

```
In [3]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
```

```
In [4]: heart_set = pd.read_csv('heart.csv')
```

```
In [5]: heart_set.head(10)
```

```
Out[5]:
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina
0	40	M	ATA	140	289	0	Normal	172	0
1	49	F	NAP	160	180	0	Normal	156	0
2	37	M	ATA	130	283	0	ST	98	0
3	48	F	ASY	138	214	0	Normal	108	0
4	54	M	NAP	150	195	0	Normal	122	0
5	39	M	NAP	120	339	0	Normal	170	0
6	45	F	ATA	130	237	0	Normal	170	0
7	54	M	ATA	110	208	0	Normal	142	0
8	37	M	ASY	140	207	0	Normal	130	0
9	48	F	ATA	120	284	0	Normal	120	0

```
In [6]: heart_set.describe()
```

Out[6]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisea
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.0000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.5533
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.4974
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.0000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.0000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.0000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.0000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.0000

In [11]: `heart_set.isnull().sum()`

```
Out[11]: 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
```

In [13]: `print(heart_set.dtypes)`

```
Age          int64
Sex          object
ChestPainType object
RestingBP    int64
Cholesterol  int64
FastingBS    int64
RestingECG   object
MaxHR        int64
ExerciseAngina object
Oldpeak      float64
ST_Slope     object
HeartDisease int64
dtype: object
```

In [15]: `print('Total: ', len(heart_set))`

Total: 918

```
In [17]: heart_disease = heart_set[heart_set['HeartDisease'] == 1]
        No_heart_disease = heart_set[heart_set['HeartDisease'] == 0]
```

```
In [19]: print('Total Patients with heart diseases: ', len(heart_disease))
        print('Total Patients without heart diseases: ', len(No_heart_disease))
```

Total Patients with heart diseases: 508
Total Patients without heart diseases: 410

```
In [21]: from sklearn.preprocessing import LabelEncoder
        le = LabelEncoder()
        for col in ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']:
            heart_set[col] = le.fit_transform(heart_set[col])
```

