

```
In [43]: # Dataset: Heart Failure Prediction Dataset
# https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction
```

```
In [44]: from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)
```

```
In [45]: import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [46]: import warnings
warnings.filterwarnings("ignore")
```

```
In [47]: from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from sklearn.feature_selection import SelectKBest, mutual_info_classif, SelectFr
```

```
In [48]: data = pd.read_csv("heart_failure_clinical_records_dataset.csv")
data.head()
```

```
Out[48]:
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pres
0	75.0	0	582	0	20	
1	55.0	0	7861	0	38	
2	65.0	0	146	0	20	
3	50.0	1	111	0	20	
4	65.0	1	160	1	20	



```
In [49]: data.columns
```

```
Out[49]: Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
               'ejection_fraction', 'high_blood_pressure', 'platelets',
               'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
               'DEATH_EVENT'],
              dtype='object')
```

```
In [50]: data.dtypes
```

```
Out[50]: age                float64
         anaemia            int64
         creatinine_phosphokinase  int64
         diabetes           int64
         ejection_fraction  int64
         high_blood_pressure  int64
         platelets          float64
         serum_creatinine    float64
         serum_sodium        int64
         sex                 int64
         smoking             int64
         time                int64
         DEATH_EVENT         int64
         dtype: object
```

```
In [51]: data.isnull().sum()
```

```
Out[51]: age                0
         anaemia            0
         creatinine_phosphokinase  0
         diabetes           0
         ejection_fraction  0
         high_blood_pressure  0
         platelets          0
         serum_creatinine    0
         serum_sodium        0
         sex                 0
         smoking             0
         time                0
         DEATH_EVENT         0
         dtype: int64
```

```
In [52]: missing_percentage = data.isnull().mean() * 100
         threshold = 30
         columns_to_drop = missing_percentage[missing_percentage > threshold].index
         df = data.drop(columns=columns_to_drop)
         print(f"Columns dropped: {list(columns_to_drop)}")
         print(f"Remaining columns: {df.columns}")
```

Columns dropped: []

Remaining columns: Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',

 'ejection_fraction', 'high_blood_pressure', 'platelets',
 'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
 'DEATH_EVENT'],
 dtype='object')

```
In [53]: for column in data.select_dtypes(include=['float64']).columns:
         mean_value = data[column].mean()
         data[column] = data[column].fillna(mean_value)

         for column in data.select_dtypes(include=['object']).columns:
         mode_value = data[column].mode()[0]
         data[column] = data[column].fillna(mode_value)

         data.isnull().sum()
```

```
Out[53]: age                0
         anaemia            0
         creatinine_phosphokinase  0
         diabetes           0
         ejection_fraction  0
         high_blood_pressure  0
         platelets          0
         serum_creatinine    0
         serum_sodium        0
         sex                0
         smoking            0
         time               0
         DEATH_EVENT         0
         dtype: int64
```

```
In [54]: X = data.drop("DEATH_EVENT", axis=1)
         y = data["DEATH_EVENT"]
```

```
In [55]: X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size=0.2, random_state=42, stratify=y
         )
```

```
In [56]: scaler = StandardScaler()
         scaler.fit(X_train)
         X_train = scaler.transform(X_train)
         X_test = scaler.transform(X_test)
```

```
In [57]: def evaluate_model(model_name, y_true, y_pred):

         accuracy = accuracy_score(y_true, y_pred)
         precision = precision_score(y_true, y_pred)
         recall = recall_score(y_true, y_pred)
         f1 = f1_score(y_true, y_pred)

         cm = confusion_matrix(y_true, y_pred)
         report = classification_report(y_true, y_pred)

         print("\n=====")
         print("Model:", model_name)
         print("Accuracy:", accuracy)
         print("Precision:", precision)
         print("Recall:", recall)
         print("F1-score:", f1)
         print("\nClassification Report:\n", report)

         plt.figure(figsize=(4,4))
         sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
         plt.title(f"Confusion Matrix - {model_name}")
         plt.xlabel("Predicted")
         plt.ylabel("Actual")
         plt.show()
```

```
In [58]: lr = LogisticRegression(max_iter=2000)

         lr_params = {
             "C": [0.1, 1, 10]
         }
```

```

lr_grid = GridSearchCV(
    lr,
    lr_params,
    cv=5,
    scoring="f1"
)

lr_grid.fit(X_train, y_train)

best_lr = lr_grid.best_estimator_
print("Best LR Parameters:", lr_grid.best_params_)

lr_preds = best_lr.predict(X_test)
evaluate_model("Logistic Regression (GridSearch)", y_test, lr_preds)

```

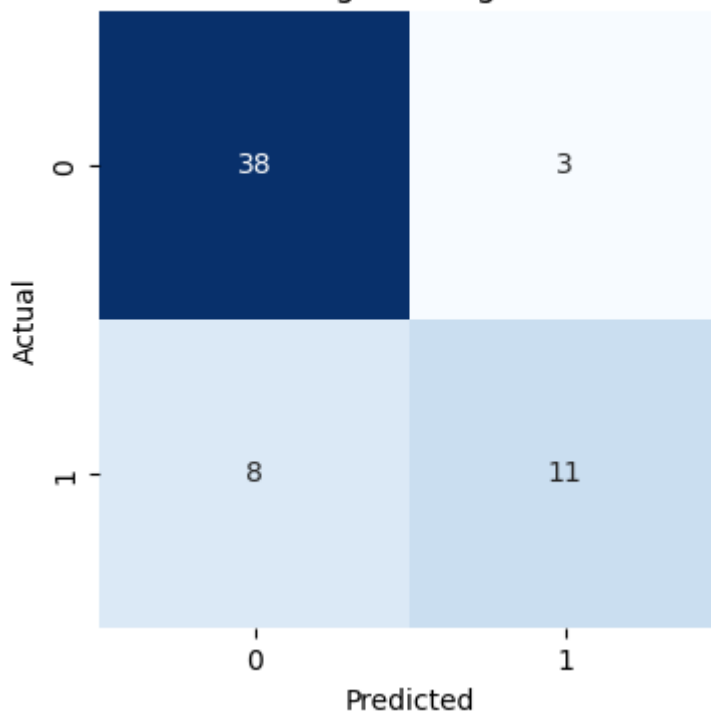
Best LR Parameters: {'C': 10}

=====
Model: Logistic Regression (GridSearch)
Accuracy: 0.8166666666666667
Precision: 0.7857142857142857
Recall: 0.5789473684210527
F1-score: 0.6666666666666666

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.93	0.87	41
1	0.79	0.58	0.67	19
accuracy			0.82	60
macro avg	0.81	0.75	0.77	60
weighted avg	0.81	0.82	0.81	60

Confusion Matrix - Logistic Regression (GridSearch)



In [59]: rf = RandomForestClassifier(random_state=42)

```

rf_params = {
    "n_estimators": [100, 200],
    "max_depth": [None, 5, 10]
}

rf_grid = GridSearchCV(
    rf,
    rf_params,
    cv=5,
    scoring="f1",
    n_jobs=-1
)

rf_grid.fit(X_train, y_train)

best_rf = rf_grid.best_estimator_
print("Best RF Parameters:", rf_grid.best_params_)

rf_preds = best_rf.predict(X_test)
evaluate_model("Random Forest (GridSearch)", y_test, rf_preds)

```

Best RF Parameters: {'max_depth': 5, 'n_estimators': 100}

```

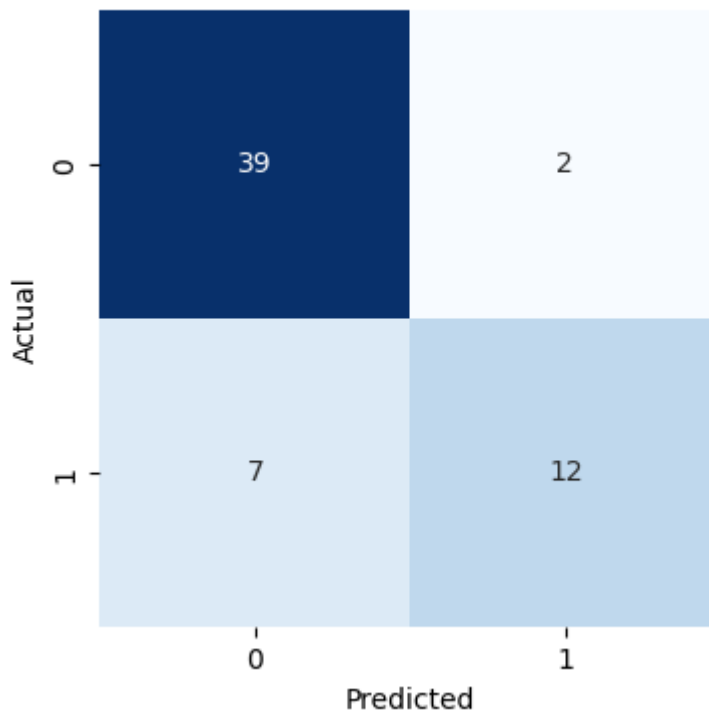
=====
Model: Random Forest (GridSearch)
Accuracy: 0.85
Precision: 0.8571428571428571
Recall: 0.631578947368421
F1-score: 0.7272727272727273

```

Classification Report:

	precision	recall	f1-score	support
0	0.85	0.95	0.90	41
1	0.86	0.63	0.73	19
accuracy			0.85	60
macro avg	0.85	0.79	0.81	60
weighted avg	0.85	0.85	0.84	60

Confusion Matrix - Random Forest (GridSearch)



```
In [60]: svm = SVC()

svm_params = {"C": [0.5, 1, 5], "kernel": ["rbf", "linear"]}

svm_grid = GridSearchCV(svm, svm_params, cv=5, scoring="f1")
svm_grid.fit(X_train, y_train)

print("Best SVM Params:", svm_grid.best_params_)

best_svm = svm_grid.best_estimator_
svm_preds = best_svm.predict(X_test)

evaluate_model("SVM (GridSearch)", y_test, svm_preds)
```

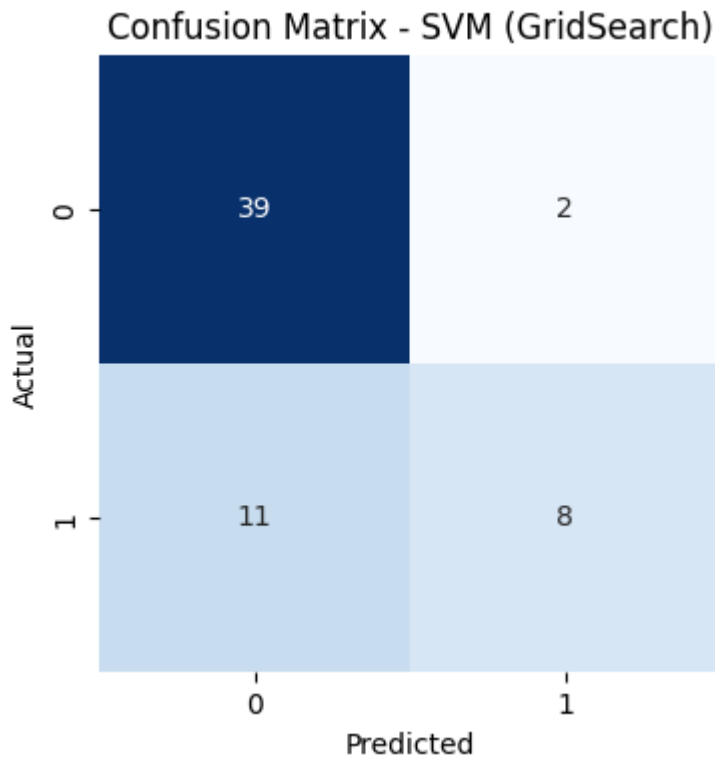
Best SVM Params: {'C': 1, 'kernel': 'linear'}

=====

Model: SVM (GridSearch)
 Accuracy: 0.7833333333333333
 Precision: 0.8
 Recall: 0.42105263157894735
 F1-score: 0.5517241379310345

Classification Report:

	precision	recall	f1-score	support
0	0.78	0.95	0.86	41
1	0.80	0.42	0.55	19
accuracy			0.78	60
macro avg	0.79	0.69	0.70	60
weighted avg	0.79	0.78	0.76	60



```
In [61]: selector_kbest = SelectKBest(
          score_func=mutual_info_classif,
          k=5
        )

X_train_k = selector_kbest.fit_transform(X_train, y_train)
X_test_k = selector_kbest.transform(X_test)

print("Shape before FS:", X_train.shape)
print("Shape after FS1:", X_train_k.shape)
```

Shape before FS: (239, 12)
Shape after FS1: (239, 5)

```
In [62]: # Logistic Regression after SelectKBest
lr_fs = LogisticRegression(max_iter=2000)
lr_fs.fit(X_train_k, y_train)
preds_k = lr_fs.predict(X_test_k)
evaluate_model("LR + SelectKBest", y_test, preds_k)
```

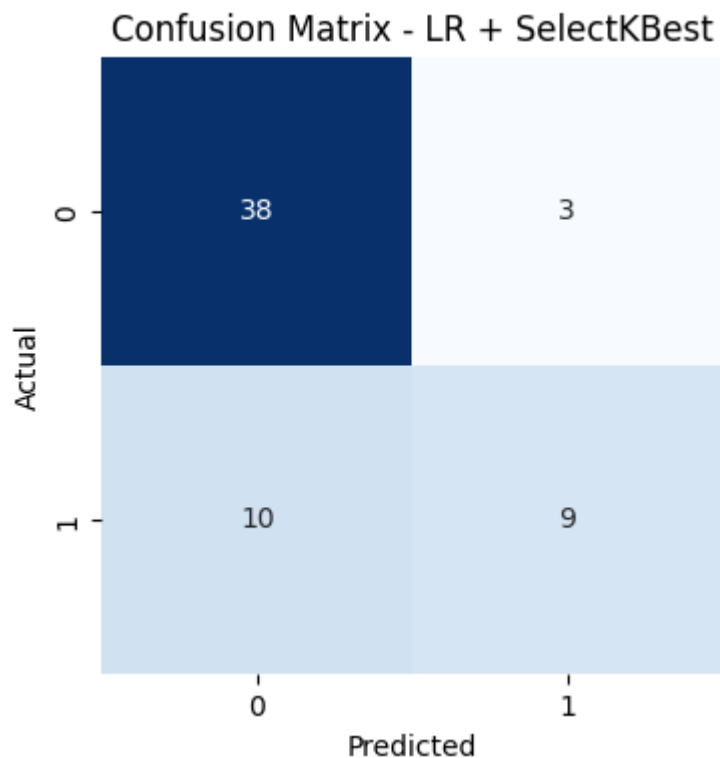
```

=====
Model: LR + SelectKBest
Accuracy: 0.7833333333333333
Precision: 0.75
Recall: 0.47368421052631576
F1-score: 0.5806451612903226

```

Classification Report:

	precision	recall	f1-score	support
0	0.79	0.93	0.85	41
1	0.75	0.47	0.58	19
accuracy			0.78	60
macro avg	0.77	0.70	0.72	60
weighted avg	0.78	0.78	0.77	60



```

In [63]: rf_fs = RandomForestClassifier(n_estimators=200, random_state=42)
rf_fs.fit(X_train, y_train)

selector_model = SelectFromModel(
    rf_fs,
    threshold="median",
    prefit=True
)

X_train_m = selector_model.transform(X_train)
X_test_m = selector_model.transform(X_test)

print("Shape before FS:", X_train.shape)
print("Shape after FS2:", X_train_m.shape)

```

Shape before FS: (239, 12)

Shape after FS2: (239, 6)

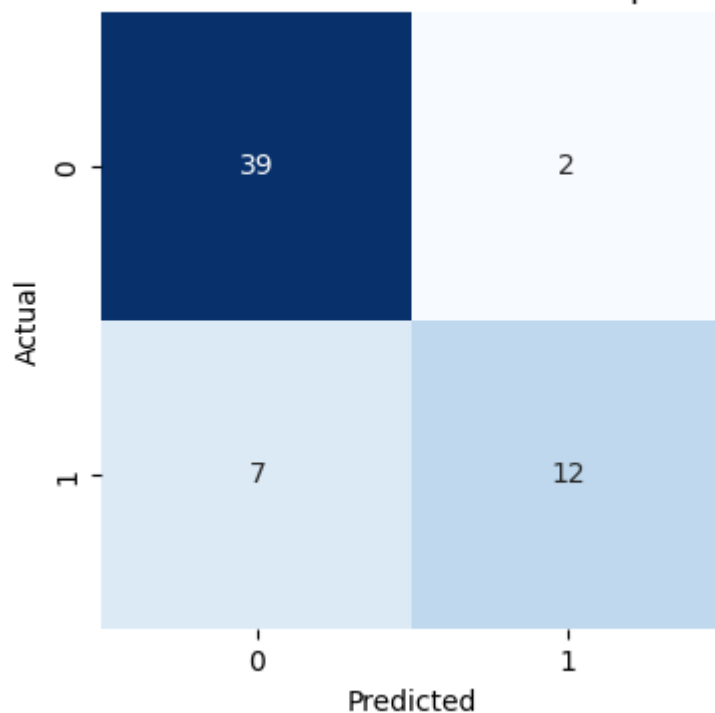
```
In [64]: # Random Forest after SelectFromModel
rf_fs2 = RandomForestClassifier(random_state=42)
rf_fs2.fit(X_train_m, y_train)
preds_m = rf_fs2.predict(X_test_m)
evaluate_model("RF + Feature Importance", y_test, preds_m)
```

```
=====
Model: RF + Feature Importance
Accuracy: 0.85
Precision: 0.8571428571428571
Recall: 0.631578947368421
F1-score: 0.7272727272727273
```

Classification Report:

	precision	recall	f1-score	support
0	0.85	0.95	0.90	41
1	0.86	0.63	0.73	19
accuracy			0.85	60
macro avg	0.85	0.79	0.81	60
weighted avg	0.85	0.85	0.84	60

Confusion Matrix - RF + Feature Importance



In []: