Hands-On

Hands-On ini digunakan pada kegiatan Microcredential Associate Data Scientist 2021

PROJECT AKHIR-SPADA

Pertemuan Terakhir pada Microcredential Associate Data Scientist 2021 dianjurkan untuk menganalisis data berdasarkan data yang ada dengan metode sesuai keinginan peserta.

Latihan (1)

Melakukan import library yang dibutuhkan

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from statsmodels.stats.outliers_influence import variance_inflation_factor
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import CategoricalNB
from sklearn.ensemble import GradientBoostingClassifier
```

Load Dataset

In [2]:
```
#Panggil file (load file bernama Stock.csv) dan simpan dalam dataframe
df = pd.read_csv('heart.csv')
```

In [3]:
```
# tampilkan 5 baris data
df.head()
```

Out[3]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAn
0	40	M	ATA	140	289	0	Normal	172	
1	49	F	NAP	160	180	0	Normal	156	
2	37	M	ATA	130	283	0	ST	98	
3	48	F	ASY	138	214	0	Normal	108	
4	54	M	NAP	150	195	0	Normal	122	

In [4]:
```
def describe(df):

  columns=df.columns.to_list()
  ncol=df.describe().columns.to_list()
  ccol=[]
  for i in columns:
    if(ncol.count(i)==0):
      ccol.append(i)
    else:
      continue
  print('----------------------------------------------------------DESKRIPSI DATA--
  print('Nama semua kolom pada dataframe:')
  print(columns)
  print('-------------------------------------------------------------------------
  print('Jumlah kolom dalam dataframe:')
  print(len(columns))
  print('-------------------------------------------------------------------------
  print('Nama semua kolom numerik pada dataframe:')
  print(ncol)
  print('-------------------------------------------------------------------------
  print('Jumlah kolom numerik dalam dataframe:')
  print(len(ncol))
  print('-------------------------------------------------------------------------
  print('Nama semua kolom kategoris pada dataframe:')
  print(ccol)
  print('-------------------------------------------------------------------------
  print('Jumlah kolom kategori dalam dataframe:')
  print(len(ccol))
  print('-------------------------------------------------------------------------
  print('Jumlah Nilai Null di Setiap Kolom:')
  print('')
  print(df.isnull().sum())
  print('')
  print('Jumlah Nilai Unik di Setiap Kolom:')
  print('')
  print(df.nunique())
  print('')
  print('Statistik Dasar dan Ukuran untuk Kolom Numerik:')
  print('')
  print(df.describe().T)
  print('')
  print('')
  print('Metadata Relevan Lainnya Mengenai Kerangka Data:')
  print('')
  print(df.info())
  print('')

describe(df)
```

```
----------------------------------------------------------DESKRIPSI DATA---------
-----------------------------------------------
Nama semua kolom pada dataframe:
['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS', 'Resti
ngECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope', 'HeartDisease']
-------------------------------------------------------------------------------
```

```
------------------------------------------------
Jumlah kolom dalam dataframe:
12
------------------------------------------------------------------------------
------------------------------------------------
Nama semua kolom numerik pada dataframe:
['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak', 'HeartDise
ase']
------------------------------------------------------------------------------
------------------------------------------------
Jumlah kolom numerik dalam dataframe:
7
------------------------------------------------------------------------------
------------------------------------------------
Nama semua kolom kategoris pada dataframe:
['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']
------------------------------------------------------------------------------
------------------------------------------------
Jumlah kolom kategori dalam dataframe:
5
------------------------------------------------------------------------------
------------------------------------------------
Jumlah Nilai Null di Setiap Kolom:

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

Jumlah Nilai Unik di Setiap Kolom:

Age                50
Sex                 2
ChestPainType       4
RestingBP          67
Cholesterol       222
FastingBS           2
RestingECG          3
MaxHR             119
ExerciseAngina      2
Oldpeak            53
ST_Slope            3
HeartDisease        2
dtype: int64

Statistik Dasar dan Ukuran untuk Kolom Numerik:

              count        mean         std   min     25%    50%    75%    max
```

```
Age           918.0   53.510893    9.432617  28.0   47.00   54.0   60.0   77.0
RestingBP     918.0  132.396514   18.514154   0.0  120.00  130.0  140.0  200.0
Cholesterol   918.0  198.799564  109.384145   0.0  173.25  223.0  267.0  603.0
FastingBS     918.0    0.233115    0.423046   0.0    0.00    0.0    0.0    1.0
MaxHR         918.0  136.809368   25.460334  60.0  120.00  138.0  156.0  202.0
Oldpeak       918.0    0.887364    1.066570  -2.6    0.00    0.6    1.5    6.2
HeartDisease  918.0    0.553377    0.497414   0.0    0.00    1.0    1.0    1.0


Metadata Relevan Lainnya Mengenai Kerangka Data:

<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
None
```

Review Data

In [5]:
```
# Melihat Informasi lebih detail mengenai struktur DataFrame dapat dilihat menggu
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
```

In [6]:
```
df.tail()
```

Out[6]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseA
913	45	M	TA	110	264	0	Normal	132	
914	68	M	ASY	144	193	1	Normal	141	
915	57	M	ASY	130	131	0	Normal	115	
916	57	F	ATA	130	236	0	LVH	174	
917	38	M	NAP	138	175	0	Normal	173	

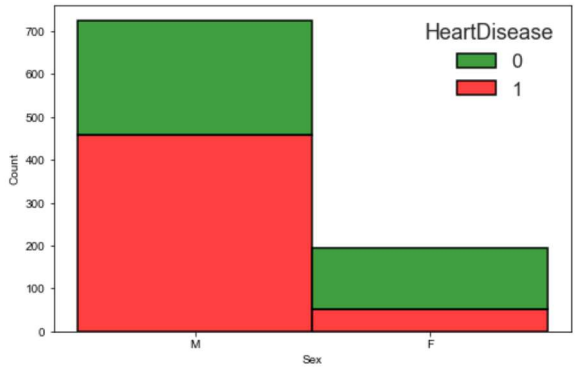
In [7]:
```
df.shape
```

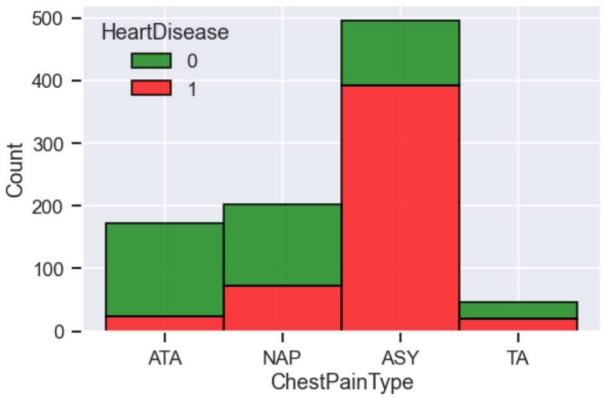
Out[7]:
```
(918, 12)
```

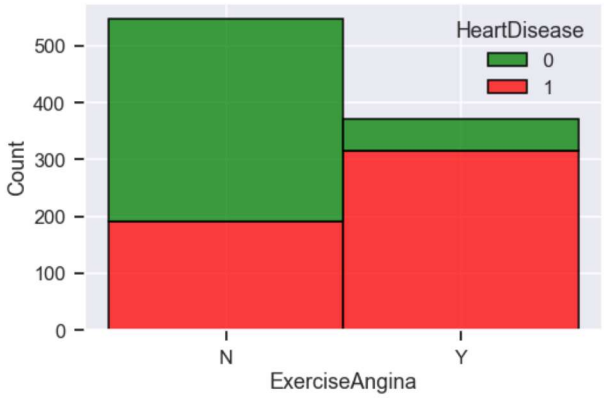
Visualisasikan Data dalam Hal Variabel Target

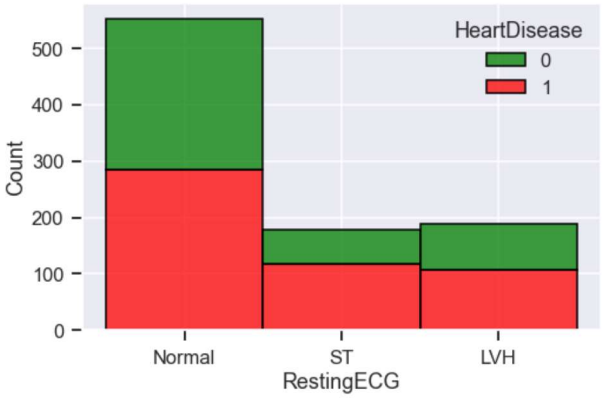
In [8]:
```
oe=['g','r']
fig = plt.figure(figsize=(15,15))

plt.subplot(3,2,1)
plt.style.use('seaborn')
plt.tight_layout()
sns.set_context('talk')
sns.histplot(data=df, x="Sex", hue="HeartDisease",multiple="stack",palette=oe)
#ax.set(xlabel='Sex ', ylabel='Count')

plt.subplot(3,2,2)
plt.style.use('seaborn')
plt.tight_layout()
sns.set_context('talk')
sns.histplot(data=df, x="ChestPainType", hue="HeartDisease",multiple="stack",pale
#ax.set(xlabel='ChestPainType', ylabel='Count')

plt.subplot(3,2,3)
plt.style.use('seaborn')
plt.tight_layout()
sns.set_context('talk')
sns.histplot(data=df, x="ExerciseAngina", hue="HeartDisease",multiple="stack",pal
#ax.set(xlabel='ExerciseAngina', ylabel='Count')

plt.subplot(3,2,4)
plt.style.use('seaborn')
plt.tight_layout()
sns.set_context('talk')
sns.histplot(data=df, x="RestingECG", hue="HeartDisease",multiple="stack",palette
#ax.set(xlabel='RestingECG', ylabel='Count')

plt.subplot(3,2,5)
plt.style.use('seaborn')
plt.tight_layout()
sns.set_context('talk')
sns.histplot(data=df, x="ST_Slope", hue="HeartDisease",multiple="stack",palette=o
#ax.set(xlabel='ST_Slope', ylabel='Count')

plt.subplot(3,2,6)
plt.style.use('seaborn')
plt.tight_layout()
sns.set_context('talk')
sns.histplot(data=df, x="FastingBS", hue="HeartDisease",multiple="stack",palette=
#ax.set(xlabel='FastingBS', ylabel='Count')
```

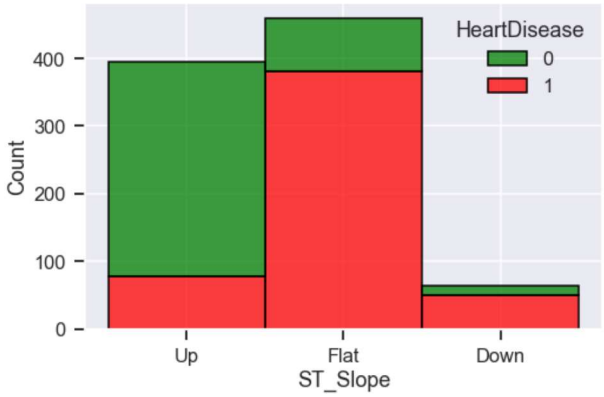
Out[8]:
```
<AxesSubplot:xlabel='FastingBS', ylabel='Count'>
```

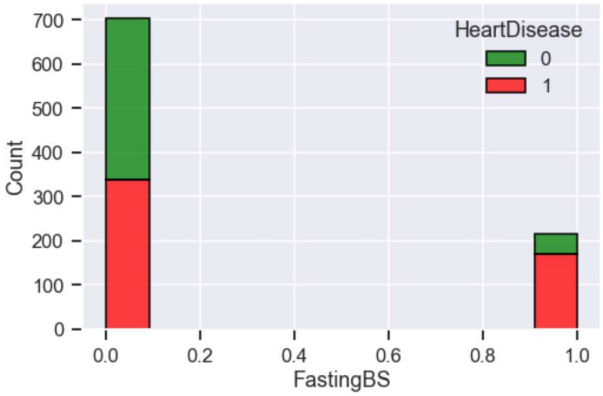










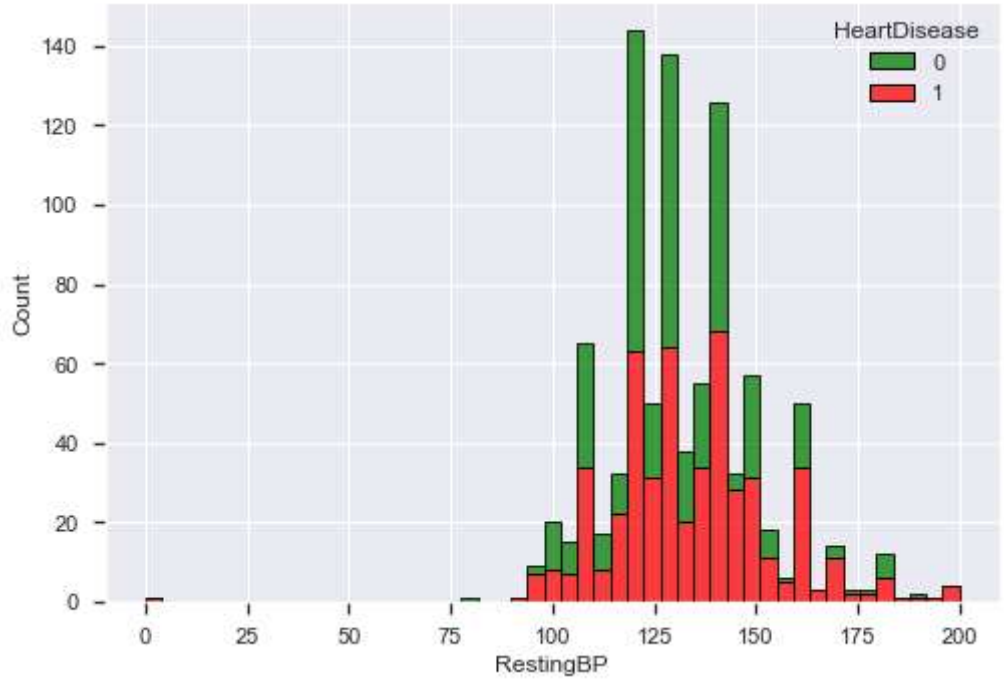


Outlier Detection

In [9]:
```
def outliers(df_column):
  q75, q25 = np.percentile(df_column, [75 ,25])
  iqr = q75 - q25
  print('q75: ',q75)
  print('q25: ',q25)
  print('Inter Quartile Range: ',iqr)
  print('Outliers lie before', q25-1.8*iqr, 'and beyond', q75+1.8*iqr)

  # Usually 1.5 times IQR is considered, but we have used 1.8 for broader range

  print('Number of Rows with Left Extreme Outliers:', len(df[df_column <q25-1.8*i
  print('Number of Rows with Right Extreme Outliers:', len(df[df_column>q75+1.8*i
  plt.tight_layout()
  plt.style.use('seaborn')
  sns.set_context('notebook')
  sns.histplot(data=df, x=df_column, hue="HeartDisease",multiple="stack",palette=
```

In [10]:
```
outliers(df['RestingBP'])
```

```
q75:  140.0
q25:  120.0
Inter Quartile Range:  20.0
Outliers lie before 84.0 and beyond 176.0
Number of Rows with Left Extreme Outliers: 2
Number of Rows with Right Extreme Outliers: 23
```



In [11]:
```
# Since outliers are slightly skewed towards the right extreme side, we will keep
df=df[df.RestingBP>=84]
len(df)
```

Out[11]: 916

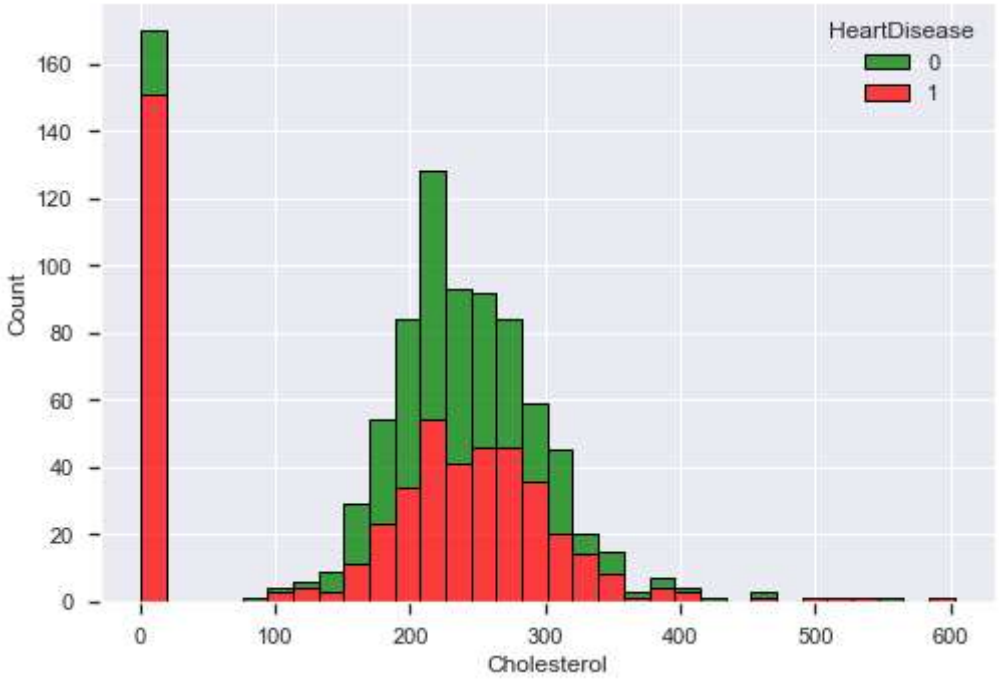
In [12]:
```
df.head()
```

Out[12]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAn
0	40	M	ATA	140	289	0	Normal	172	
1	49	F	NAP	160	180	0	Normal	156	
2	37	M	ATA	130	283	0	ST	98	
3	48	F	ASY	138	214	0	Normal	108	
4	54	M	NAP	150	195	0	Normal	122	

In [13]:
```
outliers(df['Cholesterol'])
# Multiple Values of 0 value are present
#Since Cholesterol value can't be zero, these values are missing and have to be c
```

```
q75:  267.0
q25:  174.75
Inter Quartile Range:  92.25
Outliers lie before 8.699999999999989 and beyond 433.05
Number of Rows with Left Extreme Outliers: 170
Number of Rows with Right Extreme Outliers: 8
```

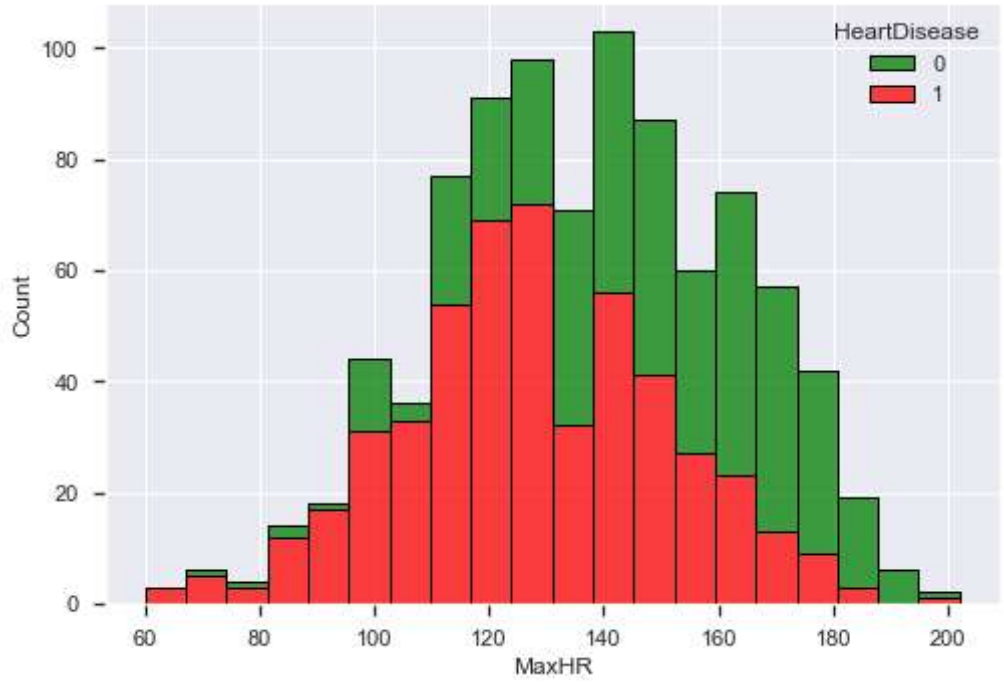


In [14]:
```
df=df[df.Cholesterol<=500]
df.head()
len(df)
```

Out[14]: 912

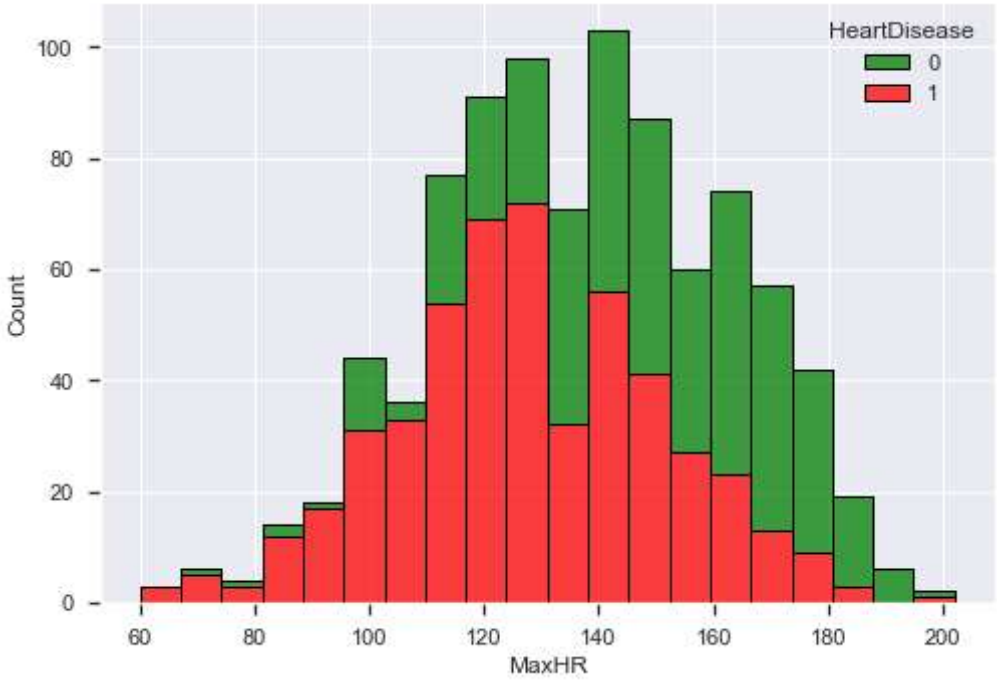
In [15]:
```
outliers(df['MaxHR'])
```

```
q75:  156.0
q25:  120.0
Inter Quartile Range:  36.0
Outliers lie before 55.2 and beyond 220.8
Number of Rows with Left Extreme Outliers: 0
Number of Rows with Right Extreme Outliers: 0
```



In [16]:
```
outliers(df['MaxHR'])
```

```
q75:  156.0
q25:  120.0
Inter Quartile Range:  36.0
Outliers lie before 55.2 and beyond 220.8
Number of Rows with Left Extreme Outliers: 0
Number of Rows with Right Extreme Outliers: 0
```



In [17]:
```
df[df.Cholesterol==0].mean() #Effort to understand how data looks like where valu
```

Out[17]:
```
Age             56.264706
RestingBP      130.735294
Cholesterol      0.000000
FastingBS        0.523529
MaxHR          121.682353
Oldpeak          0.814706
HeartDisease     0.888235
dtype: float64
```

In [18]:
```
df[df.Cholesterol>0].mean()
```

Out[18]:
```
Age              52.889488
RestingBP       133.055256
Cholesterol     242.970350
FastingBS         0.167116
MaxHR           140.247978
Oldpeak           0.902965
HeartDisease      0.475741
dtype: float64
```

In [19]:
```
print('Mean: ',df['Cholesterol'].mean())
print('Median: ',df['Cholesterol'].median())
```

```
Mean:  197.6798245614035
Median:  222.5
```

In [20]:
```
df[df['Cholesterol']>0].Cholesterol.mean()
```

Out[20]:
```
242.97035040431265
```

In [21]:
```
df.describe().T
```

Out[21]:

	count	mean	std	min	25%	50%	75%	max
Age	912.0	53.518640	9.425989	28.0	47.00	54.00	60.00	77.0
RestingBP	912.0	132.622807	17.943938	92.0	120.00	130.00	140.00	200.0
Cholesterol	912.0	197.679825	106.763321	0.0	173.75	222.50	266.25	491.0
FastingBS	912.0	0.233553	0.423323	0.0	0.00	0.00	0.00	1.0
MaxHR	912.0	136.787281	25.519987	60.0	120.00	138.00	156.00	202.0
Oldpeak	912.0	0.886513	1.068172	-2.6	0.00	0.55	1.50	6.2
HeartDisease	912.0	0.552632	0.497495	0.0	0.00	1.00	1.00	1.0

Ubah Variabel Kategori menjadi Numerik dengan One Hot Encoding

In [22]:
```
def OHE(dfcolumn):
  global df
  dfcolumn.nunique()
  len(df.columns)
  finallencol = (dfcolumn.nunique() - 1) + (len(df.columns)-1)
  dummies = pd.get_dummies(dfcolumn, drop_first=True, prefix=dfcolumn.name)
  df=pd.concat([df,dummies],axis='columns')
  df.drop(columns=dfcolumn.name,axis=1,inplace=True) # We have to drop columns to
  if(finallencol==len(df.columns)):
    print('One Hot Encoding was successful!')
    print('')
  else:
    print('Error in OHE XXXX')
  return df
```

In [23]:
```
OHE(df['ChestPainType'])
OHE(df['Sex'])
OHE(df['RestingECG'])
OHE(df['ExerciseAngina'])
OHE(df['ST_Slope'])
```

```
One Hot Encoding was successful!

One Hot Encoding was successful!

One Hot Encoding was successful!

One Hot Encoding was successful!

One Hot Encoding was successful!
```

Out[23]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease	ChestPainType_ATA
0	40	140	289	0	172	0.0	0	1
1	49	160	180	0	156	1.0	1	0
2	37	130	283	0	98	0.0	0	1
3	48	138	214	0	108	1.5	1	0
4	54	150	195	0	122	0.0	0	0
...	...	...	...	...	...	...	...	...
913	45	110	264	0	132	1.2	1	0
914	68	144	193	1	141	3.4	1	0
915	57	130	131	0	115	1.2	1	0
916	57	130	236	0	174	0.0	1	1
917	38	138	175	0	173	0.0	0	0

912 rows × 16 columns

Transformasi Data Sebelum Pembuatan Model

In [24]:
```
# Imupute Values for 0 Cholesterol, Method Used: KNN-Imputation

df['Cholesterol'].replace(to_replace = 0, value =np.nan, inplace=True)
KNN_imputed = KNNImputer(n_neighbors=5)
I=KNN_imputed.fit_transform(df)
Cholesterol=[]
for i in range(0,len(df)):
  Cholesterol.append(I[i][2])
df['Cholesterol']=Cholesterol
```

In [25]:
```
# Find VIF (Variance Inflation Factor) To Check for collinearity among independer
vif = df.copy()
vif.drop(columns='HeartDisease',axis=1,inplace=True)
vif_data = pd.DataFrame()
vif_data["feature"] = vif.columns
vif_data["VIF"] = [variance_inflation_factor(vif.values, i)
                           for i in range(len(vif.columns))]
```

In [26]:
```
vif_data # Standard practice is to deal with variables with VIF >= 10
```

Out[26]:

	feature	VIF
0	Age	33.387612
1	RestingBP	53.889014
2	Cholesterol	23.587329
3	FastingBS	1.431929
4	MaxHR	27.855792
5	Oldpeak	2.569567
6	ChestPainType_ATA	1.841542
7	ChestPainType_NAP	1.626664
8	ChestPainType_TA	1.184258
9	Sex_M	4.895356
10	RestingECG_Normal	3.820894
11	RestingECG_ST	1.977411
12	ExerciseAngina_Y	2.634832
13	ST_Slope_Flat	8.388073
14	ST_Slope_Up	9.226993

In [27]:
```
# Scale Data For Higher Efficiency

from sklearn.preprocessing import StandardScaler # Converts Columnar Data into St
scaler=StandardScaler()
scaler.fit(vif)
scaled_data=scaler.transform(vif)
scaled_data
```

Out[27]:
```
array([[-1.43497494,  0.41135011,  0.90884654, ..., -0.82435309,
        -0.99780942,  1.14405453],
       [-0.47964407,  1.52654424, -1.2497362 , ..., -0.82435309,
         1.00219539, -0.87408421],
       [-1.75341857, -0.14624695,  0.79002547, ..., -0.82435309,
        -0.99780942,  1.14405453],
       ...,
       [ 0.36953893, -0.14624695, -2.22010825, ...,  1.21307243,
         1.00219539, -0.87408421],
       [ 0.36953893, -0.14624695, -0.14073956, ..., -0.82435309,
         1.00219539, -0.87408421],
       [-1.64727069,  0.2998307 , -1.34875375, ..., -0.82435309,
        -0.99780942,  1.14405453]])
```

In [28]:
```
from sklearn.decomposition import PCA # Reduce Dimensions by Principal Component
pca=PCA(n_components=11)
pca.fit(scaled_data)
x_pca=pca.transform(scaled_data)
x_pca
```

Out[28]:
```
array([[ 2.96196536, -0.32607017,  1.65159492, ..., -0.48242366,
        -0.96897897, -0.29772444],
       [ 0.284037  , -1.05263117, -1.10757972, ..., -1.99653336,
        -0.69976878,  0.44105539],
       [ 1.5284447 ,  2.19764344,  1.21517963, ...,  0.81540077,
        -1.25161928,  0.37361669],
       ...,
       [-1.51604508, -1.64883654, -0.71441383, ..., -0.5043577 ,
         0.13243627,  0.33962526],
       [ 1.01169527,  0.3660804 ,  1.5499496 , ..., -1.26051025,
        -0.1881711 , -1.24151054],
       [ 2.67043799, -0.61322532, -2.16689401, ..., -0.922637  ,
        -0.80415661,  0.55521724]])
```

Model Akhir dan Algoritma yang Digunakan

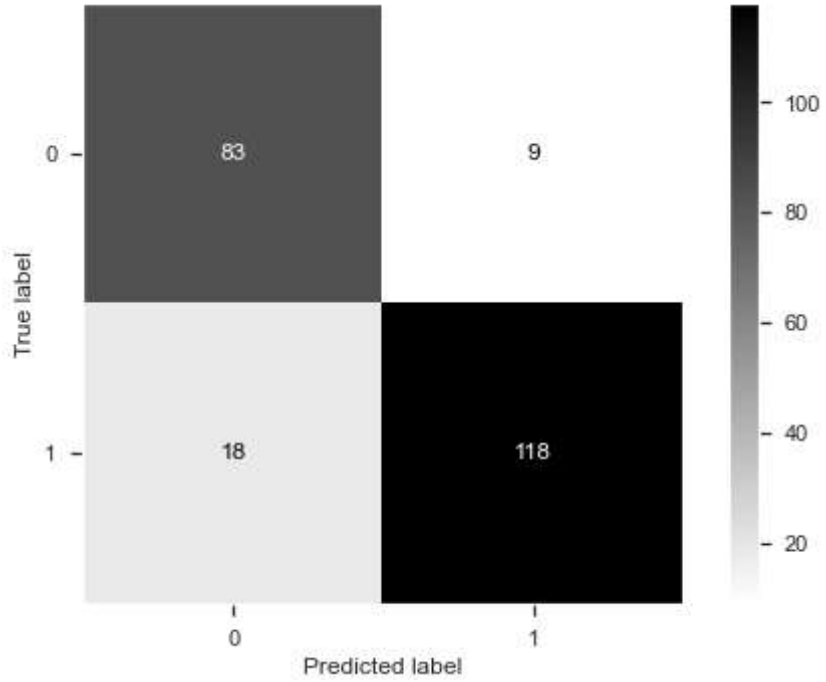
In [29]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_pca, df['HeartDisease'], te
```

Mendukung Mesin Vektor (Kernel Fungsi Basis Radial)

In [30]:
```
from sklearn.svm import SVC
classifier = SVC(kernel='rbf', random_state=0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
```

In [31]:
```
from sklearn.metrics import classification_report
import warnings
warnings.filterwarnings("ignore")
print(classification_report(y_test, y_pred))
print('')
print('------------------------')
print('Confusion Matrix')
print('------------------------')
print('')
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
print(confusion_matrix(y_test, y_pred))
plot_confusion_matrix(classifier, x_test, y_test,cmap="binary")
plt.grid(False)
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86        92
           1       0.93      0.87      0.90       136

    accuracy                           0.88       228
   macro avg       0.88      0.88      0.88       228
weighted avg       0.89      0.88      0.88       228


------------------------
Confusion Matrix
------------------------

[[ 83   9]
 [ 18 118]]
```



Fungsi Regresi Logistik

In [32]:
```
logreg = LogisticRegression()
logreg.fit(x_train, y_train)
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(log
print('')
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy of logistic regression classifier on test set: 0.89

              precision    recall  f1-score   support

           0       0.82      0.90      0.86        92
           1       0.93      0.87      0.90       136

    accuracy                           0.88       228
   macro avg       0.88      0.88      0.88       228
weighted avg       0.89      0.88      0.88       228

[[ 83   9]
 [ 18 118]]
```

Random Forest Classifier

In [33]:
```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 103, stop = 300, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

In [34]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
rf = RandomForestClassifier()
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

Out[34]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [103, 124, 146, 168,
                                                         190, 212, 234, 256,
                                                         278, 300]},
                   random_state=42, verbose=2)
```

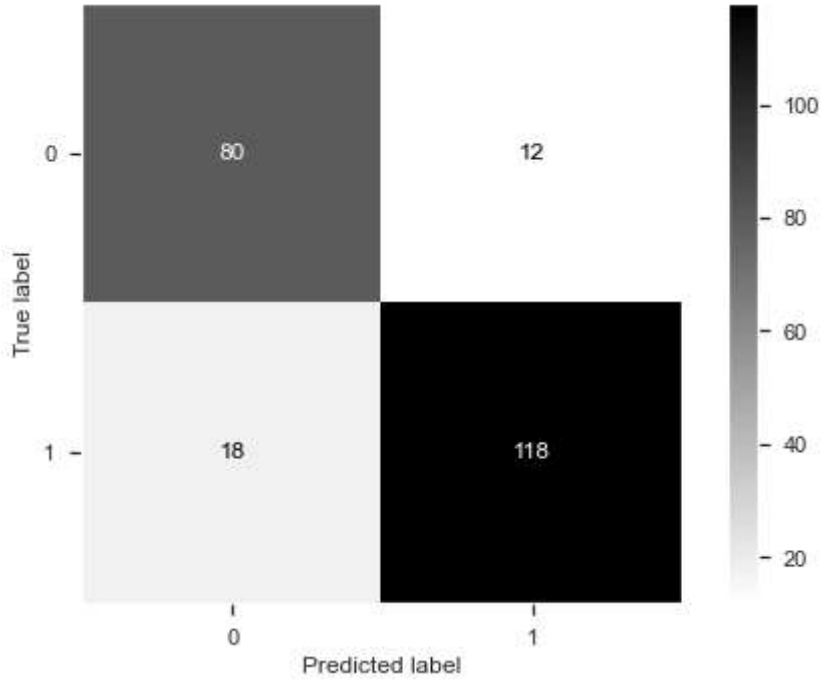
In [35]:
```
rf_random.best_params_
```

Out[35]:
```
{'n_estimators': 168,
 'min_samples_split': 10,
 'min_samples_leaf': 2,
 'max_features': 'auto',
 'max_depth': None,
 'bootstrap': False}
```

Confusion Matrix Random Forest

In [36]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_pca, df['HeartDisease'], te
clf=RandomForestClassifier(n_estimators=124,min_samples_split= 2,
                           min_samples_leaf= 1,max_features='sqrt',max_depth=None
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)
import warnings
warnings.filterwarnings("ignore")
print(classification_report(y_test, y_pred))
print('')
print('------------------------')
print('Confusion Matrix')
print('------------------------')
print('')
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
print(confusion_matrix(y_test, y_pred))
plot_confusion_matrix(clf, x_test, y_test,cmap="binary")
plt.grid(False)
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.82      0.87      0.84        92
           1       0.91      0.87      0.89       136

    accuracy                           0.87       228
   macro avg       0.86      0.87      0.86       228
weighted avg       0.87      0.87      0.87       228


------------------------
Confusion Matrix
------------------------

[[ 80  12]
 [ 18 118]]
```



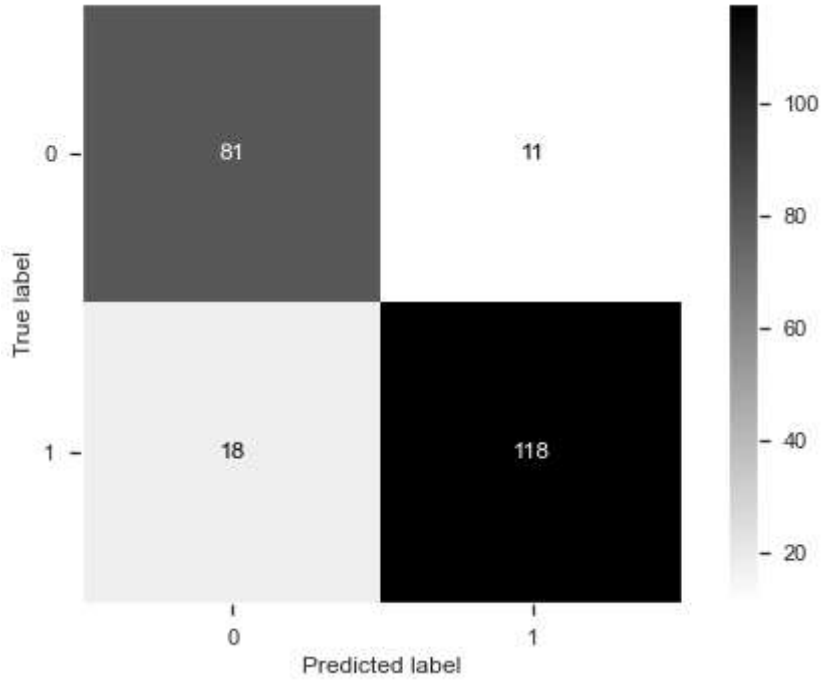
Support Vector Machine (Radial Basis Function Kernel) with Grid Search CV

In [37]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3, cv=5)
grid.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.533 total time=
0.0s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.540 total time=
0.0s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.540 total time=
0.0s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.540 total time=
0.0s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.537 total time=
0.0s
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.869 total time=
0.0s
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.861 total time=
0.0s
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.861 total time=
0.0s
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.825 total time=
0.0s
```

In [38]:
```
print(grid.best_params_)
print(grid.best_estimator_)
y_pred = grid.predict(x_test)
print(classification_report(y_test, y_pred))
print('')
print('------------------------')
print('Confusion Matrix')
print('------------------------')
print('')
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
print(confusion_matrix(y_test, y_pred))
plot_confusion_matrix(grid, x_test, y_test,cmap="binary")
plt.grid(False)
plt.show()
```

```
{'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
SVC(C=1000, gamma=0.001)
              precision    recall  f1-score   support

           0       0.82      0.88      0.85        92
           1       0.91      0.87      0.89       136

    accuracy                           0.87       228
   macro avg       0.87      0.87      0.87       228
weighted avg       0.88      0.87      0.87       228


------------------------
Confusion Matrix
------------------------

[[ 81  11]
 [ 18 118]]
```



Neural Network Using Tensor flow

In [39]:
```
import tensorflow as tf
tf.random.set_seed(0)


model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(
    loss=tf.keras.losses.binary_crossentropy,
    optimizer=tf.keras.optimizers.Adam(lr=0.03),
    metrics=[
        tf.keras.metrics.BinaryAccuracy(name='accuracy'),
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]
)

history = model.fit(x_train, y_train, epochs=100)
```

```
Epoch 1/100
22/22 [==============================] - 1s 2ms/step - loss: 0.4474 - accura
cy: 0.8041 - precision: 0.8047 - recall: 0.8397
Epoch 2/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3378 - accura
cy: 0.8699 - precision: 0.8514 - recall: 0.9185
Epoch 3/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3214 - accura
cy: 0.8801 - precision: 0.8743 - recall: 0.9076
Epoch 4/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3046 - accura
cy: 0.8787 - precision: 0.8645 - recall: 0.9185
Epoch 5/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3017 - accura
cy: 0.8874 - precision: 0.8819 - recall: 0.9130
Epoch 6/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2981 - accura
cy: 0.8787 - precision: 0.8780 - recall: 0.8995
Epoch 7/100
```

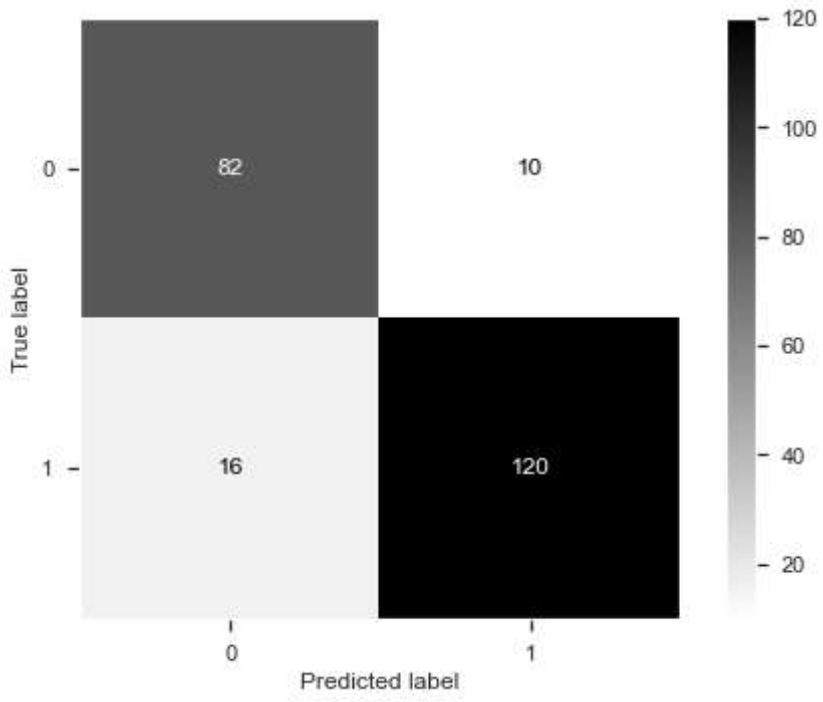
In [40]:
```
predictions = model.predict(x_test)
y_pred = [
    1 if prob > 0.5 else 0 for prob in np.ravel(predictions)
]
print(classification_report(y_test, y_pred))
print('')
print('------------------------')
print('Confusion Matrix')
print('------------------------')
print('')
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.73      0.89      0.80        92
           1       0.91      0.78      0.84       136

    accuracy                           0.82       228
   macro avg       0.82      0.84      0.82       228
weighted avg       0.84      0.82      0.83       228


------------------------
Confusion Matrix
------------------------

[[ 82  10]
 [ 30 106]]
```

KNN Classifier

In [41]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_pca, df['HeartDisease'], te
knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(x_train, y_train)
y_pred=knn.predict(x_test)
print(classification_report(y_test, y_pred))
print('')
print('------------------------')
print('Confusion Matrix')
print('------------------------')
print('')
print(confusion_matrix(y_test, y_pred))
plot_confusion_matrix(knn, x_test, y_test,cmap="binary")
plt.grid(False)
plt.show()
```

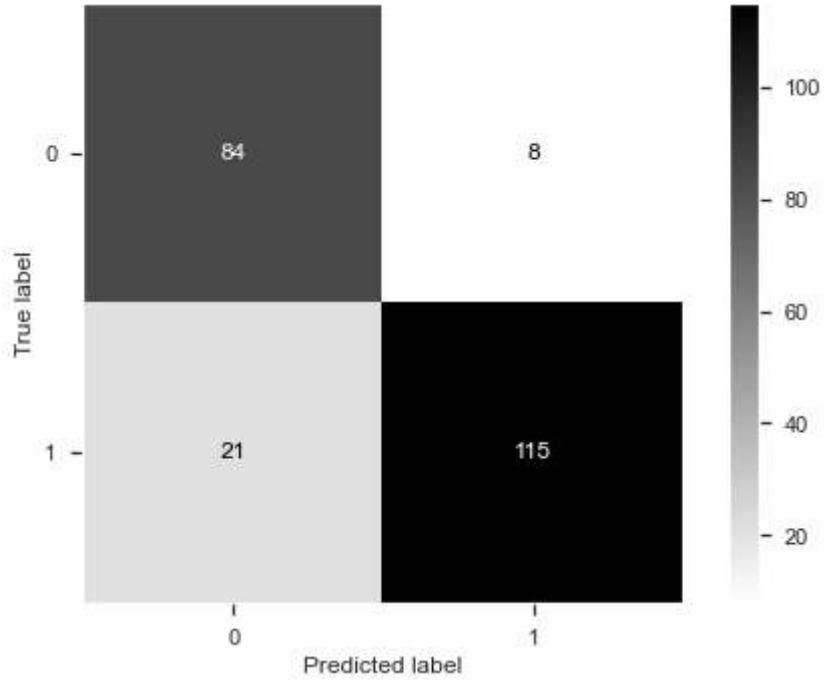
```
              precision    recall  f1-score   support

           0       0.84      0.89      0.86        92
           1       0.92      0.88      0.90       136

    accuracy                           0.89       228
   macro avg       0.88      0.89      0.88       228
weighted avg       0.89      0.89      0.89       228


------------------------
Confusion Matrix
------------------------

[[ 82  10]
 [ 16 120]]
```



Gaussian Navie Bayes Algorithm

In [42]:
```
gnb = GaussianNB()
gnb.fit(x_train, y_train)
y_pred=gnb.predict(x_test)
print(classification_report(y_test, y_pred))
print('')
print('------------------------')
print('Confusion Matrix')
print('------------------------')
print('')
print(confusion_matrix(y_test, y_pred))
plot_confusion_matrix(gnb, x_test, y_test,cmap="binary")
plt.grid(False)
plt.show()
```

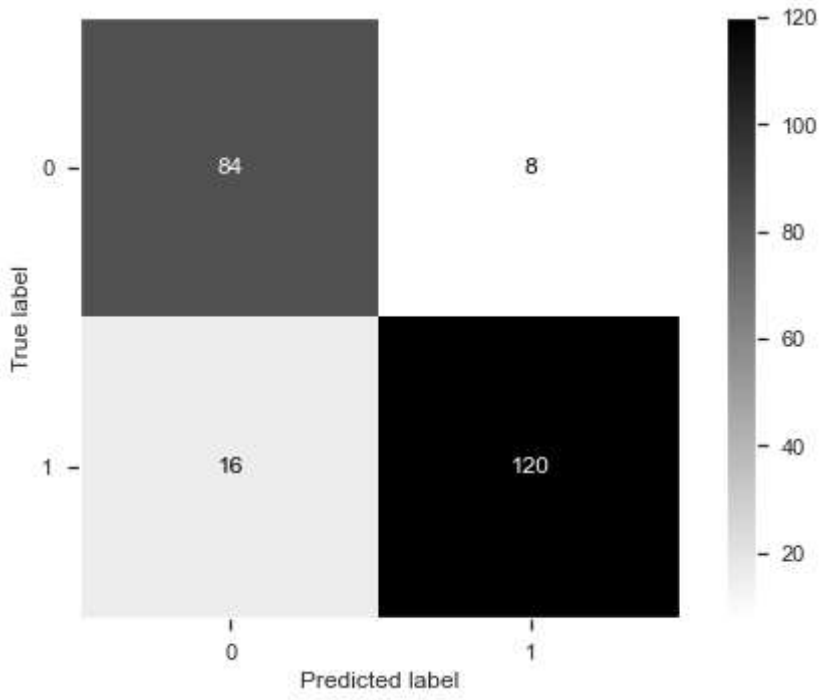
```
              precision    recall  f1-score   support

           0       0.80      0.91      0.85        92
           1       0.93      0.85      0.89       136

    accuracy                           0.87       228
   macro avg       0.87      0.88      0.87       228
weighted avg       0.88      0.87      0.87       228


------------------------
Confusion Matrix
------------------------

[[ 84   8]
 [ 21 115]]
```



Gradient Boosting Algorithm

In [43]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_pca, df['HeartDisease'], te
clff = GradientBoostingClassifier(n_estimators=100, learning_rate=0.2, max_depth=
clff.fit(x_train, y_train)
y_pred=clff.predict(x_test)
print(classification_report(y_test, y_pred))
print('')
print('------------------------')
print('Confusion Matrix')
print('------------------------')
print('')
print(confusion_matrix(y_test, y_pred))
plot_confusion_matrix(clff, x_test, y_test,cmap="binary")
plt.grid(False)
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.84      0.91      0.87        92
           1       0.94      0.88      0.91       136

    accuracy                           0.89       228
   macro avg       0.89      0.90      0.89       228
weighted avg       0.90      0.89      0.90       228


------------------------
Confusion Matrix
------------------------

[[ 84   8]
 [ 16 120]]
```



Berikan Kesimpulan Anda!

Semua model dapat diperbaiki dengan penyetelan parameter hiper yang lebih ketat. Hal ini dapat dicapai dengan pemilihan acak, metode brute force, dll. Berbagai pengklasifikasi lain juga dapat digunakan, tetapi pengklasifikasi paling standar telah dipertimbangkan dalam notebook ini.

Recall terbaik yang dicapai adalah 91% dengan dan rata-rata 89,5%. Akurasi 94% juga telah dicapai.

Merekomendasikan praktik standar untuk transformasi data, deteksi outlier, dan substitusi nilai nol telah dimasukkan dalam notebook ini

FRISKA ANDALUSIA-MICROCREDENTIAL AI-TELKOM UNIVERSITY