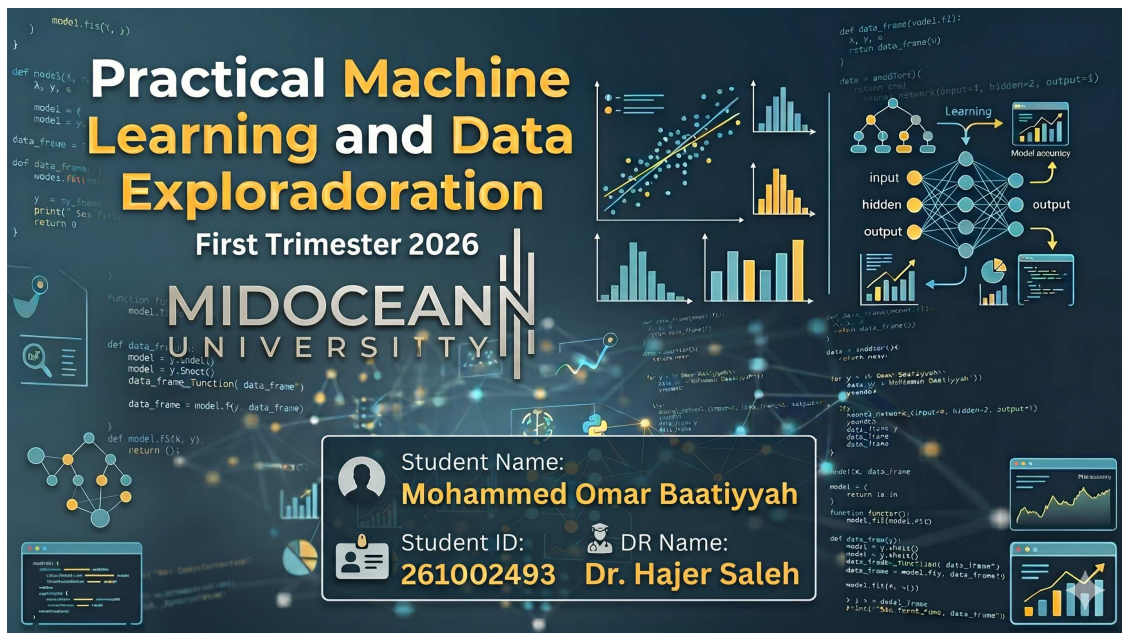


Project_2_2026

April 18, 2026

```
[1]: from IPython.display import Image, display
display(Image(filename='Project 2 image.png'))
```



```
[2]: import pandas as pd
import numpy as np
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.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.metrics import classification_report, accuracy_score, \
    confusion_matrix
```

```

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import GridSearchCV
import json
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import RFE

```

```
[3]: df = pd.read_csv('kidney_disease.csv')
```

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

```
[4]:
```

	age	bp	sg	al	su	bgr \
count	389.000000	386.000000	352.000000	353.000000	350.000000	354.000000
mean	51.395887	76.502591	1.017429	1.014164	0.451429	147.943503
std	17.168437	13.692210	0.005711	1.353582	1.100500	79.469325
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000
50%	54.000000	80.000000	1.020000	0.000000	0.000000	121.000000
75%	64.000000	80.000000	1.020000	2.000000	0.000000	162.750000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000

	bu	sc	sod	pot	hemo
count	379.000000	381.000000	311.000000	310.000000	347.000000
mean	57.512401	3.075197	137.541801	4.628065	12.534582
std	50.592153	5.755759	10.431645	3.204167	2.912820
min	1.500000	0.400000	4.500000	2.500000	3.100000
25%	27.000000	0.900000	135.000000	3.800000	10.350000
50%	42.000000	1.300000	138.000000	4.400000	12.700000
75%	66.000000	2.800000	142.000000	4.900000	15.000000
max	391.000000	76.000000	163.000000	47.000000	17.800000

```
[5]: df.dtypes
```

```
[5]: age          float64
bp            float64
sg            float64
al            float64
su            float64
rbc           str
pc            str
pcc           str
ba            str
bgr          float64
bu            float64
sc            float64
```

```
sod          float64
pot          float64
hemo         float64
pcv          str
wc           str
rc           str
htn          str
dm           str
cad          str
appet        str
pe           str
ane          str
classification str
dtype: object
```

1. Handle missing values appropriately to prepare the datasets for applying machine learning models

```
[6]: from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
for column in df.select_dtypes(include=['object']).columns:
    df[column] = label_encoder.fit_transform(df[column])
df.head()
```

C:\Users\mob85\AppData\Local\Temp\ipykernel_6052\3271453723.py:3:

Pandas4Warning: For backward compatibility, 'str' dtypes are included by select_dtypes when 'object' dtype is specified. This behavior is deprecated and will be removed in a future version. Explicitly pass 'str' to `include` to select them, or to `exclude` to remove them and silence this warning.

See https://pandas.pydata.org/docs/user_guide/migration-3-strings.html#string-migration-select-dtypes for details on how to write code that works with pandas 2 and 3.

```
for column in df.select_dtypes(include=['object']).columns:
```

```
[6]:   age    bp    sg    al    su  rbc  pc  pcc  ba    bgr  ...  pcv  wc  rc  \
0  48.0  80.0  1.020  1.0  0.0    2   1    0   0  121.0  ...   32  72  31
1   7.0  50.0  1.020  4.0  0.0    2   1    0   0    NaN  ...   26  56  46
2  62.0  80.0  1.010  2.0  3.0    1   1    0   0  423.0  ...   19  70  46
3  48.0  70.0  1.005  4.0  0.0    1   0    1   0  117.0  ...   20  62  18
4  51.0  80.0  1.010  2.0  0.0    1   1    0   0  106.0  ...   23  68  25

      htn  dm  cad  appet  pe  ane  classification
0     1   4   1      0  0   0
1     0   3   1      0  0   0
2     0   4   1      1  0   1
3     1   3   1      1  1   1
4     0   3   1      0  0   0
```

[5 rows x 25 columns]

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

```
[7]: age          9
bp          12
sg          46
al          45
su          48
rbc         0
pc          0
pcc         0
ba          0
bgr         44
bu          19
sc          17
sod         87
pot         88
hemo        51
pcv         0
wc          0
rc          0
htn         0
dm          0
cad         0
appet       0
pe          0
ane         0
classification 0
dtype: int64
```

```
[8]: for column in df.select_dtypes(include=['float64']).columns:
    mean_value = df[column].mean()
    df[column] = df[column].fillna(mean_value)

    for column in df.select_dtypes(include=['int64']).columns:
        mode_value = df[column].mode()[0]
        df[column] = df[column].fillna(mode_value)
```

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

```
[9]: age          0
bp          0
sg          0
al          0
su          0
rbc         0
```

```

pc          0
pcc         0
ba          0
bgr         0
bu          0
sc          0
sod         0
pot         0
hemo        0
pcv         0
wc          0
rc          0
htn         0
dm          0
cad         0
appet       0
pe          0
ane         0
classification 0
dtype: int64

```

```

[10]: df['classification'].value_counts()

X = df.drop(['classification'], axis=1)
y = df['classification']

```

```

[11]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
↳ random_state=42)

```

2-Apply machine learning models with a grid search optimizer to the cleaned datasets and evaluate the models' performance.

```

[12]: def evaluate_model(model_name, y_true, y_pred):
    print(f"--- Performance Evaluation: {model_name} ---")
    accuracy = accuracy_score(y_true, y_pred)
    print(f"Accuracy: {accuracy:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))
    return {"model": model_name, "accuracy": accuracy}

```

```

[13]: # Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)
param_grid_RF = {
    '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_RF,
    cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
best_RF = grid_search.best_estimator_
y_pred = best_RF.predict(X_test)
evaluation_results = evaluate_model('RandomForestClassifier', y_test, y_pred)

print("Best Parameters:", grid_search.best_params_)
best_rf = grid_search.best_estimator_
y_pred_rf = best_rf.predict(X_test)

print("=" * 20)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,6))
sns.heatmap(
    cm,
    annot=True,
    fmt='d',
    cmap='Blues',
    cbar=False,
    square=True
)
plt.title("Confusion Matrix for Random Forest", fontsize=14)
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.xticks([0.5, 1.5], ['0', '1'])
plt.yticks([0.5, 1.5], ['0', '1'], rotation=0)
plt.tight_layout()
plt.show()
print("=" * 20)
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

--- Performance Evaluation: RandomForestClassifier ---
Accuracy: 1.0000

Classification Report:

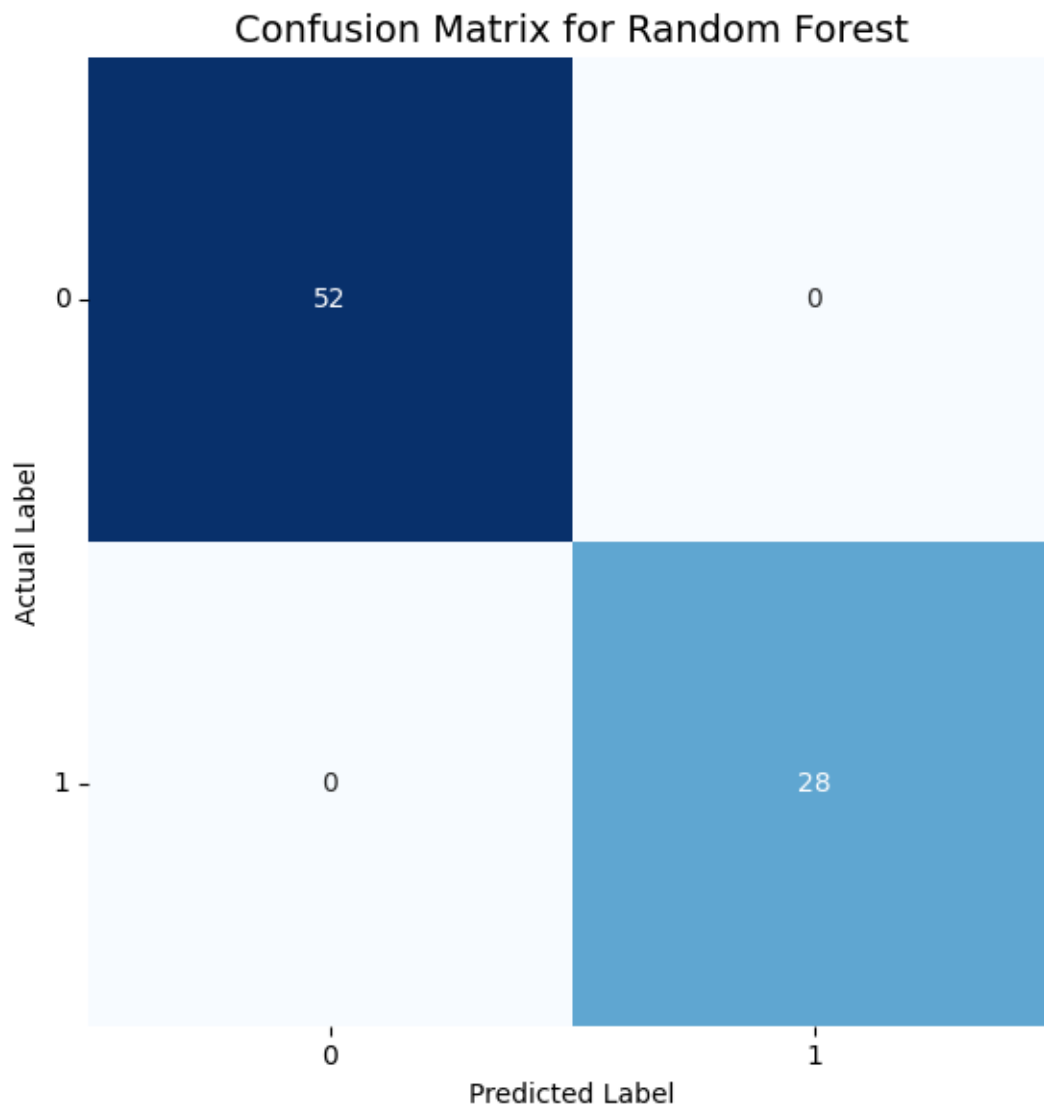
	precision	recall	f1-score	support
0	1.00	1.00	1.00	52
1	1.00	1.00	1.00	28
accuracy			1.00	80
macro avg	1.00	1.00	1.00	80
weighted avg	1.00	1.00	1.00	80

Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}

=====

Random Forest Accuracy: 1.0

	precision	recall	f1-score	support
0	1.00	1.00	1.00	52
1	1.00	1.00	1.00	28
accuracy			1.00	80
macro avg	1.00	1.00	1.00	80
weighted avg	1.00	1.00	1.00	80



=====

Best hyperparameters found by GridSearchCV:

```
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 10,
'n_estimators': 100}
```

```
[14]: # Logistic Regression model
LR_classifier = LogisticRegression(random_state=42)

param_grid_Lr = {
    'C': [0.01, 0.1, 1, 10],
    'solver': ['liblinear', 'lbfgs'],
    'max_iter': [1500, 2500, 3500],
}
grid_lr = GridSearchCV(LR_classifier, param_grid_Lr, cv=5)
grid_search = GridSearchCV(estimator=LR_classifier, param_grid=param_grid_Lr,
    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)
grid_lr.fit(X_train, y_train)
print("Best Parameters:", grid_lr.best_params_)
#best_lr = grid_lr.best_estimator_
y_pred_lr = best_LR.predict(X_test)
print("=" * 20)
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_lr))
print(classification_report(y_test, y_pred_lr))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,6))
sns.heatmap(
    cm,
    annot=True,
    fmt='d',
    cmap='Blues',
    cbar=False,
    square=True
)
plt.title("Confusion Matrix for LogisticRegression", fontsize=14)
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.xticks([0.5, 1.5], ['0', '1'])
plt.yticks([0.5, 1.5], ['0', '1'], rotation=0)
plt.tight_layout()
plt.show()
print("=" * 20)
```

```
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

--- Performance Evaluation: LogisticRegression ---
Accuracy: 0.9375

Classification Report:

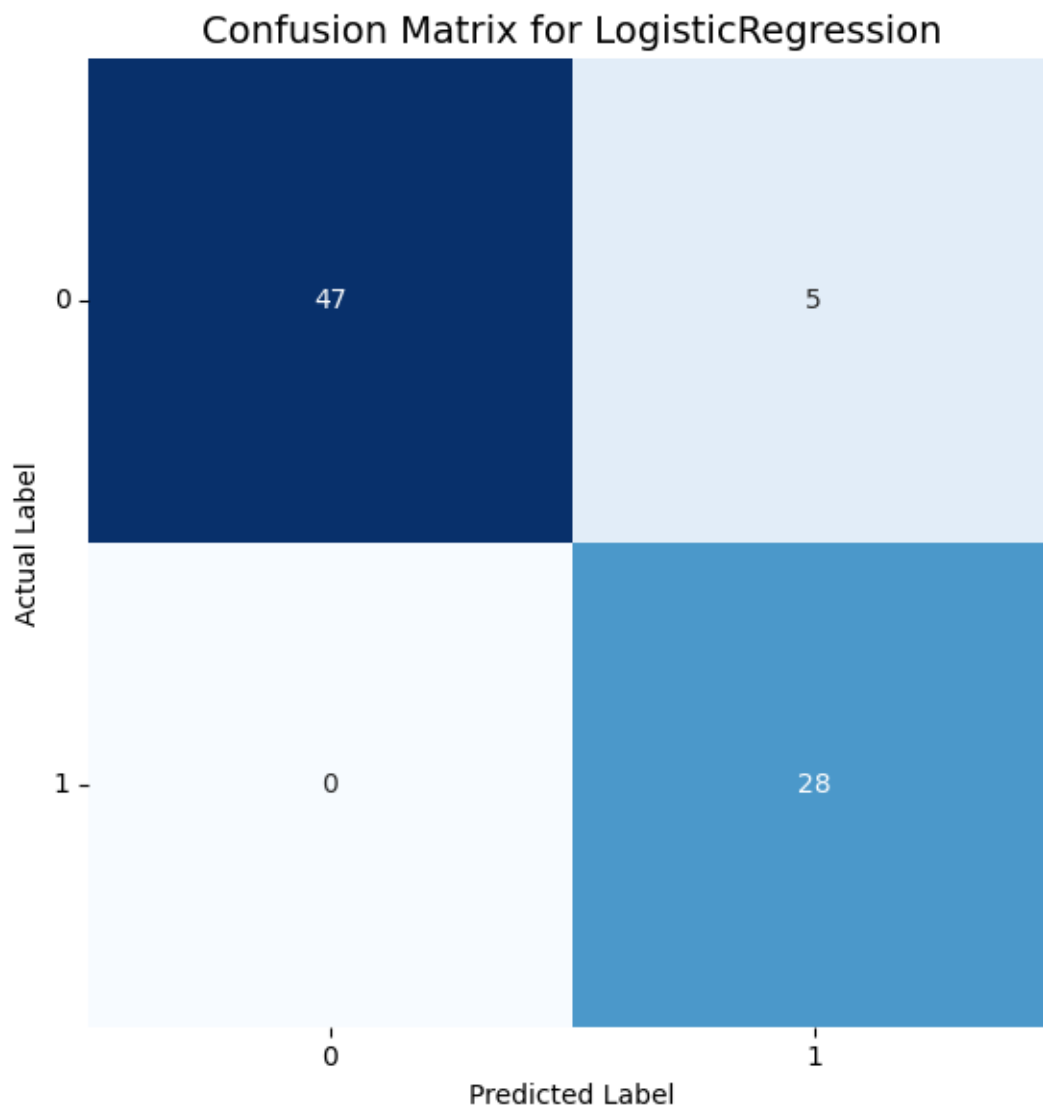
	precision	recall	f1-score	support
0	1.00	0.90	0.95	52
1	0.85	1.00	0.92	28
accuracy			0.94	80
macro avg	0.92	0.95	0.93	80
weighted avg	0.95	0.94	0.94	80

Best Parameters: {'C': 1, 'max_iter': 1500, 'solver': 'lbfgs'}

=====

Logistic Regression Accuracy: 0.9375

	precision	recall	f1-score	support
0	1.00	0.90	0.95	52
1	0.85	1.00	0.92	28
accuracy			0.94	80
macro avg	0.92	0.95	0.93	80
weighted avg	0.95	0.94	0.94	80



=====

Best hyperparameters found by GridSearchCV:
 {'C': 1, 'max_iter': 1500, 'solver': 'lbfgs'}

```
[15]: # Decision Tree model
DT_classifier = DecisionTreeClassifier(random_state=42)
param_grid_DT = {
    '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_classifier, param_grid=param_grid_DT,
    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)

print("Best Parameters:", grid_lr.best_params_)
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test, y_pred)
#best_lr = grid_lr.best_estimator_
y_pred_DT = best_DT.predict(X_test)
print("=" * 20)
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_DT))
print(classification_report(y_test, y_pred_DT))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,6))
sns.heatmap(
    cm,
    annot=True,
    fmt='d',
    cmap='Blues',
    cbar=False,
    square=True
)
plt.title("Confusion Matrix for Decision Tree", fontsize=14)
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.xticks([0.5, 1.5], ['0', '1'])
plt.yticks([0.5, 1.5], ['0', '1'], rotation=0)
plt.tight_layout()
plt.show()
print("=" * 20)
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Best Parameters: {'C': 1, 'max_iter': 1500, 'solver': 'lbfgs'}

--- Performance Evaluation: DecisionTreeClassifier ---

Accuracy: 0.9750

Classification Report:

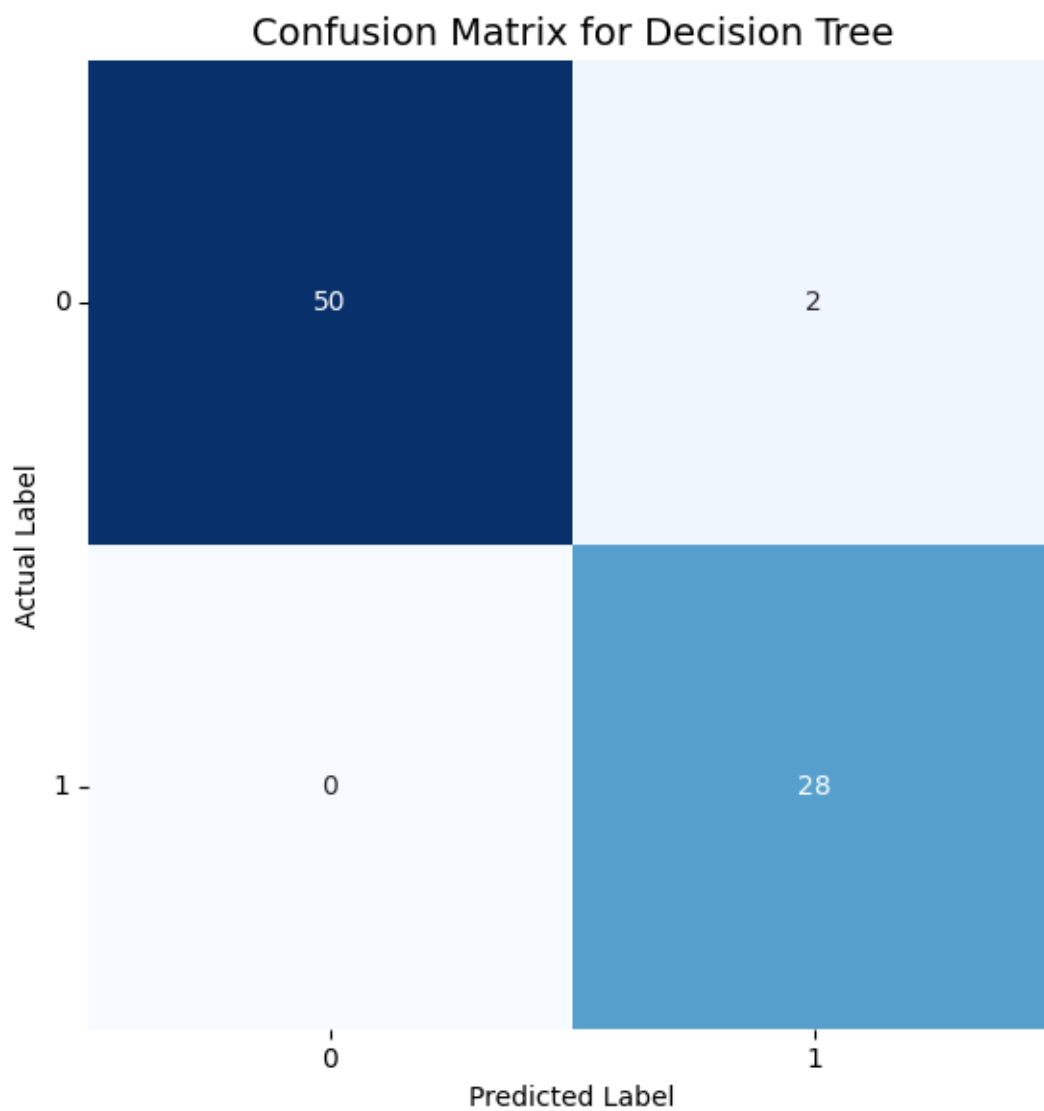
	precision	recall	f1-score	support
0	1.00	0.96	0.98	52
1	0.93	1.00	0.97	28
accuracy			0.97	80
macro avg	0.97	0.98	0.97	80

weighted avg	0.98	0.97	0.98	80
--------------	------	------	------	----

=====

Decision Tree Accuracy: 0.975

	precision	recall	f1-score	support
0	1.00	0.96	0.98	52
1	0.93	1.00	0.97	28
accuracy			0.97	80
macro avg	0.97	0.98	0.97	80
weighted avg	0.98	0.97	0.98	80



=====

Best hyperparameters found by GridSearchCV:

```
{'criterion': 'entropy', 'max_depth': None, 'max_features': None,
'min_samples_leaf': 1, 'min_samples_split': 2}
```

```
[16]: # Support Vector Machine model
SVM_classifier = SVC(random_state=42)
param_grid_SVM = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

grid_search = GridSearchCV(estimator=SVM_classifier, param_grid=param_grid_SVM,
    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)

print("Best Parameters:", grid_search.best_estimator_)
#evaluation_results = evaluate_model('Support Vector Machine', y_test, y_pred)
y_pred_SVM = best_SVM.predict(X_test)
print("=" * 20)
print("Support Vector Machine Accuracy:", accuracy_score(y_test, y_pred_SVM))
print(classification_report(y_test, y_pred_SVM))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,6))
sns.heatmap(
    cm,
    annot=True,
    fmt='d',
    cmap='Blues',
    cbar=False,
    square=True
)
plt.title("Confusion Matrix for Support Vector Machine", fontsize=14)
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.xticks([0.5, 1.5], ['0', '1'])
plt.yticks([0.5, 1.5], ['0', '1'], rotation=0)
plt.tight_layout()
plt.show()
print("=" * 20)
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

--- Performance Evaluation: Support Vector Machine ---
Accuracy: 0.9500

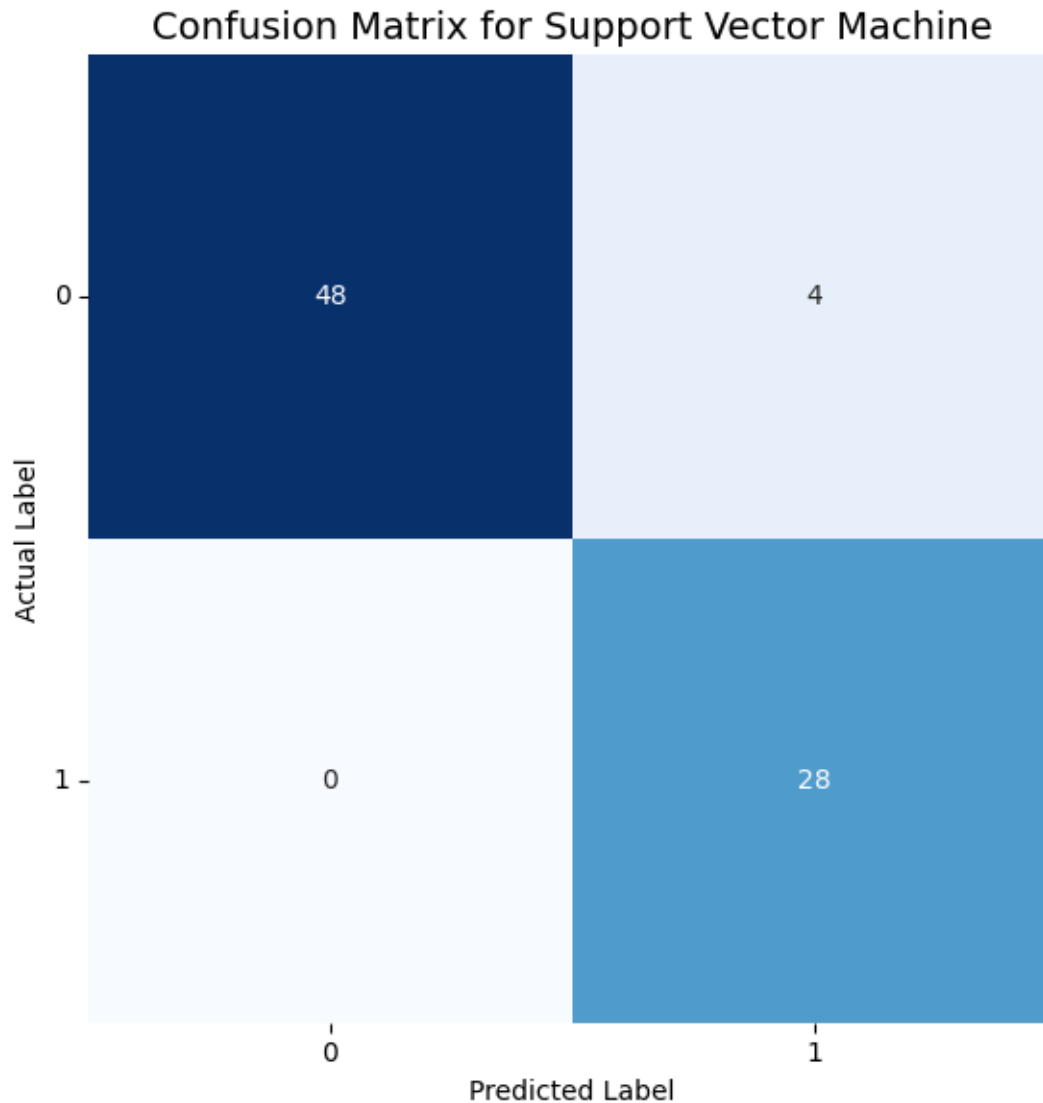
Classification Report:

	precision	recall	f1-score	support
0	1.00	0.92	0.96	52
1	0.88	1.00	0.93	28
accuracy			0.95	80
macro avg	0.94	0.96	0.95	80
weighted avg	0.96	0.95	0.95	80

Best Parameters: SVC(C=0.1, kernel='linear', random_state=42)
=====

Support Vector Machine Accuracy: 0.95

	precision	recall	f1-score	support
0	1.00	0.92	0.96	52
1	0.88	1.00	0.93	28
accuracy			0.95	80
macro avg	0.94	0.96	0.95	80
weighted avg	0.96	0.95	0.95	80



=====

Best hyperparameters found by GridSearchCV:
{'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}

```
[17]: # K-Nearest Neighbors model
knn_classifier = KNeighborsClassifier()
param_grid_knn = {
    'n_neighbors': [1, 3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}
```

```

grid_search = GridSearchCV(estimator=knn_classifier, param_grid=param_grid_knn,
    cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
best_knn = grid_search.best_estimator_
y_pred_knn = best_knn.predict(X_test)
evaluation_results = evaluate_model('K-Nearest Neighbors', y_test, y_pred)

print("Best Parameters:", grid_search.best_estimator_)
#evaluation_results = evaluate_model('K-Nearest Neighbors model', y_test, y_pred)
print("=" * 20)
print("K-Nearest Neighbors Accuracy:", accuracy_score(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,6))
sns.heatmap(
    cm,
    annot=True,
    fmt='d',
    cmap='Blues',
    cbar=False,
    square=True
)
plt.title("Confusion Matrix for K-Nearest Neighbors", fontsize=14)
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.xticks([0.5, 1.5], ['0', '1'])
plt.yticks([0.5, 1.5], ['0', '1'], rotation=0)
plt.tight_layout()
plt.show()
print("=" * 20)
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

--- Performance Evaluation: K-Nearest Neighbors ---

Accuracy: 0.9500

Classification Report:

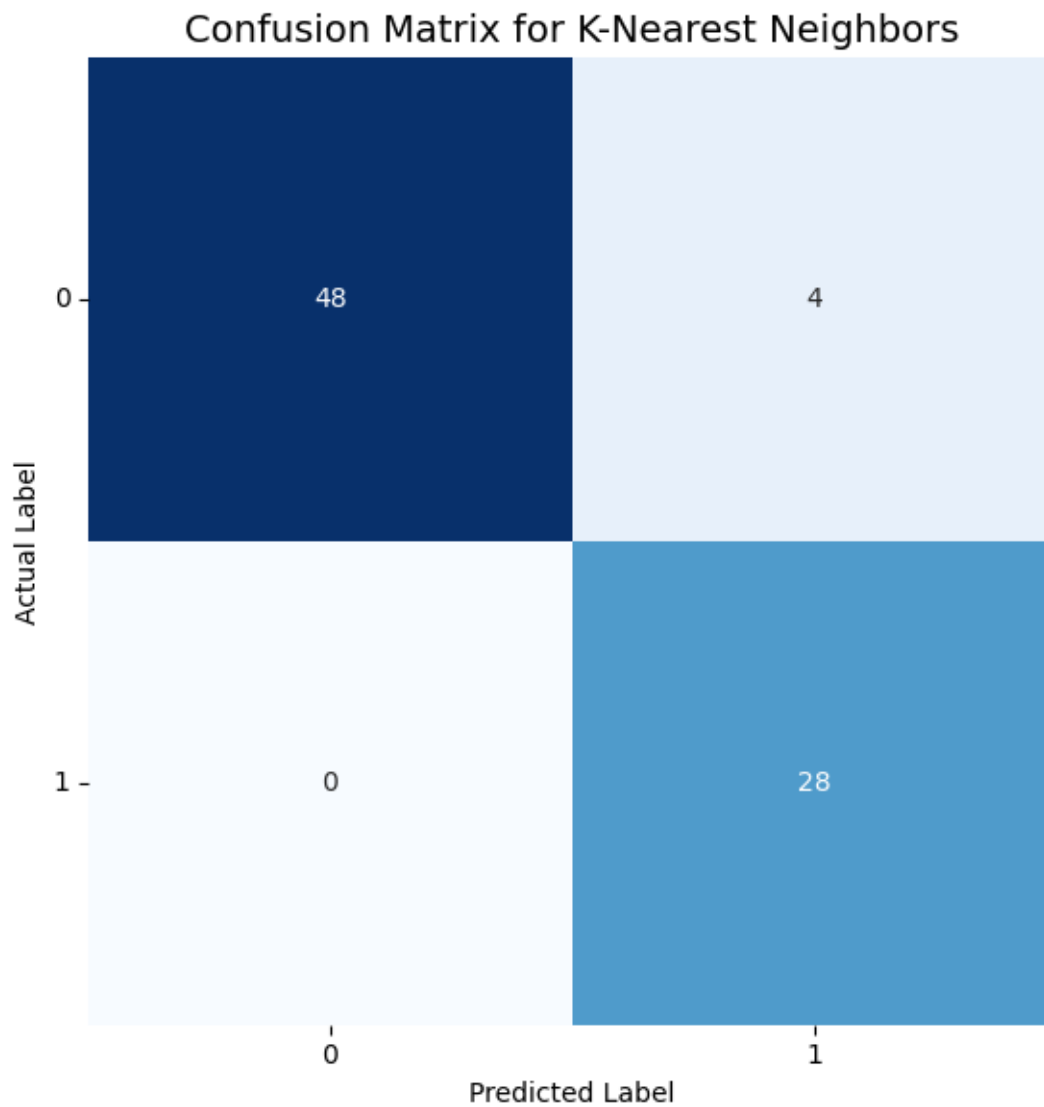
	precision	recall	f1-score	support
0	1.00	0.92	0.96	52
1	0.88	1.00	0.93	28
accuracy			0.95	80
macro avg	0.94	0.96	0.95	80
weighted avg	0.96	0.95	0.95	80

Best Parameters: KNeighborsClassifier(metric='manhattan')

=====

K-Nearest Neighbors Accuracy: 0.875

	precision	recall	f1-score	support
0	0.98	0.83	0.90	52
1	0.75	0.96	0.84	28
accuracy			0.88	80
macro avg	0.86	0.90	0.87	80
weighted avg	0.90	0.88	0.88	80



=====

Best hyperparameters found by GridSearchCV:

```
{'algorithm': 'auto', 'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'uniform'}
```

3. Use two feature selection methods to identify the most important features from the datasets

```
[18]: # Method 1: Statistical Selection using SelectKBest
from sklearn.feature_selection import RFE
kbest_selector = SelectKBest(score_func=f_classif, k=10)
X_train_selected_kbest = kbest_selector.fit_transform(X_train, y_train)
X_test_selected_kbest = kbest_selector.transform(X_test)
selected_features_names = X_train.columns[kbest_selector.get_support()].tolist()
print("Features selected using SelectKBest:", selected_features_names)

# Method 2: Recursive Feature Elimination (RFE)
rfe_selector = RFE(estimator=best_DT, n_features_to_select=10)
X_train_selected_rfe = rfe_selector.fit_transform(X_train, y_train)
X_test_selected_rfe = rfe_selector.transform(X_test)
selected_features_names_rfe = X_train.columns[rfe_selector.get_support()].
    ↪tolist()
print("Features selected using RFE:", selected_features_names_rfe)
```

Features selected using SelectKBest: ['sg', 'al', 'su', 'bgr', 'bu', 'hemo', 'htn', 'dm', 'appet', 'pe']

Features selected using RFE: ['sg', 'al', 'sc', 'hemo', 'pcv', 'wc', 'rc', 'appet', 'pe', 'ane']

4. Apply machine learning models to the selected features and evaluate the models.

```
[19]: ## Method 1 SelectKBest
# 1. Random Forest Classifier on Selected Features
best_RF.fit(X_train_selected_kbest, y_train)
y_pred_rf_fs = best_RF.predict(X_test_selected_kbest)
evaluate_model('Random Forest (SelectKBest)', y_test, y_pred_rf_fs)

# 2. Logistic Regression on Selected Features
best_LR.fit(X_train_selected_kbest, y_train)
y_pred_lr_fs = best_LR.predict(X_test_selected_kbest)
evaluate_model('Logistic Regression (SelectKBest)', y_test, y_pred_lr_fs)

# 3. Decision Tree Classifier on Selected Features
best_DT.fit(X_train_selected_kbest, y_train)
y_pred_dt_fs = best_DT.predict(X_test_selected_kbest)
evaluate_model('Decision Tree (SelectKBest)', y_test, y_pred_dt_fs)

# 4. Support Vector Machine (SVM) on Selected Features
best_SVM.fit(X_train_selected_kbest, y_train)
```

```

y_pred_svm_fs = best_SVM.predict(X_test_selected_kbest)
evaluate_model('SVM (SelectKBest)', y_test, y_pred_svm_fs)

# 5. K-Nearest Neighbors (KNN) on Selected Features
best_knn.fit(X_train_selected_kbest, y_train)
y_pred_knn_fs = best_knn.predict(X_test_selected_kbest)
evaluate_model('KNN (SelectKBest)', y_test, y_pred_knn_fs)

models = {
    "Random Forest": best_RF,
    "Logistic Regression": best_LR,
    "Decision Tree": best_DT,
    "SVM": best_SVM,
    "KNN": best_knn
}

metrics = []

print("\n" + "="*60)
print("FINAL PERFORMANCE EVALUATION ON SELECTED FEATURES (SelectKBest)")
print("="*60)

for name, model in models.items():
    model.fit(X_train_selected_kbest, y_train)

    y_pred = model.predict(X_test_selected_kbest)

    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_kbest = pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_kbest)

```

--- Performance Evaluation: Random Forest (SelectKBest) ---
Accuracy: 0.9750

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.96	0.98	52
1	0.93	1.00	0.97	28
accuracy			0.97	80

macro avg	0.97	0.98	0.97	80
weighted avg	0.98	0.97	0.98	80

--- Performance Evaluation: Logistic Regression (SelectKBest) ---
Accuracy: 0.9375

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.92	0.95	52
1	0.87	0.96	0.92	28
accuracy			0.94	80
macro avg	0.93	0.94	0.93	80
weighted avg	0.94	0.94	0.94	80

--- Performance Evaluation: Decision Tree (SelectKBest) ---
Accuracy: 0.9625

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.94	0.97	52
1	0.90	1.00	0.95	28
accuracy			0.96	80
macro avg	0.95	0.97	0.96	80
weighted avg	0.97	0.96	0.96	80

--- Performance Evaluation: SVM (SelectKBest) ---
Accuracy: 0.9375

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.92	0.95	52
1	0.87	0.96	0.92	28
accuracy			0.94	80
macro avg	0.93	0.94	0.93	80
weighted avg	0.94	0.94	0.94	80

--- Performance Evaluation: KNN (SelectKBest) ---
Accuracy: 0.8750

Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	1.00	0.81	0.89	52
1	0.74	1.00	0.85	28
accuracy			0.88	80
macro avg	0.87	0.90	0.87	80
weighted avg	0.91	0.88	0.88	80

```
=====
FINAL PERFORMANCE EVALUATION ON SELECTED FEATURES (SelectKBest)
=====
```

Model Performance:

	Model	Accuracy	Precision	Recall
0	Random Forest	0.9750	0.976667	0.9750
1	Logistic Regression	0.9375	0.941573	0.9375
2	Decision Tree	0.9625	0.966129	0.9625
3	SVM	0.9375	0.941573	0.9375
4	KNN	0.8750	0.907895	0.8750

```
[20]: ## Method 2 RFE

models_dict = {
    "Random Forest": best_RF,
    "Logistic Regression": best_LR,
    "Decision Tree": best_DT,
    "SVM": best_SVM,
    "KNN": best_knn
}

print("\n" + "="*60)
print("FINAL PERFORMANCE EVALUATION ON SELECTED FEATURES (RFE)")
print("="*60)

metrics = []

for name, model in models_dict.items():
    model.fit(X_train_selected_rfe, y_train)
    y_pred = model.predict(X_test_selected_rfe)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted')
    rec = recall_score(y_test, y_pred, average='weighted')

    metrics.append({
        "Model": name,
        "Accuracy": acc,
```

```

        "Precision": prec,
        "Recall": rec
    })

    evaluate_model(f'{name} (Selected Features)', y_test, y_pred)

results_rfe = pd.DataFrame(metrics)
print("\nFinal Model Comparison Table:")
print(results_rfe)

```

```

=====
FINAL PERFORMANCE EVALUATION ON SELECTED FEATURES (RFE)
=====

```

```

--- Performance Evaluation: Random Forest (Selected Features) ---
Accuracy: 0.9875

```

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.98	0.99	52
1	0.97	1.00	0.98	28
accuracy			0.99	80
macro avg	0.98	0.99	0.99	80
weighted avg	0.99	0.99	0.99	80

```

--- Performance Evaluation: Logistic Regression (Selected Features) ---
Accuracy: 0.9500

```

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.94	0.96	52
1	0.90	0.96	0.93	28
accuracy			0.95	80
macro avg	0.94	0.95	0.95	80
weighted avg	0.95	0.95	0.95	80

```

--- Performance Evaluation: Decision Tree (Selected Features) ---
Accuracy: 0.9750

```

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.96	0.98	52
1	0.93	1.00	0.97	28

accuracy			0.97	80
macro avg	0.97	0.98	0.97	80
weighted avg	0.98	0.97	0.98	80

--- Performance Evaluation: SVM (Selected Features) ---
Accuracy: 0.9375

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.92	0.95	52
1	0.87	0.96	0.92	28

accuracy			0.94	80
macro avg	0.93	0.94	0.93	80
weighted avg	0.94	0.94	0.94	80

--- Performance Evaluation: KNN (Selected Features) ---
Accuracy: 0.8375

Classification Report:

	precision	recall	f1-score	support
0	0.95	0.79	0.86	52
1	0.70	0.93	0.80	28

accuracy			0.84	80
macro avg	0.83	0.86	0.83	80
weighted avg	0.87	0.84	0.84	80

Final Model Comparison Table:

	Model	Accuracy	Precision	Recall
0	Random Forest	0.9875	0.987931	0.9875
1	Logistic Regression	0.9500	0.952000	0.9500
2	Decision Tree	0.9750	0.976667	0.9750
3	SVM	0.9375	0.941573	0.9375
4	KNN	0.8375	0.865713	0.8375

5.Models evaluatcion

```
[21]: comparison = pd.DataFrame({
    "Model": results_rfe["Model"],
    "RFE Accuracy": results_rfe["Accuracy"],
    "SelectKBest Accuracy": results_kbest["Accuracy"]
})
```

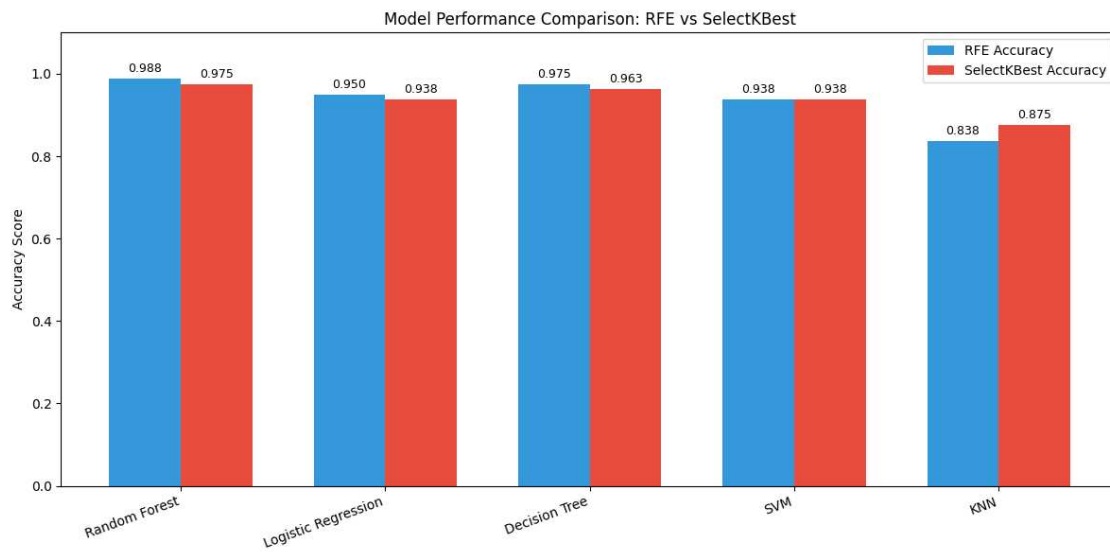
```

print(comparison)

models = comparison["Model"]
rfe_acc = comparison["RFE Accuracy"]
kbest_acc = comparison["SelectKBest Accuracy"]
x = np.arange(len(models))
width = 0.35
fig, ax = plt.subplots(figsize=(12, 6))
rects1 = ax.bar(x - width/2, rfe_acc, width, label='RFE Accuracy',
    color='#3498db')
rects2 = ax.bar(x + width/2, kbest_acc, width, label='SelectKBest Accuracy',
    color='#e74c3c')
ax.set_ylabel('Accuracy Score')
ax.set_title('Model Performance Comparison: RFE vs SelectKBest')
ax.set_xticks(x)
ax.set_xticklabels(models, rotation=20, ha='right')
ax.legend()
ax.set_ylim(0, 1.1)
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.3f}',
            xy=(rect.get_x() + rect.get_width() / 2, height),
            xytext=(0, 3),
            textcoords="offset points",
            ha='center', va='bottom', fontsize=9)
autolabel(rects1)
autolabel(rects2)
plt.tight_layout()
plt.savefig('accuracy_comparison.png')

```

	Model	RFE Accuracy	SelectKBest Accuracy
0	Random Forest	0.9875	0.9750
1	Logistic Regression	0.9500	0.9375
2	Decision Tree	0.9750	0.9625
3	SVM	0.9375	0.9375
4	KNN	0.8375	0.8750



[]: