Liver Disease Prediction using a Liver Patient Dataset

Rajesh Devaguptapu- 101178054

In [97]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
data=pd.read_csv('Indian Liver Patient Dataset (ILPD).csv')
data
```

Out[2]:
```
     65  Female   0.7  0.1  187  16   18  6.8  3.3   0.9  1
  0  62    Male  10.9  5.5  699  64  100  7.5  3.2  0.74  1
  1  62    Male   7.3  4.1  490  60   68  7.0  3.3  0.89  1
  2  58    Male   1.0  0.4  182  14   20  6.8  3.4  1.00  1
  3  72    Male   3.9  2.0  195  27   59  7.3  2.4  0.40  1
  4  46    Male   1.8  0.7  208  19   14  7.6  4.4  1.30  1
...  ...    ...   ...  ...  ...  ...  ...  ...  ...   ... ...
577  60    Male   0.5  0.1  500  20   34  5.9  1.6  0.37  2
578  40    Male   0.6  0.1   98  35   31  6.0  3.2  1.10  1
579  52    Male   0.8  0.2  245  48   49  6.4  3.2  1.00  1
580  31    Male   1.3  0.5  184  29   32  6.8  3.4  1.00  1
581  38    Male   1.0  0.3  216  21   24  7.3  4.4  1.50  2

582 rows × 11 columns
```

In [3]:
```
data.head()
```

Out[3]:
```
   65  Female   0.7  0.1  187  16   18  6.8  3.3   0.9  1
0  62    Male  10.9  5.5  699  64  100  7.5  3.2  0.74  1
1  62    Male   7.3  4.1  490  60   68  7.0  3.3  0.89  1
2  58    Male   1.0  0.4  182  14   20  6.8  3.4  1.00  1
3  72    Male   3.9  2.0  195  27   59  7.3  2.4  0.40  1
4  46    Male   1.8  0.7  208  19   14  7.6  4.4  1.30  1
```

In [4]:
```
data.tail()
```

Out[4]:
```
     65  Female  0.7  0.1  187  16  18  6.8  3.3   0.9  1
577  60    Male  0.5  0.1  500  20  34  5.9  1.6  0.37  2
578  40    Male  0.6  0.1   98  35  31  6.0  3.2  1.10  1
579  52    Male  0.8  0.2  245  48  49  6.4  3.2  1.00  1
580  31    Male  1.3  0.5  184  29  32  6.8  3.4  1.00  1
581  38    Male  1.0  0.3  216  21  24  7.3  4.4  1.50  2
```

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 582 entries, 0 to 581
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   65      582 non-null    int64  
 1   Female  582 non-null    object 
 2   0.7     582 non-null    float64
 3   0.1     582 non-null    float64
 4   187     582 non-null    int64  
 5   16      582 non-null    int64  
 6   18      582 non-null    int64  
 7   6.8     582 non-null    float64
 8   3.3     582 non-null    float64
 9   0.9     578 non-null    float64
 10  1       582 non-null    int64  
dtypes: float64(5), int64(5), object(1)
memory usage: 50.1+ KB
```

In [6]:
```
data.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
65	582.0	44.711340	16.181921	4.0	33.00	45.00	57.75	90.0
0.7	582.0	3.303265	6.213926	0.4	0.80	1.00	2.60	75.0
0.1	582.0	1.488488	2.810324	0.1	0.20	0.30	1.30	19.7
187	582.0	290.754296	243.108929	63.0	175.25	208.00	298.00	2110.0
16	582.0	80.824742	182.757696	10.0	23.00	35.00	60.75	2000.0
18	582.0	110.068729	289.141876	10.0	25.00	42.00	87.00	4929.0
6.8	582.0	6.482646	1.086306	2.7	5.80	6.60	7.20	9.6
3.3	582.0	3.141581	0.796176	0.9	2.60	3.10	3.80	5.5
0.9	578.0	0.947145	0.319863	0.3	0.70	0.94	1.10	2.8
1	582.0	1.286942	0.452723	1.0	1.00	1.00	2.00	2.0

In [7]:
```
data1=data.copy()
```

In [8]:
```
data1.duplicated().sum()
```

Out[8]:
```
13
```

In [9]:
```
data1.head()
```

Out[9]:

	65	Female	0.7	0.1	187	16	18	6.8	3.3	0.9	1
0	62	Male	10.9	5.5	699	64	100	7.5	3.2	0.74	1
1	62	Male	7.3	4.1	490	60	68	7.0	3.3	0.89	1
2	58	Male	1.0	0.4	182	14	20	6.8	3.4	1.00	1
3	72	Male	3.9	2.0	195	27	59	7.3	2.4	0.40	1
4	46	Male	1.8	0.7	208	19	14	7.6	4.4	1.30	1

In [10]:
```
data1.rename(columns={'65': 'Age of the patient', 'Female': 'Gender of the patient','0.7':'Total Bilir
                      'Direct Bilirubin','187':'Alkaline Phosphatase','16':'Alanine Aminotransferase',
                      'Aspartate Aminotransferase','6.8':'Total Proteins','3.3':'Albumin','0.9':'Album
                      '1':'Target'}, inplace=True)
```

In [11]:
```
data1.head()
```

Out[11]:

	Age of the patient	Gender of the patient	Total Bilirubin	Direct Bilirubin	Alkaline Phosphatase	Alanine Aminotransferase	Aspartate Aminotransferase	Total Proteins	Albumin	Albumin and Globulin Ratio	Target
0	62	Male	10.9	5.5	699	64	100	7.5	3.2	0.74	1
1	62	Male	7.3	4.1	490	60	68	7.0	3.3	0.89	1
2	58	Male	1.0	0.4	182	14	20	6.8	3.4	1.00	1
3	72	Male	3.9	2.0	195	27	59	7.3	2.4	0.40	1
4	46	Male	1.8	0.7	208	19	14	7.6	4.4	1.30	1

In [12]:
```
data1.describe()
```

Out[12]:

	Age of the patient	Total Bilirubin	Direct Bilirubin	Alkaline Phosphatase	Alanine Aminotransferase	Aspartate Aminotransferase	Total Proteins	Albumin	Albu a Globu Ra
count	582.000000	582.000000	582.000000	582.000000	582.000000	582.000000	582.000000	582.000000	578.000
mean	44.711340	3.303265	1.488488	290.754296	80.824742	110.068729	6.482646	3.141581	0.947
std	16.181921	6.213926	2.810324	243.108929	182.757696	289.141876	1.086306	0.796176	0.319
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000	2.700000	0.900000	0.300
25%	33.000000	0.800000	0.200000	175.250000	23.000000	25.000000	5.800000	2.600000	0.700
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000	6.600000	3.100000	0.940
75%	57.750000	2.600000	1.300000	298.000000	60.750000	87.000000	7.200000	3.800000	1.100
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	9.600000	5.500000	2.800

In [13]:
```
data1.isnull().sum()
```

Out[13]:
```
Age of the patient            0
Gender of the patient         0
Total Bilirubin               0
Direct Bilirubin              0
Alkaline Phosphatase          0
Alanine Aminotransferase      0
Aspartate Aminotransferase    0
Total Proteins                0
Albumin                       0
Albumin and Globulin Ratio    4
Target                        0
dtype: int64
```

In [14]:
```
data1=data1.fillna(data1.mean())
```

In [15]:
```
data1.isnull().sum()
```

Out[15]:
```
Age of the patient            0
Gender of the patient         0
Total Bilirubin               0
Direct Bilirubin              0
Alkaline Phosphatase          0
Alanine Aminotransferase      0
Aspartate Aminotransferase    0
Total Proteins                0
Albumin                       0
Albumin and Globulin Ratio    0
Target                        0
dtype: int64
```

In [16]:
```
data1.describe(include="O").T
```

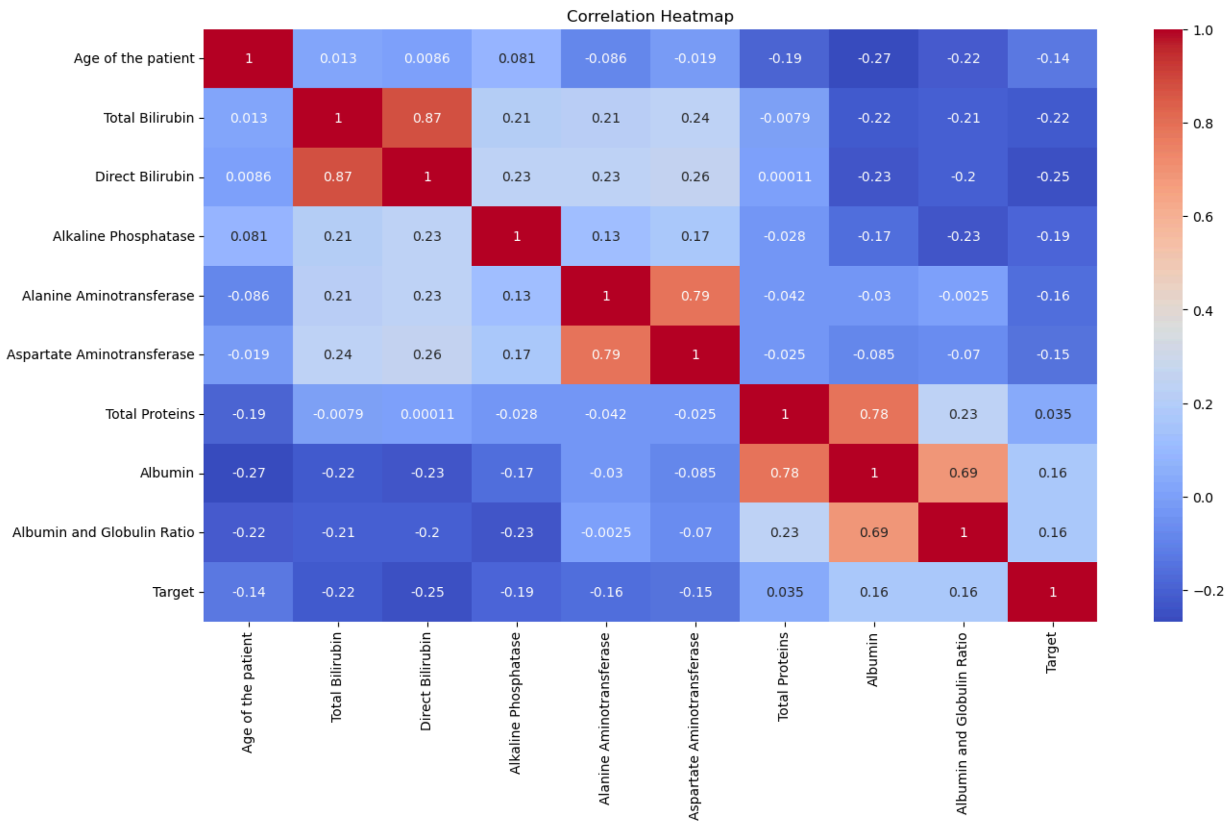
Out[16]:

	count	unique	top	freq
Gender of the patient	582	2	Male	441

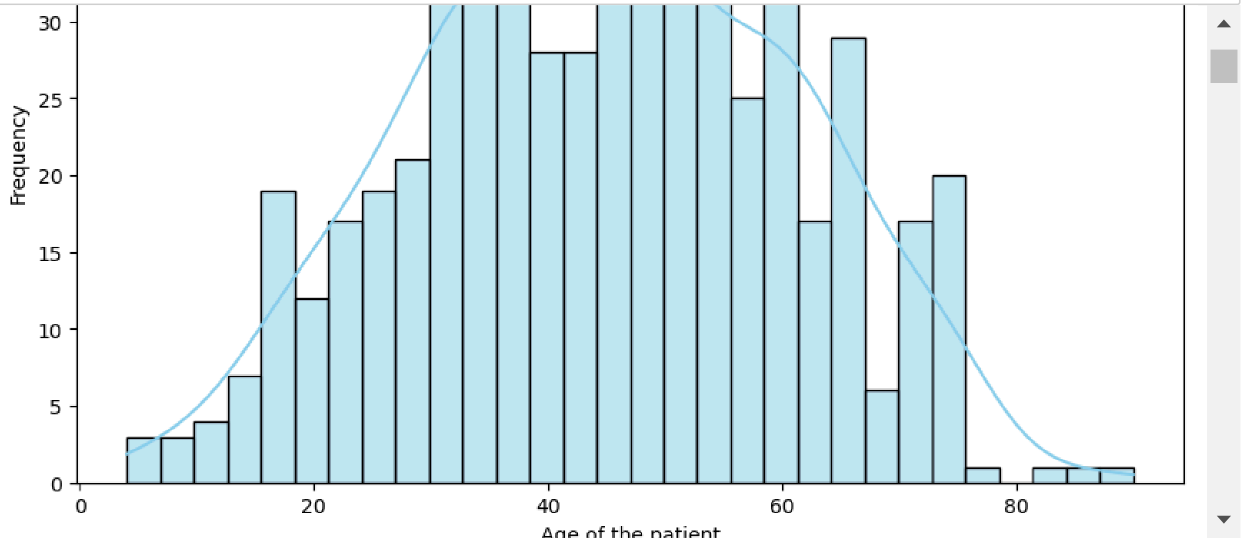
In [17]:
```
data1.duplicated().sum()
```

Out[17]:
```
13
```

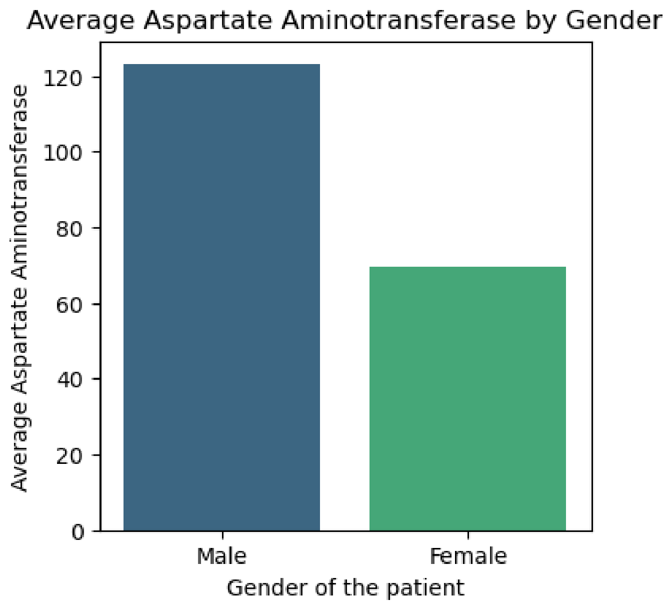
In [18]:
```

# Explore relationships between features and the target variable
plt.figure(figsize=(15, 8))
sns.heatmap(data1.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



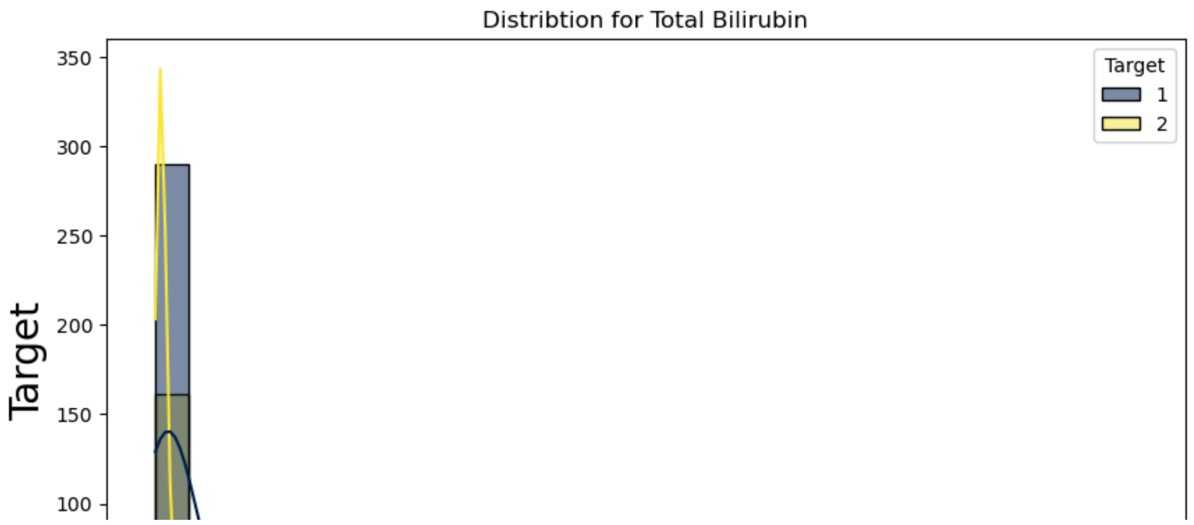
In [19]:
```
for column in data1.columns:
    if data1[column].dtype == 'O':
        # For categorical features, use bar plots
        plt.figure(figsize=(10, 6))
        sns.countplot(x=column, data=data1, palette='viridis')
        plt.title(f'Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Count')
        plt.xticks(rotation=45)
        plt.show()
    else:
        # For numerical features, use histograms
        plt.figure(figsize=(10, 6))
        sns.histplot(data1[column], bins=30, kde=True, color='skyblue')
        plt.title(f'Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.show()
```



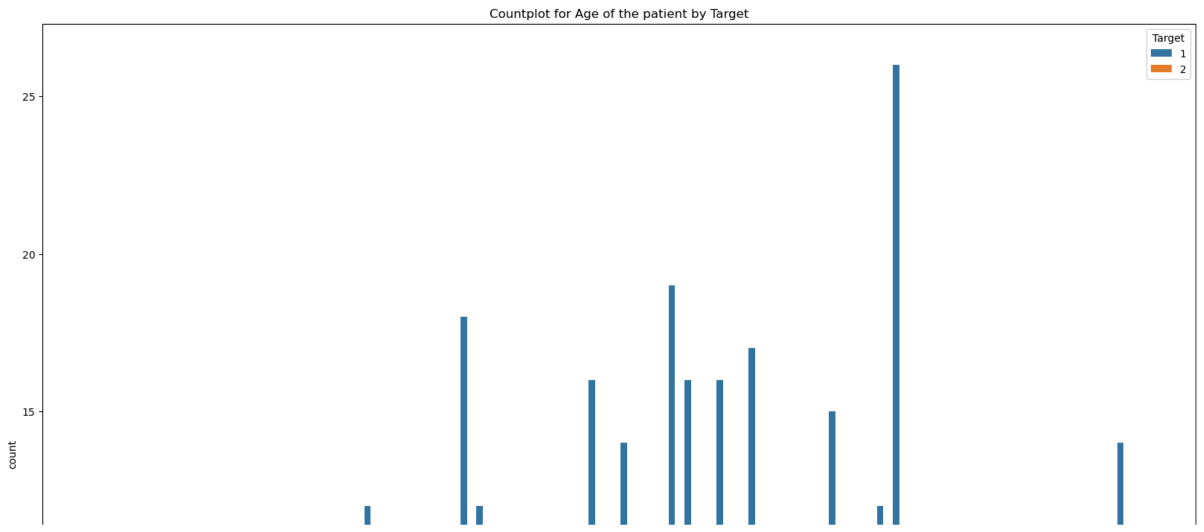
In [20]:
```
plt.figure(figsize=(4, 4))
sns.barplot(x='Gender of the patient', y='Aspartate Aminotransferase', data=data1, ci=None, palette='v
plt.title('Average Aspartate Aminotransferase by Gender')
plt.xlabel('Gender of the patient')
plt.ylabel('Average Aspartate Aminotransferase')
plt.show()
```



In [21]:
```python
#Histogram

columns=data1[['Total Bilirubin','Direct Bilirubin','Alkaline Phosphatase','Alanine Aminotransferase',
               'Albumin and Globulin Ratio']]

for column in columns:
        plt.figure(figsize=(10, 6))
        sns.histplot(data=data1, x=data1[column], hue='Target', palette='cividis', bins=30, kde=True)
        plt.title(f'Distribtion for {column}')
        plt.xlabel(column,fontsize=20)
        plt.ylabel('Target',fontsize=20)
        plt.xticks(rotation=90)
        plt.show()
```



In [22]:
```python
#Barplot
plt.figure(figsize=(20, 15))
sns.countplot(x='Age of the patient', data=data1, hue='Target')
plt.title('Countplot for Age of the patient by Target')
plt.show()

plt.figure(figsize=(20, 15))
sns.countplot(x='Gender of the patient', data=data1, hue='Target')
plt.title('Countplot for Gender by Target')
plt.show()

plt.figure(figsize=(20, 15))
sns.countplot(x='Total Proteins', data=data1, hue='Target')
plt.title('Countplot for Total Proteins by Target')
plt.show()

plt.figure(figsize=(20, 15))
sns.countplot(x='Albumin', data=data1, hue='Target')
plt.title('Countplot for Albumin by Target')
plt.show()
```



In [23]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
```

In [24]:
```
data1['Gender of the patient'] = le.fit_transform(data1['Gender of the patient'])
```

In [25]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 582 entries, 0 to 581
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Age of the patient           582 non-null    int64  
 1   Gender of the patient        582 non-null    int32  
 2   Total Bilirubin              582 non-null    float64
 3   Direct Bilirubin             582 non-null    float64
 4   Alkaline Phosphatase         582 non-null    int64  
 5   Alanine Aminotransferase     582 non-null    int64  
 6   Aspartate Aminotransferase   582 non-null    int64  
 7   Total Proteins               582 non-null    float64
 8   Albumin                      582 non-null    float64
 9   Albumin and Globulin Ratio   582 non-null    float64
 10  Target                       582 non-null    int64  
dtypes: float64(5), int32(1), int64(5)
memory usage: 47.9 KB
```

In [26]:
```
print(data1['Target'].unique())
```

```
[1 2]
```

In [27]:
```
data1['Target'] = data1['Target'].replace(2, 0)
```

In [30]:
```
from sklearn.preprocessing import StandardScaler

# Create a StandardScaler object
scaler = StandardScaler()
# Split x and y
x = data1.drop(columns = ['Target'],axis=1) # Independent variables
y = data1['Target'] # Dependent / target variable
```

In [31]:
```
# Create a StandardScaler object
scaler = StandardScaler()

# Fit the scaler to the data
scaler.fit(x, y)

# Transform the data
x_std = scaler.transform(x)
```

In [32]:
```
from sklearn.model_selection import train_test_split
```

In [33]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=101)
```

```
In [34]: x_train
```

Out[34]:

	Age of the patient	Gender of the patient	Total Bilirubin	Direct Bilirubin	Alkaline Phosphatase	Alanine Aminotransferase	Aspartate Aminotransferase	Total Proteins	Albumin	Albumin and Globulin Ratio
38	47	1	2.7	1.3	275	123	73	6.2	3.3	1.1
432	41	0	0.9	0.2	201	31	24	7.6	3.8	1.0
288	57	1	4.5	2.3	315	120	105	7.0	4.0	1.3
517	45	1	2.9	1.4	210	74	68	7.2	3.6	1.0
179	75	1	2.8	1.3	250	23	29	2.7	0.9	0.5
...	...	...	...	...	...	...	...	...	...	...
393	48	0	1.0	0.3	310	37	56	5.9	2.5	0.7
75	31	1	0.9	0.2	518	189	17	5.3	2.3	0.7
575	32	1	15.0	8.2	289	58	80	5.3	2.2	0.7
337	75	1	1.8	0.8	405	79	50	6.1	2.9	0.9
523	29	1	0.8	0.2	156	12	15	6.8	3.7	1.1

407 rows × 10 columns

```
In [35]: x_test
```

Out[35]:

	Age of the patient	Gender of the patient	Total Bilirubin	Direct Bilirubin	Alkaline Phosphatase	Alanine Aminotransferase	Aspartate Aminotransferase	Total Proteins	Albumin	Albumin and Globulin Ratio
227	65	0	1.0	0.3	202	26	13	5.3	2.6	0.90
128	45	1	2.8	1.7	263	57	65	5.1	2.3	0.80
428	73	1	1.9	0.7	1750	102	141	5.5	2.0	0.50
141	30	1	1.6	0.4	332	84	139	5.6	2.7	0.90
422	53	1	1.6	0.9	178	44	59	6.5	3.9	1.50
...	...	...	...	...	...	...	...	...	...	...
536	10	0	0.8	0.1	395	25	75	7.6	3.6	0.90
567	20	0	16.7	8.4	200	91	101	6.9	3.5	1.02
317	38	1	3.7	2.2	216	179	232	7.8	4.5	1.30
333	13	0	0.7	0.2	350	17	24	7.4	4.0	1.10
127	58	0	1.7	0.8	1896	61	83	8.0	3.9	0.95

175 rows × 10 columns

```
In [36]: y_train
```

Out[36]:

```
38     1
432    0
288    1
517    1
179    1
      ..
393    1
75     1
575    1
337    1
523    0
Name: Target, Length: 407, dtype: int64
```

In [37]:
```
y_test
```

Out[37]:
```
227    0
128    1
428    1
141    1
422    0
      ..
536    1
567    1
317    1
333    1
127    1
Name: Target, Length: 175, dtype: int64
```

In [38]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

In [39]:
```
logistic_regression_model = LogisticRegression()
```

In [40]:
```
logistic_regression_model.fit(x,y)
```

Out[40]:
```
▾ LogisticRegression
LogisticRegression()
```

In [41]:
```
y_pred = logistic_regression_model.predict(x_test)
```

In [42]:
```
log1_accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {log1_accuracy:.2f}")
```

```
Accuracy: 0.70
```

In [43]:
```

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

In [44]:
```
# Define the hyperparameter grid to search
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100]
}
```

In [45]:
```

# Use GridSearchCV to search for the best hyperparameters
grid_search = GridSearchCV(estimator=logistic_regression_model, param_grid=param_grid, scoring='accura
grid_search.fit(x_train, y_train)
```

Out[45]:
```
▸ GridSearchCV
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

In [46]:
```
# Get the best hyperparameters
best_params = grid_search.best_params_
print(f'Best Hyperparameters: {best_params}')

```

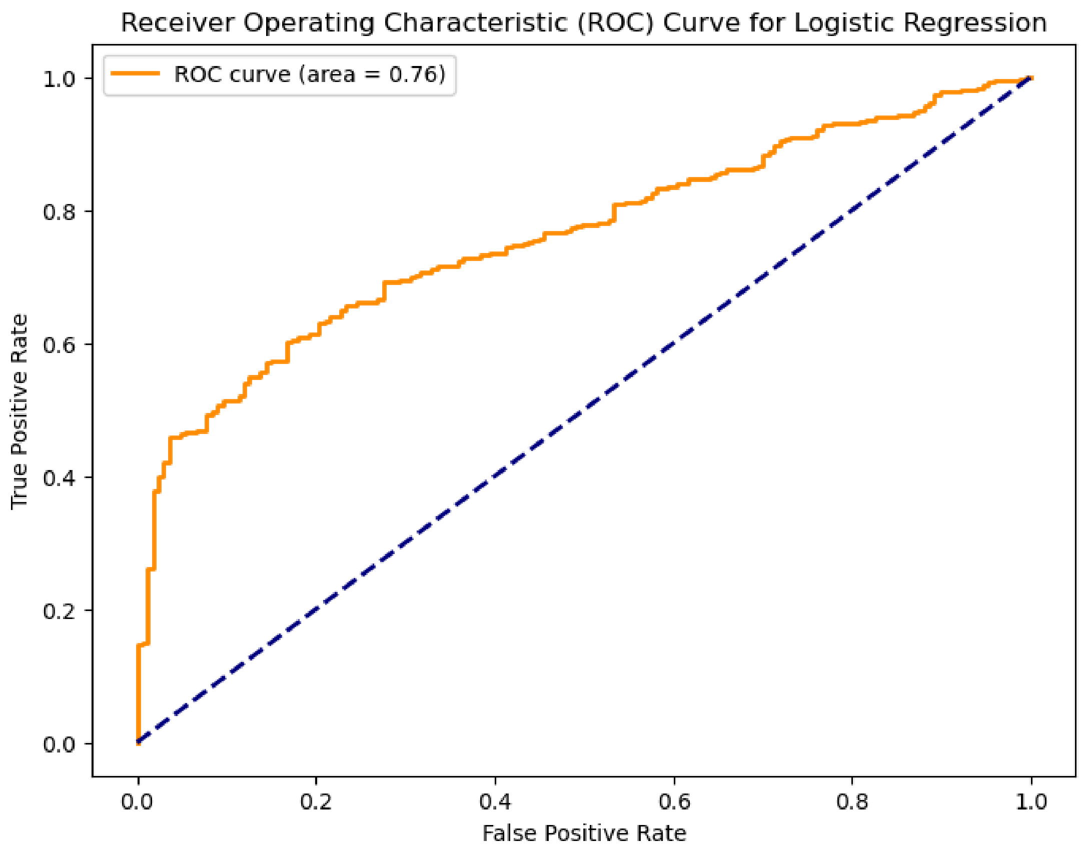
```
Best Hyperparameters: {'C': 0.001, 'penalty': 'l2'}
```

In [47]:
```
# Use the best model to make predictions
best_logreg_model = grid_search.best_estimator_
y_pred = best_logreg_model.predict(x_test)
```

In [48]:
```
# Evaluate the model
log_accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy on Test Set: {log_accuracy:.2f}')
```

```
Accuracy on Test Set: 0.75
```

In [49]:
```
from sklearn.metrics import roc_curve, auc
# Assuming your original target variable is in 'y'
# Use LabelEncoder to convert {1, 2} to {0, 1}
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Assuming you have a trained logistic regression model named 'logreg'
# Make sure to replace 'logreg' with the actual variable name of your logistic regression model
fpr, tpr, _ = roc_curve(y_encoded, logistic_regression_model.predict_proba(x)[:, 1])
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Logistic Regression')
plt.legend()
plt.show()

# Print classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
```



```
Classification Report:
               precision    recall  f1-score   support

           0       0.62      0.33      0.43        49
           1       0.78      0.92      0.84       126

    accuracy                           0.75       175
   macro avg       0.70      0.62      0.64       175
weighted avg       0.73      0.75      0.73       175
```

In [50]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [52]:
```
decision_tree_classifier = DecisionTreeClassifier(random_state=42)
```

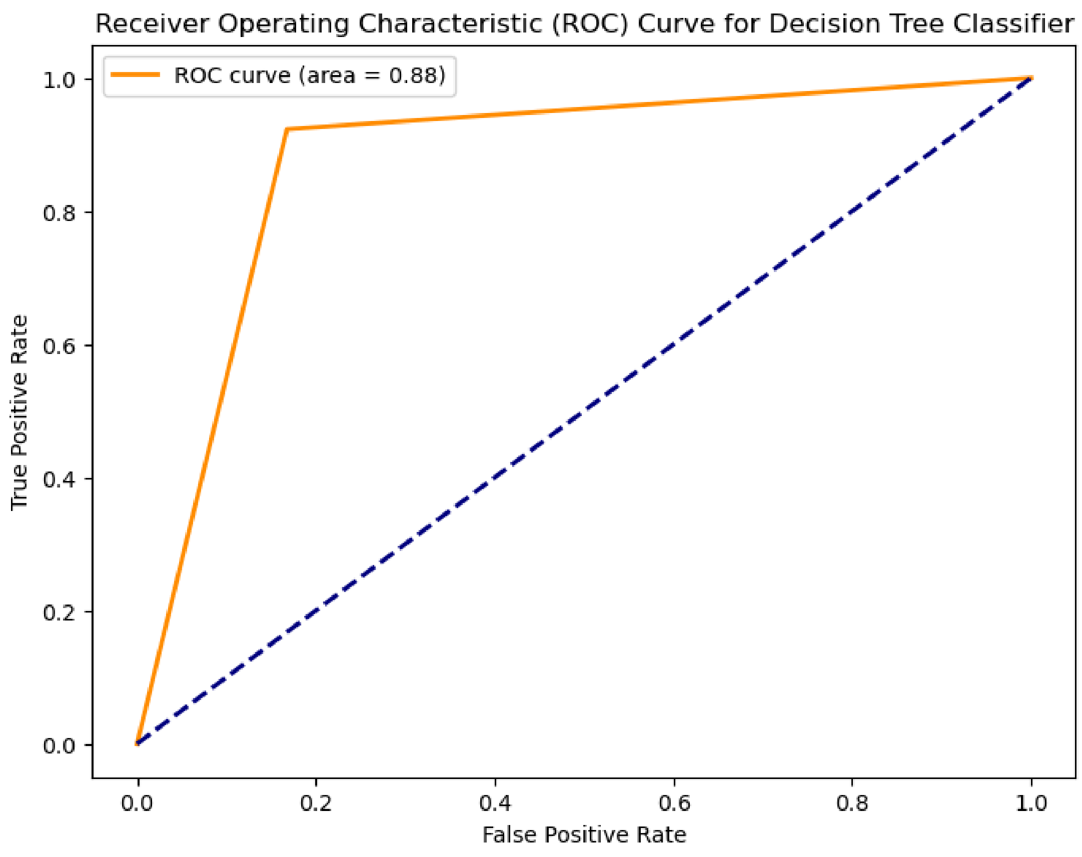
In [55]:
```
decision_tree_classifier.fit(x_train, y_train)
```

Out[55]:
```
▾         DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [56]:
```
y_pred = decision_tree_classifier.predict(x_test)
```

In [57]:
```
dt_accuracy = accuracy_score(y_test, y_pred)
```

In [58]:
```
fpr, tpr, _ = roc_curve(y_encoded, decision_tree_classifier.predict_proba(x)[:, 1])
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Decision Tree Classifier')
plt.legend()
plt.show()
# Print classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
```



```
Classification Report:
               precision    recall  f1-score   support

           0       0.40      0.43      0.41        49
           1       0.77      0.75      0.76       126

    accuracy                           0.66       175
   macro avg       0.58      0.59      0.58       175
weighted avg       0.67      0.66      0.66       175
```

In [59]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [60]:
```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [61]:
```
rf_classifier.fit(x_train, y_train)
```

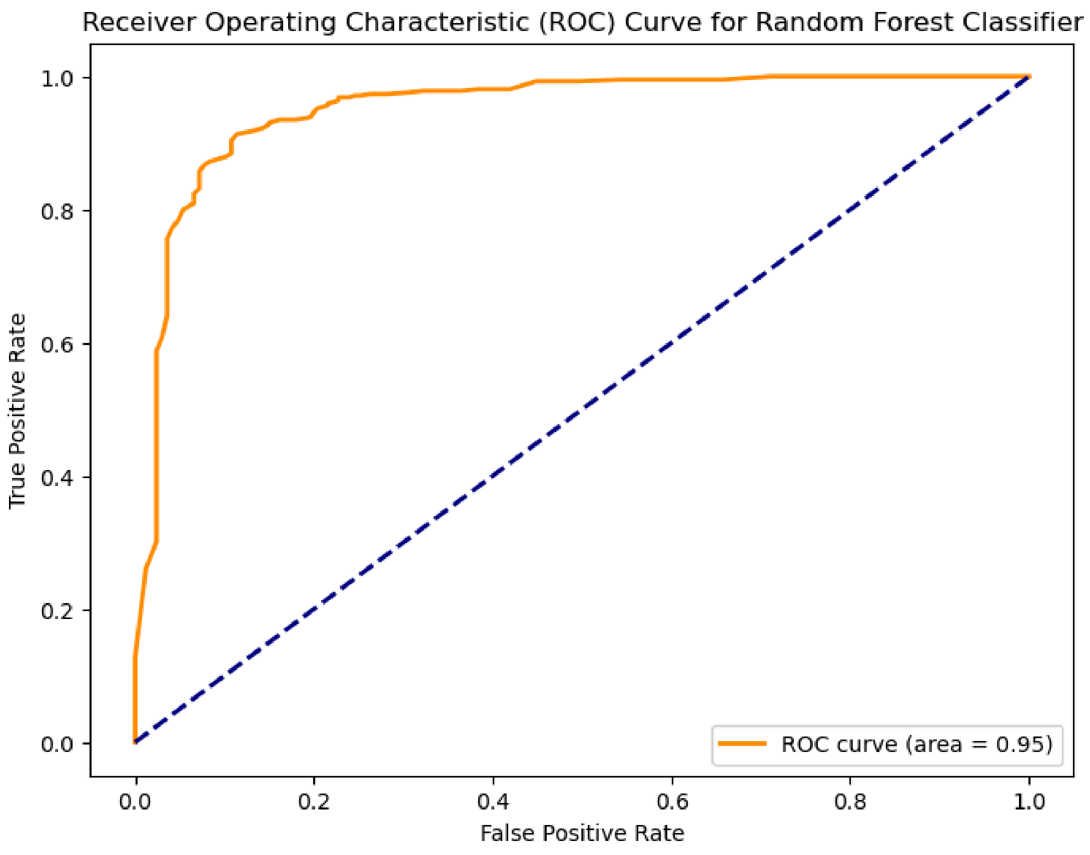
Out[61]:
```
▾         RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [62]:
```
y_pred = rf_classifier.predict(x_test)
```

In [63]:
```
rd_accuracy = accuracy_score(y_test, y_pred)
print(rd_accuracy)
```

```
0.6685714285714286
```

In [64]:
```
fpr, tpr, _ = roc_curve(y_encoded, rf_classifier.predict_proba(x)[:, 1])
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Random Forest Classifier')
plt.legend()
plt.show()

# Print classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
```



```
Classification Report:
               precision    recall  f1-score   support

           0       0.40      0.35      0.37        49
           1       0.76      0.79      0.78       126

    accuracy                           0.67       175
   macro avg       0.58      0.57      0.57       175
weighted avg       0.66      0.67      0.66       175
```

In [65]:
```
from sklearn.svm import SVC
```

In [66]:
```
svm_classifier = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)
```

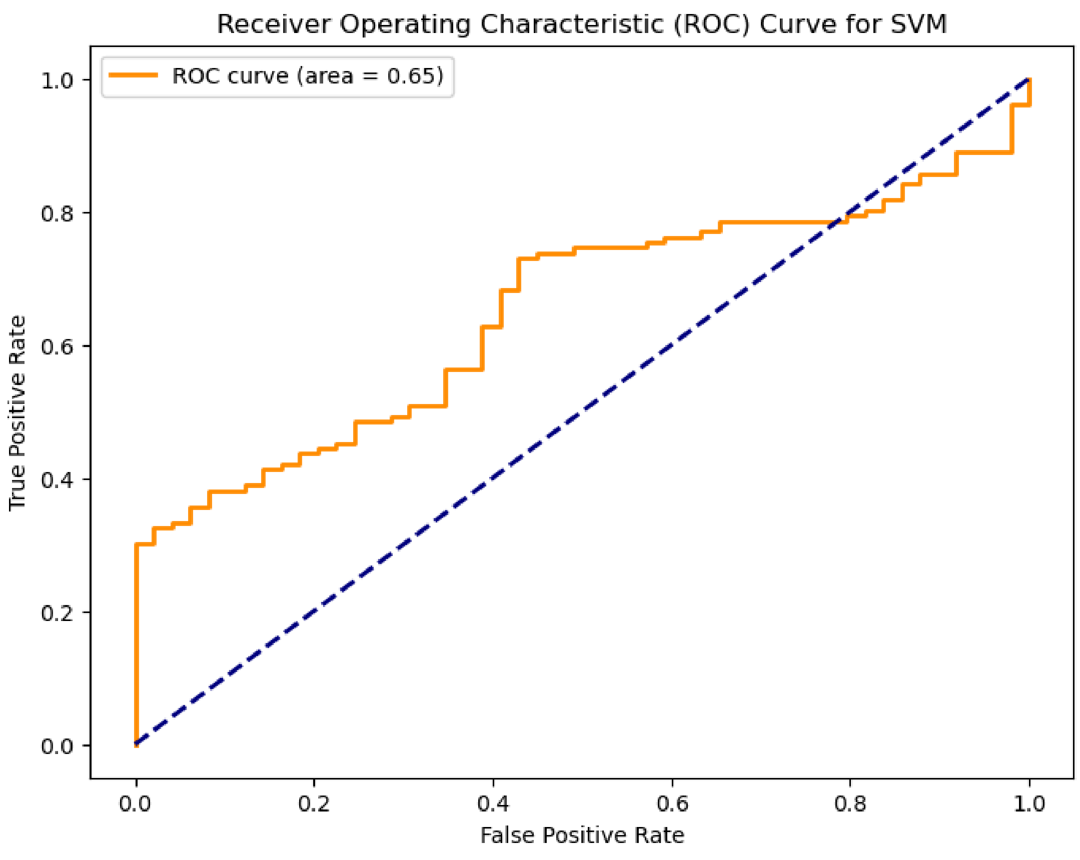
In [67]:
```
svm_classifier.fit(x_train, y_train)
```

Out[67]:
```
▾          SVC
SVC(random_state=42)
```

In [68]:
```
y_pred = svm_classifier.predict(x_test)
svc_accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy on Test Set: {svc_accuracy:.2f}')
```

```
Accuracy on Test Set: 0.72
```

In [69]:
```
# Assuming you have a trained SVM model
# Make sure to replace 'svm_model' with the actual variable name of your SVM model
svm_model = SVC(probability=True)  # Assuming a basic SVM model, you may need to adjust hyperparameter
svm_model.fit(x_train, y_train)

# Calculate ROC curve
fpr, tpr, _ = roc_curve(y_test, svm_model.predict_proba(x_test)[:, 1])
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for SVM')
plt.legend()
plt.show()

# Generate predictions on the test set
y_pred = svm_model.predict(x_test)

# Print classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
```



```
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00        49
           1       0.72      1.00      0.84       126

    accuracy                           0.72       175
   macro avg       0.36      0.50      0.42       175
weighted avg       0.52      0.72      0.60       175
```

In [70]:
```
#from sklearn import decision tree
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()#object creation for decision Tree
dt.fit(x_train,y_train)#training model
y_hat=dt.predict(x_test)#prediction
```

In [71]:
```

#test acc
test_acc=accuracy_score(y_test,y_hat)#testing accuracy
test_acc
```

Out[71]:
```
0.6685714285714286
```

In [72]:
```

## importing the model library
from sklearn.ensemble import GradientBoostingClassifier
gbm=GradientBoostingClassifier() ## object creation
gbm.fit(x_train,y_train) ## fitting the data
y_gbm=gbm.predict(x_test)#predicting the price
```

In [73]:
```
## evaluatin the model
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,classification_report
accu_scor=accuracy_score(y_test,y_gbm)
accu_scor
```

Out[73]:
```
0.6971428571428572
```

In [75]:
```
pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-2.0.3-py3-none-win_amd64.whl (99.8 MB)
                                              0.0/99.8 MB ? eta -:--:--
                                             0.0/99.8 MB 660.6 kB/s eta 0:02:31
                                              0.2/99.8 MB 2.0 MB/s eta 0:00:51
                                              0.4/99.8 MB 2.9 MB/s eta 0:00:34
                                              0.7/99.8 MB 3.4 MB/s eta 0:00:29
                                              0.9/99.8 MB 3.8 MB/s eta 0:00:26
                                              1.2/99.8 MB 4.1 MB/s eta 0:00:25
                                              1.4/99.8 MB 4.4 MB/s eta 0:00:23
                                              1.7/99.8 MB 4.6 MB/s eta 0:00:22
                                              2.1/99.8 MB 4.9 MB/s eta 0:00:20
     -                                        2.5/99.8 MB 5.5 MB/s eta 0:00:18
     -                                        2.9/99.8 MB 5.6 MB/s eta 0:00:18
     -                                        3.3/99.8 MB 5.8 MB/s eta 0:00:17
     -                                        3.7/99.8 MB 6.1 MB/s eta 0:00:16
     -                                        3.8/99.8 MB 6.2 MB/s eta 0:00:16
     -                                        4.3/99.8 MB 6.2 MB/s eta 0:00:16
     -                                        4.7/99.8 MB 6.4 MB/s eta 0:00:15
```

In [76]:
```
import xgboost as xgb
```

In [77]:
```
## model creation
from xgboost import XGBClassifier#importing the model library
xgb_r=XGBClassifier() ## object creation
```

In [78]:
```
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 407 entries, 38 to 523
Data columns (total 10 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Age of the patient          407 non-null    int64  
 1   Gender of the patient       407 non-null    int32  
 2   Total Bilirubin             407 non-null    float64
 3   Direct Bilirubin            407 non-null    float64
 4   Alkaline Phosphatase        407 non-null    int64  
 5   Alanine Aminotransferase    407 non-null    int64  
 6   Aspartate Aminotransferase  407 non-null    int64  
 7   Total Proteins              407 non-null    float64
 8   Albumin                     407 non-null    float64
 9   Albumin and Globulin Ratio  407 non-null    float64
dtypes: float64(5), int32(1), int64(4)
memory usage: 33.4 KB
```

In [79]:
```
x_train['Total Bilirubin']=x_train['Total Bilirubin'].astype('int64')
x_train['Direct Bilirubin']=x_train['Direct Bilirubin'].astype('int64')
x_train['Total Proteins']=x_train['Total Proteins'].astype('int64')
x_train['Albumin']=x_train['Albumin'] .astype('int64')
x_train['Albumin and Globulin Ratio']=x_train['Albumin and Globulin Ratio'].astype('int64')
```

In [80]:
```
## model creation
from xgboost import XGBClassifier#importing the model library
xgb_r=XGBClassifier() ## object creation
xgb_r.fit(x_train,y_train)# fitting the data
y_hat=xgb_r.predict(x_test)#predicting the price range
xgb1_accuracy=accuracy_score(y_test,y_hat)
print(xgb1_accuracy)
```

```
0.7371428571428571
```

In [81]:
```
param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'n_estimators': [100, 200, 300],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0],
}
```

In [82]:
```
grid_search = GridSearchCV(estimator=xgb_r, param_grid=param_grid, scoring='accuracy', cv=3, n_jobs=-1
```

In [84]:
```
grid_search.fit(x_train, y_train)
```

Out[84]:
```
▸        GridSearchCV
▸ estimator: XGBClassifier
      ▸ XGBClassifier
```

In [85]:
```
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)
```

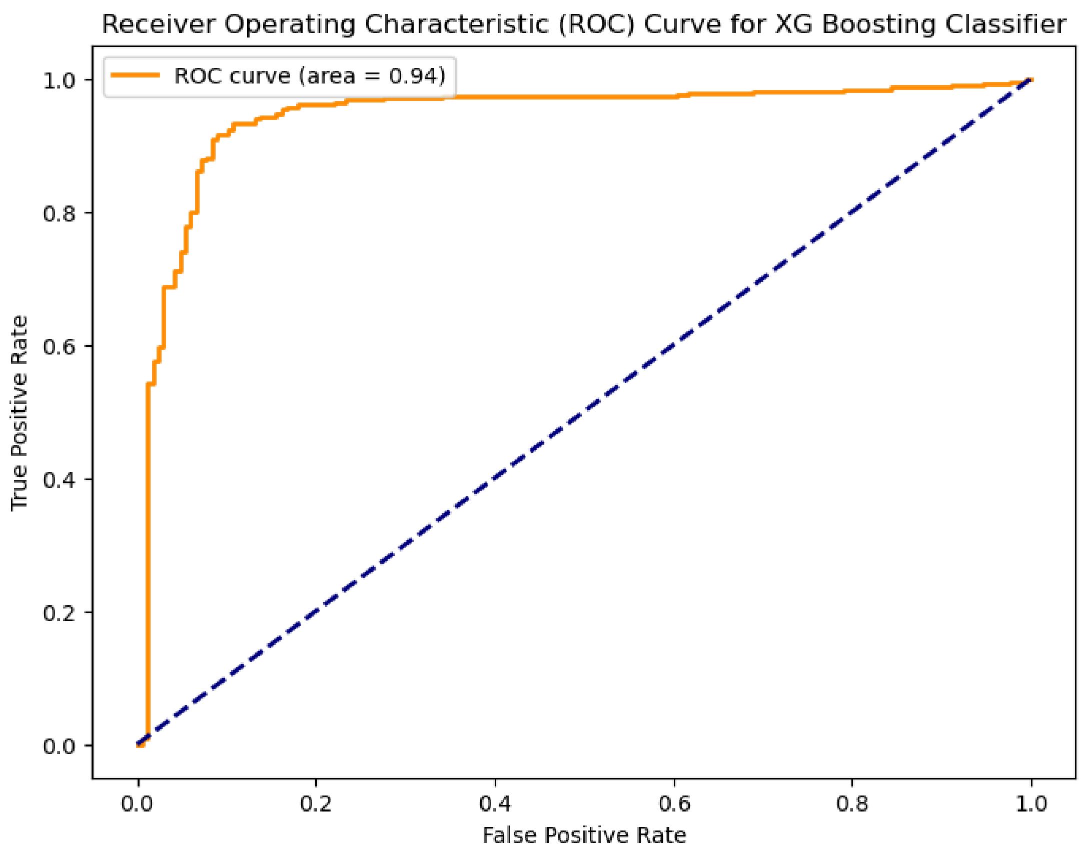
```
Best Hyperparameters: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators':
100, 'subsample': 0.9}
```

In [86]:
```
best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_test)
```

In [87]:
```
xgb_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", xgb_accuracy)
```

```
Accuracy: 0.7085714285714285
```

In [88]:
```
fpr, tpr, _ = roc_curve(y_encoded, xgb_r.predict_proba(x)[:, 1])
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for XG Boosting Classifier')
plt.legend()
plt.show()

# Print classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
```

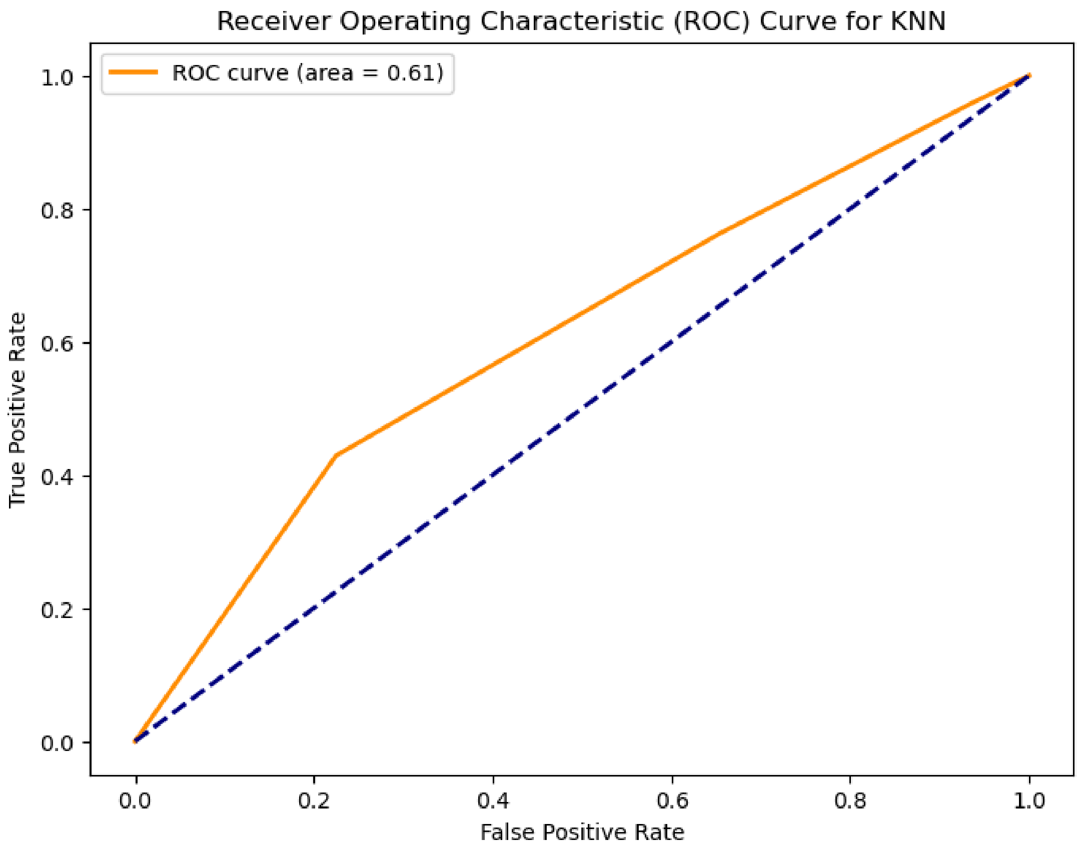


```
Classification Report:
               precision    recall  f1-score   support

           0       0.48      0.43      0.45        49
           1       0.79      0.82      0.80       126

    accuracy                           0.71       175
   macro avg       0.63      0.62      0.63       175
weighted avg       0.70      0.71      0.70       175
```

In [89]:
```
from sklearn.neighbors import KNeighborsClassifier

# Create KNN classifier with K=3 (for example)
knn = KNeighborsClassifier(n_neighbors=3)

# Fit the model to the training data
knn.fit(x_train, y_train)

# Make predictions on the test data
y_pred = knn.predict(x_test)
```

In [90]:
```
knn_accuracy = accuracy_score(y_test, y_pred)
```

In [91]:
```
# Calculate ROC curve
fpr, tpr, _ = roc_curve(y_test, knn.predict_proba(x_test)[:, 1])
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for KNN')
plt.legend()
plt.show()
# Print classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
```

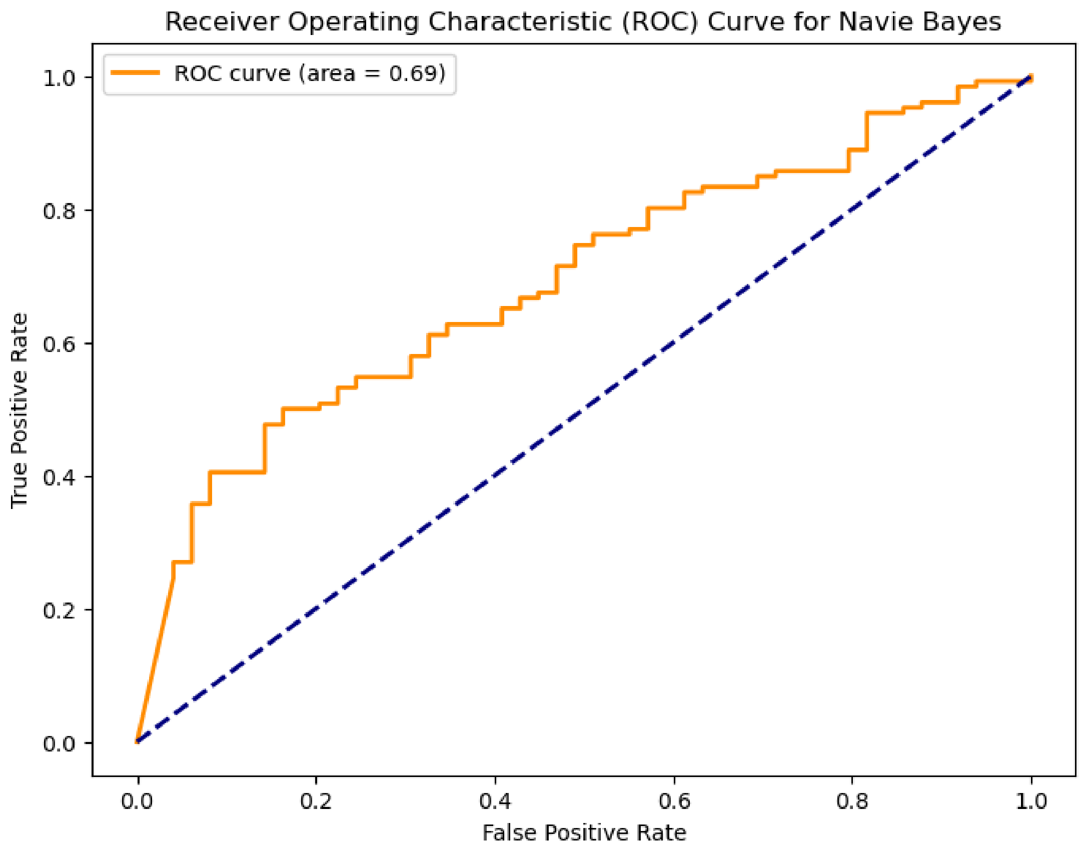


```
Classification Report:
               precision    recall  f1-score   support

           0       0.36      0.35      0.35        49
           1       0.75      0.76      0.76       126

    accuracy                           0.65       175
   macro avg       0.56      0.55      0.56       175
weighted avg       0.64      0.65      0.64       175
```

In [92]:
```
from sklearn.naive_bayes import GaussianNB

# Create Gaussian Naive Bayes classifier
naive_bayes = GaussianNB()

# Fit the model to the training data
naive_bayes.fit(x_train, y_train)

# Make predictions on the test data
y_pred = naive_bayes.predict(x_test)
```

In [93]:
```
nav_accuracy = accuracy_score(y_test, y_pred)
```

In [94]:
```
# Calculate ROC curve
fpr, tpr, _ = roc_curve(y_test, naive_bayes.predict_proba(x_test)[:, 1])
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Navie Bayes')
plt.legend()
plt.show()

# Print classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
```

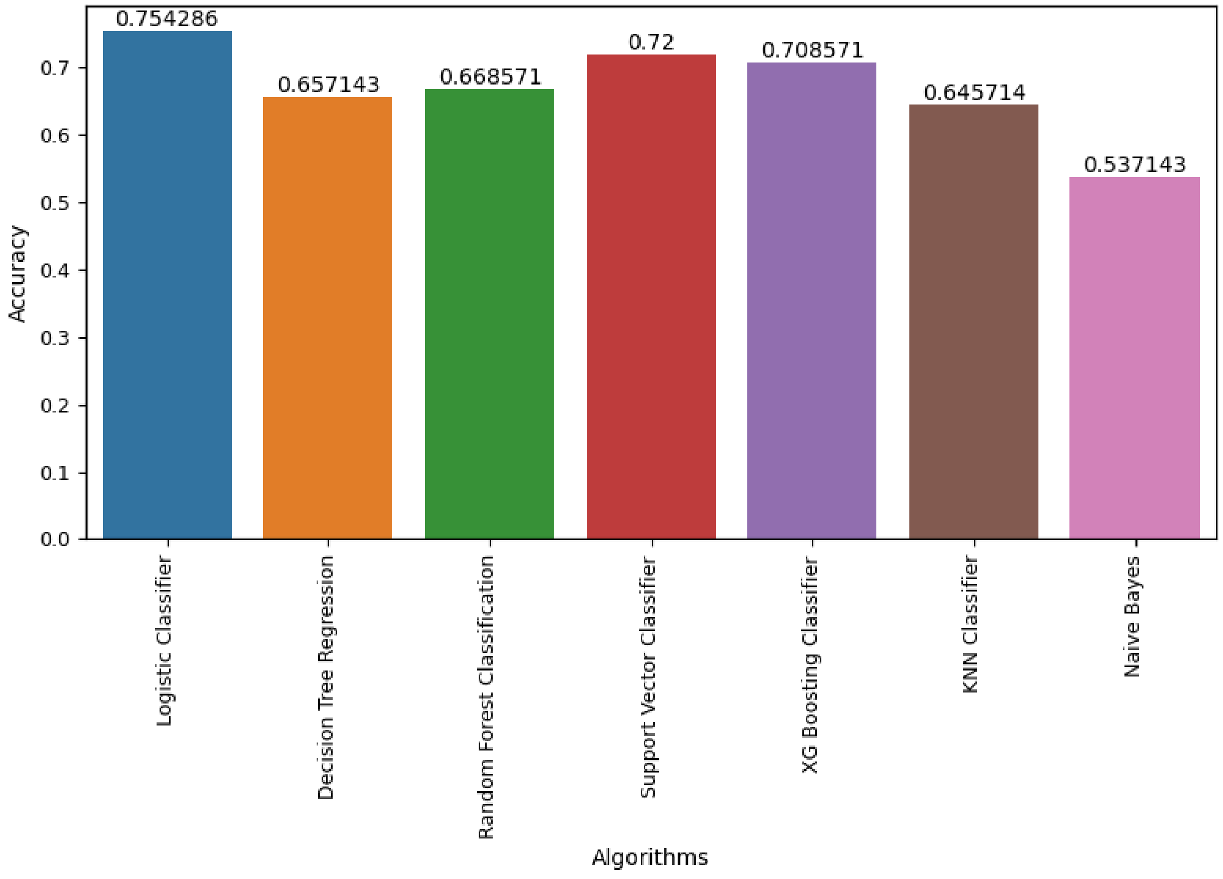


```
Classification Report:
               precision    recall  f1-score   support

           0       0.36      0.88      0.51        49
           1       0.89      0.40      0.56       126

    accuracy                           0.54       175
   macro avg       0.63      0.64      0.54       175
weighted avg       0.75      0.54      0.55       175
```

In [95]:
```
scores=[log_accuracy,dt_accuracy,rd_accuracy,svc_accuracy,xgb_accuracy,knn_accuracy,nav_accuracy]
algorithms=['Logistic Classifier','Decision Tree Regression','Random Forest Classification','Support V
            'XG Boosting Classifier','KNN Classifier','Naive Bayes']
for i in range(len(algorithms)):
    print("The Accuracy acheived using " + algorithms[i] + ' is: ' + str(scores[i])+"%")
```

```
The Accuracy acheived using Logistic Classifier is: 0.7542857142857143%
The Accuracy acheived using Decision Tree Regression is: 0.6571428571428571%
The Accuracy acheived using Random Forest Classification is: 0.6685714285714286%
The Accuracy acheived using Support Vector Classifier is: 0.72%
The Accuracy acheived using XG Boosting Classifier is: 0.7085714285714285%
The Accuracy acheived using KNN Classifier is: 0.6457142857142857%
The Accuracy acheived using Naive Bayes is: 0.5371428571428571%
```

In [96]:
```
# plotting the barplot between algoriths and their r2_scores
plt.figure(figsize=(8,6))
plt.xlabel("Algorithms")
plt.ylabel("Accuracy")
ax=sns.barplot(x=algorithms,y=scores)
for label in ax.containers:
    ax.bar_label(label)
    plt.xticks(rotation=90)
plt.tight_layout()
plt.tick_params(labelsize=9)
```



The project successfully addressed the problem of liver disease prediction using a liver patient dataset. Through detailed data analysis and the implementation of various classifiers, we gained insights into the dataset's characteristics. The models showed varying levels of accuracy and performance, with Random Forest and XGBoosting demonstrating higher efficacy. The ROC curves provided a visual representation of the models' ability to distinguish between positive and negative instances. The analysis contributes to building a predictive model for liver disease, offering potential benefits for early diagnosis and intervention.

In [ ]:
```
```