SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

B22CI0501 – MACHINE LEARNING

PREDICTION OF DIABETES PAITENTS USING DIABETES DATASET

SUBMITTED BY

NAME: TARUN KUMAR S

SRN: R22EA063

SEMESTER & BRANCH: 5th SEM AIML B

DATE OF SUBMISSION:

SIGNATURE:

VERIFIED BY:

MARKS:

SIGNATURE OF FACULTY:

QUESTIONS TO BE ANSWERED:

1. What is the nature of the dataset (structured, unstructured, or semi-structured)?

- The dataset is **structured**, with clear columns representing variables (features) and the "Survived" as the target label. It is a **labelled** dataset.

2. What is the source of the dataset (public, proprietary, or in-house)?

- The Titanic dataset primarily comes from passenger records of the RMS Titanic, a British passenger liner that sank on its maiden voyage in 1912.

3. What is the size of the dataset in terms of samples or instances before applying PCA?

- The Titanic dataset from Kaggle typically has:
- 891 samples (instances) in the training set.
- 418 samples in the test set.

4. Are the labels accurate and meaningful for the intended task?

- The labels in the "Survived" column are binary (0 for Not Survived, 1 for Survived). They appear meaningful for the task of predicting.

5. How consistent is the labeling across the dataset?

- Since the dataset is structured and fully populated, the labeling seems consistent across all instances.

6. Are there missing values or corrupted data points in the dataset?

- No missing values are detected in this dataset. However, some values (like zero values for Age and Cabin) may need to be addressed as potential data issues or outliers.

7. Are there any legal or ethical issues related to using this dataset?

- No, there is no legal or ethical issues related to using this dataset.

8. What is the class distribution, and is it balanced or imbalanced?

- The target class distribution (Survived) will need to be checked to see if it's balanced or imbalanced.
- Based on analysis, The dataset is slightly imbalanced:

  Class 0 (Not Survived): 60%

  Class 1 (Survived): 40%

CODE OF THE ML MODEL:

```
#Step 1: Data Pre-processing step

#importing 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

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay, classification_report

from sklearn.preprocessing import StandardScaler


#importing dataset

#Load the data

train_data = pd.read_csv('../dataset/raw/train.csv')

test_data = pd.read_csv('../dataset/raw/test.csv')

test_data_survived = pd.read_csv('../dataset/raw/gender_submission.csv')
```

```
# Handle missing values
train_data.fillna(method='ffill', inplace=True)
test_data.fillna(method='ffill', inplace=True)

# Convert categorical variables to numerical
train_data = pd.get_dummies(train_data, columns=['Sex', 'Embarked'], drop_first=True)
test_data = pd.get_dummies(test_data, columns=['Sex', 'Embarked'], drop_first=True)

#Check the data
print("Tarin Data:")
print(train_data.head())
print("\nTest Data:")
print(test_data.head())
print(test_data_survived["Survived"].head())
```

```
Tarin Data:
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name   Age  SibSp  Parch  \
0                            Braund, Mr. Owen Harris  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  38.0      1      0
2                             Heikkinen, Miss. Laina  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  35.0      1      0
4                           Allen, Mr. William Henry  35.0      0      0

             Ticket     Fare Cabin  Sex_male  Embarked_Q  Embarked_S
0         A/5 21171   7.2500   NaN      True       False        True
1          PC 17599  71.2833   C85     False       False       False
2  STON/O2. 3101282   7.9250   C85     False       False        True
3            113803  53.1000  C123     False       False        True
4            373450   8.0500  C123      True       False        True

Test Data:
   PassengerId  Pclass                                          Name   Age  \
0          892       3                              Kelly, Mr. James  34.5
1          893       3              Wilkes, Mrs. James (Ellen Needs)  47.0
2          894       2                     Myles, Mr. Thomas Francis  62.0
3          895       3                              Wirz, Mr. Albert  27.0
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  22.0

   SibSp  Parch   Ticket     Fare Cabin  Sex_male  Embarked_Q  Embarked_S
0      0      0   330911   7.8292   NaN      True        True       False
1      1      0   363272   7.0000   NaN     False       False        True
2      0      0   240276   9.6875   NaN      True        True       False
3      0      0   315154   8.6625   NaN      True       False        True
4      1      1  3101298  12.2875   NaN     False       False        True
0    0
1    1
2    0
3    0
4    1
```

```
# Select features and target
X_train = train_data.drop(['Survived', 'Name', 'Ticket', 'Cabin'], axis=1)
y_train = train_data['Survived']
X_test = test_data.drop(['Name', 'Ticket', 'Cabin'], axis=1)


#Fit the train and test data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test).


#Displaying the data
print("Training set of Independent after features scaling:")
print(X_train[:5])
print("\nTraining set of Dependent:")
print(y_train[:5])
print("\nTesting set of Independent after features scaling:")
print(X_test[:5])
print("\nesting set of Dependent:")
print(y_test[:5])
```

```
Training set of Independent variables after features scaling:
[[-1.73010796  0.82737724 -0.52119766  0.43279337 -0.47367361 -0.50244517
   0.73769513 -0.30974338  0.61930636]
 [-1.72622007 -1.56610693  0.57872934  0.43279337 -0.47367361  0.78684529
  -1.35557354 -0.30974338 -1.61470971]
 [-1.72233219  0.82737724 -0.24621591 -0.4745452  -0.47367361 -0.48885426
  -1.35557354 -0.30974338  0.61930636]
 [-1.71844431 -1.56610693  0.37249302  0.43279337 -0.47367361  0.42073024
  -1.35557354 -0.30974338  0.61930636]
 [-1.71455642  0.82737724  0.37249302 -0.4745452  -0.47367361 -0.48633742
   0.73769513 -0.30974338  0.61930636]]

Training set of Dependent variables:
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64

Testing set of Independent variables after features scaling:
[[ 1.73399584  0.82737724  0.33812031 -0.4745452  -0.47367361 -0.49078316
   0.73769513  3.22847904 -1.61470971]
 [ 1.73788372  0.82737724  1.19743827  0.43279337 -0.47367361 -0.50747884
  -1.35557354 -0.30974338  0.61930636]
 [ 1.74177161 -0.36936484  2.22861983 -0.4745452  -0.47367361 -0.45336687
   0.73769513  3.22847904 -1.61470971]
 [ 1.74565949  0.82737724 -0.17747047 -0.4745452  -0.47367361 -0.47400493
   0.73769513 -0.30974338  0.61930636]
 [ 1.74954737  0.82737724 -0.52119766  0.43279337  0.76762988 -0.40101668
  -1.35557354 -0.30974338  0.61930636]]

Testing set of Dependent variables:
0    0
1    1
2    0
3    0
4    1
```

#Step 2: Training the model

#Fitting Logistic Regression to the training set

model = LogisticRegression(max_iter=10, random_state=7)

model.fit(X_train, y_train)

```
LogisticRegression
LogisticRegression(max_iter=10, random_state=7)
```

#Step 3: Predicting the test set result

Predict on the test data

test_predictions = model.predict(X_test)

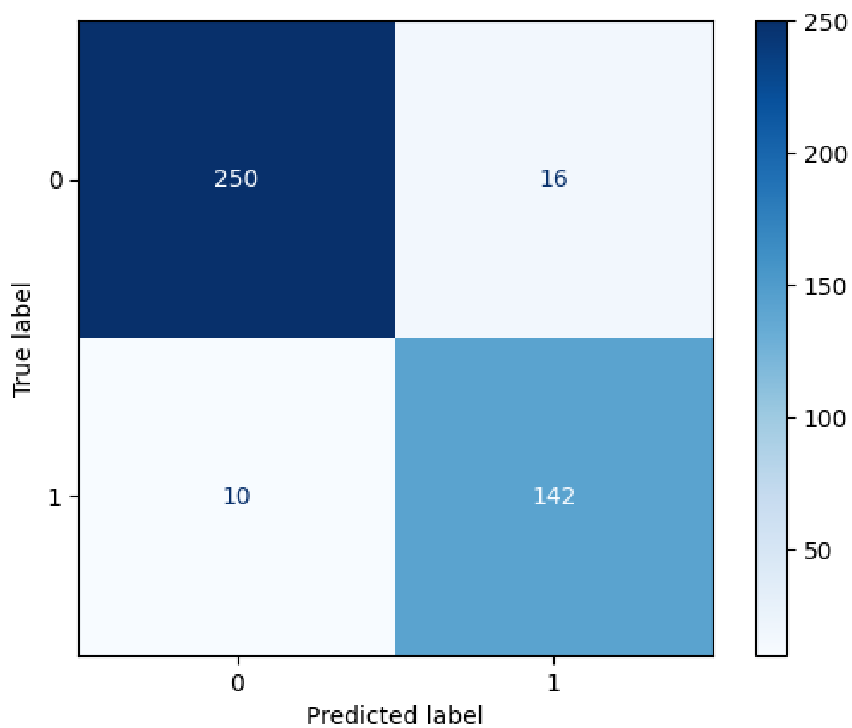
print(test_predictions)

```
[0 0 0 0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 0 0 0 0 0 0 0 1 0 1
 1 0 0 0 0 0 1 1 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 1 1 1 0 1 1
 1 1 0 1 0 1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0
 1 1 1 1 0 0 1 1 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0
 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 1 1 0 1 1 0 0 1 0 1
 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 1 0 1 0 1 0
 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 1 1 0 1 0 0 0 0 1
 0 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0
 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0
 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 1 0
 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1 0 1 1 0 0 0
 0 1 1 1 1 1 0 1 0 0 0]
Test Accuracy: 0.937799043062201
```

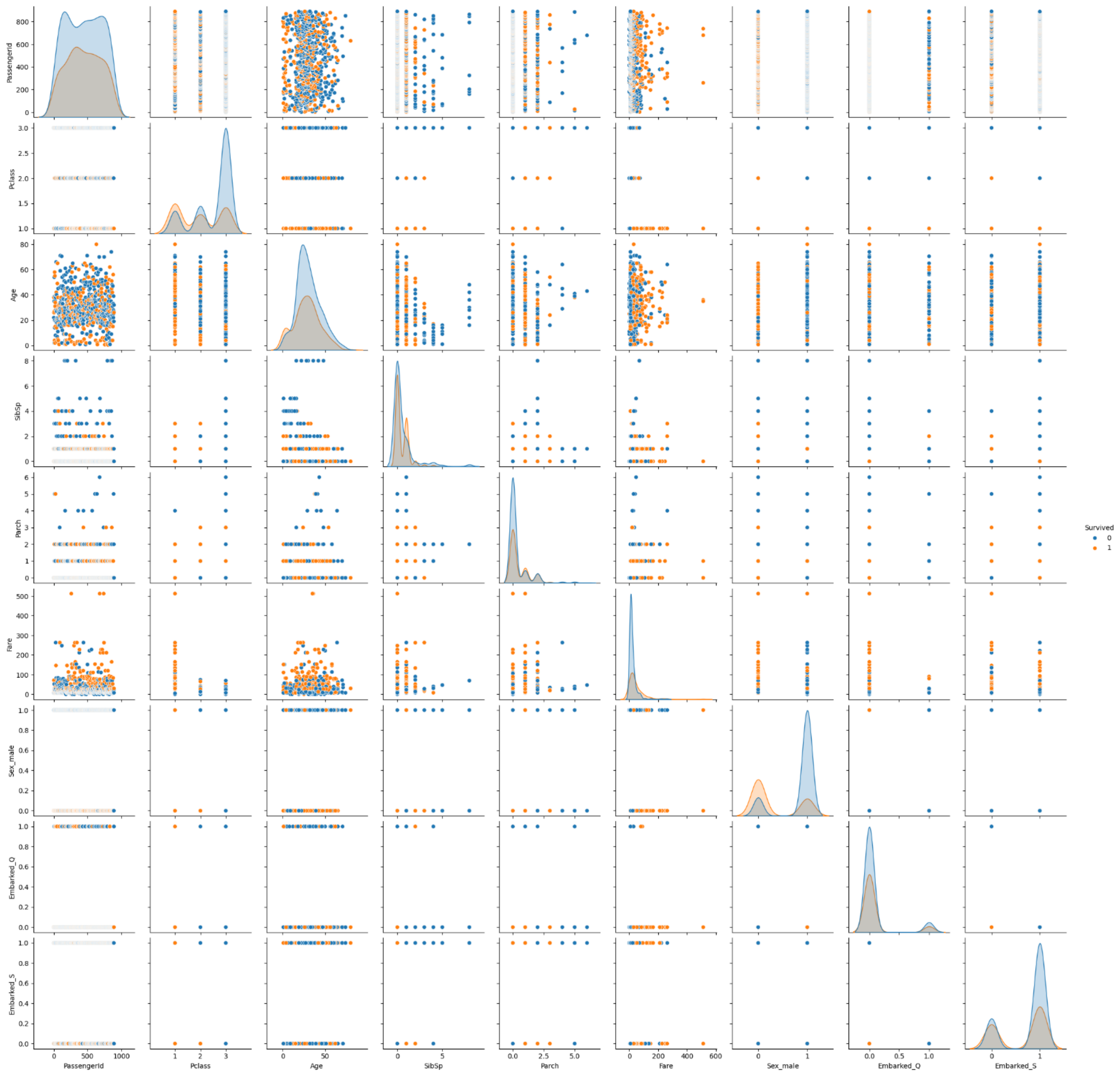
```
#Step 4: Creating the confusion matrix and checking other performance metrics

# Calculate the accuracy

accuracy = accuracy_score(y_test, test_predictions)

print(f'Test Accuracy: {accuracy}')


# Calculate the confusion matrix

conf_matrix = confusion_matrix(y_test, test_predictions)

print("\nConfusion Matrix:")

print(conf_matrix)


# Display the classification report

class_report = classification_report(y_test, test_predictions, target_names=['Not
Survived', 'Survived'], output_dict=True)

class_report_df = pd.DataFrame(class_report).transpose()


class_report_df = class_report_df.round(2)

print("\nFormatted Classification Report:")

print(class_report_df)
```

```
# Display the confusion matrix using matplotlib

disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix)

disp.plot(cmap=plt.cm.Blues)

plt.show()
```

```
Test Accuracy: 0.937799043062201

Confusion Matrix:
[[250  16]
 [ 10 142]]

Formatted Classification Report:
             precision  recall  f1-score  support
Not Survived      0.96    0.94      0.95   266.00
Survived          0.90    0.93      0.92   152.00
accuracy          0.94    0.94      0.94     0.94
macro avg         0.93    0.94      0.93   418.00
weighted avg      0.94    0.94      0.94   418.00
```



```
# Pair plot to visualize pairwise relationships between features

sns.pairplot(train_data, hue='Survived', diag_kind='kde')

plt.show()
```



Predicting the output based on user input

```
#Predicting the output based on user input
import joblib
import numpy as np
import os
from sklearn.preprocessing import StandardScaler

# Function to load all models from a specified folder
def load_models_from_folder(folder_path):
    models = {}
    for filename in os.listdir(folder_path):
        if filename.endswith('.pkl'):
            model_name = filename.split('.')[0]
            model_path = os.path.join(folder_path, filename)
            models[model_name] = joblib.load(model_path)
    return models

# Function to take input for the Titanic dataset
def get_titanic_input():
    Pclass = int(input("Enter Pclass (1, 2, or 3): "))
    Sex = input("Enter Sex (male or female): ")
    Age = float(input("Enter Age: "))
    SibSp = int(input("Enter number of siblings/spouses aboard: "))
    Parch = int(input("Enter number of parents/children aboard: "))
    Fare = float(input("Enter Fare: "))
    Embarked = input("Enter Embarked (C, Q, or S): ")
```

```
    # Convert categorical variables to numerical
    Sex = 1 if Sex == 'male' else 0
    Embarked_C = 1 if Embarked == 'C' else 0
    Embarked_Q = 1 if Embarked == 'Q' else 0
    Embarked_S = 1 if Embarked == 'S' else 0

    # Create input array
    input_data = np.array([[Pclass, Sex, Age, SibSp, Parch, Fare, Embarked_C,
Embarked_Q, Embarked_S]])
    print("Pclass:", Pclass)
    print("Sex:", Sex)
    print("Age:", Age)
    print("Siblings/Spouses:", SibSp)
    print("Parents/Children:", Parch)
    print("Fare:", Fare)
    print("Embarked_C:", Embarked_C)
    print("Embarked_Q:", Embarked_Q)
    print("Embarked_S:", Embarked_S)
    print("\n")
    return input_data

# Load models from the specified folder
folder_path = '../models'  # Replace with the path to your models folder
models = load_models_from_folder(folder_path)

# Get input data
input_data = get_titanic_input()
```

```
survive_dict = {
    0: "Not Survived",
    1: "Survived"
}

# Predict using each model and print the outputs
for model_name, model in models.items():
    output = model.predict(input_data)
    print(f'Output from {model_name}: {survive_dict[output[0]]}')
```

```
Pclass: 3
Sex: 1
Age: 35.0
Siblings/Spouses: 0
Parents/Children: 0
Fare: 8.05
Embarked_C: 0
Embarked_Q: 0
Embarked_S: 1


Output from decision_tree: Not Survived
Output from gradient_boosting: Survived
Output from k-nearest_neighbors: Not Survived
Output from logistic_regression: Not Survived
Output from logistic_regression_model_main: Not Survived
Output from naive_bayes: Survived
Output from neural_network_(mlp): Not Survived
Output from random_forest: Not Survived
Output from support_vector_machine: Not Survived
```

```
#Using same dataset on different ML Algorithms
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

#Defining different models to a Dictionary
models = {
    "Support Vector Machine": SVC(kernel='linear', random_state=0),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(random_state=0),
    "Logistic Regression": LogisticRegression(random_state=0),
    "Random Forest": RandomForestClassifier(random_state=0),
    "Gradient Boosting": GradientBoostingClassifier(random_state=0),
    "Neural Network (MLP)": MLPClassifier(random_state=0),
    "Naive Bayes": GaussianNB(),
}
```

```
#Storing the accuracies of every model
results = []
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{model_name} Accuracy: {accuracy * 100:.2f}%")
    results.append([model_name, accuracy])
    file_name = model_name.lower().replace(" ", "_") + ".pkl"
    joblib.dump(model, f"../models/{file_name}")

# Create a DataFrame for the results
results_df = pd.DataFrame(results, columns=["Model", "Accuracy"])

#Ploting different models accuracy
plt.figure(figsize=(12, 6))  # Adjust figure size as needed
ax = sns.barplot(x="Model", y="Accuracy", data=results_df)
plt.title("Comparison of Model Accuracies")
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.ylim(0, 1)  # Set y-axis limits to 0-1 for accuracy
plt.xticks(rotation=45, ha="right")  # Rotate x-axis labels for better readability

# Add text labels on the bars
for p in ax.patches:
    ax.annotate(f'{p.get_height():.2f}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
```

```
                ha='center', va='center',
                xytext=(0, 9),
                textcoords='offset points')
plt.tight_layout()
plt.show()
```

```
Support Vector Machine Accuracy: 100.00%
K-Nearest Neighbors Accuracy: 81.34%
Decision Tree Accuracy: 77.03%
Logistic Regression Accuracy: 93.78%
Random Forest Accuracy: 80.38%
Gradient Boosting Accuracy: 82.78%
Neural Network (MLP) Accuracy: 76.32%
Naive Bayes Accuracy: 91.39%
```