```
In [23]: heart_set
```

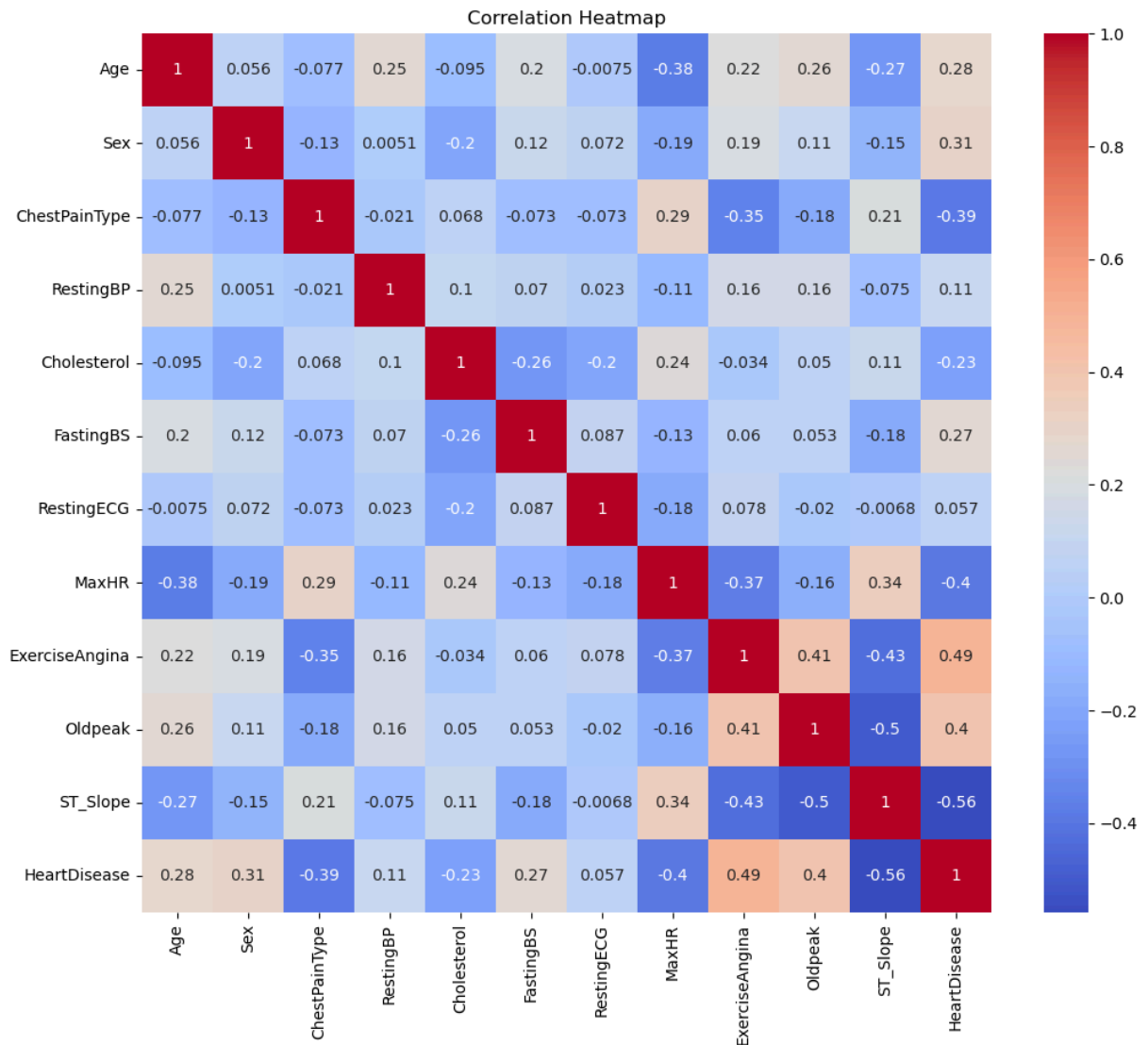
```
Out[23]:
```

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

918 rows × 12 columns



```
In [25]: plt.figure(figsize=(12, 10))
        sns.heatmap(heart_set.corr(), annot=True, cmap='coolwarm')
        plt.title('Correlation Heatmap')
        plt.show()
```



```
In [26]: X = heart_set.drop('HeartDisease', axis = 1).values
y = heart_set['HeartDisease'].values
```

```
In [29]: X
```

```
Out[29]: array([[40. , 1. , 1. , ..., 0. , 0. , 2. ],
 [49. , 0. , 2. , ..., 0. , 1. , 1. ],
 [37. , 1. , 1. , ..., 0. , 0. , 2. ],
 ...,
 [57. , 1. , 0. , ..., 1. , 1.2, 1. ],
 [57. , 0. , 1. , ..., 0. , 0. , 1. ],
 [38. , 1. , 2. , ..., 0. , 0. , 2. ]])
```

```
In [31]: y
```

```
Out[31]: array([0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
                0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
                1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,
                0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,
                1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
                0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
                1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
                1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
                1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1,
                1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
                1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,
                0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
                1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
                1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
                1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
                1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
                1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1,
                0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
                1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1,
                0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0,
                1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
                0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1,
                1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
                1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
                1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
                1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
                0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
                0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0], dtype=int64)
```

```
In [33]: sc = StandardScaler()
X = sc.fit_transform(X)
print(X)
```

```

[[-1.4331398  0.51595242  0.22903206 ... -0.8235563 -0.83243239
  1.05211381]
 [-0.47848359 -1.93816322  1.27505906 ... -0.8235563  0.10566353
 -0.59607813]
 [-1.75135854  0.51595242  0.22903206 ... -0.8235563 -0.83243239
  1.05211381]
 ...
 [ 0.37009972  0.51595242 -0.81699495 ...  1.21424608  0.29328271
 -0.59607813]
 [ 0.37009972 -1.93816322  0.22903206 ... -0.8235563 -0.83243239
 -0.59607813]
 [-1.64528563  0.51595242  1.27505906 ... -0.8235563 -0.83243239
  1.05211381]]

```

