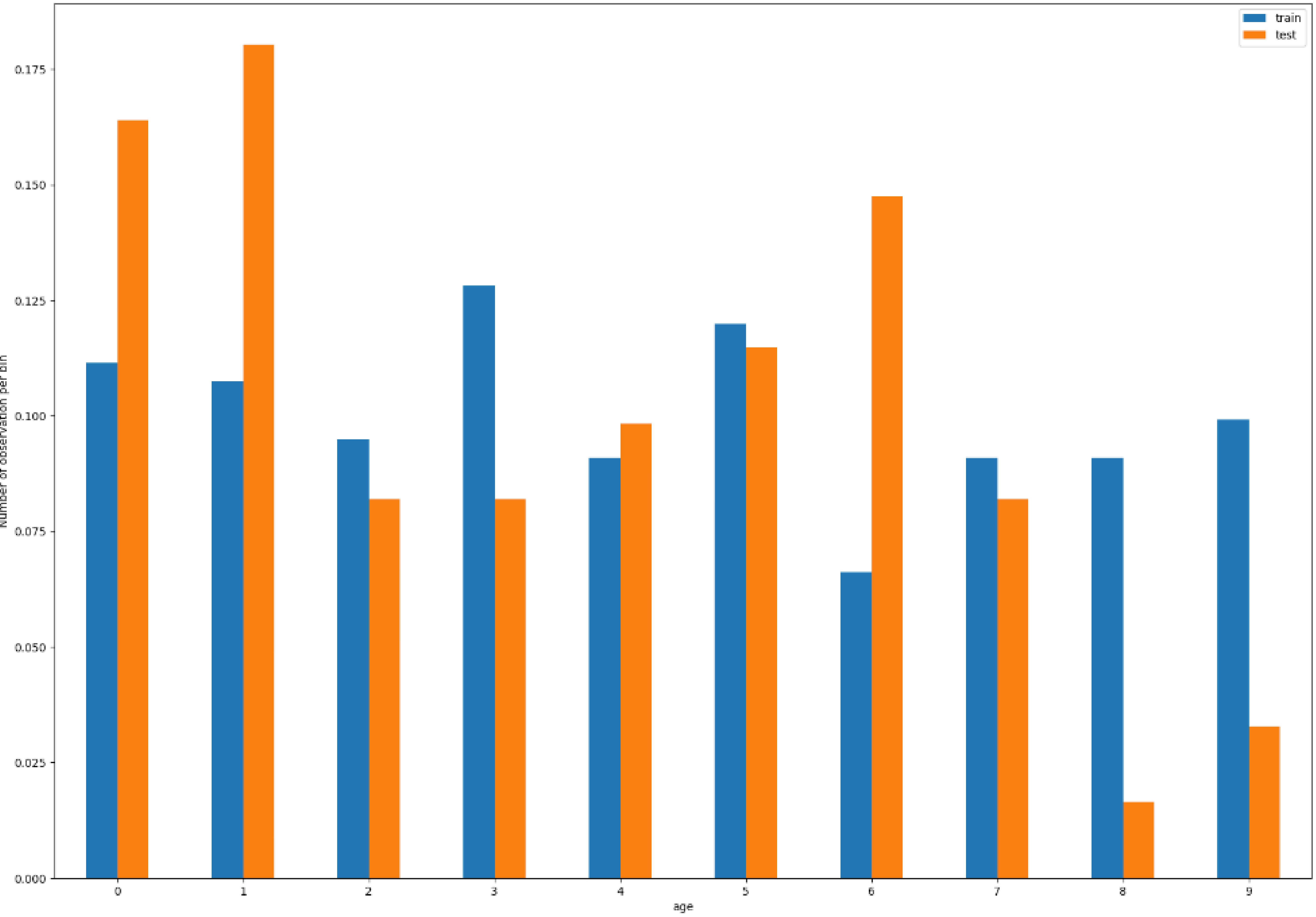
```python
# EqualFrequencyDiscretizer
disc = EqualFrequencyDiscretiser(
    q = 10,
    variables = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
)

disc.fit(imputer_MeanMedian)
```

```
EqualFrequencyDiscretiser(variables=['age', 'trestbps', 'chol', 'thalach',
                                     'oldpeak'])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```python
disc.binner_dict_
```

```
{'age': [-inf, 42.0, 47.0, 51.0, 54.0, 56.0, 58.0, 60.0, 63.0, 66.0, inf],
 'trestbps': [-inf,
  110.0,
  120.0,
  122.0,
  128.0,
  130.0,
  132.0,
  140.0,
  144.8,
  154.9,
  inf],
 'chol': [-inf,
  196.0,
  206.0,
  218.3,
  229.4,
  243.0,
  252.60000000000002,
  266.70000000000005,
  283.0,
  309.0,
  inf],
 'thalach': [-inf,
  116.0,
  130.2,
  140.0,
  145.4,
  151.5,
  156.0,
  162.0,
  166.8,
  174.9,
  inf],
```



train
test
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Number of observation per bin
0
1
2
3
4
5
6
7
8
9
age

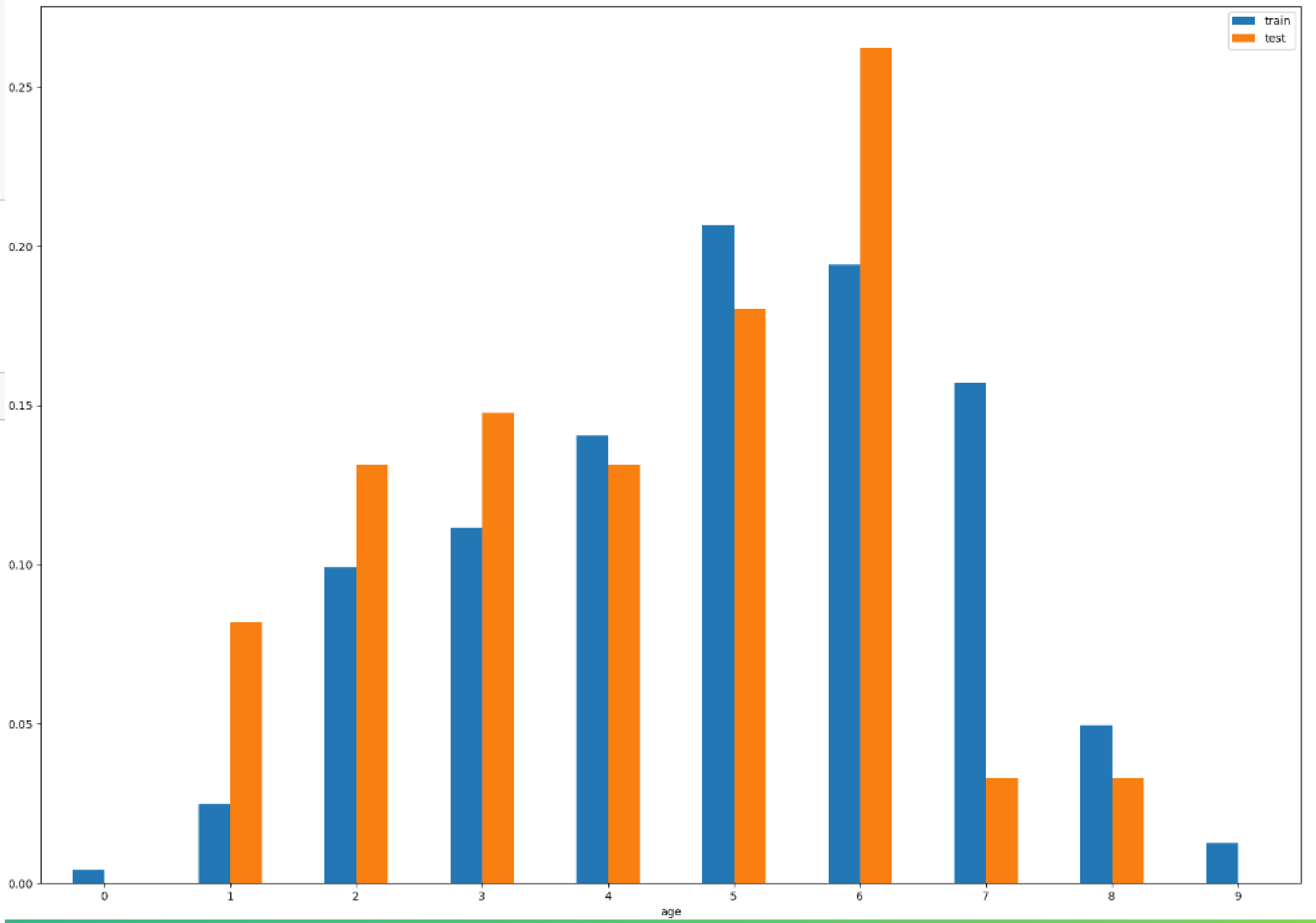
```python
#EqualWidthDiscretizer
disc = EqualWidthDiscretiser(
    bins = 10,
    variables =  ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
)

disc.fit(imputer_MeanMedian)
```

```
EqualWidthDiscretiser(variables=['age', 'trestbps', 'chol', 'thalach',
                                 'oldpeak'])
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```python
disc.binner_dict_
```

```
{'age': [-inf,
  33.8,
  38.6,
  43.4,
  48.2,
  53.0,
  57.8,
  62.6,
  67.4,
  72.19999999999999,
  inf],
 'trestbps': [-inf,
  104.6,
  115.2,
  125.8,
  136.4,
  147.0,
  157.6,
  168.2,
  178.8,
  189.39999999999998,
  inf],
 'chol': [-inf,
  174.3,
  217.6,
  260.9,
  304.2,
  347.5,
  390.79999999999995,
  434.09999999999997,
  477.4,
  520.7,
  inf],
 'thalach': [-inf,
```



EqualWidth Discretization

Model Results

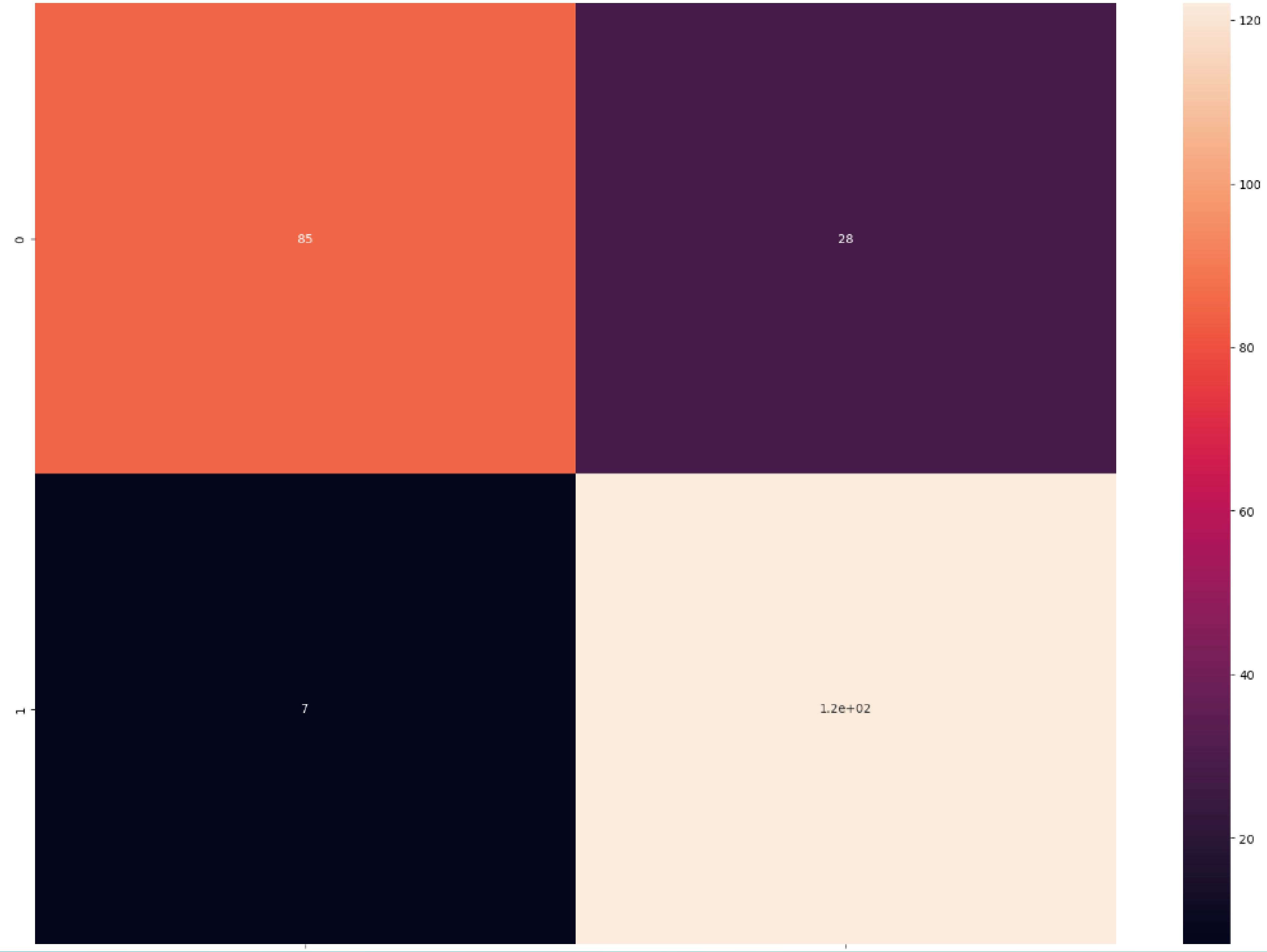
```
1 metrics_score(X_train, X_test, y_train, y_test, best_DT_model)
```

```
              Roc-auc score

Train set: 82.039%
Test set: 81.944%


              Accuracy score

Train set: 75.583%
Test set: 85.238%


            Standart deviation

Train set: 7.773%
Test set: 17.236%


          Classification report

              precision    recall  f1-score   support

           0       0.72      0.72      0.72        25
           1       0.81      0.81      0.81        36

    accuracy                           0.77        61
   macro avg       0.76      0.76      0.76        61
weighted avg       0.77      0.77      0.77        61
```

