```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score
from sklearn import linear_model, tree, ensemble
```

```
dataframe=pd.read_csv("/content/Heart_Disease_Prediction.csv")
dataframe=dataframe.dropna()
dataframe.head(10)
```

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST
0	70	1	4	130	322	0	2	109	0	2.4	2
1	67	0	3	115	564	0	2	160	0	1.6	2
2	57	1	2	124	261	0	0	141	0	0.3	1
3	64	1	4	128	263	0	0	105	1	0.2	2
4	74	0	2	120	269	0	2	121	1	0.2	1
5	65	1	4	120	177	0	0	140	0	0.4	1
6	56	1	3	130	256	1	2	142	1	0.6	2
7	59	1	4	110	239	0	2	142	1	1.2	2

```
dataframe=pd.read_csv("/content/Heart_Disease_Prediction.csv")
dataframe=dataframe.dropna()
dataframe.head(10)
```

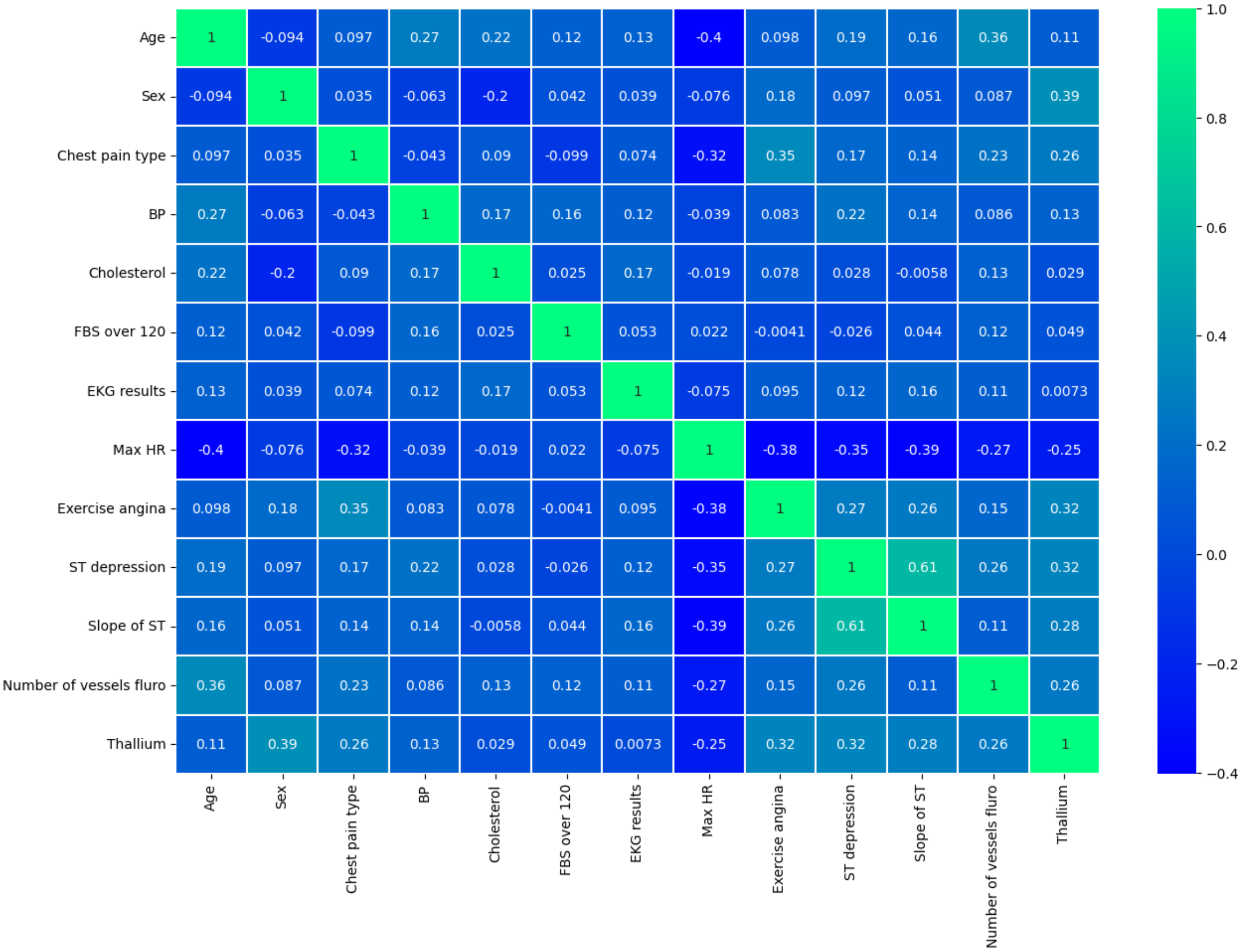
	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST
0	70	1	4	130	322	0	2	109	0	2.4	2
1	67	0	3	115	564	0	2	160	0	1.6	2
2	57	1	2	124	261	0	0	141	0	0.3	1
3	64	1	4	128	263	0	0	105	1	0.2	2
4	74	0	2	120	269	0	2	121	1	0.2	1
5	65	1	4	120	177	0	0	140	0	0.4	1
6	56	1	3	130	256	1	2	142	1	0.6	2
7	59	1	4	110	239	0	2	142	1	1.2	2

```
dataframe.isna().sum()
```

```
Age                        0
Sex                        0
Chest pain type            0
BP                         0
Cholesterol                0
FBS over 120               0
EKG results                0
Max HR                     0
Exercise angina            0
ST depression              0
Slope of ST                0
Number of vessels fluro    0
Thallium                   0
Heart Disease              0
dtype: int64
```

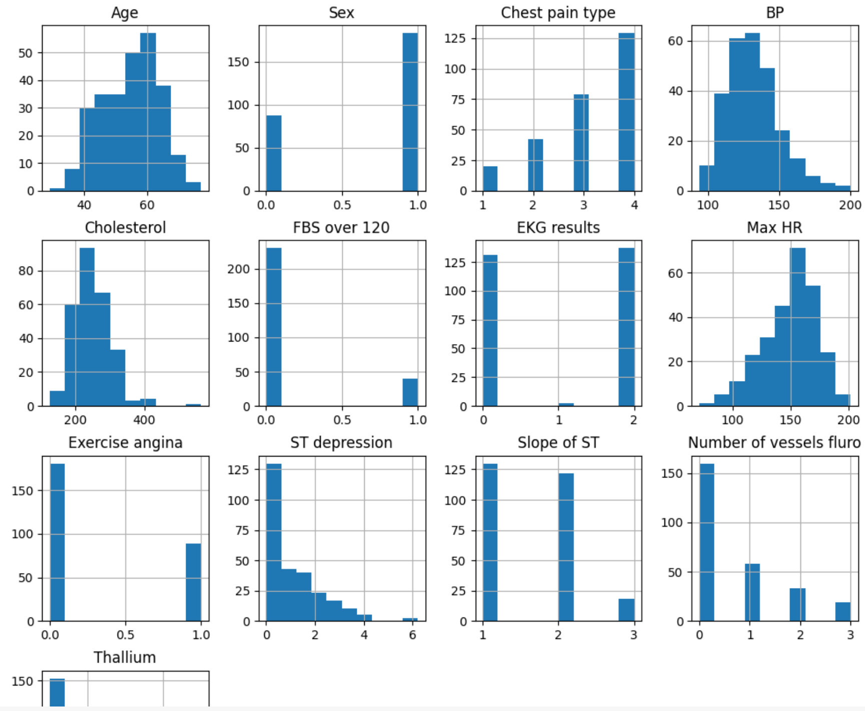
```
plt.figure(figsize=(15,10))
sns.heatmap(dataframe.corr(),linewidth=.01,annot=True,cmap="winter")
plt.show()
plt.savefig('correlationfigure')
```

```
<ipython-input-25-762dfe51a80e>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  sns.heatmap(dataframe.corr(),linewidth=.01,annot=True,cmap="winter")
```



```
<Figure size 640x480 with 0 Axes>
```

```
dataframe.hist(figsize=(12,12))
plt.savefig('featuresplot')
```

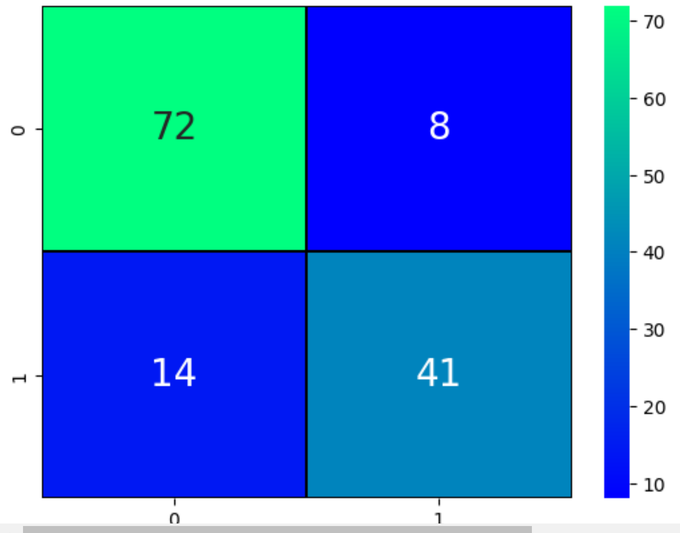


```
X=dataframe.iloc[:,:-1].values
y=dataframe.iloc[:,-1].values
```

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=40)

# For X_train and y_train:
train_data = pd.DataFrame(X_train)
train_data['target'] = y_train

# Remove rows with NaN values
train_data_cleaned = train_data.dropna()

# Separate X_train and y_train from the cleaned data
X_train = train_data_cleaned.drop('target', axis=1)
y_train = train_data_cleaned['target']

# For X_test and y_test:
test_data = pd.DataFrame(X_test)
test_data['target'] = y_test

# Remove rows with NaN values
test_data_cleaned = test_data.dropna()

# Separate X_test and y_test from the cleaned data
X_test = test_data_cleaned.drop('target', axis=1)
y_test = test_data_cleaned['target']

# Now X_train_cleaned, y_train_cleaned, X_test_cleaned, and y_test_cleaned do not contain NaN values and are 1-dimensional.
```

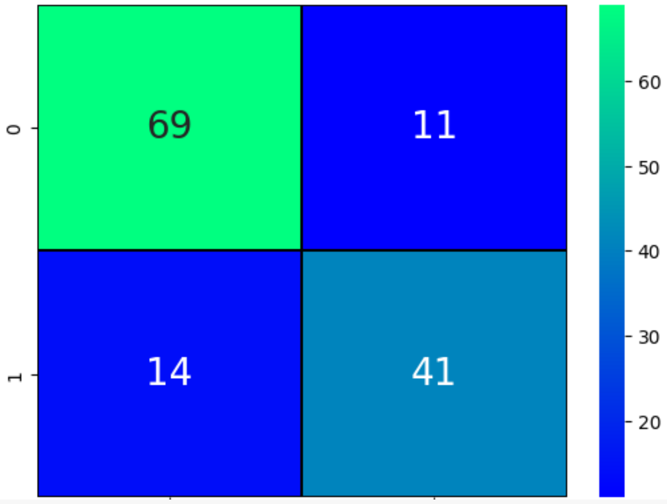
```
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
model1=lr.fit(X_train,y_train)
prediction1=model1.predict(X_test)
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,prediction1)
cm
sns.heatmap(cm, annot=True,cmap='winter',linewidths=0.3, linecolor='black',annot_kws={"size": 20})
TP=cm[0][0]
TN=cm[1][1]
FN=cm[1][0]
FP=cm[0][1]
precision1=TP/(TP+FP)
recall1=TP/(TP+FN)


print('Testing Accuracy for Logistic Regression:',(TP+TN)/(TP+TN+FN+FP))
print('Testing Sensitivity for Logistic Regression:',recall1)
print('Testing Specificity for Logistic Regression:',(TN/(TN+FP)))
print('Testing Precision for Logistic Regression:',precision1)
print('Testing F1-measure for Logistic Regression:',(2*precision1*recall1)/(precision1+recall1))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Testing Accuracy for Logistic Regression: 0.837037037037037
Testing Sensitivity for Logistic Regression: 0.8372093023255814
Testing Specificity for Logistic Regression: 0.8367346938775511
Testing Precision for Logistic Regression: 0.9
Testing F1-measure for Logistic Regression: 0.8674698795180723
```



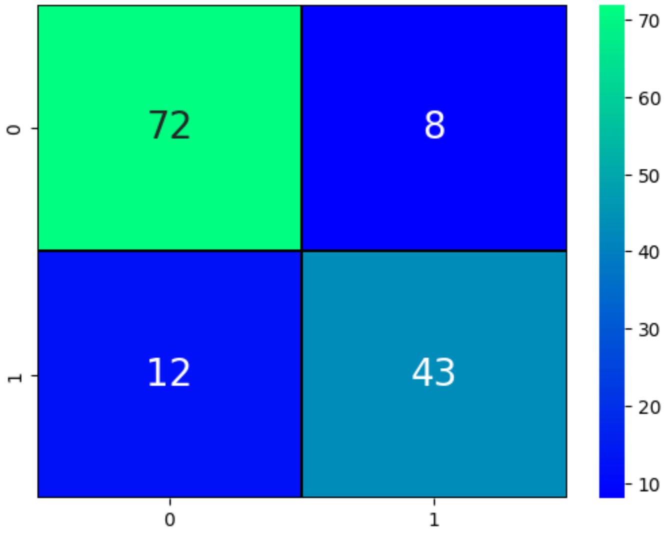
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
rfc=RandomForestClassifier()
model3 = rfc.fit(X_train, y_train)
prediction3 = model3.predict(X_test)
cm3=confusion_matrix(y_test, prediction3)
sns.heatmap(cm3, annot=True,cmap='winter',linewidths=0.3, linecolor='black',annot_kws={"size": 20})
TP=cm3[0][0]
TN=cm3[1][1]
FN=cm3[1][0]
FP=cm3[0][1]
precision3=TP/(TP+FP)
recall3=TP/(TP+FN)
print(round(accuracy_score(prediction3,y_test)*100,2))
print('Testing Accuracy for Random Forest:',(TP+TN)/(TP+TN+FN+FP))
print('Testing Sensitivity for Random Forest:',recall3)
print('Testing Specificity for Random Forest:',(TN/(TN+FP)))
print('Testing Precision for Random Forest:',precision3)
print('Testing F1-measure for Random Forest:',(2*precision3*recall3)/(precision3+recall3))
```

```
81.48
Testing Accuracy for Random Forest: 0.8148148148148148
Testing Sensitivity for Random Forest: 0.8313253012048193
Testing Specificity for Random Forest: 0.7884615384615384
Testing Precision for Random Forest: 0.8625
Testing F1-measure for Random Forest: 0.8466257668711658
```



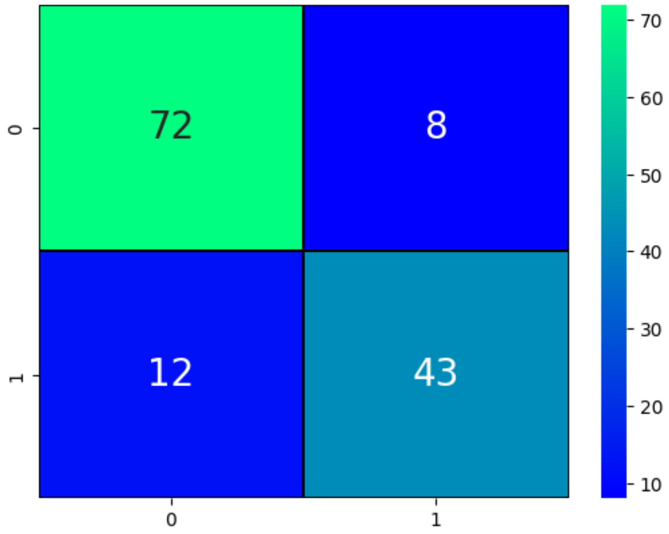
```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
svm = SVC(C=12, kernel='linear')
model4 = svm.fit(X_train, y_train)
prediction4 = model4.predict(X_test)
cm4 = confusion_matrix(y_test, prediction4)
sns.heatmap(cm4, annot=True, cmap='winter', linewidths=0.3, linecolor='black', annot_kws={"size": 20})

TP = cm4[0][0]
TN = cm4[1][1]
FN = cm4[1][0]
FP = cm4[0][1]
precision4=TP/(TP+FP)
recall4=TP/(TP+FN)

print('Testing Accuracy for SVM:', (TP + TN) / (TP + TN + FN + FP))
print('Testing Sensitivity for SVM:', recall4)
print('Testing Specificity for SVM:', (TN / (TN + FP)))
print('Testing Precision for SVM:', precision4)
print('Testing F1-measure for SVM:',(2*precision4*recall4)/(precision4+recall4))
```

```
Testing Accuracy for SVM: 0.8518518518518519
Testing Sensitivity for SVM: 0.8571428571428571
Testing Specificity for SVM: 0.8431372549019608
Testing Precision for SVM: 0.9
Testing F1-measure for SVM: 0.8780487804878048
```



```
from sklearn.naive_bayes import GaussianNB


nb = GaussianNB()

model5=nb.fit(X_train,y_train)
prediction5 = nb.predict(X_test)
cm5=confusion_matrix(y_test, prediction5)
#score_nb = round(accuracy_score(y_pred_nb,y_test)*100,2)
sns.heatmap(cm4, annot=True, cmap='winter', linewidths=0.3, linecolor='black', annot_kws={"size": 20})

TP = cm5[0][0]
TN = cm5[1][1]
FN = cm5[1][0]
FP = cm5[0][1]
precision5=TP/(TP+FP)
recall5=TP/(TP+FN)

print('Testing Accuracy for Naive Bayes:', (TP + TN) / (TP + TN + FN + FP))
print('Testing Sensitivity for Naive Bayes:', recall5)
print('Testing Specificity for Naive Bayes:', (TN / (TN + FP)))
print('Testing Precision for Naive Bayes:', precision5)
print('Testing F1-measure for Naive Bayes:',(2*precision5*recall5)/(precision5+recall5))

#print("The accuracy score achieved using Naive Bayes is: "+str(score_nb)+" %")
```

```
Testing Accuracy for Naive Bayes: 0.8518518518518519
Testing Sensitivity for Naive Bayes: 0.8488372093023255
Testing Specificity for Naive Bayes: 0.8571428571428571
Testing Precision for Naive Bayes: 0.9125
Testing F1-measure for Naive Bayes: 0.8795180722891567
```



```
from sklearn.tree import DecisionTreeClassifier

max_accuracy = 0
for x in range(200):
    dt = DecisionTreeClassifier(random_state=x)
    dt.fit(X_train,y_train)
    y_pred_dt = dt.predict(X_test)
    current_accuracy = round(accuracy_score(y_pred_dt,y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

#print(max_accuracy)
#print(best_x)


dt = DecisionTreeClassifier(random_state=best_x)
model6=dt.fit(X_train,y_train)
prediction6= dt.predict(X_test)
cm6=confusion_matrix(y_test, prediction6)
sns.heatmap(cm4, annot=True, cmap='winter', linewidths=0.3, linecolor='black', annot_kws={"size": 20})

TP = cm6[0][0]
TN = cm6[1][1]
FN = cm6[1][0]
FP = cm6[0][1]
precision6=TP/(TP+FP)
recall6=TP/(TP+FN)
print('Testing Accuracy for DecisionTreeClassifier:', (TP + TN) / (TP + TN + FN + FP))
print('Testing Sensitivity for DecisionTreeClassifier:', recall6)
print('Testing Specificity for DecisionTreeClassifier:', (TN / (TN + FP)))
print('Testing Precision for DecisionTreeClassifier:', precision6)
print('Testing F1-measure for DecisionTreeClassifier:',(2*precision6*recall6)/(precision6+recall6))
```

```
import numpy as np

# Assuming new_data is your new dataset with features for prediction
new_data = np.array([54, 0, 3, 110, 214, 0, 0, 158, 0, 1.6, 2, 0, 3])

# Reshape to 2D array
new_data_2d = new_data.reshape(1, -1)  # or new_data[np.newaxis, :]

# Make predictions
new_prediction = model4.predict(new_data_2d)

# Display the predicted target values
print(" Heart Disease ",end='')
print(new_prediction)
```

```
Heart Disease ['Absence']
```