```

In [35]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_st

```

```

In [37]: param_grid_lr = {
        'C': [0.1, 1, 10],
        'penalty': ['l2'],
        'solver': ['lbfgs', 'liblinear']}
    }
    grid_lr = GridSearchCV(LogisticRegression(random_state=0, max_iter=1000),
                           param_grid_lr, cv=5, scoring='accuracy', n_jobs=-1)
    grid_lr.fit(X_train, y_train)

    classifier = grid_lr.best_estimator_
    print("Best LR params:", grid_lr.best_params_)

    y_predict_test = classifier.predict(X_test)
    print(classification_report(y_test, y_predict_test))

```

```

Best LR params: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
precision    recall  f1-score   support

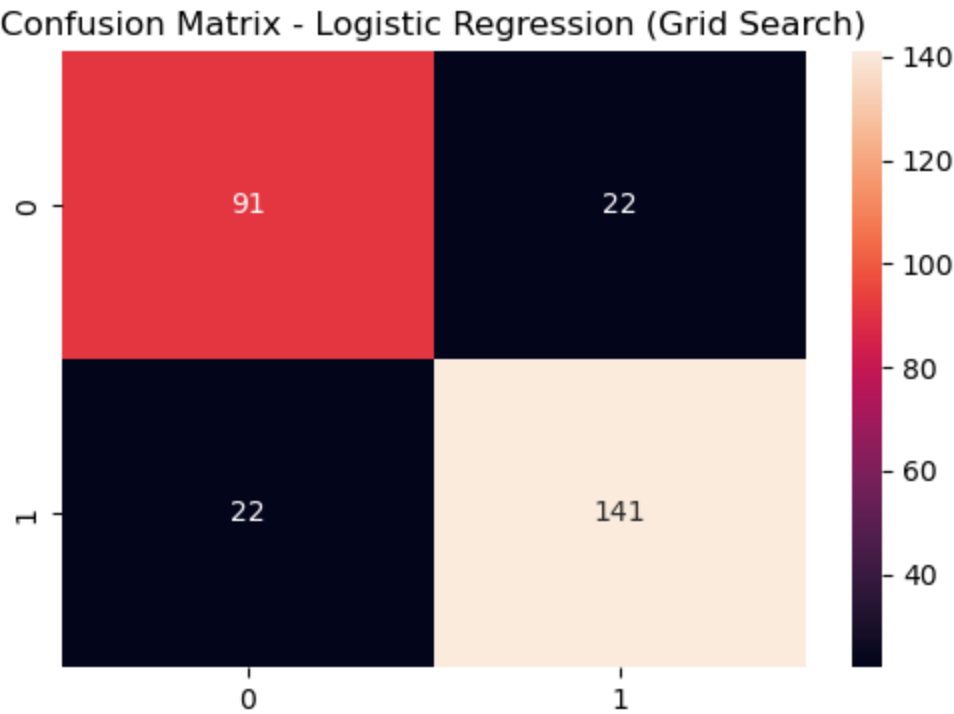
```

0	0.81	0.81	0.81	113
1	0.87	0.87	0.87	163
accuracy			0.84	276
macro avg	0.84	0.84	0.84	276
weighted avg	0.84	0.84	0.84	276

```

In [39]: cm = confusion_matrix(y_test, y_predict_test)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d')
plt.title('Confusion Matrix - Logistic Regression (Grid Search)')
plt.show()

```