Best Results Decision Tree Model

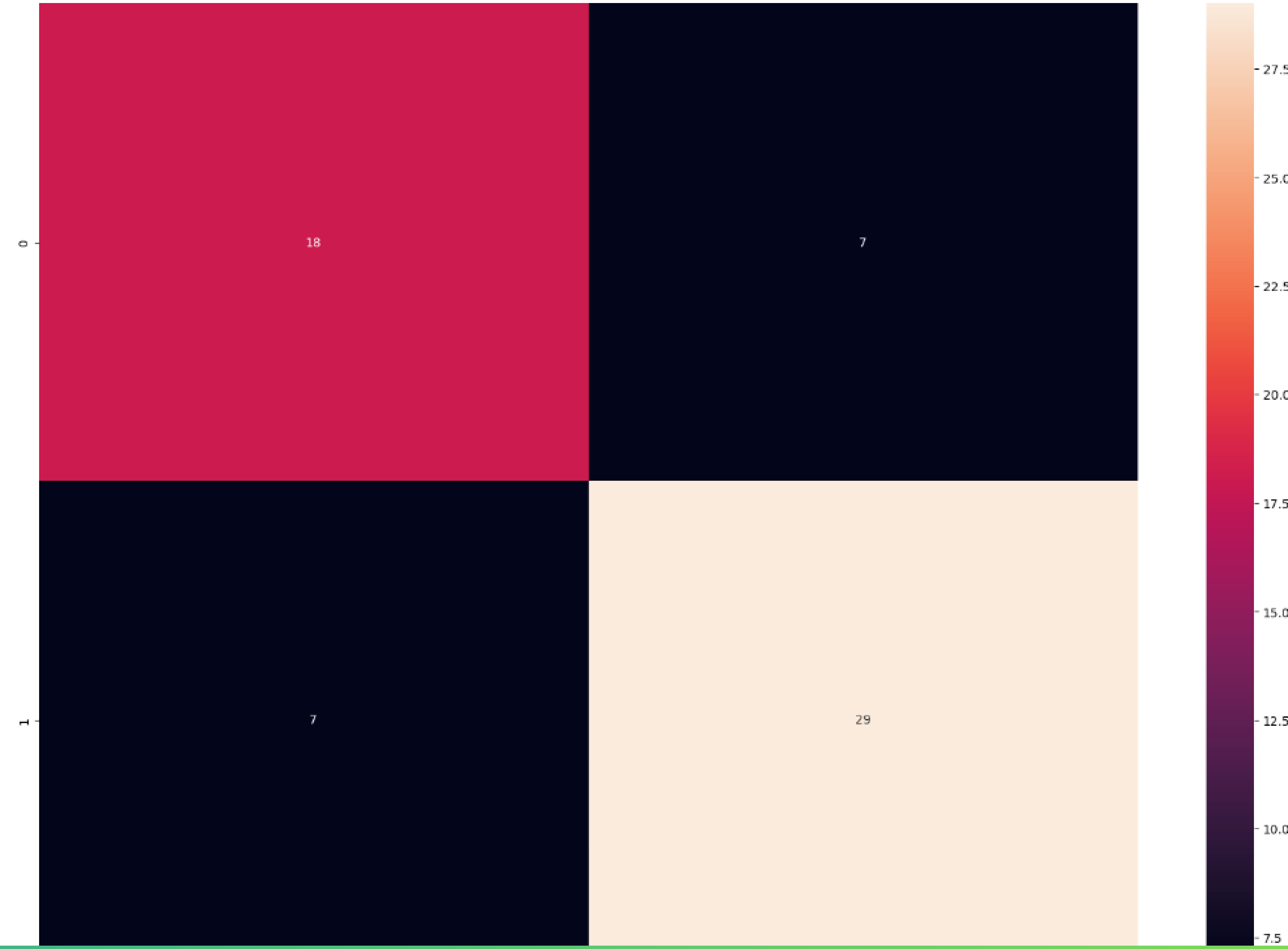
```python
y_pred_train = best_DT_model.predict(X_train)
cm = confusion_matrix(y_train, y_pred_train)
sns.heatmap(cm, annot = True)
```

<AxesSubplot: >




```python
y_pred_test = best_DT_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot = True)
```

<AxesSubplot: >



Train Test

Confusion Matrics

Decision Tree

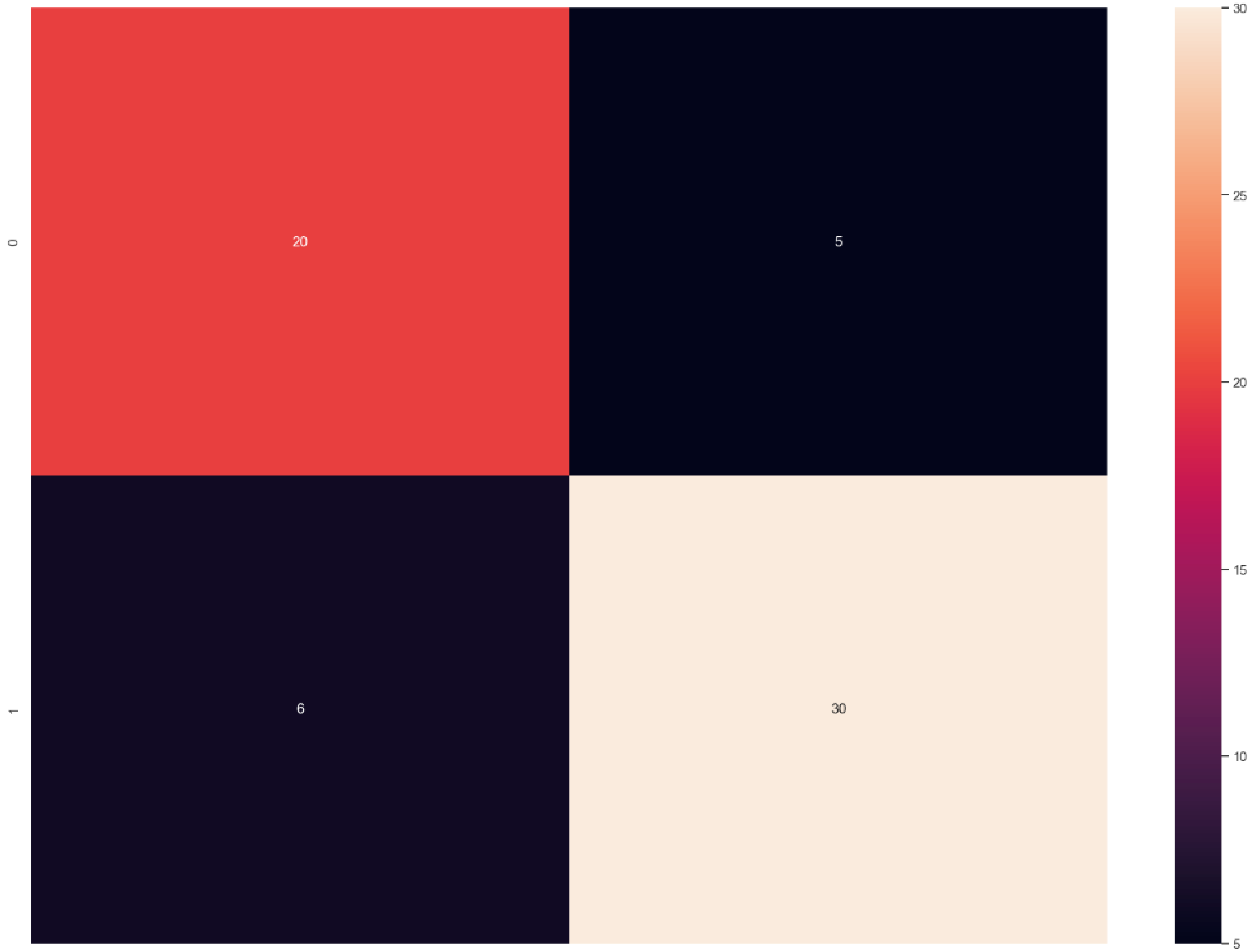
```
metrics_score(X_train, X_test, y_train, y_test, rf_best_model)
```

```
            Roc-auc score

Train set: 90.170%
Test set: 90.556%


            Accuracy score

Train set: 81.417%
Test set: 85.000%


          Standart deviation

Train set: 5.101%
Test set: 12.877%


        Classification report

              precision    recall  f1-score   support

           0       0.90      0.72      0.80        25
           1       0.83      0.94      0.88        36

    accuracy                           0.85        61
   macro avg       0.86      0.83      0.84        61
weighted avg       0.86      0.85      0.85        61
```

Best Results Random Forest Classifier

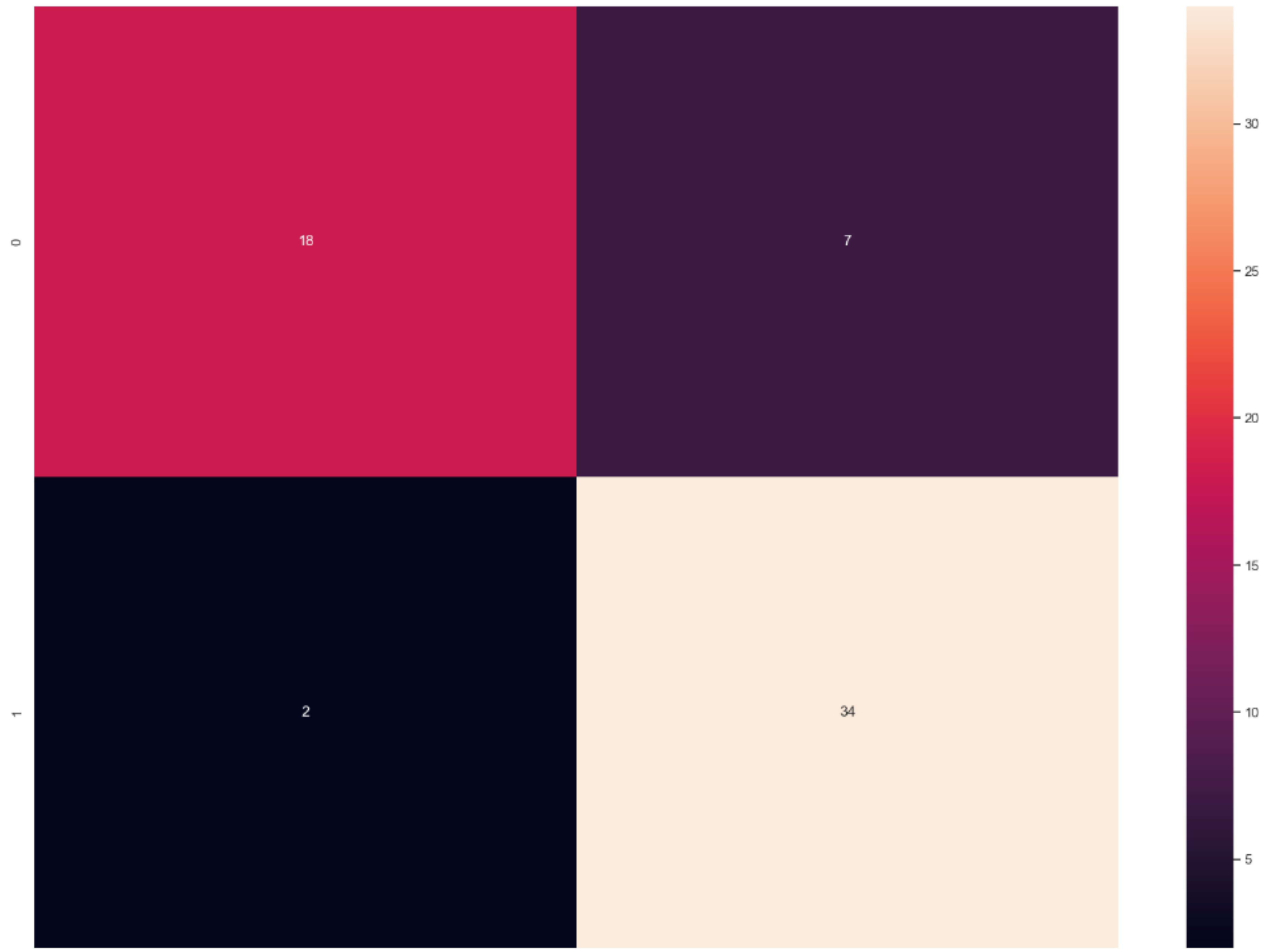
```python
y_pred_train = rf_best_model.predict(X_train)
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot = True)
```

```
<AxesSubplot: >
```




```python
y_pred_test = rf_best_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot = True)
```

```
<AxesSubplot: >
```



Train

Test

Confusion Matrics

Random Forest

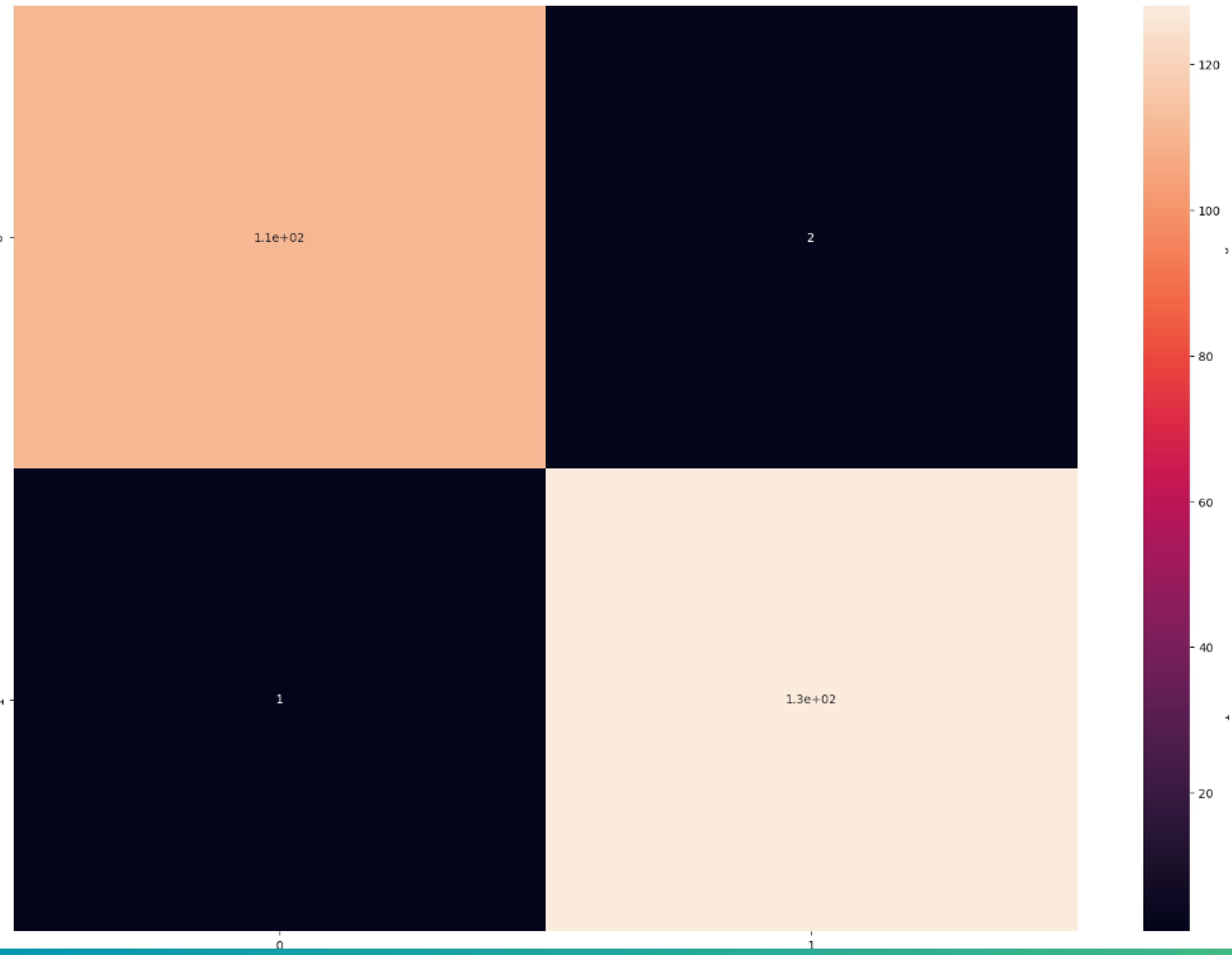
```
1 metrics_score(X_train, X_test, y_train, y_test, best_model_KNNeighbors)
```

```
            Roc-auc score

Train set: 87.669%
Test set: 89.444%


            Accuracy score

Train set: 80.567%
Test set: 76.667%


          Standart deviation

Train set: 7.313%
Test set: 10.869%


        Classification report

              precision    recall  f1-score   support

           0       0.81      0.68      0.74        25
           1       0.80      0.89      0.84        36

    accuracy                           0.80        61
   macro avg       0.80      0.78      0.79        61
weighted avg       0.80      0.80      0.80        61
```

Best Results KNNeighbors Classifier

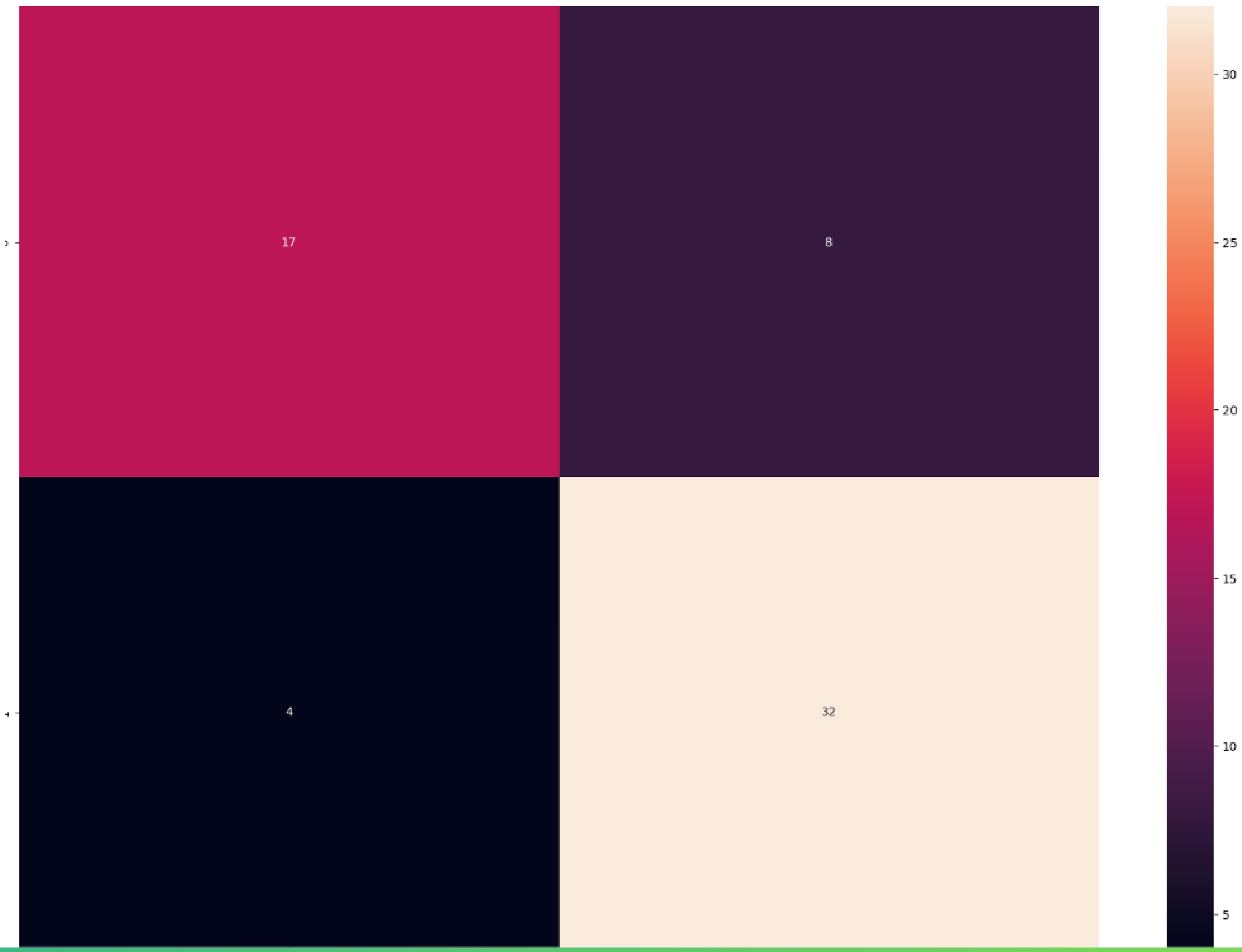
```
y_pred_train = best_model_KNNeighbors.predict(X_train)
cm = confusion_matrix(y_train, y_pred_train)
sns.heatmap(cm, annot = True)
```

AxesSubplot: >




```
y_pred_test = best_model_KNNeighbors.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot = True)
```

AxesSubplot: >



Train Test

Confusion Matrics

KNNeighbors Classifier

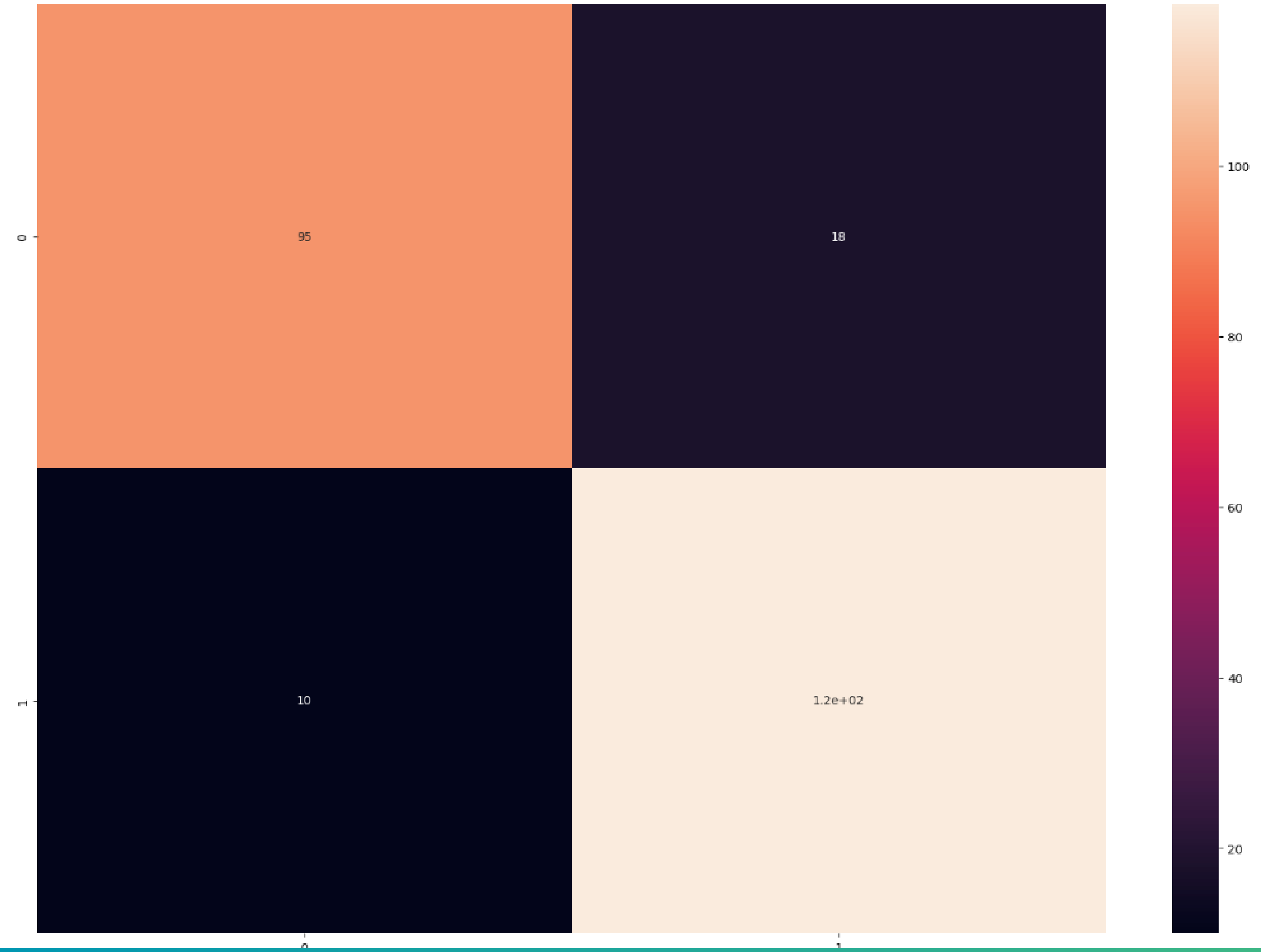
```
1 metrics_score(X_train, X_test, y_train, y_test, best_model_SVC)
```

```
             Roc-auc score

Train set: 89.604%
Test set: 89.722%


             Accuracy score

Train set: 82.217%
Test set: 86.667%


           Standart deviation

Train set: 7.627%
Test set: 10.833%


         Classification report

              precision    recall  f1-score   support

           0       0.71      0.80      0.75        25
           1       0.85      0.78      0.81        36

    accuracy                           0.79        61
   macro avg       0.78      0.79      0.78        61
weighted avg       0.79      0.79      0.79        61
```

Best Results Support Vector Classifier

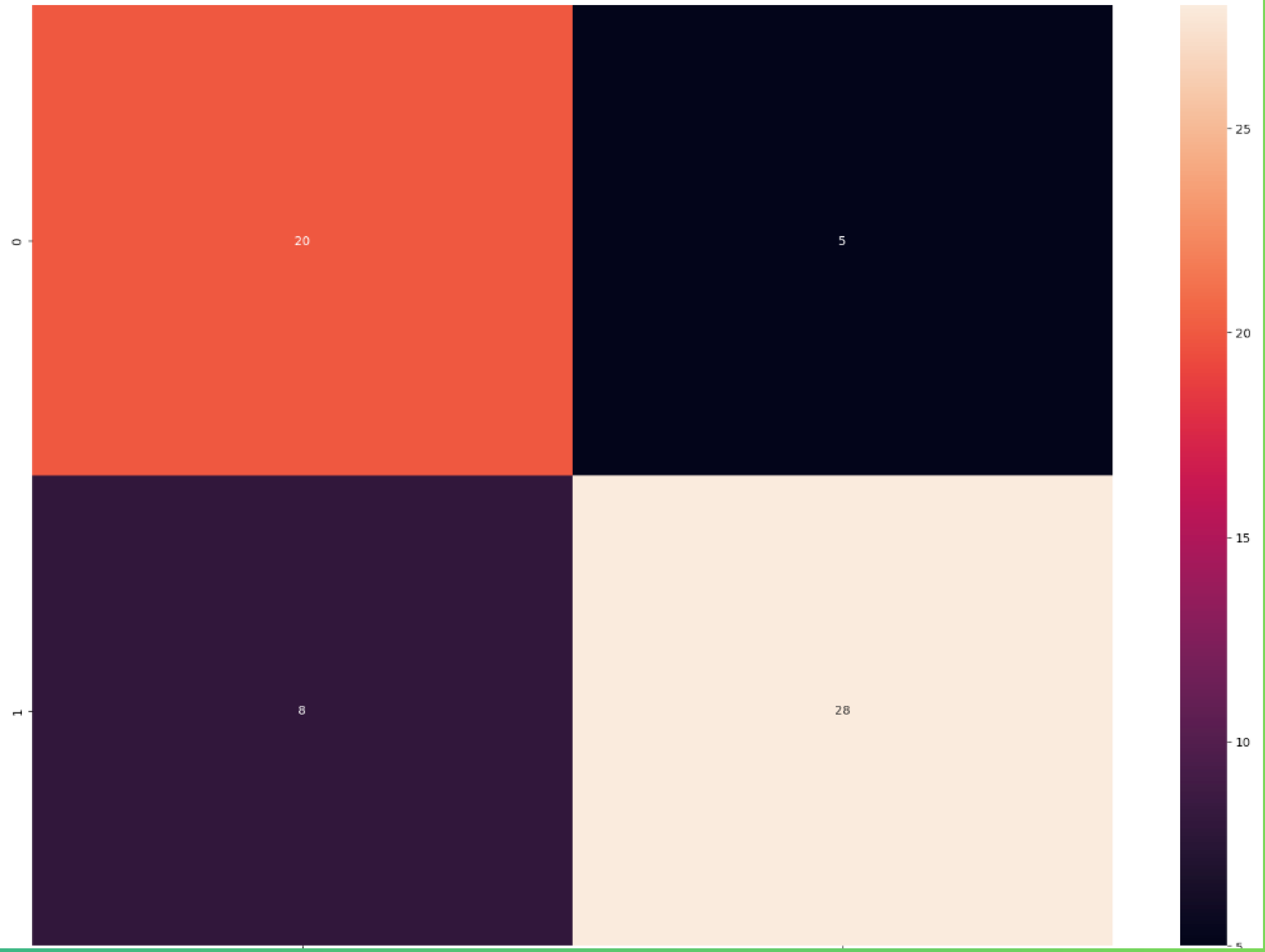
```
y_pred_train = best_model_SVC.predict(X_train)
cm = confusion_matrix(y_train, y_pred_train)
sns.heatmap(cm, annot = True)
```

<AxesSubplot: >




```
y_pred_test = best_model_SVC.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot = True)
```

<AxesSubplot: >



Train Test

Confusion Matrics

Support Vector Classifier

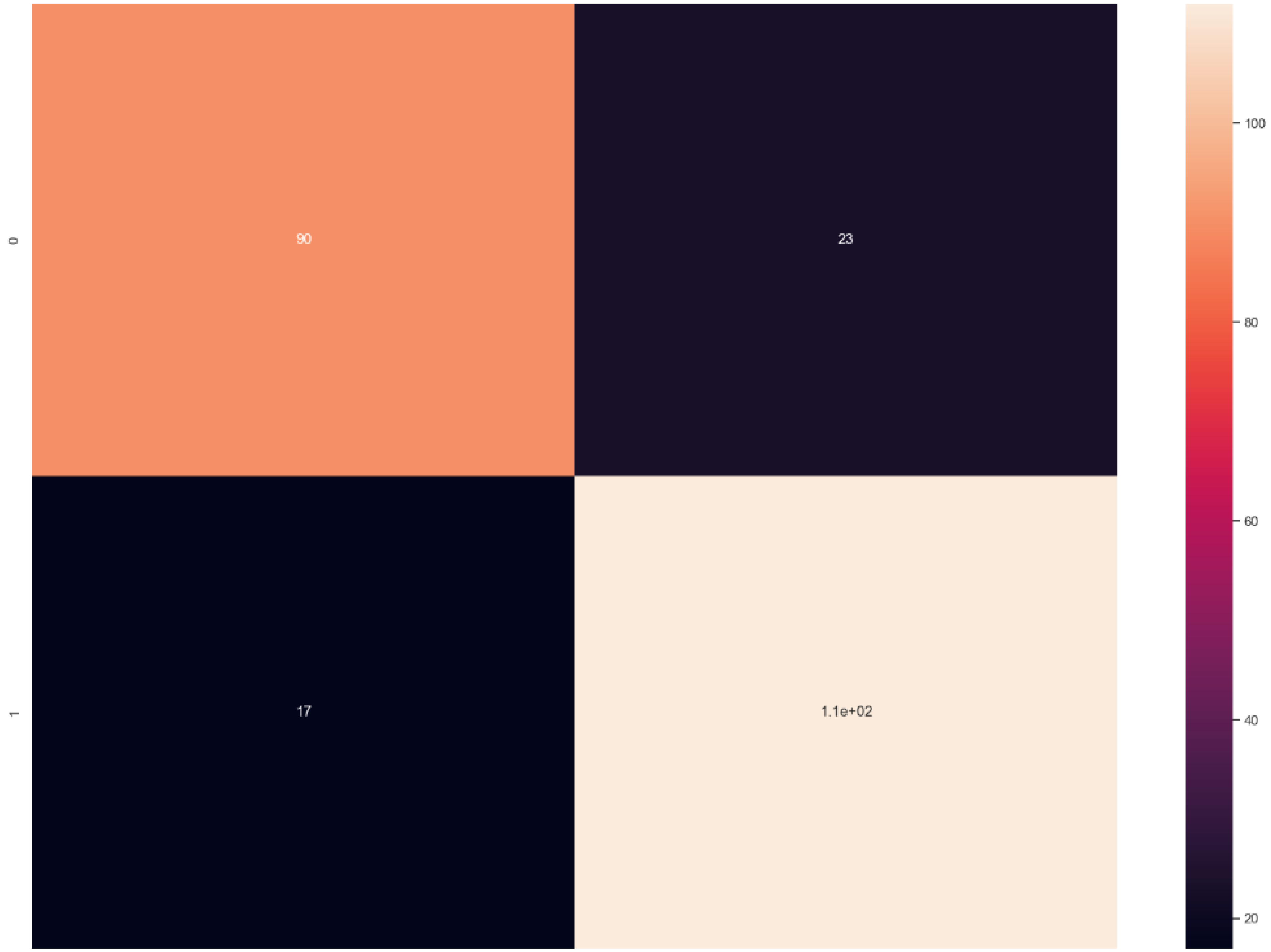
```
metrics_score(X_train, X_test, y_train, y_test, best_model_logit)
```

```
            Roc-auc score

Train set: 90.058%
Test set: 89.444%


            Accuracy score

Train set: 82.633%
Test set: 83.333%


          Standart deviation

Train set: 6.117%
Test set: 11.167%


        Classification report

              precision    recall  f1-score   support

           0       0.77      0.80      0.78        25
           1       0.86      0.83      0.85        36

    accuracy                           0.82        61
   macro avg       0.81      0.82      0.81        61
weighted avg       0.82      0.82      0.82        61
```

Best Results Logistic Regression

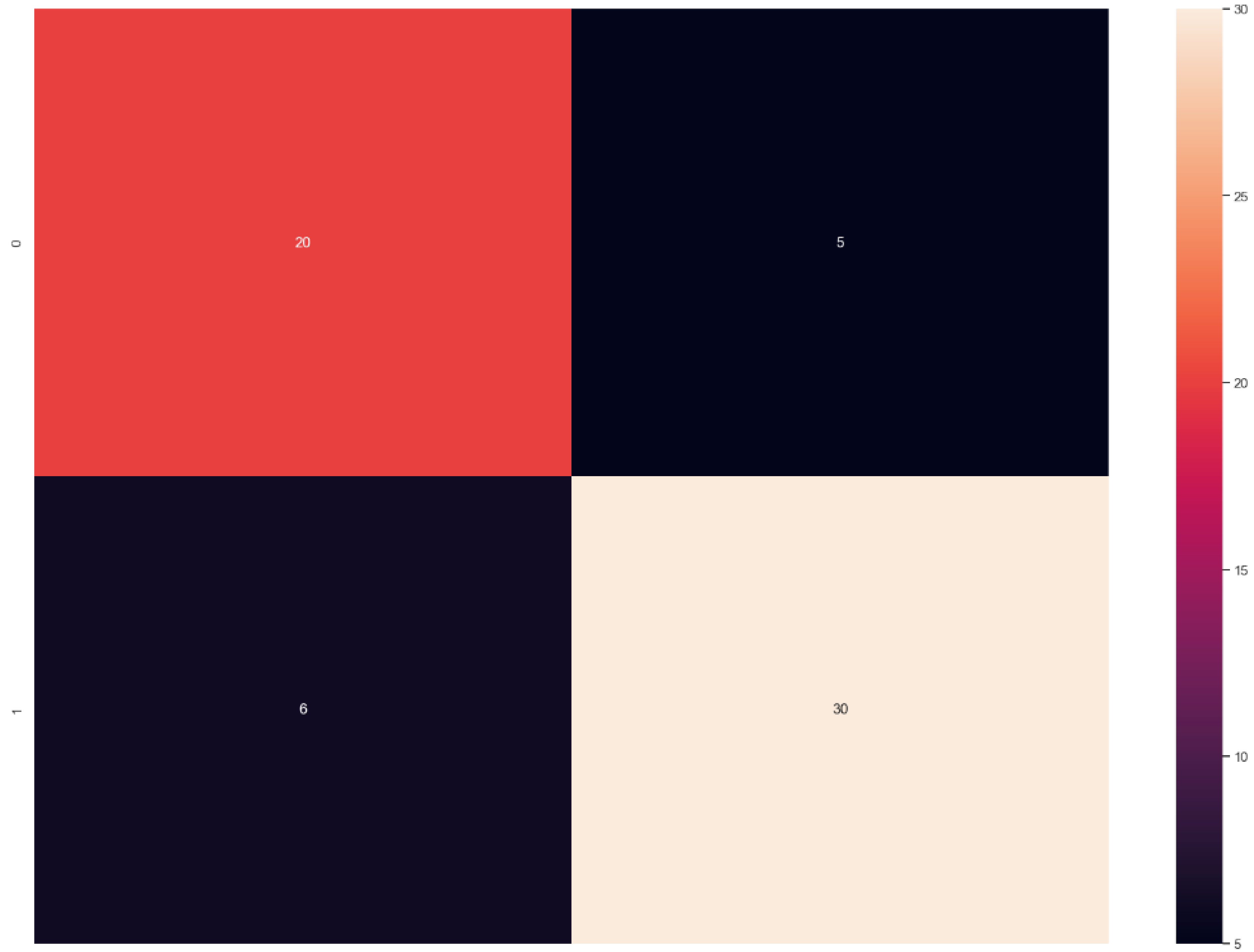
```
y_pred_test = best_model_logit.predict(X_train)
cm = confusion_matrix(y_train, y_pred_train)
sns.heatmap(cm, annot = True)
```

```
<AxesSubplot: >
```




```
y_pred_test = best_model_logit.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot = True)
```

```
<AxesSubplot: >
```



Train Test

Confusion Matrics

Logistic Regression

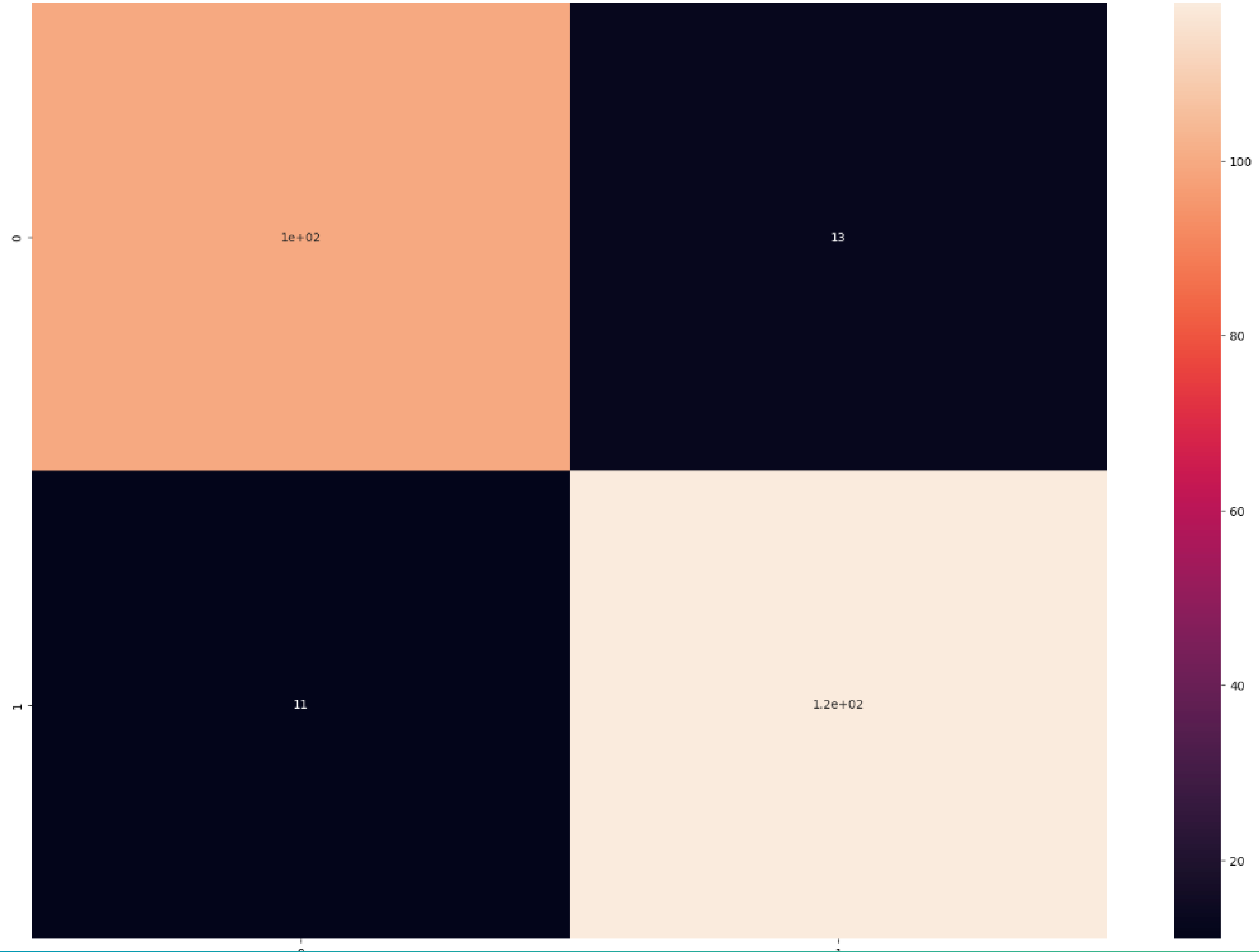
```
1 metrics_score(X_train, X_test, y_train, y_test, best_model_ada)
```

```
            Roc-auc score

Train set: 89.704%
Test set: 90.278%


            Accuracy score

Train set: 83.083%
Test set: 86.667%


          Standart deviation

Train set: 5.739%
Test set: 14.540%


        Classification report

              precision    recall  f1-score   support

           0       0.81      0.84      0.82        25
           1       0.89      0.86      0.87        36

    accuracy                           0.85        61
   macro avg       0.85      0.85      0.85        61
weighted avg       0.85      0.85      0.85        61
```

Best Results AdaBoost Classifier

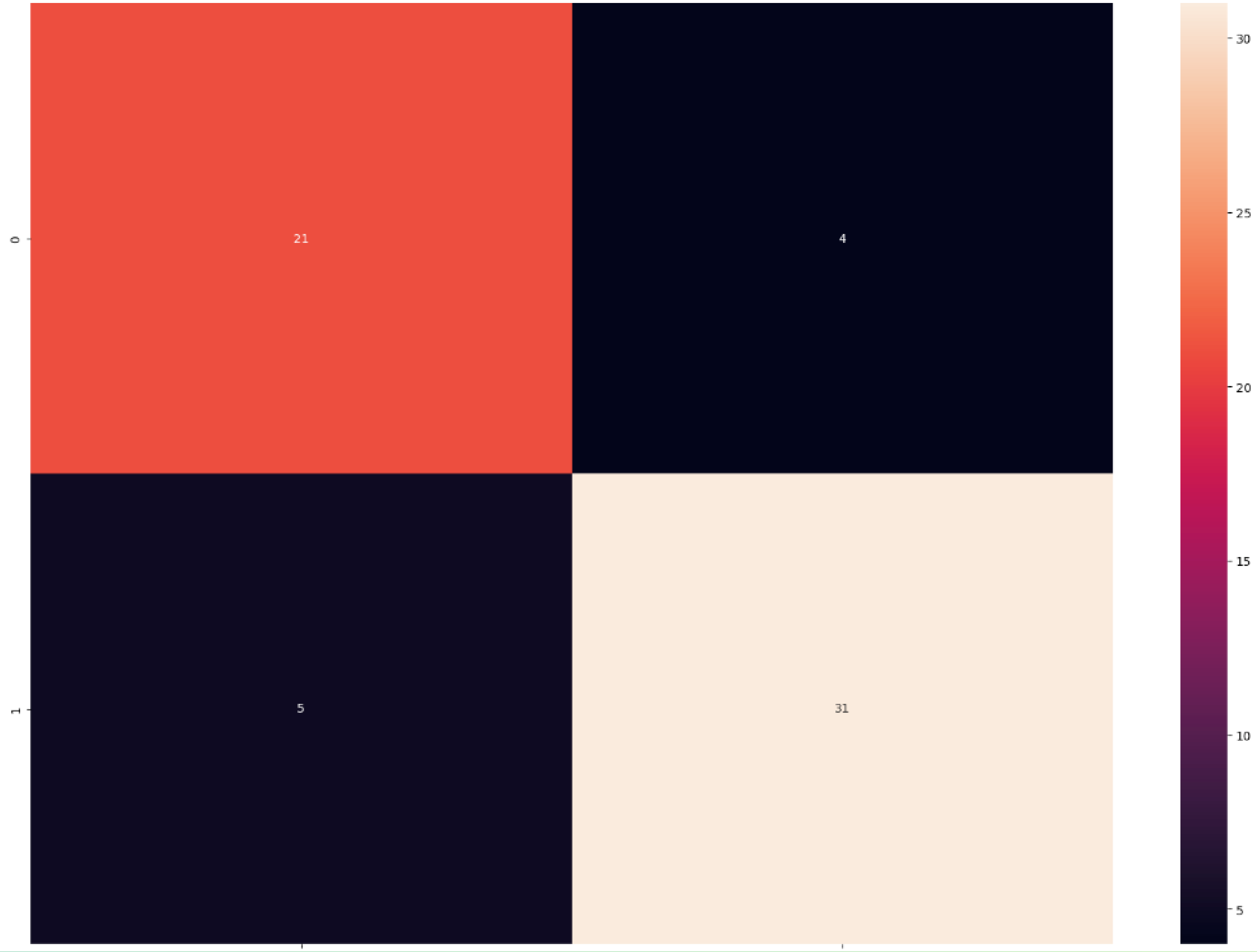
```python
y_pred_train = best_model_ada.predict(X_train)
cm = confusion_matrix(y_train, y_pred_train)
sns.heatmap(cm, annot = True)
```

<AxesSubplot: >




```python
y_pred_test = best_model_ada.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot = True)
```

<AxesSubplot: >



Train Test

Confusion Matrics

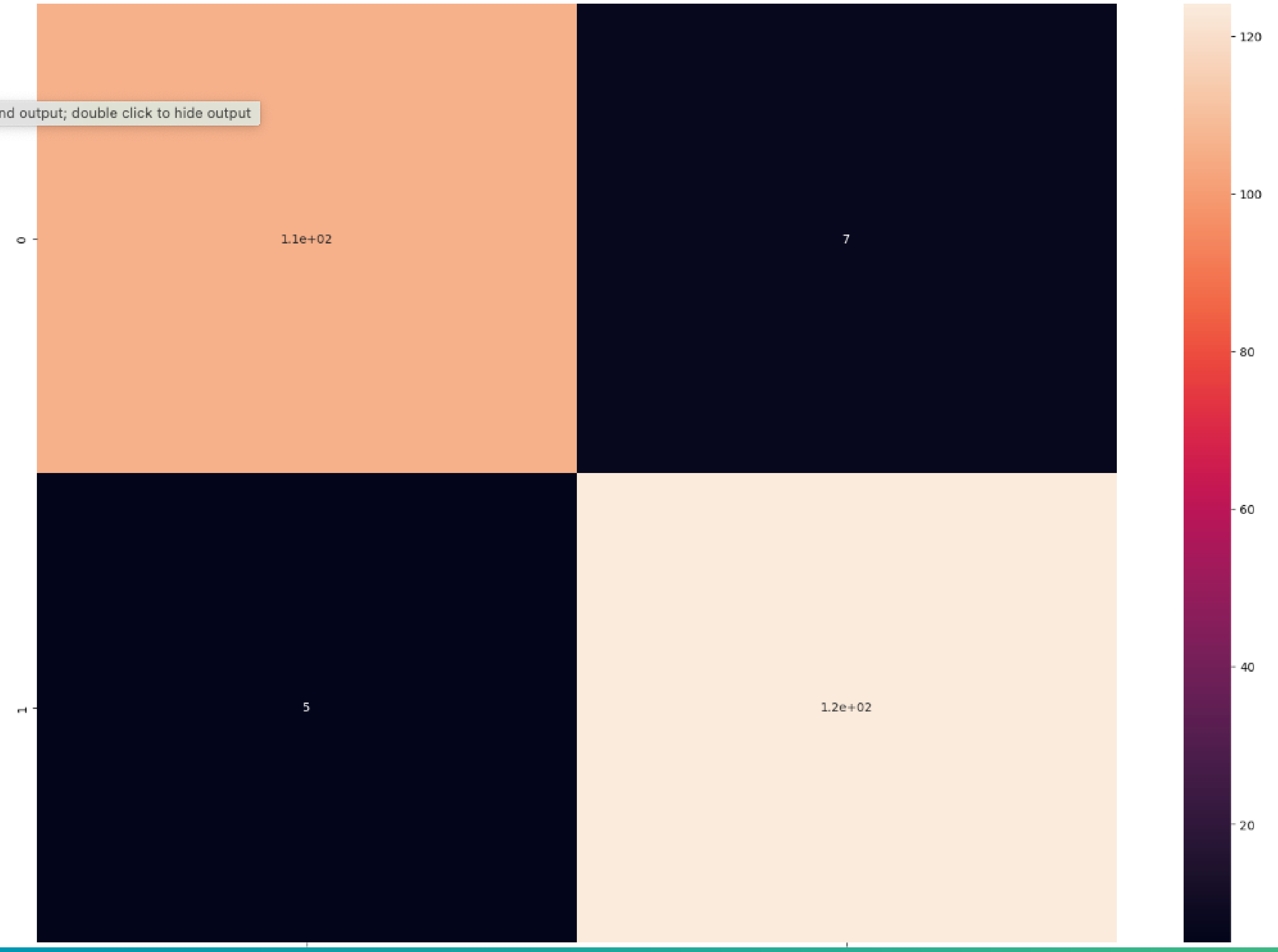
AdaBoost Classifier

Best Results XGBoost Classifier

```
           Roc-auc score

Train set: 87.068%
Test set: 90.694%


           Accuracy score

Train set: 80.200%
Test set: 83.333%


         Standart deviation

Train set: 5.586%
Test set: 12.851%


       Classification report

              precision    recall  f1-score   support

           0       0.90      0.76      0.83        25
           1       0.85      0.94      0.89        36

    accuracy                           0.87        61
   macro avg       0.88      0.85      0.86        61
weighted avg       0.87      0.87      0.87        61
```

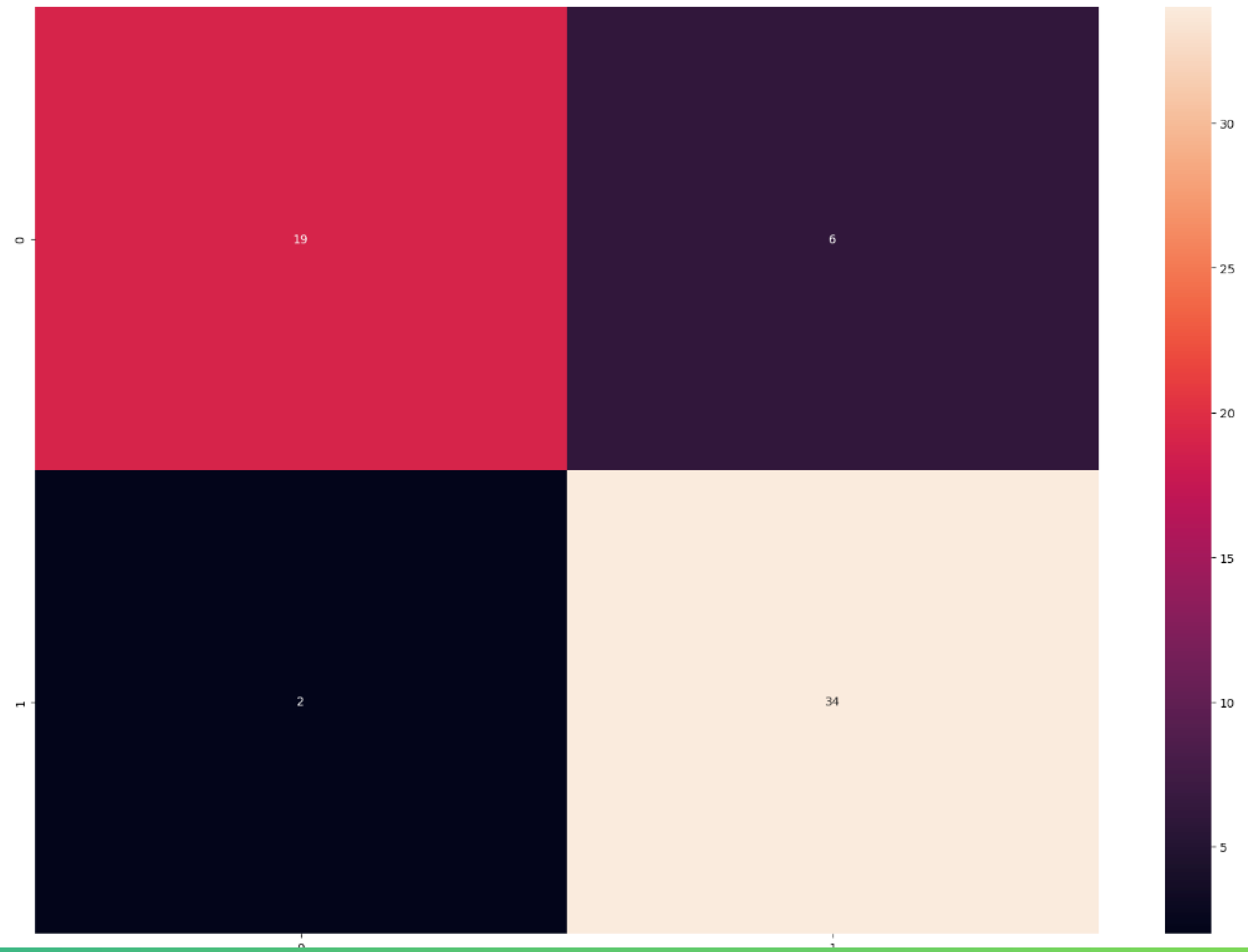
```python
y_pred_train = best_model_XGB.predict(X_train)
cm = confusion_matrix(y_train, y_pred_train)
sns.heatmap(cm, annot = True)
```

<AxesSubplot: >




```python
y_pred_test = best_model_XGB.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot = True)
```

<AxesSubplot: >



Train **Test**

Confusion Matrics

XGBoost Classifier

Best Results Bagging Classifier

Roc-auc score

Train set: 88.946%
Test set: 94.167%

Accuracy score

Train set: 81.333%
Test set: 85.000%

Standart deviation

Train set: 5.485%
Test set: 9.497%

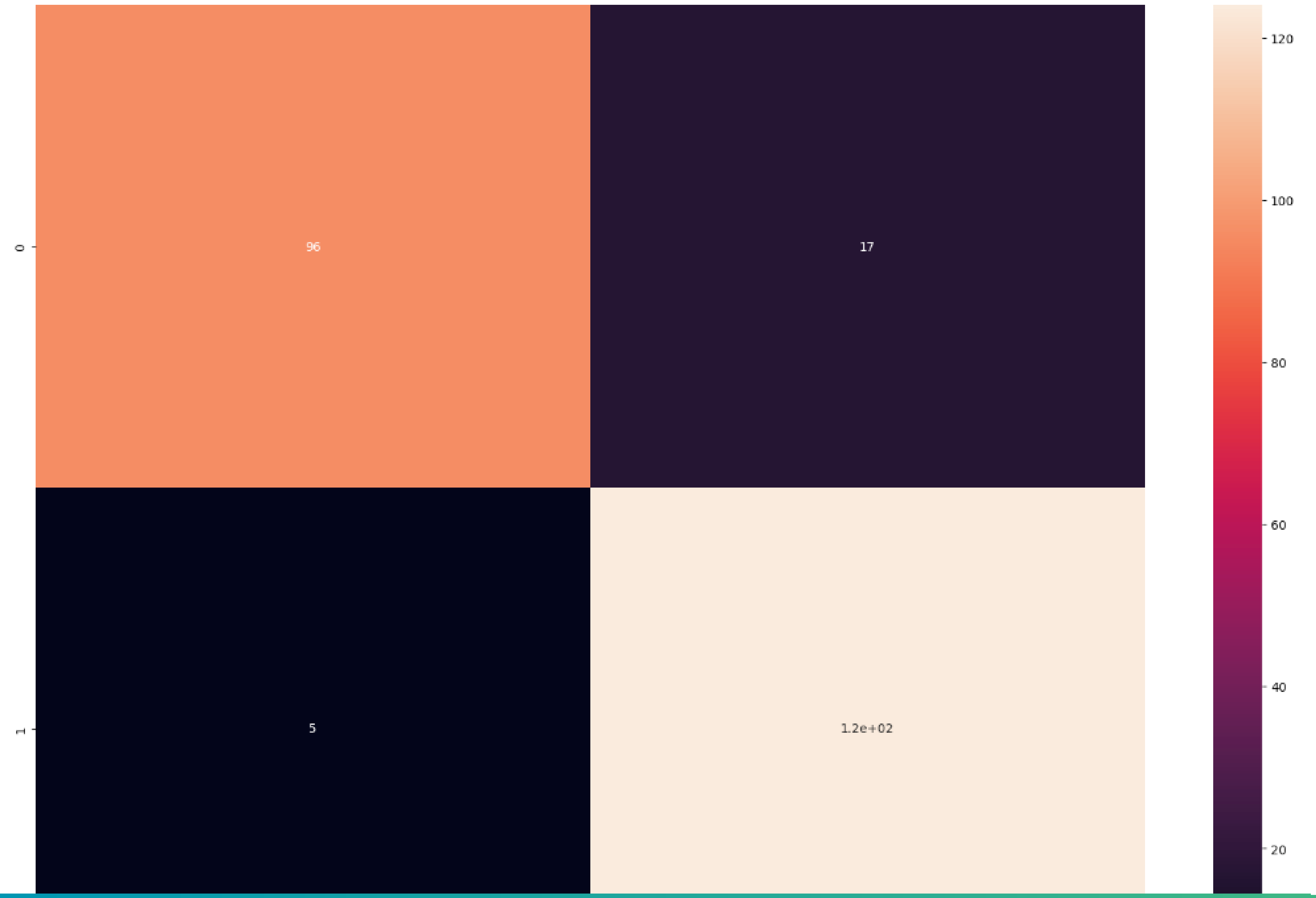
Classification report

	precision	recall	f1-score	support
0	0.91	0.80	0.85	25
1	0.87	0.94	0.91	36
accuracy			0.89	61
macro avg	0.89	0.87	0.88	61
weighted avg	0.89	0.89	0.88	61

```
y_pred_train = best_model_Bagg.predict(X_train)
cm = confusion_matrix(y_train, y_pred_train)
sns.heatmap(cm, annot = True)
```

```
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done   3 out of   8 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=8)]: Done   8 out of   8 | elapsed:    0.0s finished
```

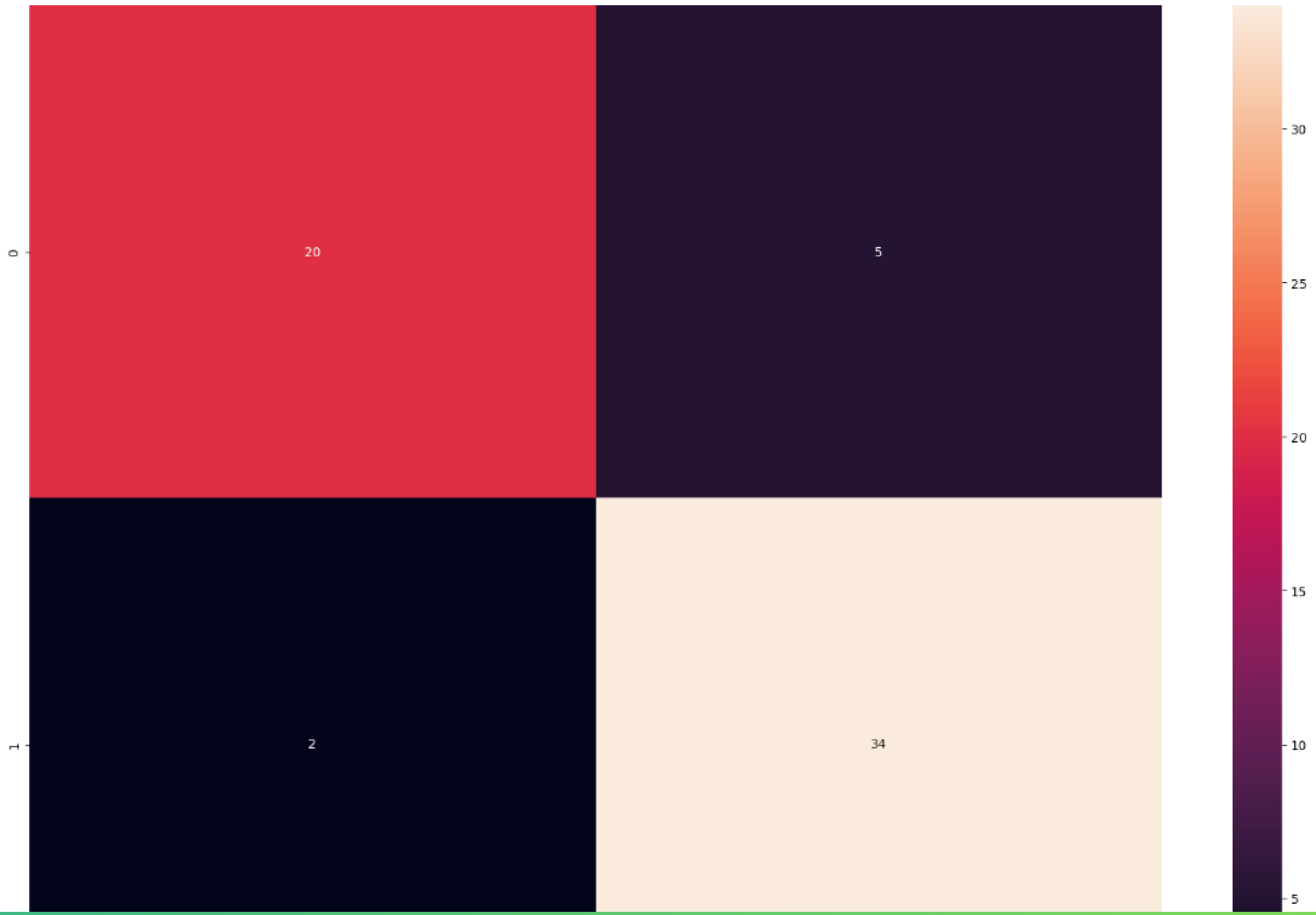
```
<AxesSubplot: >
```




```
y_pred_test = best_model_Bagg.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot = True)
```

```
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done   3 out of   8 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=8)]: Done   8 out of   8 | elapsed:    0.0s finished
```

```
<AxesSubplot: >
```



Train **Test**

Confusion Matrics

Bagging Classifier

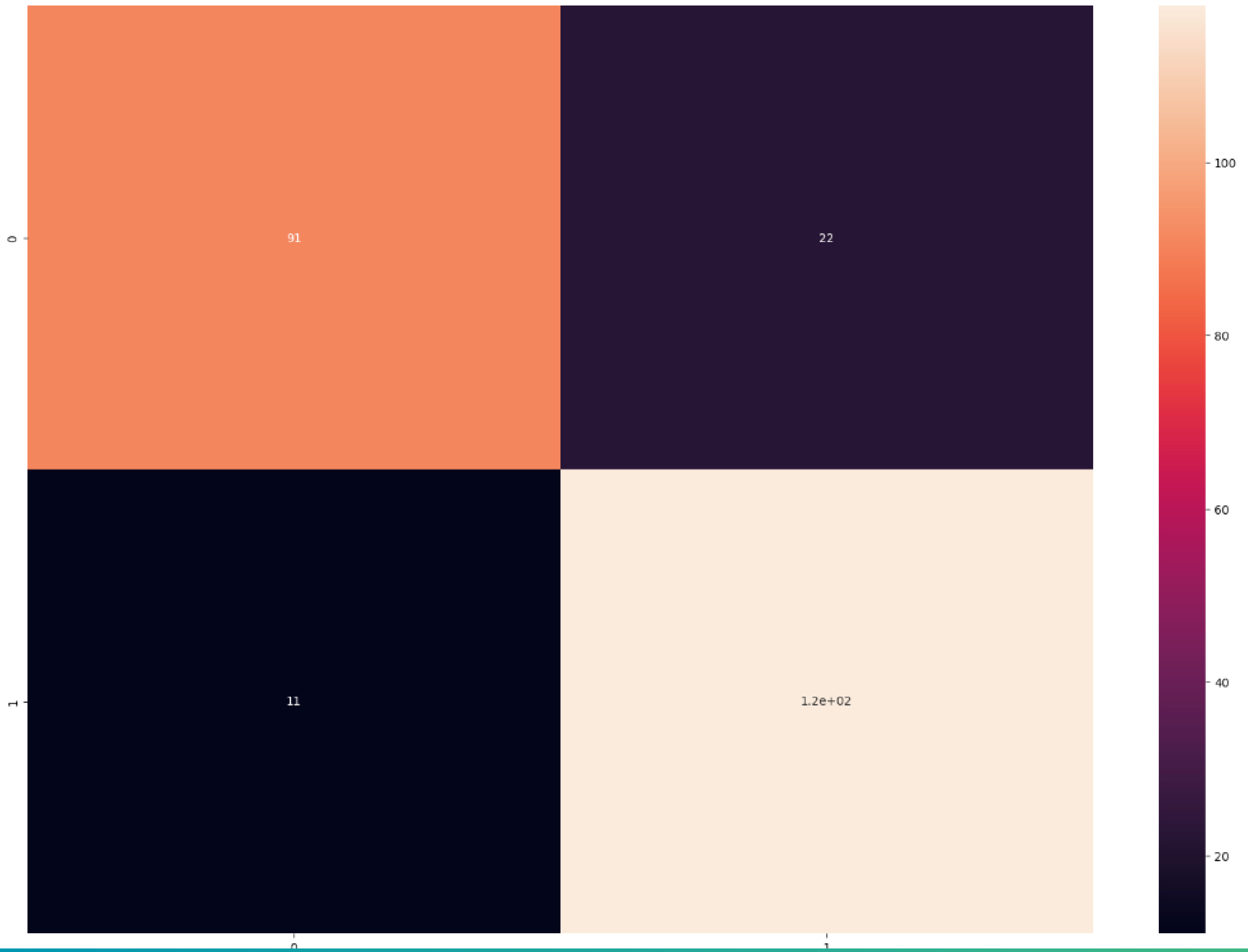
```
metrics_score(X_train, X_test, y_train, y_test, best_model_Extra)
```

```
            Roc-auc score

Train set: 90.420%
Test set: 91.806%


            Accuracy score

Train set: 82.183%
Test set: 85.000%


          Standart deviation

Train set: 4.041%
Test set: 10.476%


        Classification report

              precision    recall  f1-score   support

           0       0.86      0.72      0.78        25
           1       0.82      0.92      0.87        36

    accuracy                           0.84        61
   macro avg       0.84      0.82      0.83        61
weighted avg       0.84      0.84      0.83        61
```

Best Results ExtraTree Classifier

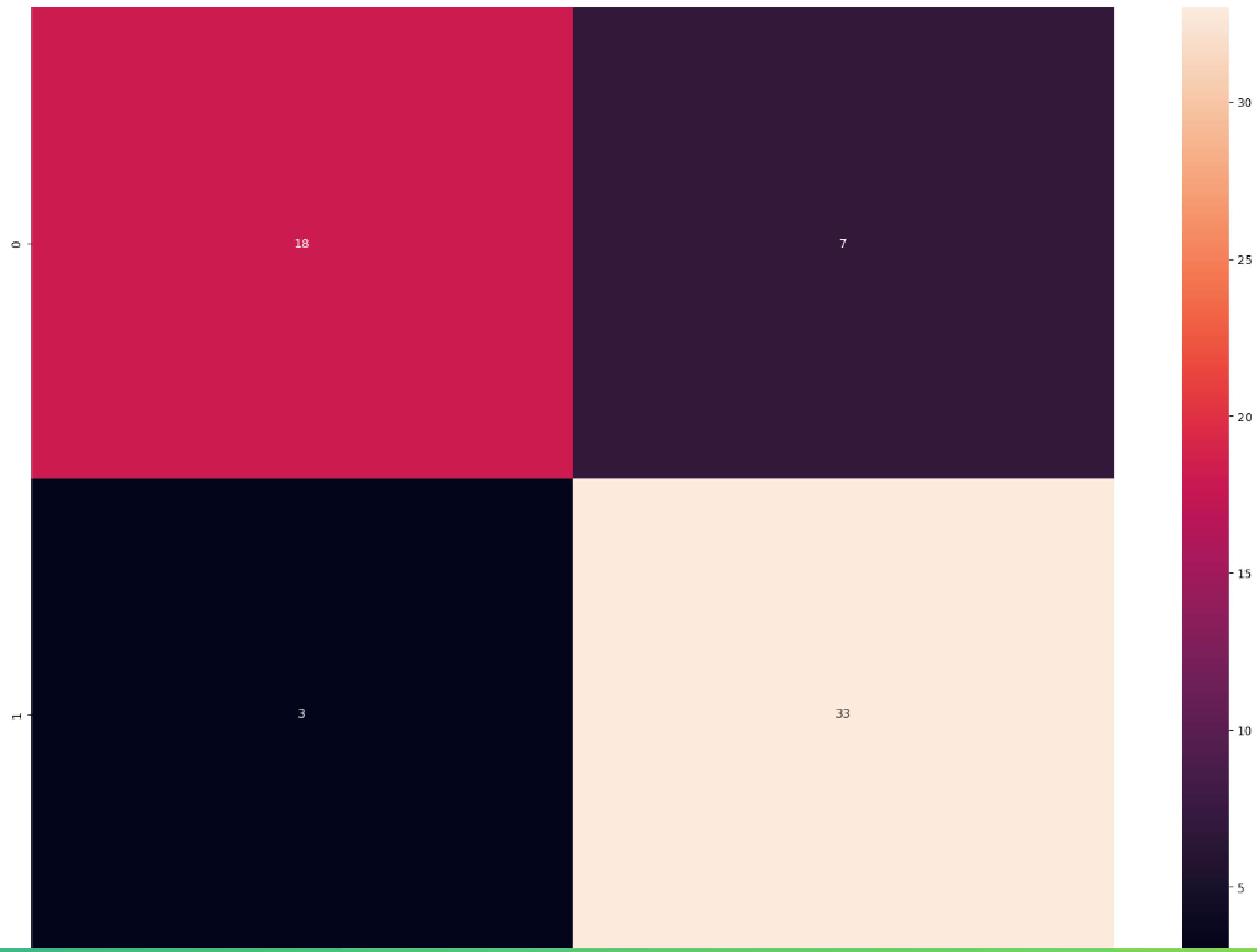
```
y_pred_train = best_model_Extra.predict(X_train)
cm = confusion_matrix(y_train, y_pred_train)
sns.heatmap(cm, annot = True)
```

<AxesSubplot: >




```
y_pred_test = best_model_Extra.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot = True)
```

<AxesSubplot: >



Train

Test

Confusion Matrics

ExtraTree Classifier

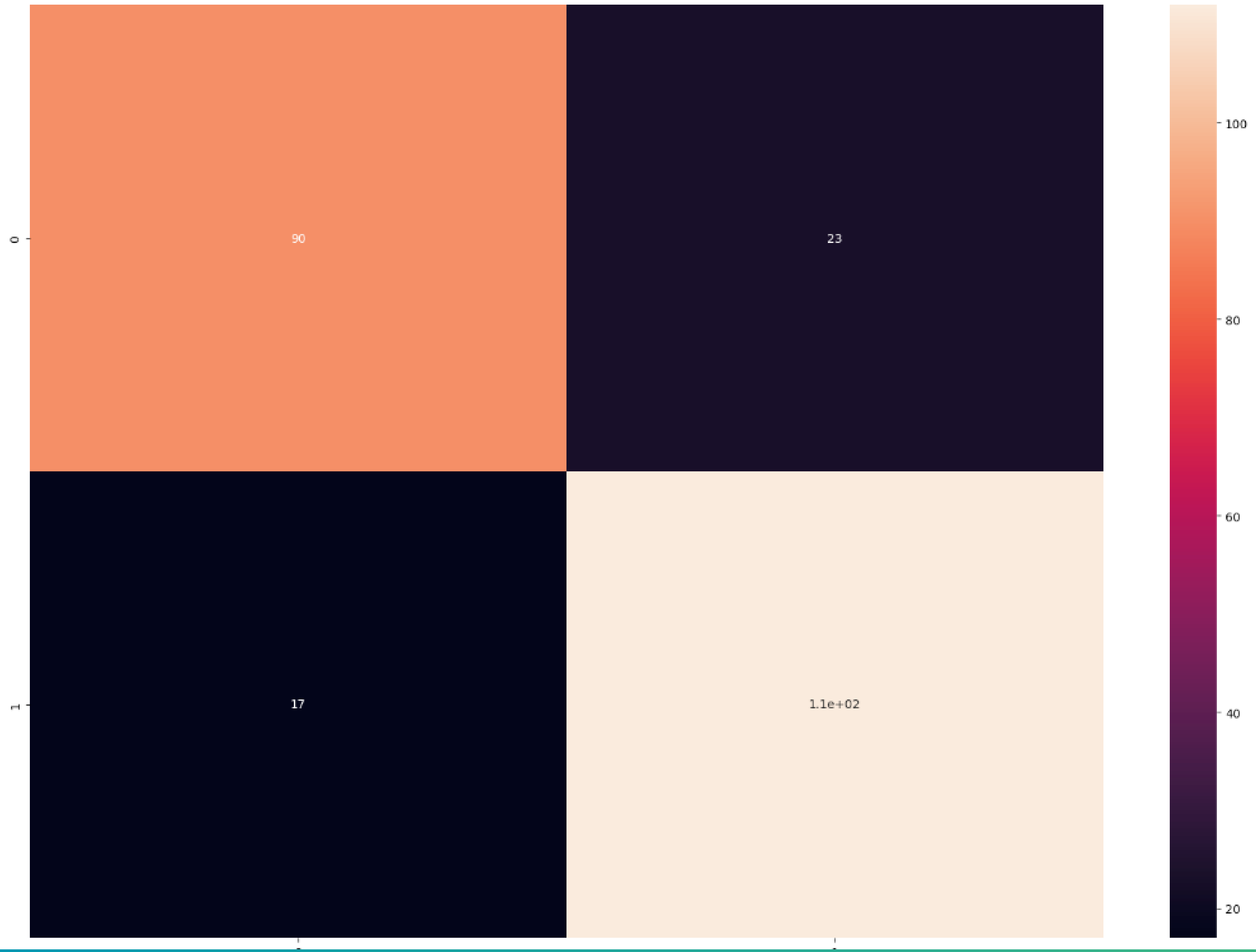
```
metrics_score(X_train, X_test, y_train, y_test, best_model_NB)
```

```
             Roc-auc score

Train set: 88.903%
Test set: 92.917%


             Accuracy score

Train set: 80.967%
Test set: 80.000%


           Standart deviation

Train set: 5.608%
Test set: 9.470%


         Classification report

              precision    recall  f1-score   support

           0       0.80      0.80      0.80        25
           1       0.86      0.86      0.86        36

    accuracy                           0.84        61
   macro avg       0.83      0.83      0.83        61
weighted avg       0.84      0.84      0.84        61
```

Best Results GaussianNB Classifier

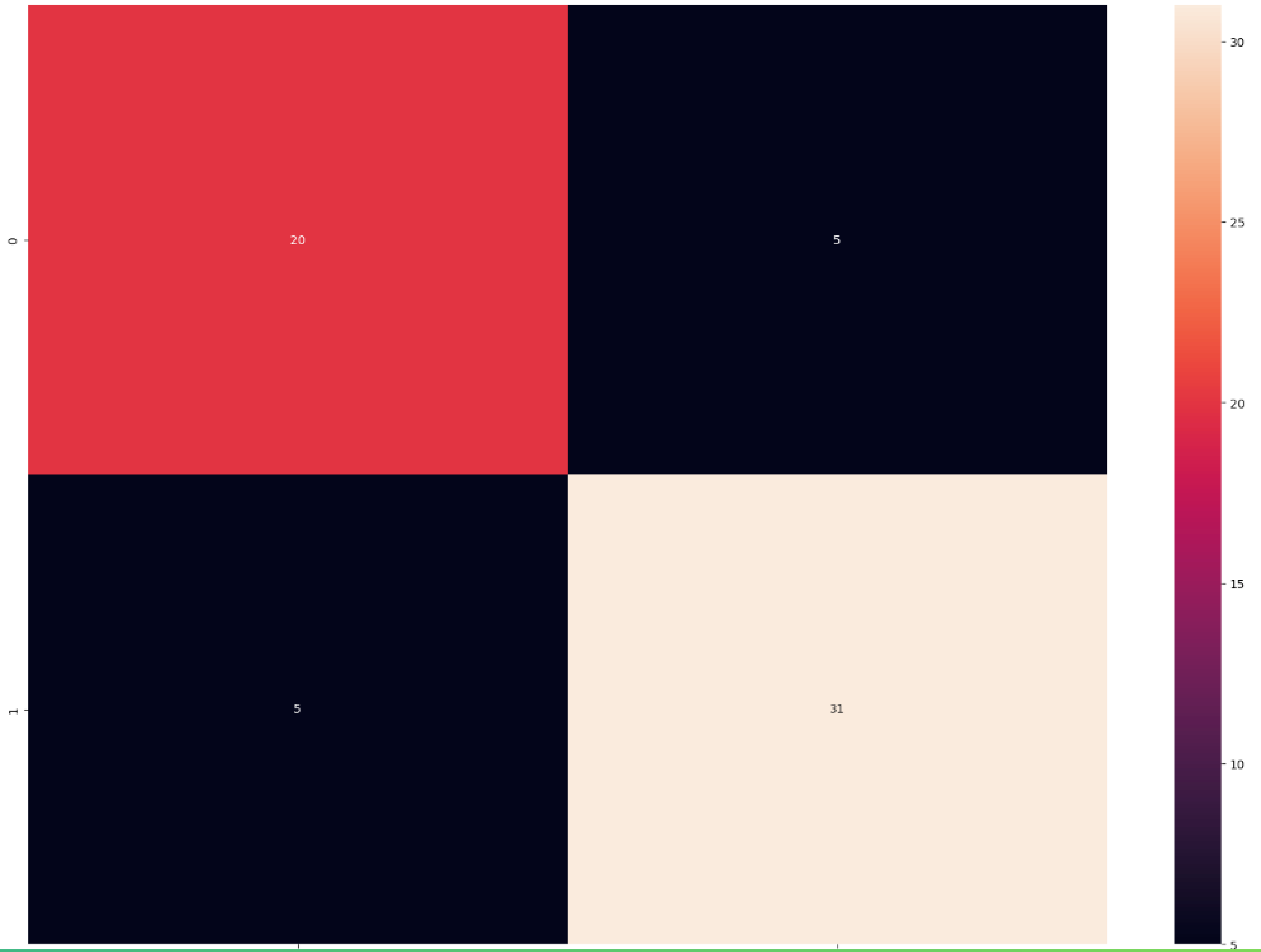
```python
y_pred_train = best_model_NB.predict(X_train)
cm = confusion_matrix(y_train, y_pred_train)
sns.heatmap(cm, annot = True)
```

<AxesSubplot: >




```python
y_pred_test = best_model_NB.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot = True)
```

<AxesSubplot: >



Train **Test**

Confusion Matrics

GaussianNB Classifier

All Best Results

```
Decision Tree by accuracy: 77%
Random Forest by accuracy: 85%
KNeighbor Classifier by accuracy: 80%
SVC by accuracy: 79%
Ada Boost by accuarcy: 85%
Bagging Classifier by accuracy: 89%
Extra Tree Classifier by accuarcy: 84%
XGBoost Classifier by accuarcy: 83%
GaussianNB Classifier by accuary: 84%
Logistic Regression by accuracy: 82%
```

Conclusion

1 Goal summary:

find the best machine learning model, in this project we solved the problem of binary classification

Where there is heart disease or not

2 Main Results

as a results of our project, not bad scores were achived for the following metrics such as Accuracy,

roc_auc score, precision, recall, f1-score and Standart Deviation

3 Models

Of all the algorithms, Bagging Classifier give us the highest accuracy score