

```
from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Step 0: Import Libraries

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
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.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import (
    confusion_matrix, accuracy_score, precision_score,
    recall_score, f1_score, classification_report
)

from sklearn.feature_selection import SelectKBest, chi2, RFECV

plt.style.use("seaborn-v0.8")
```

Step 1: Load Dataset & Initial Check

```
df = pd.read_csv('/content/drive/MyDrive/Project1/Project 2/Titanic_train.csv')

print("First 5 rows:")
display(df.head())
```

First 5 rows:												
PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cummings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
print("\nInfo:")
print(df.info())
```

Info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column      Non-Null Count  Dtype
---  --
0   PassengerId  891 non-null     int64
1   Survived     891 non-null     int64
2   Pclass       891 non-null     int64
3   Name         891 non-null     object
4   Sex          891 non-null     object
5   Age          714 non-null     float64
6   SibSp        891 non-null     int64
7   Parch        891 non-null     int64
8   Ticket       891 non-null     object
9   Fare         891 non-null     float64
10  Cabin        204 non-null     object
11  Embarked     889 non-null     object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None
```

```
print("\nDescriptive statistics:")
display(df.describe(include="all"))
```

Descriptive statistics:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	204	889
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147	3
top	NaN	NaN	NaN	Dooley, Mr. Patrick	male	NaN	NaN	NaN	347082	NaN	G6	S
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4	644
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN	32.204208	NaN	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN	49.693429	NaN	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN	NaN

```
print("\nMissing values per column:")
print(df.isnull().sum())
```

Missing values per column:

```
PassengerId    0
Survived        0
Pclass          0
Name            0
Sex             0
Age            177
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin          687
Embarked        2
dtype: int64
```

```
df_model = df.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin'])
print("Columns in df_model:", df_model.columns.tolist())
print("\nMissing values before handling:")
print(df_model.isnull().sum())
```

Columns in df_model: ['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']

Missing values before handling:

```
Survived    0
Pclass      0
Sex         0
Age        177
SibSp       0
Parch       0
Ticket      0
Fare        0
Cabin      687
Embarked    2
dtype: int64
```

```
# ملء القيم المفقودة بمتوسط العمر
df_model['Age'].fillna(df_model['Age'].mean(), inplace=True)

# ملء القيم المفقودة بمتوسط الدرجة
df_model['Embarked'].fillna(df_model['Embarked'].mode()[0], inplace=True)
```

ملء القيم المفقودة بمتوسط العمر

```
df_model['Age'].fillna(df_model['Age'].mean(), inplace=True)
```

ملء القيم المفقودة بمتوسط الدرجة

```
df_model['Embarked'].fillna(df_model['Embarked'].mode()[0], inplace=True)
```

```
print("\nMissing values after handling:")
print(df_model.isnull().sum())
```

Missing values after handling:

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

```
display(df_model.head())
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

Step 3: Encode Categorical Columns

```
cat_cols = df_model.select_dtypes(include=['object', 'category']).columns
print("Categorical columns:", cat_cols.tolist())
```

Categorical columns: ['Sex', 'Embarked']

```
label_encoders = {}

for col in cat_cols:
    le = LabelEncoder()
    df_model[col] = le.fit_transform(df_model[col])
    label_encoders[col] = le
```

```
print("\nAfter encoding:")
print(df_model.info())
display(df_model.head())
```

After encoding:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
#   Column      Non-Null Count  Dtype
---  --
0   Survived     891 non-null     int64
1   Pclass       891 non-null     int64
2   Sex          891 non-null     int64
3   Age          891 non-null     float64
4   SibSp        891 non-null     int64
5   Parch        891 non-null     int64
6   Fare         891 non-null     float64
7   Embarked     891 non-null     int64
dtypes: float64(2), int64(6)
memory usage: 55.8 KB
None
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	22.0	1	0	7.2500	2
1	1	1	0	38.0	1	0	71.2833	0
2	1	3	0	26.0	0	0	7.9250	2
3	1	1	0	35.0	1	0	53.1000	2
4	0	3	1	35.0	0	0	8.0500	2

Step 4: Define Target & Features + Train/Test Split + Scaling

```
target_column = "Survived"
```

```
feature_cols = df_model.drop(columns=[target_column]).columns
print("Feature columns:", feature_cols.tolist())

Feature columns: ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']

X_all = df_model[feature_cols].values
y_all = df_model[target_column].values

print("Shape of X_all:", X_all.shape)
print("Shape of y_all:", y_all.shape)

Shape of X_all: (891, 7)
Shape of y_all: (891,)

X_train_raw, X_test_raw, y_train, y_test = train_test_split(
    X_all, y_all,
    test_size=0.30,
    random_state=42,
    stratify=y_all
)

print("Train raw shape:", X_train_raw.shape)
print("Test raw shape:", X_test_raw.shape)

Train raw shape: (623, 7)
Test raw shape: (268, 7)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train_raw)
X_test = scaler.transform(X_test_raw)

print("Train scaled shape:", X_train.shape)
print("Test scaled shape:", X_test.shape)

Train scaled shape: (623, 7)
Test scaled shape: (268, 7)
```

Step 5: Evaluation Function

```
def evaluate_model(model_name, y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='binary')
    recall = recall_score(y_true, y_pred, average='binary')
    f1 = f1_score(y_true, y_pred, average='binary')
    cm = confusion_matrix(y_true, y_pred)
    report = classification_report(y_true, y_pred)

    # Confusion Matrix Plot
    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 {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

    print(f'=== {model_name} ===')
    print(f'Accuracy : {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall   : {recall:.4f}')
    print(f'F1 Score  : {f1:.4f}')
    print("\nClassification Report:\n", report)

    return {
        "Model Name": model_name,
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1 Score": f1,
        "Classification Report": report
    }
```

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

evaluate_model("Logistic Regression", y_test, y_pred)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)

evaluate_model("Logistic Regression", y_test, y_pred)
```

Confusion Matrix for Logistic Regression

True

0	140	25
1	29	74

Predicted

0 1

=== Logistic Regression ===

Accuracy : 0.7985

Precision: 0.7475

Recall : 0.7184

F1 Score : 0.7327

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.85	0.84	165
1	0.75	0.72	0.73	103
accuracy			0.80	268
macro avg	0.79	0.78	0.79	268
weighted avg	0.80	0.80	0.80	268

Confusion Matrix for Logistic Regression

True

0	140	25
1	29	74

Predicted

0 1

=== Logistic Regression ===

Accuracy : 0.7985

Precision: 0.7475

Recall : 0.7184

F1 Score : 0.7327

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.85	0.84	165
1	0.75	0.72	0.73	103
accuracy			0.80	268
macro avg	0.79	0.78	0.79	268
weighted avg	0.80	0.80	0.80	268

{'Model Name': 'Logistic Regression',
'Accuracy': 0.798507620865071,
'Precision': 0.7474747474747475,
'Recall': 0.71846601947476,
'F1 Score': 0.732673267327327,
'Classification Report':
precision recall f1-score support\n\n 0 0.83 0.85 0.84 165\n 1 0.75 0.72 0.73 103\n\n accuracy 0.80 268\n macro avg 0.79 0.78 0.79 268\n weighted avg 0.80 0.80 0.80 268\n'}

Step 6: Baseline Models (No Grid Search)

```
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "SVM": SVC(random_state=42),
    "Gaussian NB": GaussianNB(),
    "KNN": KNeighborsClassifier()
}
```

```
baseline_results = []
```

```
for name, model in models.items():
    print(f"Training {name}")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    metrics = evaluate_model(name, y_test, y_pred)
    baseline_results.append(metrics)
```

```
=====
Training Logistic Regression

Confusion Matrix for Logistic Regression

True
0      140      25
1       29      74

Predicted
0      1
1
```

=== Logistic Regression ===
Accuracy : 0.7985
Precision: 0.7475
Recall : 0.7184
F1 Score : 0.7327

	precision	recall	f1-score	support
0	0.83	0.85	0.84	165
1	0.75	0.72	0.73	103
accuracy			0.80	268
macro avg	0.79	0.78	0.79	268
weighted avg	0.80	0.80	0.80	268

```
=====
Training Decision Tree

Confusion Matrix for Decision Tree

True
0      138      27
1       36      67

Predicted
0      1
1
```

== Decision Tree ==
Accuracy : 0.7649
Precision: 0.7128
Recall : 0.6585
F1 Score : 0.6802

	precision	recall	f1-score	support
0	0.79	0.84	0.81	165
1	0.71	0.65	0.68	103
accuracy			0.76	268
macro avg	0.75	0.74	0.75	268
weighted avg	0.76	0.76	0.76	268

```
=====
Training Random Forest

Confusion Matrix for Random Forest

True
0      146      19
1       36      67

Predicted
0      1
1
```

=== Random Forest ===
Accuracy : 0.7948
Precision: 0.7731
Recall : 0.6585
F1 Score : 0.7090

	precision	recall	f1-score	support
0	0.80	0.88	0.84	165
1	0.78	0.65	0.71	103
accuracy			0.79	268
macro avg	0.79	0.77	0.78	268
weighted avg	0.79	0.79	0.79	268

```
=====
Training SVM

Confusion Matrix for SVM

True
0      145      20
1       33      70

Predicted
0      1
1
```

=== SVM ===
Accuracy : 0.8022
Precision: 0.7778
Recall : 0.6796
F1 Score : 0.7254

	precision	recall	f1-score	support
0	0.81	0.88	0.85	165
1	0.78	0.68	0.73	103
accuracy			0.80	268
macro avg	0.80	0.78	0.79	268
weighted avg	0.80	0.80	0.80	268

```
=====
Training Gaussian NB

Confusion Matrix for Gaussian NB

True
0      136      29
1       34      69

Predicted
0      1
1
```

Step 7: Grid Search (RandomForest, SVM, KNN)

```
# ----- Random Forest -----
grid_results = {}
```

```
print("\n===== Grid Search: Random Forest =====")
rf = RandomForestClassifier(random_state=42)
```

```
Recall : 0.7379
F1 Score : 0.7308
===== Grid Search: Random Forest =====
```

Classification Report:

```
rf_param_grid = {
    "n_estimators": [100, 200],
    "max_depth": [None, 5, 10],
    "min_samples_split": [2, 5],
    "min_samples_leaf": [1, 2]
}
```

```
weighted avg
```

```
rf_grid = GridSearchCV(
    estimator=rf,
    param_grid=rf_param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)
```

```
rf_grid.fit(X_train, y_train)
best_rf = rf_grid.best_estimator_
print("Best RF params:", rf_grid.best_params_)
```

```
Best RF params: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
```

```
y_pred_rf = best_rf.predict(X_test)
grid_results["Random Forest (Grid)"] = evaluate_model("Random Forest (Grid)", y_test, y_pred_rf)
```

```
Confusion Matrix for Random Forest (Grid)

True
0      148      17
1       34      69

Predicted
0      1
1
```

=== Random Forest (Grid) ===
Accuracy : 0.8097
Precision: 0.8023
Recall : 0.6699
F1 Score : 0.7302

	precision	recall	f1-score	support
0	0.81	0.90	0.85	165
1	0.80	0.67	0.73	103
accuracy			0.81	268
macro avg	0.81	0.78	0.79	268
weighted avg	0.81	0.81	0.81	268

```
# ----- SVM -----
print("\n===== Grid Search: SVM =====")
svm = SVC(random_state=42)
```

===== Grid Search: SVM =====

```
svm_param_grid = {
    "C": [0.1, 1, 10],
    "kernel": ["linear", "rbf"],
    "gamma": ["scale", "auto"]
}
```

```
svm_grid = GridSearchCV(
    estimator=svm,
```

```
param_grid=svm_param_grid,
cv=5,
scoring="accuracy",
n_jobs=-1
)

svm_grid.fit(X_train, y_train)
best_svm = svm_grid.best_estimator_
print("Best SVM params:", svm_grid.best_params_)

Best SVM params: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
```

```
svm_grid.fit(X_train, y_train)
best_svm = svm_grid.best_estimator_
print("Best SVM params:", svm_grid.best_params_)

y_pred_svm = best_svm.predict(X_test)
grid_results["SVM (Grid)"] = evaluate_model("SVM (Grid)", y_test, y_pred_svm)

Best SVM params: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
```

Confusion Matrix for SVM (Grid)

	0	1
0	147	18
1	33	70

=== SVM (Grid) ===
Accuracy : 0.8897
Precision: 0.7955
Recall : 0.6796
F1 Score : 0.7330

Classification Report:

	precision	recall	f1-score	support
0	0.82	0.89	0.85	165
1	0.80	0.68	0.73	103
accuracy			0.81	268
macro avg	0.81	0.79	0.79	268
weighted avg	0.81	0.81	0.81	268

```
# ----- KNN -----
print("\n===== Grid Search: KNN =====")
knn = KNeighborsClassifier()

===== Grid Search: KNN =====
```

```
knn_param_grid = {
    "n_neighbors": [3, 5, 7, 9],
    "weights": ["uniform", "distance"],
    "metric": ["euclidean", "manhattan"]
}
```

```
knn_grid = GridSearchCV(
    estimator=knn,
    param_grid=knn_param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)
```

```
knn_grid.fit(X_train, y_train)
best_knn = knn_grid.best_estimator_
print("Best KNN params:", knn_grid.best_params_)

Best KNN params: {'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'uniform'}
```

```
y_pred_knn = best_knn.predict(X_test)
grid_results["KNN (Grid)"] = evaluate_model("KNN (Grid)", y_test, y_pred_knn)
```

Confusion Matrix for KNN (Grid)

	0	1
0	145	20
1	33	70

=== KNN (Grid) ===
Accuracy : 0.8822
Precision: 0.7778
Recall : 0.6796
F1 Score : 0.7254

Classification Report:

	precision	recall	f1-score	support
0	0.81	0.88	0.85	165
1	0.78	0.68	0.73	103
accuracy			0.80	268
macro avg	0.80	0.78	0.79	268
weighted avg	0.80	0.80	0.80	268

Step 8: Feature Selection - Method 1 (SelectKBest + Chi2)

```
# عدد المميزات المختارة
k = 5
selector_kbest = SelectKBest(score_func=chi2, k=k)
selector_kbest.fit(X_train_raw, y_train)
```

SelectKBest

SelectKBest(k=5, score_func=<function chi2 at 0x799bba2d8a40>)

```
selected_mask_kbest = selector_kbest.get_support()
selected_features_kbest = feature_cols[selected_mask_kbest]
scores_kbest = selector_kbest.scores_[selected_mask_kbest]
```

```
kbest_df = pd.DataFrame({
    "feature": selected_features_kbest,
    "score": scores_kbest
}).sort_values(by="score", ascending=False)
```

```
print("Top features according to SelectKBest + Chi2:")
display(kbest_df)
```

Top features according to SelectKBest + Chi2:

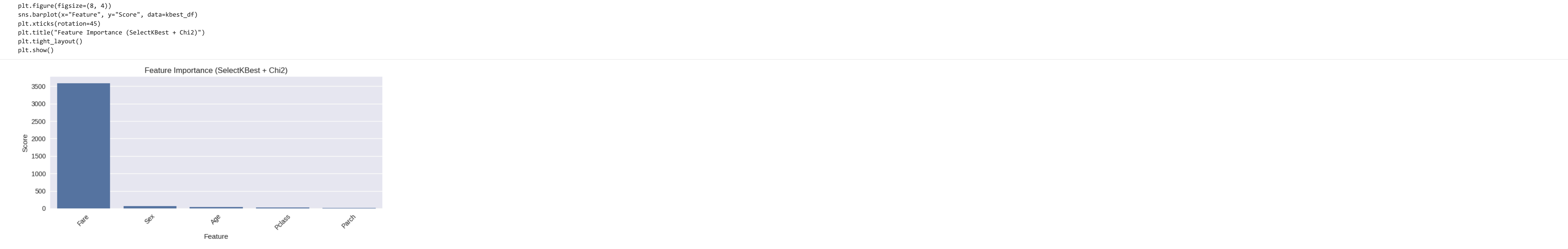
	Feature	Score
4	Fare	3587.277799
1	Sex	66.254123
2	Age	33.748965
0	Pclass	20.819285
3	Parch	12.023104

Next steps:

[Generate code with kbest_df](#)

[New interactive sheet](#)

```
# استخدام نفس القاع على البيانات الجديدة
X_train_kbest = X_train[:, selected_mask_kbest]
X_test_kbest = X_test[:, selected_mask_kbest]
```



Step 9: Feature Selection - Method 2 (RFE + RandomForest)

```
rf_for_rfe = RandomForestClassifier(n_estimators=200, random_state=42)
```

```
n_features_rfe = 5
rfe = RFE(estimator=rf_for_rfe, n_features_to_select=n_features_rfe)
rfe.fit(X_train_raw, y_train)
```

RFE

estimator:
RandomForestClassifier
• RandomForestClassifier

```
selected_mask_rfe = rfe.get_support()
selected_features_rfe = feature_cols[selected_mask_rfe]
ranking_rfe = rfe.ranking_[selected_mask_rfe]
```

```
rfe_df = pd.DataFrame({
    "feature": selected_features_rfe,
    "RFE_Ranking": ranking_rfe
}).sort_values(by="RFE_Ranking", ascending=True)
```

```
print("Selected features by RFE + RandomForest:")
display(rfe_df)
```

Selected features by RFE + RandomForest:

	Feature	RFE_Ranking
0	Pclass	1
1	Sex	1
2	Age	1
3	SibSp	1
4	Fare	1

Next steps:

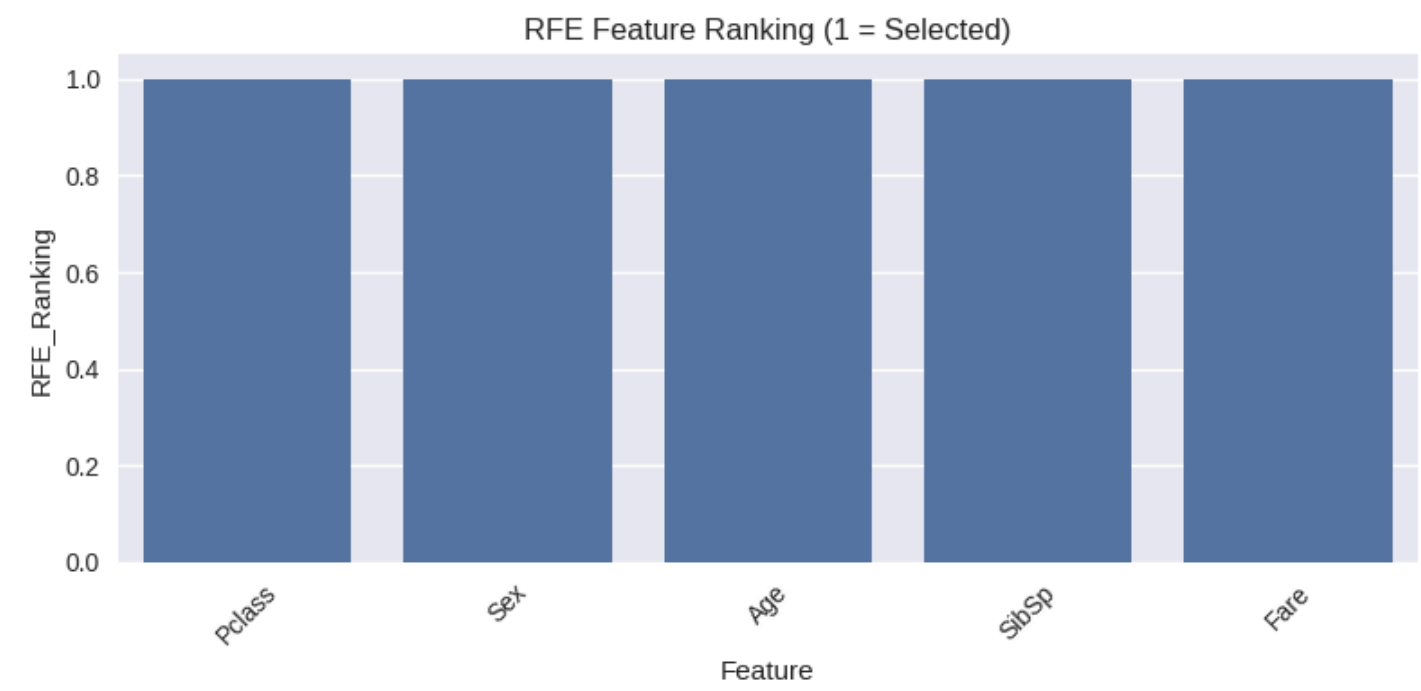
[Generate code with rfe_df](#)

[New interactive sheet](#)

```
# استخدام نفس القاع على البيانات الجديدة
X_train_rfe = X_train[:, selected_mask_rfe]
```

X_test_rfe = X_test[:, selected_mask_rfe]

```
plt.figure(figsize=(8, 4))
sns.barplot(x="Feature", y="RFE_Ranking", data=rfe_df)
plt.xticks(rotation=45)
plt.title("RFE Feature Ranking (1 = Selected)")
plt.tight_layout()
plt.show()
```



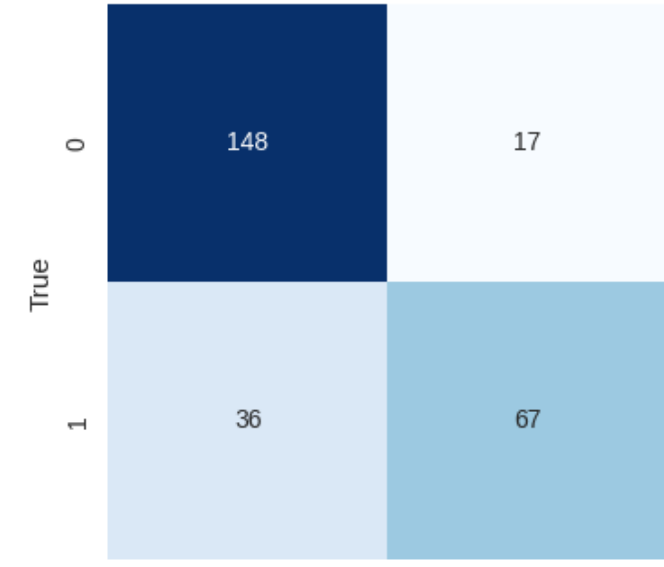
Step 10: Models on Selected Features (KBest & RFE)

```
print("=== Models on SelectKBest features ===")
models_selected_kbest = {
    "Random Forest (Grid) - KBest": best_rf,
    "SVM (Grid) - KBest": best_svm
}
```

=== Models on SelectKBest features ===

```
for name, model in models_selected_kbest.items():
    print("\n", "-"*60)
    print(name)
    model.fit(X_train_kbest, y_train)
    y_pred = model.predict(X_test_kbest)
    evaluate_model(name, y_test, y_pred)
```

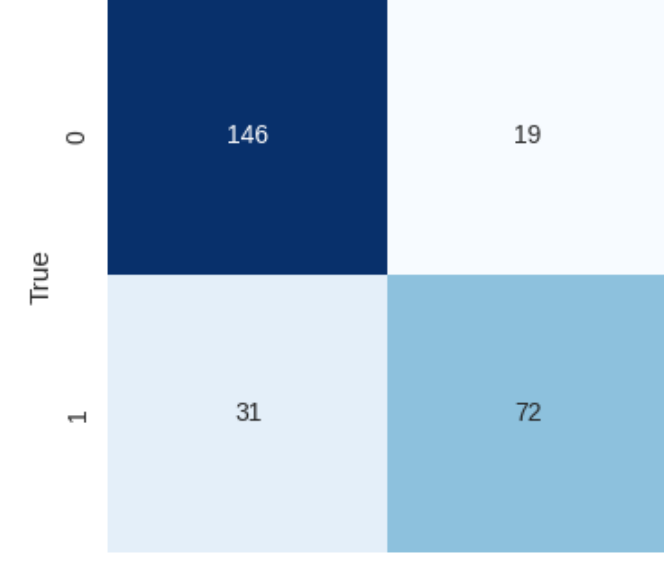
Random Forest (Grid) - KBest
Confusion Matrix for Random Forest (Grid) - KBest



=== Random Forest (Grid) - KBest ===
Accuracy : 0.8822
Precision: 0.7976
Recall : 0.6505
F1 Score : 0.7166

Classification Report:				
	precision	recall	f1-score	support
0	0.80	0.90	0.85	165
1	0.80	0.65	0.72	103
accuracy			0.80	268
macro avg	0.80	0.77	0.78	268
weighted avg	0.80	0.80	0.80	268

SVM (Grid) - KBest
Confusion Matrix for SVM (Grid) - KBest



=== SVM (Grid) - KBest ===
Accuracy : 0.8134
Precision: 0.7912
Recall : 0.6990
F1 Score : 0.7423

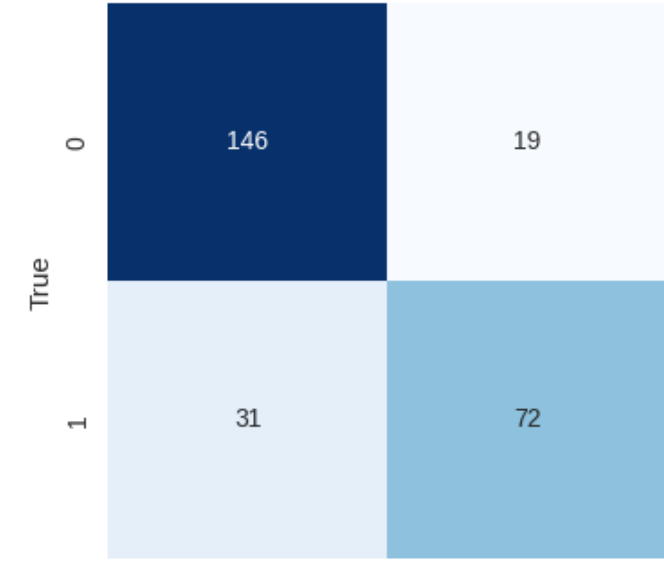
Classification Report:				
	precision	recall	f1-score	support
0	0.82	0.88	0.85	165
1	0.79	0.70	0.74	103
accuracy			0.81	268
macro avg	0.81	0.79	0.80	268
weighted avg	0.81	0.81	0.81	268

```
print("\n=== Models on RFE features ===")
models_selected_rfe = {
    "Random Forest (Grid) - RFE": best_rf,
    "SVM (Grid) - RFE": best_svm
}
```

=== Models on RFE features ===

```
for name, model in models_selected_rfe.items():
    print("\n", "-"*60)
    print(name)
    model.fit(X_train_rfe, y_train)
    y_pred = model.predict(X_test_rfe)
    evaluate_model(name, y_test, y_pred)
```

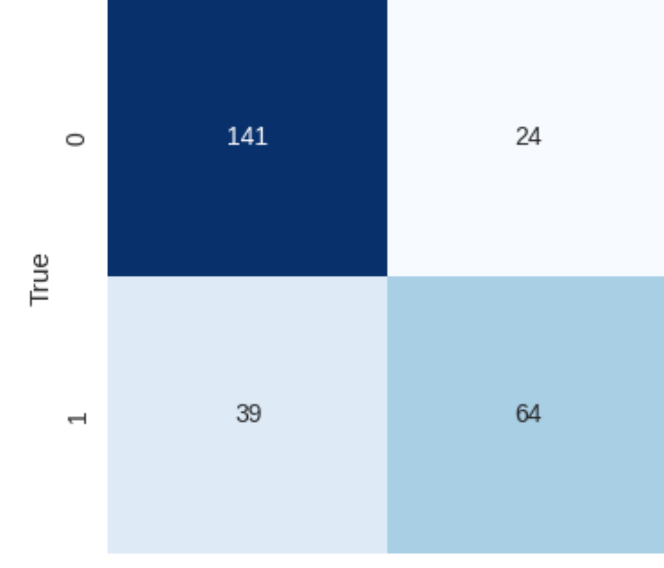
Random Forest (Grid) - RFE
Confusion Matrix for Random Forest (Grid) - RFE



=== Random Forest (Grid) - RFE ===
Accuracy : 0.8134
Precision: 0.7912
Recall : 0.6990
F1 Score : 0.7423

Classification Report:				
	precision	recall	f1-score	support
0	0.82	0.88	0.85	165
1	0.79	0.70	0.74	103
accuracy			0.81	268
macro avg	0.81	0.79	0.80	268
weighted avg	0.81	0.81	0.81	268

SVM (Grid) - RFE
Confusion Matrix for SVM (Grid) - RFE



=== SVM (Grid) - RFE ===
Accuracy : 0.7649
Precision: 0.7273
Recall : 0.6214
F1 Score : 0.6782

Classification Report:				
	precision	recall	f1-score	support
0	0.78	0.85	0.82	165
1	0.73	0.62	0.67	103
accuracy			0.76	268
macro avg	0.76	0.74	0.74	268
weighted avg	0.76	0.76	0.76	268