```
In [41]: param_grid_svc = {
        'C': [0.1, 1, 10],
        'gamma': ['scale', 'auto', 0.1, 1],
        'kernel': ['rbf', 'linear']
    }
    grid_svc = GridSearchCV(SVC(random_state=0), param_grid_svc, cv=5, scoring='accuracy')
    grid_svc.fit(X_train, y_train)
```

Out[41]:

GridSearchCV ⓘ ?

estimator: SVC

SVC ?

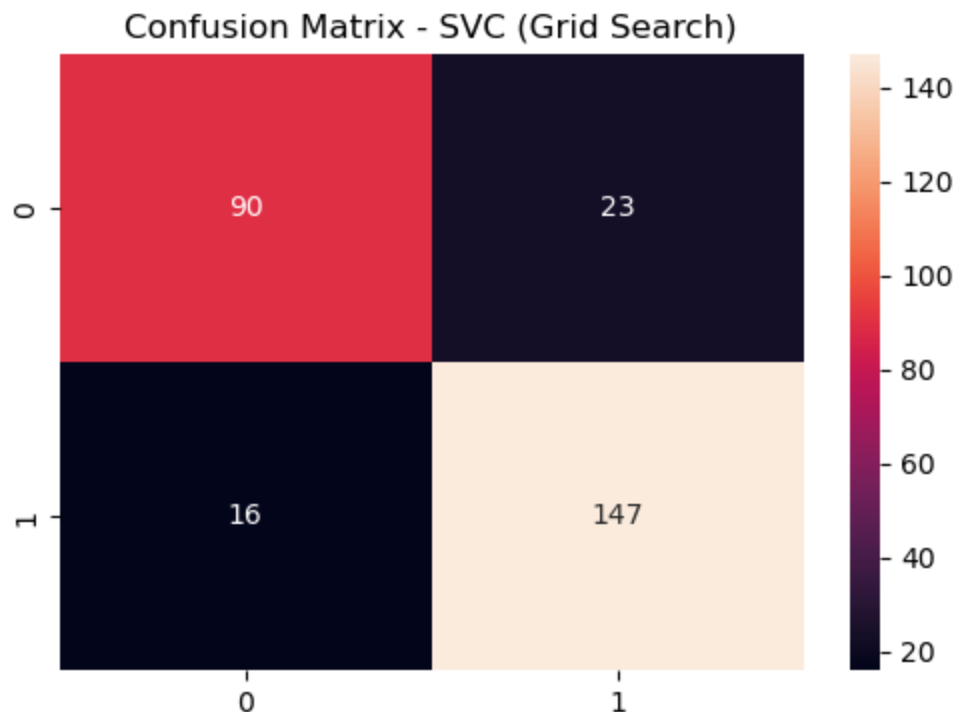
```
In [43]: best_svc = grid_svc.best_estimator_
        print("Best SVC params:", grid_svc.best_params_)
```

Best SVC params: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}

```
In [45]: y_pred_svc = best_svc.predict(X_test)
        print(classification_report(y_test, y_pred_svc))
```

	precision	recall	f1-score	support
0	0.85	0.80	0.82	113
1	0.86	0.90	0.88	163
accuracy			0.86	276
macro avg	0.86	0.85	0.85	276
weighted avg	0.86	0.86	0.86	276

```
In [47]: cm_svc = confusion_matrix(y_test, y_pred_svc)
plt.figure(figsize=(6,4))
sns.heatmap(cm_svc, annot=True, fmt='d')
plt.title('Confusion Matrix - SVC (Grid Search)')
plt.show()
```



```
In [51]: param_grid_knn = {
        'n_neighbors': [3, 5, 7, 9, 11],
        'weights': ['uniform', 'distance'],
        'p': [1, 2]
    }
grid_knn = GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5, scoring='accu
grid_knn.fit(X_train, y_train)
```

