A STUDY ON AUTISM SPECTRUM DISORDER USING MACHINE LEARNING TECHNIQUE

IS4007 | Individual Activity3 | s15355 – Sandali Gunathilaka

ABSTRACT

Autism Spectrum Disorder (ASD) is a neurodevelopmental condition characterized by a range of difficulties in social interaction, communication, and behavior. Symptoms typically emerge in the first two years of life, though diagnosis can occur at any age. ASD, often described as a "behavioral disorder," starts in childhood and continues into adolescence and adulthood.

This study aims to develop a machine learning model to predict whether individuals have ASD or not through different variables of diagnostic measures with the aid of some machine learning techniques.

Use the ML Olympiad Autism Prediction Challenge provided by the Kaggle Dataset to achieve our aim. The Dataset consists of 11 individual characteristics and 10 behavioral features. 800 Records are included in the dataset. Splitting a Dataset into 70% and 30%. Here use 70% of the data for training a model and 30% of the data for testing a model.

Here, utilize Support Vector Machine, Logistic Regression, Random Forest, LightGBM, K-Neighbors Algorithm, and XGBoost to create predictive models. These models undergo rigorous training, and validation on the dataset after their performance is evaluated. After evaluating model performance determine which model is the best for predicting the ACD.

To find which model is the best model for predicting the individuals have ASD or not here gives different evaluation metrics, including accuracy, precision, r2_score, f1_score, recall, macro-average, weighted average, and ROC Area under the curve were used to assess the performance of the machine learning models. The Accuracy of all the models varies from 78% to 87.08%. and all the accuracy levels of the models are good. But the Highest accuracy given model gives the best accurate results. Therefore, a comparison between models gives that the K Neighbors Algorithm-based model achieved the highest accuracy of 87.08%. It can be concluded that the best prediction model for the prediction of autism spectrum disorder is given by the K-Neighbors algorithm.

Keywords: Autism Spectrum Disorder, Machine Learning, Support Vector Machine, Logistic Regression, Random Forest, LightGBM, K-Neighbors Algorithm, and XGBoost

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

The autism spectrum disorder (ASD) has across the world. Getting an accurate early diagnosis and intervention has increasingly been essential to improve the outcomes of individuals with ASD. However, at times it remains challenging due to the variability of symptoms and the nature associated with behavioral assessments. With the development of machine learning, it should be possible to develop predictive models that will help in the early detection of ASD.

Therefore, it can identify underlying indicators not manifest under traditional diagnostic conditions through data patterns under the guidance of machine learning algorithms. This research aims to develop a machine learning model to predict whether individuals have ASD or not through different variables of diagnostic measures with the aid of some machine learning techniques.

Objective of the study

This research aims to develop a machine learning model to predict whether individuals have ASD or not through different variables of diagnostic measures with the aid of some machine learning techniques.

To achieve the above objective, we have to follow the following steps in this research,

- Identify the key variables affecting ASD Symptoms.
- Develop a model for machine learning for predicting ASD.
- Evaluate the developed model performance using the dataset.
- Compare model performance using different machine learning techniques.
- Select the best-predicting model for ASD as a high-performance machine learning technique.

Considering the Significance of the Study, this study provides a proper and better diagnostic accuracy prediction model for ASD. It facilitates early detection models for improving an ASD person's quality of life and Save Diagnostic Costs. It improves diagnostic accuracy. With machine learning algorithms, we analyze data much more efficiently.

Another one of the most important benefits of this study is reduced processing time to determine ASD or not. The speed and efficiency in diagnosing health issues are paramount, particularly in cases of Autism. Traditional Diagnosing Autism can be a lengthy process, often taking up to six months, as it requires consultations with numerous specialists including developmental pediatricians, neurologists, psychiatrists, or psychologists. Currently, the traditional method of diagnosing Autism is a time-consuming and costly process. Therefore, Better ASD prediction models would lead to better and early responses to ASD Symptoms.

2.0 LITERATURE REVIEW

Nishat Akhtar and Mairead Feeney [6] conducted Predictive analytics using a Machine Learning model to recommend the most suitable intervention technology for Autism related deficits. Use training data and model built that was capable of efficiency of identifying and detecting for Autism related deficits. The first model Decision Tree Classifier had 67% accuracy and the second model named the Ensemble Vote Classifier, contained 75% of accuracy. As a result, it is applied to autistic people, and the findings of the study can be used to compare models. According to these details best highest accuracy given by the Ensemble Vote Classifier which is 75% accuracy.

Sarif and Khan [7] employed various machine learning algorithms, including Linear Discriminant Analysis, Random Forest, SVM, K-Nearest Neighbors (KNN), and Multi-layer Perceptron, to detect ASD using the ABIDE-I dataset. Their results indicated accuracies ranging from 55% to 65%.

Usta et al. [8] intended to test machine learning methods on big datasets to determine outcome factors. The 254 baseline form components were used to assess four machine learning methods: Naive Bayes, Generalized Linear Model (GLM), Logistic Regression, and Decision Tree. They got the best AUC and accuracy using a decision tree (AUC = 70.7%, sensitivity = 81.1%, specificity = 61.3%)

Heinsfeld et al [9] (2018) applied a deep learning algorithm and neural network to identify ASD patients using a sizable brain imaging dataset from the Autism Imaging Data Exchange (ABIDE I) and achieved a mean classification accuracy of 70% with an accuracy range of 66% to 71%. The SVM classifier achieved a mean accuracy of 65%; while the random forest classifier (RFC) achieved a mean accuracy of 63%.

Sherkatghanad et al [10] (2020) conducted the research Automated Detection of Autism Spectrum Disorder Using a Convolutional Neural Network (CNN) model on the WHO, ASD affects one patient dataset and achieved an accuracy rate of 70.22%.

In all of the above studies, the major problem is that they do not offer high accuracy in the prediction of autism spectrum disorder. Therefore, we cannot rely on such prediction results to conclude if the individuals have the disorder or not. These accuracy rates cannot be enough to correctly point out people with autism. As such, we aim to develop a model with the best accuracy in the prediction of ASD.

3.0 DATA

3.1 VARIABLE DESCRIPTION OF DATASET

For this project, utilized a dataset from the ML Olympiad Autism Prediction Challenge provided by Kaggle. The dataset comprises demographic information and the Autism Quotient Test (AQ), featuring 10. This dataset was selected because of its prevalent use in ASD studies. The primary measure is the AQ questionnaire. This questionnaire assesses attention switching, attention to detail, communication, and imagination, offering a score that indicates 'Autistic-like' behavior based on ten specific questions answered by parents, family members, or professionals.

The Dataset consists of 11 individual characteristics and 10 behavioral features. 800 Records are included in the dataset.

Below is a detailed description of the dataset features used in this study:

Table 1 : Variable Description

Variable	Description
ID	The ID of the patient
A1_Score to A10_Score	Score based on Autism Spectrum Quotient (AQ) 10-item screening tool
age	Age of the patient in years
gender	The gender of the patient
ethnicity	The ethnicity of the patient
jaundice	Whether the patient had jaundice at the time of birth
autism	Whether an immediate family member has been diagnosed with autism
contry_of_res	Country of residence of the patient
used_app_before	Whether the patient has undergone a screening test before
result	Score for AQ1-10 screening test
relation	Relation of patient who completed the test
Class/ASD	Classified result as 0 or 1. Here 0 represents No and 1 represents Yes. This is the target column

Here A1_A10 variables collect questionary as follows:

Table 2 : A1_A10 variable collect questionary

A1_Score	I often notice small sounds when others do not.
A2_Score	I usually concentrate more on the whole picture, rather than the small details.
A3_Score	I find it easy to do more than one thing at once.

A4_Score	If there is an interruption, I can switch back to what I was doing very quickly.
A5_Score	I find it easy to read between the lines when someone is talking to me.
A6_Score	I know how to tell if someone listening to me is getting bored.
A7_Score	When I'm reading a story, I find it difficult to work out the character's intention.
A8_Score	I like to collect information about categories of things (e.g. types of cars, types of birds, types of trains, types of plants, etc.).
A9_Score	I find it easy to work out what someone is thinking or feeling just by looking at their face.
A10_Score	I find it difficult to work out people's intentions.

Dataset data types are as follows:

Table 3 : Data Type

Variable	Data Type
A1_Score	Binary(0,1)
A2_Score	Binary(0,1)
A3_Score	Binary(0,1)
A4_Score	Binary(0,1)
A5_Score	Binary(0,1)
A6_Score	Binary(0,1)
A7_Score	Binary(0,1)
A8_Score	Binary(0,1)
A9_Score	Binary (0,1)
A10_Score	Binary (0,1)
Age	String
Gender	Boolean('f', 'm')
Ethnicity	String
Jaundice	Boolean('YES', 'NO)
Autism	Boolean('YES', 'NO)
Country of Residence	String
Used app before	Boolean('YES', 'NO)
Relation	String
Class/ASD	Binary(0,1)

3.2 DATA PREPROCESSING

A well-organized pipeline and a number of performance metrics are used to properly examine the logistic regression model's performance. The pipeline starts with a ColumnTransformer that uses StandardScaler for numerical features and OrdinalEncoder for categorical features to analyze both numeric and categorical input. Before putting the data into the logistic regression model, the Synthetic Minority Over-sampling Technique, or SMOTE, is used to address class imbalance.

- ### 3.2.1 Missing Values Imputation

```
df[df == '?'] = np.nan
df.isna().any()
```

```
ID                 False
A1_Score           False
A2_Score           False
A3_Score           False
A4_Score           False
A5_Score           False
A6_Score           False
A7_Score           False
A8_Score           False
A9_Score           False
A10_Score          False
age                False
gender             False
ethnicity           True
jaundice           False
autism             False
contry_of_res      False
used_app_before    False
result             False
age_desc           False
relation            True
Class_ASD          False
dtype: bool
```

```
df.isnull().sum()
```

```
ID                   0
A1_Score             0
A2_Score             0
A3_Score             0
A4_Score             0
A5_Score             0
A6_Score             0
A7_Score             0
A8_Score             0
A9_Score             0
A10_Score            0
age                  0
gender               0
ethnicity          151
jaundice             0
autism               0
contry_of_res        0
used_app_before      0
result               0
age_desc             0
relation            77
Class_ASD            0
dtype: int64
```

Figure 1: Compute the Missing Values

The dataset has a few unusual observations like '?'. First record these unusual observations into the missing values and after checking the missing values it shows that the ethnicity column and relation columns have missing values. These variables are categorical. Therefore, missing values can recorded by these variables using their mode value. This way imputes missing values and after checking null values it shows no missing values in the dataset.

- ### 3.2.2 Preprocessing Using Pipeline

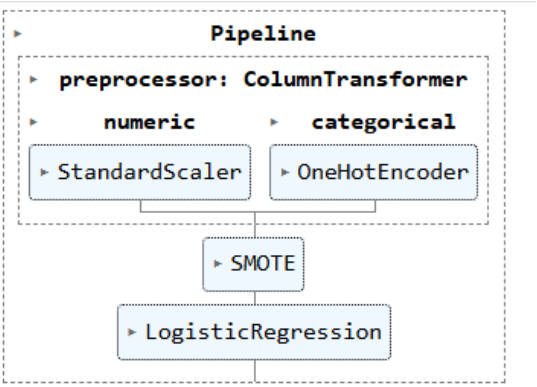


Figure 2: Pipeline Model

3.2.2.1Encoding Categorical Data

Here the OneHotEncoder technique to convert categorical features ('gender', 'ethnicity', 'jaundice', 'autism', 'contry_of_res') into binary format for the training dataset. This method allows our machine-learning models to process the data effectively.

3.2.2.2 Scaling Numerical Features

By using, the Standard Scaler Method for numerical features ('result') values are centered around the mean zero and the standard deviation is one.

3.2.2.4 SMOTE for Balancing the Dataset

```
Counts of label in y_train '1': 134
Counts of label in y_train '0': 426
```

Figure 3: Check target variable class imbalance

Initially checking the Counts of the 1 and 0 in the target variable it's showing by above figure_____. Therefore, to address the class imbalance, we applied SMOTE (Synthetic Minority Over-sampling Technique) for the pipeline.

- **3.2.3 Data Splitting**

We split the dataset into training and testing partitions to ensure the model's effectiveness with new, unseen data. The split maintained an 80%of data for test data and 30% of data for train data.

4.0 THEORY AND METHODOLOGY

The proposed workflow involves several key steps: pre-processing the data, training and testing with specified models, evaluating the results, and predicting ASD. This entire process is implemented using Anaconda python Jupiter notebook.

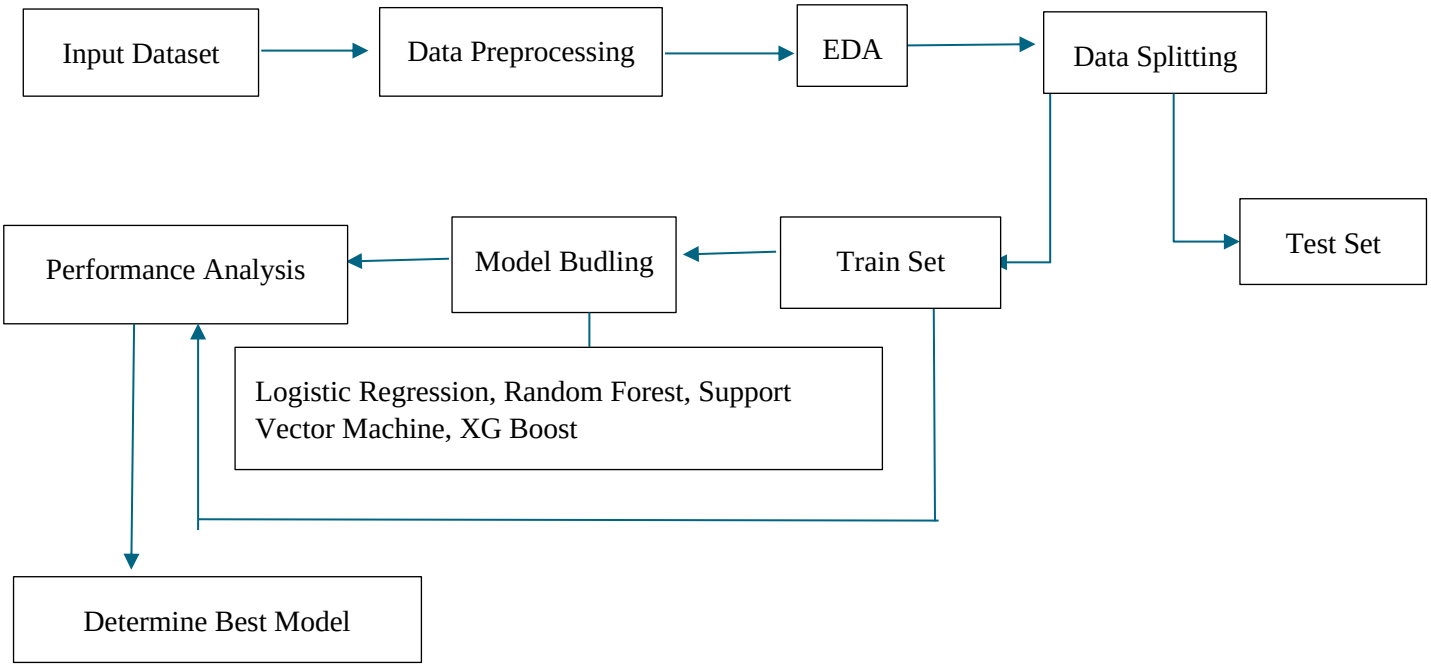


Figure 4: Methodology for the Process for ML Methods

- **4.1 Chi-Square Test**

  Chi-Squared Test to investigate the association between two variables like these with the following hypotheses:

  H0 : There is no association between two variables.

  H1 : There is an association between two variables.

- **4.2 Training and Testing Model**

The dataset has been divided into two segments: 70% for training and 30% for testing. Illustrates the final configuration of the training, testing, and validation sets used for classification.

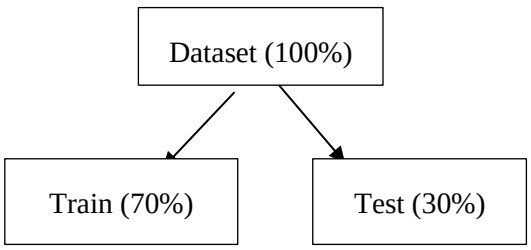


Figure 5: Dataset Split for Train and Test Dataset

- **4.3 Logistic Regression**

Logistic Regression is a regression tool used to analyze binary dependent variables, with output values of either 0 or 1. It is applied to continuous value datasets and explains the relationship between a binary dependent variable and a nominal or ordinal independent variable. The model is represented by the sigmoidal function.

Sigmoid Function: $f(x) = \frac{1}{1+e^{-x}}$

- **4.4 Random Forest**

Random Forest is an ensemble learning algorithm that constructs numerous decision trees during training. Each tree is trained on random subsets of the data, and the final prediction is made by aggregating (voting or averaging) the predictions from individual trees. This algorithm is favored for its ability to manage complex data relationships, resist overfitting, and remain robust against noise. Key elements in the construction of decision trees within Random Forests include Gini Impurity, Information Gain, and Entropy, all of which are crucial to the algorithm's effectiveness.

- **4.5 Support Vector Machine (SVM)**

Support Vector Machine (SVM) is a supervised learning algorithm primarily used for classification tasks. It seeks to find a hyperplane that effectively separates data into distinct classes. The "support vectors" are the data points nearest to the decision boundary, and the margin of the distance between the hyperplane and these observations is maximized to improve robustness, meaning the maximum separation between classes. SVM, a linear supervised learning method, is also applicable to regression tasks and pattern recognition problems, and it effectively avoids overfitting by defining a clear decision boundary.

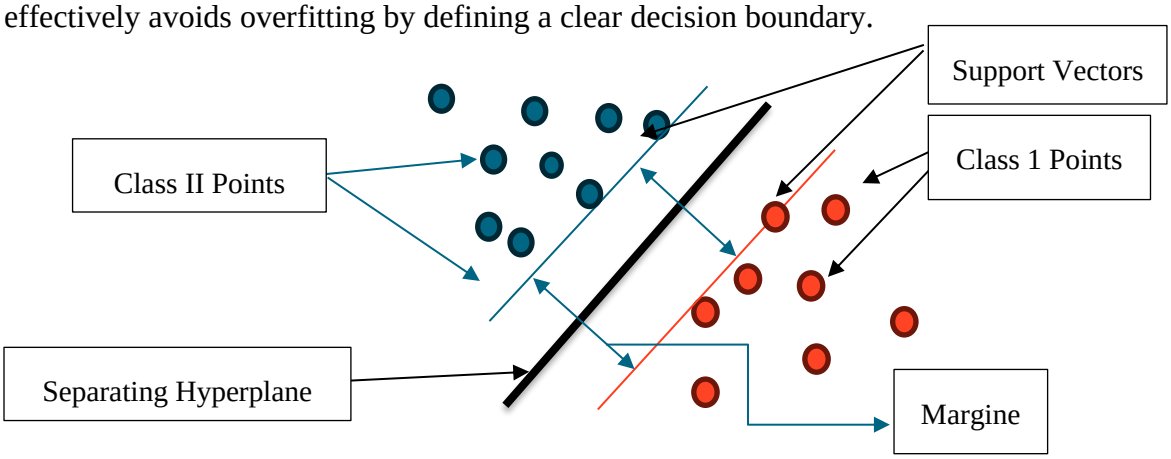


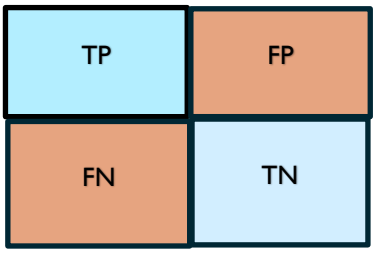
Figure 6: SVM find support vectors

- **4.6 XG Boost**

eXtreme Gradient Boosting (XGBoost) is an ensemble learning algorithm optimized for speed and performance. It constructs a sequence of decision trees, each aimed at correcting the errors of its predecessor. XGBoost incorporates a regularization term to manage model complexity and prevent overfitting while using a gradient-based optimization strategy for efficient training. The primary goal of XGBoost is to minimize a loss function, which quantifies the difference between predicted values and true labels.

