

Faisal_Almokayed_Project

December 18, 2025

Practical_Machine_Learning_and_Data_Exploration 2025

Dr.Hajer Saleh

Faisal Ahmed Almokayed - 253001589

Practice on Prediction of Sepsis Dataset

```
[4]: from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
[32]: import warnings
warnings.filterwarnings("ignore")
# =====
# Core libraries
# =====
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# =====
# Scikit-learn: data handling
# =====
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
# =====
# Scikit-learn: preprocessing & feature selection
# =====
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.feature_selection import SelectKBest, chi2, RFE
# =====
# Scikit-learn: models
# =====
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
# =====
# Scikit-learn: evaluation metrics
# =====
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, \
    recall_score, f1_score, classification_report
# =====
# Imbalanced-learn: sampling methods
# =====
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTETomek

```

Dataset Link: https://www.kaggle.com/datasets/imnoob/kidney-disease?select=kidney_disease.csv

```
[6]: df = pd.read_csv('/content/drive/MyDrive/Project 2/kidney_disease.csv')
```

```
[7]: df.head()
```

```
[7]:
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	\
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	

	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	121.0	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	NaN	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	423.0	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	117.0	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	106.0	...	35	7300	4.6	no	no	no	good	no	no	ckd

[5 rows x 25 columns]

```
[8]: df['classification'].value_counts()
```

```
[8]: classification
ckd      248
notckd   150
Name: count, dtype: int64
```

```
[9]: df.dtypes
```

```
[9]: age          float64
     bp          float64
     sg          float64
     al          float64
     su          float64
     rbc         object
     pc          object
     pcc         object
     ba          object
     bgr         float64
     bu          float64
     sc          float64
     sod         float64
     pot         float64
     hemo        float64
     pcv         object
     wc          object
     rc          object
     htn         object
     dm          object
     cad         object
     appet       object
     pe          object
     ane         object
     classification object
     dtype: object
```

```
[10]: df.columns
```

```
[10]: Index(['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr', 'bu',
            'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad',
            'appet', 'pe', 'ane', 'classification'],
            dtype='object')
```

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

```
[11]: age          9
     bp          12
     sg          46
     al          45
     su          48
     rbc        151
     pc          64
     pcc         4
     ba          4
     bgr         44
     bu          19
```

```

sc          17
sod         87
pot         88
hemo        51
pcv         69
wc          105
rc          129
htn         2
dm          2
cad         2
appet       1
pe          1
ane         1
classification 0
dtype: int64

```

```

[12]: missing_percentage = df.isnull().mean() * 100
      # Filter columns with more than 30% missing values
      columns_with_missing_30 = missing_percentage[missing_percentage > 30]
      # Display the columns with more than 30% missing values
      print("Columns with more than 30% missing values:")
      print(columns_with_missing_30)

```

```

Columns with more than 30% missing values:
rbc      37.939698
rc       32.412060
dtype: float64

```

```

[13]: missing_percentage = df.isnull().mean() * 100
      threshold = 30
      # Filter columns with more than the threshold percentage of missing values
      columns_to_drop = missing_percentage[missing_percentage > threshold].index
      # Drop the columns
      df = df.drop(columns=columns_to_drop)
      # Display the resulting dataframe after removing columns with too many missing
      # values
      print(f"Columns dropped: {list(columns_to_drop)}")
      print(f"Remaining columns: {df.columns}")

```

```

Columns dropped: ['rbc', 'rc']
Remaining columns: Index(['age', 'bp', 'sg', 'al', 'su', 'pc', 'pcc', 'ba',
                        'bgr', 'bu', 'sc',
                        'sod', 'pot', 'hemo', 'pcv', 'wc', 'htn', 'dm', 'cad', 'appet', 'pe',
                        'ane', 'classification'],
                        dtype='object')

```

Pre-processing on dataset

```
[14]: 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()
```

```
[14]:   age    bp    sg    al    su    pc    pcc    ba    bgr    bu    ...    hemo    pcv    wc    \
0  48.0  80.0  1.020  1.0  0.0    1     0     0  121.0  36.0  ...   15.4    32    72
1   7.0  50.0  1.020  4.0  0.0    1     0     0   NaN   18.0  ...   11.3    26    56
2  62.0  80.0  1.010  2.0  3.0    1     0     0  423.0  53.0  ...    9.6    19    70
3  48.0  70.0  1.005  4.0  0.0    0     1     0  117.0  56.0  ...   11.2    20    62
4  51.0  80.0  1.010  2.0  0.0    1     0     0  106.0  26.0  ...   11.6    23    68

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

[5 rows x 23 columns]

Filling missing values based on feature type

```
[15]: # Fill missing values in columns with mode
for column in df.select_dtypes(include=['float64']).columns:
    mean_value = df[column].mean() # Get the mode (most frequent value)
    df[column] = df[column].fillna(mean_value)
# Fill missing values in categorical columns with mode
for column in df.select_dtypes(include=['object']).columns:
    mode_value = df[column].mode()[0] # Get the mode (most frequent value)
    df[column] = df[column].fillna(mode_value)
```

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

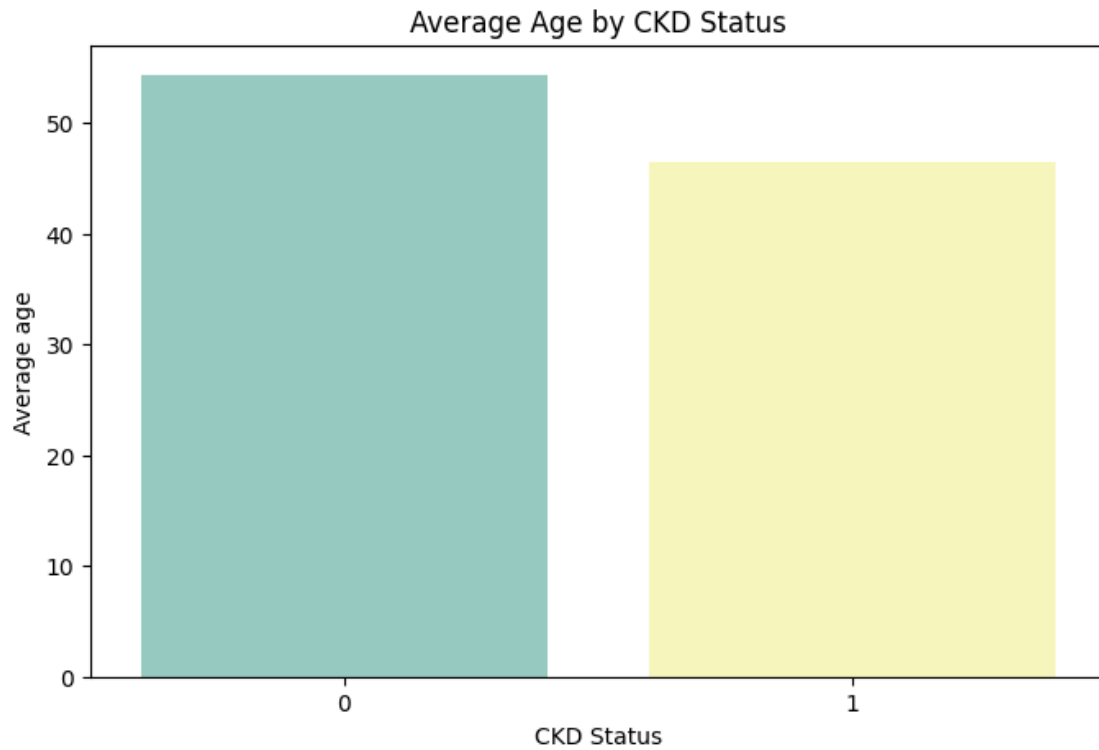
```
[16]: age          0
bp            0
sg            0
al            0
su            0
pc            0
pcc           0
ba            0
bgr           0
bu            0
sc            0
```

```
sod          0
pot          0
hemo         0
pcv          0
wc           0
htn          0
dm           0
cad          0
appet        0
pe           0
ane          0
classification 0
dtype: int64
```

Save Cleaned Dataset

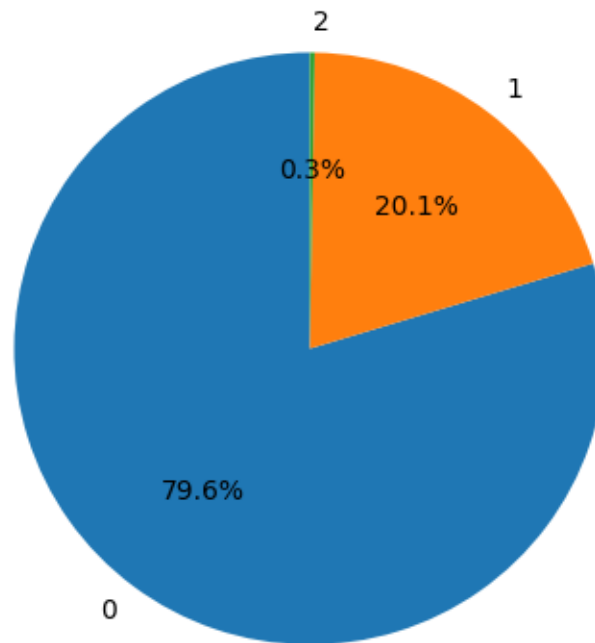
```
[17]: df.to_csv('/content/drive/MyDrive/Project 2/cleaned_kidney_disease.csv',  
        ↪index=False)
```

```
[18]: plt.figure(figsize=(8,5))
sns.barplot(
    data=df,
    x='classification',
    y='age',
    estimator='mean',
    palette='Set3',
    errorbar=None
)
plt.title("Average Age by CKD Status")
plt.xlabel("CKD Status")
plt.ylabel("Average age")
plt.show()
```

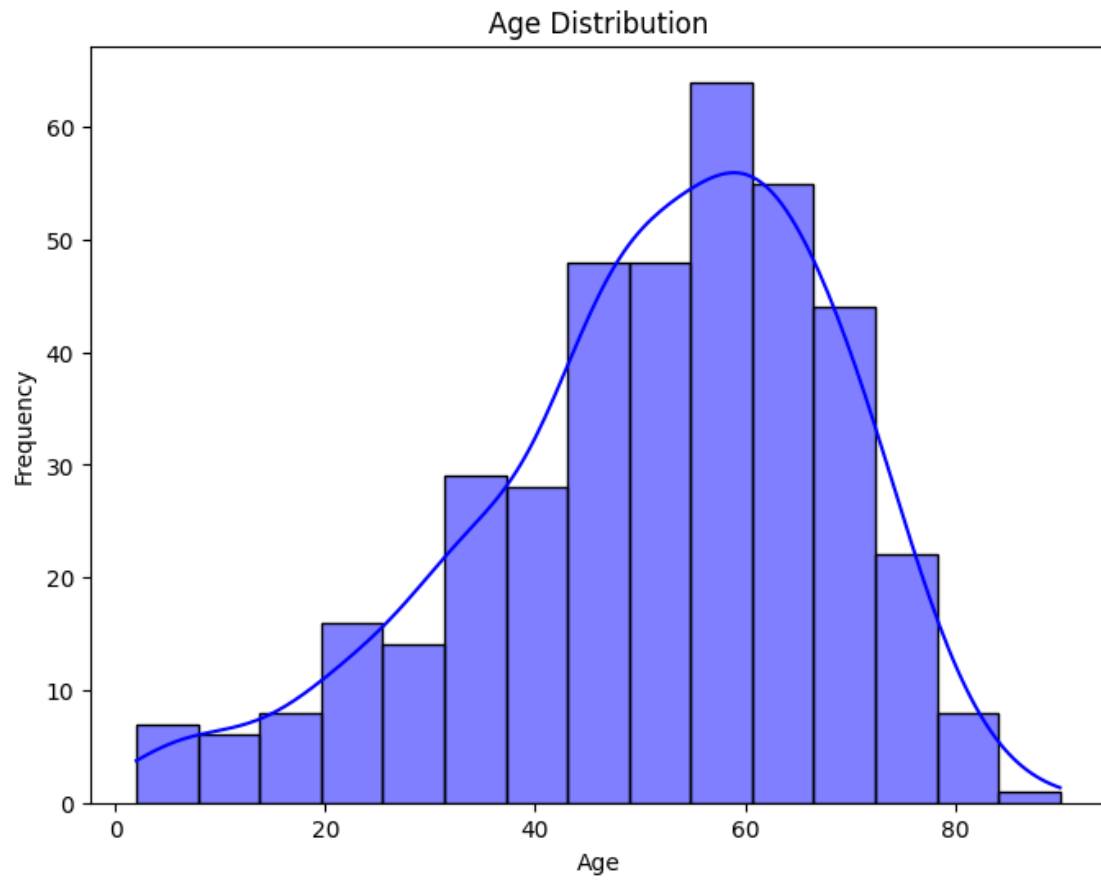


```
[19]: appet_counts = df['appet'].value_counts()
plt.figure(figsize=(5,5))
plt.pie(appet_counts, labels=appet_counts.index, autopct='%1.1f%%',
startangle=90)
plt.title("Appetite Status Distribution")
plt.show()
```

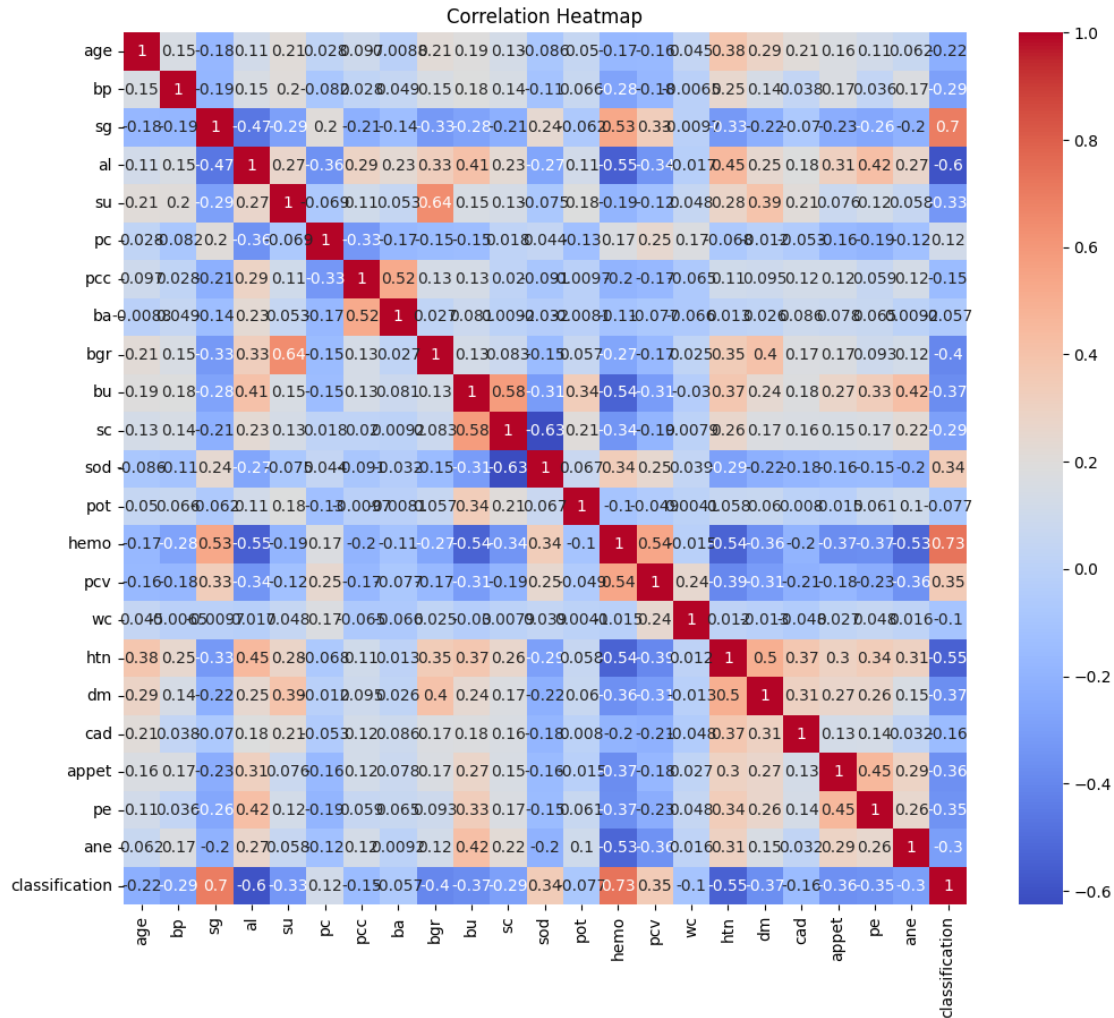
Appetite Status Distribution



```
[20]: cat_cols = [col for col in df.columns if df[col].dtype=='object']
num_cols = [col for col in df.columns if df[col].dtype!='object']
for col in cat_cols:
    print(f'{col} has {df[col].unique()} values')
# 1. Distribution of Age
plt.figure(figsize=(8, 6))
sns.histplot(df['age'], bins=15, kde=True, color='blue')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



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



```
[22]: df['classification'].value_counts()
```

```
[22]: classification
0      248
1      150
Name: count, dtype: int64
```

1 Split data into training and testing sets

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

```
[30]: X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.30,
      random_state=42, stratify=y)
```

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

    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    # Create a report
    report = classification_report(y_true, y_pred)

    # Output results
    metrics = {
        'Model Name': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,

        'Classification Report': report
    }

    # Plot Confusion Matrix
    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 Label')
    plt.ylabel('True Label')
    plt.show()

    return metrics
```

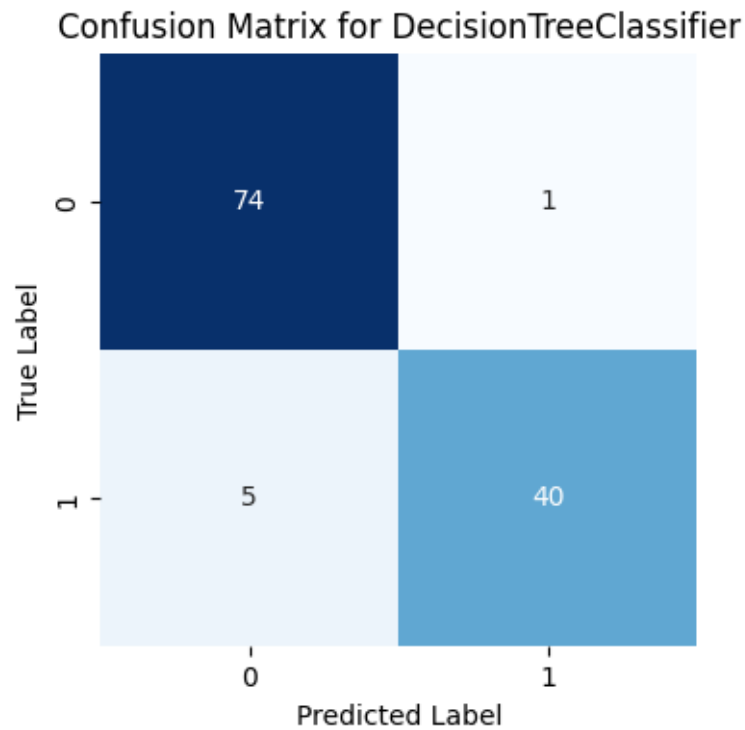
Decision Tree

```
[35]: DT = DecisionTreeClassifier()
    # Train the model
    DT.fit(X_train, y_train)
    # Make predictions
    y_pred = DT.predict(X_test)
    # Evaluate the model
    evaluation_results = evaluate_model('DecisionTreeClassifier', y_test, y_pred)
    # Print the evaluation results
    for key, value in evaluation_results.items():
```

```

if key == 'Classification Report':
    print(value) # Print report separately for better readability
else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```



Model Name:

DecisionTreeClassifier

Accuracy: 0.9500

Precision: 0.9513

Recall: 0.9500

F1 Score: 0.9495

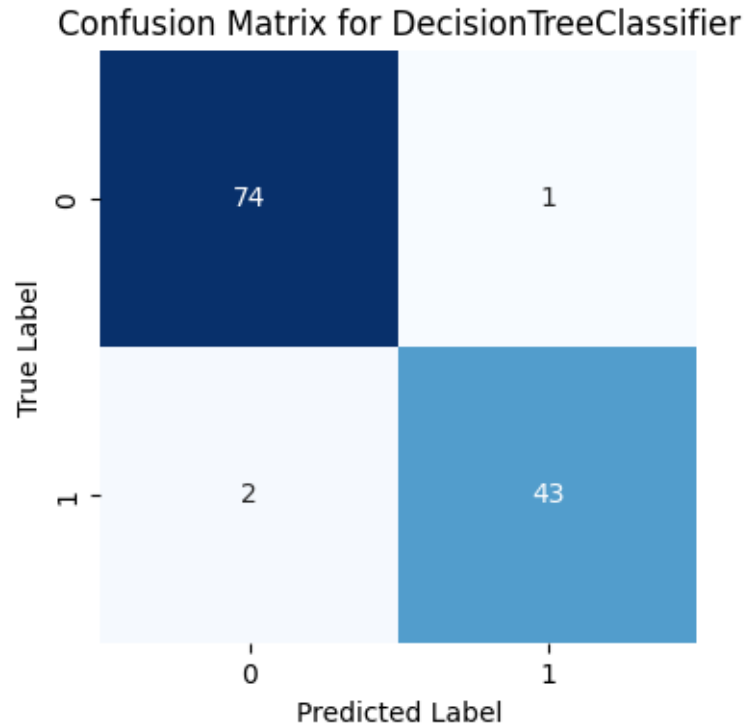
	precision	recall	f1-score	support
0	0.94	0.99	0.96	75
1	0.98	0.89	0.93	45
accuracy			0.95	120
macro avg	0.96	0.94	0.95	120
weighted avg	0.95	0.95	0.95	120

DecisionTreeClassifier with grid search

```

[36]: # Define the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)
# Define the hyperparameters for grid search
param_grid = {
    'criterion': ['gini', 'entropy'],          # The function to measure the
    ↪quality of a split
    'max_depth': [None, 10, 20, 30],          # The maximum depth of the
    ↪tree
    'min_samples_split': [2, 5, 10],          # The minimum number of
    ↪samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],           # The minimum number of
    ↪samples required to be at a leaf node
    'max_features': [None, 'sqrt', 'log2']    # The number of features to
    ↪consider when looking for the best split
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=DT, param_grid=param_grid, cv=5,
    ↪scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_DT = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_DT.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```



Model Name:

DecisionTreeClassifier

Accuracy: 0.9750

Precision: 0.9750

Recall: 0.9750

F1 Score: 0.9749

	precision	recall	f1-score	support
0	0.97	0.99	0.98	75
1	0.98	0.96	0.97	45
accuracy			0.97	120
macro avg	0.98	0.97	0.97	120
weighted avg	0.98	0.97	0.97	120

Best hyperparameters found by GridSearchCV:

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

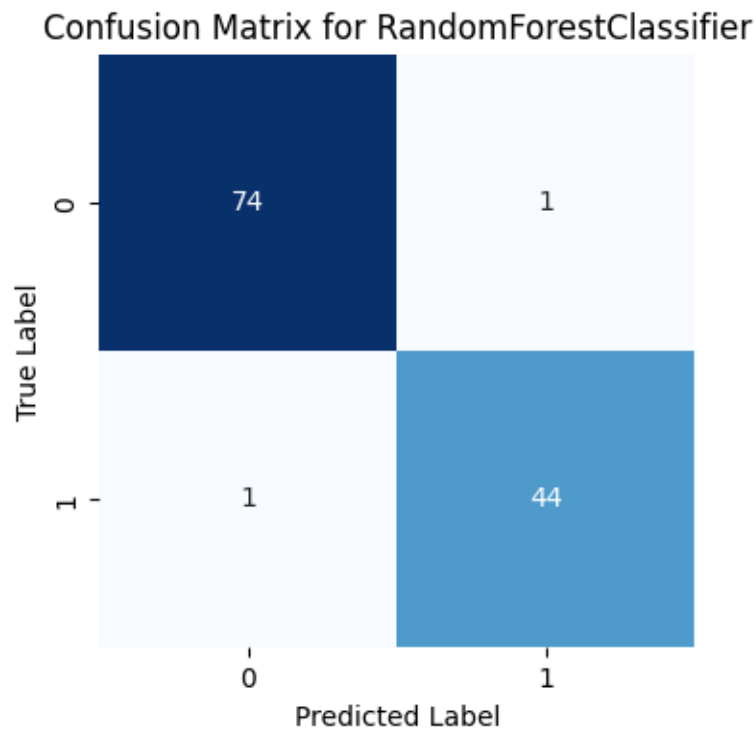
Random Forest (RF)

```
[37]: rf_classifier = RandomForestClassifier(n_estimators=100,
random_state=42)
```

```

# Train the model
rf_classifier.fit(X_train, y_train)
# Make predictions
y_pred = rf_classifier.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```



Model Name:

RandomForestClassifier

Accuracy: 0.9833

Precision: 0.9833

Recall: 0.9833

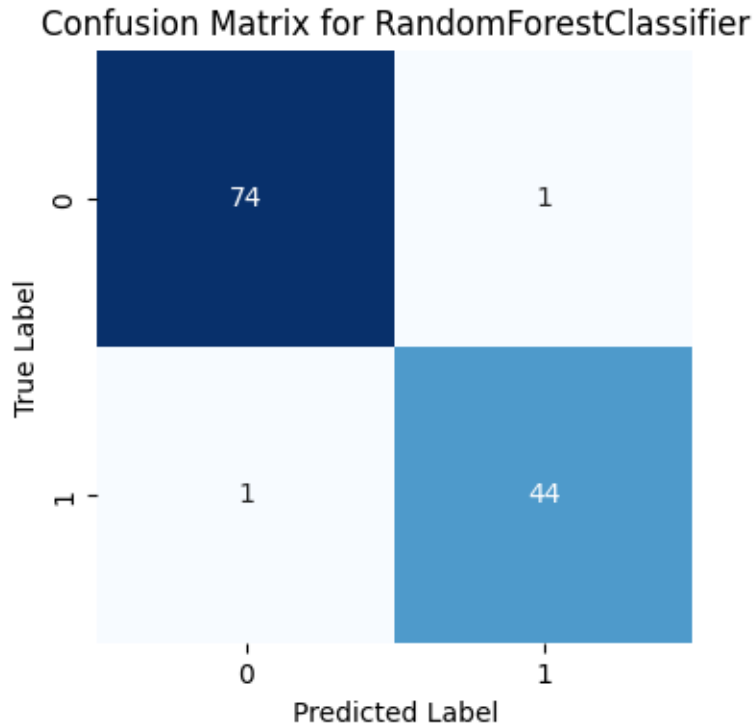
F1 Score: 0.9833

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

0	0.99	0.99	0.99	75
1	0.98	0.98	0.98	45
accuracy			0.98	120
macro avg	0.98	0.98	0.98	120
weighted avg	0.98	0.98	0.98	120

Random Forest (RF) with grid search

```
[38]: # Define the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)
# Define the hyperparameters for grid search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier,
param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_rf_classifier = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_rf_classifier.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
# Print the best parameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```



Model Name:

RandomForestClassifier

Accuracy: 0.9833

Precision: 0.9833

Recall: 0.9833

F1 Score: 0.9833

	precision	recall	f1-score	support
0	0.99	0.99	0.99	75
1	0.98	0.98	0.98	45
accuracy			0.98	120
macro avg	0.98	0.98	0.98	120
weighted avg	0.98	0.98	0.98	120

Best hyperparameters found by GridSearchCV:

```
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
```

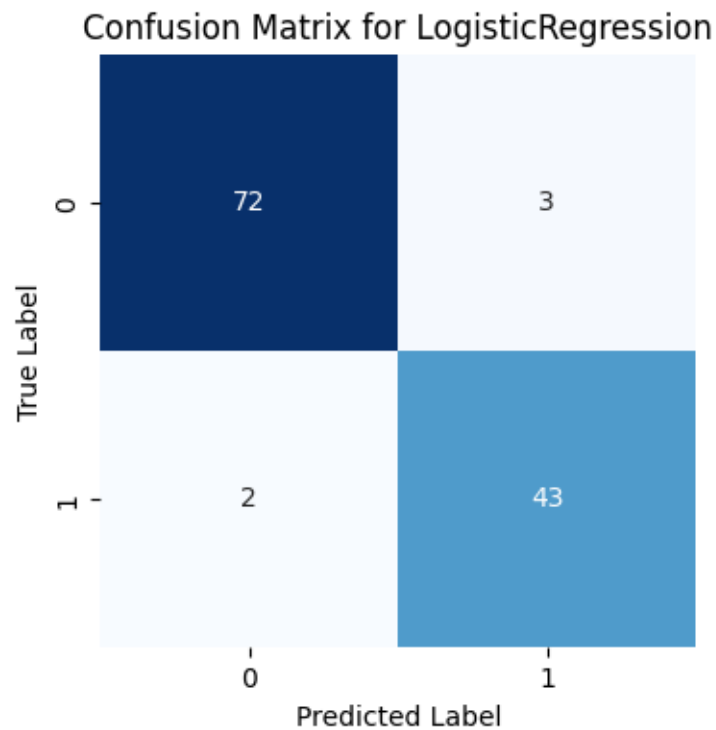
Logistic Regression

```
[39]: LR = LogisticRegression()  
      # Train the model
```

```

LR.fit(X_train, y_train)
# Make predictions
y_pred = LR.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```



Model Name:

LogisticRegression

Accuracy: 0.9583

Precision: 0.9587

Recall: 0.9583

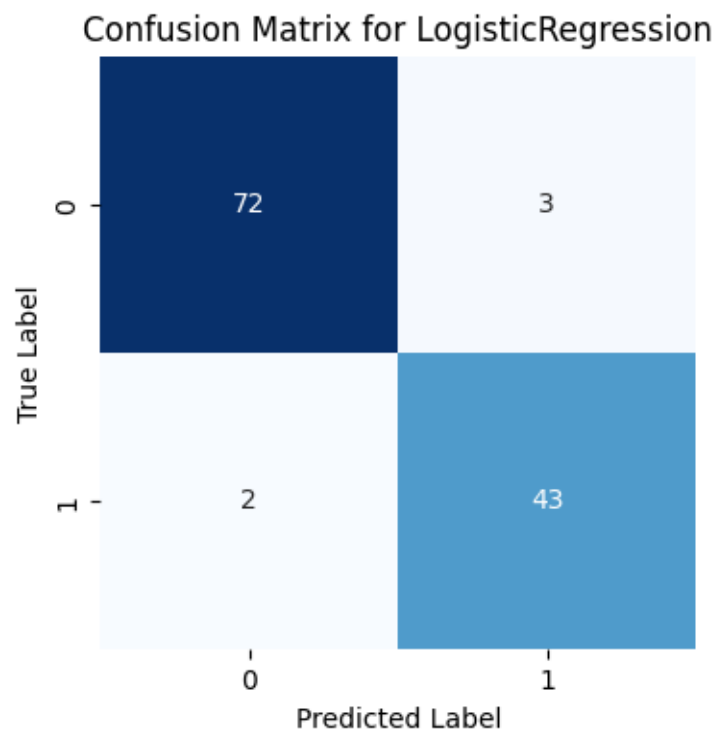
F1 Score: 0.9584

	precision	recall	f1-score	support
0	0.97	0.96	0.97	75

	1	0.93	0.96	0.95	45
accuracy				0.96	120
macro avg	0.95	0.96	0.96	0.96	120
weighted avg	0.96	0.96	0.96	0.96	120

LogisticRegression with grid search

```
[40]: LR = LogisticRegression(random_state=42)
# Define the hyperparameters for grid search
param_grid = {
    'C': [0.01, 0.1, 1, 10],          # Regularization strength
    'solver': ['liblinear', 'lbfgs'],  # Solver type
    'max_iter': [100, 200, 300]      # Maximum iterations for convergence
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=LR, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_LR = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_LR.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```



Model Name:
LogisticRegression
Accuracy: 0.9583
Precision: 0.9587
Recall: 0.9583
F1 Score: 0.9584

	precision	recall	f1-score	support
0	0.97	0.96	0.97	75
1	0.93	0.96	0.95	45
accuracy			0.96	120
macro avg	0.95	0.96	0.96	120
weighted avg	0.96	0.96	0.96	120

Best hyperparameters found by GridSearchCV:
{'C': 0.1, 'max_iter': 300, 'solver': 'lbfgs'}

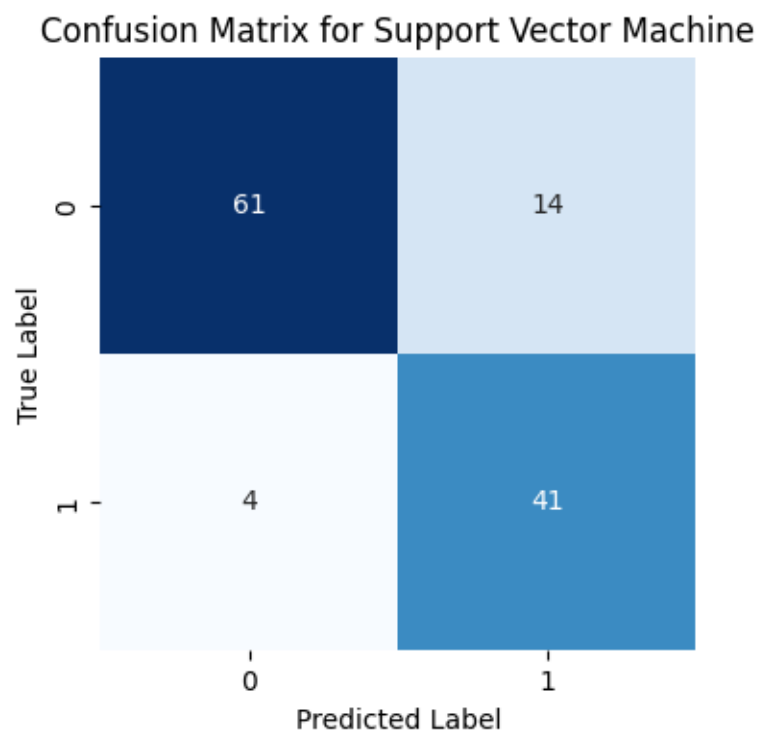
Support Vector Machine

```
[41]: SVM = SVC()
      # Train the model
      SVM.fit(X_train, y_train)
```

```

# Make predictions
y_pred = SVM.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```



Model Name:

Support Vector Machine

Accuracy: 0.8500

Precision: 0.8661

Recall: 0.8500

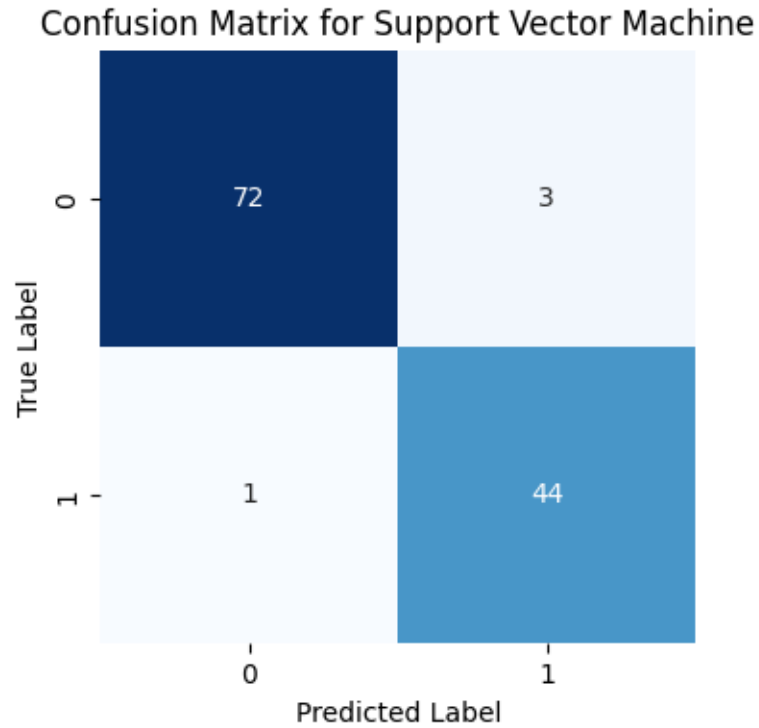
F1 Score: 0.8521

	precision	recall	f1-score	support
0	0.94	0.81	0.87	75
1	0.75	0.91	0.82	45

accuracy			0.85	120
macro avg	0.84	0.86	0.85	120
weighted avg	0.87	0.85	0.85	120

Support Vector Machine with grid search

```
[42]: SVM = SVC(random_state=42)
# Define the hyperparameters for grid search
param_grid = {
    'C': [0.01, 0.1, 1, 10],          # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly'], # Kernel type
    'gamma': ['scale', 'auto'],        # Kernel coefficient for 'rbf',
    ↪ 'poly', 'sigmoid'
    'degree': [3, 4, 5],              # Degree of the polynomial kernel
    ↪ function ('poly')
    'class_weight': [None, 'balanced'] # Weighing classes in the decision
    ↪ function
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=SVM, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_SVM = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_SVM.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```



Model Name:

Support Vector Machine

Accuracy: 0.9667

Precision: 0.9675

Recall: 0.9667

F1 Score: 0.9668

	precision	recall	f1-score	support
0	0.99	0.96	0.97	75
1	0.94	0.98	0.96	45
accuracy			0.97	120
macro avg	0.96	0.97	0.96	120
weighted avg	0.97	0.97	0.97	120

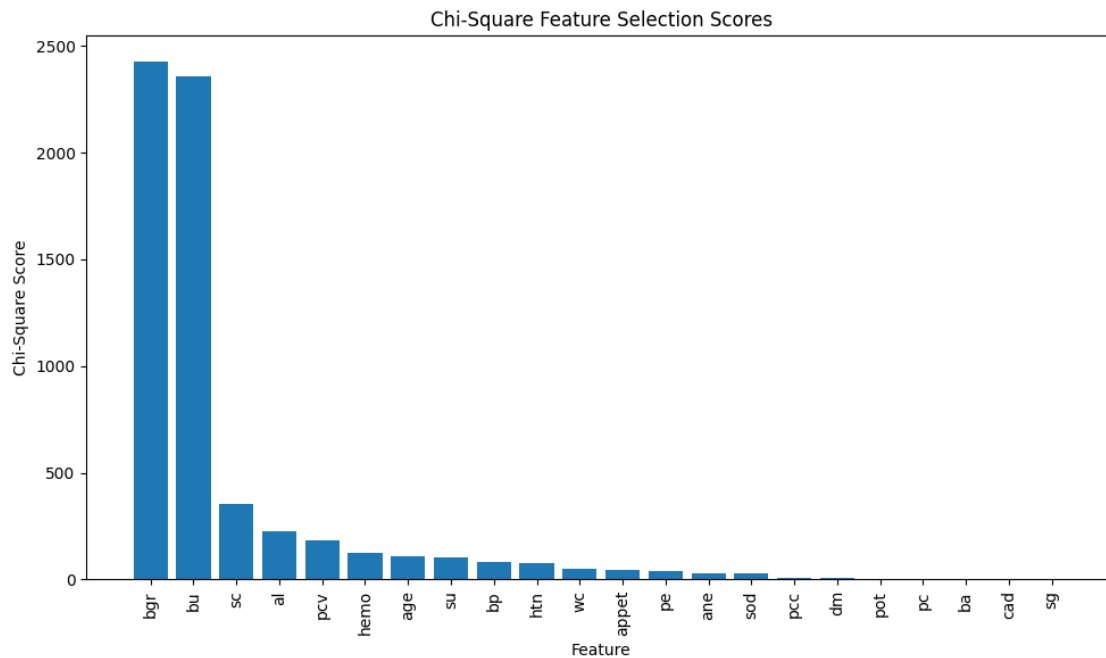
Best hyperparameters found by GridSearchCV:

```
{'C': 0.01, 'class_weight': None, 'degree': 3, 'gamma': 'scale', 'kernel': 'linear'}
```

Chi-Square Test

Chi-Square as feature selection

```
[43]: # Create selector (choose number of features you want, e.g., k=5)
selector = SelectKBest(score_func=chi2, k=10)
# Fit and transform
selector.fit(X, y)
# Create DataFrame of features and chi-square scores
data = pd.DataFrame({'Feature': X.columns, 'Score': selector.scores_})
# Sort by score (optional, but makes plot clearer)
data = data.sort_values(by='Score', ascending=False)
# Plot bar chart
plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Score'])
plt.xlabel("Feature")
plt.ylabel("Chi-Square Score")
plt.title("Chi-Square Feature Selection Scores")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



Comparison of models using features selected by the Chi-Square

```
[44]: # 1. TRAIN / TEST SPLIT
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪ 30, random_state=42)
# 2. CHI-SQUARE FEATURE SELECTION
k= 10
selector = SelectKBest(score_func=chi2, k=k)
```

```

X_train_k = selector.fit_transform(X_train, y_train)
X_test_k = selector.transform(X_test)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", list(selected_features))
# 4. MODELS
models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf', random_state=42)
}
metrics = []

# 5. TRAIN + EVALUATE
for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)
    print(f"\n===== Classification Report: {name} =====")
    print(classification_report(y_test, y_pred))
    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')
    })
# 6. RESULTS AS DATAFRAME
results_df = pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_df)

```

Selected Features: ['age', 'al', 'su', 'bgr', 'bu', 'sc', 'hemo', 'pcv', 'wc', 'htn']

```

===== Classification Report: Random Forest =====

```

	precision	recall	f1-score	support
0	1.00	0.96	0.98	76
1	0.94	1.00	0.97	44
accuracy			0.97	120
macro avg	0.97	0.98	0.97	120
weighted avg	0.98	0.97	0.98	120

```

===== Classification Report: Decision Tree =====

```

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

0	0.96	0.93	0.95	76
1	0.89	0.93	0.91	44
accuracy			0.93	120
macro avg	0.93	0.93	0.93	120
weighted avg	0.93	0.93	0.93	120

==== Classification Report: SVM =====

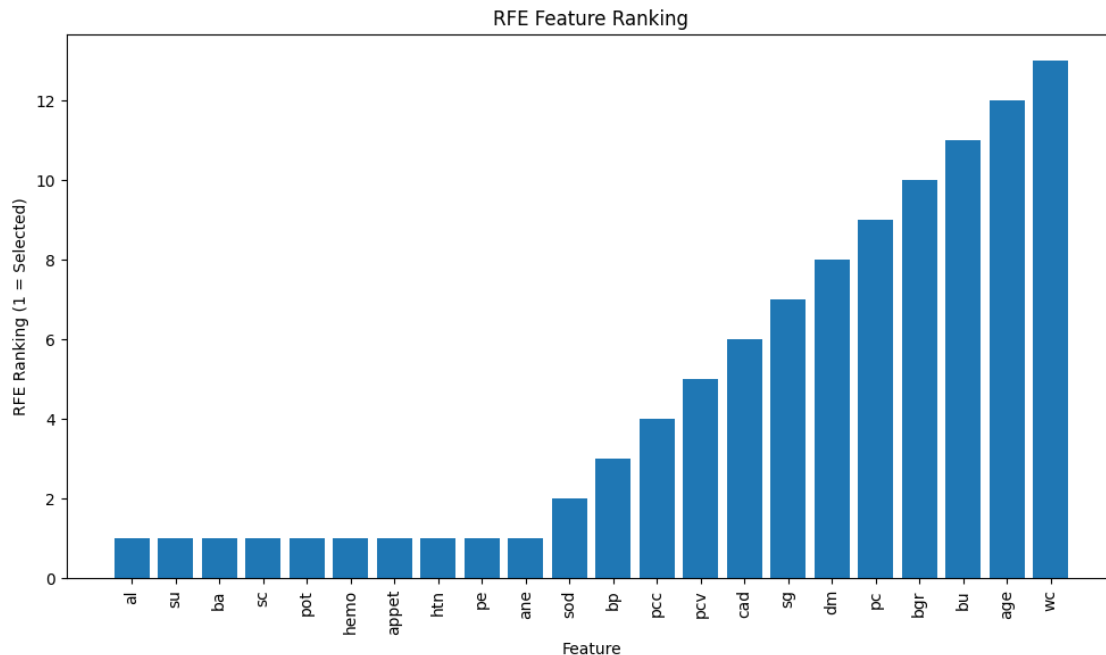
	precision	recall	f1-score	support
0	0.93	0.74	0.82	76
1	0.67	0.91	0.77	44
accuracy			0.80	120
macro avg	0.80	0.82	0.80	120
weighted avg	0.84	0.80	0.80	120

Model Performance:

	Model	Accuracy	Precision	Recall
0	Random Forest	0.975000	0.976596	0.975000
1	Decision Tree	0.933333	0.934469	0.933333
2	SVM	0.800000	0.835556	0.800000

Recursive Feature Elimination (RFE) with Logistic Regression

```
[49]: model = LogisticRegression(max_iter=500)
rfe = RFE(model, n_features_to_select=10)
rfe.fit(X, y)
feature_ranking = pd.DataFrame({
    'Feature': X.columns,
    'Ranking': rfe.ranking_
})
feature_ranking = feature_ranking.sort_values(by='Ranking',
ascending=True)
plt.figure(figsize=(10, 6))
plt.bar(feature_ranking['Feature'], feature_ranking['Ranking'])
plt.xlabel("Feature")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("RFE Feature Ranking")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



Comparison of models using features selected by RFE with Logistic Regression

```
[50]: # 2. TRAIN / TEST SPLIT
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=42
)
model = LogisticRegression(max_iter=500)
rfe = RFE(model, n_features_to_select=10)
rfe.fit(X, y)
X_train_k = rfe.fit_transform(X_train, y_train)
X_test_k = rfe.transform(X_test)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", list(selected_features))

# 4. MODELS
models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf', random_state=42)
}
metrics = []

# 5. TRAIN + EVALUATE
for name, model in models.items():
    model.fit(X_train_k, y_train)
```

```

y_pred = model.predict(X_test_k)
print(f"\n==== Classification Report: {name} =====")
print(classification_report(y_test, y_pred))
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')
})

# 6. RESULTS AS DATAFRAME
results_df = pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_df)

```

Selected Features: ['age', 'al', 'su', 'bgr', 'bu', 'sc', 'hemo', 'pcv', 'wc', 'htn']

==== Classification Report: Random Forest =====

	precision	recall	f1-score	support
0	0.99	0.97	0.98	76
1	0.96	0.98	0.97	44
accuracy			0.97	120
macro avg	0.97	0.98	0.97	120
weighted avg	0.98	0.97	0.98	120

==== Classification Report: Decision Tree =====

	precision	recall	f1-score	support
0	0.99	0.91	0.95	76
1	0.86	0.98	0.91	44
accuracy			0.93	120
macro avg	0.92	0.94	0.93	120
weighted avg	0.94	0.93	0.93	120

==== Classification Report: SVM =====

	precision	recall	f1-score	support
0	0.63	1.00	0.78	76
1	0.00	0.00	0.00	44

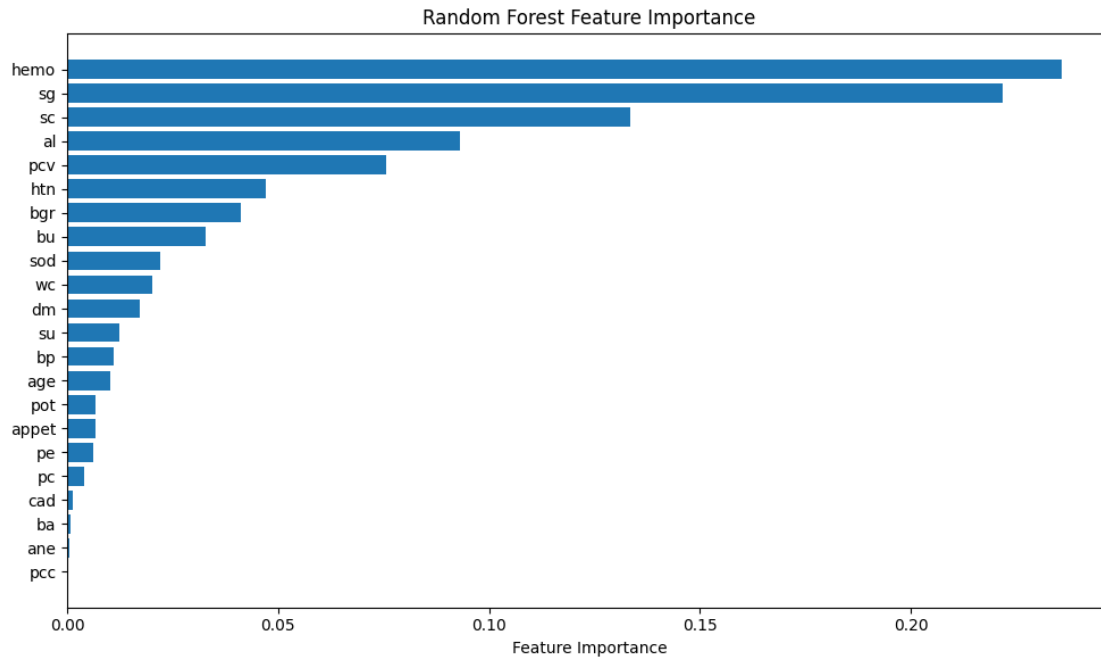
accuracy			0.63	120
macro avg	0.32	0.50	0.39	120
weighted avg	0.40	0.63	0.49	120

Model Performance:

	Model	Accuracy	Precision	Recall
0	Random Forest	0.975000	0.975259	0.975000
1	Decision Tree	0.933333	0.939619	0.933333
2	SVM	0.633333	0.401111	0.633333

Random Forest Feature Selection

```
[52]: # Train Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)
# Extract importances
importances = rf.feature_importances_
# Use feature names from X (correct and safe)
feature_names = X.columns if hasattr(X, "columns") else [f"Feature{i}" for i in
↳range(X.shape[1])]
# Create DataFrame
df_importance = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
})
# Sort by importance
df_importance = df_importance.sort_values(by='Importance',
ascending=False)
# Plot
plt.figure(figsize=(10, 6))
plt.barh(df_importance['Feature'][::-1], df_importance['Importance']
[:::-1])
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.tight_layout()
plt.show()
```



Comparison of models using features selected by RFE with Random Forest

```
[56]: # 1. TRAIN/TEST SPLIT
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 2. SELECT FEATURES USING RANDOM FOREST IMPORTANCE
k = 18 # number of top important features to keep
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)
# Get importance scores
importances = rf.feature_importances_
# Sort indices by importance (descending)
sorted_idx = np.argsort(importances)[::-1]
# Select top-k feature indices
top_k_idx = sorted_idx[:k]
# Get feature names
selected_features = X.columns[top_k_idx]
print("Selected Features by RF Importance:", list(selected_features))
# Reduce datasets to selected features
X_train_k = X_train[selected_features]
X_test_k = X_test[selected_features]

# 3. MODELS TO COMPARE
models = {
    "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42),
```

```

    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}
metrics = []
# 4. TRAIN + EVALUATE

for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)
    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'),
        "F1 Score": f1_score(y_test, y_pred, average='weighted')
    })
# 5. RESULTS DATAFRAME

results_df = pd.DataFrame(metrics)
print("\nModel Performance Using RF-Selected Features:")
results_df

```

Selected Features by RF Importance: ['hemo', 'sg', 'sc', 'al', 'bgr', 'pcv', 'htn', 'sod', 'wc', 'bu', 'dm', 'bp', 'su', 'pe', 'age', 'appet', 'pot', 'pc']

Model Performance Using RF-Selected Features:

```

[56]:
      Model  Accuracy  Precision  Recall  F1 Score
0  Random Forest    0.9875    0.987931  0.9875    0.987549
1  Decision Tree    0.9625    0.966129  0.9625    0.962896
2         SVM      0.7625    0.826493  0.7625    0.767574

```

imbalanced methods

```

[53]: models = {
    "SVM": Pipeline([
        ("scaler", StandardScaler()),
        ("clf", SVC(kernel="rbf", probability=True, random_state=42))
    ]),
    "Random Forest": RandomForestClassifier(
        n_estimators=200,
        random_state=42,
        n_jobs=-1
    ),
    "Decision Tree": DecisionTreeClassifier(
        random_state=42
    )
}

```

```

# Evaluation Function (MULTICLASS SAFE)
def evaluate_models(Xt, yt, method_name):
    print(f"\n{'='*15} {method_name} {'='*15}")
    results = []
    for model_name, model in models.items():
        print(f"\n--- {model_name} ---")
        model.fit(Xt, yt)
        preds = model.predict(X_test)
        acc = accuracy_score(y_test, preds)
        prec = precision_score(y_test, preds, average="weighted",
zero_division=0)
        rec = recall_score(y_test, preds, average="weighted",
zero_division=0)
        f1 = f1_score(y_test, preds, average="weighted",
zero_division=0)
        print(classification_report(y_test, preds))
        results.append({
            "Sampling": method_name,
            "Model": model_name,
            "Accuracy": acc,
            "Precision": prec,
            "Recall": rec,
            "F1-score": f1
        })
    return results
# Run Experiments
all_results = []
all_results += evaluate_models(
    X_train, y_train,
    "Original Imbalanced Data"
)
ros = RandomOverSampler(random_state=42)
X_ros, y_ros = ros.fit_resample(X_train, y_train)
all_results += evaluate_models(
    X_ros, y_ros,
    "Random Oversampling"
)
rus = RandomUnderSampler(random_state=42)
X_rus, y_rus = rus.fit_resample(X_train, y_train)
all_results += evaluate_models(
    X_rus, y_rus,
    "Random Undersampling"
)
smote = SMOTE(random_state=42)
X_sm, y_sm = smote.fit_resample(X_train, y_train)
all_results += evaluate_models(
    X_sm, y_sm,

```

```

        "SMOTE Oversampling"
    )
    smt = SMOTETomek(random_state=42)
    X_smt, y_smt = smt.fit_resample(X_train, y_train)
    all_results += evaluate_models(
        X_smt, y_smt,
        "SMOTE + Tomek Links"
    )
# Final Comparison Table

results_df = pd.DataFrame(all_results)
summary = results_df.pivot_table(
    index="Sampling",
    columns="Model",
    values=["Accuracy", "Precision", "Recall", "F1-score"]
).round(4)
print("\n\n===== FINAL COMPARISON =====")
print(summary)

```

===== Original Imbalanced Data =====

--- SVM ---

	precision	recall	f1-score	support
0	0.99	0.96	0.97	76
1	0.93	0.98	0.96	44
accuracy			0.97	120
macro avg	0.96	0.97	0.96	120
weighted avg	0.97	0.97	0.97	120

--- Random Forest ---

	precision	recall	f1-score	support
0	1.00	0.99	0.99	76
1	0.98	1.00	0.99	44
accuracy			0.99	120
macro avg	0.99	0.99	0.99	120
weighted avg	0.99	0.99	0.99	120

--- Decision Tree ---

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

0	1.00	0.93	0.97	76
1	0.90	1.00	0.95	44
accuracy			0.96	120
macro avg	0.95	0.97	0.96	120
weighted avg	0.96	0.96	0.96	120

===== Random Oversampling =====

--- SVM ---

	precision	recall	f1-score	support
0	0.99	0.93	0.96	76
1	0.90	0.98	0.93	44
accuracy			0.95	120
macro avg	0.94	0.96	0.95	120
weighted avg	0.95	0.95	0.95	120

--- Random Forest ---

	precision	recall	f1-score	support
0	1.00	0.99	0.99	76
1	0.98	1.00	0.99	44
accuracy			0.99	120
macro avg	0.99	0.99	0.99	120
weighted avg	0.99	0.99	0.99	120

--- Decision Tree ---

	precision	recall	f1-score	support
0	1.00	0.96	0.98	76
1	0.94	1.00	0.97	44
accuracy			0.97	120
macro avg	0.97	0.98	0.97	120
weighted avg	0.98	0.97	0.98	120

===== Random Undersampling =====

--- SVM ---

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

0	0.99	0.89	0.94	76
1	0.84	0.98	0.91	44
accuracy			0.93	120
macro avg	0.91	0.94	0.92	120
weighted avg	0.93	0.93	0.93	120

--- Random Forest ---

	precision	recall	f1-score	support
0	1.00	0.99	0.99	76
1	0.98	1.00	0.99	44
accuracy			0.99	120
macro avg	0.99	0.99	0.99	120
weighted avg	0.99	0.99	0.99	120

--- Decision Tree ---

	precision	recall	f1-score	support
0	1.00	0.95	0.97	76
1	0.92	1.00	0.96	44
accuracy			0.97	120
macro avg	0.96	0.97	0.96	120
weighted avg	0.97	0.97	0.97	120

===== SMOTE Oversampling =====

--- SVM ---

	precision	recall	f1-score	support
0	0.99	0.95	0.97	76
1	0.91	0.98	0.95	44
accuracy			0.96	120
macro avg	0.95	0.96	0.96	120
weighted avg	0.96	0.96	0.96	120

--- Random Forest ---

	precision	recall	f1-score	support
0	1.00	0.99	0.99	76
1	0.98	1.00	0.99	44

accuracy			0.99	120
macro avg	0.99	0.99	0.99	120
weighted avg	0.99	0.99	0.99	120

--- Decision Tree ---

	precision	recall	f1-score	support
0	1.00	0.93	0.97	76
1	0.90	1.00	0.95	44

accuracy			0.96	120
macro avg	0.95	0.97	0.96	120
weighted avg	0.96	0.96	0.96	120

===== SMOTE + Tomek Links =====

--- SVM ---

	precision	recall	f1-score	support
0	0.99	0.93	0.96	76
1	0.90	0.98	0.93	44

accuracy			0.95	120
macro avg	0.94	0.96	0.95	120
weighted avg	0.95	0.95	0.95	120

--- Random Forest ---

	precision	recall	f1-score	support
0	1.00	0.99	0.99	76
1	0.98	1.00	0.99	44

accuracy			0.99	120
macro avg	0.99	0.99	0.99	120
weighted avg	0.99	0.99	0.99	120

--- Decision Tree ---

	precision	recall	f1-score	support
0	1.00	0.93	0.97	76
1	0.90	1.00	0.95	44

accuracy			0.96	120
----------	--	--	------	-----

macro avg	0.95	0.97	0.96	120
weighted avg	0.96	0.96	0.96	120

===== FINAL COMPARISON =====

Model	Accuracy		F1-score \	
	Decision Tree	Random Forest	SVM	Decision Tree
Sampling				
Original Imbalanced Data	0.9583	0.9917	0.9667	0.9587
Random Oversampling	0.9750	0.9917	0.9500	0.9752
Random Undersampling	0.9667	0.9917	0.9250	0.9669
SMOTE + Tomek Links	0.9583	0.9917	0.9500	0.9587
SMOTE Oversampling	0.9583	0.9917	0.9583	0.9587

Model	Precision \			
	Random Forest	SVM	Decision Tree	Random Forest
Sampling				
Original Imbalanced Data	0.9917	0.9668	0.9626	0.9919
Random Oversampling	0.9917	0.9504	0.9766	0.9919
Random Undersampling	0.9917	0.9260	0.9694	0.9919
SMOTE + Tomek Links	0.9917	0.9504	0.9626	0.9919
SMOTE Oversampling	0.9917	0.9586	0.9626	0.9919

Model	Recall			
	SVM	Decision Tree	Random Forest	SVM
Sampling				
Original Imbalanced Data	0.9675	0.9583	0.9917	0.9667
Random Oversampling	0.9530	0.9750	0.9917	0.9500
Random Undersampling	0.9333	0.9667	0.9917	0.9250
SMOTE + Tomek Links	0.9530	0.9583	0.9917	0.9500
SMOTE Oversampling	0.9601	0.9583	0.9917	0.9583