```
Out[51]:
```

GridSearchCV ⓘ ?

▶ **estimator: KNeighborsClassifier**

▶ **KNeighborsClassifier** ?

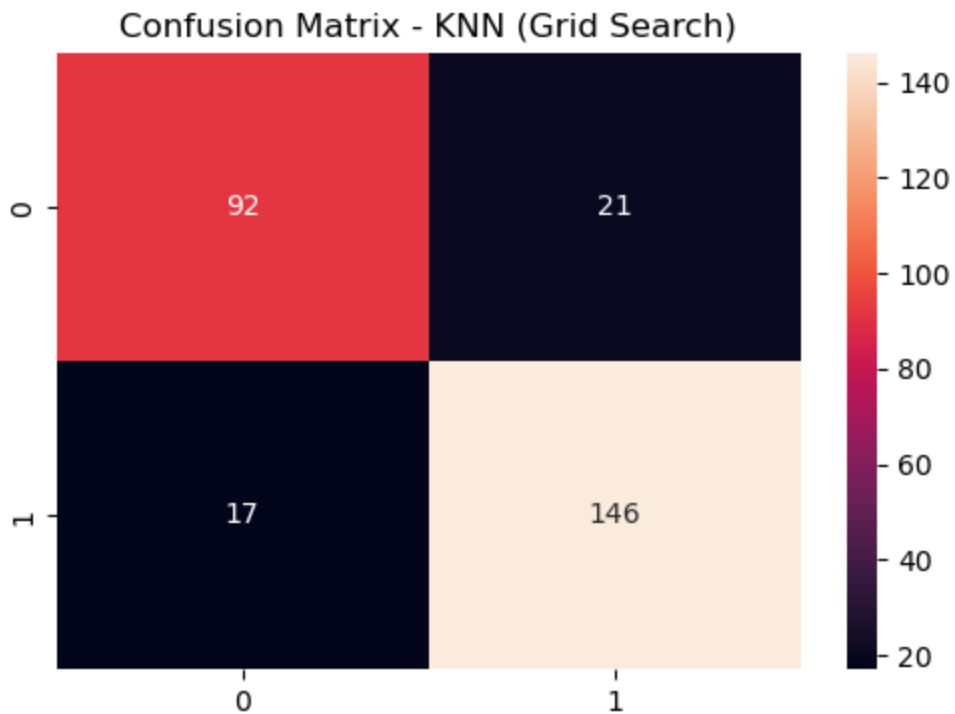
```
In [53]: best_knn = grid_knn.best_estimator_
print("Best KNN params:", grid_knn.best_params_)
```

Best KNN params: {'n_neighbors': 11, 'p': 1, 'weights': 'uniform'}

```
In [55]: y_pred_knn = best_knn.predict(X_test)
print(classification_report(y_test, y_pred_knn))
```

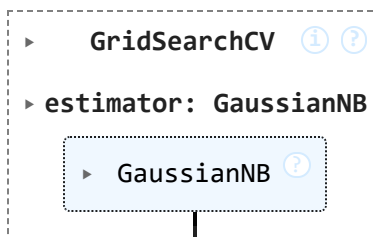
	precision	recall	f1-score	support
0	0.84	0.81	0.83	113
1	0.87	0.90	0.88	163
accuracy			0.86	276
macro avg	0.86	0.85	0.86	276
weighted avg	0.86	0.86	0.86	276

```
In [57]: cm_knn = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(6,4))
sns.heatmap(cm_knn, annot=True, fmt='d')
plt.title('Confusion Matrix - KNN (Grid Search)')
plt.show()
```