- **4.7 Confusion Matrix**

True Positive (TP): The actual value was positive, and the model predicted a positive value.
True Negative (TN): The actual value was negative, and the model predicted a negative value.
False Positive (FP) – Type I Error: The actual value was negative, but the model predicted a positive value.
False Negative (FN) – Type II Error: The actual value was positive, but the model predicted a negative value.

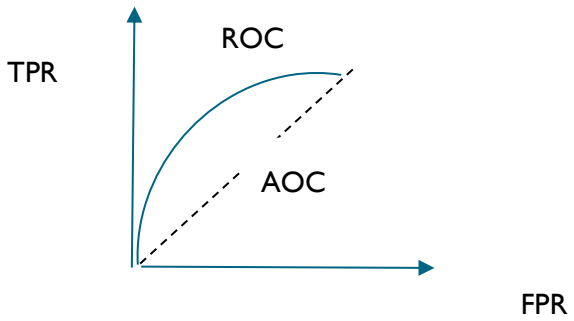


- **4.8 ROC Curve**

The ROC curve, or Receiver Operating Characteristic curve, is a graphical representation used in binary classification. It plots the True Positive Rate (TPR) on the y-axis against the False Positive Rate (FPR) on the x-axis.

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{TN + FP}$$

To create the ROC curve, TPR and FPR values are plotted at various threshold levels. The area under the ROC curve (AUC) ranges from 0 to 1. A completely random model, which is represented by a straight line, has a 0.5 ROC. The amount of deviation a ROC has from this straight line denotes the efficiency of the model.



- **4.9 Accuracy, Precision, Recall, F1 Score, Macro Average and r2 Score**

Accuracy is a key metric that evaluates a model's correctness. It is determined by the ratio of correct predictions to the total number of predictions. The formula for calculating accuracy is:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

Precision tells us how many of the correctly predicted cases actually turned out to be positive.

$$Precision = \frac{TP}{TP + FP}$$

Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives. Recall is the Fraction of positives that were correctly identified.

$$Recall = \frac{TP}{TP + FN}$$

F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. The F1 Score measures how well our model balances between precision and recall.

$$F1\ Score = \frac{2 * Recall * Precision}{Recall + Precision}$$

Macro Average Simply the average precision, recall, and f1 score between classes. Macro treats all classes equally.

r2 (r squared) is also known as the Coefficient of Determination sometimes also known as Goodness of fit. Otherwise, it's calculated how well the model fits the data.

5.0 EXPLORATORY DATA ANALYSIS

This section aims to analyze how dataset variables impact autism patients.

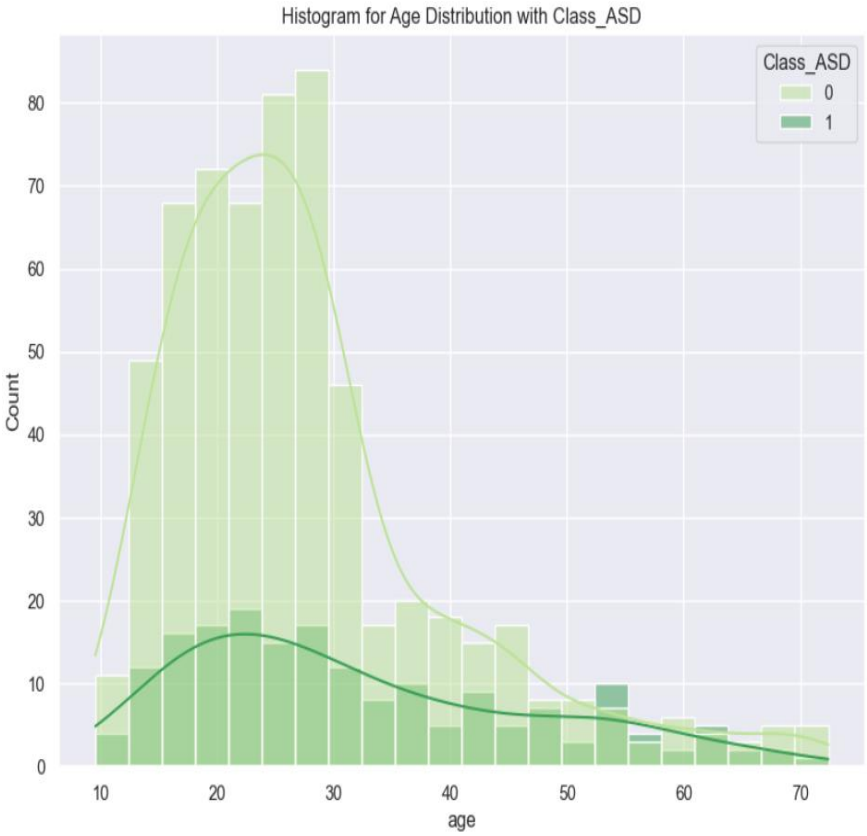


Figure 7: Histogram for Age Distribution with Class_ASD

This valuable plot (Figure 7) shows the age distribution of the dataset individuals within two district classes in the target variable Class_ASD which is autism spectrum disorder (ASD) have or not. The histogram plot shows most of the individuals are in both cases (ASD have or not) distributed in the lower age range which peaks around 20-30 years. Furthermore, Notably, the class '0' (ASD not have) histogram shows higher counts across most age ranges compared to class '1' (ASD have). These two distributions are positively skewed. These insights indicate valuable information for understanding the age distribution of the target variable (Class_ASD).

This density plot (Figure 8) indicates the distribution of screening test results (Score for AQ1-10 screening test) for Class_ASD (ASD have or not). Here Class_ASD is 0 test results are approximately showing normal distribution and Class_ASD is 1 test score shows a negative skewed distribution. Observations indicate that individuals without ASD typically have lower result scores with peak density around a score of 5. Conversely, Individuals with ASD show peak density between scores of 10 and 12, indicating higher test scores. The overlap between the

Figure 8 : Screening Test Result Distribution

two density plots highlights some common test scores. However, each group's density plot peaks are distinct. Therefore, This plot provides valuable information on the screen test results between individuals with and without ASD.

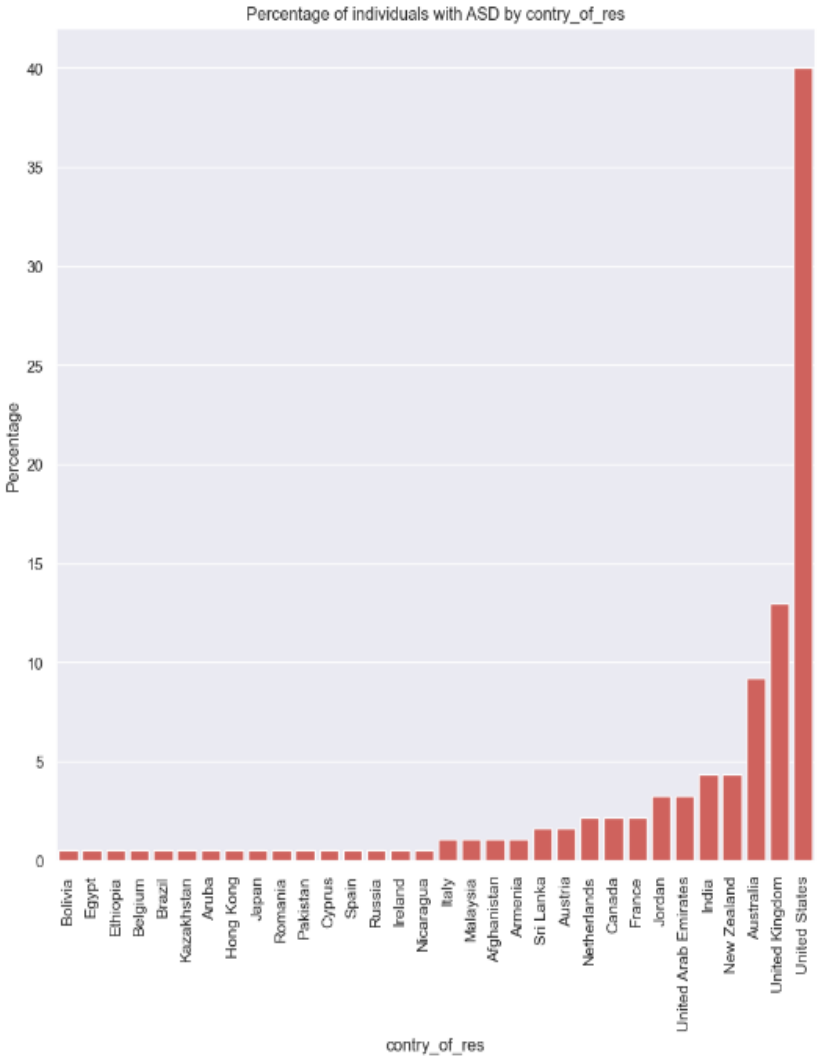


Figure 9 : Percentage of Individuals with ASD by Country_of_res

The bar chart (Figure 10) shows the percentage of individuals with autism spectrum disorder (ASD) across different countries. It indicates the variability in ASD prevalence by country. Then, this sheds light on geographical disparities.
Considering key observations includes are, The United States (US) has the highest percentage of individuals with ASD. It significantly surpassing all other countries with percentage rate exceedingly more than 35%. This is provided notably high diagnosis or reporting rate in the US. By the way, Countries like the United Kingdom (UK) and Australia also provided relatively high percentages compared to the other countries excluding the US, with rates around 7.5% - 15%. Other several countries show very low percentage rates of ASD, often close to zero. They include countries like Brazil, Russia, Japan, Ethiopia, Egypt, Hong Kong, and Pakistan. So, this could be due to some cases like underdiagnosis. A range of some other countries has moderate prevalence rates between nearly 1-5% shows. They are countries like France, Netherlands, Austria, and Sri Lanka. It can indicate a case to balanced level of awareness or reporting. By considering all these cases this bar chart provided useful insight for understanding the distribution of ASD percentage across different regions of countries in the dataset.

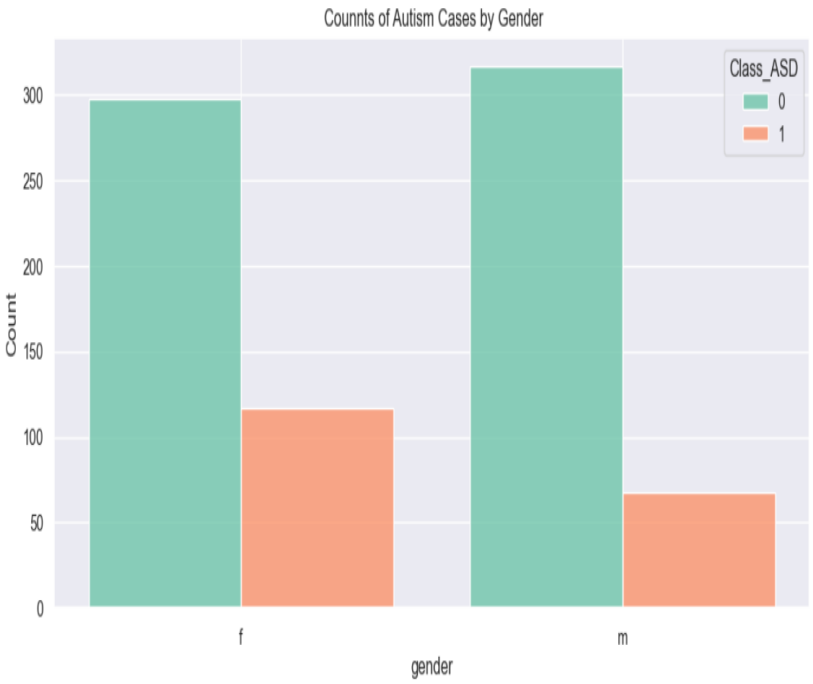


Figure 10 : Counts of Autism Cases by Gender

This bar chart (Figure 10) provides the counts of Class_ASD (Autism have or not) cases by gender (Male and Female). The chart indicates that in autistic individuals female group count is greater than compare of the male group. Here both genders show a significant number of individuals greater than those without ASD (Class_ASD='0') compared to those with ASD (Class_ASD='1'). Therefore, this plot clearly shows an understanding of the count of ASD cases by gender.

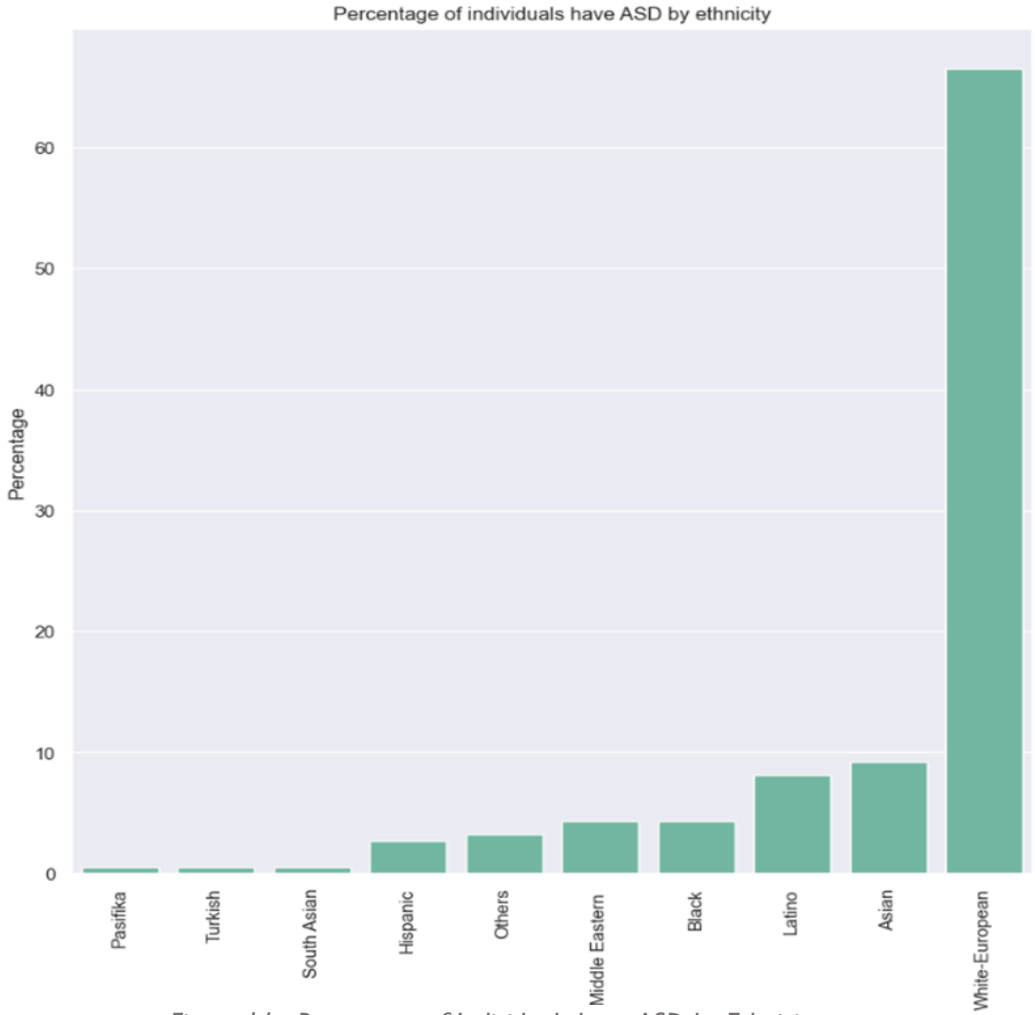


Figure 11 : Percentage of Individuals have ASD by Ethnicity

This bar chat (Figure 11) represented by the percentage of individuals with autism spectrum disorder (ASD) across different ethnicities. Here different ethnicities such as Pasifika, Turkish, South Asian, Hispanic, Others, Middle Eastern, Black, Latino, Asian, and White European show percentages of individuals who have been diagnosed with ASD (Class_ASD='1'). The results show that the percentage of people with ASD is higher among White Europeans compared to any other group. White Europeans exceed 60%. The percentage of another ethnicity is lower than 10%. But Pasifika, Turkish, and South Asian have very low percentages compared to the other ethnic groups. Furthermore, this plot clearly represented how parentage ethnicity groups are represented in the dataset according to the ascending order. The data shows a stark contrast between White Europeans and all other ethnicities. This plot serves as an initial step in understanding the distribution of ASD across ethnicities.

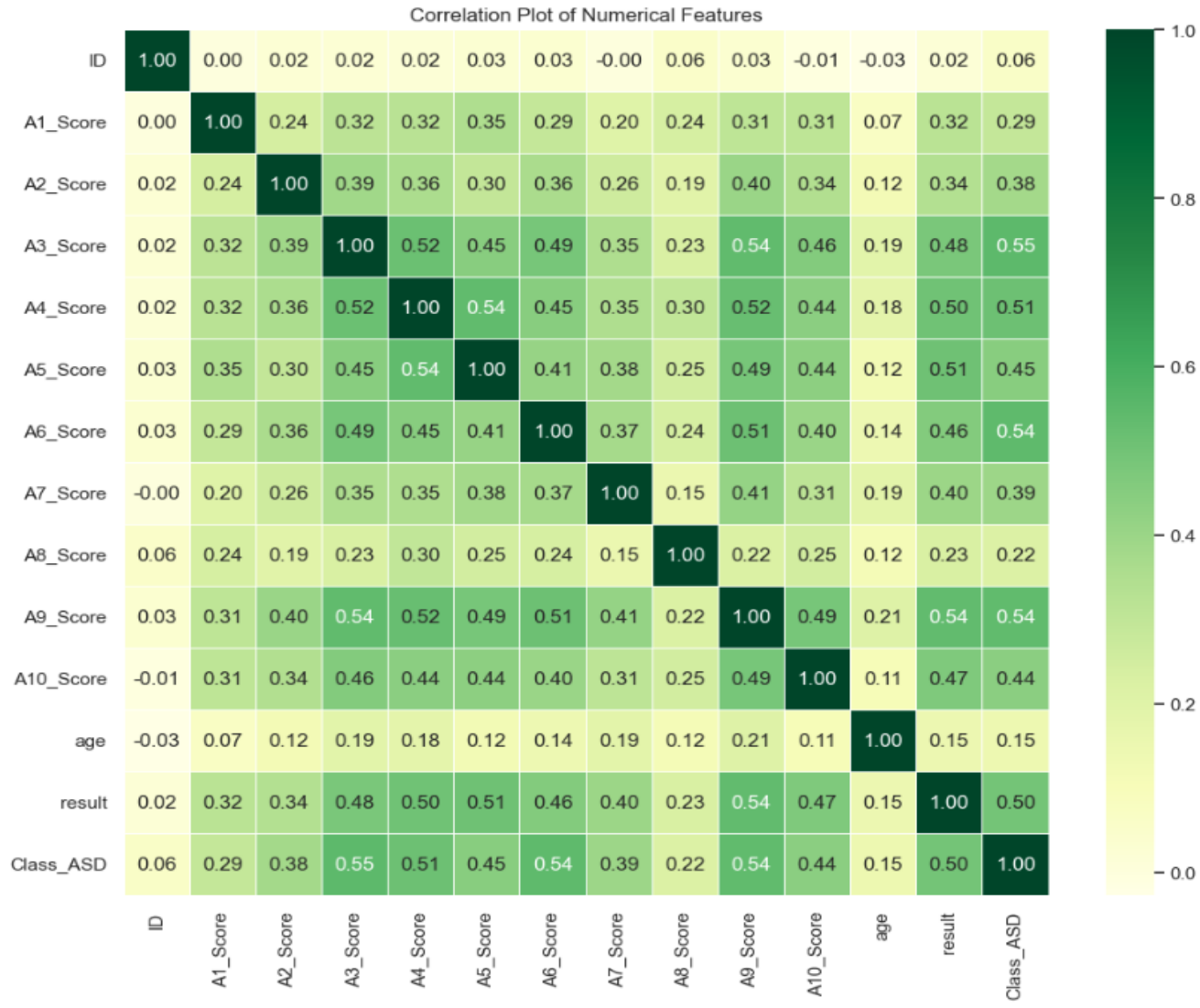


Figure 12: Correlation Map for the Continuous Variables

This correlation plot (Figure 12) provides an important analysis thorough visual representation of the correlations between the dataset's numerical factors. The correlation coefficients are shown on the heatmap. The values range from -1 to 1, with values nearer 1 indicating a strong positive correlation, values nearer -1 indicating a strong negative correlation, and values near 0 indicating a very weak or no association. There are notable correlations between the target variable Class_ASD and several factors. The A3_Score (0.55), A6_Score (0.54), and A9_Score (0.54) show strong positive correlations with the target variable. It suggests that these features are good predictors for ASD classification. But here, A1_Score, A8_Score, age variables have less correlation between Class_ASD (<0.4). Then we can determine when we developed the predicted model for the target variable Class_ASD that these three variables are not significantly affected by the target variable. Therefore, these variables can be removed from when we train the model for better accuracy.

Chi-Square Test Results:

	Feature	p-value
0	gender	5.696647e-04
1	ethnicity	1.510274e-10
2	jaundice	1.169010e-07
3	autism	5.258030e-26
4	contry_of_res	1.346083e-09
5	used_app_before	8.076826e-01
6	age_desc	1.000000e+00
7	relation	8.661287e-02

Figure 13 : Chi-Square Test Results for Select the Categorical Variables

Numerical variables are only shown in the correlation plot (Figure 13). Therefore, if we consider what are the most affected categorical variables to the target variable for developing the predicted model for Class_ASD we have to do the Chi-Square Test. Considering Hypothesis testing,

H0: There is no association between the categorical feature and target variable Class_ASD.

H1: There is an association between categorical feature and target variable Class_ASD.

Here used_app_before, age_desc, relation variables p-value (<0.05). Then these variables do not significantly affect the Class_ASD. Therefore, these categorical variables can be removed from the study when we develop the train model for predicted Class_ASD.

6.0 ADVANCE ANALYSIS

Here Analysis Comparison between 6 Machine Learning Models to Find which model is the best Model for Predicting whether an individual has autism or not.

1) Logistic Regression Model
2) Support Vector Classification Model
3) LightGBM Model
4) Random Forest Model
5) K-Neighbors Classification Model
6) XGBoost Model

CLASSIFICATION REPORTS FOR COMPARISON TO THE MODELS

Accuracy Score: 0.7833333333333333
Classification Report:

	precision	recall	f1-score	support
0	0.93	0.79	0.85	189
1	0.49	0.76	0.60	51
accuracy			0.78	240
macro avg	0.71	0.78	0.73	240
weighted avg	0.83	0.78	0.80	240

Figure 14: Classification Report for Logistic Regression

Accuracy Score: 0.8
Classification Report:

	precision	recall	f1-score	support
0	0.93	0.81	0.86	189
1	0.52	0.76	0.62	51
accuracy			0.80	240
macro avg	0.72	0.79	0.74	240
weighted avg	0.84	0.80	0.81	240

Figure 15: Classification Report for SVC

Accuracy Score: 0.8
Classification Report:

	precision	recall	f1-score	support
0	0.89	0.85	0.87	189
1	0.53	0.61	0.56	51
accuracy			0.80	240
macro avg	0.71	0.73	0.72	240
weighted avg	0.81	0.80	0.81	240

Figure 16: Classification Report for LightGBM Model

Accuracy Score: 0.8375
Classification Report:

	precision	recall	f1-score	support
0	0.90	0.89	0.90	189
1	0.61	0.65	0.63	51
accuracy			0.84	240
macro avg	0.76	0.77	0.76	240
weighted avg	0.84	0.84	0.84	240

Figure 17: Classification Report for the Random Forest

```
Accuracy Score: 0.8708333333333333
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.95      0.92       189
           1       0.75      0.59      0.66        51

    accuracy                           0.87       240
   macro avg       0.82      0.77      0.79       240
weighted avg       0.86      0.87      0.86       240
```

Figure 18: Classification Report for the K-Neighbors Classifier

```
Accuracy Score: 0.8375
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.89      0.90       189
           1       0.61      0.65      0.63        51

    accuracy                           0.84       240
   macro avg       0.76      0.77      0.76       240
weighted avg       0.84      0.84      0.84       240
```

Figure 19: Classification Report for the XGBoost Classifier

Analyzing the Classification report provides valuable insights into the algorithm's performance.

By Considering the Above Classification Reports All the algorithms have shown moderately high accuracy ranging from 78% to 87%. This indicates their ability to accurately distinguish between individuals with autism and those without.

Considering class 0 precision in all the 6 model it's ranging from 89% to 93%, indicating their ability to correctly identify actual non-autism cases. The recall for class 0 for all six models ranged 79% to 95% highlighting the algorithm's ability to correctly identify non-autism cases out of the total number of non-autism cases.

For class 1 precision is varying 49% to 75% and recall value is varying 59% to 76%. It indicates the algorithm's ability to correctly identify actual autism cases and recall them from the total number of autism cases.

Considering F1 measures and accuracy, The F1 measure takes both precision and recall into account and provides a single measure to evaluate the algorithm's overall performance. The F1 measure values are most algorithms were high, ranging from for class 0, 85% to 92%. It indicates a good balance of precision and recall.

Here relatively for class 0 Precession, Recall, and F1 Score, all three high values are given by the K-Neighbors Classifier Model compared to the other models. These values are respectively 0.9, 0.95, and 0.92. These values are above 90%. It indicates their ability to correctly identify actual non-autism cases by K-Neighbors Classifier Method. Otherwise, the highest precision value for class 1 is also given by the K-Neighbors algorithm method which is 0.75. But comparing all the cases class 1 detected these measurement values are lower than class 0 detection values. Considering the macro average all classes equally contribute and give the final averaged values for precision, recall, and f1 score. The macro average for precision all the models vary from 71% to 82%. It's indicating all the models give accurate values for autism or not cases. But here also highest macro average precision value is given by the K-Neighbors algorithm.

When choosing the best model Based on the results, the best algorithms for autism detection can be discussed. Therefore, Considering the above result we can determine K-Neighbors algorithm is the most suitable algorithm the autism and non-autism cases compared to the other machine learning models.

CONFUSION MATRIXES FOR COMPARISON TO THE MODELS

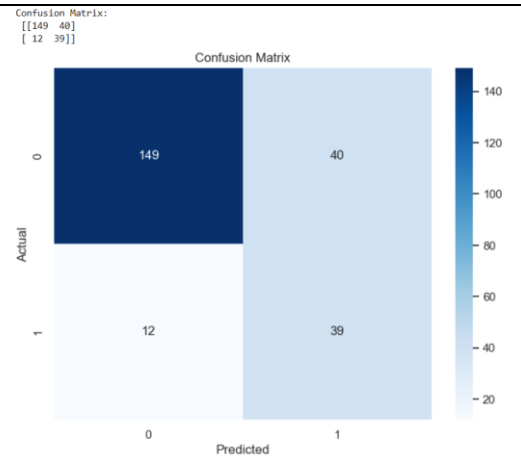


Figure 20: Confusion Matrix for Logistic Regression

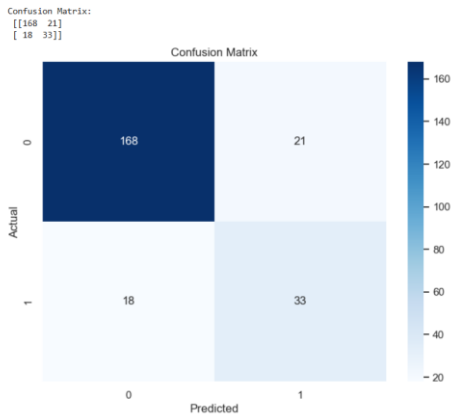


Figure 21: Confusion Matrix for Random Forest Model

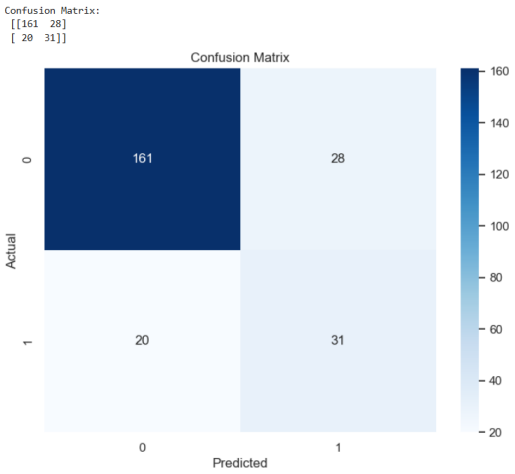


Figure 22: Confusion Matrix for LightGBM Model

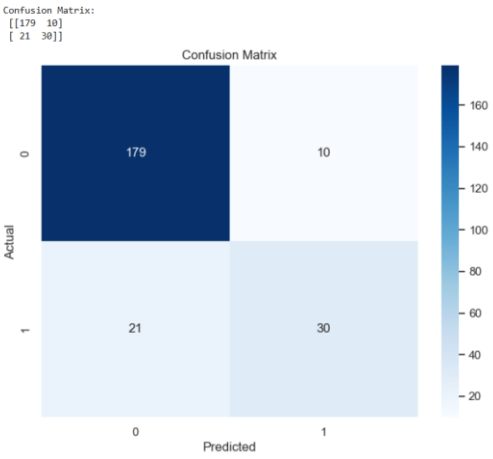


Figure 23: Confusion Matrix for K-Neighbors Classifier

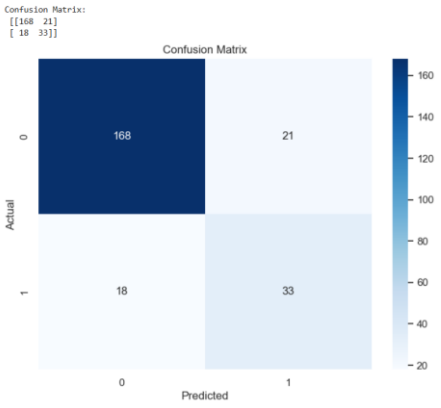


Figure 24:Confusion Matrix for XGBoost

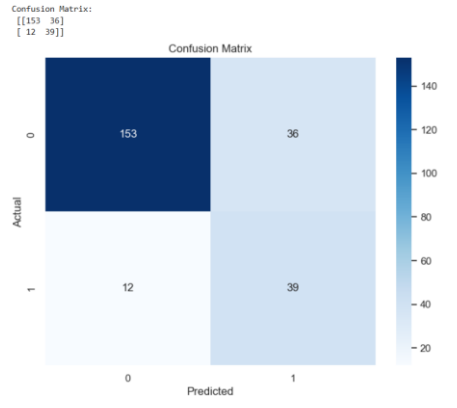


Figure 25 : Confusion Matrix for SVC

Table 4 : Confusion Matrix Data

ML Models	True Positive (TP)	False Positive (FP)	True Negative (TN)	False Negative (FN)
Logistic Regression	39	40	149	12
Random Forest	33	21	168	18
LightGBM	31	28	161	20
K-Neighbors Classifier	30	10	179	21
XGBoost	33	21	168	18
SVC	39	36	153	12

Based on the provided confusion matrices, it appears that several machine learning models were used to detect autism spectrum disorder (ASD) in a case study and Table 4 describes Elements of a Confusion Matrix. Here is a summary of the results in Table 4.

In general, all the models achieved reasonably high true positive rates, indicating that they were able to correctly identify individuals with ASD. The true negative rates were also consistently high, suggesting that the models were effective in identifying individuals without ASD.

Among the models, Decision Tree, and Random Forest, performed particularly well, achieving a perfect true positive rate and true negative rate. K-Neighbors had a slightly lower true positive rate but still performed well overall. However, the K-Neighbors algorithm gives the highest True Negative rate compared to all the other ML models.

Here indicate True Negative values in the all the six models are high compare to the False Negative values. It's indicated the dataset Autism Not having individuals is most accurately determined by all the six ML models. In here Lowest False Positive rate, Highest True Negative rate and the Highest False Negative rate are determined by the K-Neighbors Machine Learning Model. Therefore, it can indicate that the K-Neighbors model is the best model for determining autism and non-autism cases.

MATRIX COMPARISSON AND ROC CURVE FOR COMPARISON TO THE MODELS

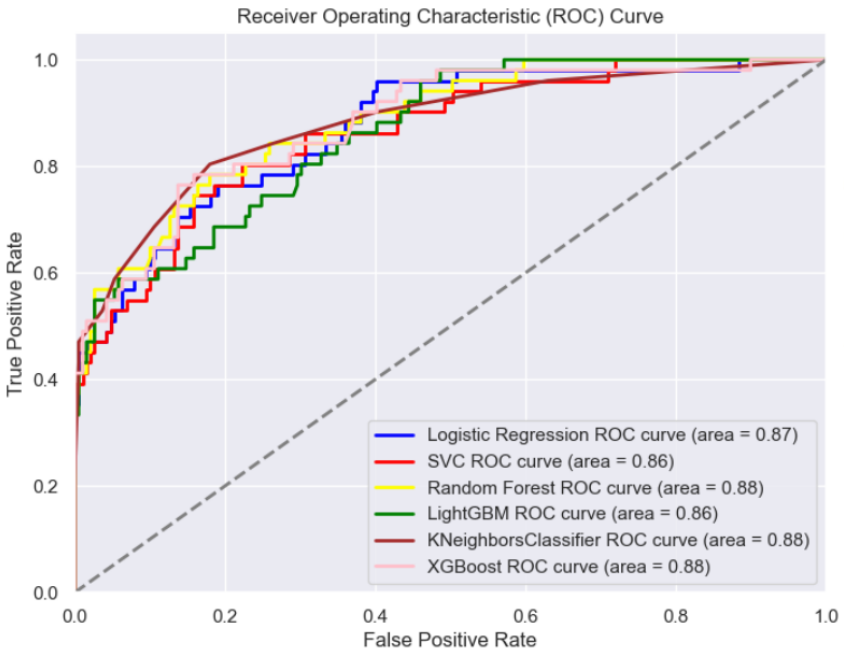


Figure 26 : All ML Models ROC Curves

Metrics Comparison:

	Model	Accuracy	Precision	R2 Score	ROC AUC
0	Logistic Regression	0.783333	0.493671	-0.294740	0.873120
1	SVC	0.800000	0.520000	-0.195145	0.863783
2	Random Forest	0.837500	0.611111	0.028945	0.884013
3	LightGBM	0.800000	0.525424	-0.195145	0.863316
4	KNeighborsClassifier	0.870833	0.750000	0.228136	0.880122
5	XGBoost	0.837500	0.611111	0.028945	0.880693

Figure 27: All ML Models Results

To determine the best model from the provided ROC curves and metric comparisons, here evaluate the models based on several criteria: accuracy, precision, R2 score, and ROC AUC. Finding best model for predicting data gives high accuracy results for the predicting results for ASD. These include logistic regression, Support Vector Classifier, Random Forest, LightGBM, K-Neighbors Classifier, and XGBoost ML technique.

The ROC curve and the summarization result table give a comparative landscape for these models.

Considering the accuracy of the model K-Neighbors Model gives high accuracy (0.870833). Therefore K-Neighbors machine learning model correctly predicts the outcome individual have autism or not. The highest Precision value (0.75) is also given by the K-Neighbors model. This model also gives the highest r2_score (0.228136) it's indicated that the model is a good fit for the data compared to the other models.

The Random Forest method gives the highest ROC Area Under the Curve value (0.884013) but this value is also approximately very close to the XGBoost (0.880693) and K-Neighbors (0.880122) ROC AUC values. It means that the K-Neighbors model also can perfectly distinguish between all the Positive and the Negative class points. Considering all this determine by K-Neighbors Model is the best model for predicting whether an individual has ASD or not.

7.0 GENERAL DISCUSSION AND CONCLUSION

Maximum significant progress for autism spectrum disorder is possible with proper treatment and the treatment is most effective with an early-stage diagnosis. However, a diagnosis of autism is complicated because sometimes it doesn't show in medical tests, such as blood work or brain scans, sort of a biopsy, to detect the disorder. Therefore, Here Our study focused on predicting autism spectrum disorder through machine learning algorithms, specifically exploring models such as include logistic regression, Support Vector Classifier, Random Forest, LightGBM, K-Neighbors Classifier, and XGBoost ML technique to improve early diagnosis of individuals who have ASD or not.

Using the Above advanced analysis part, Classification Report, the K-Neighbors algorithm gives high precision, recall, and f1 scores for both classes.

- After a thorough assessment based on the above K-Neighbors algorithm emerged as the most suitable model for our dataset, boasting the highest accuracy at 87.08%.
- The Confusion Matrix shows the Highest true positive and true negative values are given by the K-Neighbors algorithm.
- The ROC curve also can determine the highest approximate area under the curve given by this K-Neighbors algorithm.
- The Matrix Comparison clearly shows the highest accuracy, precision, and r2 scores given by the K-Neighbors Algorithm.

By considering all the methods it's proven that the best ML model for predicting whether individuals have autism or not is the K-Neighbors algorithm. Therefore, we can easily detect whether these K-Neighbors machine learning model people have autism or not. The first answer to the A1-A10 small test question then fills in some information about the individual's data and it is tested by this here training the K-Neighbor ML model can easily and accurately determine whether individuals have autism or not. Using this model can accurately and early detection of autism and improve autistic people's lives. Also, it reduces diagnosis costs. By dealing with autistic people, it is possible to minimize their symptoms and improve their overall development by helping them adopt new skills and abilities that will allow them to be more self-reliant throughout the journey of their life.

In future work, we Look forward to improving the accuracy of our ASD prediction models. Firstly, expanding our datasets to get more information will allow us to create a more accurate prediction model. Secondly, transitioning from machine learning to advanced Deep learning methodologies will empower us to handle larger and more intricate datasets, ultimately improving the reliability of our autism spectrum disorder predictions.

8.0 REFERENCES

[1] A.S.Shanthi and A.S.Shanthi. “Support Vector Machine for MRI Stroke Classification”, International Journal on Computer Science and Engineering (IJCSE), ISSN: 0975-3397 in April 2014.

[2] F. Soleimani, A. Khakshour, Z. Abasi, S. Khayat, S. Z. Ghaemi, andN. A. H. Golchin, “Review of autism screening tests,” InternationalJournal of Pediatrics, vol. 2, no. 4, pp. 319–329, 2014

[3] Soul, J. S. & Spence, S. J. Predicting autism spectrum disorder in very preterm infants. Paediatrics 146(4), e2020019448 (2020)

[4] Alenizi A.S. and Al-Karawi K.A. (2023c). Machine learning approach for diabetes prediction. In: International Congress on Information and Communication Technology, Springer.

[5] Cruz J.A. and Wishart D.S. (2006). Applications of machine learning in cancer prediction and prognosis. Cancer Inform., 2, 117693510600200030

[6] N. Akhtar and M. Feeney, “Predictive analytics using a machine learning model to recommend the most suitable intervention technology for autism-related deficits,” in 2020 31st Irish Signals and Systems Conference (ISSC), DOI: 10.1109/ISSC49989.2020.9180213

[7] Hamza Sharif & Rizwan Ahmed Khan (2022) A Novel Machine Learning Based Framework for Detection of Autism Spectrum Disorder (ASD), Applied Artificial Intelligence, 36:1, 2004655, DOI: 10.1080/08839514.2021.2004655

[8]Usta M.B., Karabekiroglu K., Sahin B., Aydin M., Bozkurt A., Karaosman T., Aral A., Cobanoglu C., Kurt A.D., Kesim N., *et al.* Use of machine learning methods in prediction of short-term outcome in autism spectrum disorders Psychiatry Clin. Psychopharmacology., 29 (3) (2019), pp. 320-325

[9] Heinsfeld A. S., Franco A. R., Craddock R. C., Buchweitz A., Meneguzz F., & Lafer B. (2018). Identification of autism spectrum disorder using deep learning and theABIDE dataset. NeuroImage: Clinical, 17, 16-23.

[10] Sherkatghanad Z., Akhondzadeh M., Salari S., ZomorodiMoghadam M., Abdar M., Acharya U.R., Khosrowabadi R., Salari V. Automated detection of autism spectrum disorder using a convolutional neural network Front. Neurosci., 13 (2020), p. 1325

9.0 APPENDIX

autism-prediction-model-using-ml

July 4, 2024

0.0.1 Import necessary libraries

[1]:
```
import pandas as pd
import numpy as np
```

0.0.2 Loading a dataset

[4]:
```
df = pd.read_csv('train.csv')
df.head()
```

[4]:

	ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	…	gender	ethnicity	jaundice	austim	contry_of_res	used_app_before	result	age_desc	relation	Class/ASD
0	1	1	0	1	1	1	1	0	1	1	…	f	White-European	no	no	United States	no	7.819715	18 and more	Self	0
1	2	0	0	0	0	0	0	0	0	0	…	f	South Asian	no	no	Australia	no	10.544296	18 and more	?	0
2	3	1	1	1	1	1	1	0	0	1	…	f	White-European	no	no	United Kingdom	no	13.167506	18 and more	Self	1
3	4	0	0	0	1	0	0	0	0	0	…	f	South Asian	no	no	New Zealand	no	1.530098	18 and more	?	0
4	5	0	0	0	0	1	0	0	0	1	…	m	Black	no	yes	Italy	no	7.949723	18 and more	Self	0

[5 rows x 22 columns]

[6]:
```
# Renaming columns
df.rename(columns={'Class/ASD': 'Class_ASD'}, inplace=True)
```

[8]:
```
df.rename(columns={'austim': 'autism'}, inplace=True)
```

[10]:
```
df.head()
```

[10]:

	ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	\
0	1	1	0	1	1	1	1	0	
1	2	0	0	0	0	0	0	0	
2	3	1	1	1	1	1	1	0	
3	4	0	0	0	1	0	0	0	
4	5	0	0	0	0	1	0	0	

	A8_Score	A9_Score	…	gender	ethnicity	jaundice	autism	\
0	1	1	…	f	White-European	no	no	
1	0	0	…	f	South Asian	no	no	
2	0	1	…	f	White-European	no	no	
3	0	0	…	f	South Asian	no	no	
4	0	1	…	m	Black	no	yes	

	contry_of_res	used_app_before	result	age_desc	relation	Class_ASD
0	United States	no	7.819715	18 and more	Self	0
1	Australia	no	10.544296	18 and more	?	0
2	United Kingdom	no	13.167506	18 and more	Self	1
3	New Zealand	no	1.530098	18 and more	?	0
4	Italy	no	7.949723	18 and more	Self	0

[5 rows x 22 columns]

[12]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 22 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ID               800 non-null    int64
 1   A1_Score         800 non-null    int64
 2   A2_Score         800 non-null    int64
 3   A3_Score         800 non-null    int64
 4   A4_Score         800 non-null    int64
 5   A5_Score         800 non-null    int64
 6   A6_Score         800 non-null    int64
 7   A7_Score         800 non-null    int64
 8   A8_Score         800 non-null    int64
 9   A9_Score         800 non-null    int64
 10  A10_Score        800 non-null    int64
 11  age              800 non-null    float64
 12  gender           800 non-null    object
 13  ethnicity        800 non-null    object
 14  jaundice         800 non-null    object
```

```
 15  autism           800 non-null    object
 16  contry_of_res    800 non-null    object
 17  used_app_before  800 non-null    object
 18  result           800 non-null    float64
 19  age_desc         800 non-null    object
 20  relation         800 non-null    object
 21  Class_ASD        800 non-null    int64
dtypes: float64(2), int64(12), object(8)
memory usage: 137.6+ KB
None
```

0.0.3 Handling Missing Values

[15]:
```
df[df == '?'] = np.nan
df.isna().any()
```

[15]:
```
ID                 False
A1_Score           False
A2_Score           False
A3_Score           False
A4_Score           False
A5_Score           False
A6_Score           False
A7_Score           False
A8_Score           False
A9_Score           False
A10_Score          False
age                False
gender             False
ethnicity           True
jaundice           False
autism             False
contry_of_res      False
used_app_before    False
result             False
age_desc           False
relation            True
Class_ASD          False
dtype: bool
```

[17]:
```
df.isnull().sum()
```

[17]:
```
ID                 0
A1_Score           0
A2_Score           0
A3_Score           0
A4_Score           0
```

```
A5_Score              0
A6_Score              0
A7_Score              0
A8_Score              0
A9_Score              0
A10_Score             0
age                   0
gender                0
ethnicity           151
jaundice              0
autism                0
contry_of_res         0
used_app_before       0
result                0
age_desc              0
relation             77
Class_ASD             0
dtype: int64
```

[19]:
```
df['ethnicity'] = df['ethnicity'].fillna(df['ethnicity'].mode()[0])
df['relation'] = df['relation'].fillna(df['relation'].mode()[0])
```

[21]:
```
df.isnull().sum()
```

[21]:
```
ID                  0
A1_Score            0
A2_Score            0
A3_Score            0
A4_Score            0
A5_Score            0
A6_Score            0
A7_Score            0
A8_Score            0
A9_Score            0
A10_Score           0
age                 0
gender              0
ethnicity           0
jaundice            0
autism              0
contry_of_res       0
used_app_before     0
result              0
age_desc            0
relation            0
Class_ASD           0
dtype: int64
```

[23]:
```
df.duplicated().sum()
```

[23]:
```
0
```

0.0.4 Check Unique Values of Categorical Variables

[26]:
```
print(df.ethnicity.unique())
```

```
['White-European' 'South Asian' 'Black' 'Asian' 'Middle Eastern ' 'others'
 'Latino' 'Turkish' 'Others' 'Hispanic' 'Pasifika']
```

[28]:
```
print(df.gender.unique())
```

```
['f' 'm']
```

[30]:
```
print(df.jaundice.unique())
```

```
['no' 'yes']
```

[32]:
```
print(df.autism.unique())
```

```
['no' 'yes']
```

[34]:
```
print(df.contry_of_res.unique())
```

```
['United States' 'Australia' 'United Kingdom' 'New Zealand' 'Italy'
 'Nicaragua' 'Canada' 'United Arab Emirates' 'Netherlands' 'Sri Lanka'
 'India' 'Armenia' 'Sierra Leone' 'Argentina' 'Azerbaijan' 'Iceland'
 'Egypt' 'Serbia' 'Afghanistan' 'Costa Rica' 'Jordan' 'Angola' 'Pakistan'
 'Brazil' 'Ireland' 'Kazakhstan' 'Viet Nam' 'Ethiopia' 'Austria' 'Finland'
 'France' 'Malaysia' 'Japan' 'Spain' 'Philippines' 'Iran' 'Czech Republic'
 'Russia' 'Romania' 'Mexico' 'Belgium' 'Aruba' 'Uruguay' 'Indonesia'
 'Ukraine' 'AmericanSamoa' 'Germany' 'China' 'Iraq' 'Tonga' 'South Africa'
 'Saudi Arabia' 'Hong Kong' 'Bahamas' 'Ecuador' 'Cyprus' 'Bangladesh'
 'Oman' 'Bolivia' 'Sweden' 'Niger']
```

[36]:
```
print(df.used_app_before.unique())
```

```
['no' 'yes']
```

[38]:
```
print(df.age_desc.unique())
```

```
['18 and more']
```

[40]:
```
print(df.relation.unique())
```

```
['Self' 'Health care professional' 'Parent' 'Relative' 'Others']
```

[42]:
```
print(df.Class_ASD.unique())
```

```
[0 1]
```

0.0.5 Rename the some mistake unique values

[45]:
```
df['ethnicity']=df['ethnicity'].replace('others', 'Others')
```

[47]:
```
print(df.ethnicity.unique())
```

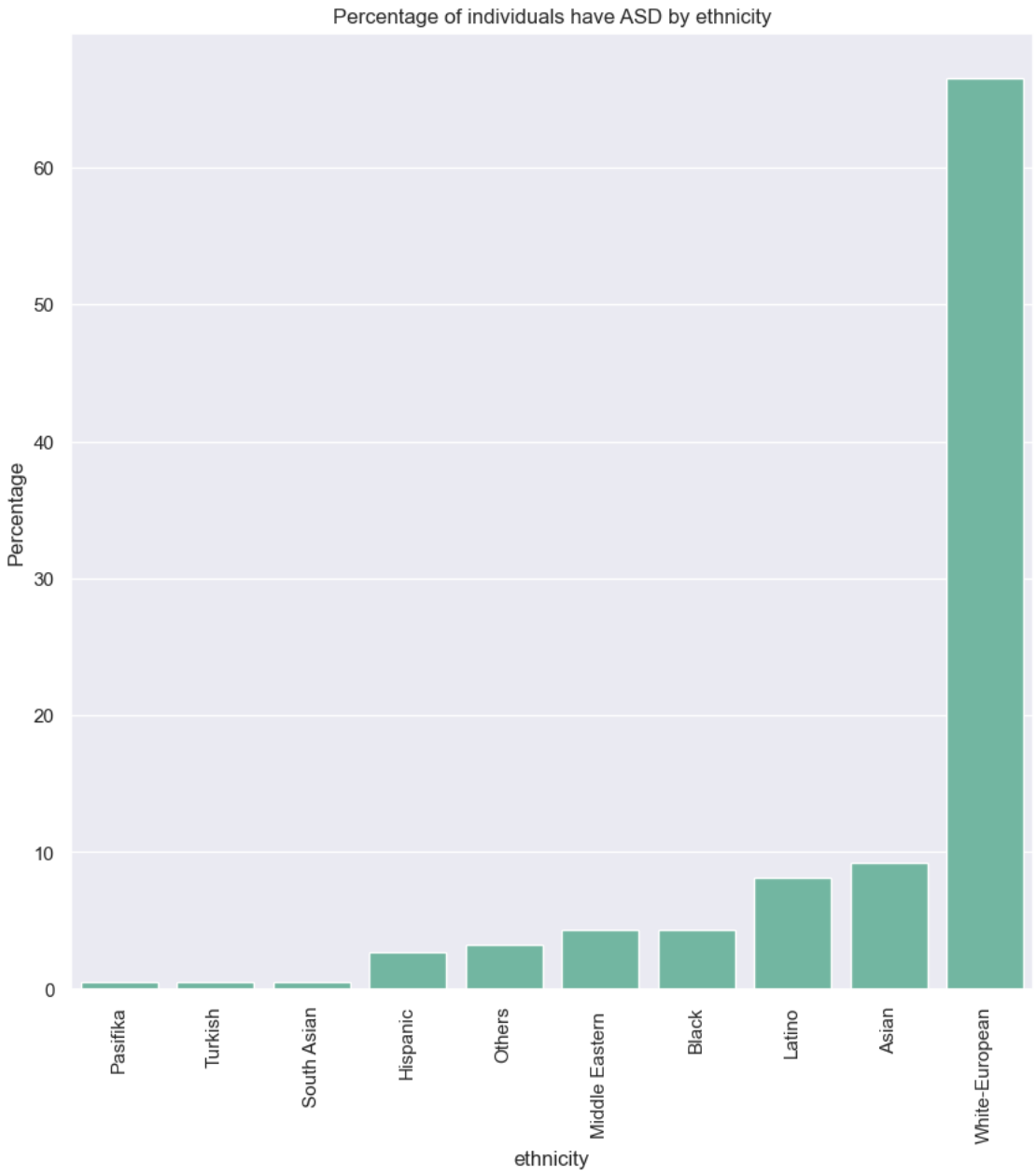
```
['White-European' 'South Asian' 'Black' 'Asian' 'Middle Eastern ' 'Others'
 'Latino' 'Turkish' 'Hispanic' 'Pasifika']
```

0.0.6 Data Vizualization

[50]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

Percentage of individuals with ASD by ethnicity

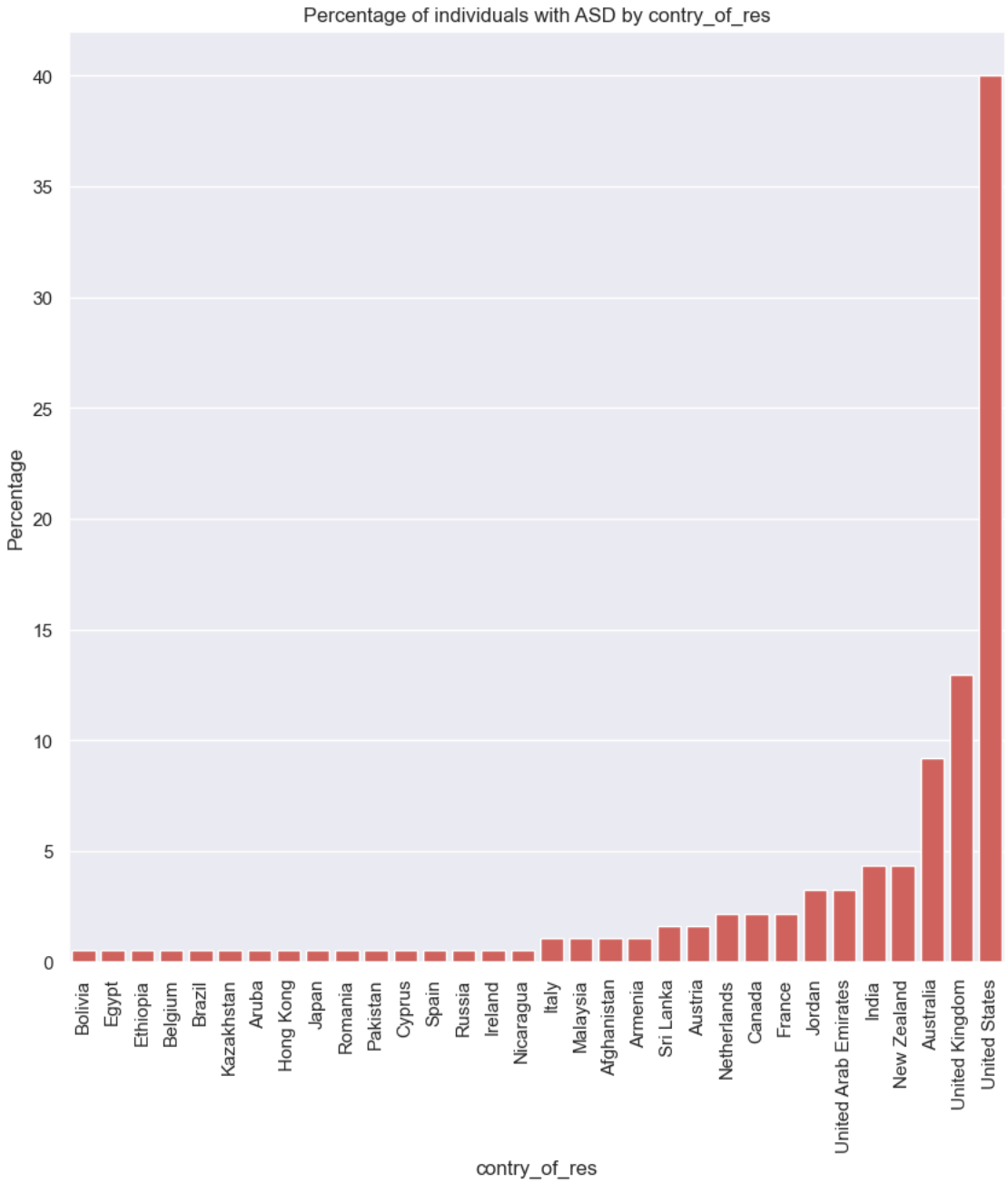
[52]:
```
df_asd = df[df['Class_ASD'] == 1]
total_samples = len(df_asd)
percentage_df = df_asd['ethnicity'].value_counts(normalize=True).reset_index()
percentage_df.columns = ['ethnicity', 'percentage']
percentage_df['percentage'] = percentage_df['percentage'] * 100
percentage_df = percentage_df.sort_values('percentage')

sns.set_theme(style="darkgrid",palette="Set2")

plt.figure(figsize=(10, 10))
sns.barplot(x='ethnicity', y='percentage', data=percentage_df,␣
 ↪order=percentage_df['ethnicity'])
plt.ylabel('Percentage')
plt.title('Percentage of individuals have ASD by ethnicity')
plt.xticks(rotation=90)
plt.show()
```



Percentage of individuals with ASD by contry_of_res

[56]:
```
df_asd = df[df['Class_ASD'] == 1]
total_samples = len(df_asd)
percentage_df = df_asd['contry_of_res'].value_counts(normalize=True).
 ↪reset_index()
percentage_df.columns = ['contry_of_res', 'percentage']
percentage_df['percentage'] = percentage_df['percentage'] * 100
percentage_df = percentage_df.sort_values('percentage')
```

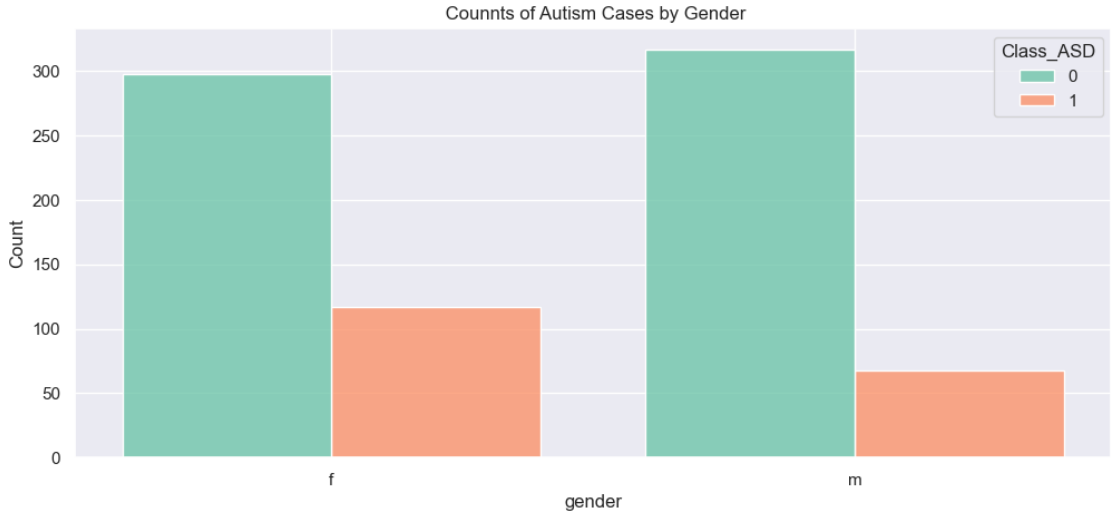
```
sns.set_theme(style="darkgrid", palette="Spectral")

plt.figure(figsize=(10, 10))
sns.barplot(x='contry_of_res', y='percentage', data=percentage_df,␣
 ↪order=percentage_df['contry_of_res'])
plt.ylabel('Percentage')
plt.title('Percentage of individuals with ASD by contry_of_res')
plt.xticks(rotation=90);
plt.show()
```



0.1 Count of the Gender by Class_AS

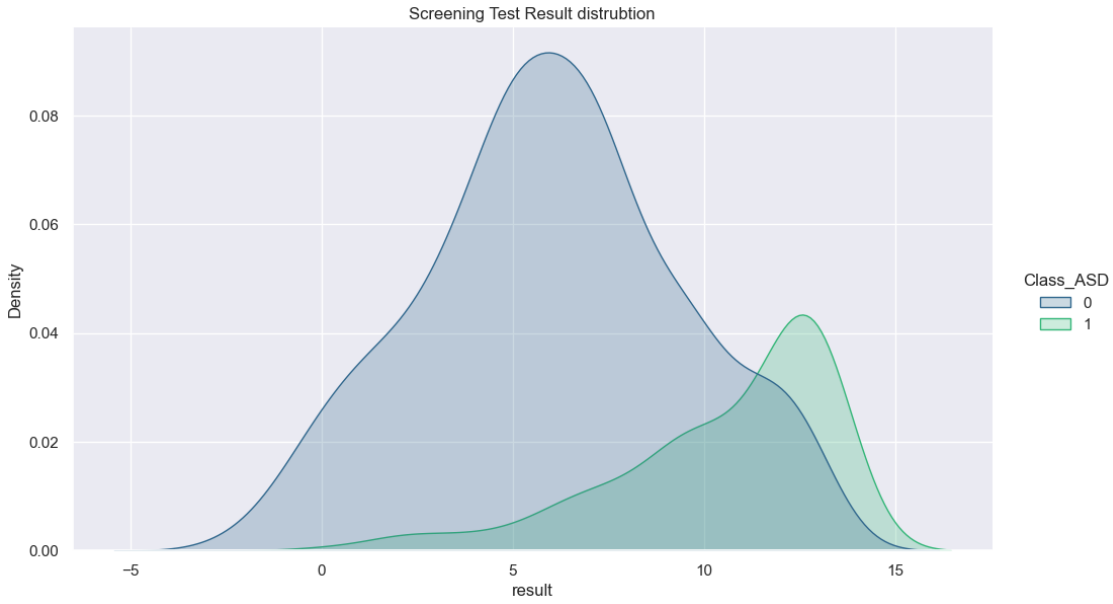
[59]:
```
plt.figure(figsize=(12,5))
sns.histplot(data=df, x="gender", hue="Class_ASD", multiple="dodge", palette␣
↪='Set2',shrink=.8)
plt.title(' Counnts of Autism Cases by Gender')
```

[59]:
```
Text(0.5, 1.0, ' Counnts of Autism Cases by Gender')
```



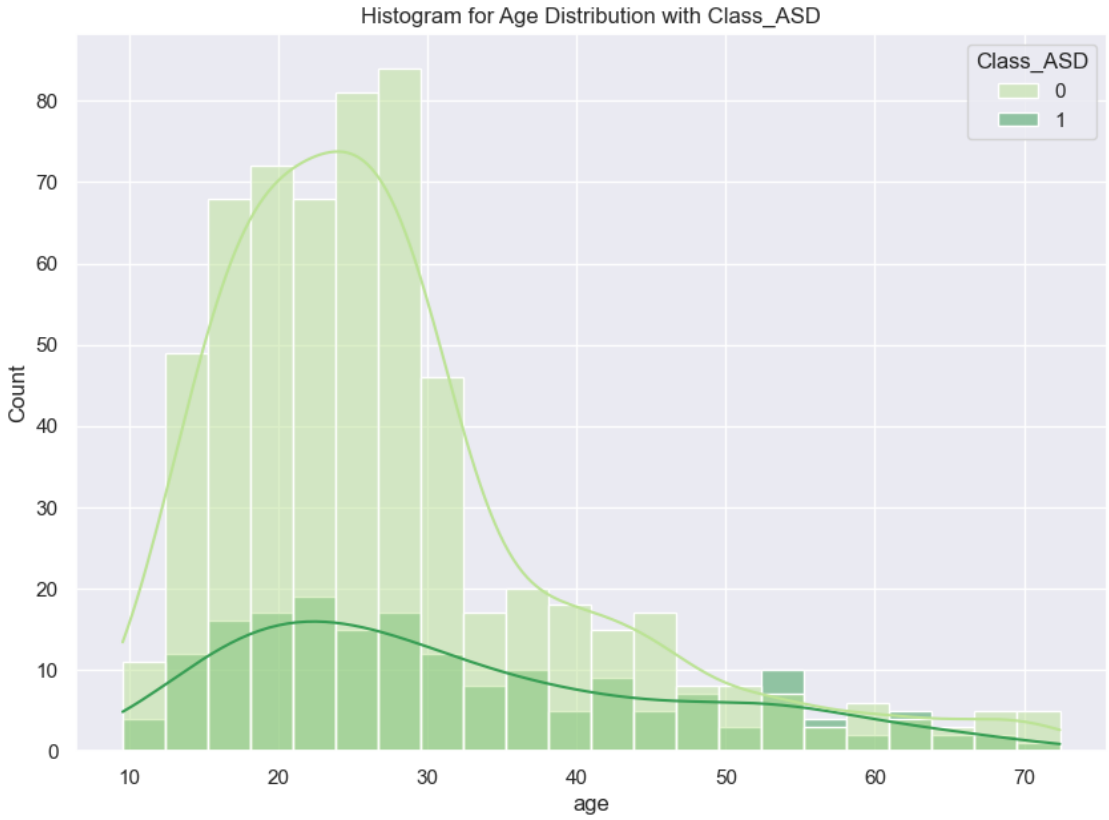
0.2 Density Plot for Class_ASD

[62]:
```
sns.displot(data=df,x='result',hue="Class_ASD" ,kind="kde", palette ="viridis"␣
↪,fill=True,height=6, aspect=1.7) ;
plt.title("Screening Test Result distrubtion")
plt.show()
```



0.3 Histograms For Age Distribution with Class_ASD

[65]:
```
plt.figure(figsize=(10, 7))
age_plot = df.groupby(['age', 'Class_ASD']).size().reset_index(name="size")
sns.histplot(data=age_plot, x='age', hue="Class_ASD", kde=True, palette='YlGn')
plt.title("Histogram for Age Distribution with Class_ASD")
plt.show()
```



Define the correlation between numerical variables and the Class_ASD

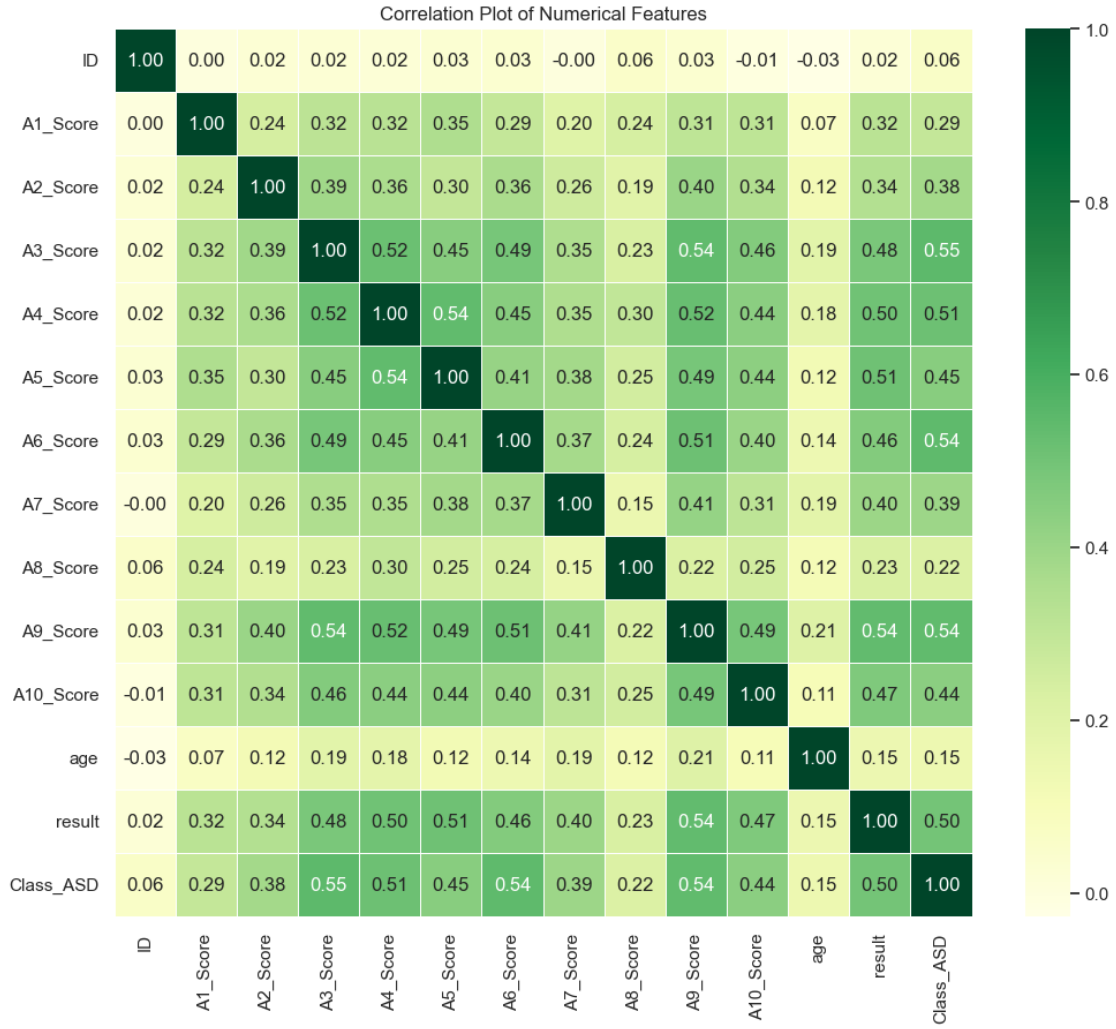
[68]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 22 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ID               800 non-null    int64
 1   A1_Score         800 non-null    int64
 2   A2_Score         800 non-null    int64
 3   A3_Score         800 non-null    int64
 4   A4_Score         800 non-null    int64
 5   A5_Score         800 non-null    int64
 6   A6_Score         800 non-null    int64
 7   A7_Score         800 non-null    int64
 8   A8_Score         800 non-null    int64
 9   A9_Score         800 non-null    int64
 10  A10_Score        800 non-null    int64
 11  age              800 non-null    float64
 12  gender           800 non-null    object
```

```
 13  ethnicity         800 non-null    object
 14  jaundice          800 non-null    object
 15  autism            800 non-null    object
 16  contry_of_res     800 non-null    object
 17  used_app_before   800 non-null    object
 18  result            800 non-null    float64
 19  age_desc          800 non-null    object
 20  relation          800 non-null    object
 21  Class_ASD         800 non-null    int64
dtypes: float64(2), int64(12), object(8)
memory usage: 137.6+ KB
```

[70]:
```
# pip install seaborn --upgrade
```

[72]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

numerical_columns = df.select_dtypes(include=['int64', 'float64'])
# Create a correlation matrix
correlation_matrix = numerical_columns.corr()

# Set up the matplotlib figure
plt.figure(figsize=(12, 10))

# Create a heatmap using seaborn to visualize the correlation matrix
sns.heatmap(correlation_matrix, annot=True, cmap="YlGn", fmt=".2f", linewidths=.
 ↪5)

# Show the plot
plt.title("Correlation Plot of Numerical Features")
plt.show()
```



Here: A1_Score, A8_Score, age variables have very less correlation between Classs_ASD (<0.4). Then there are not significant affect to the Class_ASD

Define the correlation between categorical variables and the Class_ASD

[76]:
```
import pandas as pd
import scipy.stats as stats

# Selecting categorical columns
categorical_columns = df.select_dtypes(include=['object', 'category']).columns

# Results dictionary to store p-values of Chi-Square tests
chi_square_results = {}

for column in categorical_columns:
```

```
    if column != 'Class_ASD':
        contingency_table = pd.crosstab(df[column], df['Class_ASD'])
        chi2, p, dof, ex = stats.chi2_contingency(contingency_table)
        chi_square_results[column] = p

# Convert results to a DataFrame for better readability
chi_square_results_df = pd.DataFrame(list(chi_square_results.items()),␣
 ↪columns=['Feature', 'p-value'])

print("Chi-Square Test Results:")
print(chi_square_results_df)
```

```
Chi-Square Test Results:
           Feature       p-value
0           gender  5.696647e-04
1        ethnicity  1.510274e-10
2         jaundice  1.169010e-07
3           autism  5.258030e-26
4    contry_of_res  1.346083e-09
5  used_app_before  8.076826e-01
6         age_desc  1.000000e+00
7         relation  8.661287e-02
```

Here: used_app_before, age_desc, relation variables are not significant

0.3.1 Identify Shape of the Dataset

[80]:
```
# Get the number of rows and columns
num_rows, num_columns = df.shape
print(f"Number of rows: {num_rows}")
print(f"Number of columns: {num_columns}")
```

```
Number of rows: 800
Number of columns: 22
```

[82]:
```
df['Class_ASD'].value_counts()
```

[82]:
```
Class_ASD
0    615
1    185
Name: count, dtype: int64
```

0.3.2 Split a Dataset

[85]:
```
# Split into training and testing
from sklearn.model_selection import train_test_split
X=df.drop(columns=['ID','Class_ASD', 'A1_Score', 'A8_Score', 'age',␣
 ↪'used_app_before', 'age_desc', 'relation' ])
y=df['Class_ASD']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
 ↪3,random_state=123)
```

[86]:
```
# Check the splits are correct
print(f"Train size: {round(len(X_train) / len(X) * 100)}% \n\
Test size: {round(len(X_test) / len(X) * 100)}%")
```

```
Train size: 70%
Test size: 30%
```

[89]:
```
print("Counts of label in y_train '1': {}".format(sum(y_train == 1)))
print("Counts of label in y_train '0': {} \n".format(sum(y_train == 0)))
```

```
Counts of label in y_train '1': 134
Counts of label in y_train '0': 426
```

0.3.3 Create a Pipeline for Preproceser, SMOTE and Classifier

[92]:
```
from imblearn.over_sampling import SMOTE
```

[93]:
```
# ! pip install imblearn
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from imblearn.pipeline import Pipeline
```

[94]:
```
#transforming
numeric_transformer = Pipeline(steps=[('scaler', StandardScaler())])
categorical_transformer = Pipeline(steps=[('encoder',␣
 ↪OneHotEncoder(handle_unknown='ignore'))])
```

[98]:
```
numeric_features=[ 'result']
categorical_features = ['gender', 'ethnicity', 'jaundice', 'autism',␣
 ↪'contry_of_res']
preprocessor = ColumnTransformer( transformers=[('numeric',␣
 ↪numeric_transformer, numeric_features) , ('categorical',␣
 ↪categorical_transformer, categorical_features)])
```

Logistic Regression

[101]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

[103]:
```
pipeline=Pipeline(steps=[('preprocessor',preprocessor),('smote',SMOTE(random_state=42)),('clas
 ↪LogisticRegression())])

param_grid={
        'classifier__C': [0.1, 1, 10, 100],
        'classifier__penalty': ['l1', 'l2'],
        'classifier__solver': ['liblinear']
    }
```

[105]:
```
from sklearn.model_selection import StratifiedKFold
```

[107]:
```
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
```

[109]:
```
clf=GridSearchCV(pipeline,param_grid,cv=SKF,n_jobs=5,verbose=1)
```

[111]:
```
model=clf.fit(X_train,y_train)
print(model)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=0, shuffle=True),
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('numeric',
Pipeline(steps=[('scaler',
          StandardScaler())]),
['result']),
('categorical',
Pipeline(steps=[('encoder',
          OneHotEncoder(handle_unknown='ignore'))]),
['gender',
'ethnicity',
'jaundice',
'autism',
'contry_of_res'])])),
                                       ('smote', SMOTE(random_state=42)),
                                       ('classifier', LogisticRegression())]),
             n_jobs=5,
             param_grid={'classifier__C': [0.1, 1, 10, 100],
                         'classifier__penalty': ['l1', 'l2'],
                         'classifier__solver': ['liblinear']},
             verbose=1)
```

[113]:
```
print(len(X_train))
```

```
560
```

[115]:
```
print(len(y_train))
```

```
560
```

[117]:
```
print(clf.best_params_)
```

```
{'classifier__C': 0.1, 'classifier__penalty': 'l2', 'classifier__solver':
'liblinear'}
```

[119]:
```
best_model = clf.best_estimator_
best_model
```

[119]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('numeric',
                                                  Pipeline(steps=[('scaler',
StandardScaler())]),
                                                  ['result']),
                                                 ('categorical',
                                                  Pipeline(steps=[('encoder',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['gender', 'ethnicity',
                                                   'jaundice', 'autism',
                                                   'contry_of_res'])])),
                ('smote', SMOTE(random_state=42)),
                ('classifier', LogisticRegression(C=0.1, solver='liblinear'))])
```

[121]:
```
from sklearn.metrics import accuracy_score
preds_lr = best_model.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, preds_lr))

from sklearn.metrics import classification_report
print("Classification Report:\n", classification_report(y_test, preds_lr))
```

```
Accuracy Score: 0.7833333333333333
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.79      0.85       189
           1       0.49      0.76      0.60        51

    accuracy                           0.78       240
   macro avg       0.71      0.78      0.73       240
weighted avg       0.83      0.78      0.80       240
```

[123]:
```
from sklearn.metrics import r2_score
```

[125]:
```
print (r2_score(y_test, preds_lr))
```

```
-0.2947401182695304
```

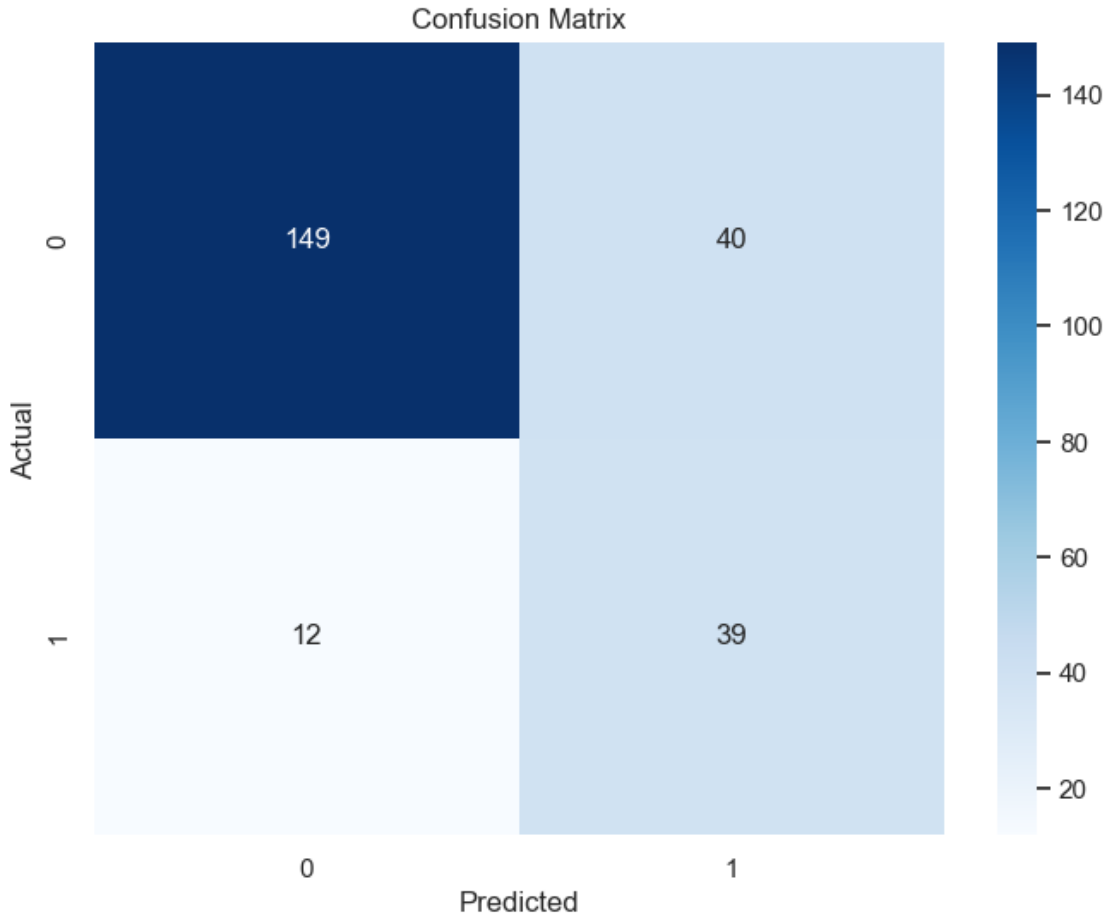
[127]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix, roc_curve, auc, precision_score
```

[129]:
```
# Print precision score
precision = precision_score(y_test, preds_lr)
print("Precision Score:", precision)
```

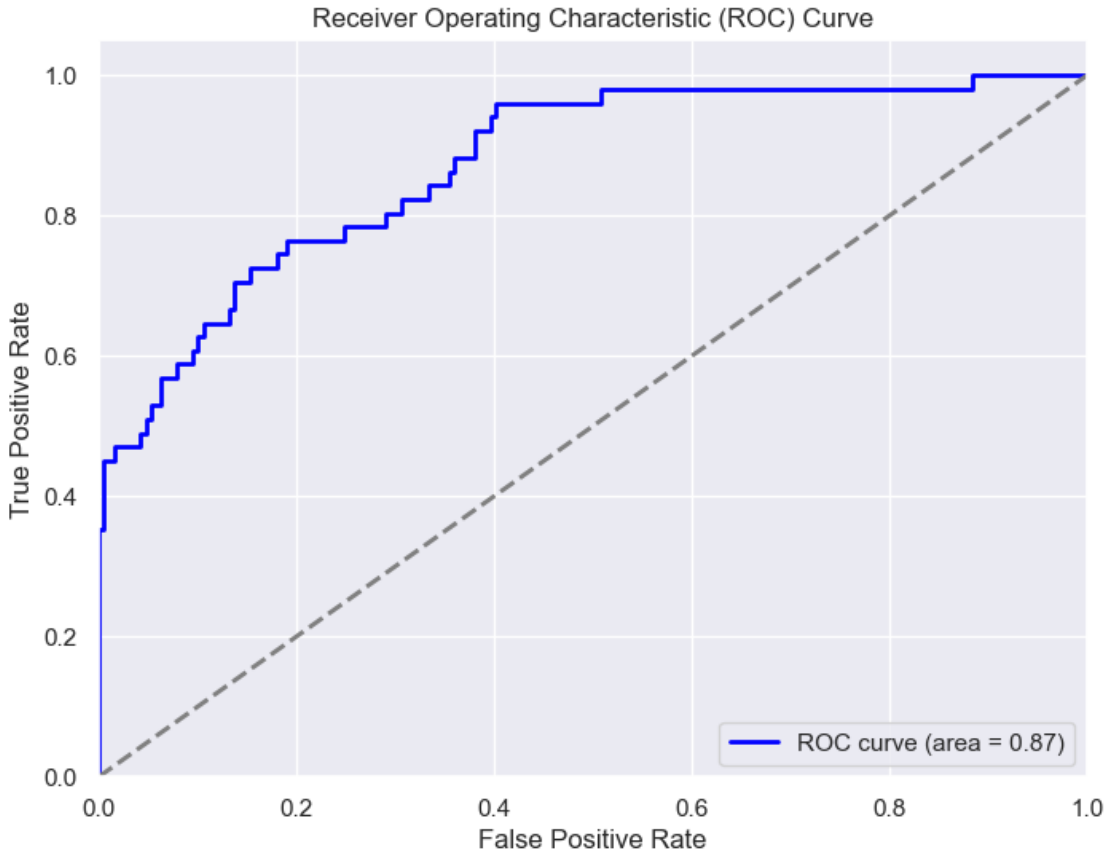
```
Precision Score: 0.4936708860759494
```

[131]:
```
# Print confusion matrix
conf_matrix = confusion_matrix(y_test, preds_lr)
print("Confusion Matrix:\n", conf_matrix)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
 [[149  40]
 [ 12  39]]
```



[133]:
```
# Compute ROC curve and ROC area
y_proba_lr = best_model.predict_proba(X_test)[:, 1]
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_proba_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, color='blue', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc_lr)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 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()
```



Support Vector Classifier

[136]:
```
from sklearn.svm import SVC
```

[138]:
```
pipeline=Pipeline(steps=[('preprocessor',preprocessor),('smote',SMOTE(random_state=42)),('clas
 ↪SVC(probability=True))])
param_grid={
        'classifier__C': [0.1, 1, 10],
        'classifier__kernel': ['linear', 'rbf'],
        'classifier__gamma': ['scale','auto']
    }
```

[140]:
```
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
```

[142]:
```
clf=GridSearchCV(pipeline,param_grid,cv=SKF,n_jobs=5,verbose=1)
```

[144]:
```
model=clf.fit(X_train,y_train)
print(model)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=0, shuffle=True),
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('numeric',
Pipeline(steps=[('scaler',
          StandardScaler())]),
['result']),
('categorical',
Pipeline(steps=[('encoder',
          OneHotEncoder(handle_unknown='ignore'))]),
['gender',
'ethnicity',
'jaundice',
'autism',
'contry_of_res'])])),
                                       ('smote', SMOTE(random_state=42)),
                                       ('classifier', SVC(probability=True))]),
             n_jobs=5,
             param_grid={'classifier__C': [0.1, 1, 10],
                         'classifier__gamma': ['scale', 'auto'],
                         'classifier__kernel': ['linear', 'rbf']},
             verbose=1)
```

[146]:
```
print(clf.best_params_)
```

```
{'classifier__C': 1, 'classifier__gamma': 'scale', 'classifier__kernel': 'rbf'}
```

[148]:
```
best_model = clf.best_estimator_
best_model
```

[148]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('numeric',
                                                  Pipeline(steps=[('scaler',
StandardScaler())]),
                                                  ['result']),
                                                 ('categorical',
                                                  Pipeline(steps=[('encoder',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['gender', 'ethnicity',
                                                   'jaundice', 'autism',
                                                   'contry_of_res'])])),
                ('smote', SMOTE(random_state=42)),
                ('classifier', SVC(C=1, probability=True))])
```

[150]:
```
preds_svc = best_model.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, preds_svc))

print("Classification Report:\n", classification_report(y_test, preds_svc))
```

```
Accuracy Score: 0.8
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.81      0.86       189
           1       0.52      0.76      0.62        51

    accuracy                           0.80       240
   macro avg       0.72      0.79      0.74       240
weighted avg       0.84      0.80      0.81       240
```

[152]:
```
from sklearn.metrics import r2_score
```

[154]:
```
print (r2_score(y_test, preds_svc))
```

```
-0.19514472455648968
```

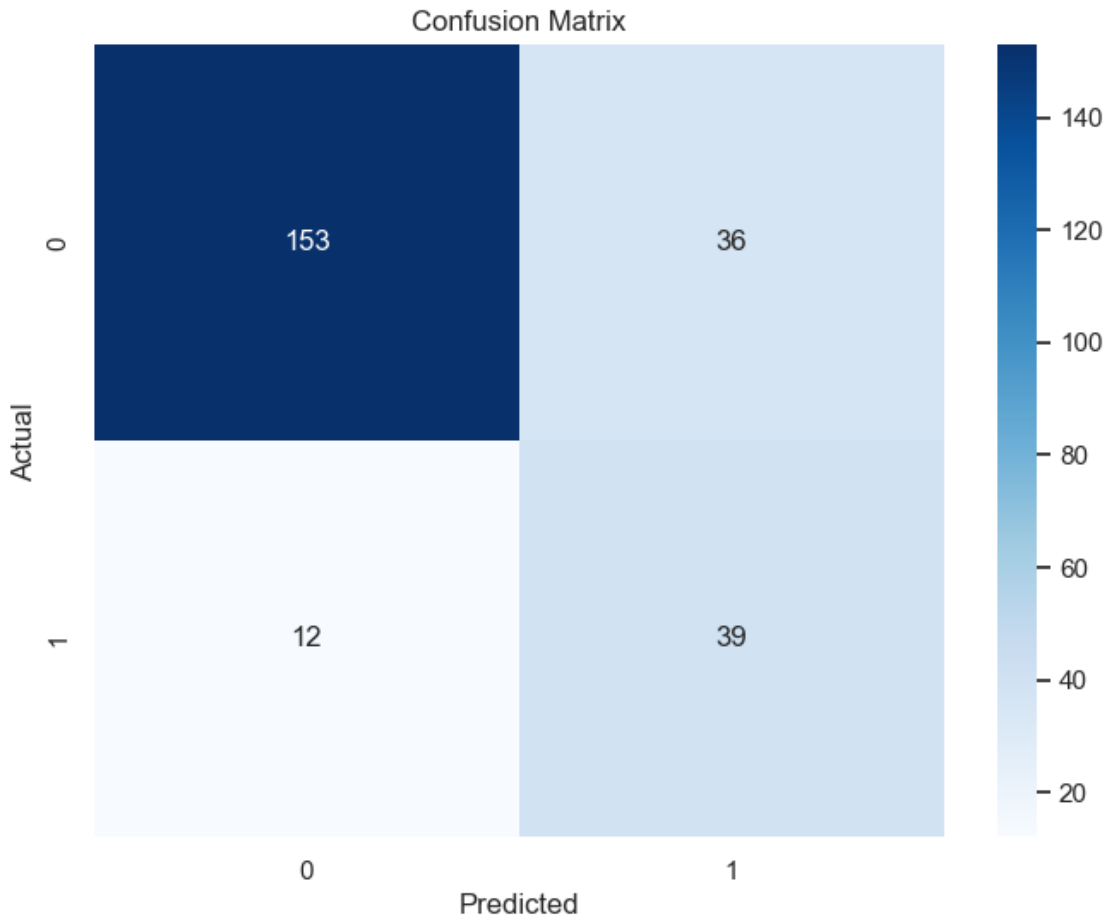
[156]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix, roc_curve, auc, precision_score
```

[158]:
```
# Print precision score
precision = precision_score(y_test, preds_svc)
print("Precision Score:", precision)
```

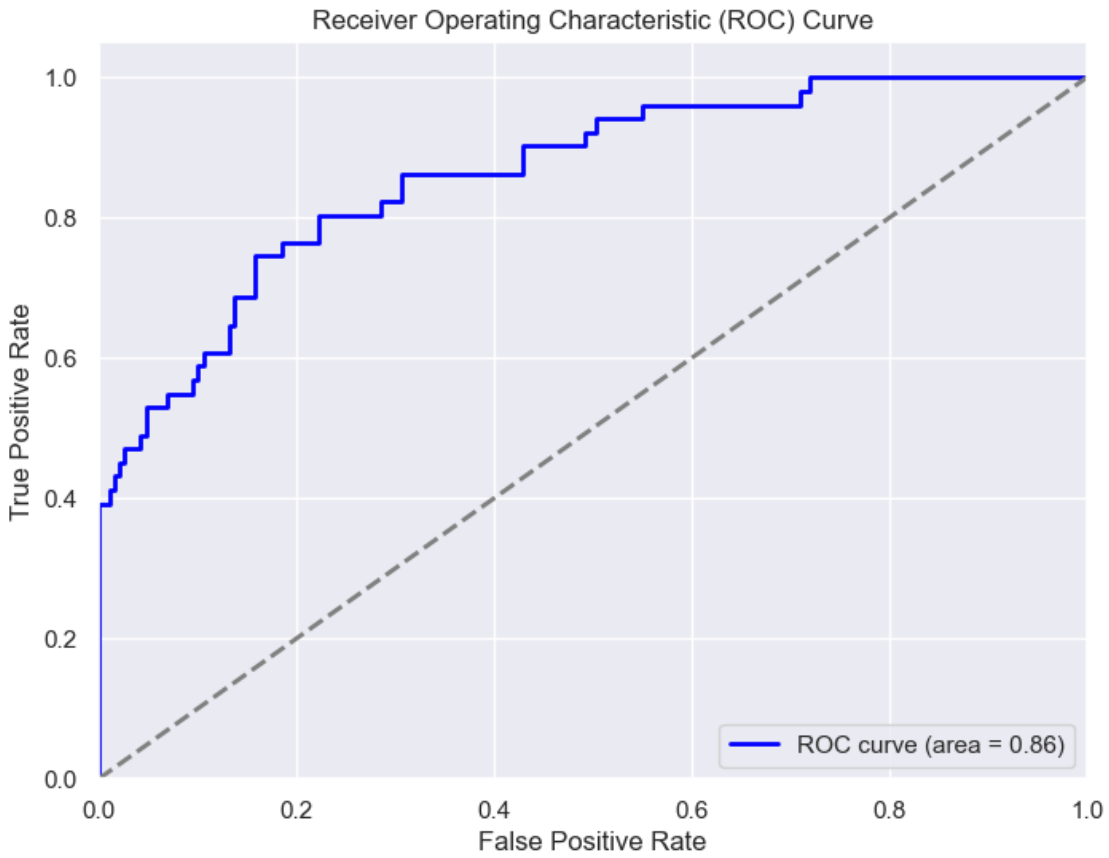
```
Precision Score: 0.52
```

[160]:
```
# Print confusion matrix
conf_matrix = confusion_matrix(y_test, preds_svc)
print("Confusion Matrix:\n", conf_matrix)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
 [[153  36]
 [ 12  39]]
```



[162]:
```
# Compute ROC curve and ROC area
y_proba_svc = best_model.predict_proba(X_test)[:, 1]
fpr_svc, tpr_svc, _ = roc_curve(y_test, y_proba_svc)
roc_auc_svc = auc(fpr_svc, tpr_svc)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_svc, tpr_svc, color='blue', lw=2, label='ROC curve (area = %0.2f)'␣
 ↪% roc_auc_svc)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 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()
```



Random Forest Classifier

[165]:
```
from sklearn.ensemble import RandomForestClassifier
```

[167]:
```
pipeline=Pipeline(steps=[('preprocessor',preprocessor),('smote',SMOTE(random_state=42)),('clas
 ↪RandomForestClassifier())])
```

[169]:
```
param_grid = {
        'classifier__n_estimators': [50, 100],
        'classifier__max_depth': [None, 10],
        'classifier__min_samples_split': [2, 5],
        'classifier__min_samples_leaf': [1, 2],
        'classifier__bootstrap': [True, False]
    }
```

[171]:
```
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
```

[173]:
```
clf=GridSearchCV(pipeline,param_grid,cv=SKF,n_jobs=5,verbose=1)
```

[175]:
```
model=clf.fit(X_train,y_train)
print(model)
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=0, shuffle=True),
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('numeric',
Pipeline(steps=[('scaler',
          StandardScaler())]),
['result']),
('categorical',
Pipeline(steps=[('encoder',
          OneHotEncoder(handle_unknown='ignore'))]),
['gender',
'ethnicity',
'jaundice',
'autism',
'contry_of_res'])])),
                                       ('smote', SMOTE(random_state=42)),
                                       ('classifier',
                                        RandomForestClassifier())]),
             n_jobs=5,
             param_grid={'classifier__bootstrap': [True, False],
                         'classifier__max_depth': [None, 10],
                         'classifier__min_samples_leaf': [1, 2],
                         'classifier__min_samples_split': [2, 5],
                         'classifier__n_estimators': [50, 100]},
             verbose=1)
```

[177]:
```
print(clf.best_params_)
```

```
{'classifier__bootstrap': True, 'classifier__max_depth': 10,
'classifier__min_samples_leaf': 2, 'classifier__min_samples_split': 2,
'classifier__n_estimators': 50}
```

[179]:
```
best_model = clf.best_estimator_
best_model
```

[179]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('numeric',
                                                  Pipeline(steps=[('scaler',
StandardScaler())]),
                                                  ['result']),
                                                 ('categorical',
                                                  Pipeline(steps=[('encoder',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['gender', 'ethnicity',
```

```
                                                  'jaundice', 'autism',
                                                  'contry_of_res'])])),
                ('smote', SMOTE(random_state=42)),
                ('classifier',
                 RandomForestClassifier(max_depth=10, min_samples_leaf=2,
                                        n_estimators=50))])
```

[181]:
```
preds_rf = best_model.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, preds_rf))

print("Classification Report:\n", classification_report(y_test, preds_rf))
```

```
Accuracy Score: 0.8416666666666667
Classification Report:
               precision    recall  f1-score   support

           0       0.90      0.89      0.90       189
           1       0.62      0.65      0.63        51

    accuracy                           0.84       240
   macro avg       0.76      0.77      0.77       240
weighted avg       0.84      0.84      0.84       240
```

[183]:
```
from sklearn.metrics import r2_score
```

[185]:
```
print (r2_score(y_test, preds_rf))
```

```
0.05384375972611233
```

[187]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix, roc_curve, auc, precision_score
```

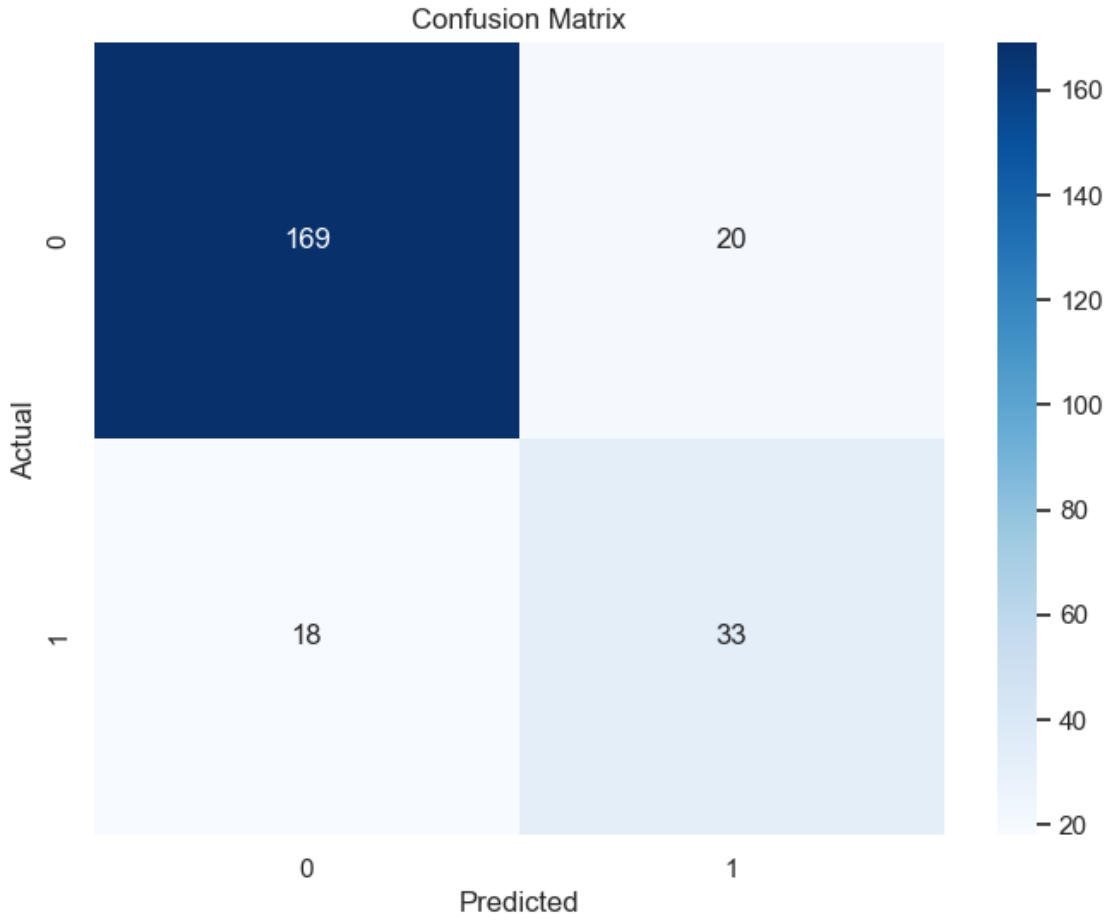
[189]:
```
# Print precision score
precision = precision_score(y_test, preds_rf)
print("Precision Score:", precision)
```

```
Precision Score: 0.6226415094339622
```

[191]:
```
# Print confusion matrix
conf_matrix = confusion_matrix(y_test, preds_rf)
print("Confusion Matrix:\n", conf_matrix)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
```

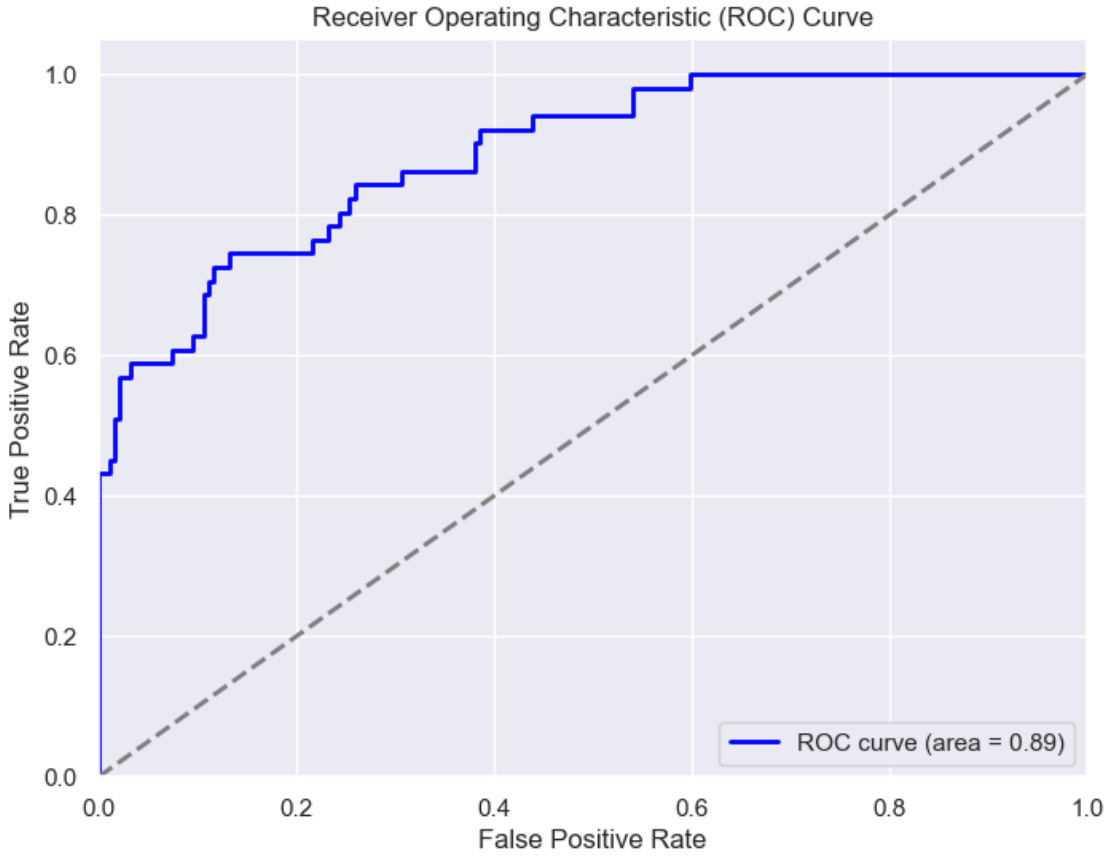
```
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
 [[169  20]
 [ 18  33]]
```



[193]:
```
# Compute ROC curve and ROC area
y_proba_rf = best_model.predict_proba(X_test)[:, 1]
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_proba_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='blue', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc_rf)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
```

```
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 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()
```



Lightgbm

[196]:
```
# !pip install lightgbm
```

[198]:
```
import lightgbm as lgb
```

[199]:
```
pipeline=Pipeline(steps=[('preprocessor',preprocessor),('smote',SMOTE(random_state=42)),('clas
↪lgb.LGBMClassifier())])
```

[200]:
```
param_grid = {'classifier__n_estimators': [50, 150, 200],
        'classifier__max_depth': [4, 6, 8],
        'classifier__learning_rate': [0.025, 0.05, 0.075, 0.1]}
```

[201]:
```
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
```

[202]:
```
clf=GridSearchCV(pipeline,param_grid,cv=SKF,n_jobs=5, verbose=20)
```

[203]:
```
model=clf.fit(X_train,y_train)
print(model)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[LightGBM] [Info] Number of positive: 426, number of negative: 426
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.001296 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 453
[LightGBM] [Info] Number of data points in the train set: 852, number of used
features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=0, shuffle=True),
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('numeric',
Pipeline(steps=[('scaler',
          StandardScaler())]),
['result']),
('categorical',
Pipeline(steps=[('encoder',
          OneHotEncoder(handle_unknown='ignore'))]),
['gender',
'ethnicity',
'jaundice',
'autism',
'contry_of_res'])])),
                                       ('smote', SMOTE(random_state=42)),
                                       ('classifier', LGBMClassifier())]),
             n_jobs=5,
             param_grid={'classifier__learning_rate': [0.025, 0.05, 0.075, 0.1],
                         'classifier__max_depth': [4, 6, 8],
                         'classifier__n_estimators': [50, 150, 200]},
             verbose=20)
```

[210]:
```
print(clf.best_params_)
```

```
{'classifier__learning_rate': 0.025, 'classifier__max_depth': 6,
'classifier__n_estimators': 150}
```

[212]:
```
best_model = clf.best_estimator_
best_model
```

[212]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('numeric',
                                                  Pipeline(steps=[('scaler',
StandardScaler())]),
                                                  ['result']),
                                                 ('categorical',
                                                  Pipeline(steps=[('encoder',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['gender', 'ethnicity',
                                                   'jaundice', 'autism',
                                                   'contry_of_res'])])),
                ('smote', SMOTE(random_state=42)),
                ('classifier',
                 LGBMClassifier(learning_rate=0.025, max_depth=6,
                                n_estimators=150))])
```

[214]:
```
preds_lgb = best_model.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, preds_lgb))

print("Classification Report:\n", classification_report(y_test, preds_lgb))
```

```
Accuracy Score: 0.8
Classification Report:
               precision    recall  f1-score   support

           0       0.89      0.85      0.87       189
           1       0.53      0.61      0.56        51

    accuracy                           0.80       240
   macro avg       0.71      0.73      0.72       240
weighted avg       0.81      0.80      0.81       240
```

[216]:
```
from sklearn.metrics import r2_score
```

[218]:
```
print (r2_score(y_test, preds_lgb))
```

```
-0.19514472455648968
```

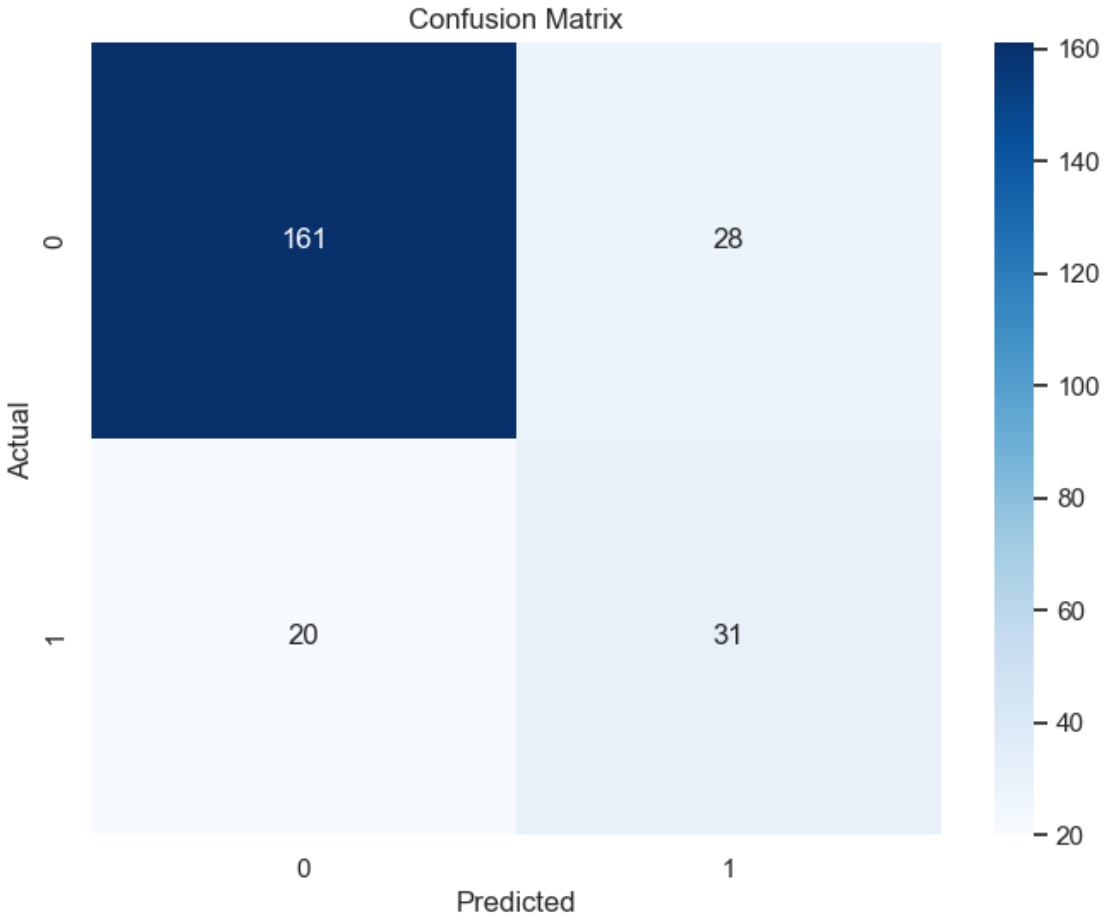
[220]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix, roc_curve, auc, precision_score
```

[222]:
```
# Print precision score
precision = precision_score(y_test, preds_lgb)
print("Precision Score:", precision)
```

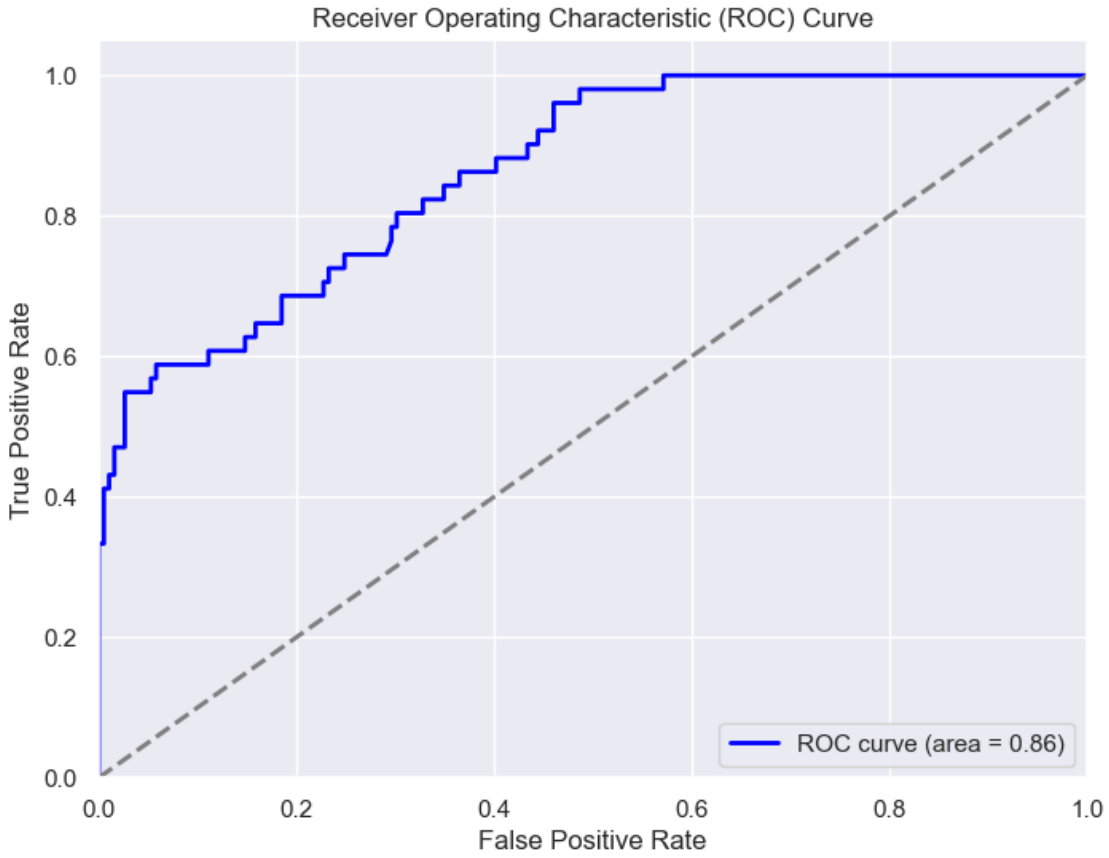
```
Precision Score: 0.5254237288135594
```

[224]:
```
# Print confusion matrix
conf_matrix = confusion_matrix(y_test, preds_lgb)
print("Confusion Matrix:\n", conf_matrix)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
 [[161  28]
 [ 20  31]]
```



[226]:
```
# Compute ROC curve and ROC area
y_proba_lgb = best_model.predict_proba(X_test)[:, 1]
fpr_lgb, tpr_lgb, _ = roc_curve(y_test, y_proba_lgb)
roc_auc_lgb = auc(fpr_lgb, tpr_lgb)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_lgb, tpr_lgb, color='blue', lw=2, label='ROC curve (area = %0.2f)'␣
 ↪% roc_auc_lgb)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 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()
```



KNeighborsClassifier

[229]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[231]:
```
pipeline=Pipeline(steps=[('preprocessor',preprocessor),('smote',SMOTE(random_state=42)),('clas
 ↪KNeighborsClassifier())])
```

[233]:
```
param_grid={
        'classifier':[KNeighborsClassifier()],
        'classifier__n_neighbors':[3,5,7,9,11,13],
        'classifier__weights':['uniform', 'distance'],
        'classifier__p': [1,2]

    }
```

[235]:
```
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
```

[237]:
```
clf=GridSearchCV(pipeline,param_grid,cv=SKF,n_jobs=5,verbose=1)
```

[239]:
```
model=clf.fit(X_train,y_train)
print(model)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=0, shuffle=True),
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('numeric',
Pipeline(steps=[('scaler',
          StandardScaler())]),
['result']),
('categorical',
Pipeline(steps=[('encoder',
          OneHotEncoder(handle_unknown='ignore'))]),
['gender',
'ethnicity',
'jaundice',
'autism',
'contry_of_res'])])),
                                       ('smote', SMOTE(random_state=42)),
                                       ('classifier', KNeighborsClassifier())]),
             n_jobs=5,
             param_grid={'classifier': [KNeighborsClassifier(n_neighbors=11,
                                                             p=1)],
                         'classifier__n_neighbors': [3, 5, 7, 9, 11, 13],
                         'classifier__p': [1, 2],
                         'classifier__weights': ['uniform', 'distance']},
             verbose=1)
```

[241]:
```
print(clf.best_params_)
```

```
{'classifier': KNeighborsClassifier(n_neighbors=11, p=1),
'classifier__n_neighbors': 11, 'classifier__p': 1, 'classifier__weights':
'uniform'}
```

[243]:
```
best_model = clf.best_estimator_
best_model
```

[243]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('numeric',
                                                  Pipeline(steps=[('scaler',
StandardScaler())]),
                                                  ['result']),
                                                 ('categorical',
                                                  Pipeline(steps=[('encoder',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['gender', 'ethnicity',
                                                   'jaundice', 'autism',
                                                   'contry_of_res'])])),
                ('smote', SMOTE(random_state=42)),
                ('classifier', KNeighborsClassifier(n_neighbors=11, p=1))])
```

[245]:
```
preds_kn = best_model.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, preds_kn))

print("Classification Report:\n", classification_report(y_test, preds_kn))
```

```
Accuracy Score: 0.8708333333333333
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.95      0.92       189
           1       0.75      0.59      0.66        51

    accuracy                           0.87       240
   macro avg       0.82      0.77      0.79       240
weighted avg       0.86      0.87      0.86       240
```

[247]:
```
print (r2_score(y_test, preds_kn))
```

```
0.2281356987239338
```

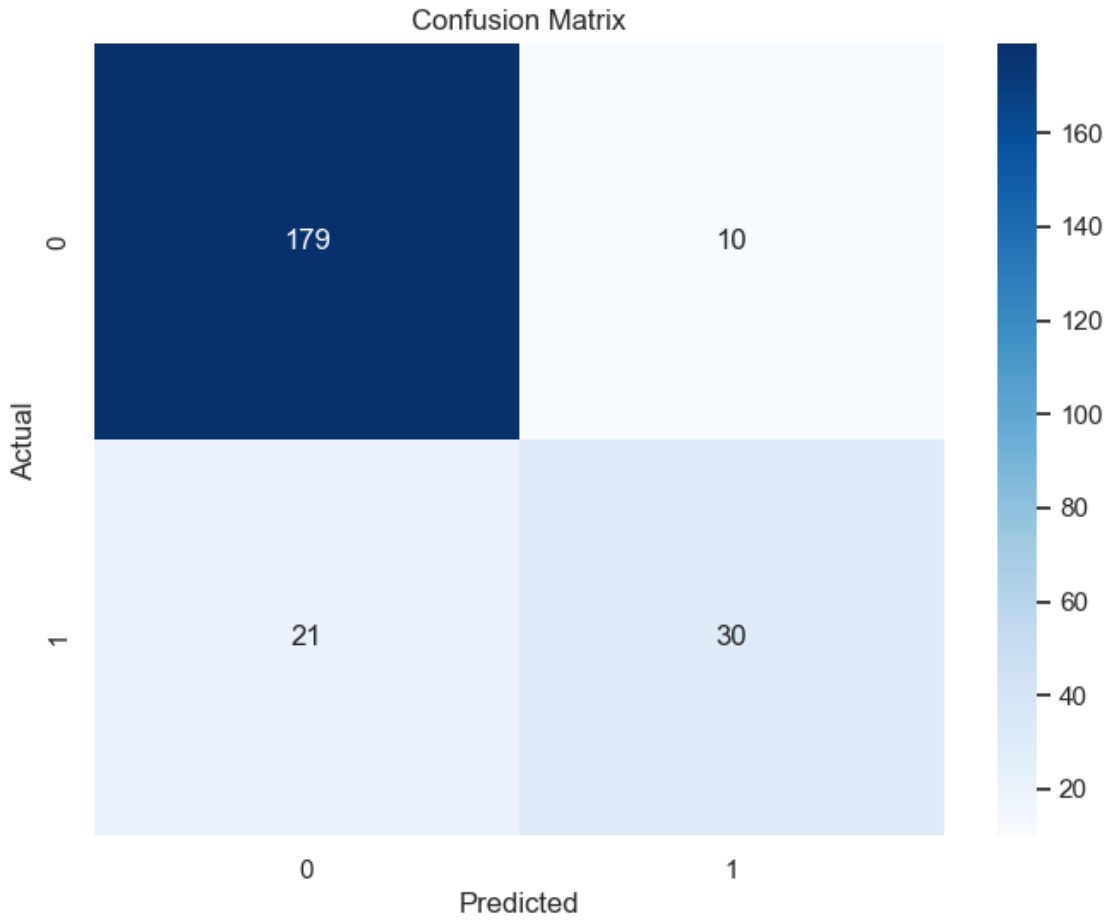
[249]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix, roc_curve, auc, precision_score
```

[251]:
```
# Print precision score
precision = precision_score(y_test, preds_kn)
print("Precision Score:", precision)
```

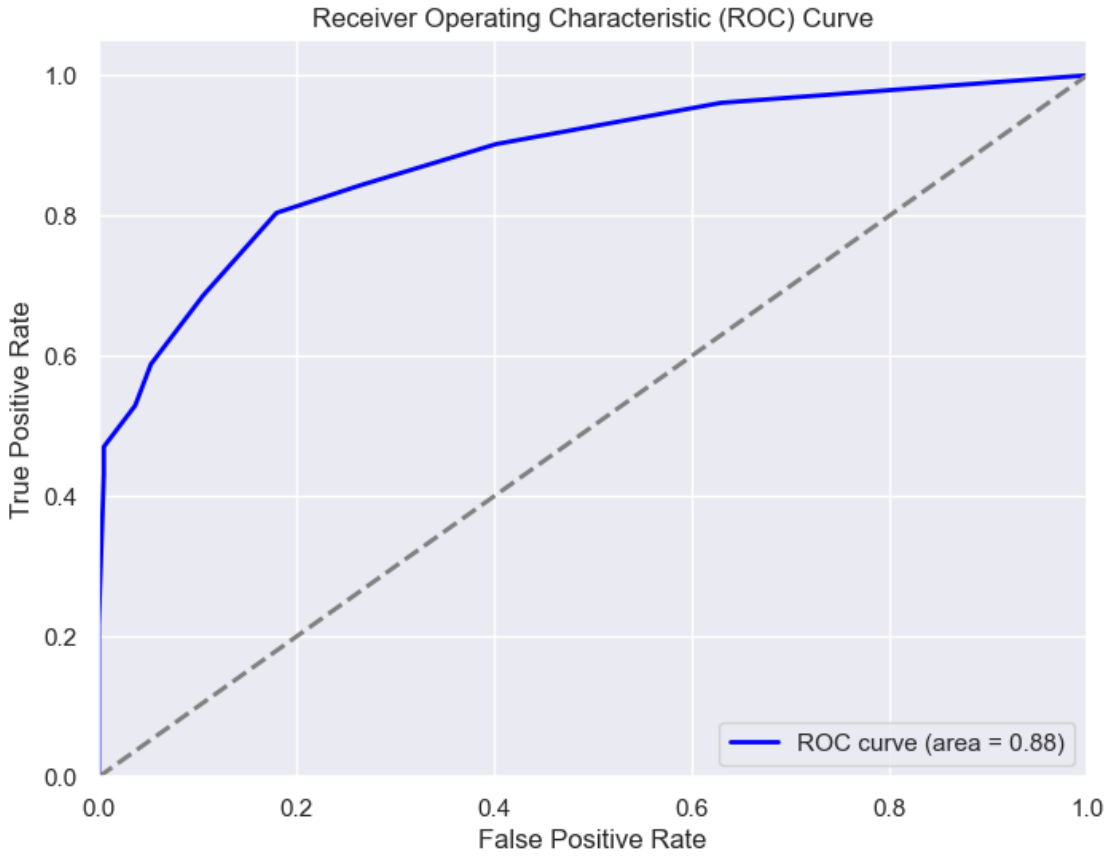
```
Precision Score: 0.75
```

[253]:
```python
# Print confusion matrix
conf_matrix = confusion_matrix(y_test, preds_kn)
print("Confusion Matrix:\n", conf_matrix)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
 [[179  10]
 [ 21  30]]
```



[255]:
```
# Compute ROC curve and ROC area
y_proba_kn = best_model.predict_proba(X_test)[:, 1]
fpr_kn, tpr_kn, _ = roc_curve(y_test, y_proba_kn)
roc_auc_kn = auc(fpr_kn, tpr_kn)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_kn, tpr_kn, color='blue', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc_kn)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 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()
```



XGBoost

[258]:
```
# !pip install xgboost
```

[260]:
```
from xgboost import XGBClassifier
```

[262]:
```
pipeline=Pipeline(steps=[('preprocessor',preprocessor),('smote',SMOTE(random_state=42)),('clas
↪XGBClassifier())])
```

[264]:
```
param_grid = {
    'classifier__max_depth': [2, 3, 5],
    'classifier__learning_rate': [0.01, 0.1],
    'classifier__n_estimators': [100, 200],
    'classifier__subsample': [0.6, 0.8, 1.0],
    'classifier__colsample_bytree': [0.6, 0.8, 1.0]
}
```

[266]:
```
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
```

[268]:
```
clf=GridSearchCV(pipeline,param_grid,cv=SKF,n_jobs=5,verbose=1)
```

[270]:
```
model=clf.fit(X_train,y_train)
print(model)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=0, shuffle=True),
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('numeric',
Pipeline(steps=[('scaler',
          StandardScaler())]),
['result']),
('categorical',
Pipeline(steps=[('encoder',
          OneHotEncoder(handle_unknown='ignore'))]),
['gender',
'ethnicity',
'jaundice',
'autism',
'contry_o…
                                                  missing=nan,
                                                  monotone_constraints=None,
                                                  multi_strategy=None,
                                                  n_estimators=None,
                                                  n_jobs=None,
                                                  num_parallel_tree=None,
                                                  random_state=None,
…))]),
             n_jobs=5,
             param_grid={'classifier__colsample_bytree': [0.6, 0.8, 1.0],
```

