

```

In [1]: import numpy as np

In [2]: import pandas as pd

In [3]: import matplotlib.pyplot as plt

In [4]: import seaborn as sns

In [5]: from sklearn.model_selection import train_test_split, GridSearchCV

In [6]: from sklearn.preprocessing import StandardScaler, LabelEncoder

In [7]: from sklearn.feature_selection import SelectKBest, chi2, RFE

In [8]: from sklearn.linear_model import LogisticRegression

In [9]: from sklearn.tree import DecisionTreeClassifier

In [10]: from sklearn.ensemble import RandomForestClassifier

In [11]: from sklearn.svm import SVC

In [12]: from sklearn.naive_bayes import GaussianNB

In [13]: from sklearn.neighbors import KNeighborsClassifier

In [14]: from sklearn.metrics import (
    confusion_matrix, accuracy_score, precision_score,
    recall_score, f1_score, classification_report
)

```

Read dataset

```

In [15]: df = pd.read_csv("Final_Project_Heart.csv")
df.head()

```

```

Out[15]:

```

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

Check missing values

```
In [16]: print(df.isnull().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
```

No missing values found and the dataset is clean

Encode Categorical Features

```
In [24]: label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column].astype(str))
    label_encoders[column] = le
```

Define features and target

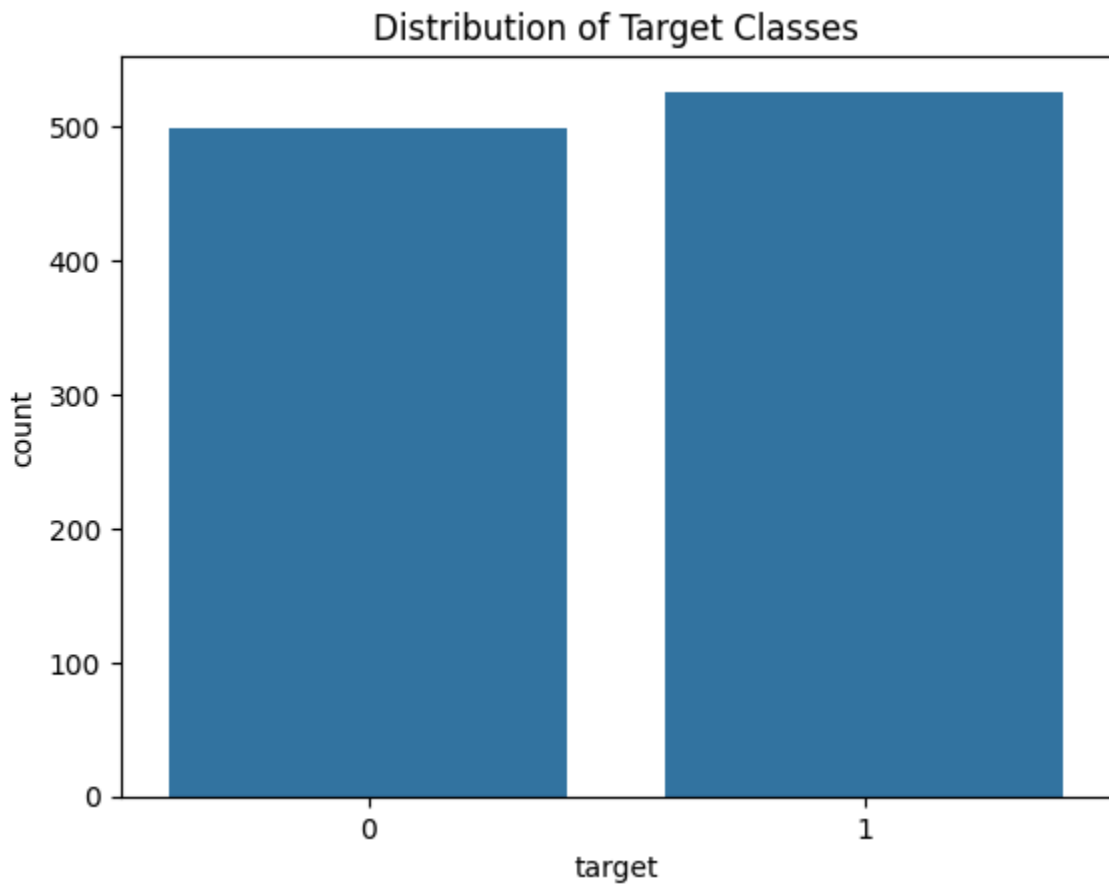
```
In [25]: X = df.drop("target", axis=1)
y = df["target"]

# Split dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
```

Exploratory Data Analysis (EDA)

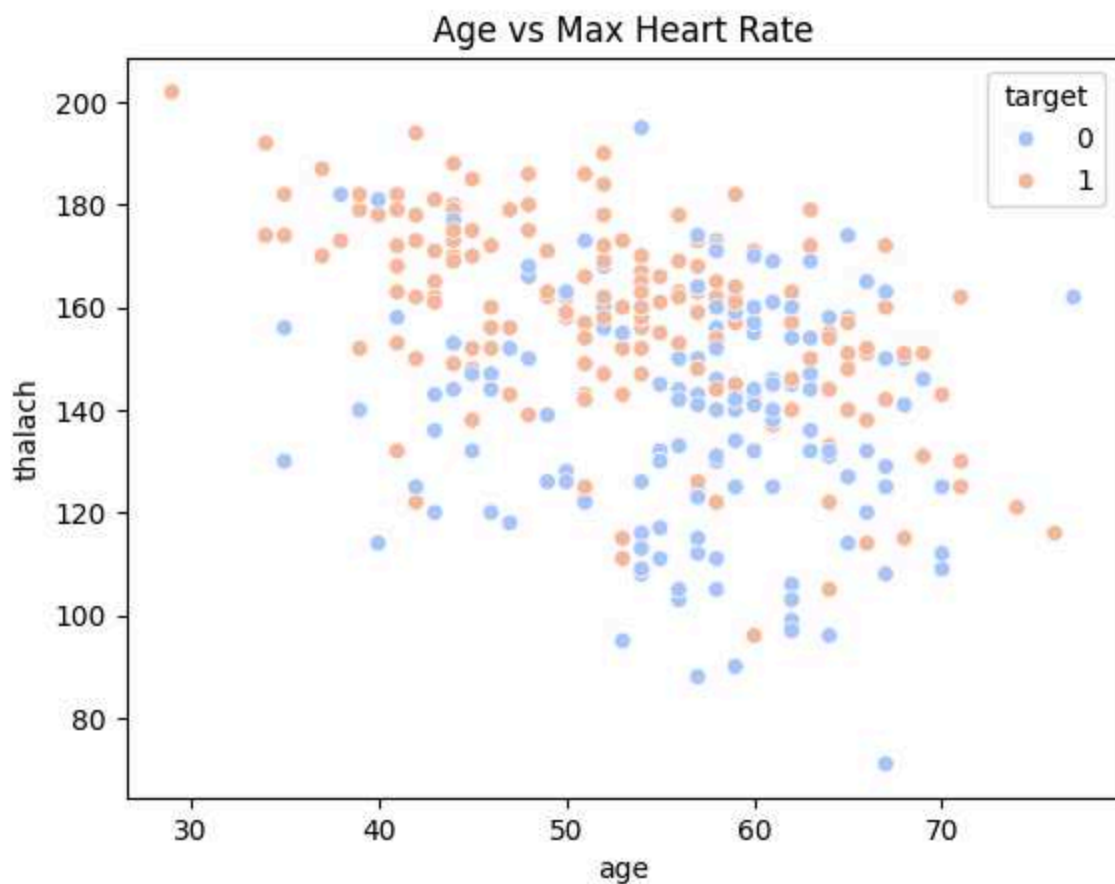
Target distribution

```
In [27]: sns.countplot(x='target', data=df)
plt.title("Distribution of Target Classes")
plt.show()
```