```
In [59]: param_grid_nb = {
          'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]
        }
grid_nb = GridSearchCV(GaussianNB(), param_grid_nb, cv=5, scoring='accuracy', n_job
grid_nb.fit(X_train, y_train)
```

Out[59]:



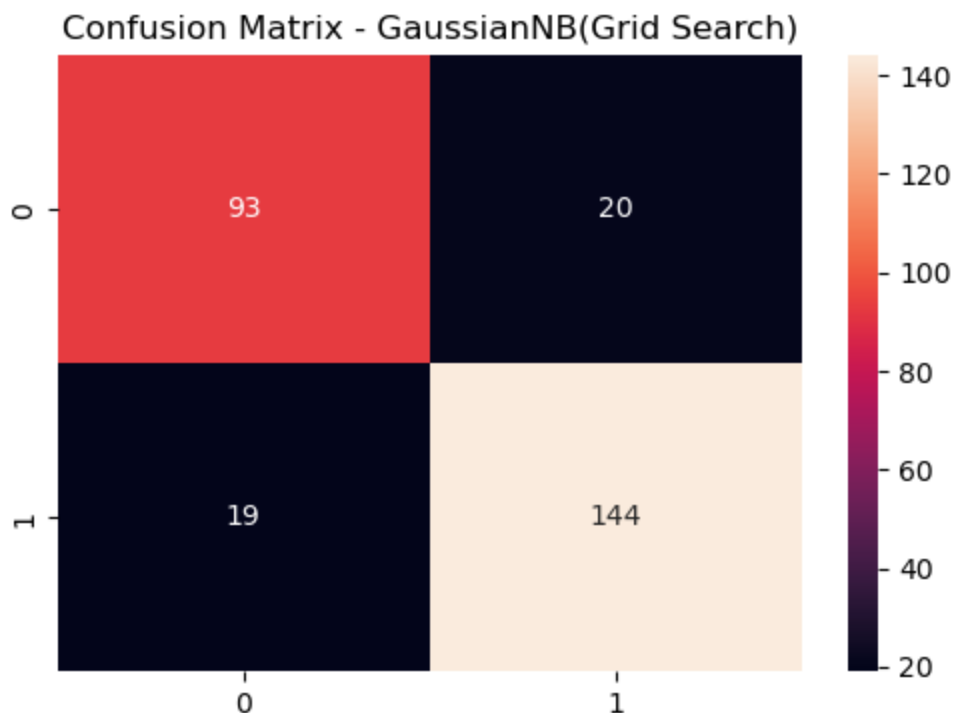
```
In [61]: best_nb = grid_nb.best_estimator_
print("Best GaussianNB params:", grid_nb.best_params_)
```

Best GaussianNB params: {'var_smoothing': 1e-09}

```
In [63]: y_pred_nb = best_nb.predict(X_test)
print(classification_report(y_test, y_pred_nb))
```

	precision	recall	f1-score	support
0	0.83	0.82	0.83	113
1	0.88	0.88	0.88	163
accuracy			0.86	276
macro avg	0.85	0.85	0.85	276
weighted avg	0.86	0.86	0.86	276

```
In [65]: cm_nb = confusion_matrix(y_test, y_pred_nb)
plt.figure(figsize=(6,4))
sns.heatmap(cm_nb, annot=True, fmt='d')
plt.title('Confusion Matrix - GaussianNB(Grid Search)')
plt.show()
```



```
In [69]: results = []
for name, y_pred in [('Logistic Regression', y_predict_test),
                    ('SVC', y_pred_svc),
                    ('KNN', y_pred_knn),
                    ('GaussianNB', y_pred_nb)]:
    acc = accuracy_score(y_test, y_pred)
    results.append([name, acc])

results_df = pd.DataFrame(results, columns=['Model', 'Test accuracy'])
print(results_df.sort_values('Test accuracy', ascending=False))
```

	Model	Test accuracy
2	KNN	0.862319
1	SVC	0.858696
3	GaussianNB	0.858696
0	Logistic Regression	0.840580

```
In [71]: from sklearn.feature_selection import SelectKBest, f_classif, RFE
        from sklearn.linear_model import LogisticRegression
```

```
In [73]: feature_names = ['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol',
                        'FastingBS', 'RestingECG', 'MaxHR', 'ExerciseAngina',
                        'Oldpeak', 'ST_Slope']
```

```
In [75]: selector_kbest = SelectKBest(score_func=f_classif, k=8) # keep top 8 features
        X_train_kbest = selector_kbest.fit_transform(X_train, y_train)
        X_test_kbest = selector_kbest.transform(X_test)
```

```
In [77]: selected_kbest = [feature_names[i] for i in selector_kbest.get_support(indices=True)]
        print("Selected features by SelectKBest:", selected_kbest)
```

Selected features by SelectKBest: ['Age', 'Sex', 'ChestPainType', 'FastingBS', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope']

```
In [79]: estimator_rfe = LogisticRegression(max_iter=1000, random_state=0)
        selector_rfe = RFE(estimator=estimator_rfe, n_features_to_select=8)
        X_train_rfe = selector_rfe.fit_transform(X_train, y_train)
        X_test_rfe = selector_rfe.transform(X_test)
```

```
In [81]: selected_rfe = [feature_names[i] for i in selector_rfe.get_support(indices=True)]
        print("Selected features by RFE:", selected_rfe)
```

Selected features by RFE: ['Sex', 'ChestPainType', 'FastingBS', 'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope']

```
In [83]: def evaluate_model(model, X_tr, X_te, y_tr, y_te, model_name, feature_method):
        model.fit(X_tr, y_tr)
        y_pred = model.predict(X_te)
        acc = accuracy_score(y_te, y_pred)
        print(f"\n{model_name} | {feature_method} | Accuracy: {acc:.4f}")
        print(classification_report(y_te, y_pred))
        return acc
```

```
In [85]: lr_best = LogisticRegression(C=0.1, penalty='l2', solver='liblinear', max_iter=1000)
        svc_best = SVC(C=1, gamma='scale', kernel='rbf', random_state=0)
        knn_best = KNeighborsClassifier(n_neighbors=11, p=1, weights='uniform')
        nb_best = GaussianNB(var_smoothing=1e-09)
```

```
In [87]: print("="*50)
        print("Results using SelectKBest features")
        print("="*50)
        evaluate_model(lr_best, X_train_kbest, X_test_kbest, y_train, y_test, "Logistic Reg
        evaluate_model(svc_best, X_train_kbest, X_test_kbest, y_train, y_test, "SVC", "Sele
```

```
=====
Results using SelectKBest features
=====
```

```
Logistic Regression | SelectKBest | Accuracy: 0.8370
      precision      recall  f1-score   support

      0      0.81      0.79      0.80      113
      1      0.86      0.87      0.86      163

   accuracy                0.84      276
  macro avg      0.83      0.83      0.83      276
weighted avg      0.84      0.84      0.84      276
```

```
SVC | SelectKBest | Accuracy: 0.8442
      precision      recall  f1-score   support

      0      0.82      0.80      0.81      113
      1      0.86      0.88      0.87      163

   accuracy                0.84      276
  macro avg      0.84      0.84      0.84      276
weighted avg      0.84      0.84      0.84      276
```

Out[87]: 0.8442028985507246

```
In [89]: print("\n" + "="*50)
print("Results using RFE features")
print("="*50)
evaluate_model(lr_best, X_train_rfe, X_test_rfe, y_train, y_test, "Logistic Regress
evaluate_model(svc_best, X_train_rfe, X_test_rfe, y_train, y_test, "SVC", "RFE")
```

```
=====
Results using RFE features
=====
```

Logistic Regression RFE Accuracy: 0.8297					
	precision	recall	f1-score	support	
0	0.80	0.78	0.79	113	
1	0.85	0.87	0.86	163	
accuracy			0.83	276	
macro avg	0.82	0.82	0.82	276	
weighted avg	0.83	0.83	0.83	276	

SVC RFE Accuracy: 0.8442					
	precision	recall	f1-score	support	
0	0.82	0.80	0.81	113	
1	0.86	0.88	0.87	163	
accuracy			0.84	276	
macro avg	0.84	0.84	0.84	276	
weighted avg	0.84	0.84	0.84	276	

Out[89]: 0.8442028985507246

In []: