```
pip install --upgrade scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: numpy<2.0,>=1.17.3 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dis
```

Importing packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

FEATURES

age

sex

cp : chest pain type (4 values)

1.typical angina

2.atypical angina

3.non-anginal pain

4.asymptomatic

trestbps : resting blood pressure

chol : serum cholestoral in mg/dl

fbs :fasting blood sugar > 120 mg/dl

restecg : resting electrocardiographic results (values 0,1,2)

thalach : maximum heart rate achieved

exang : exercise induced angina

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 : 0 = normal; 1 = fixed defect; 2 = reversable defect

Reading the dataset

```
df=pd.read_csv('/content/archive (1).zip')
df.shape
```

```
(1025, 14)
```

Exploring the Data Initial Stage

> 1-What problem is to be solved?

> 2-What kind of data is present?

> 3-Does this data have missing values?

> 4-Are there any outliers, sporious vectors?

> 5-Can we add or remove some of the features?

```
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	tha
0	52	1	0	125	212	0	1	168	0	1.0	2	2	
1	53	1	0	140	203	1	0	155	1	3.1	0	0	
2	70	1	0	145	174	0	1	125	1	2.6	0	0	
3	61	1	0	148	203	0	1	161	0	0.0	2	1	
4	62	0	0	138	294	1	1	106	0	1.9	1	3	

```
df.tail()
```

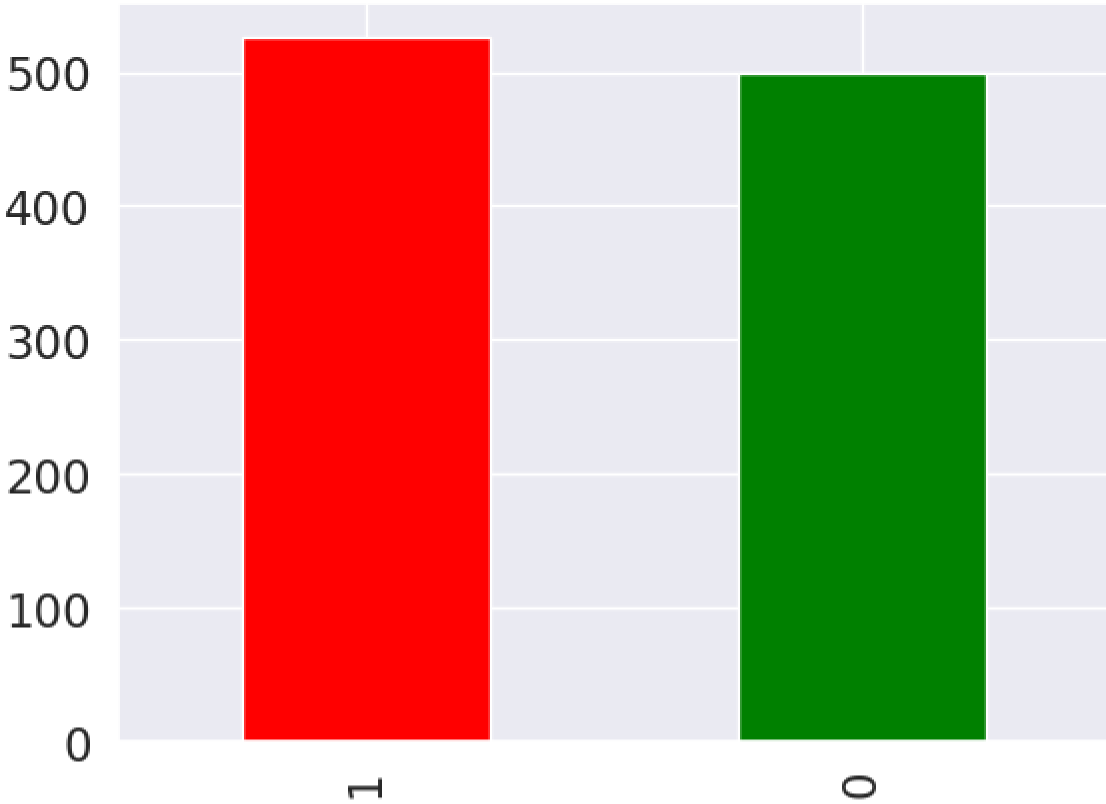
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
1020	59	1	1	140	221	0	1	164	1	0.0	2	0
1021	60	1	0	125	258	0	0	141	1	2.8	1	1
1022	47	1	0	110	275	0	0	118	1	1.0	1	1
1023	50	0	0	110	254	0	0	159	0	0.0	2	0
1024	54	1	0	120	188	0	1	113	0	1.4	1	1

```
df['target'].value_counts()
```

```
1    526
0    499
Name: target, dtype: int64
```

```
df['target'].value_counts().plot(kind='bar',color=['red','green'])
```

```
<Axes: >
```



Checking the missing values

```
df.isna().sum()
```

```
age         0
sex         0
cp          0
```

```
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Checking the outliers

```
import numpy as np
from scipy.stats import zscore

z_scores = zscore(df['thalach'])
z_scoresa = zscore(df['age'])
z_scorescp = zscore(df['cp'])
z_scoresfbs = zscore(df['cp'])
outliers_zscore = np.where(np.abs(z_scores) > 3)[0]
outliers_zscorea = np.where(np.abs(z_scoresa) > 3)[0]
outliers_zscorecp = np.where(np.abs(z_scorescp) > 3)[0]
outliers_zscorefbs = np.where(np.abs(z_scoresfbs) > 3)[0]
print("Outliers using Z-score for thalach:", outliers_zscore)
print("Outliers using Z-score for age:", outliers_zscorea)
print("Outliers using Z-score for cp:", outliers_zscorecp)
print("Outliers using Z-score for fbs:", outliers_zscorefbs)
```

```
Outliers using Z-score for thalach: [267 296 378 559]
Outliers using Z-score for age: []
Outliers using Z-score for cp: []
Outliers using Z-score for fbs: []
```

Checking Statistics

```
df.describe()
```

	age	sex	cp	trestbps	chol	fbs	
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	102
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	

```
df.sex.value_counts()
```

```
1    713
0    312
Name: sex, dtype: int64
```

```
len(df)
```

```
1025
```

```
print("Percent of males:",312/1025)
```

```
Percent of males: 0.304390243902439
```

```
print("Percent of females:",713/1025)
```

```
Percent of females: 0.6956097560975609
```

```
pd.crosstab(df.target,df.sex)
```

sex	0	1
target		
0	86	413
1	226	300

```
print("Percent of males with disease:",226/312)
```

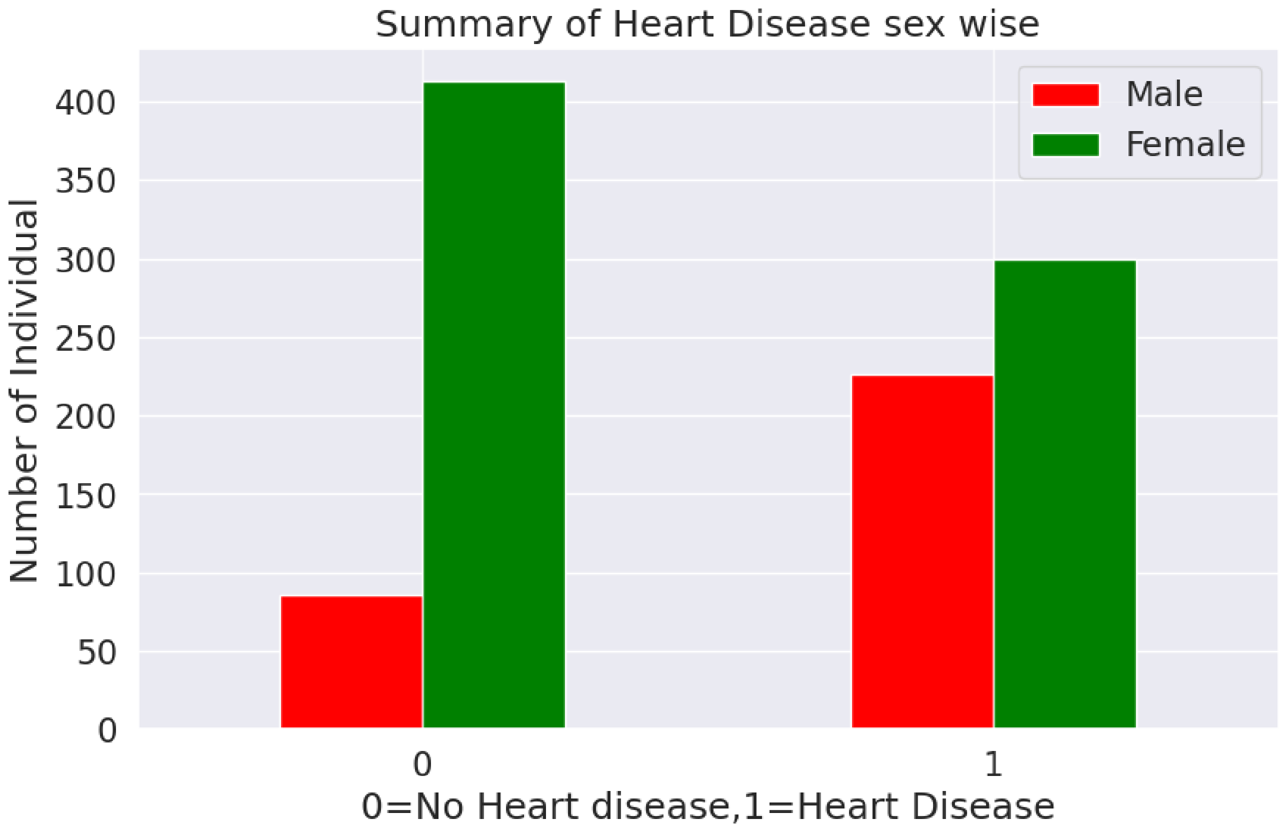
```
Percent of males with disease: 0.7243589743589743
```

```
print("Percent of females with disease:",300/713)
```

```
Percent of females with disease: 0.42075736325385693
```

```
pd.crosstab(df.target,df.sex).plot(kind='bar',figsize=(10,6),color=['red','green'])
plt.title('Summary of Heart Disease sex wise')
plt.xlabel('0=No Heart disease,1=Heart Disease')
plt.ylabel('Number of Individual')
plt.legend(['Male','Female'])
plt.xticks(rotation=0)
```

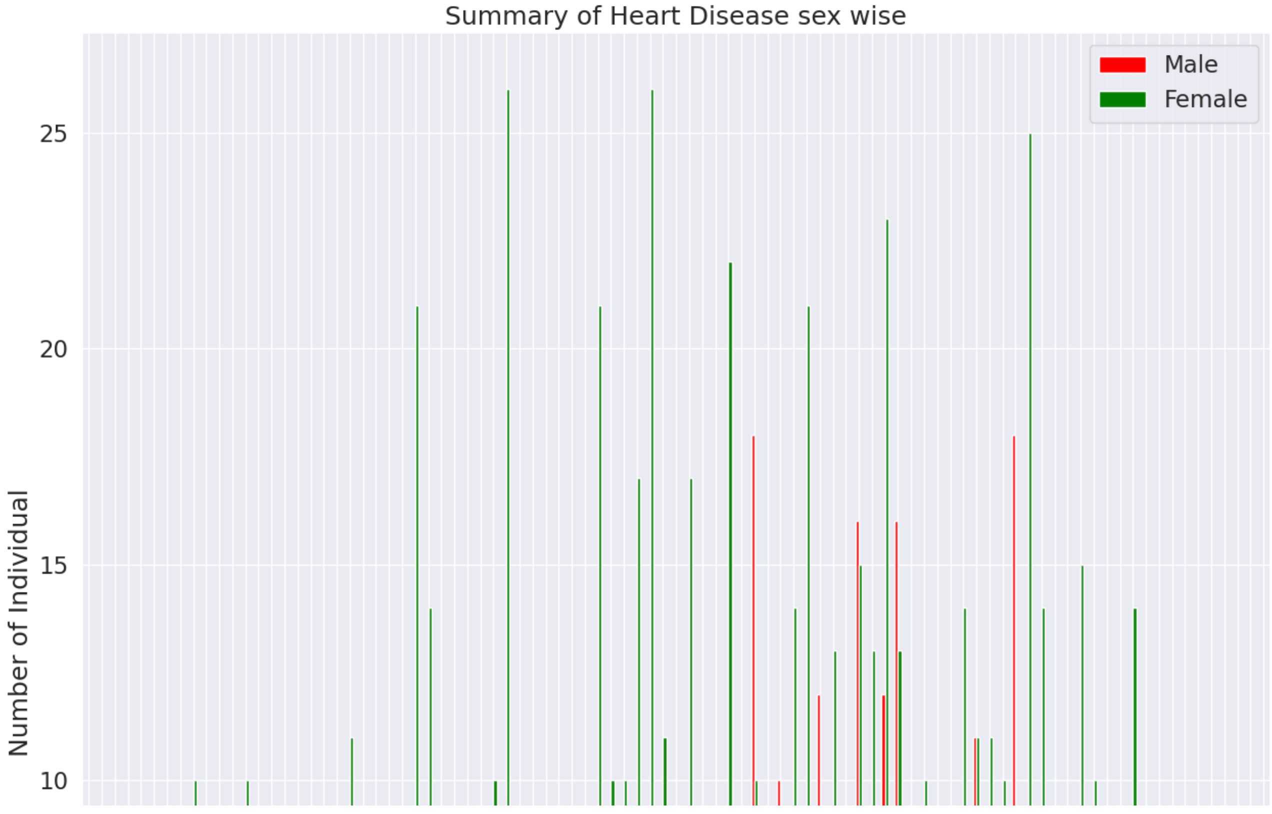
```
(array([0, 1]), [Text(0, 0, '0'), Text(1, 0, '1')])
```

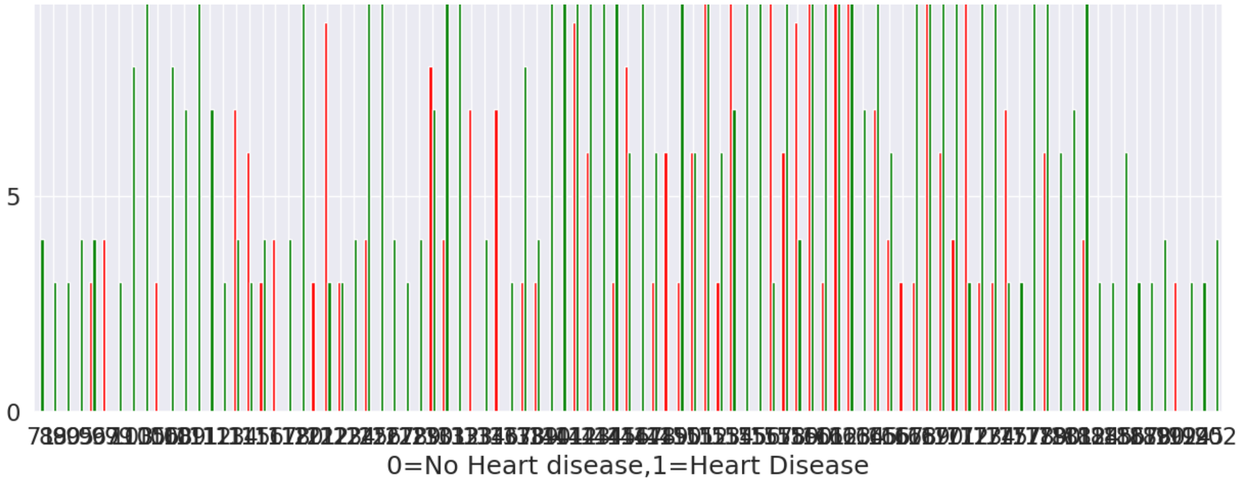



```
pd.crosstab(df.thalach,df.sex).plot(kind='bar',figsize=(15,15),color=['red','green'])
plt.title('Summary of Heart Disease sex wise')
plt.xlabel('0=No Heart disease,1=Heart Disease')
plt.ylabel('Number of Individual')
plt.legend(['Male','Female'])
plt.xticks(rotation=0)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
        68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
        85, 86, 87, 88, 89, 90]),
 [Text(0, 0, '71'),
  Text(1, 0, '88'),
  Text(2, 0, '90'),
  Text(3, 0, '95'),
  Text(4, 0, '96'),
  Text(5, 0, '97'),
  Text(6, 0, '99'),
  Text(7, 0, '103'),
  Text(8, 0, '105'),
  Text(9, 0, '106'),
  Text(10, 0, '108'),
  Text(11, 0, '109'),
  Text(12, 0, '111'),
  Text(13, 0, '112'),
  Text(14, 0, '113'),
  Text(15, 0, '114'),
  Text(16, 0, '115'),
  Text(17, 0, '116'),
  Text(18, 0, '117'),
  Text(19, 0, '118'),
  Text(20, 0, '120'),
  Text(21, 0, '121'),
  Text(22, 0, '122'),
  Text(23, 0, '123'),
  Text(24, 0, '124'),
  Text(25, 0, '125'),
  Text(26, 0, '126'),
  Text(27, 0, '127'),
  Text(28, 0, '128'),
  Text(29, 0, '129'),
  Text(30, 0, '130'),
  Text(31, 0, '131'),
  Text(32, 0, '132'),
  Text(33, 0, '133'),
  Text(34, 0, '134'),
  Text(35, 0, '136'),
  Text(36, 0, '137'),
  Text(37, 0, '138'),
  Text(38, 0, '139'),
  Text(39, 0, '140'),
  Text(40, 0, '141'),
  Text(41, 0, '142'),
  Text(42, 0, '143'),
  Text(43, 0, '144'),
  Text(44, 0, '145'),
  Text(45, 0, '146'),
  Text(46, 0, '147'),
  Text(47, 0, '148'),
  Text(48, 0, '149'),
  Text(49, 0, '150'),
  Text(50, 0, '151'),
  Text(51, 0, '152'),
  Text(52, 0, '153'),
  Text(53, 0, '154'),
```

```
Text(54, 0, '155'),
Text(55, 0, '156'),
Text(56, 0, '157'),
Text(57, 0, '158'),
Text(58, 0, '159'),
Text(59, 0, '160'),
Text(60, 0, '161'),
Text(61, 0, '162'),
Text(62, 0, '163'),
Text(63, 0, '164'),
Text(64, 0, '165'),
Text(65, 0, '166'),
Text(66, 0, '167'),
Text(67, 0, '168'),
Text(68, 0, '169'),
Text(69, 0, '170'),
Text(70, 0, '171'),
Text(71, 0, '172'),
Text(72, 0, '173'),
Text(73, 0, '174'),
Text(74, 0, '175'),
Text(75, 0, '177'),
Text(76, 0, '178'),
Text(77, 0, '179'),
Text(78, 0, '180'),
Text(79, 0, '181'),
Text(80, 0, '182'),
Text(81, 0, '184'),
Text(82, 0, '185'),
Text(83, 0, '186'),
Text(84, 0, '187'),
Text(85, 0, '188'),
Text(86, 0, '190'),
Text(87, 0, '192'),
Text(88, 0, '194'),
Text(89, 0, '195'),
Text(90, 0, '202')])
```



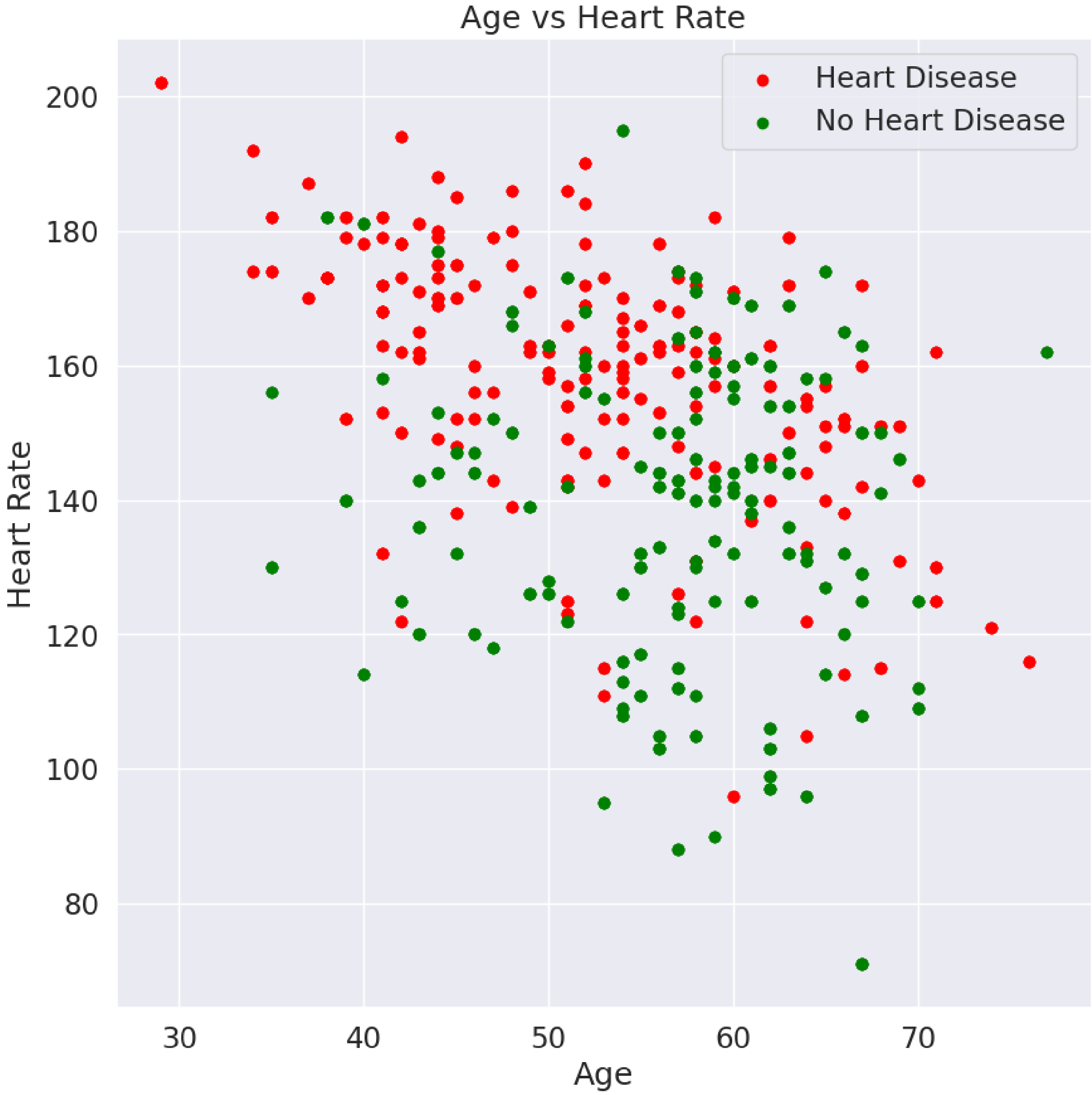



```
df.thalach.value_counts()
```

```
162    35
160    31
163    29
173    28
152    28
       ..
194     3
185     3
106     3
88      3
113     3
Name: thalach, Length: 91, dtype: int64
```

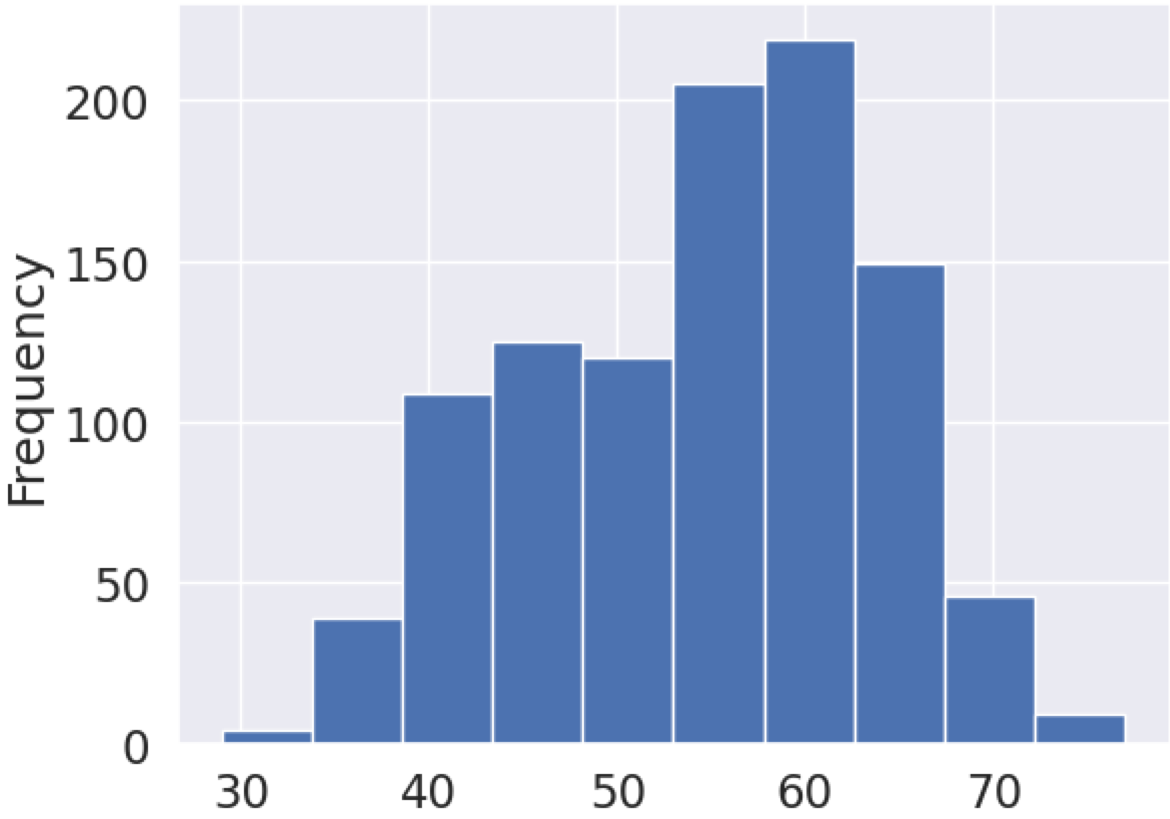
```
plt.figure(figsize=(10,10))
plt.scatter(df.age[df.target==1],df.thalach[df.target==1],c='red')
plt.scatter(df.age[df.target==0],df.thalach[df.target==0],c='green')
plt.title('Age vs Heart Rate')
plt.xlabel('Age')
plt.ylabel('Heart Rate')
plt.legend(['Heart Disease','No Heart Disease'])
```

```
<matplotlib.legend.Legend at 0x79d862c2ab30>
```




```
df.age.plot.hist()
```

```
<Axes: ylabel='Frequency'>
```




```
df.cp.value_counts()
```

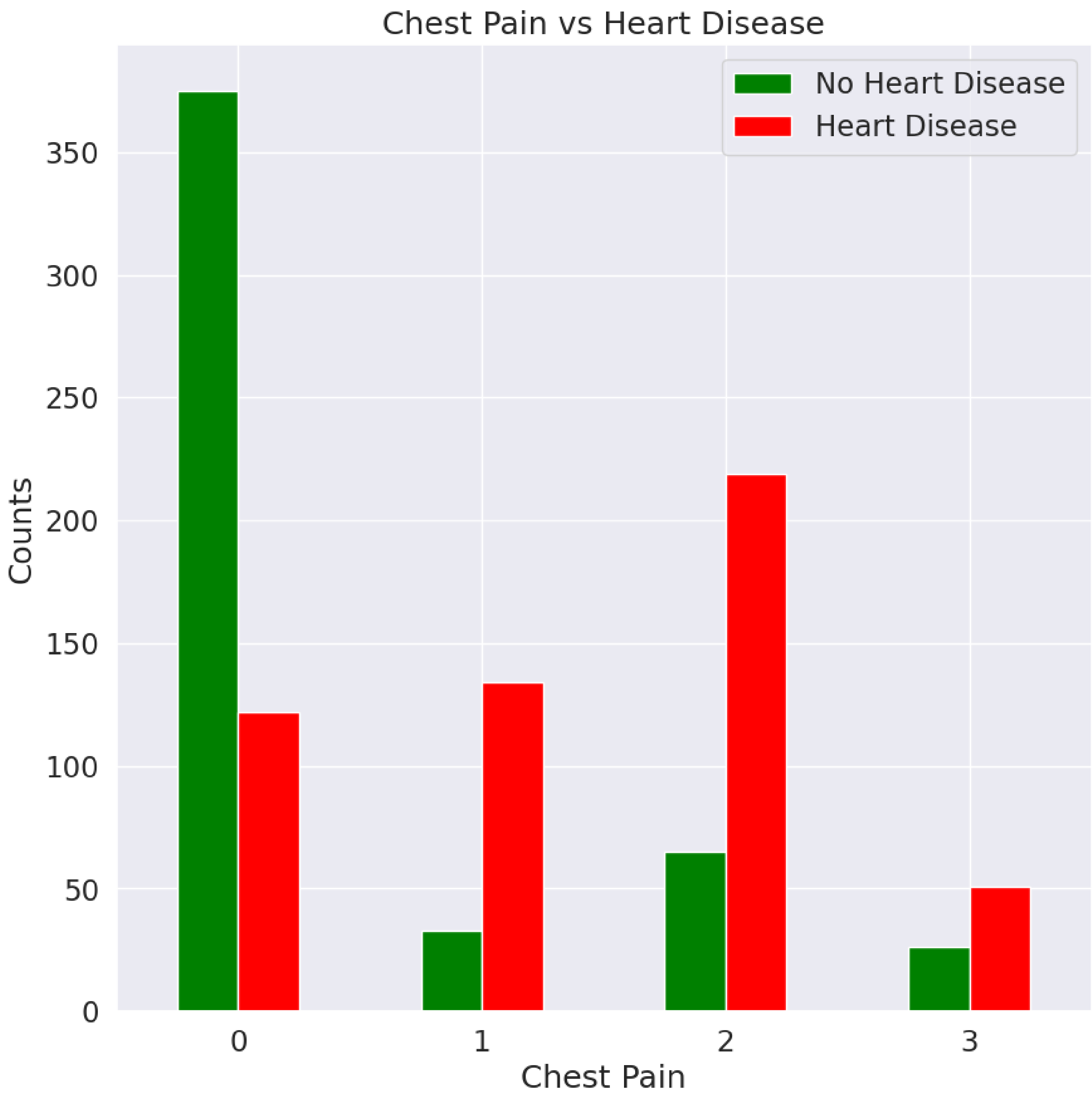
```
0    497
2    284
1    167
3     77
Name: cp, dtype: int64
```

```
pd.crosstab(df.cp,df.target)
```

target	0	1
cp		
0	375	122
1	33	134
2	65	219
3	26	51

```
pd.crosstab(df.cp,df.target).plot(kind='bar',figsize=(10,10),color=['green','red'])
plt.title('Chest Pain vs Heart Disease')
plt.xlabel('Chest Pain')
plt.ylabel('Counts')
plt.legend(['No Heart Disease','Heart Disease'])
plt.xticks(rotation=0)
```

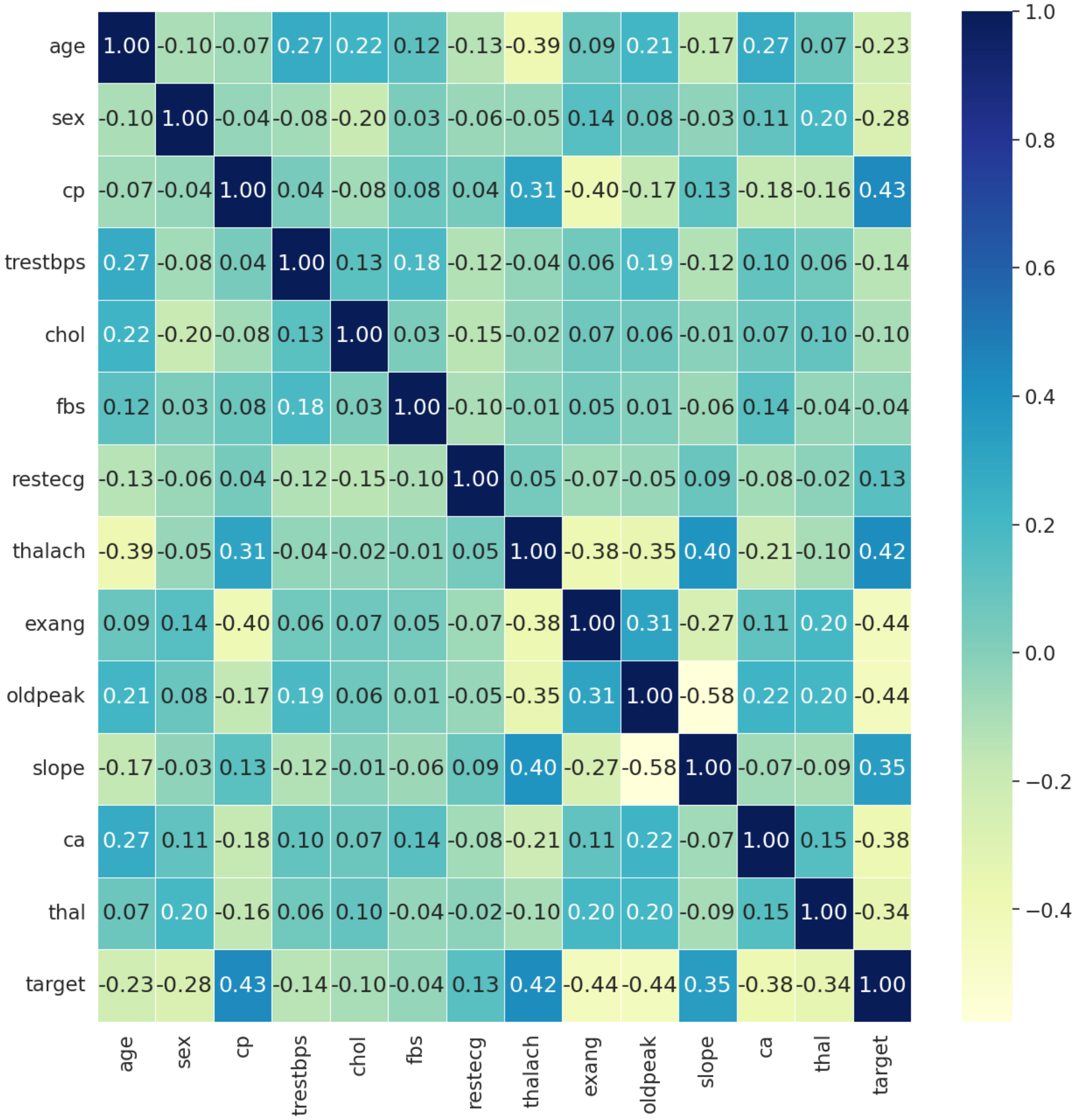
```
(array([0, 1, 2, 3]),
 [Text(0, 0, '0'), Text(1, 0, '1'), Text(2, 0, '2'), Text(3, 0, '3')])
```



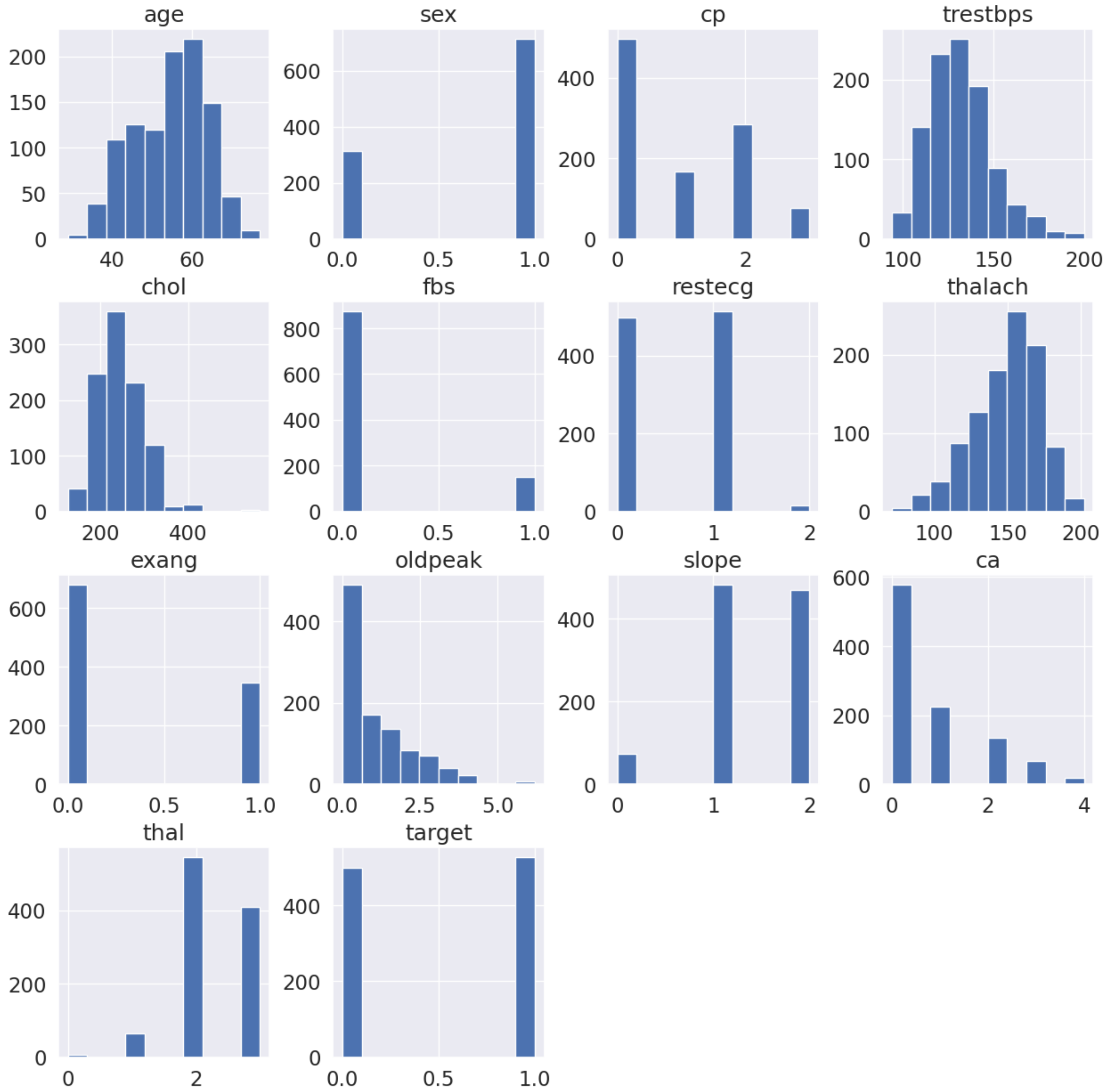

```
df.corr()
```

	age	sex	cp	trestbps	chol	fbs	restecg	tha
age	1.000000	-0.103240	-0.071966	0.271121	0.219823	0.121243	-0.132696	-0.39
sex	-0.103240	1.000000	-0.041119	-0.078974	-0.198258	0.027200	-0.055117	-0.04
cp	-0.071966	-0.041119	1.000000	0.038177	-0.081641	0.079294	0.043581	0.30
trestbps	0.271121	-0.078974	0.038177	1.000000	0.127977	0.181767	-0.123794	-0.03
chol	0.219823	-0.198258	-0.081641	0.127977	1.000000	0.026917	-0.147410	-0.02
fbs	0.121243	0.027200	0.079294	0.181767	0.026917	1.000000	-0.104051	-0.00
restecg	-0.132696	-0.055117	0.043581	-0.123794	-0.147410	-0.104051	1.000000	0.04
thalach	-0.390227	-0.049365	0.306839	-0.039264	-0.021772	-0.008866	0.048411	1.00
exang	0.088163	0.139157	-0.401513	0.061197	0.067382	0.049261	-0.065606	-0.38
oldpeak	0.208137	0.084687	-0.174733	0.187434	0.064880	0.010859	-0.050114	-0.34
slope	-0.169105	-0.026666	0.131633	-0.120445	-0.014248	-0.061902	0.086086	0.39
ca	0.271551	0.111729	-0.176206	0.104554	0.074259	0.137156	-0.078072	-0.20
thal	0.072297	0.198424	-0.163341	0.059276	0.100244	-0.042177	-0.020504	-0.09
target	-0.229324	-0.279501	0.434854	-0.138772	-0.099966	-0.041164	0.134468	0.42

```
home,room=plt.subplots(figsize=(15,15))
room=sns.heatmap(df.corr(),annot=True,linewidths=0.5,fmt='0.2f',cmap='YlGnBu')
```




```
df.hist(figsize=(15,15))
plt.savefig('featuresplot')
```




```
x=df.drop('target',axis=1)
y=df['target']
```

```
x
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
0	52	1	0	125	212	0	1	168	0	1.0	2	2
1	53	1	0	140	203	1	0	155	1	3.1	0	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1
4	62	0	0	138	294	1	1	106	0	1.9	1	3
...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0
1021	60	1	0	125	258	0	0	141	1	2.8	1	1
1022	47	1	0	110	275	0	0	118	1	1.0	1	1
1023	50	0	0	110	254	0	0	159	0	0.0	2	0
1024	54	1	0	120	188	0	1	113	0	1.4	1	1

```
y
```

```
0       0
1       0
2       0
3       0
4       0
       ..
1020    1
1021    0
1022    0
1023    1
1024    0
Name: target, Length: 1025, dtype: int64
```

1.LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

```
model=LogisticRegression(max_iter=1000)
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
```

```
model.fit(x_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

```
model.score(x_train,y_train)
```

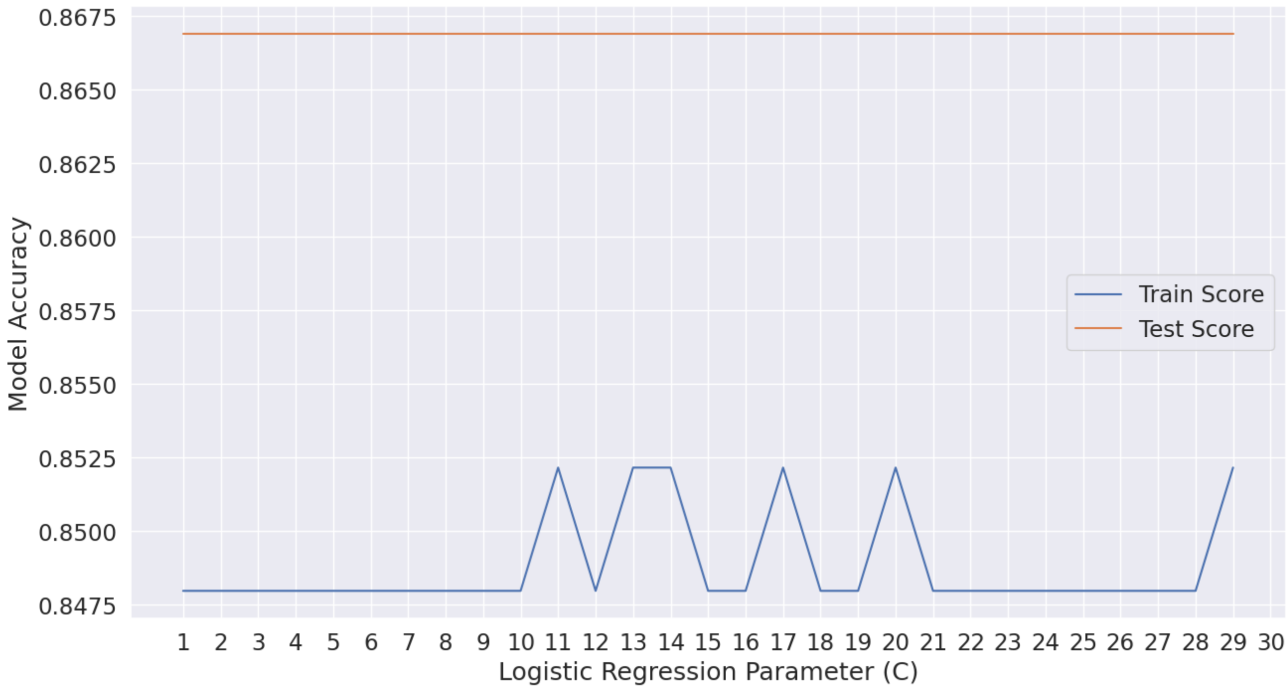
```
0.8479776847977685
```

```
model.score(x_test,y_test)
```

```
0.8668831168831169
```

```
train_score = []
test_score = []
logistic_params = range(1, 30)
for i in logistic_params:
    LR = LogisticRegression(max_iter=5000, C=i)
    LR.fit(x_train, y_train)
    train_score.append(LR.score(x_train, y_train))
    test_score.append(LR.score(x_test, y_test))
```

```
plt.figure(figsize=(15, 8))
plt.plot(logistic_params, train_score, label='Train Score')
plt.plot(logistic_params, test_score, label='Test Score')
plt.xticks(np.arange(1, 31, 1))
plt.xlabel('Logistic Regression Parameter (C)')
plt.ylabel('Model Accuracy')
plt.legend()
plt.show()
print(f'Max Logistic Regression Score: {max(test_score)*100:0.2f}%')
```




```
Max Logistic Regression Score: 86.69%
```

```
train_sizes, train_scores, test_scores = learning_curve(
    LogisticRegression(max_iter=5000, C=logistic_params[min_test_error_index]),
    x_train, y_train, cv=5, scoring='accuracy', train_sizes=np.linspace(0.1, 1.0, 10)
)

plt.figure(figsize=(15, 8))
plt.plot(train_sizes, 1 - np.mean(train_scores, axis=1), label='Train Error Rate')
plt.plot(train_sizes, 1 - np.mean(test_scores, axis=1), label='Test Error Rate')
plt.xlabel('Training Set Size')
plt.ylabel('Error Rate')
plt.legend()
plt.show()
```

2.SUPPORT VECTOR MACHINE

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

```
model=SVC()
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
```

```
model.fit(x_train,y_train)
```

▾ SVC

SVC()

```
model.score(x_train,y_train)
```

0.691771269177127

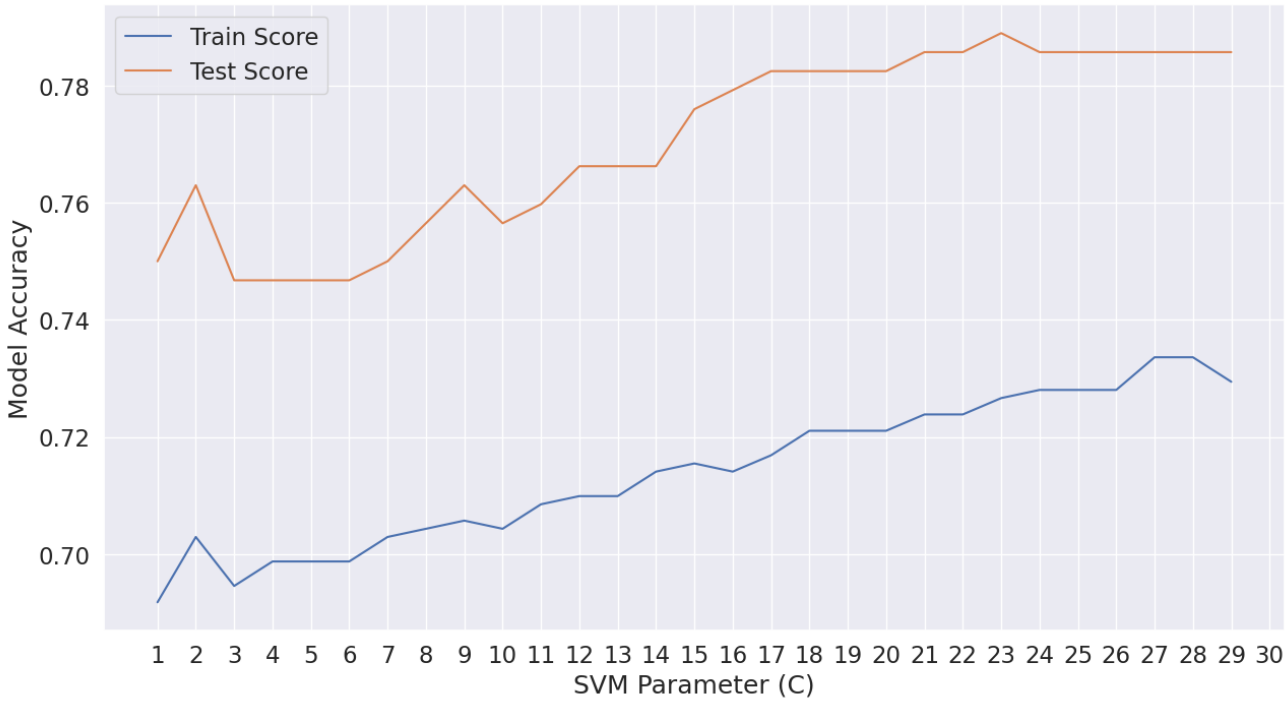
```
model.score(x_test,y_test)
```

0.75

```
train_score = []
test_score = []
svm_params = range(1, 30)
for i in svm_params:
    SVM = SVC(C=i)
    SVM.fit(x_train, y_train)
    train_score.append(SVM.score(x_train, y_train))
    test_score.append(SVM.score(x_test, y_test))
```

```
plt.figure(figsize=(15, 8))
plt.plot(svm_params, train_score, label='Train Score')
plt.plot(svm_params, test_score, label='Test Score')
plt.xticks(np.arange(1, 31, 1))
plt.xlabel('SVM Parameter (C)')
plt.ylabel('Model Accuracy')
plt.legend()
plt.show()

print(f'Max SVM Score: {max(test_score)*100:0.2f}%')
```




```
Max SVM Score: 78.90%
```

```
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
svm_classifier = SVC(kernel='linear', C=26, random_state=42)

train_sizes, train_scores, test_scores = learning_curve(
    svm_classifier,
    x_train_scaled, y_train, cv=5, scoring='accuracy', train_sizes=np.linspace(0.1, 1.0, 10)
)

plt.figure(figsize=(15, 8))
plt.plot(train_sizes, 1 - np.mean(train_scores, axis=1), label='Train Error Rate')
plt.plot(train_sizes, 1 - np.mean(test_scores, axis=1), label='Test Error Rate')
plt.xlabel('Training Set Size')
plt.ylabel('Error Rate')
plt.legend()
plt.show()
```

3.DECISION TREE CLASSIFIER

```
from sklearn.tree import DecisionTreeClassifier
```

```
model=DecisionTreeClassifier()
```

```
model.fit(x_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
model.score(x_train,y_train)
```

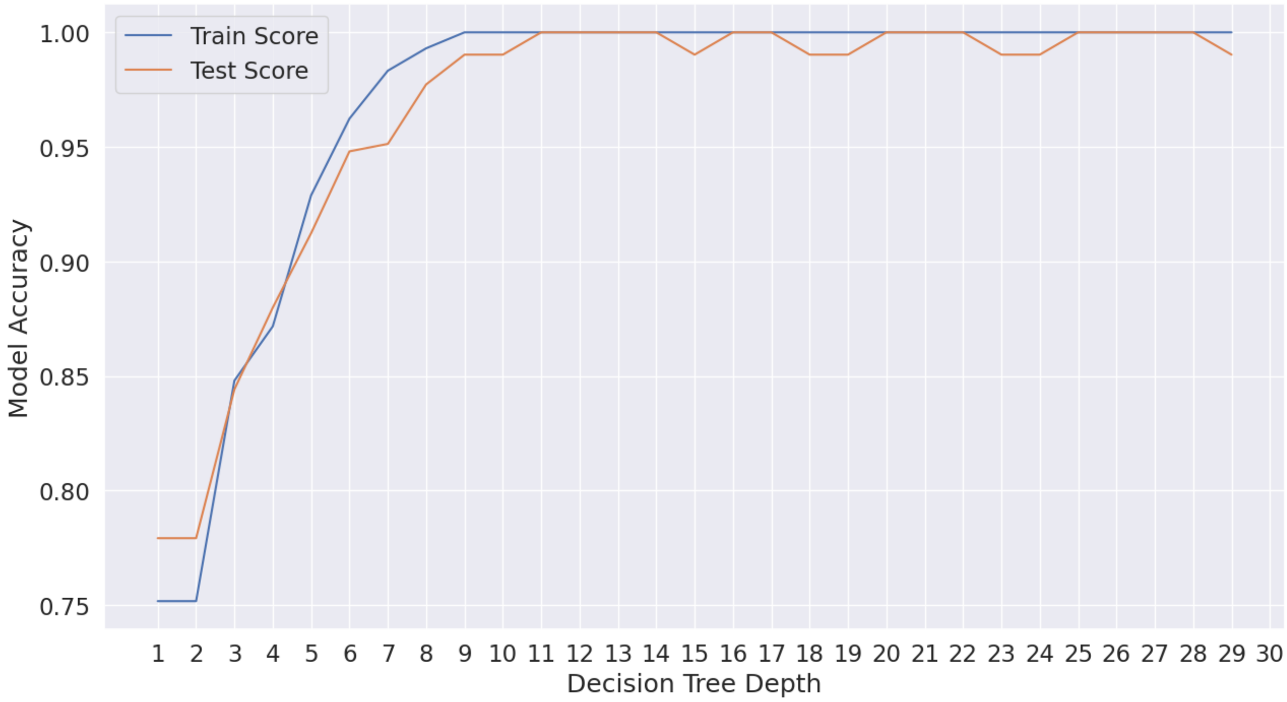
```
1.0
```

```
model.score(x_test,y_test)
```

```
1.0
```

```
train_score = []
test_score = []
tree_depths = range(1, 30)
for depth in tree_depths:
    tree_clf = DecisionTreeClassifier(max_depth=depth)
    tree_clf.fit(x_train, y_train)
    train_score.append(tree_clf.score(x_train, y_train))
    test_score.append(tree_clf.score(x_test, y_test))
```

```
plt.figure(figsize=(15, 8))
plt.plot(tree_depths, train_score, label='Train Score')
plt.plot(tree_depths, test_score, label='Test Score')
plt.xticks(np.arange(1, 31, 1))
plt.xlabel('Decision Tree Depth')
plt.ylabel('Model Accuracy')
plt.legend()
plt.show()
print(f'Max Decision Tree Score: {max(test_score)*100:0.2f}%')
```



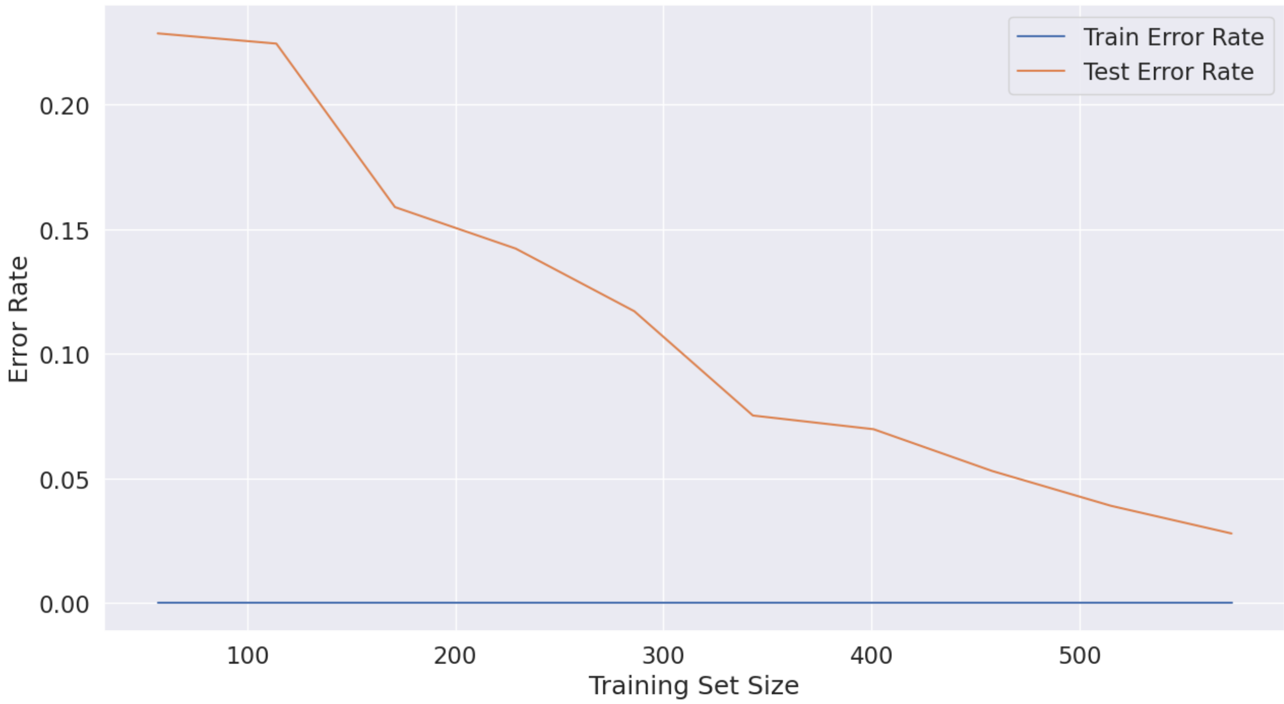

```
Max Decision Tree Score: 100.00%
```

```
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

dt_classifier = DecisionTreeClassifier(random_state=42)

train_sizes, train_scores, test_scores = learning_curve(
    dt_classifier,
    x_train, y_train, cv=5, scoring='accuracy', train_sizes=np.linspace(0.1, 1.0, 10)
)

plt.figure(figsize=(15, 8))
plt.plot(train_sizes, 1 - np.mean(train_scores, axis=1), label='Train Error Rate')
plt.plot(train_sizes, 1 - np.mean(test_scores, axis=1), label='Test Error Rate')
plt.xlabel('Training Set Size')
plt.ylabel('Error Rate')
plt.legend()
plt.show()
```



4.RANDOM FOREST REGRESSOR

```
from sklearn.ensemble import RandomForestRegressor
```

```
model=RandomForestRegressor(n_estimators=30,random_state=0)
```

```
model.fit(x_train,y_train)
```

```
RandomForestRegressor
RandomForestRegressor(n_estimators=30, random_state=0)
```

```
model.score(x_train,y_train)
```

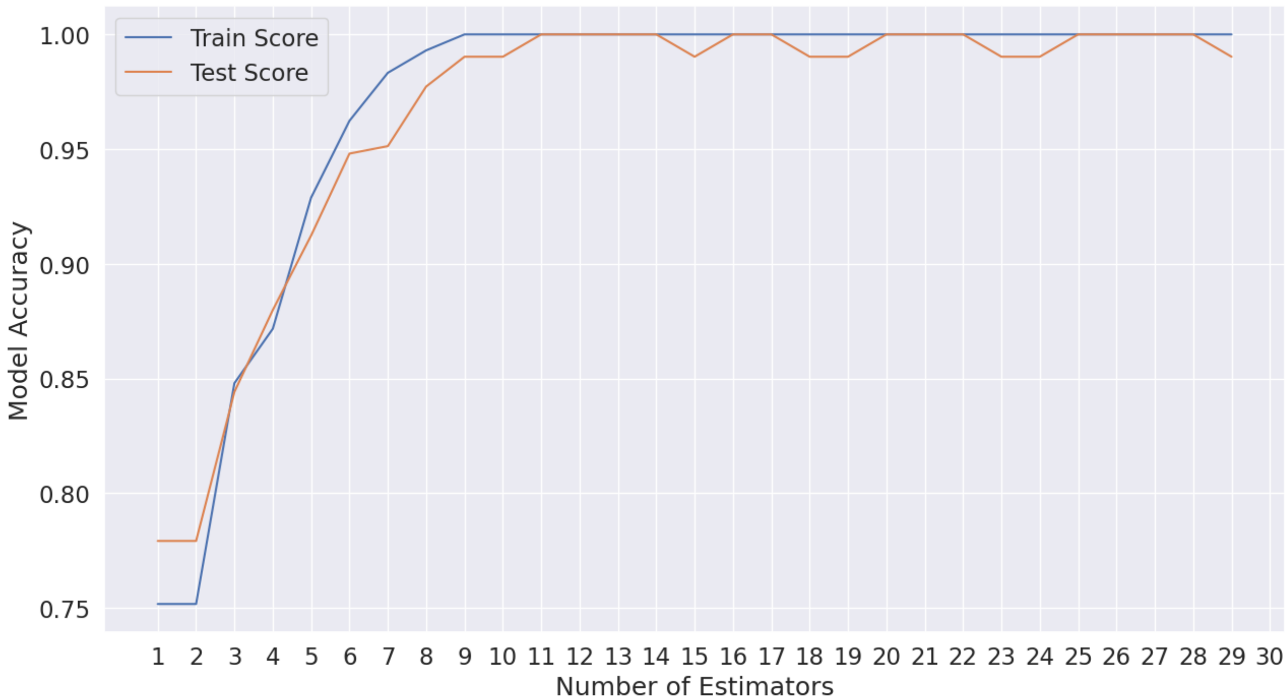
```
0.9898945801102966
```

```
model.score(x_test,y_test)
```

```
0.9593706132618762
```

```
train_score = []
test_score = []
n_estimators_range = range(1, 30)
for n_estimators in n_estimators_range:
    rf_regressor = RandomForestRegressor(n_estimators=n_estimators)
    rf_regressor.fit(x_train, y_train)
    train_score.append(rf_regressor.score(x_train, y_train))
    test_score.append(rf_regressor.score(x_test, y_test))


plt.figure(figsize=(15, 8))
plt.plot(n_estimators_range, train_score, label='Train Score')
plt.plot(n_estimators_range, test_score, label='Test Score')
plt.xticks(np.arange(1, 31, 1))
plt.xlabel('Number of Estimators')
plt.ylabel('Model Accuracy')
plt.legend()
plt.show()
print(f'Max Random Forest Regressor Score: {max(test_score)*100:0.2f}%')
```




```
Max Random Forest Regressor Score: 100.00%
```

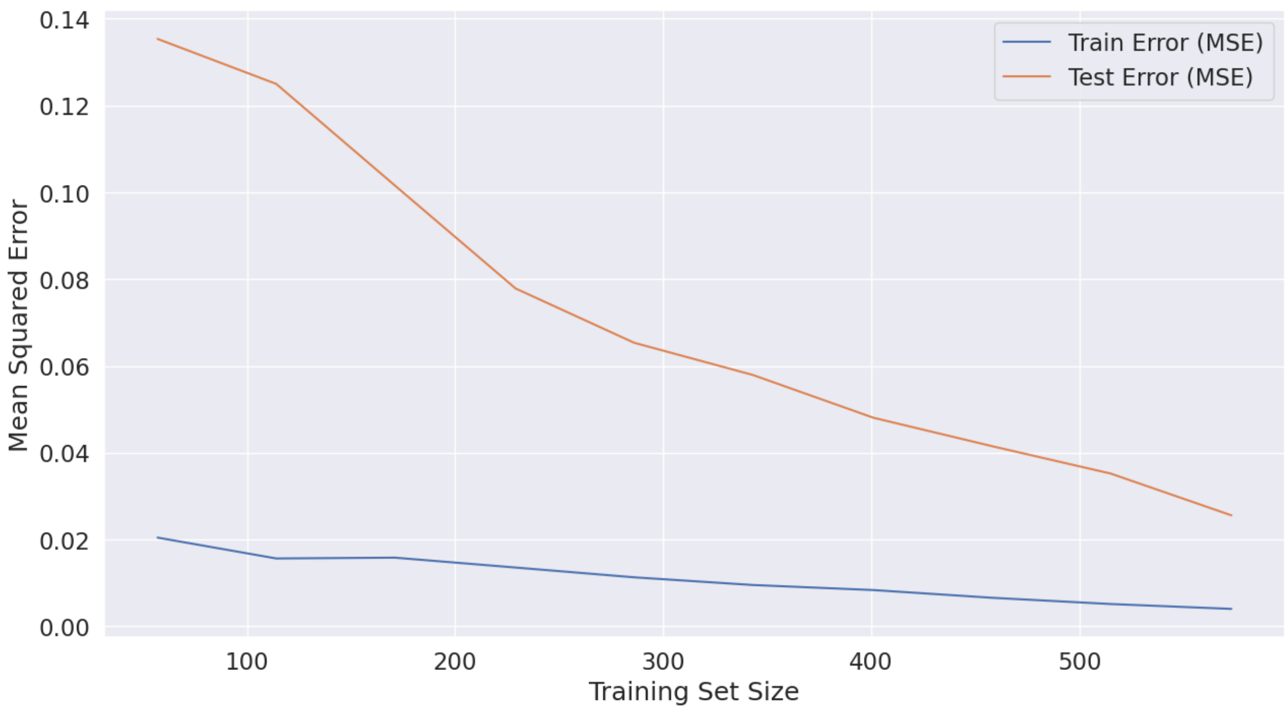
```
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

rf_regressor = RandomForestRegressor(random_state=42)

train_sizes, train_scores, test_scores = learning_curve(
    rf_regressor,
    x_train, y_train, cv=5, scoring='neg_mean_squared_error', train_sizes=np.linspace(0.1, 1.0, 10)
)

train_errors = -train_scores
test_errors = -test_scores

plt.figure(figsize=(15, 8))
plt.plot(train_sizes, np.mean(train_errors, axis=1), label='Train Error (MSE)')
plt.plot(train_sizes, np.mean(test_errors, axis=1), label='Test Error (MSE)')
plt.xlabel('Training Set Size')
plt.ylabel('Mean Squared Error')
plt.legend()
plt.show()
```



5.K NEAREST NEIGHBORS

```
from sklearn.neighbors import KNeighborsClassifier
```

```
model=KNeighborsClassifier(n_neighbors=5)
```

```
model.fit(x_train,y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
model.score(x_train,y_train)
```

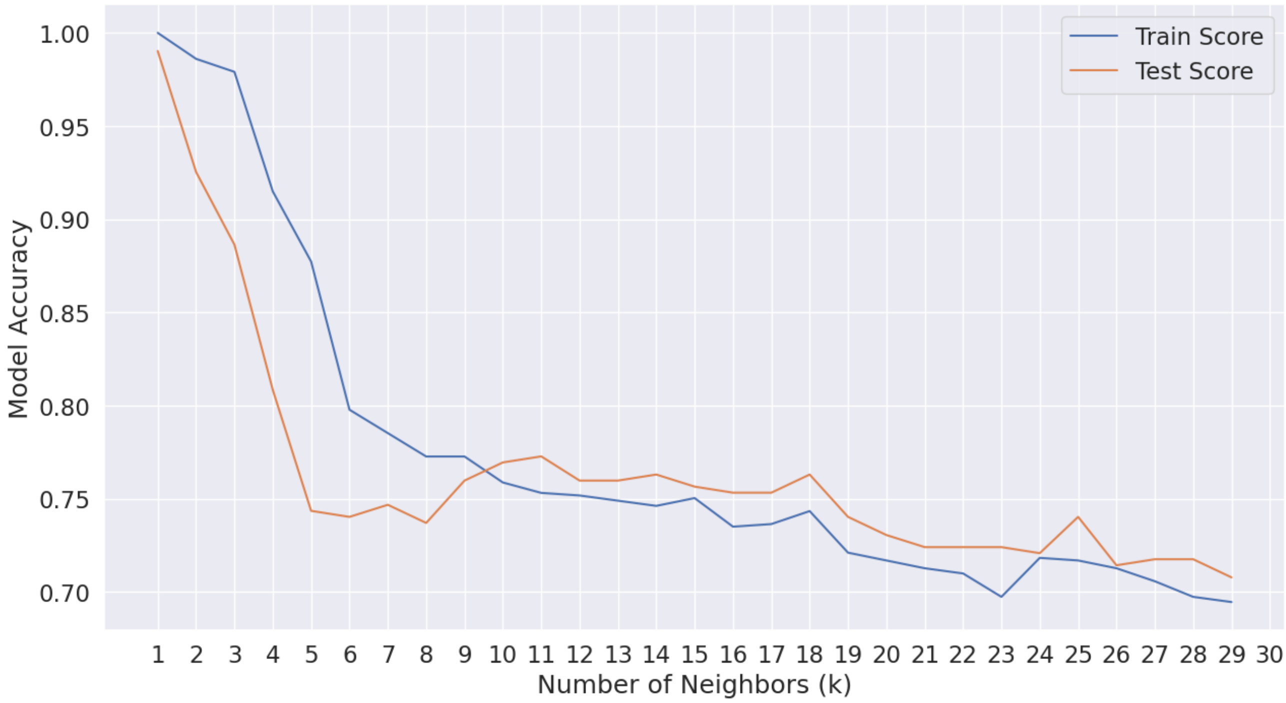
```
0.8772663877266388
```

```
model.score(x_test,y_test)
```

```
0.7435064935064936
```

```
train_score = []
test_score = []
k_values = range(1, 30)
for k in k_values:
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(x_train, y_train)
    train_score.append(knn_classifier.score(x_train, y_train))
    test_score.append(knn_classifier.score(x_test, y_test))
```

```
plt.figure(figsize=(15, 8))
plt.plot(k_values, train_score, label='Train Score')
plt.plot(k_values, test_score, label='Test Score')
plt.xticks(np.arange(1, 31, 1))
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Model Accuracy')
plt.legend()
plt.show()
print(f'Max KNN Score: {max(test_score)*100:0.2f}%')
```



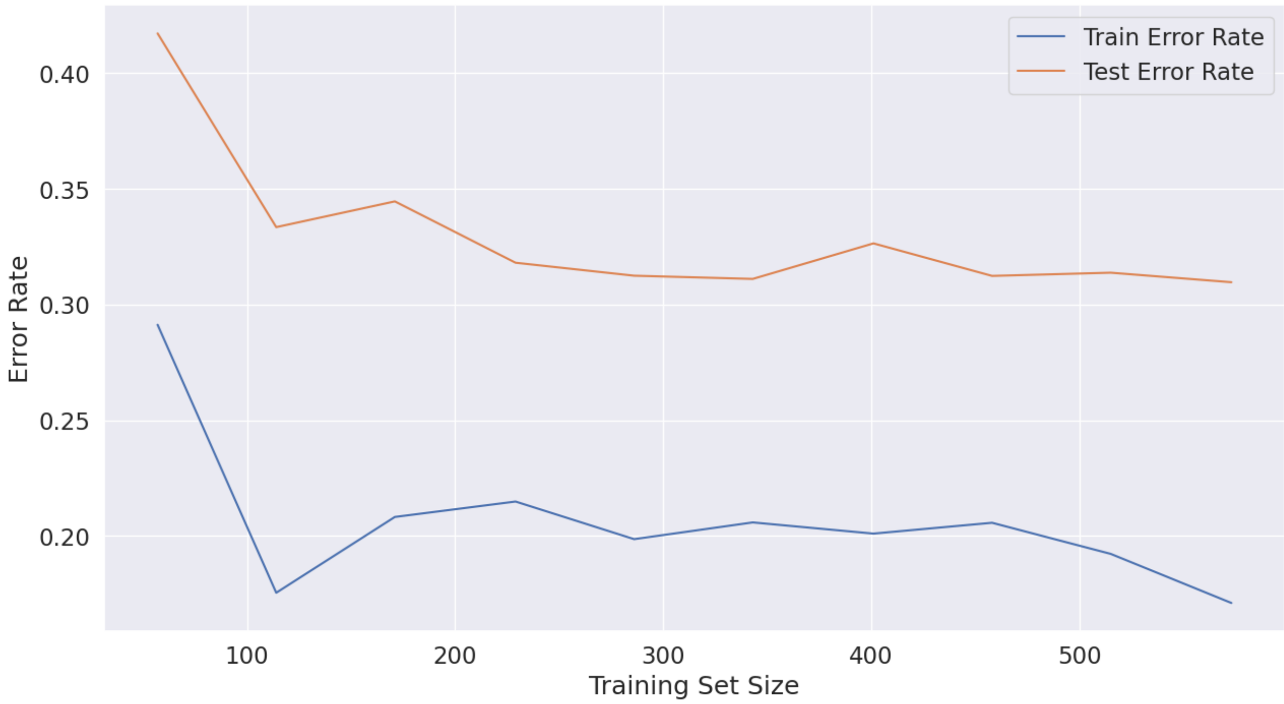

```
Max KNN Score: 99.03%
```

```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve


knn_classifier = KNeighborsClassifier()


train_sizes, train_scores, test_scores = learning_curve(
    knn_classifier,
    x_train, y_train, cv=5, scoring='accuracy', train_sizes=np.linspace(0.1, 1.0, 10)
)

plt.figure(figsize=(15, 8))
plt.plot(train_sizes, 1 - np.mean(train_scores, axis=1), label='Train Error Rate')
plt.plot(train_sizes, 1 - np.mean(test_scores, axis=1), label='Test Error Rate')
plt.xlabel('Training Set Size')
plt.ylabel('Error Rate')
plt.legend()
plt.show()
```



6.NAIVE BAYES

```
from sklearn.naive_bayes import GaussianNB
```

```
model=GaussianNB()
```

```
model.fit(x_train,y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
model.score(x_train,y_train)
```

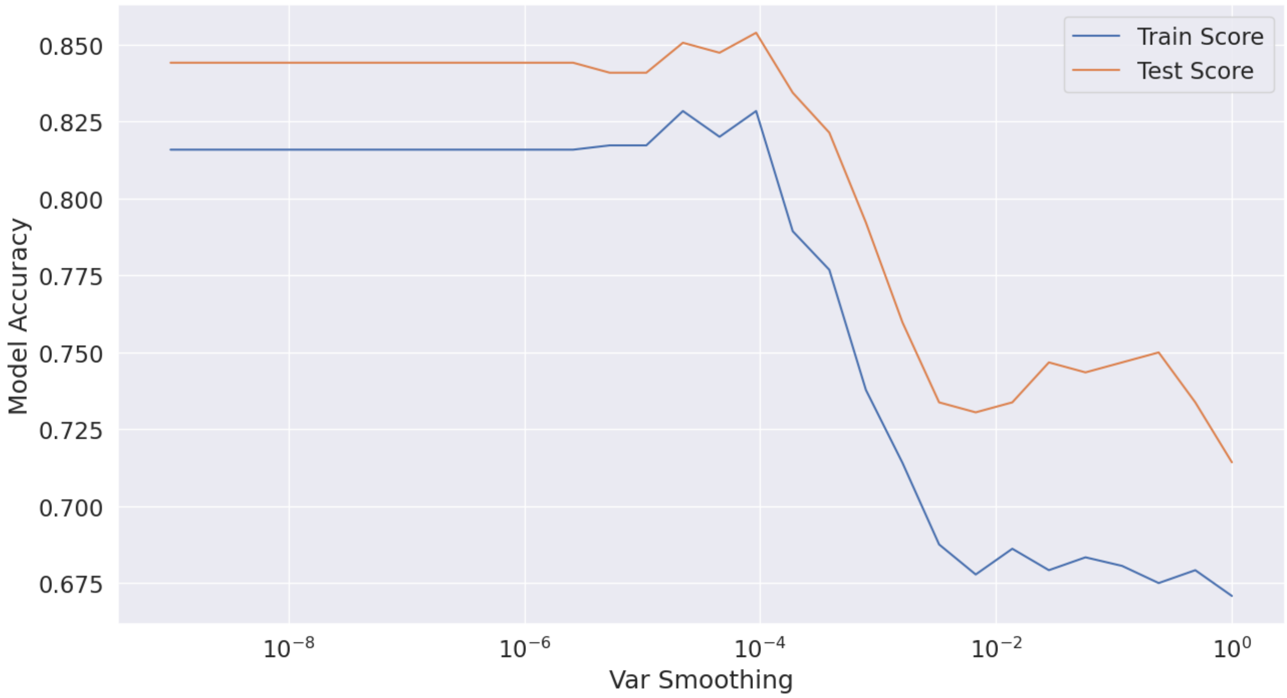
```
0.8158995815899581
```

```
model.score(x_test,y_test)
```

```
0.8441558441558441
```

```
train_score = []
test_score = []
var_smoothing_values = np.logspace(-9, 0, num=30)
for var_smoothing in var_smoothing_values:
    gnb = GaussianNB(var_smoothing=var_smoothing)
    gnb.fit(x_train, y_train)
    train_score.append(gnb.score(x_train, y_train))
    test_score.append(gnb.score(x_test, y_test))


plt.figure(figsize=(15, 8))
plt.plot(var_smoothing_values, train_score, label='Train Score')
plt.plot(var_smoothing_values, test_score, label='Test Score')
plt.xscale('log')
plt.xlabel('Var Smoothing')
plt.ylabel('Model Accuracy')
plt.legend()
plt.show()
max_test_score_index = np.argmax(test_score)
print(f'Max Naive Bayes Score: {test_score[max_test_score_index]*100:0.2f}%')
```



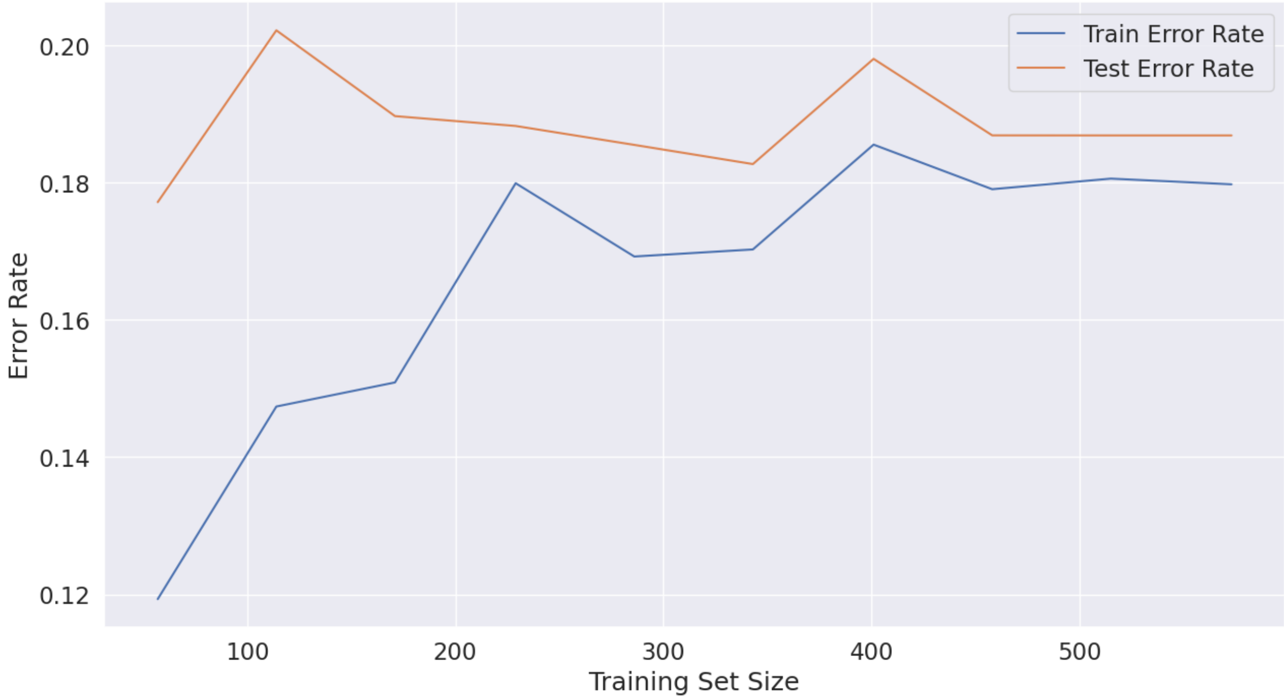

```
Max Naive Bayes Score: 85.39%
```

```
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

gnb_classifier = GaussianNB()

train_sizes, train_scores, test_scores = learning_curve(
    gnb_classifier,
    x_train, y_train, cv=5, scoring='accuracy', train_sizes=np.linspace(0.1, 1.0, 10)
)

plt.figure(figsize=(15, 8))
plt.plot(train_sizes, 1 - np.mean(train_scores, axis=1), label='Train Error Rate')
plt.plot(train_sizes, 1 - np.mean(test_scores, axis=1), label='Test Error Rate')
plt.xlabel('Training Set Size')
plt.ylabel('Error Rate')
plt.legend()
plt.show()
```



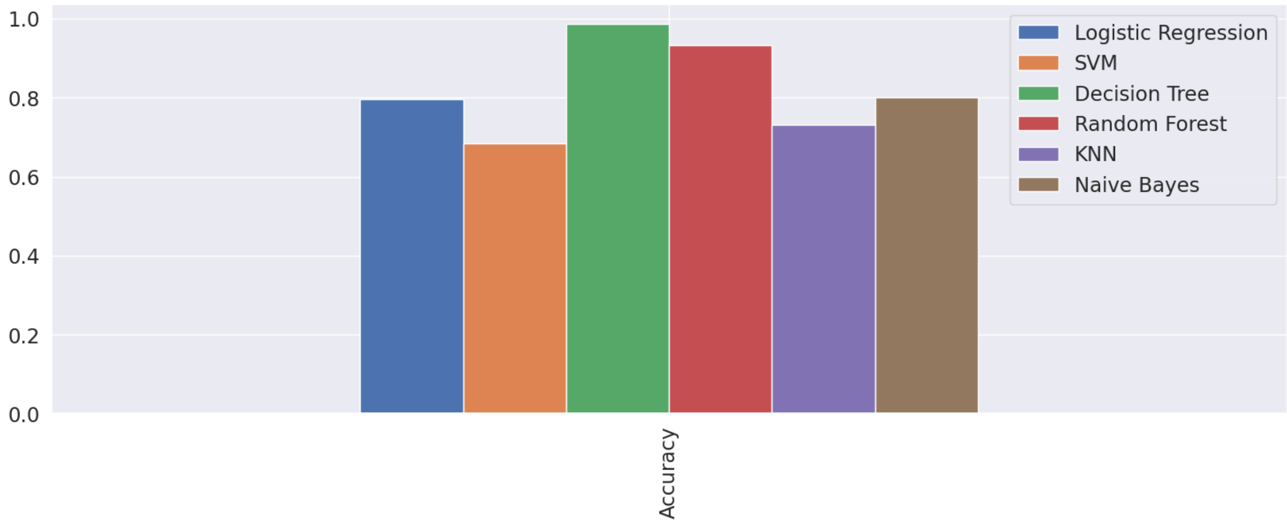

```
models={'Logistic Regression':LogisticRegression(max_iter=1000),'SVM':SVC(),'Decision Tree':DecisionTre
def fit_and_score(models,x_train,x_test,y_train,y_test):
  np.random.seed(7)
  model_score = {}
  for name,model in models.items():
    model.fit(x_train,y_train)
    model_score[name]=model.score(x_test,y_test)
  return model_score
```

```
model_score=fit_and_score(models=models,x_train=x_train,x_test=x_test,y_train=y_train,y_test=y_test)
model_score
```

```
{'Logistic Regression': 0.8668831168831169,
 'SVM': 0.75,
 'Decision Tree': 0.9902597402597403,
 'Random Forest': 0.9609758070657922,
```

```
        'KNN': 0.7435064935064936,
        'Naive Bayes': 0.8441558441558441}
```

```
model_compare=pd.DataFrame(model_score,index=['Accuracy'])
model_compare.plot.bar(figsize=(18,6));
```




```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score

dt_classifier = DecisionTreeClassifier(random_state=42, max_depth=None)

cv_scores = cross_val_score(dt_classifier, x, y, cv=5)
```