```
                          'classifier__learning_rate': [0.01, 0.1],
                          'classifier__max_depth': [2, 3, 5],
                          'classifier__n_estimators': [100, 200],
                          'classifier__subsample': [0.6, 0.8, 1.0]},
                verbose=1)
```

[272]:
```
print(clf.best_params_)
```

```
{'classifier__colsample_bytree': 0.6, 'classifier__learning_rate': 0.1,
'classifier__max_depth': 2, 'classifier__n_estimators': 200,
'classifier__subsample': 1.0}
```

[274]:
```
best_model = clf.best_estimator_
best_model
```

[274]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('numeric',
                                                  Pipeline(steps=[('scaler',
StandardScaler())]),
                                                  ['result']),
                                                 ('categorical',
                                                  Pipeline(steps=[('encoder',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['gender', 'ethnicity',
                                                   'jaundice', 'autism',
                                                   'contry_of_res'])])),
                ('smote', SMOTE(random_state=42)),
                ('classifier',
                 XGBClassifier(base_scor…
                               feature_types=None, gamma=None, grow_policy=None,
                               importance_type=None,
                               interaction_constraints=None, learning_rate=0.1,
                               max_bin=None, max_cat_threshold=None,
                               max_cat_to_onehot=None, max_delta_step=None,
                               max_depth=2, max_leaves=None,
                               min_child_weight=None, missing=nan,
                               monotone_constraints=None, multi_strategy=None,
                               n_estimators=200, n_jobs=None,
                               num_parallel_tree=None, random_state=None,
…))])
```

[276]:
```
preds_xgb = best_model.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, preds_xgb))

print("Classification Report:\n", classification_report(y_test, preds_xgb))
```

```
Accuracy Score: 0.8375
Classification Report:
```

```
                precision    recall  f1-score   support

           0       0.90      0.89      0.90       189
           1       0.61      0.65      0.63        51

    accuracy                           0.84       240
   macro avg       0.76      0.77      0.76       240
weighted avg       0.84      0.84      0.84       240
```

[278]:
```
print (r2_score(y_test, preds_xgb))
```

```
0.02894491129785215
```

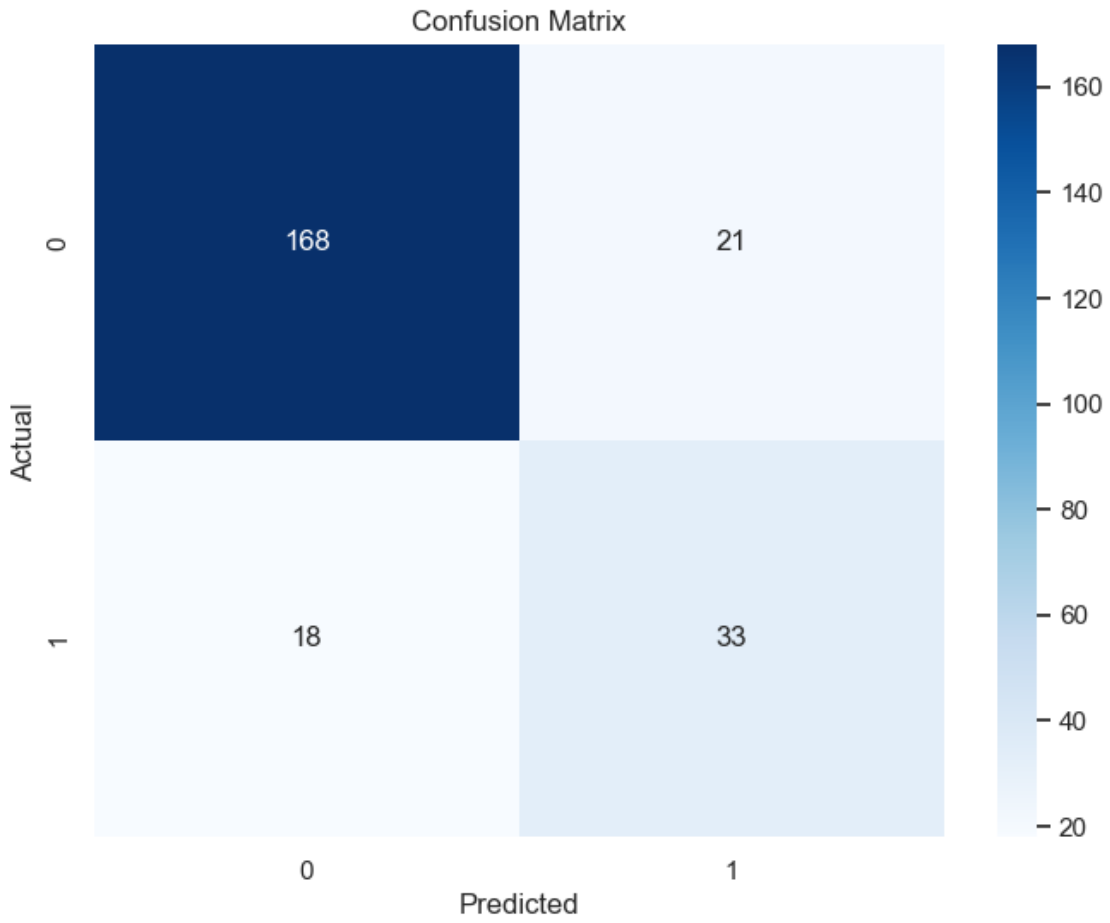
[280]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix, roc_curve, auc, precision_score
```

[282]:
```
# Print precision score
precision = precision_score(y_test, preds_xgb)
print("Precision Score:", precision)
```

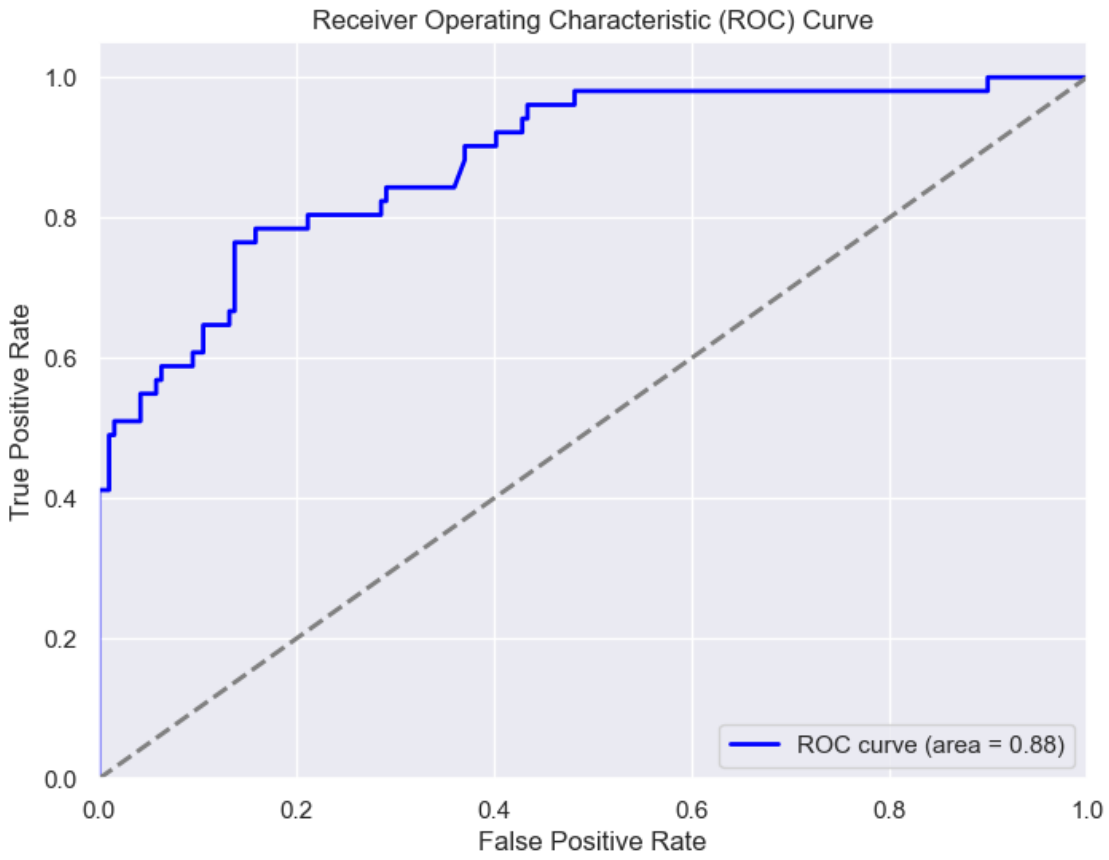
```
Precision Score: 0.6111111111111112
```

[284]:
```
# Print confusion matrix
conf_matrix = confusion_matrix(y_test, preds_xgb)
print("Confusion Matrix:\n", conf_matrix)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
 [[168  21]
 [ 18  33]]
```

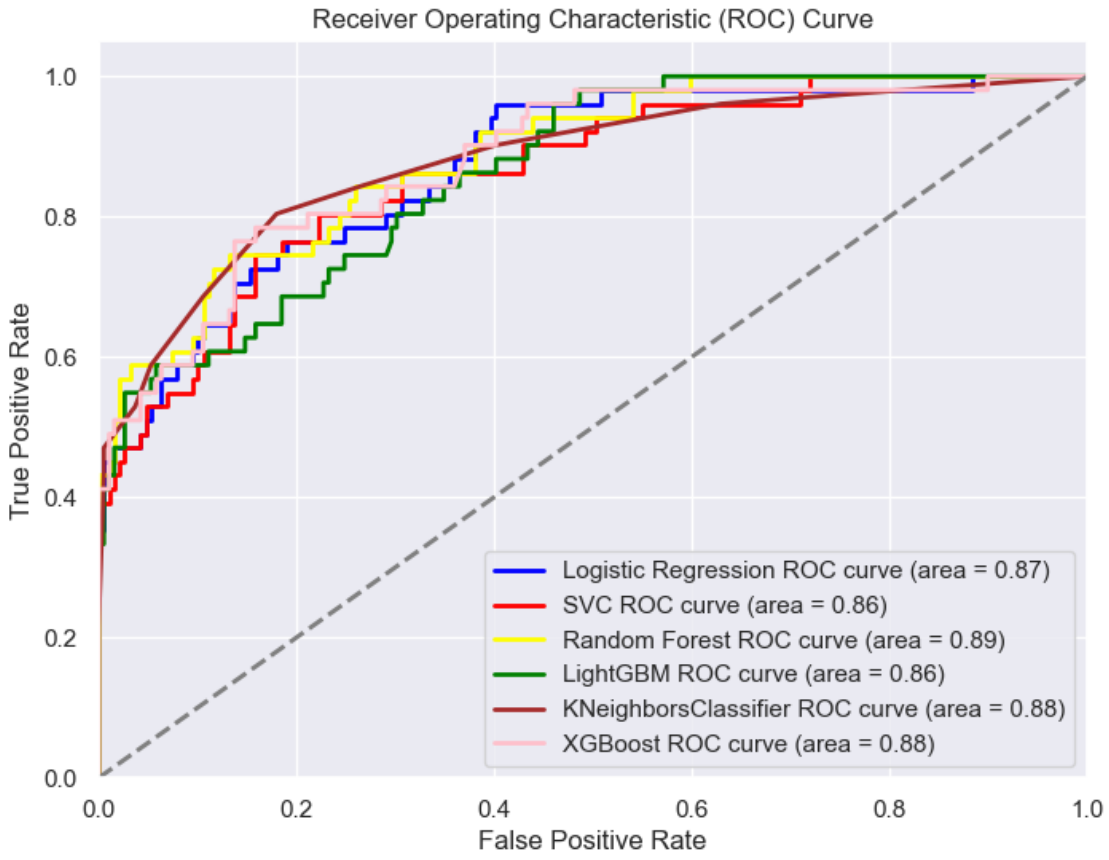


[286]:
```
# Compute ROC curve and ROC area
y_proba_xgb = best_model.predict_proba(X_test)[:, 1]
fpr_xgb, tpr_xgb, _ = roc_curve(y_test, y_proba_xgb)
roc_auc_xgb = auc(fpr_xgb, tpr_xgb)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_xgb, tpr_xgb, color='blue', lw=2, label='ROC curve (area = %0.2f)'␣
 ↪% roc_auc_xgb)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 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()
```



[288]:
```
# Plotting both ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, color='blue', lw=2, label='Logistic Regression ROC␣
 ↪curve (area = %0.2f)' % roc_auc_lr )
plt.plot(fpr_svc, tpr_svc, color='red', lw=2, label='SVC ROC curve (area = %0.
 ↪2f)' % roc_auc_svc)
plt.plot(fpr_rf, tpr_rf, color='yellow', lw=2, label='Random Forest ROC curve␣
 ↪(area = %0.2f)' % roc_auc_rf)
plt.plot(fpr_lgb, tpr_lgb, color='green', lw=2, label='LightGBM ROC curve (area␣
 ↪= %0.2f)' % roc_auc_lgb)
plt.plot(fpr_kn, tpr_kn, color='brown', lw=2, label='KNeighborsClassifier ROC␣
 ↪curve (area = %0.2f)' % roc_auc_kn)
plt.plot(fpr_xgb, tpr_xgb, color='pink', lw=2, label='XGBoost ROC curve (area =␣
 ↪%0.2f)' % roc_auc_xgb)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 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()
```



[290]:
```
from sklearn.metrics import precision_score, accuracy_score, roc_auc_score,␣
 ↪r2_score
# Calculate metrics for Logistic Regression
lr_accuracy = accuracy_score(y_test, preds_lr)
lr_precision = precision_score(y_test, preds_lr)
lr_r2_score = r2_score(y_test, preds_lr)
lr_roc_auc = roc_auc_score(y_test, y_proba_lr)

# Calculate metrics for SVC
svc_accuracy = accuracy_score(y_test, preds_svc)
svc_precision = precision_score(y_test, preds_svc)
svc_r2_score = r2_score(y_test, preds_svc)
svc_roc_auc = roc_auc_score(y_test, y_proba_svc)

# Calculate metrics for Random Forest
```

```
rf_accuracy = accuracy_score(y_test, preds_rf)
rf_precision = precision_score(y_test, preds_rf)
rf_r2_score = r2_score(y_test, preds_rf)
rf_roc_auc = roc_auc_score(y_test, y_proba_rf)

# Calculate metrics for LightGBM
lgb_accuracy = accuracy_score(y_test, preds_lgb)
lgb_precision = precision_score(y_test, preds_lgb)
lgb_r2_score = r2_score(y_test, preds_lgb)
lgb_roc_auc = roc_auc_score(y_test, y_proba_lgb)

# Calculate metrics for KNeighborsClassifier
kn_accuracy = accuracy_score(y_test, preds_kn)
kn_precision = precision_score(y_test, preds_kn)
kn_r2_score = r2_score(y_test, preds_kn)
kn_roc_auc = roc_auc_score(y_test, y_proba_kn)

# Calculate metrics for XGBoost
xgb_accuracy = accuracy_score(y_test, preds_xgb)
xgb_precision = precision_score(y_test, preds_xgb)
xgb_r2_score = r2_score(y_test, preds_xgb)
xgb_roc_auc = roc_auc_score(y_test, y_proba_xgb)
```

[292]:
```
# Create a data frame to display the metrics
results_df = pd.DataFrame({
    'Model': ['Logistic Regression', 'SVC', 'Random Forest', 'LightGBM',␣
 ↪'KNeighborsClassifier', 'XGBoost'],
    'Accuracy': [lr_accuracy, svc_accuracy, rf_accuracy, lgb_accuracy,␣
 ↪kn_accuracy, xgb_accuracy],
    'Precision': [lr_precision, svc_precision, rf_precision, lgb_precision,␣
 ↪kn_precision, xgb_precision],
    'R2 Score': [lr_r2_score, svc_r2_score, rf_r2_score, lgb_r2_score,␣
 ↪kn_r2_score, xgb_r2_score],
    'ROC AUC': [lr_roc_auc, svc_roc_auc,  rf_roc_auc,  lgb_roc_auc, ␣
 ↪kn_roc_auc,  xgb_roc_auc]
})

# Display the results
print("Metrics Comparison:")
print(results_df)
```

```
Metrics Comparison:
                 Model  Accuracy  Precision  R2 Score   ROC AUC
0  Logistic Regression  0.783333   0.493671 -0.294740  0.873120
1                  SVC  0.800000   0.520000 -0.195145  0.863575
2        Random Forest  0.841667   0.622642  0.053844  0.886710
3             LightGBM  0.800000   0.525424 -0.195145  0.863316
```

```
4  KNeighborsClassifier  0.870833   0.750000  0.228136  0.880122
5               XGBoost  0.837500   0.611111  0.028945  0.880693
```