Age vs Max Heart Rate

```
In [28]: sns.scatterplot(x='age', y='thalach', hue='target', data=df, palette='coolwarm')
plt.title("Age vs Max Heart Rate")
plt.show()
```



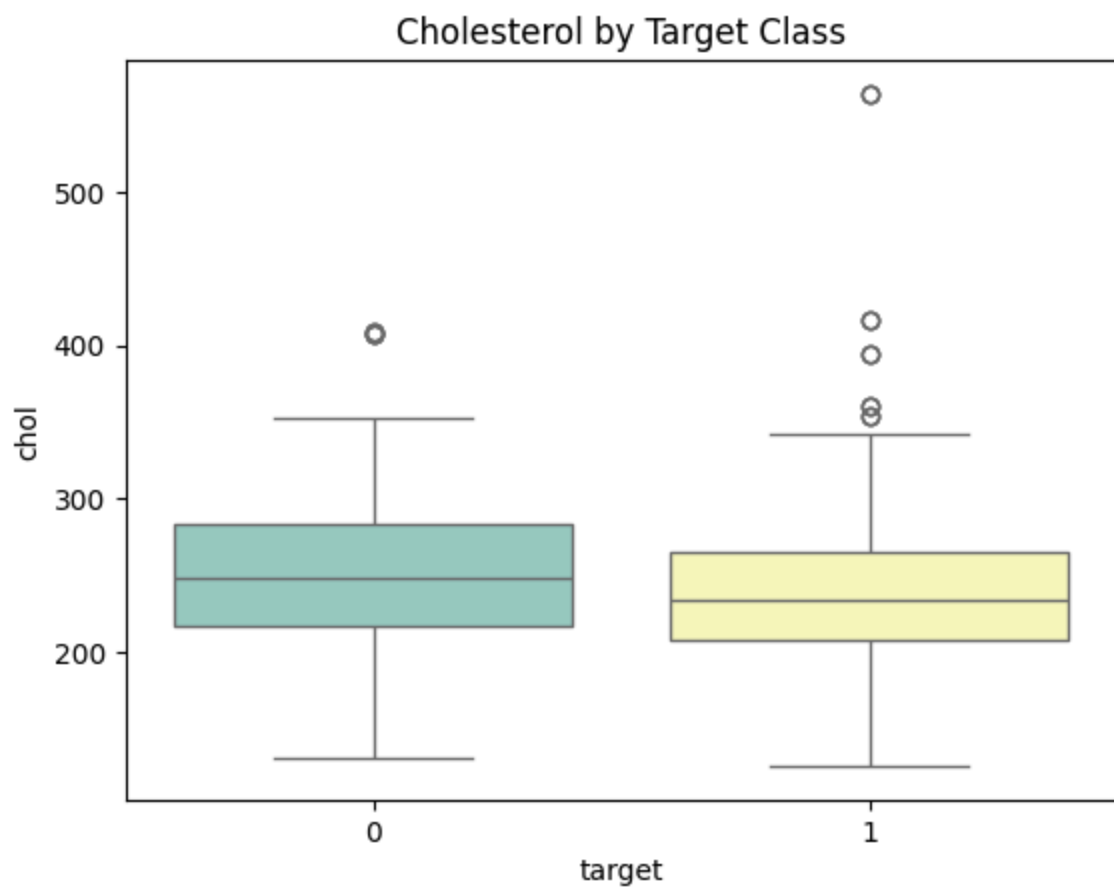
Cholesterol distribution by target

```
In [29]: sns.boxplot(x='target', y='chol', data=df, palette='Set3')
plt.title("Cholesterol by Target Class")
plt.show()
```

C:\Users\ABinAlhadi\AppData\Local\Temp\ipykernel_23468\2338091017.py:1: FutureWarning:

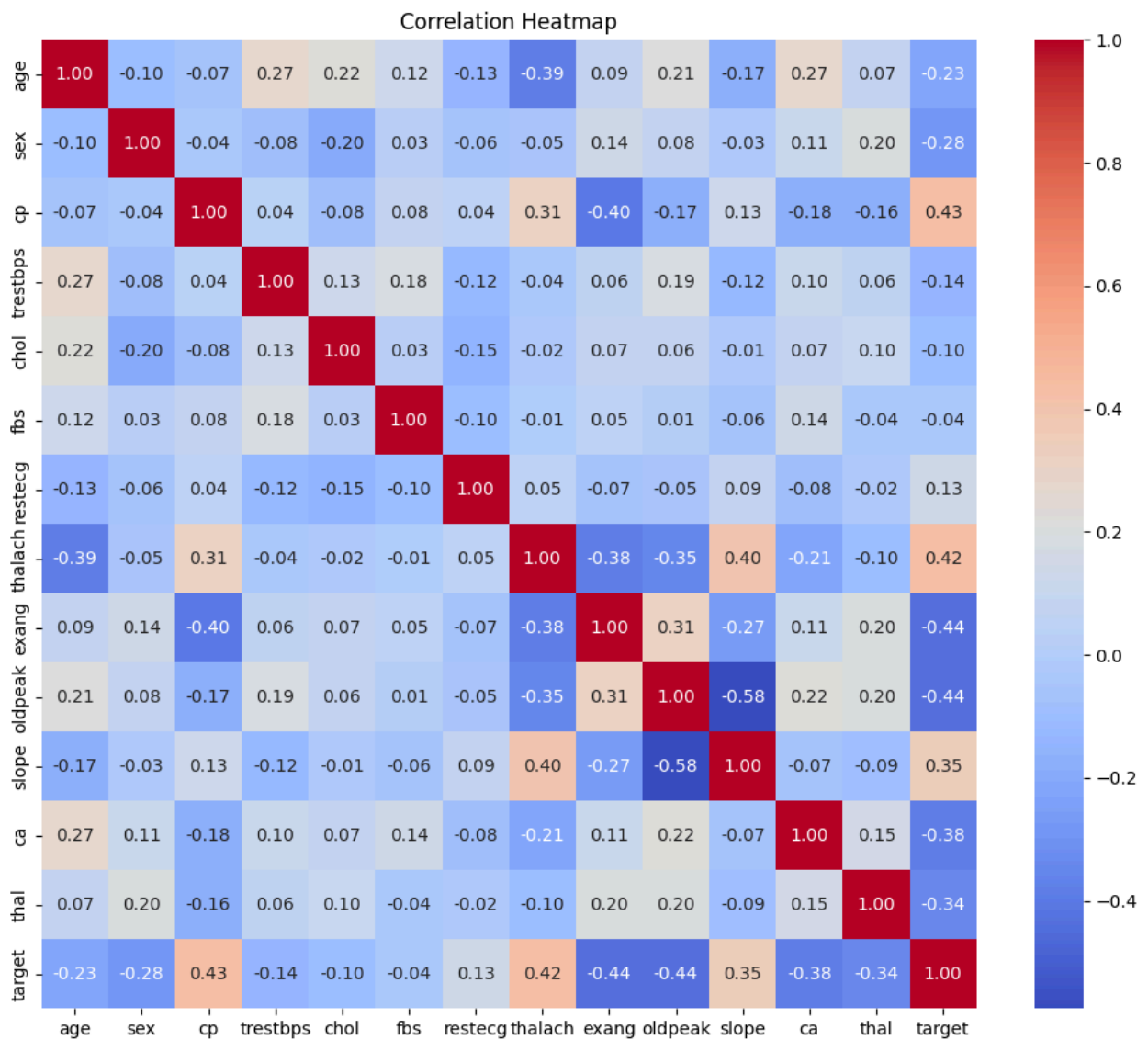
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.boxplot(x='target', y='chol', data=df, palette='Set3')
```



Correlation heatmap

```
In [30]: plt.figure(figsize=(12,10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



Evaluation Function

Function to evaluate models

```
In [34]: def evaluate_model(model_name, y_true, y_pred):
    print(f"\nModel: {model_name}")
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred, average='weighted'))
    print("Recall:", recall_score(y_true, y_pred, average='weighted'))
    print("F1 Score:", f1_score(y_true, y_pred, average='weighted'))
    print("\nClassification Report:\n", classification_report(y_true, y_pred))

    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(4,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"Confusion Matrix - {model_name}")
```

```
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

Random Forest with GridSearchCV

Random Forest with GridSearchCV

```
In [33]: rf = RandomForestClassifier(random_state=42)
param_grid_rf = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
grid_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='accuracy', n_jobs=-1)
grid_rf.fit(X_train, y_train)

best_rf = grid_rf.best_estimator_
y_pred = best_rf.predict(X_test)

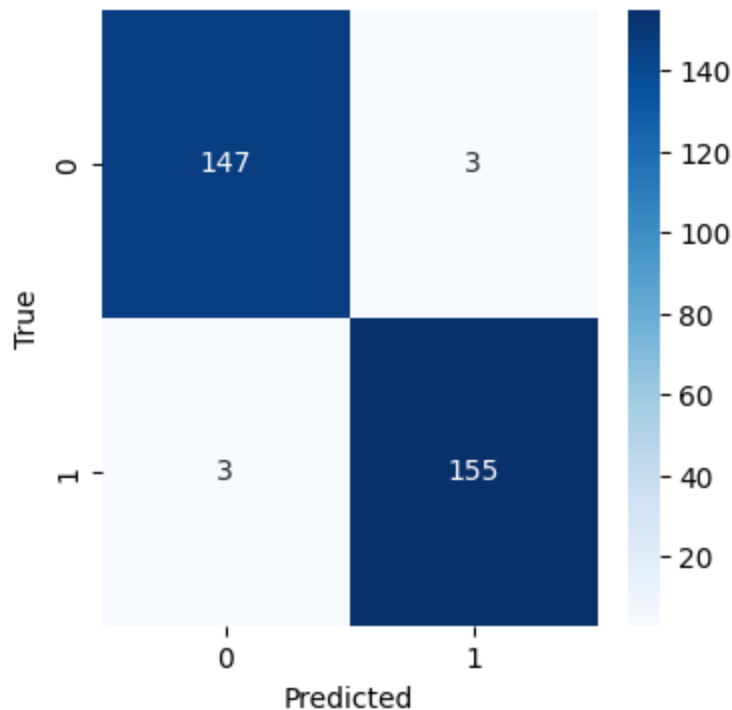
evaluate_model("Random Forest (GridSearch)", y_test, y_pred)
print("Best Params RF:", grid_rf.best_params_)
```

Model: Random Forest (GridSearch)
Accuracy: 0.9805194805194806
Precision: 0.9805194805194806
Recall: 0.9805194805194806
F1 Score: 0.9805194805194806

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.98	0.98	150
1	0.98	0.98	0.98	158
accuracy			0.98	308
macro avg	0.98	0.98	0.98	308
weighted avg	0.98	0.98	0.98	308

Confusion Matrix - Random Forest (GridSearch)



Best Params RF: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200}

Feature selection using Chi-Square

```
In [35]: selector_chi = SelectKBest(score_func=chi2, k=8)
X_chi = selector_chi.fit_transform(X, y)
selected_features_chi = X.columns[selector_chi.get_support()]
print("Chi-Square Selected Features:", list(selected_features_chi))
```

Chi-Square Selected Features: ['age', 'cp', 'trestbps', 'chol', 'thalach', 'exang', 'oldpeak', 'ca']

Feature selection using RFE

```
In [38]: scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
model_rfe = LogisticRegression(max_iter=1000, solver='liblinear')
rfe = RFE(model_rfe, n_features_to_select=8)
rfe.fit(X_scaled, y)
selected_features_rfe = X.columns[rfe.support_]
print("RFE Selected Features:", list(selected_features_rfe))
```

RFE Selected Features: ['sex', 'cp', 'trestbps', 'thalach', 'exang', 'oldpeak', 'ca', 'thal']

Re-apply Models on Selected Feature

Using Chi-Square features

```
In [39]: X_train_chi, X_test_chi, y_train, y_test = train_test_split(
          X[selected_features_chi], y, test_size=0.3, random_state=42, stratify=y
        )
        best_rf.fit(X_train_chi, y_train)
        y_pred = best_rf.predict(X_test_chi)
        evaluate_model("Random Forest (Chi-Square)", y_test, y_pred)

        # Using RFE features
        X_train_rfe, X_test_rfe, y_train, y_test = train_test_split(
          X[selected_features_rfe], y, test_size=0.3, random_state=42, stratify=y
        )
        best_rf.fit(X_train_rfe, y_train)
        y_pred = best_rf.predict(X_test_rfe)
        evaluate_model("Random Forest (RFE)", y_test, y_pred)
```

Model: Random Forest (Chi-Square)

Accuracy: 0.9707792207792207

Precision: 0.970972670593164

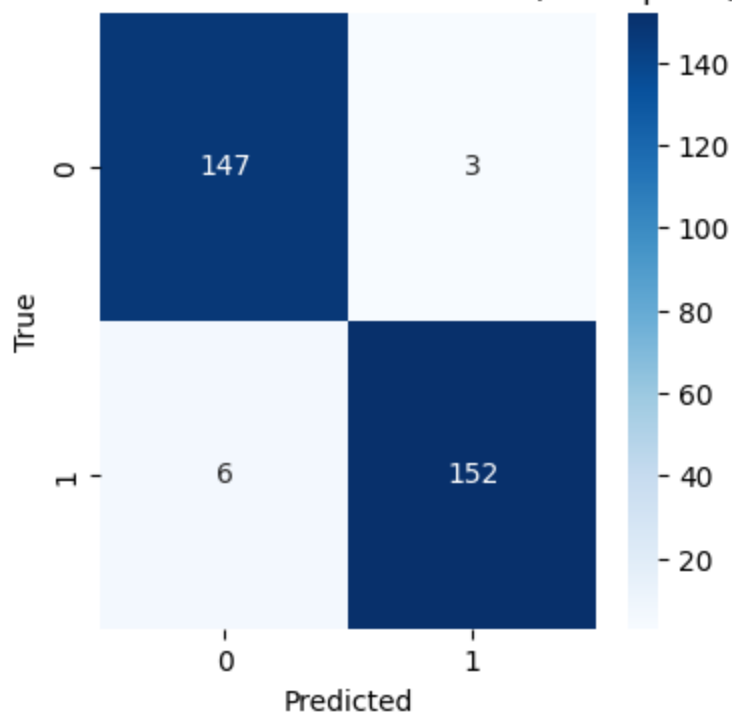
Recall: 0.9707792207792207

F1 Score: 0.9707838424189292

Classification Report:

	precision	recall	f1-score	support
0	0.96	0.98	0.97	150
1	0.98	0.96	0.97	158
accuracy			0.97	308
macro avg	0.97	0.97	0.97	308
weighted avg	0.97	0.97	0.97	308

Confusion Matrix - Random Forest (Chi-Square)

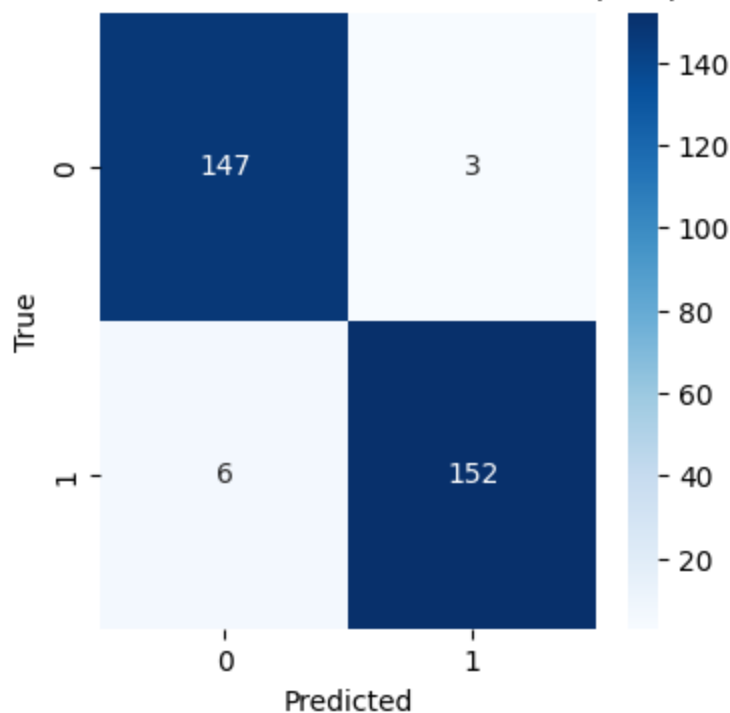


Model: Random Forest (RFE)
 Accuracy: 0.9707792207792207
 Precision: 0.970972670593164
 Recall: 0.9707792207792207
 F1 Score: 0.9707838424189292

Classification Report:

	precision	recall	f1-score	support
0	0.96	0.98	0.97	150
1	0.98	0.96	0.97	158
accuracy			0.97	308
macro avg	0.97	0.97	0.97	308
weighted avg	0.97	0.97	0.97	308

Confusion Matrix - Random Forest (RFE)



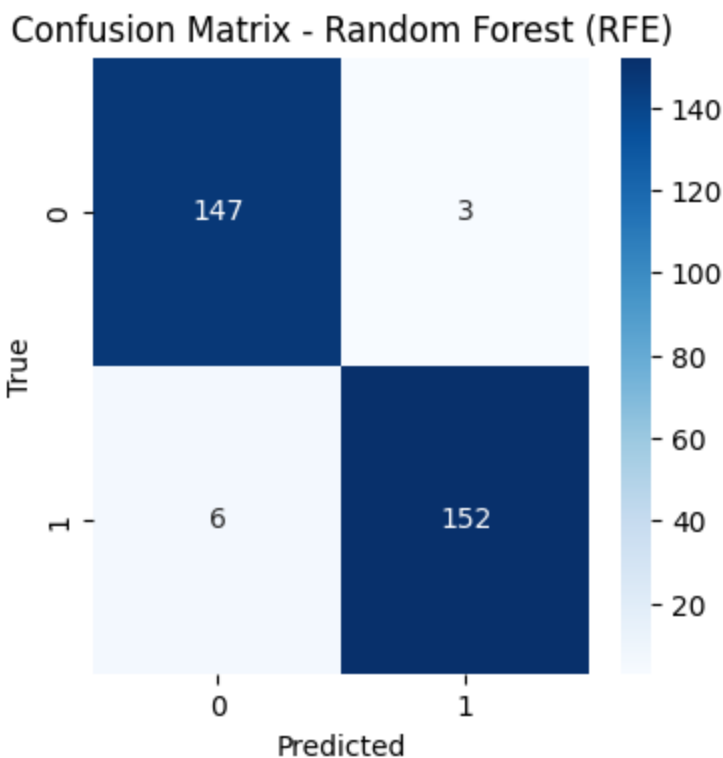
Using RFE features

```
In [40]: X_train_rfe, X_test_rfe, y_train, y_test = train_test_split(
          X[selected_features_rfe], y, test_size=0.3, random_state=42, stratify=y
        )
        best_rf.fit(X_train_rfe, y_train)
        y_pred = best_rf.predict(X_test_rfe)
        evaluate_model("Random Forest (RFE)", y_test, y_pred)
```

Model: Random Forest (RFE)
 Accuracy: 0.9707792207792207
 Precision: 0.970972670593164
 Recall: 0.9707792207792207
 F1 Score: 0.9707838424189292

Classification Report:

	precision	recall	f1-score	support
0	0.96	0.98	0.97	150
1	0.98	0.96	0.97	158
accuracy			0.97	308
macro avg	0.97	0.97	0.97	308
weighted avg	0.97	0.97	0.97	308



Model Comparison Table

Collect results into a comparison table

```
In [41]: results = []

results.append({
    'Model': 'Random Forest (GridSearch)',
    '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'),
    'F1 Score': f1_score(y_test, y_pred, average='weighted')
})

results_df = pd.DataFrame(results)
print(results_df)
```

	Model	Accuracy	Precision	Recall	F1 Score
0	Random Forest (GridSearch)	0.970779	0.970973	0.970779	0.970784

K-Nearest Neighbors with GridSearchCV

```
In [42]: knn = KNeighborsClassifier()
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],
```

```

    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

grid_knn = GridSearchCV(knn, param_grid_knn, cv=5, scoring='accuracy', n_jobs=
grid_knn.fit(X_train, y_train)

best_knn = grid_knn.best_estimator_
y_pred = best_knn.predict(X_test)

evaluate_model("K-Nearest Neighbors (GridSearch)", y_test, y_pred)
print("Best Params KNN:", grid_knn.best_params_)

```

Model: K-Nearest Neighbors (GridSearch)

Accuracy: 0.961038961038961

Precision: 0.9617783532257216

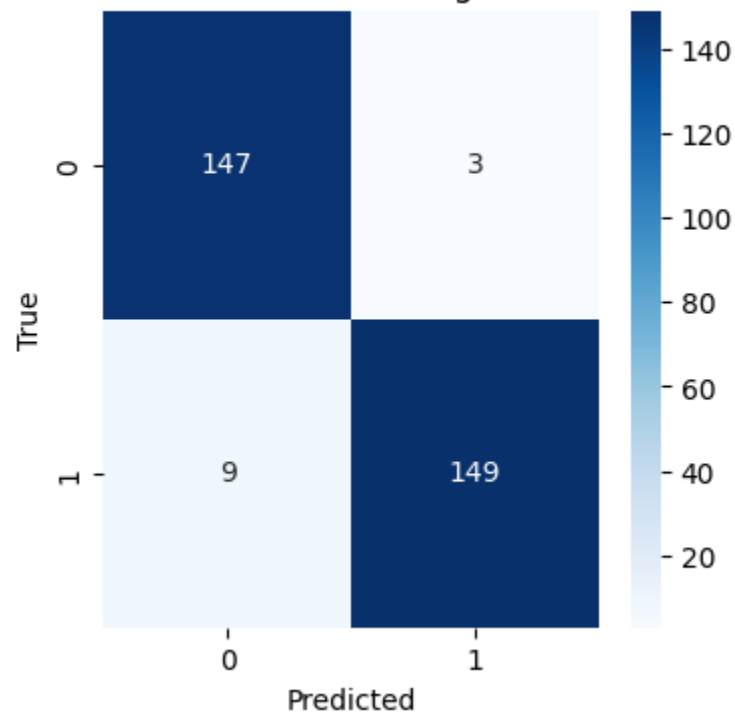
Recall: 0.961038961038961

F1 Score: 0.9610438896966411

Classification Report:

	precision	recall	f1-score	support
0	0.94	0.98	0.96	150
1	0.98	0.94	0.96	158
accuracy			0.96	308
macro avg	0.96	0.96	0.96	308
weighted avg	0.96	0.96	0.96	308

Confusion Matrix - K-Nearest Neighbors (GridSearch)



Best Params KNN: {'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'distance'}