Personalized Healthcare Recommendations Unified Mentor(1)

October 24, 2024

[10]:
```
# Import necessary libraries
import pandas as pd
```

[11]:
```
# Load the dataset
data = pd.read_csv('blood.csv')
```

[12]:
```
# Display the first few rows of the dataset
print(data.head())
```

```
   Recency  Frequency  Monetary  Time  Class
0        2         50     12500    99      1
1        0         13      3250    28      1
2        1         17      4000    36      1
3        2         20      5000    45      1
4        1         24      6000    77      0
```

[13]:
```
# Check the dataset structure and summary
print(data.info())
print(data.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 748 entries, 0 to 747
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Recency    748 non-null    int64
 1   Frequency  748 non-null    int64
 2   Monetary   748 non-null    int64
 3   Time       748 non-null    int64
 4   Class      748 non-null    int64
dtypes: int64(5)
memory usage: 29.3 KB
None
          Recency   Frequency      Monetary        Time       Class
count  748.000000  748.000000    748.000000  748.000000  748.000000
mean     9.506684    5.516043   1378.676471   34.284759    0.237968
std      8.095396    5.841825   1459.826781   24.380307    0.426124
min      0.000000    1.000000    250.000000    2.000000    0.000000
```

```
25%       2.750000     2.000000     500.000000    16.000000     0.000000
50%       7.000000     4.000000    1000.000000    28.000000     0.000000
75%      14.000000     7.000000    1750.000000    50.000000     0.000000
max      74.000000    50.000000   12500.000000    99.000000     1.000000
```

[17]:
```
# Fill or drop missing values (choose appropriate strategy)
data.fillna(data.median(), inplace=True)
```

[20]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Step 1: Separate features (X) and labels (y)
X = data.drop('Class', axis=1)
y = data['Class']
```

[21]:
```
# Step 2: Scale the numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[22]:
```
# Step 3: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42)
```

[23]:
```
from sklearn.ensemble import RandomForestClassifier

# Step 1: Initialize the model
rf_model = RandomForestClassifier(random_state=42)
```

[24]:
```
# Step 2: Train the model
rf_model.fit(X_train, y_train)
```

[24]:
```
RandomForestClassifier(random_state=42)
```

[25]:
```
from sklearn.metrics import classification_report, confusion_matrix

# Step 1: Make predictions on the test set
y_pred = rf_model.predict(X_test)
```

[26]:
```
# Step 2: Evaluate the model
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[99 14]
 [28  9]]
              precision    recall  f1-score   support

           0       0.78      0.88      0.82       113
           1       0.39      0.24      0.30        37
```

```
    accuracy                           0.72       150
   macro avg       0.59      0.56      0.56       150
weighted avg       0.68      0.72      0.70       150
```

[27]:
```
# Function to generate recommendations
def generate_recommendations(patient_data):
    # Scale the input data
    patient_data_scaled = scaler.transform(patient_data)

    # Make a prediction
    prediction = rf_model.predict(patient_data_scaled)

    # Map predictions to recommendations
    recommendation_mapping = {0: 'No blood donation needed', 1: 'Consider␣
 ↪donating blood'}

    return recommendation_mapping[prediction[0]]
```

[28]:
```
# Example of patient data
example_patient_data = pd.DataFrame({
    'Recency': [5],
    'Frequency': [10],
    'Monetary': [2500],
    'Time': [25]
})

# Generate a recommendation
print(generate_recommendations(example_patient_data))
```

```
Consider donating blood
```

[ ]:
```

```