Cardiovascular Disease Detectoni and Analysis using known Machine Learning Algorithms

Student Name: Sakib-Ul-Ahsan
ID: 20-42978-1
Dept Name: CSE
Institute Name: AIUB
City, Country: Dhaka, Bangladesh
email address:
sakib.ahsan.35@gmail.com

Student Name: Md. Minhazul Bari Fahim
ID: 20-42176-1
Dept Name: CSE
Institute Name: AIUB
City, Country: Dhaka, Bangladesh
email address:
sakib.ahsan.35@gmail.com

Abstract:

Cardiovascular diseases (CVD) represents a significant global health concern with high mortality rates. Accurate classification of CVD is crucial because any misidentification can lead to severe outcomes. Early detection and precise characterization of the disease are vital for effective treatment strategies. So, in this research there will be an attempt to analyze predictions make by known models to understand which model is the most accurate in determining CVD. Among all the models Random Forest had the highest accuracy of 72%. Further research and validation should be conducted to validate the proposed approach

I. Introduction

The prevalence of cardiovascular diseases (CVD) presents a critical challenge to global health, contributing to substantial mortality rates. Ensuring the accurate classification of CVD is of paramount importance, as any erroneous identification can result in grave consequences. Therefore, the focus of this research revolves around the analysis of predictions generated by established models, aiming to discern the model that exhibits the highest accuracy in CVD determination.

This study seeks to address the pressing need for robust classification methodologies in the context of cardiovascular diseases. By assessing the predictive capabilities of various established models, the research aims to shed light on the most accurate model for discerning the presence and characteristics of CVD. Notably, the outcomes of these analyses reveal that among all considered models, the Random Forest model emerges as the leader, boasting an accuracy of 72% in CVD classification.

While this research provides promising insights into model performance, it also underscores the necessity for further investigations to validate and refine the proposed approach. Through rigorous validation and refinement, the research community can pave the way for more effective and reliable strategies in the early detection and characterization of cardiovascular diseases.

II. Motivation of the Project

The motivation behind this research stems from the need for more efficient and accurate methods of CVD detection. Traditional diagnostic approaches often rely on manual interpretation and subjective judgment, which can be time-consuming and prone to human error. ML algorithms, with their ability to learn patterns and make predictions from large datasets, offer a promising solution to enhance CVD detection and analysis. By leveraging ML algorithms, we can potentially identify novel risk factors, improve risk stratification, and provide personalized treatment plans.

The general aim of this thesis is to contribute to the understanding of cardiovascular disease by developing and applying ML algorithms for CVD detection and analysis. By harnessing the power of ML, we aim to improve the accuracy, efficiency, and effectiveness of CVD diagnosis, risk assessment, and treatment planning. Our research seeks to bridge the gap between traditional diagnostic approaches and advanced computational techniques to enhance patient care and outcomes.

III. OBJECTIVE OF THE PROJECT

Some of the objectives of this project is described below:

I. **Comparative Model Evaluation**: This research aims to assess the comparative performance of prominent machine learning models including Random Forest, Decision Tree, KNN, Naïve Bayes and SVM.

II. **Precision of Disease Classification:** The primary goal of this study is to find the model that has the highest precision in categorizing different types of cardiovascular diseases.

III. **Early Detection Potential:** Investigating the early detection capabilities of the identified models is a significant focus of this project.

IV. **Feature importance:** Insights into the feature importance hold potential for informing medical practitioners about key indicators of CVD.

V. **Validation & Optimization:** It also places emphasis on the validation and optimization of the chosen models.

IV. METHODOLOGY

A review of the pre-existing literature it is evident that current studies focus on the detection and analysis of cardiovascular diseases (CVD). In this study, a system is proposed to contribute to this particular area**.** The proposed system aims to enhance the accuracy of CVD prediction by the proper use of machine learning models and balancing dataset. By considering multiple ML algorithms, we can determine the most effective approach for CVD analysis.

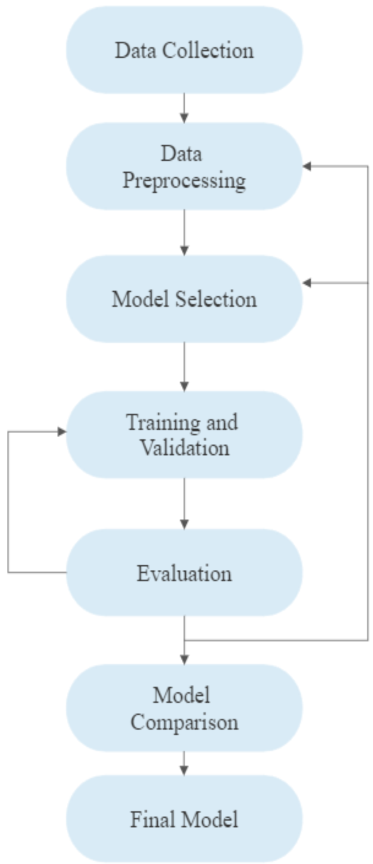


Fig: Flowchart of this study.

A. Data Collection

The dataset used in this study was collected from Kaggle, a well-known platform for data science and machine learning resources. The dataset is called "Cardiovascular Disease Dataset".

B. Data processing

The dataset was searched for missing values at first. The there was an attempt to look for any outliers. But none was found. So, we jumped to the feature selection phase. The attribute named "id" had no relevance to whether or not the person might have CVD or not. So that particular attribute was discarded. Then the age attribute was given in days which would make the calculations for the ML models slower and more complex. So, it was converted into years by dividing with 365.

C. Dataset description

After data pre-processing the dataset comprised of 12 attributes with the target attribute too. The attributes include age, gender, height, weight, systolic blood pressure, diastolic blood pressure, glucose, smoking status, alcohol, active status, cardio. The last attribute "cardio" is the target attribute which has two values(0 and 1). 0 refers to the person not having CVD and 1 says the opposite. There are 70,000 instances and 13 columns.

D. Machine Learning model development and evaluation

1. Importing all the libraries and then loading the dataset and printing the headers.

+ Code + Text

```
import pandas as pd
import numpy as np

import seaborn as sns
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, auc
from sklearn.metrics import precision_score, recall_score, f1_score, precision_recall_curve, f1_score
import matplotlib.pyplot as plt
```

```
Mounted at /content/drive
```

```
data = pd.read_csv('/content/cardio_train.csv', sep=';')
data.head()
```

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

2. Dataset Description and looking for missing values

```
[10] data.describe()
```

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
count	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000
mean	49972.419900	19468.865814	1.349571	164.359229	74.205690	128.817286	96.630414	1.366871	1.226457	0.088129	0.053771	0.803729	0.499700
std	28851.302323	2467.251667	0.476838	8.210126	14.395757	154.011419	188.472530	0.680250	0.572270	0.283484	0.225568	0.397179	0.500003
min	0.000000	10798.000000	1.000000	55.000000	10.000000	-150.000000	-70.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	25006.750000	17664.000000	1.000000	159.000000	65.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000
50%	50001.500000	19703.000000	1.000000	165.000000	72.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000
75%	74889.250000	21327.000000	2.000000	170.000000	82.000000	140.000000	90.000000	2.000000	1.000000	0.000000	0.000000	1.000000	1.000000
max	99999.000000	23713.000000	2.000000	250.000000	200.000000	16020.000000	11000.000000	3.000000	3.000000	1.000000	1.000000	1.000000	1.000000

```
null_values = data.isnull().sum()
null_values
```

```
id             0
age            0
gender         0
height         0
weight         0
ap_hi          0
ap_lo          0
cholesterol    0
gluc           0
smoke          0
alco           0
active         0
cardio         0
dtype: int64
```

3. Data Pre-processing: Removing the “id” attribute and converting age to years(was given in days)

```
[ ] data.loc[:, 'age'] = data['age'] / 365
    data.drop('id', axis=1, inplace=True)
    # Print the modified DataFrame
    print(data)
```

```
             age  gender  height  weight  ap_hi  ap_lo  cholesterol  gluc  \
0      50.391781       2     168    62.0    110     80            1     1
1      55.419178       1     156    85.0    140     90            3     1
2      51.663014       1     165    64.0    130     70            3     1
3      48.282192       2     169    82.0    150    100            1     1
4      47.873973       1     156    56.0    100     60            1     1
...          ...     ...     ...     ...    ...    ...          ...   ...
69995  52.712329       2     168    76.0    120     80            1     1
69996  61.920548       1     158   126.0    140     90            2     2
69997  52.235616       2     183   105.0    180     90            3     1
69998  61.454795       1     163    72.0    135     80            1     2
69999  56.273973       1     170    72.0    120     80            2     1

       smoke  alco  active  cardio
0          0     0       1       0
1          0     0       1       1
2          0     0       0       1
3          0     0       1       1
4          0     0       0       0
...      ...   ...     ...     ...
69995      1     0       1       0
69996      0     0       1       1
69997      0     1       0       1
69998      0     0       0       1
69999      0     0       1       0
```

4. Separating the features and keeping it in X and the target variable in y

```
X= data.drop(['cardio'], axis=1)
y = data['cardio']
print(X,y)
```

```
             age  gender  height  weight  ap_hi  ap_lo  cholesterol  gluc  \
0      50.391781       2     168    62.0    110     80            1     1
1      55.419178       1     156    85.0    140     90            3     1
2      51.663014       1     165    64.0    130     70            3     1
3      48.282192       2     169    82.0    150    100            1     1
4      47.873973       1     156    56.0    100     60            1     1
...          ...     ...     ...     ...    ...    ...          ...   ...
69995  52.712329       2     168    76.0    120     80            1     1
69996  61.920548       1     158   126.0    140     90            2     2
69997  52.235616       2     183   105.0    180     90            3     1
69998  61.454795       1     163    72.0    135     80            1     2
69999  56.273973       1     170    72.0    120     80            2     1
```

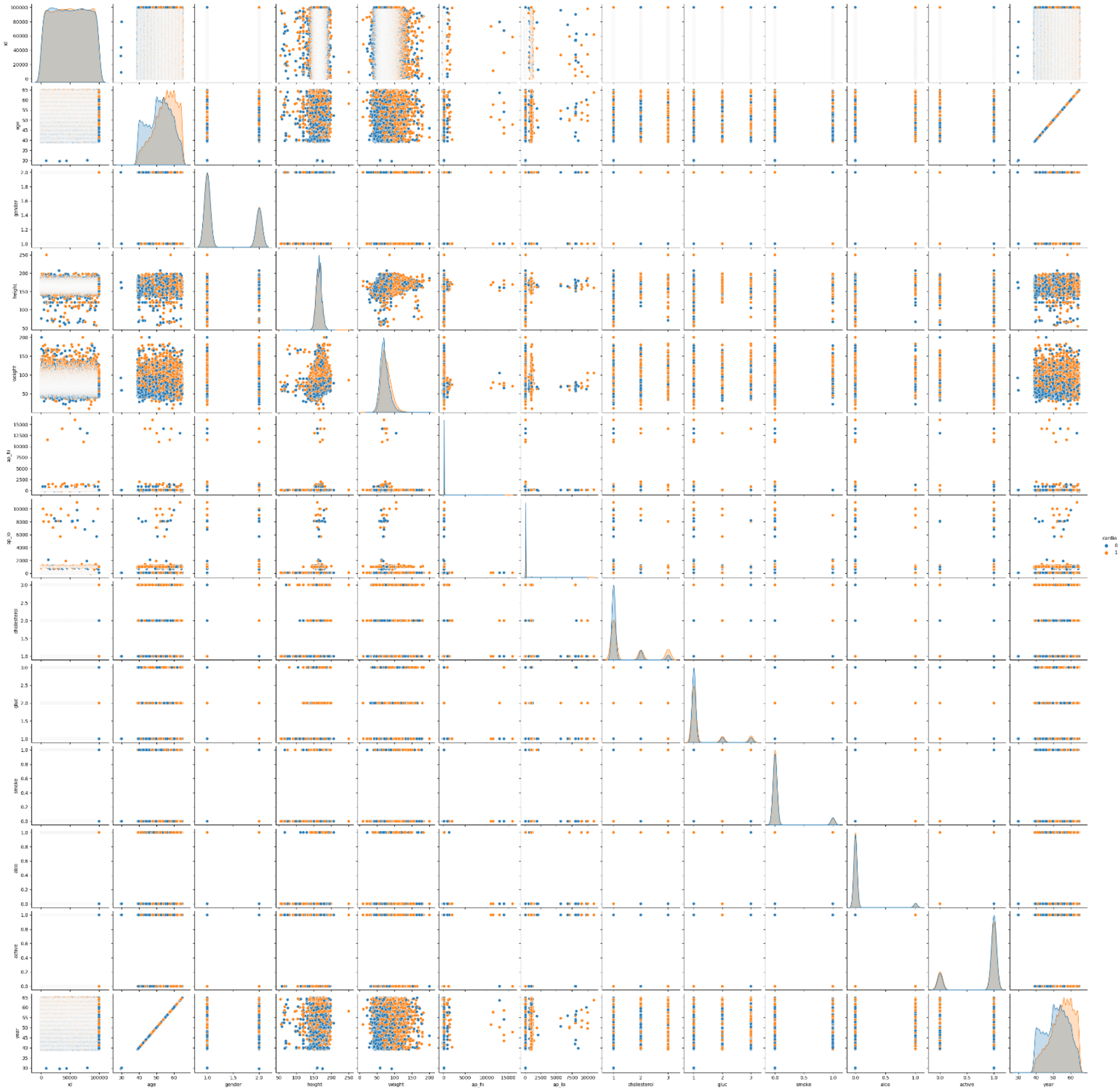
5. Creating summary of the data:

```python
# Step : Data Summary
print("Data Summary:")
print(data.head())
print("\nData Info:")
print(data.info())
print("\nData Statistics:")
print(data.describe())

# Step : Data Visualization and EDA
# Pairplot for visualizing relationships between features
sns.pairplot(data, hue='cardio', diag_kind='kde')
plt.show()

# Correlation Heatmap
corr = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()

# Distribution of the target variable 'Outcome'
sns.countplot(data['cardio'])
plt.title('Distribution of cardio')
plt.show()

# Histograms of numerical features
data.hist(figsize=(12, 10), bins=30)
plt.suptitle("Histograms of Numerical Features", y=1.02)
```

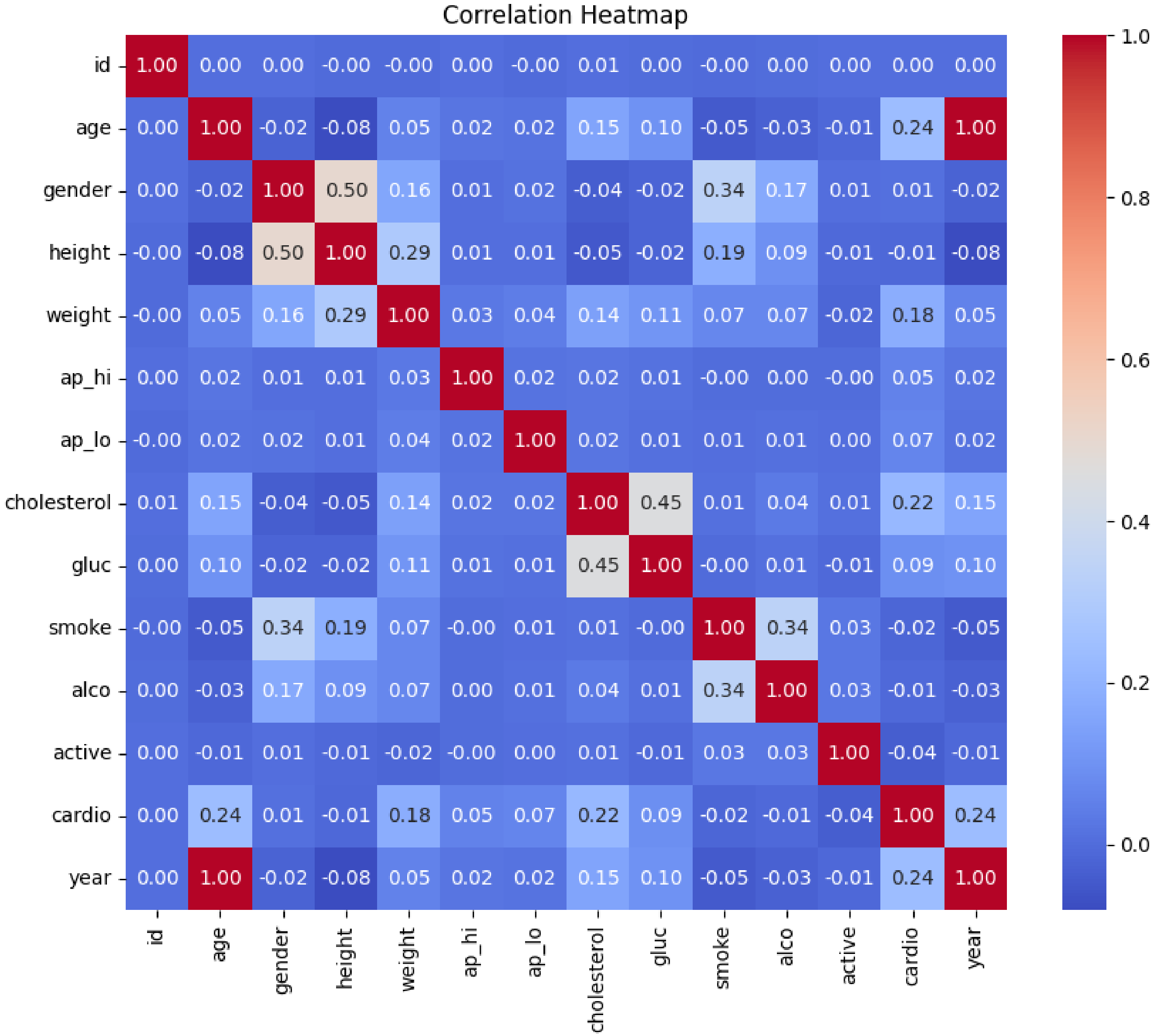
```python
    plt.subplot(2, 4, i)
    sns.boxplot(x='cardio', y=feature, data=data)
    plt.title(f'{feature} by cardio')
plt.tight_layout()
plt.show()
```

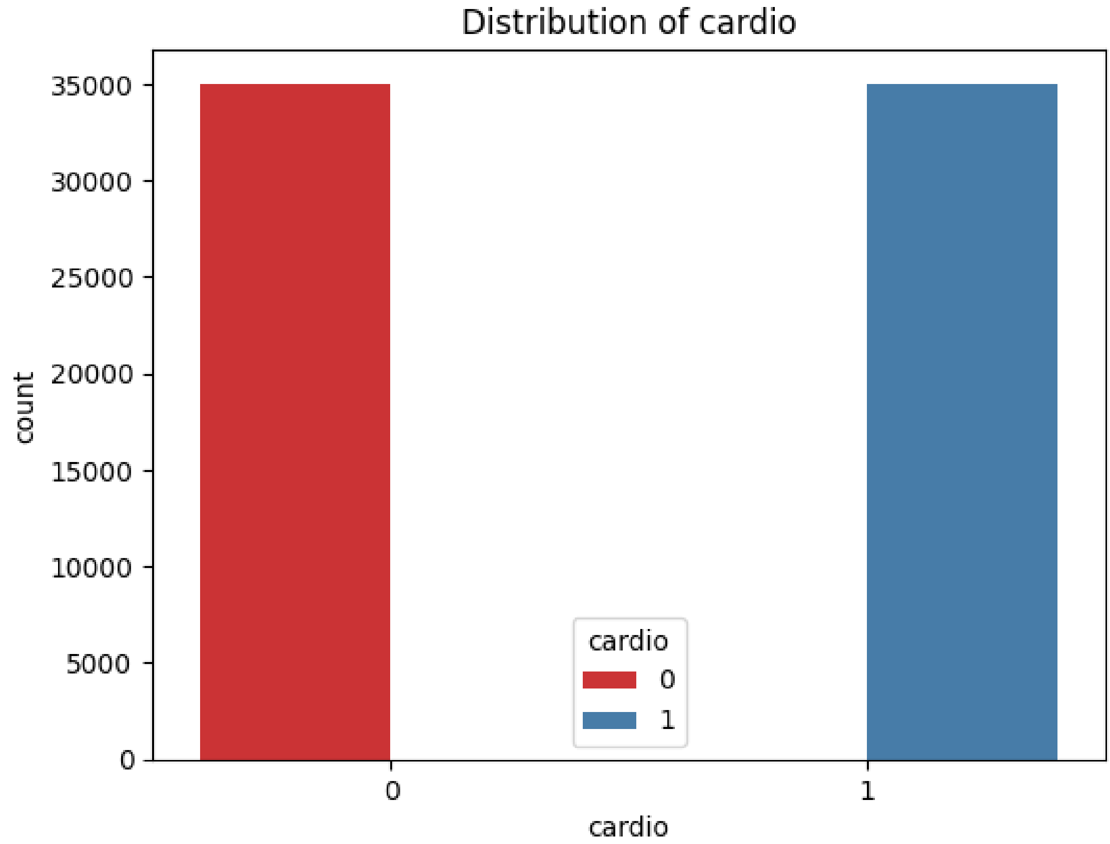
```
Data Summary:
   id        age  gender  height  weight  ap_hi  ap_lo  cholesterol  gluc  \
0   0  50.391781       2     168    62.0    110     80            1     1
1   1  55.419178       1     156    85.0    140     90            3     1
2   2  51.663014       1     165    64.0    130     70            3     1
3   3  48.282192       2     169    82.0    150    100            1     1
4   4  47.873973       1     156    56.0    100     60            1     1

   smoke  alco  active  cardio       year
0      0     0       1       0  50.391781
1      0     0       1       1  55.419178
2      0     0       0       1  51.663014
3      0     0       1       1  48.282192
4      0     0       0       0  47.873973

Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   id           70000 non-null  int64
 1   age          70000 non-null  float64
```

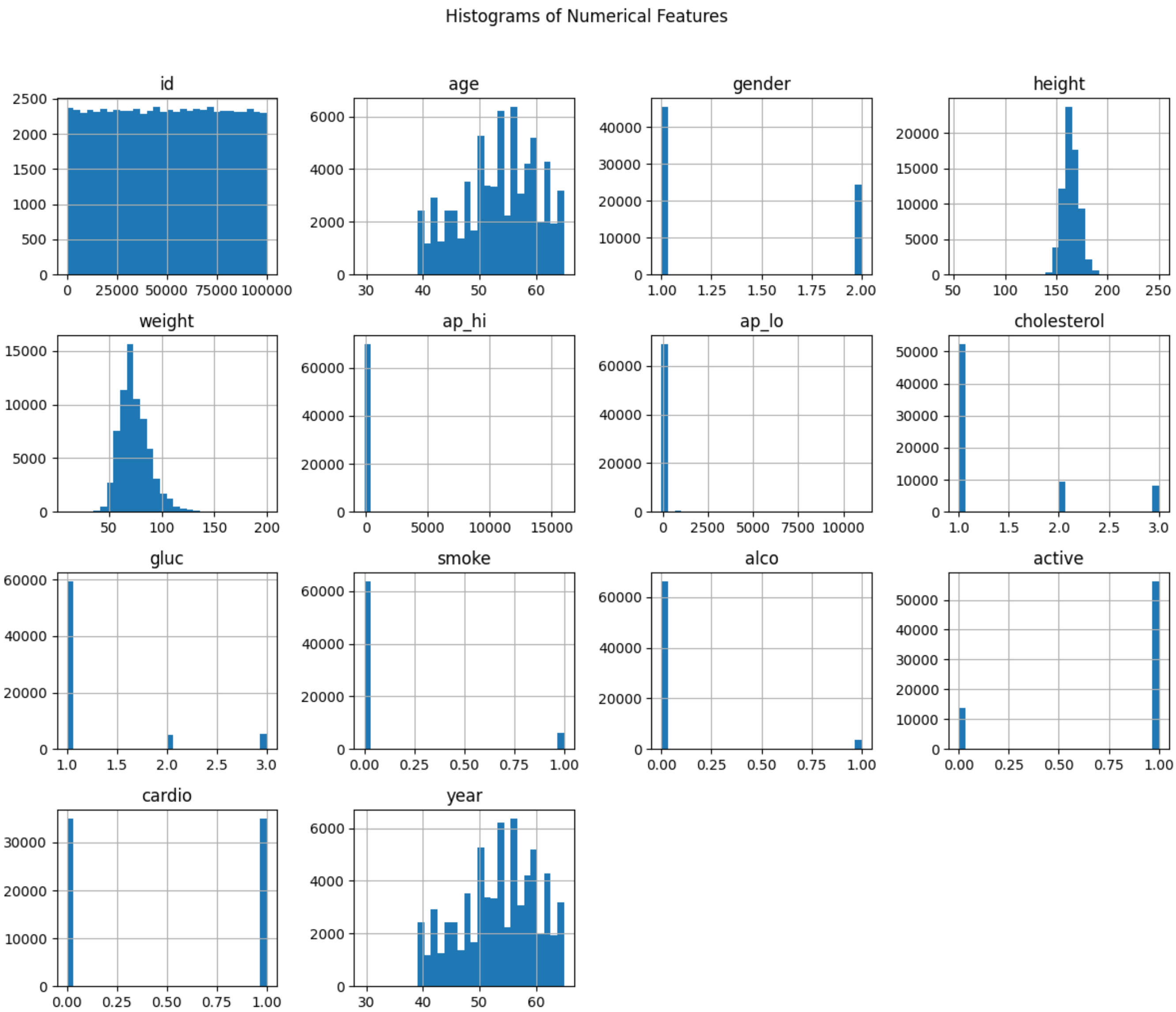
```
Data Statistics:
                 id           age        gender        height        weight  \
count  70000.000000  70000.000000  70000.000000  70000.000000  70000.000000
mean   49972.419900     53.339358      1.349571    164.359229     74.205690
std    28851.302323      6.759594      0.476838      8.210126     14.395757
min        0.000000     29.583562      1.000000     55.000000     10.000000
25%    25006.750000     48.394521      1.000000    159.000000     65.000000
50%    50001.500000     53.980822      1.000000    165.000000     72.000000
75%    74889.250000     58.430137      2.000000    170.000000     82.000000
max    99999.000000     64.967123      2.000000    250.000000    200.000000

              ap_hi         ap_lo   cholesterol          gluc         smoke  \
count  70000.000000  70000.000000  70000.000000  70000.000000  70000.000000
mean     128.817286     96.630414      1.366871      1.226457      0.088129
std      154.011419    188.472530      0.680250      0.572270      0.283484
min     -150.000000    -70.000000      1.000000      1.000000      0.000000
25%      120.000000     80.000000      1.000000      1.000000      0.000000
50%      120.000000     80.000000      1.000000      1.000000      0.000000
75%      140.000000     90.000000      2.000000      1.000000      0.000000
max    16020.000000  11000.000000      3.000000      3.000000      1.000000
```

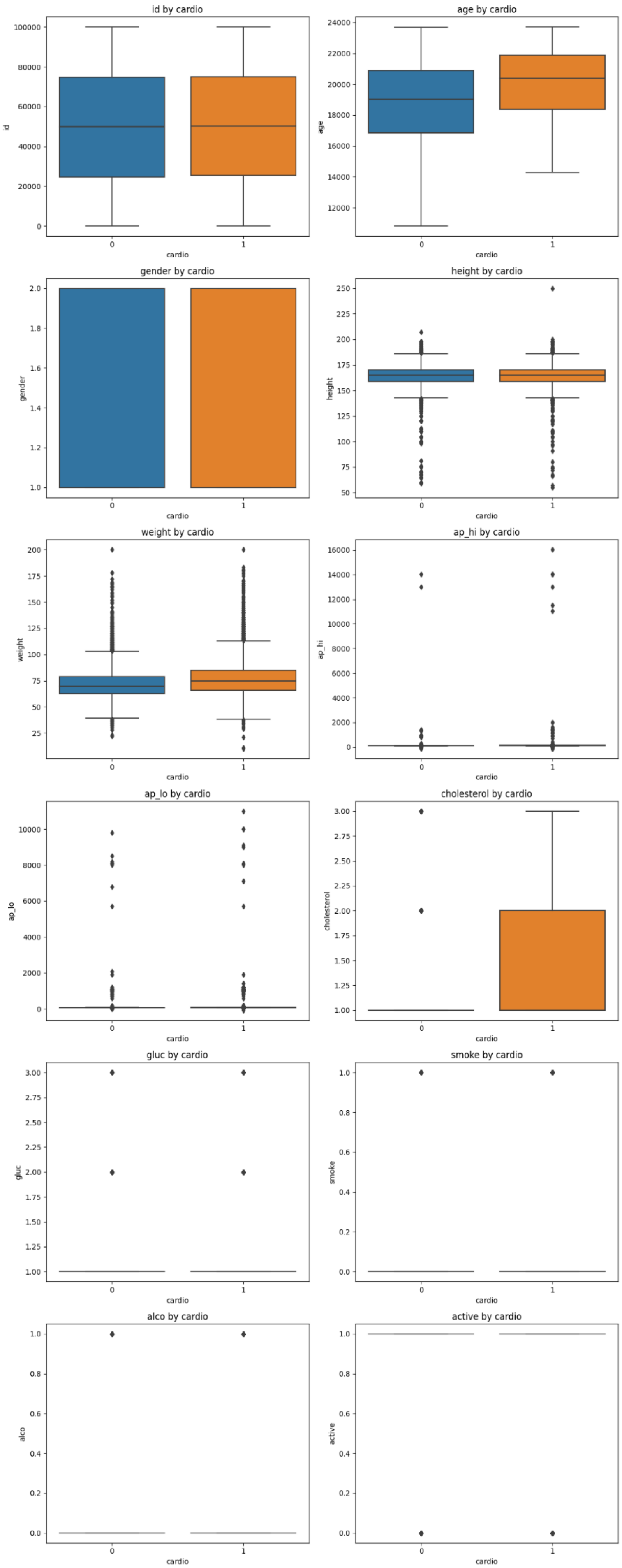


Pairplot

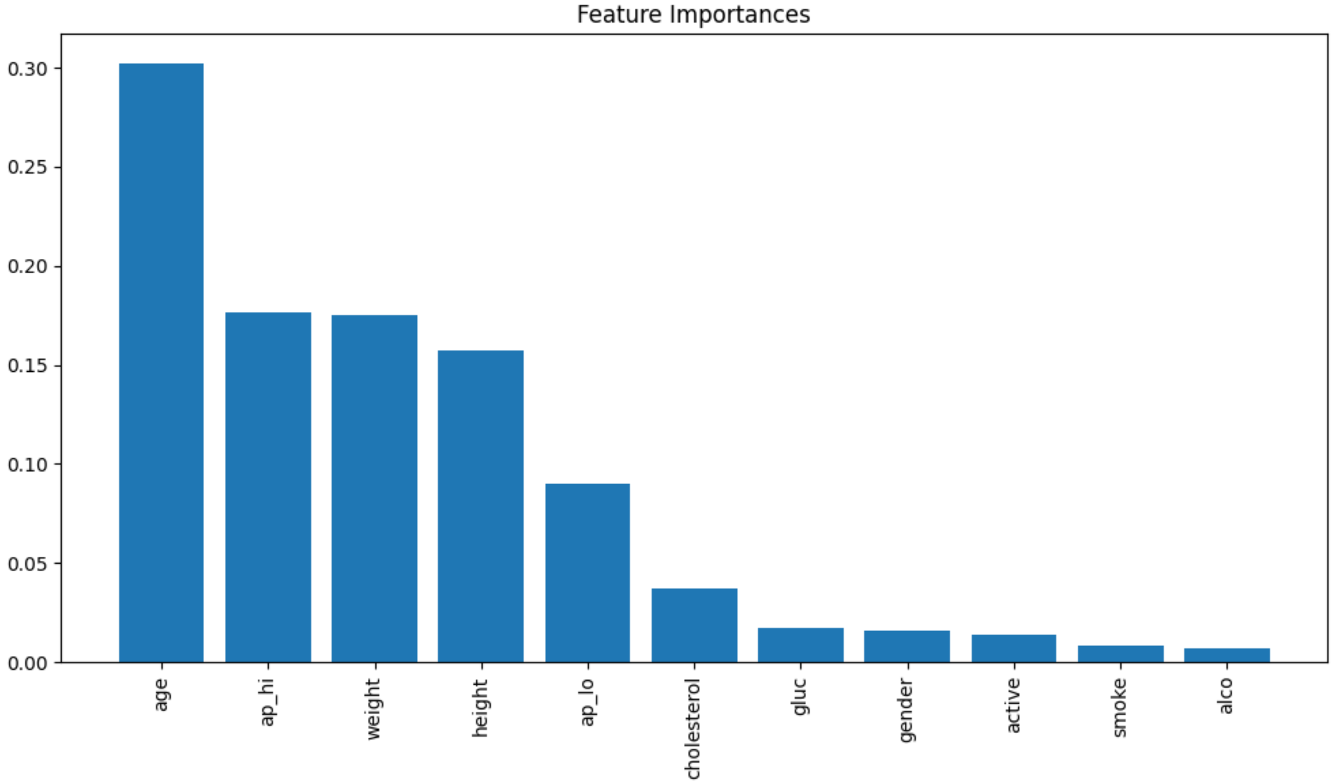

Correlation Heatmap
id
age
gender
height
weight
ap_hi
ap_lo
cholesterol
gluc
smoke
alco
active
cardio
year
1.0
0.8
0.6
0.4
0.2
0.0


Distribution of cardio
35000
30000
25000
20000
15000
10000
5000
0
count
cardio
0
1
0
1
cardio


Histograms of Numerical Features
id
age
gender
height
weight
ap_hi
ap_lo
cholesterol
gluc
smoke
alco
active
cardio
year


id by cardio
id
cardio
age by cardio
age
cardio
gender by cardio
gender
cardio
height by cardio
height
cardio
weight by cardio
weight
cardio
ap_hi by cardio
ap_hi
cardio
ap_lo by cardio
ap_lo
cardio
cholesterol by cardio
cholesterol
cardio
gluc by cardio
gluc
cardio
smoke by cardio
smoke
cardio
alco by cardio
alco
cardio
active by cardio
active
cardio

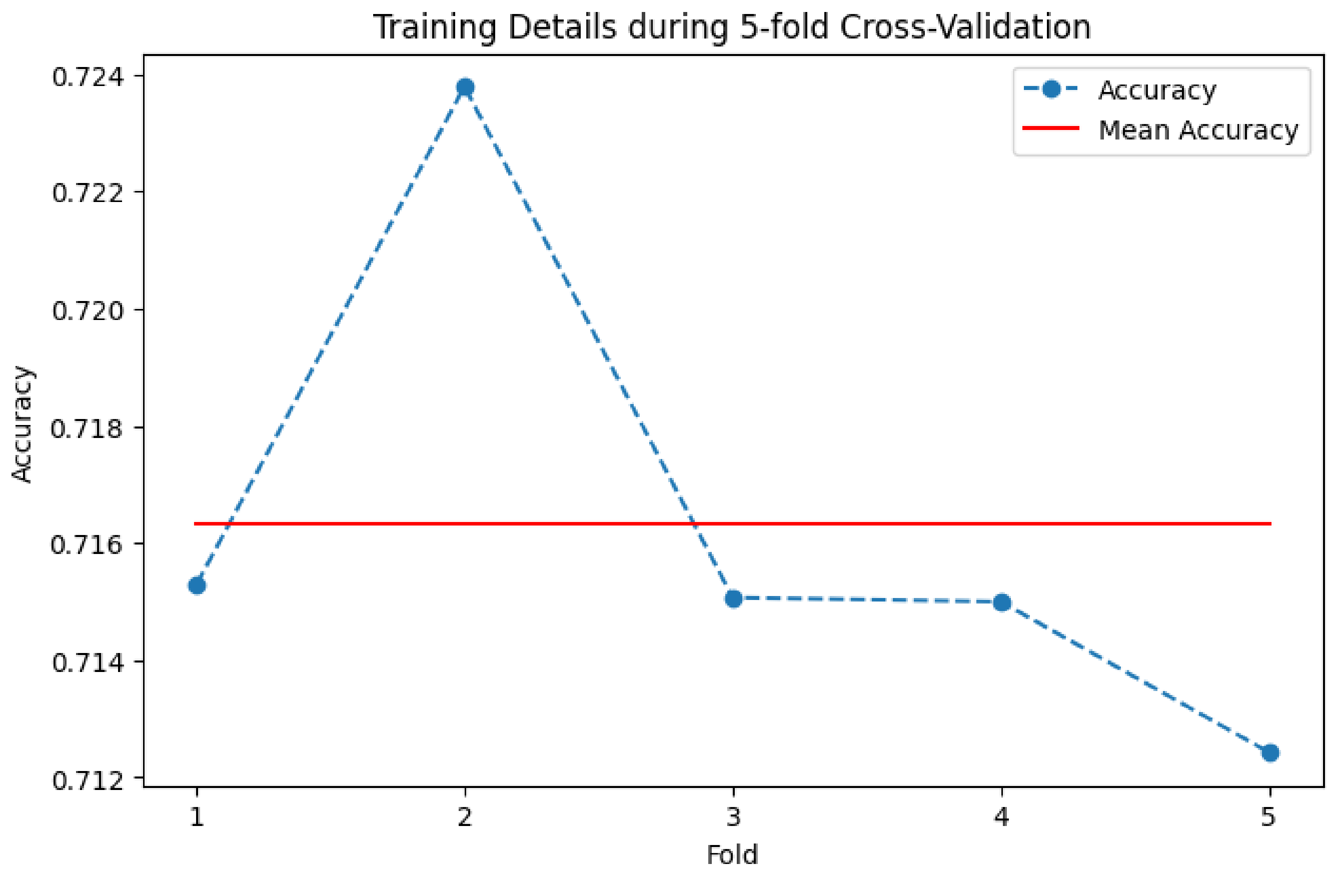
6. Feature importance



7. Accuracies of different ML models

a. Random Forest:

```
[ ] rf_classifier = RandomForestClassifier()
    cv_scores = cross_val_score(rf_classifier, X, y, cv=5, scoring='accuracy')
    cv_predictions = cross_val_predict(rf_classifier, X, y, cv=5)

    # Calculate mean cross-validation score
    mean_cv_score = np.mean(cv_scores)

    print("Cross-Validation Scores:", cv_scores)
    print("Mean Cross-Validation Score:", mean_cv_score)

    # Plot training details
    plt.figure(figsize=(8, 5))
    plt.plot(range(1, 6), cv_scores, marker='o', linestyle='--', label='Accuracy')
    plt.plot(range(1, 6), [mean_cv_score] * 5, color='r', linestyle='-', label='Mean Accuracy')
    plt.xticks(range(1, 6))
    plt.xlabel('Fold')
    plt.ylabel('Accuracy')
    plt.title('5-fold Cross-Validation of Random Forest')
    plt.legend()
    plt.show()
```

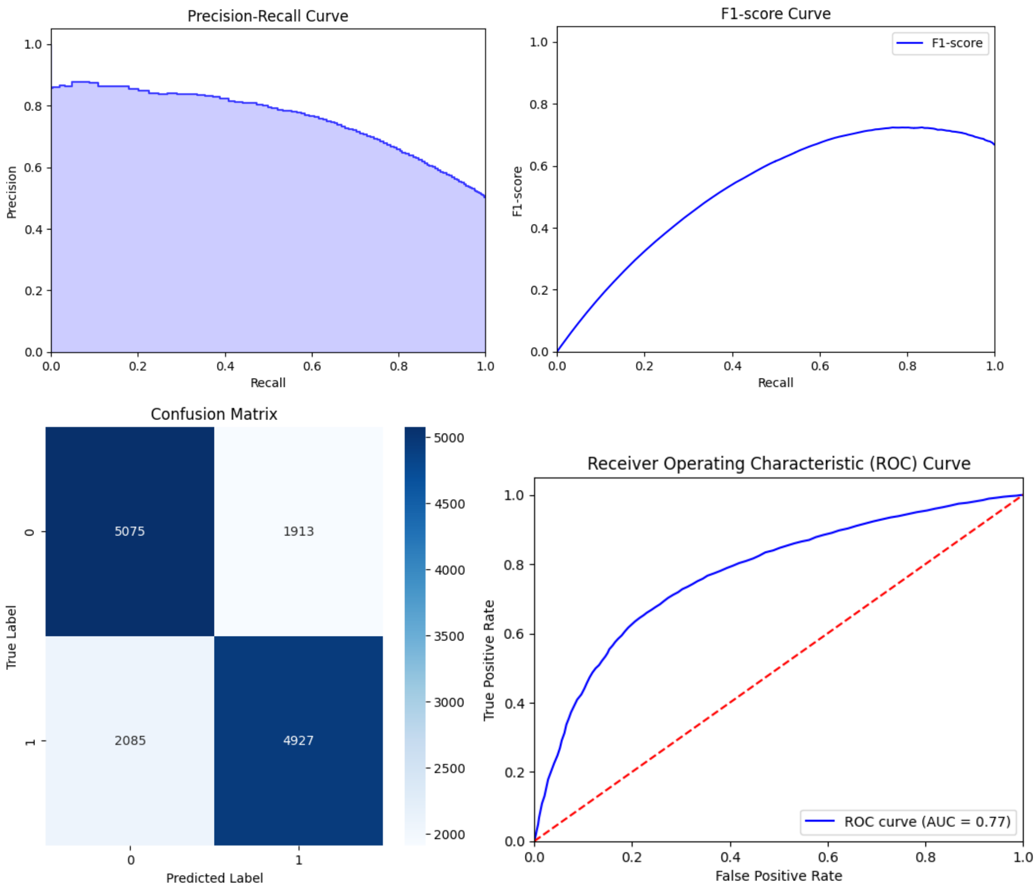



```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a RandomForestClassifier instance
rf_classifier = RandomForestClassifier()

# Fit the classifier to the training data
rf_classifier.fit(X_train, y_train)

# Model testing and evaluation
y_pred = rf_classifier.predict(X_test)

# Calculate accuracy, precision, recall, and F1-score
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

# Generate and print classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:")
print(class_report)

# Plot Precision-Recall curve
y_pred_probs = rf_classifier.predict_proba(X_test)[:, 1]
precision, recall, _ = precision_recall_curve(y_test, y_pred_probs)
```

```python
plt.step(recall, precision, color='b', alpha=0.7, where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.show()

# Plot F1-score curve
f1_values = 2 * (precision * recall) / (precision + recall)
plt.plot(recall, f1_values, color='b', label='F1-score')
plt.xlabel('Recall')
plt.ylabel('F1-score')
plt.title('F1-score Curve')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.legend()
plt.show()

# Plot Confusion matrix
confusion_mat = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Calculate ROC curve and AUC
y_pred_probs = rf_classifier.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs)
roc_auc = auc(fpr, tpr)
```

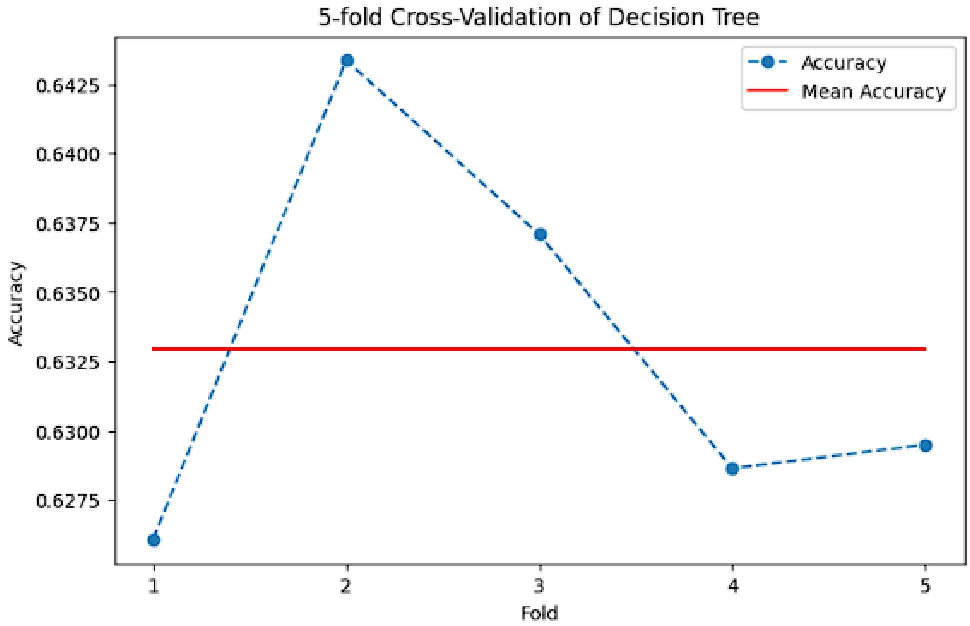
```python
# Plot ROC curve
plt.plot(fpr, tpr, color='b', label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='r', linestyle='--')
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
Accuracy: 0.7144285714285714
Precision: 0.7203216374269006
Recall: 0.7026525955504849
F1-score: 0.7113774184233325
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.73      0.72      6988
           1       0.72      0.70      0.71      7012

    accuracy                           0.71     14000
   macro avg       0.71      0.71      0.71     14000
weighted avg       0.71      0.71      0.71     14000
```

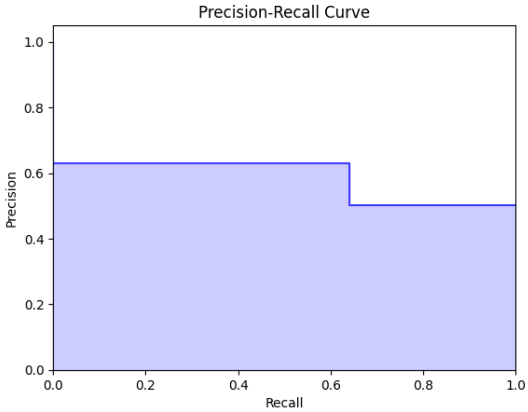


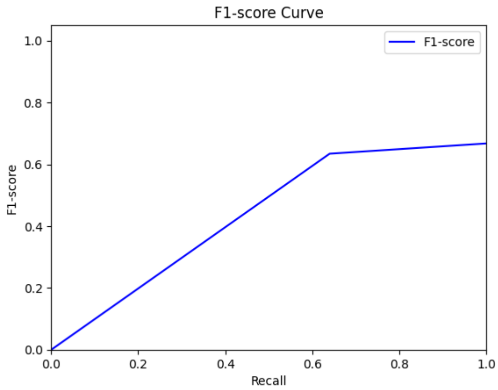
b. Decision Tree

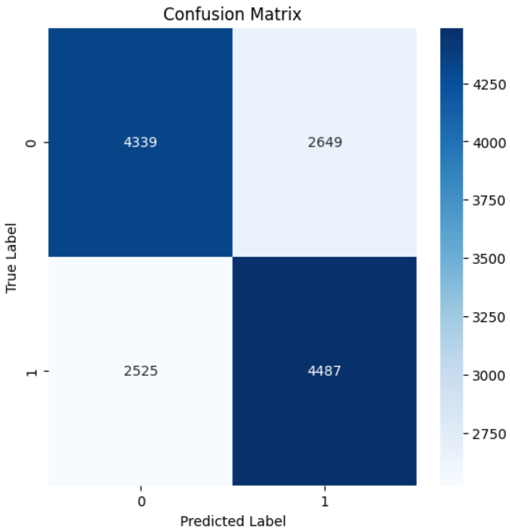
```
Decision Tree - Cross-Validation Scores: [0.62607143 0.64335714 0.63707143 0.62864286 0.6295    ]
Decision Tree - Mean Cross-Validation Score: 0.6329285714285715
```

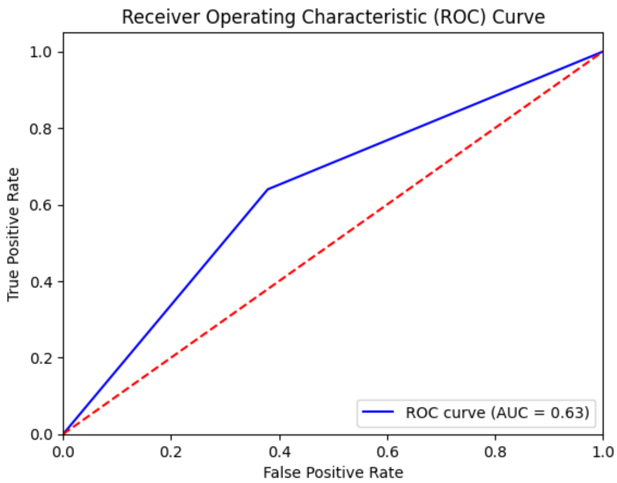


```
Accuracy: 0.6304285714285714
Precision: 0.6287836322869955
Recall: 0.6399030233884769
F1-score: 0.6342945999434549
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.62      0.63      6988
           1       0.63      0.64      0.63      7012

    accuracy                           0.63     14000
   macro avg       0.63      0.63      0.63     14000
weighted avg       0.63      0.63      0.63     14000
```



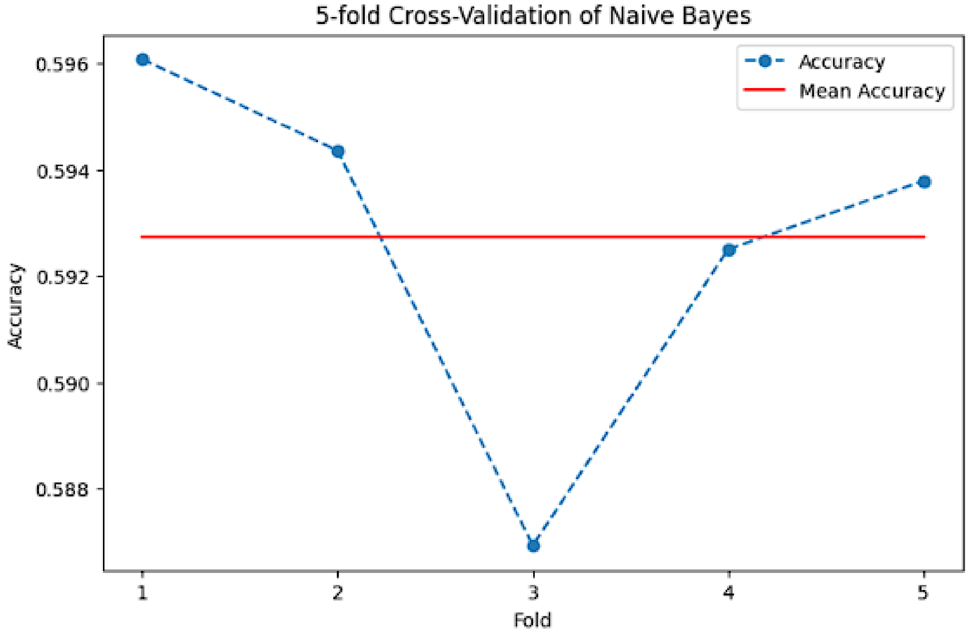




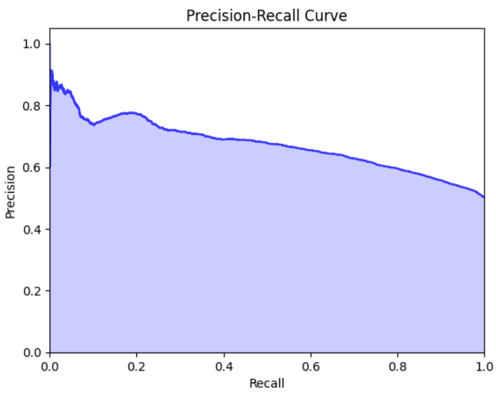


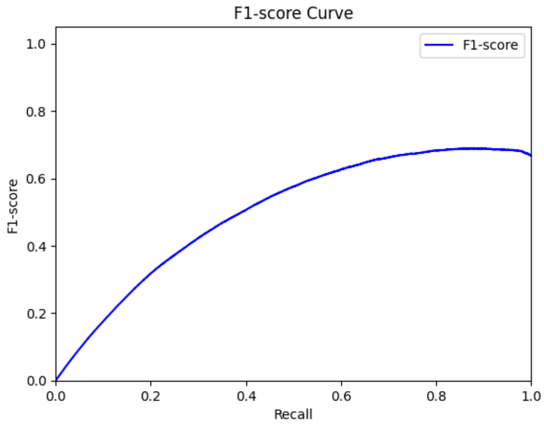
c. Naïve Bayes:

```
Naive Bayes - Cross-Validation Scores: [0.59607143 0.59435714 0.58692857 0.5925     0.59378571]
Naive Bayes - Mean Cross-Validation Score: 0.5927285714285714
```

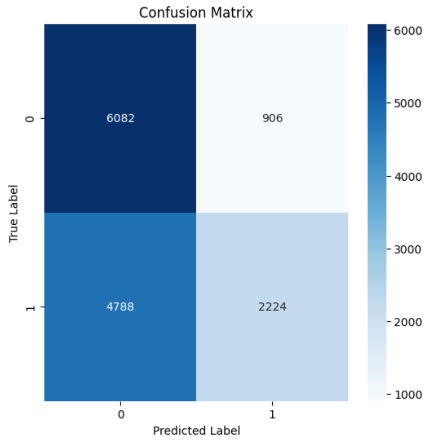


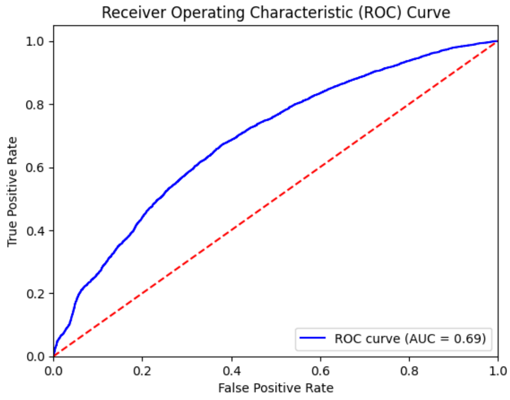

```
Accuracy: 0.5932857142857143
Precision: 0.7105431309904153
Recall: 0.31717056474614946
F1-score: 0.43857227371327157
Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.87      0.68      6988
           1       0.71      0.32      0.44      7012

    accuracy                           0.59     14000
   macro avg       0.64      0.59      0.56     14000
weighted avg       0.64      0.59      0.56     14000
```





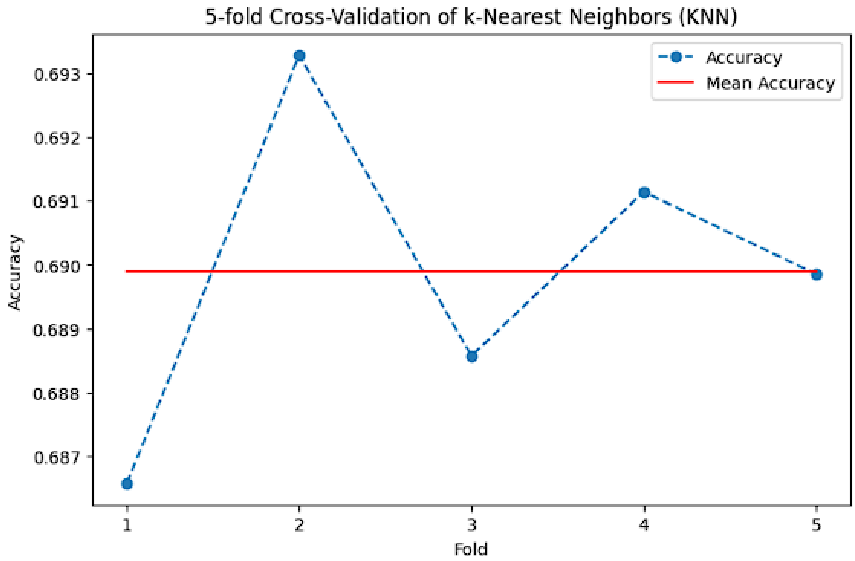
\



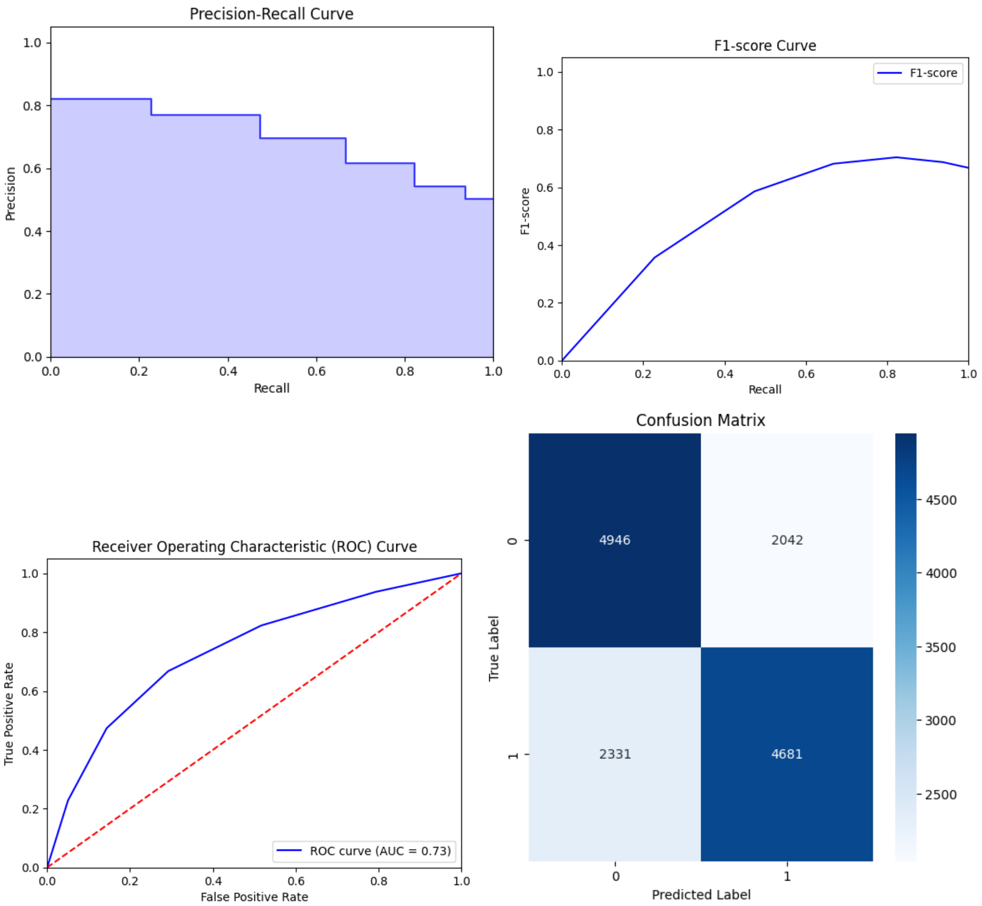


KNN:

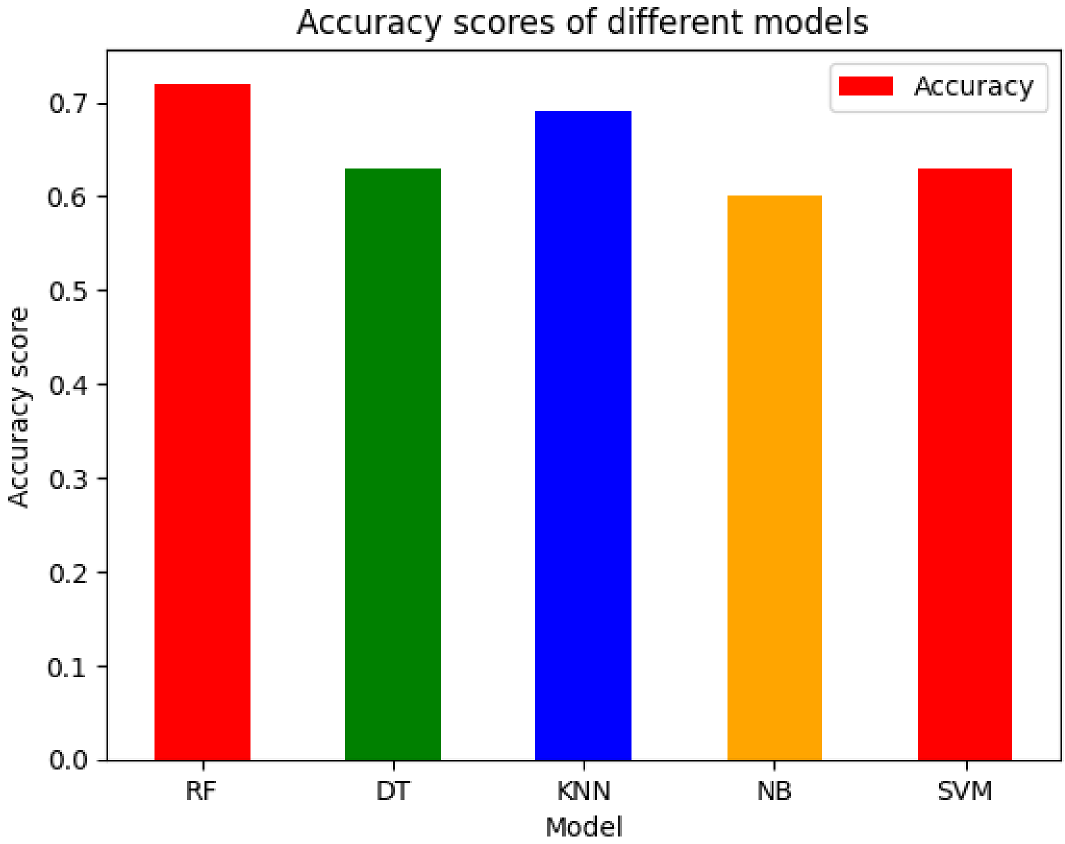
```
KNN - Cross-Validation Scores: [0.68657143 0.69328571 0.68857143 0.69114286 0.68985714]
KNN - Mean Cross-Validation Score: 0.6898857142857143
```




```
Accuracy: 0.6876428571428571
Precision: 0.6962665476721701
Recall: 0.6675698802053622
F1-score: 0.6816163087004004
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.71      0.69      6988
           1       0.70      0.67      0.68      7012

    accuracy                           0.69     14000
   macro avg       0.69      0.69      0.69     14000
weighted avg       0.69      0.69      0.69     14000
```


Precision-Recall Curve
Precision
Recall
F1-score Curve
F1-score
Recall
Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
ROC curve (AUC = 0.73)
Confusion Matrix
True Label
Predicted Label
4946
2042
2331
4681
4500
4000
3500
3000
2500

V. RESULTS



Accuracy scores of different models
Accuracy
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Accuracy score
RF
DT
KNN
NB
SVM
Model