

```
[43]: import pandas as pd
import numpy as np
```

```
[44]: from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report
from sklearn.model_selection import GridSearchCV
import seaborn as sns
import os
import matplotlib.pyplot as plt
```

```
[3]: df = pd.read_csv("heart.csv")
df.head()
```

```
[3]:
```

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

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

<class 'pandas.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  --
 0   age           1025 non-null   int64   
 1   sex           1025 non-null   int64   
 2   cp            1025 non-null   int64   
 3   trestbps      1025 non-null   int64   
 4   chol          1025 non-null   int64   
 5   fbs           1025 non-null   int64   
 6   restecg       1025 non-null   int64   
 7   thalach       1025 non-null   int64   
 8   exang         1025 non-null   int64   
 9   oldpeak       1025 non-null   float64  
10  slope         1025 non-null   int64   
11  ca            1025 non-null   int64   
12  thal          1025 non-null   int64   
13  target        1025 non-null   int64   
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

```
[5]: df.isnull().sum()
```

```
[5]: 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
```

```
[6]: df.dtypes
```

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

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

```
[7]:
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	
mean	54.434146	0.695610	0.942439	131.611707	246.000000	0.149268	0.529756	149.114146	0.336585	1.071512	1.385366	0.754146	2.
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.617755	1.030798	0.
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.
25%	48.000000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000	0.000000	2.
50%	56.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000	0.000000	2.
75%	61.000000	1.000000	2.000000	140.000000	275.000000	0.000000	1.000000	166.000000	1.000000	1.800000	2.000000	1.000000	3.
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.

```
[8]: from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for column in df.select_dtypes(include=['object']).columns:
```

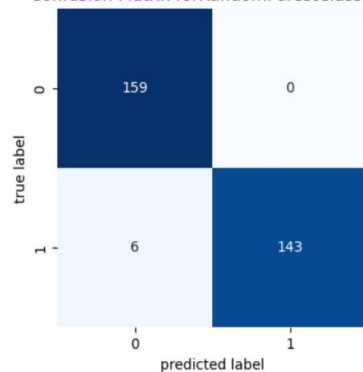
```
df[column]=le.fit_transform(df[column])
label_encoders[column] = le

x = df.drop(['target'],axis=1)
y=df['target']
x_train,x_test,y_train,y_test = train_test_split (x,y,random_state=42,test_size=0.30)
```

```
[9]: def evaluate_model(RandomForest,y_true,y_pred):
    accuracy = accuracy_score(y_true,y_pred)
    precision= precision_score(y_true,y_pred,average='weighted')
    recall=recall_score(y_true,y_pred,average='weighted')
    f1=f1_score(y_true,y_pred,average='weighted')
    cm= confusion_matrix(y_true,y_pred)
    report = classification_report(y_true,y_pred)
    metrics= {
        'Model Name': RandomForest,
        'Accuracy': accuracy,
        'Precision':precision,
        'Recall':recall,
        'F1 Score':f1,
        'Classification Report':report
    }
    plt.figure(figsize=(4,4))
    sns.heatmap(cm, annot=True, fmt='d',cmap='Blues',cbar=False,xticklabels=np.unique(y_true),
    yticklabels=np.unique(y_true))
    plt.title(f'Confusion Matrix for{RandomForest}')
    plt.xlabel('predicted label')
    plt.ylabel('true label')
    plt.show()
    return metrics
```

```
[12]: rf_classifier = RandomForestClassifier(random_state=42,n_estimators=100)
rf_classifier.fit(x_train,y_train)
y_pred= rf_classifier.predict(x_test)
evaluation_results= evaluate_model('RandomForestClassifier',y_test,y_pred)
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"(key):(value:4f)" if isinstance(value, float) else f"(key): \n{value}")
```

Confusion Matrix forRandomForestClassifier



```
Model Name:
RandomForestClassifier
Accuracy:0.980519
Precision:0.981228
Recall:0.980519
F1 Score:0.980500
```

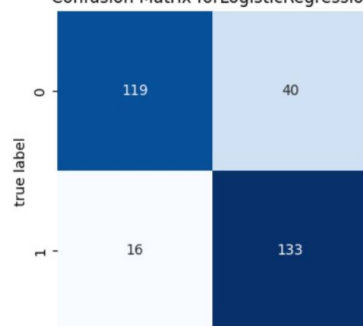
	precision	recall	f1-score	support
0	0.96	1.00	0.98	159
1	1.00	0.96	0.98	149
accuracy			0.98	308
macro avg	0.98	0.98	0.98	308
weighted avg	0.98	0.98	0.98	308

```
[13]: LR= LogisticRegression()
LR.fit(x_train,y_train)
y_pred= LR.predict(x_test)
evaluation_results= evaluate_model('LogisticRegression',y_test,y_pred)
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"(key):(value:4f)" if isinstance(value, float) else f"(key): \n{value}")
```

```
C:\Users\imad\anaconda3\envs\code1\Lib\site-packages\sklearn\linear_model\logistic.py:406: ConvergenceWarning: lbfgs failed to converge after 100 iterations(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

```
Increase the number of iterations to improve the convergence (max_iter=100).
You might also want to 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(
```

Confusion Matrix forLogisticRegression



```

Model Name:
LogisticRegression
Accuracy:0.818182
Precision:0.826963
Recall:0.818182
F1 Score:0.817537

```

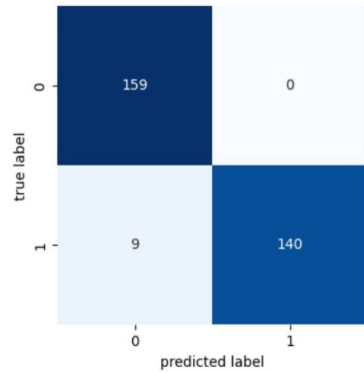
	precision	recall	f1-score	support
0	0.88	0.75	0.81	159
1	0.77	0.89	0.83	149
accuracy			0.82	308
macro avg	0.83	0.82	0.82	308
weighted avg	0.83	0.82	0.82	308

```

[14]: DT= DecisionTreeClassifier()
DT.fit(x_train,y_train)
y_pred= DT.predict(x_test)
evaluation_results= evaluate_model('DecisionTreeClassifier',y_test,y_pred)
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}:{value:4f}" if isinstance(value, float) else f"{key}: \n{value}")

```

Confusion Matrix forDecisionTreeClassifier



```

Model Name:
DecisionTreeClassifier
Accuracy:0.970779
Precision:0.972345
Recall:0.970779
F1 Score:0.970726

```

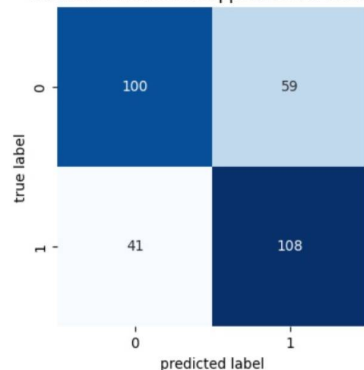
	precision	recall	f1-score	support
0	0.95	1.00	0.97	159
1	1.00	0.94	0.97	149
accuracy			0.97	308
macro avg	0.97	0.97	0.97	308
weighted avg	0.97	0.97	0.97	308

```

[15]: SVM= SVC()
SVM.fit(x_train,y_train)
y_pred= SVM.predict(x_test)
evaluation_results= evaluate_model('Support Vector Machine',y_test,y_pred)
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}:{value:4f}" if isinstance(value, float) else f"{key}: \n{value}")

```

Confusion Matrix forSupport Vector Machine



```

Model Name:
Support Vector Machine
Accuracy:0.675325
Precision:0.678978
Recall:0.675325
F1 Score:0.674831

```

	precision	recall	f1-score	support
0	0.71	0.63	0.67	159
1	0.65	0.72	0.68	149
accuracy			0.68	308
macro avg	0.68	0.68	0.68	308
weighted avg	0.68	0.68	0.67	308

```

[17]: rf_classifier = RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [100,200,300],
    'max_depth': [None,10,20,30],
    'min_samples_split': [2,5,10],
    'min_samples_leaf': [1,2,4]
}
grid_search = GridSearchCV( estimator= rf_classifier,
    param_grid=param_grid, cv=5,
    scoring='accuracy', n_jobs=-1)

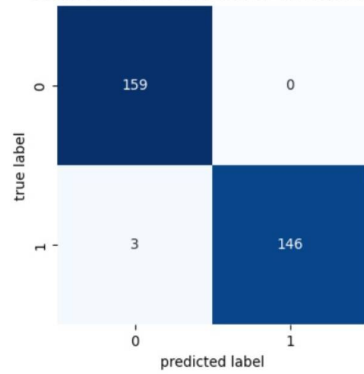
```

```

best_rf_classifier = grid_search.best_estimator_
y_pred = best_rf_classifier.predict(x_test)
evaluation_results = evaluate_model('RandomForestClassifier',y_test,y_pred)
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}:{value:4f}" if isinstance(value, float) else f"{key}: \n{value}")
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for RandomForestClassifier



Model Name:
RandomForestClassifier
Accuracy:0.990260
Precision:0.990440
Recall:0.990260
F1 Score:0.990256

	precision	recall	f1-score	support
0	0.98	1.00	0.99	159
1	1.00	0.98	0.99	149
accuracy			0.99	308
macro avg	0.99	0.99	0.99	308
weighted avg	0.99	0.99	0.99	308

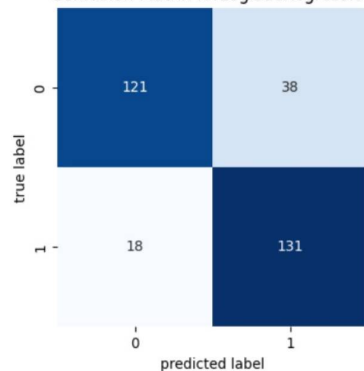
Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

```

[18]: LR= LogisticRegression(random_state=42)
param_grid = {
    'C' : [0.01,0.1,1,10],
    'solver' : ['liblinear','lbfgs'],
    'max_iter':[100,200,300],
}
grid_search = GridSearchCV( estimator= LR,
    param_grid=param_grid, cv=5,
    scoring='accuracy',n_jobs=-1)
grid_search.fit(x_train,y_train)
best_LR = grid_search.best_estimator_
y_pred = best_LR.predict(x_test)
evaluation_results = evaluate_model('LogisticRegression',y_test,y_pred)
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}:{value:4f}" if isinstance(value, float) else f"{key}: \n{value}")
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for LogisticRegression



Model Name:
LogisticRegression
Accuracy:0.818182
Precision:0.824374
Recall:0.818182
F1 Score:0.817798

	precision	recall	f1-score	support
0	0.87	0.76	0.81	159
1	0.78	0.88	0.82	149
accuracy			0.82	308
macro avg	0.82	0.82	0.82	308
weighted avg	0.82	0.82	0.82	308

Best hyperparameters found by GridSearchCV:
{'C': 1, 'max_iter': 100, 'solver': 'liblinear'}

```

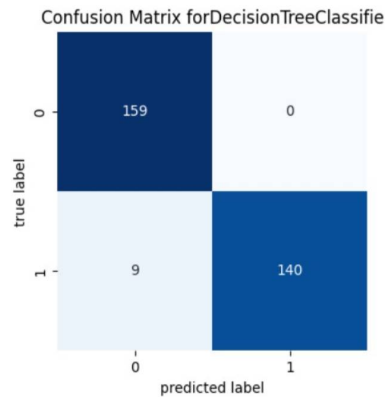
[19]: DT= DecisionTreeClassifier(random_state=42)
param_grid = {
    'criterion' : ['gini','entropy'],
    'max_depth' : [None,10,20,30],
    'min_samples_split':[2,5,10],
    'min_samples_leaf':[1,2,4],
    'max_features':['None','sqrt','log2']
}

```

```

grid_search = GridSearchCV( estimator=DT,
    param_grid=param_grid, cv=5,
    scoring='accuracy',n_jobs=-1)
grid_search.fit(x_train,y_train)
best_DT = grid_search.best_estimator_
y_pred = best_DT.predict(x_test)
evaluation_results = evaluate_model('DecisionTreeClassifie',y_test,y_pred)
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"(key):(value:4f)" if isinstance(value, float) else f"(key): \n(value)")
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```



Model Name:
DecisionTreeClassifie
Accuracy:0.970779
Precision:0.972345
Recall:0.970779
F1 Score:0.970726

	precision	recall	f1-score	support
0	0.95	1.00	0.97	159
1	1.00	0.94	0.97	149
accuracy			0.97	308
macro avg	0.97	0.97	0.97	308
weighted avg	0.97	0.97	0.97	308

Best hyperparameters found by GridSearchCV:
{'criterion': 'gini', 'max_depth': None, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2}

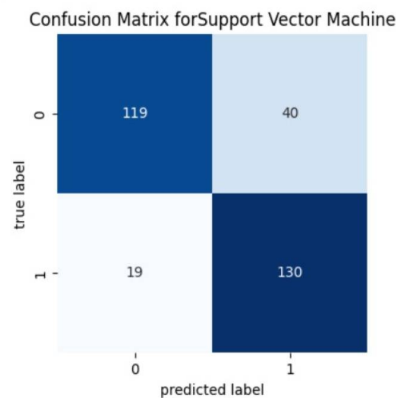
```

[20]: SVM= SVC(random_state=42)
param_grid={
    'C': [0.01,0.1,1,10],
    'kernel': ['linear'],

    'class_weight':[None,'balanced']}

grid_search = GridSearchCV( estimator= SVM,
    param_grid=param_grid, cv=5,
    scoring='accuracy',n_jobs=-1)
grid_search.fit(x_train,y_train)
best_SVM = grid_search.best_estimator_
y_pred = best_SVM.predict(x_test)
evaluation_results = evaluate_model('Support Vector Machine',y_test,y_pred)
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"(key):(value:4f)" if isinstance(value, float) else f"(key): \n(value)")
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```



Model Name:
Support Vector Machine
Accuracy:0.808442
Precision:0.815097
Recall:0.808442
F1 Score:0.807975

	precision	recall	f1-score	support
0	0.86	0.75	0.80	159
1	0.76	0.87	0.82	149
accuracy			0.81	308
macro avg	0.81	0.81	0.81	308
weighted avg	0.82	0.81	0.81	308

Best hyperparameters found by GridSearchCV:
{'C': 1, 'class_weight': None, 'kernel': 'linear'}

```

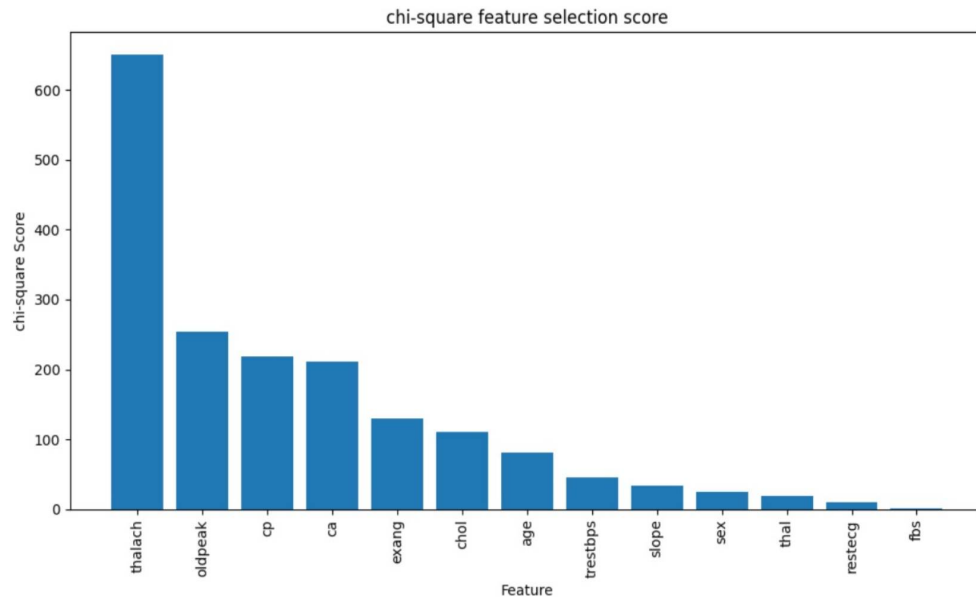
[28]: from sklearn.feature_selection import SelectKBest,chi2
selector = SelectKBest(score_func=chi2, k=5)

```

```
print("Selected Features:",selector.get_support())
print("scores:",selector.scores_)
```

```
Selected Features: [False False  True False False False False  True  True  True False  True
 False]
scores: [ 81.42536762 24.37365008 217.82392171 45.97406854 110.72336371
 1.47754962   9.73934262 650.00849349 130.47092725 253.65346109
33.67394807 210.62591949 19.37346461]
```

```
[29]: df=pd.DataFrame({'Feature':x.columns,'Score':selector.scores_})
df=df.sort_values(by='Score',ascending=False)
plt.figure(figsize=(10,6))
plt.bar(df['Feature'],df['Score'])
plt.title('chi-square feature selection score')
plt.xlabel('Feature')
plt.ylabel('chi-square Score')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



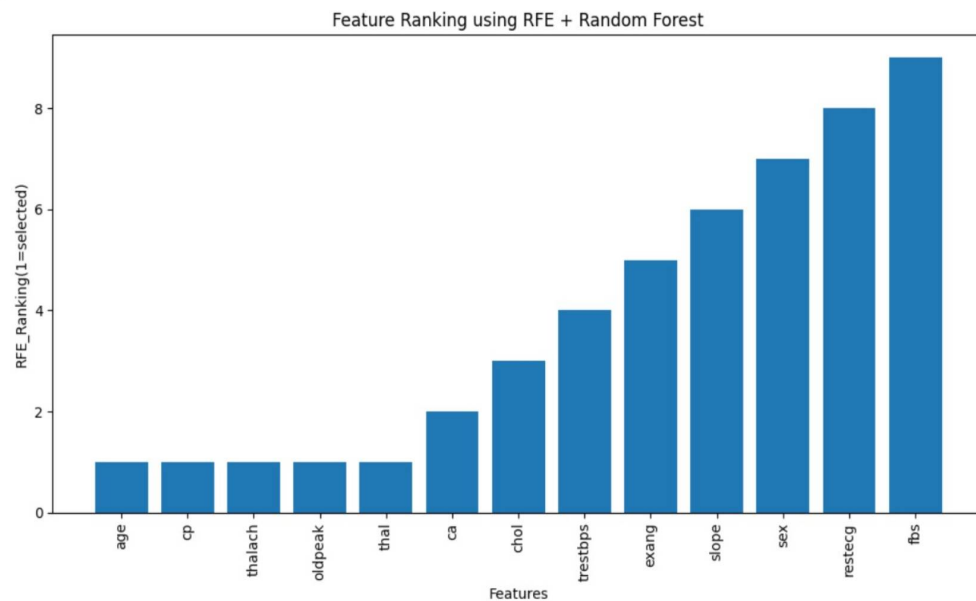
```
[30]: from sklearn.feature_selection import RFE
```

```
[31]: rf=RandomForestClassifier(n_estimators=200,random_state=42)
rfe=RFE(estimator=rf,n_features_to_select=5)
rfe.fit(x,y)
print("Selected Feature Mask:",rfe.get_support())
print("Ranking of Features:",rfe.ranking_)
```

```
Selected Feature Mask: [ True False  True False False False False  True False  True False False
 True]
Ranking of Features: [1 7 1 4 3 9 8 1 5 1 6 2 1]
```

```
[32]: ranking_df=pd.DataFrame({
    'Feature':x.columns,
    'RFE_Ranking': rfe.ranking_
}).sort_values('RFE_Ranking')

plt.figure(figsize=(10,6))
plt.bar(ranking_df['Feature'],ranking_df['RFE_Ranking'])
plt.title('Feature Ranking using RFE + Random Forest')
plt.xlabel('Features')
plt.ylabel('RFE_Ranking(1=selected)')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



```
[ ]: # comparing models using features selected by the chi-square
```

```
[33]: x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=42)
k=5
selector=SelectKBest(chi2,k=k)
x_train_k=selector.fit_transform(x_train,y_train)
x_test_k= selector.transform(x_test)
selected_features= x.columns[selector.get_support()]
```

Selected Features: ['cp', 'chol', 'thalach', 'oldpeak', 'ca']

```
models={"Random Forest": RandomForestClassifier(n_estimators=200,random_state=420,"Decision Tree":DecisionTreeClassifier(random_state=42), "SVM":SVC(kernel='rbf') }
metrics = [] for name, model in models.items(): model.fit(x_train,y_train) y_pred= model.predict(x_test_k) metrics.append( {"Model":name,
"Accuracy":accuracy_score(y_test,y_pred), "precision":precision_score(y_test,y_pred,average='weighted'), "recall": recall_score(y_test,y_pred,average='weighted')})
```

```
[42]: models={"Random Forest" : RandomForestClassifier(n_estimators=200,random_state=42),
        "Decision Tree":DecisionTreeClassifier(random_state=42),
        "SVM":SVC(kernel='rbf')
      }
metrics = []
for name, model in models.items():
    model.fit(x_train_k,y_train)
    y_pred= model.predict(x_test_k)
    metrics.append({
        "Model":name,
        "Accuracy":accuracy_score(y_test,y_pred),
        "precision":precision_score(y_test,y_pred,average='weighted'),
        "recall": recall_score(y_test,y_pred,average='weighted')
    })

results_df=pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_df)
```

Model Performance:

	Model	Accuracy	precision	recall
0	Random Forest	0.970779	0.972345	0.970779
1	Decision Tree	0.970779	0.972345	0.970779
2	SVM	0.685065	0.687828	0.685065

```
[46]: #comparing models using features selected by RFE
x_train,x_test,y_train,y_test= train_test_split(
x,y,test_size=0.2,random_state=42)
k=5
rf_selector= RandomForestClassifier(n_estimators=200,random_state=42)
rfe=RFE(estimator=rf_selector,n_features_to_select=k)
rfe.fit(x_train,y_train)
x_train_k=rfe.transform(x_train)
x_test_k=rfe.transform(x_test)
selected_features= x.columns[rfe.get_support()]
print("Selected Features by RFE:",list(selected_features))
```

Selected Features by RFE: ['cp', 'thalach', 'oldpeak', 'ca', 'thal']

```
[47]: models={"Random Forest" : RandomForestClassifier(n_estimators=200,random_state=42),
        "Decision Tree":DecisionTreeClassifier(random_state=42),
        "SVM":SVC(kernel='rbf')
      }
metrics = []
for name, model in models.items():
    model.fit(x_train_k,y_train)
    y_pred= model.predict(x_test_k)
    metrics.append({
        "Model":name,
        "Accuracy":accuracy_score(y_test,y_pred),
        "precision":precision_score(y_test,y_pred,average='weighted'),
        "recall": recall_score(y_test,y_pred,average='weighted')
    })

results_df=pd.DataFrame(metrics)
print("\nModel Performance Using RFE-Selected Features:")
print(results_df)
```

Model Performance Using RFE-Selected Features:

	Model	Accuracy	precision	recall
0	Random Forest	0.980488	0.980673	0.980488
1	Decision Tree	0.965854	0.966972	0.965854
2	SVM	0.658537	0.659897	0.658537

[]: