Autism Prediction Using Machine Learning

Importing the dependencies

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, cross_val_score,
RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import pickle
```

Data Loading & Understanding

```
# read the csv data to a pandas dataframe
df = pd.read_csv("train.csv")
```

Initial Inspection

```
df.shape
```

```
(800, 22)
```

```
df.head()
```

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

   A8_Score  A9_Score  ...  gender       ethnicity jaundice austim  \
```

```
0         0         1  ...       f               ?       no     no
1         0         0  ...       m               ?       no     no
2         1         1  ...       m  White-European       no    yes
3         0         0  ...       f               ?       no     no
4         0         0  ...       m               ?       no     no

   contry_of_res used_app_before     result     age_desc  relation
Class/ASD
0        Austria              no   6.351166  18 and more      Self
0
1          India              no   2.255185  18 and more      Self
0
2  United States              no  14.851484  18 and more      Self
1
3  United States              no   2.276617  18 and more      Self
0
4   South Africa              no  -4.777286  18 and more      Self
0

[5 rows x 22 columns]



df.tail()

      ID  A1_Score  A2_Score  A3_Score  A4_Score  A5_Score
A6_Score  \
795  796         0         1         0         0         0         0

796  797         0         1         1         0         0         1

797  798         0         0         0         0         0         0

798  799         0         0         0         0         0         0

799  800         0         1         0         0         0         0


     A7_Score  A8_Score  A9_Score  ...  gender       ethnicity
jaundice  \
795         0         0         1  ...       m        Hispanic
no
796         0         1         1  ...       m  White-European
no
797         0         0         0  ...       m     South Asian
yes
798         0         0         0  ...       f               ?
no
799         0         0         0  ...       f               ?
no
```

```
    austim         contry_of_res used_app_before     result
age_desc  \
795     no           New Zealand              no  12.999501  18 and
more
796     no                Cyprus              no  13.561518  18 and
more
797     no           New Zealand              no   2.653177  18 and
more
798     no                Canada              no   9.069342  18 and
more
799     no  United Arab Emirates             yes   2.243304  18 and
more

     relation Class/ASD
795      Self         0
796      Self         0
797      Self         0
798      Self         0
799      Self         0

[5 rows x 22 columns]
```

```
# display all columns of a dataframe
pd.set_option('display.max_columns', None)

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  austim           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
```

```
# convert age column datatype to integer
df["age"] = df["age"].astype(int)

df.head(2)
```

```
   ID  A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score
A7_Score  \
0   1         1         0         1         0         1         0
1
1   2         0         0         0         0         0         0
0

   A8_Score  A9_Score  A10_Score  age gender ethnicity jaundice austim
\
0         0         1          1   38      f         ?       no     no

1         0         0          0   47      m         ?       no     no


  contry_of_res used_app_before    result     age_desc relation
Class/ASD
0       Austria              no  6.351166  18 and more     Self
0
1         India              no  2.255185  18 and more     Self
0
```

```
for col in df.columns:
  numerical_features = ["ID", "age", "result"]
  if col not in numerical_features:
    print(col, df[col].unique())
    print("-"*50)
```

```
A1_Score [1 0]
--------------------------------------------------
A2_Score [0 1]
--------------------------------------------------
A3_Score [1 0]
--------------------------------------------------
A4_Score [0 1]
--------------------------------------------------
A5_Score [1 0]
--------------------------------------------------
A6_Score [0 1]
--------------------------------------------------
```

```
A7_Score [1 0]
--------------------------------------------------
A8_Score [0 1]
--------------------------------------------------
A9_Score [1 0]
--------------------------------------------------
A10_Score [1 0]
--------------------------------------------------
gender ['f' 'm']
--------------------------------------------------
ethnicity ['?' 'White-European' 'Middle Eastern ' 'Pasifika' 'Black'
'Others'
 'Hispanic' 'Asian' 'Turkish' 'South Asian' 'Latino' 'others']
--------------------------------------------------
jaundice ['no' 'yes']
--------------------------------------------------
austim ['no' 'yes']
--------------------------------------------------
contry_of_res ['Austria' 'India' 'United States' 'South Africa'
'Jordan'
 'United Kingdom' 'Brazil' 'New Zealand' 'Canada' 'Kazakhstan'
 'United Arab Emirates' 'Australia' 'Ukraine' 'Iraq' 'France'
'Malaysia'
 'Viet Nam' 'Egypt' 'Netherlands' 'Afghanistan' 'Oman' 'Italy'
 'AmericanSamoa' 'Bahamas' 'Saudi Arabia' 'Ireland' 'Aruba' 'Sri
Lanka'
 'Russia' 'Bolivia' 'Azerbaijan' 'Armenia' 'Serbia' 'Ethiopia'
'Sweden'
 'Iceland' 'Hong Kong' 'Angola' 'China' 'Germany' 'Spain' 'Tonga'
 'Pakistan' 'Iran' 'Argentina' 'Japan' 'Mexico' 'Nicaragua' 'Sierra
Leone'
 'Czech Republic' 'Niger' 'Romania' 'Cyprus' 'Belgium' 'Burundi'
 'Bangladesh']
--------------------------------------------------
used_app_before ['no' 'yes']
--------------------------------------------------
age_desc ['18 and more']
--------------------------------------------------
relation ['Self' 'Relative' 'Parent' '?' 'Others' 'Health care
professional']
--------------------------------------------------
Class/ASD [0 1]
--------------------------------------------------
```

```
# dropping ID & age_desc column
df = df.drop(columns=["ID", "age_desc"])

df.shape
```

```
(800, 20)
```

```
df.head(2)
```

```
   A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score
A7_Score  \
0         1         0         1         0         1         0
1
1         0         0         0         0         0         0
0

   A8_Score  A9_Score  A10_Score  age gender ethnicity jaundice austim
\
0         0         1          1   38      f         ?       no     no

1         0         0          0   47      m         ?       no     no


  contry_of_res used_app_before    result relation  Class/ASD
0       Austria              no  6.351166     Self          0
1         India              no  2.255185     Self          0
```

```
df.columns
```

```
Index(['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score',
'A6_Score',
       'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'age',
'gender',
       'ethnicity', 'jaundice', 'austim', 'contry_of_res',
'used_app_before',
       'result', 'relation', 'Class/ASD'],
      dtype='object')
```

```
df["contry_of_res"].unique()
```

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

```
# define the mapping dictionary for country names
mapping = {
    "Viet Nam": "Vietnam",
    "AmericanSamoa": "United States",
    "Hong Kong": "China"
}

# repalce value in the country column
df["contry_of_res"] = df["contry_of_res"].replace(mapping)

df["contry_of_res"].unique()
```

```
array(['Austria', 'India', 'United States', 'South Africa', 'Jordan',
       'United Kingdom', 'Brazil', 'New Zealand', 'Canada',
'Kazakhstan',
       'United Arab Emirates', 'Australia', 'Ukraine', 'Iraq',
'France',
       'Malaysia', 'Vietnam', 'Egypt', 'Netherlands', 'Afghanistan',
       'Oman', 'Italy', 'Bahamas', 'Saudi Arabia', 'Ireland', 'Aruba',
       'Sri Lanka', 'Russia', 'Bolivia', 'Azerbaijan', 'Armenia',
       'Serbia', 'Ethiopia', 'Sweden', 'Iceland', 'China', 'Angola',
       'Germany', 'Spain', 'Tonga', 'Pakistan', 'Iran', 'Argentina',
       'Japan', 'Mexico', 'Nicaragua', 'Sierra Leone', 'Czech
Republic',
       'Niger', 'Romania', 'Cyprus', 'Belgium', 'Burundi',
'Bangladesh'],
      dtype=object)
```

```
# taget class distribution
df["Class/ASD"].value_counts()
```

```
Class/ASD
0    639
1    161
Name: count, dtype: int64
```

Insights:

Exploratory Data Analysis (EDA)

```
df.shape
```

```
(800, 20)
```

```
df.columns
```

```
Index(['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score',
'A6_Score',
       'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'age',
'gender',
       'ethnicity', 'jaundice', 'austim', 'contry_of_res',
'used_app_before',
       'result', 'relation', 'Class/ASD'],
      dtype='object')
```

```
df.head(2)
```

```
   A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score
A7_Score  \
0         1         0         1         0         1         0
1
1         0         0         0         0         0         0
0

   A8_Score  A9_Score  A10_Score  age gender ethnicity jaundice austim
\
0         0         1          1   38      f         ?       no     no

1         0         0          0   47      m         ?       no     no


  contry_of_res used_app_before    result relation  Class/ASD
0       Austria              no  6.351166     Self          0
1         India              no  2.255185     Self          0
```

```
df.describe()
```

```
         A1_Score    A2_Score    A3_Score   A4_Score    A5_Score
A6_Score  \
count  800.000000  800.000000  800.000000  800.00000  800.000000
800.000000
mean     0.560000    0.530000    0.450000    0.41500    0.395000
0.303750
std      0.496697    0.499411    0.497805    0.49303    0.489157
0.460164
min      0.000000    0.000000    0.000000    0.00000    0.000000
0.000000
25%      0.000000    0.000000    0.000000    0.00000    0.000000
0.000000
50%      1.000000    1.000000    0.000000    0.00000    0.000000
0.000000
75%      1.000000    1.000000    1.000000    1.00000    1.000000
1.000000
max      1.000000    1.000000    1.000000    1.00000    1.000000
1.000000

         A7_Score    A8_Score    A9_Score   A10_Score         age
```

```
result  \
count  800.000000  800.000000  800.000000  800.000000  800.000000
800.000000
mean     0.397500    0.508750    0.495000    0.617500   27.963750
8.537303
std      0.489687    0.500236    0.500288    0.486302   16.329827
4.807676
min      0.000000    0.000000    0.000000    0.000000    2.000000   -
6.137748
25%      0.000000    0.000000    0.000000    0.000000   17.000000
5.306575
50%      0.000000    1.000000    0.000000    1.000000   24.000000
9.605299
75%      1.000000    1.000000    1.000000    1.000000   35.250000
12.514484
max      1.000000    1.000000    1.000000    1.000000   89.000000
15.853126

        Class/ASD
count  800.000000
mean     0.201250
std      0.401185
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      1.000000
```

Univariate Analysis

Numerical Columns:

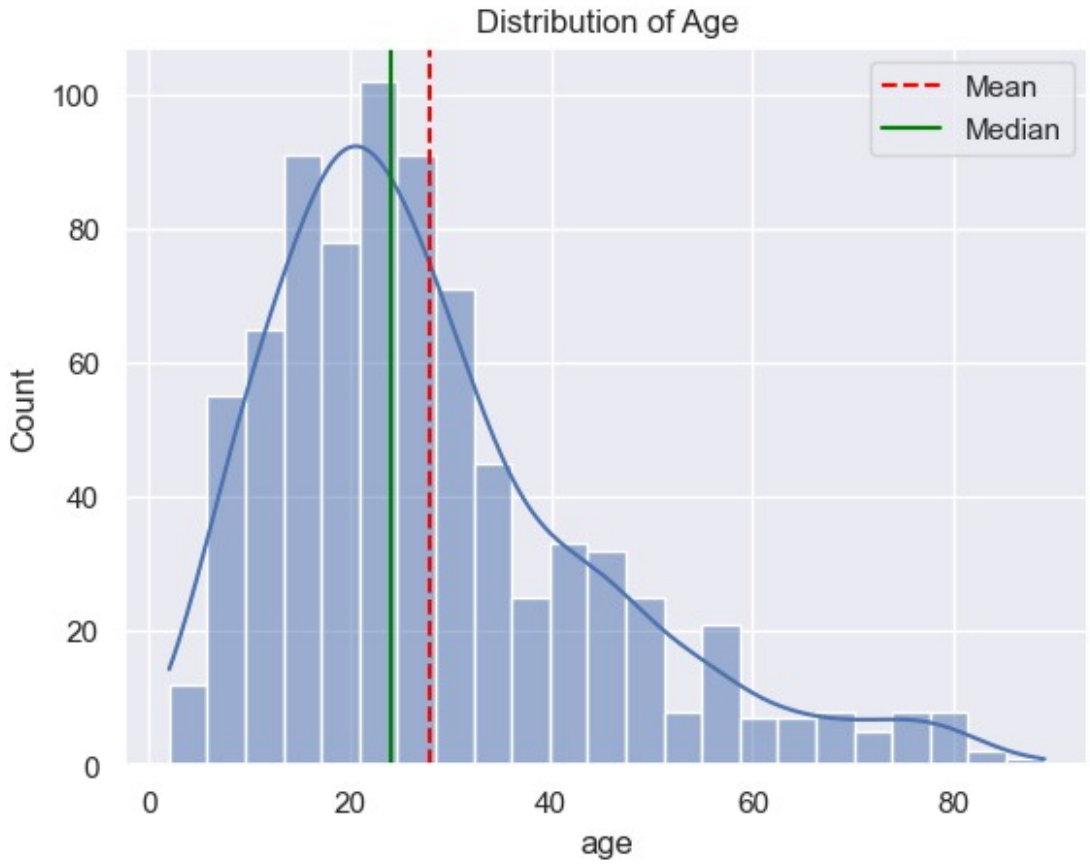
```
# set the desired theme
sns.set_theme(style="darkgrid")
```

Distribution Plots

```
# Histogram for "age"

sns.histplot(df["age"], kde=True)
plt.title("Distribution of Age")

# calculate mean and median
age_mean = df["age"].mean()
age_median = df["age"].median()

print("Mean:", age_mean)
```

```
print("Median:", age_median)


# add vertical lines for mean and median
plt.axvline(age_mean, color="red", linestyle="--", label="Mean")
plt.axvline(age_median, color="green", linestyle="-", label="Median")

plt.legend()

plt.show()
```

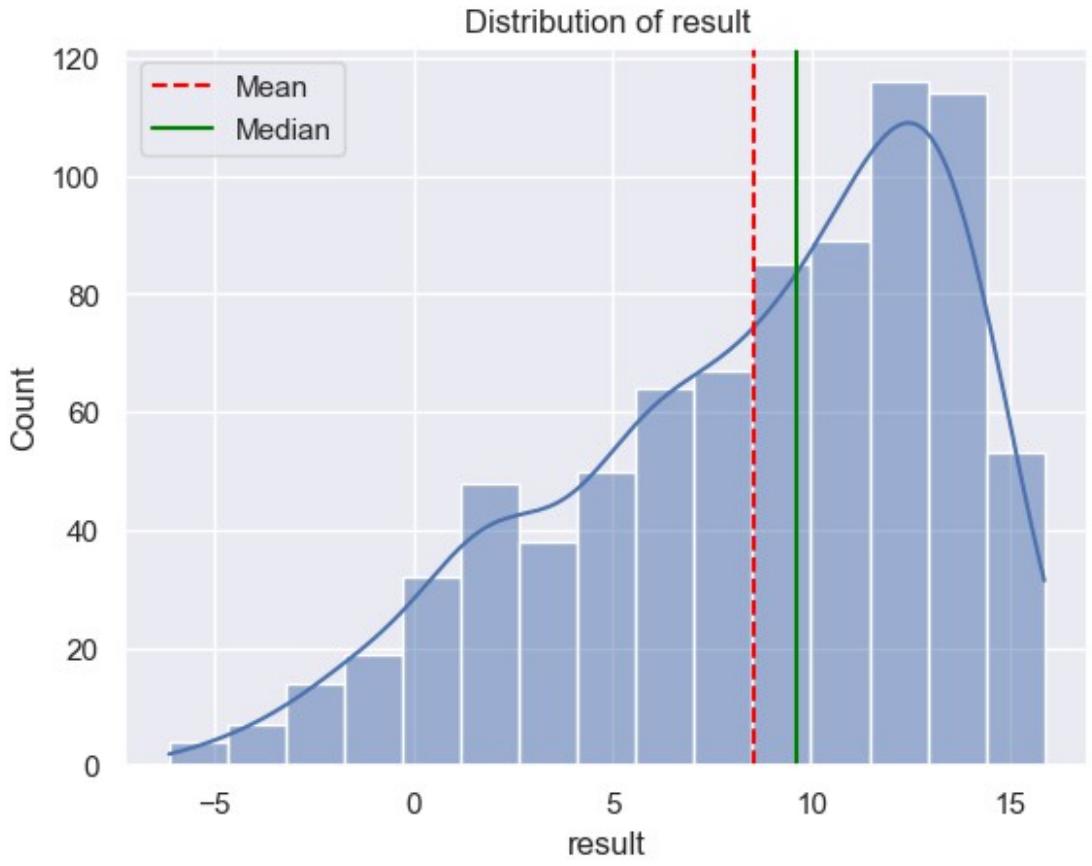
```
Mean: 27.96375
Median: 24.0
```




```
# Histogram for "result"

sns.histplot(df["result"], kde=True)
plt.title("Distribution of result")

# calculate mean and median
result_mean = df["result"].mean()
```

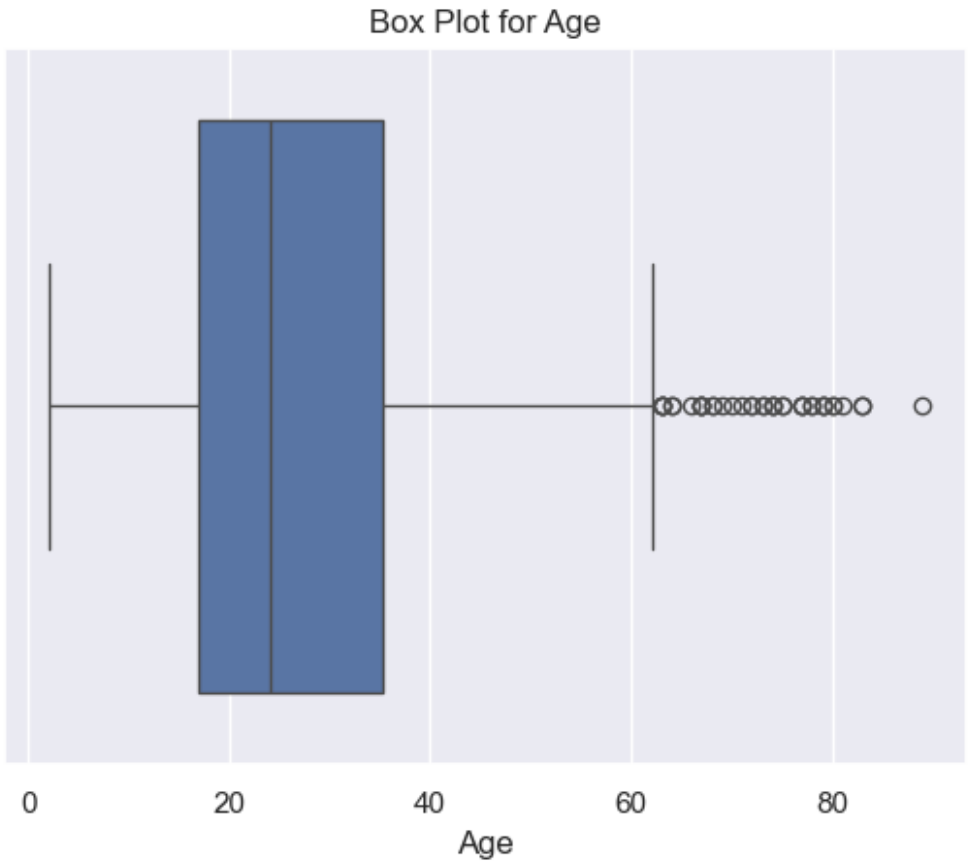
```
result_median = df["result"].median()

print("Mean:", result_mean)
print("Median:", result_median)


# add vertical lines for mean and median
plt.axvline(result_mean, color="red", linestyle="--", label="Mean")
plt.axvline(result_median, color="green", linestyle="-",
label="Median")

plt.legend()

plt.show()
```

```
Mean: 8.537303106501248
Median: 9.605299308
```

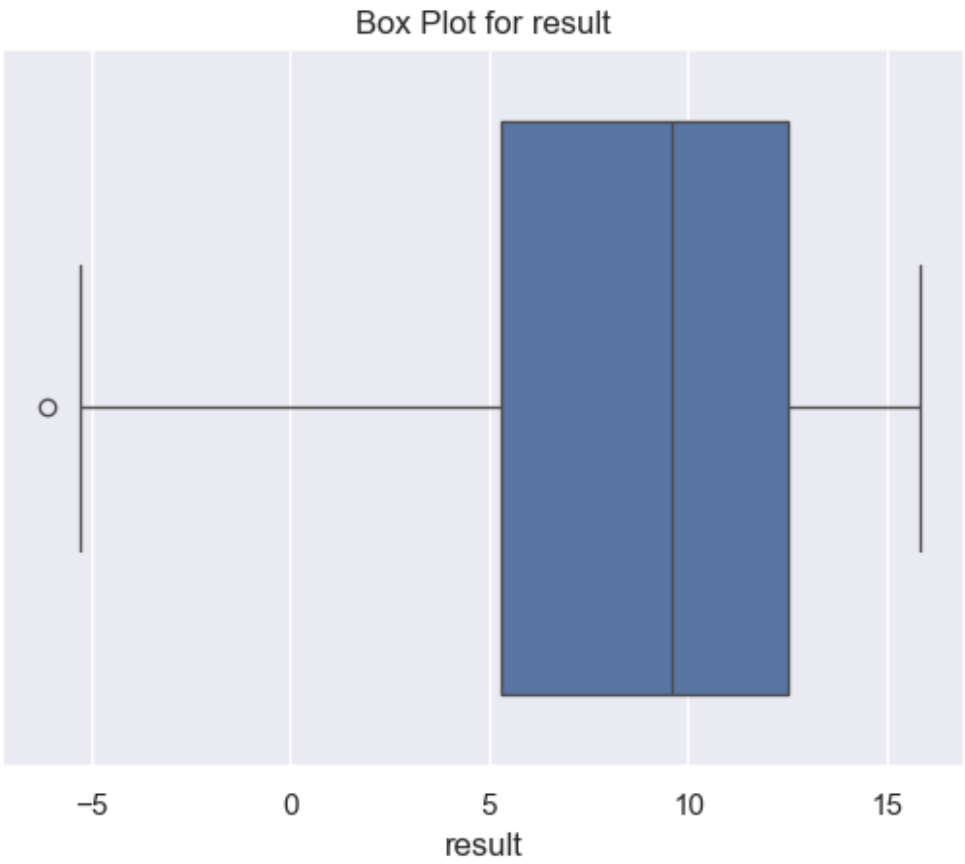


Box plots for identifying outliers in the numerical columns

```
# box plot
sns.boxplot(x=df["age"])
plt.title("Box Plot for Age")
plt.xlabel("Age")
plt.show()
```




```
# box plot
sns.boxplot(x=df["result"])
plt.title("Box Plot for result")
plt.xlabel("result")
plt.show()
```




```
# count the outliers using IQR method
Q1 = df["age"].quantile(0.25)
Q3 = df["age"].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
age_outliers = df[(df["age"] < lower_bound) | (df["age"] >
upper_bound)]

len(age_outliers)
```

```
39
```

```
# count the outliers using IQR method
Q1 = df["result"].quantile(0.25)
Q3 = df["result"].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
result_outliers = df[(df["result"] < lower_bound) | (df["result"] >
upper_bound)]
```

```
len(result_outliers)
```

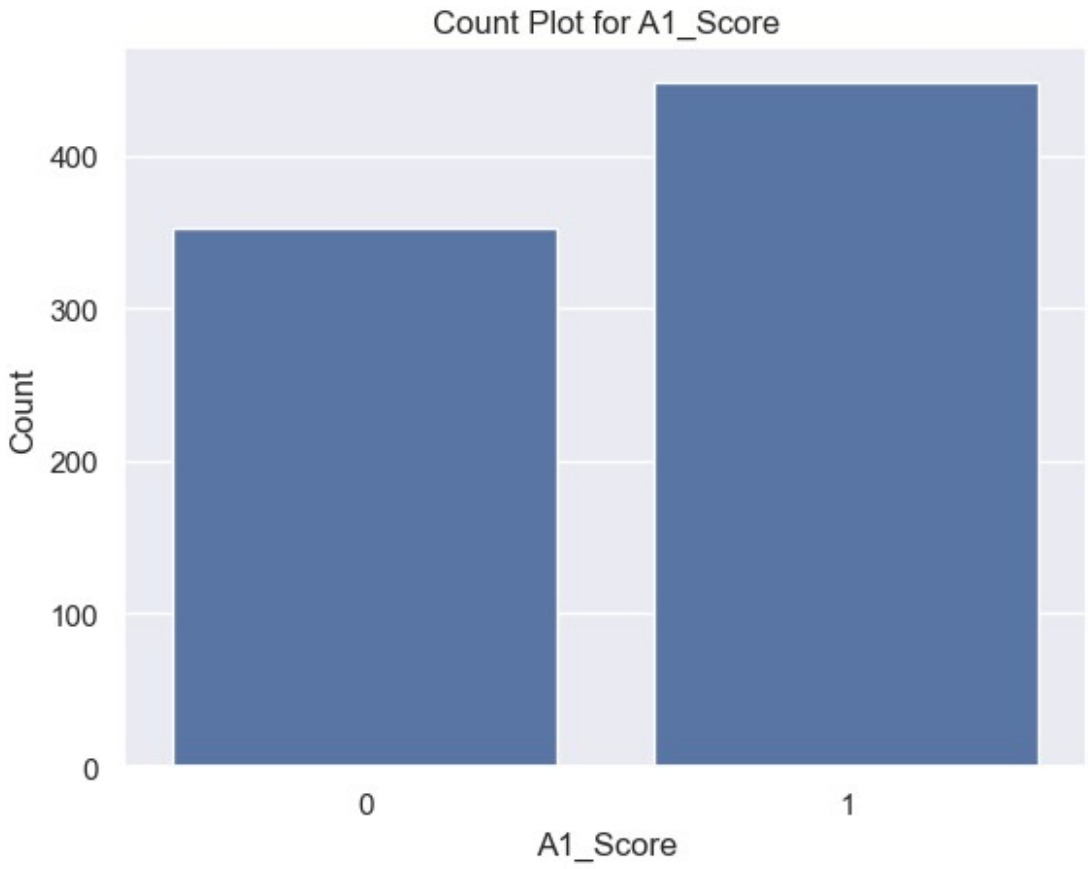
```
1
```

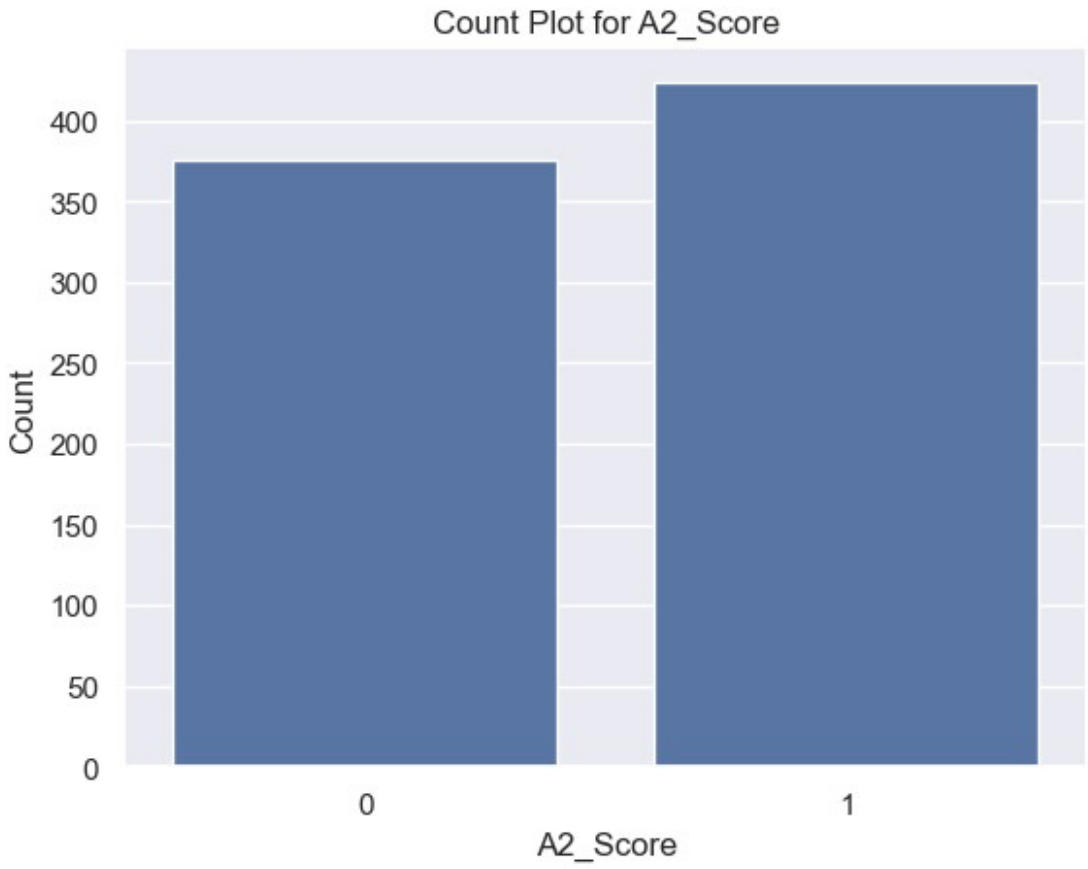
Univariate analysis of Categorical columns

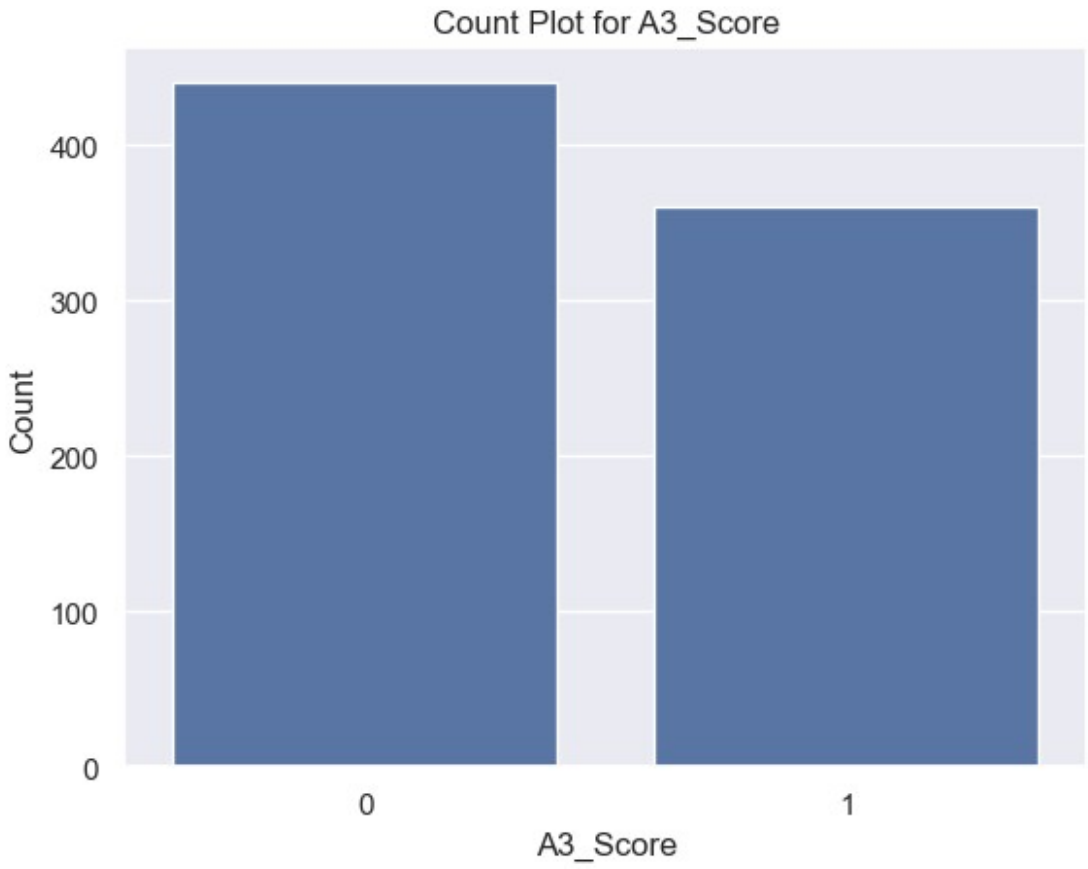
```
df.columns
```

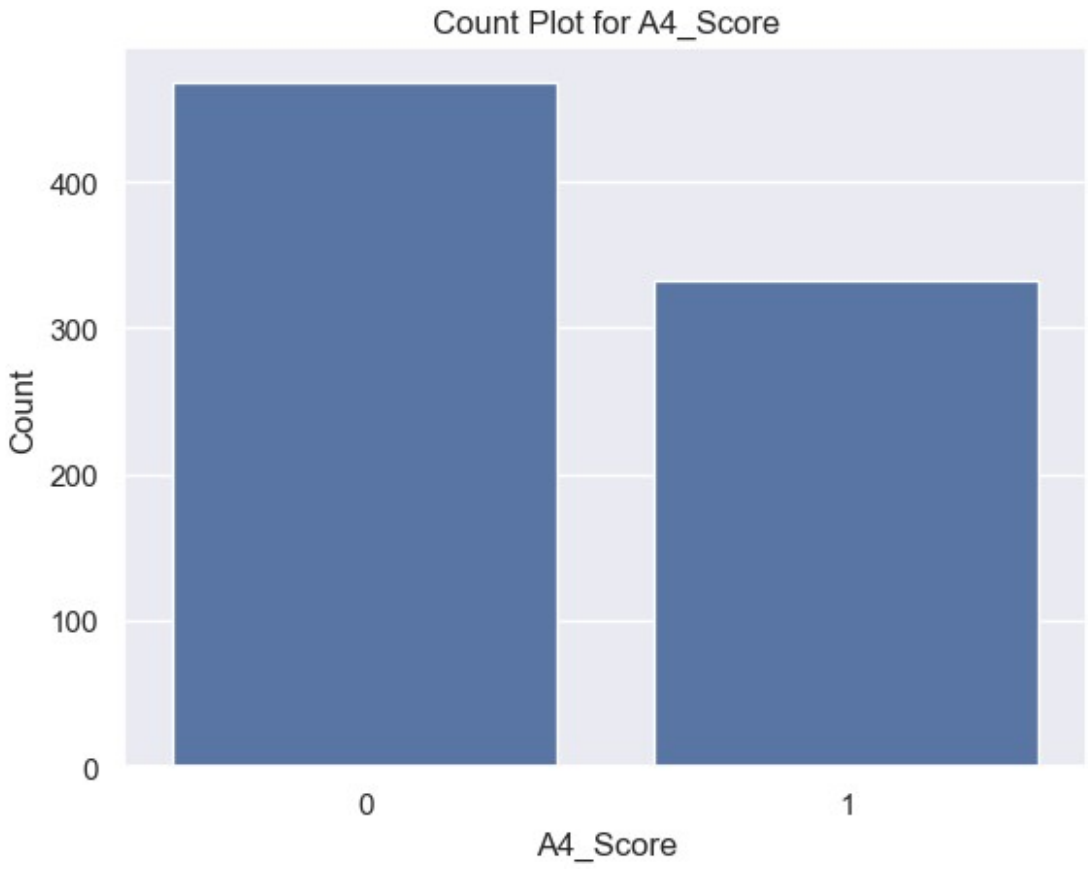
```
Index(['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score',
'A6_Score',
       'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'age',
'gender',
       'ethnicity', 'jaundice', 'austim', 'contry_of_res',
'used_app_before',
       'result', 'relation', 'Class/ASD'],
      dtype='object')
```

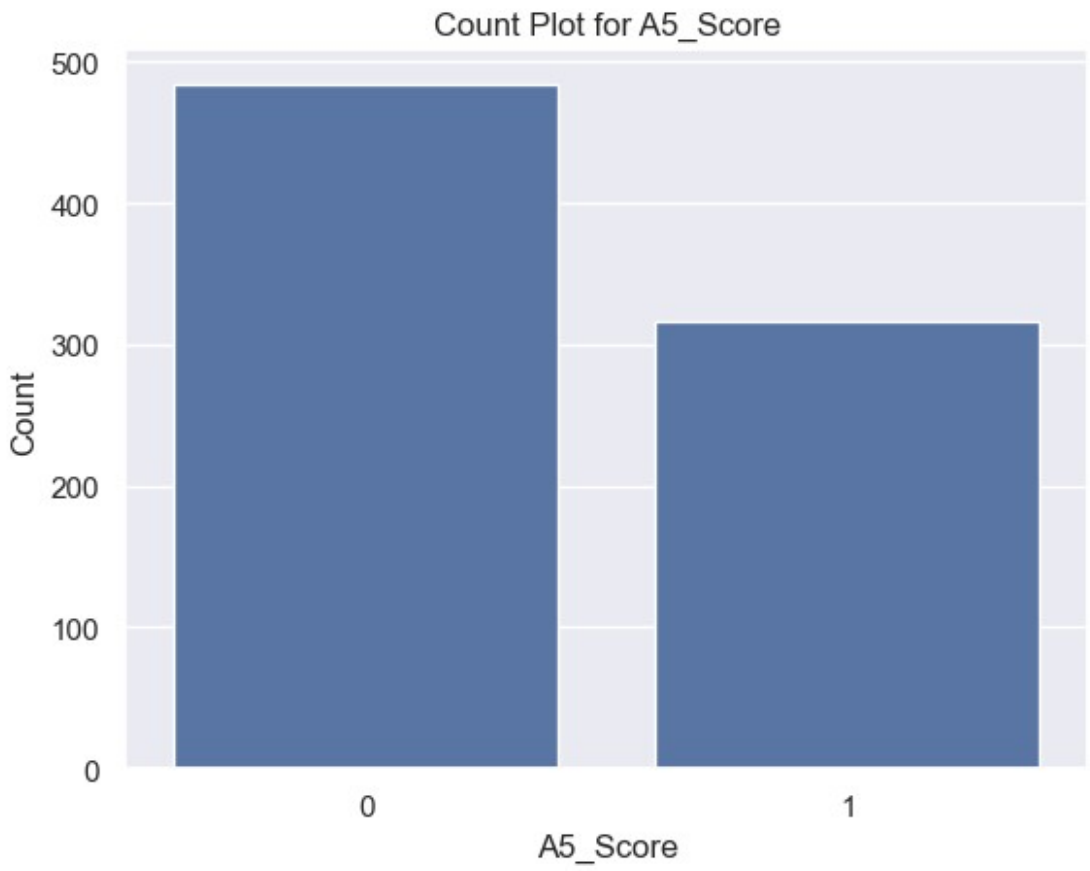
```
categorical_columns = ['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score',
'A5_Score', 'A6_Score',
       'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'gender',
       'ethnicity', 'jaundice', 'austim', 'contry_of_res',
'used_app_before',
       'relation']

for col in categorical_columns:
  sns.countplot(x=df[col])
  plt.title(f"Count Plot for {col}")
  plt.xlabel(col)
  plt.ylabel("Count")
  plt.show()
```

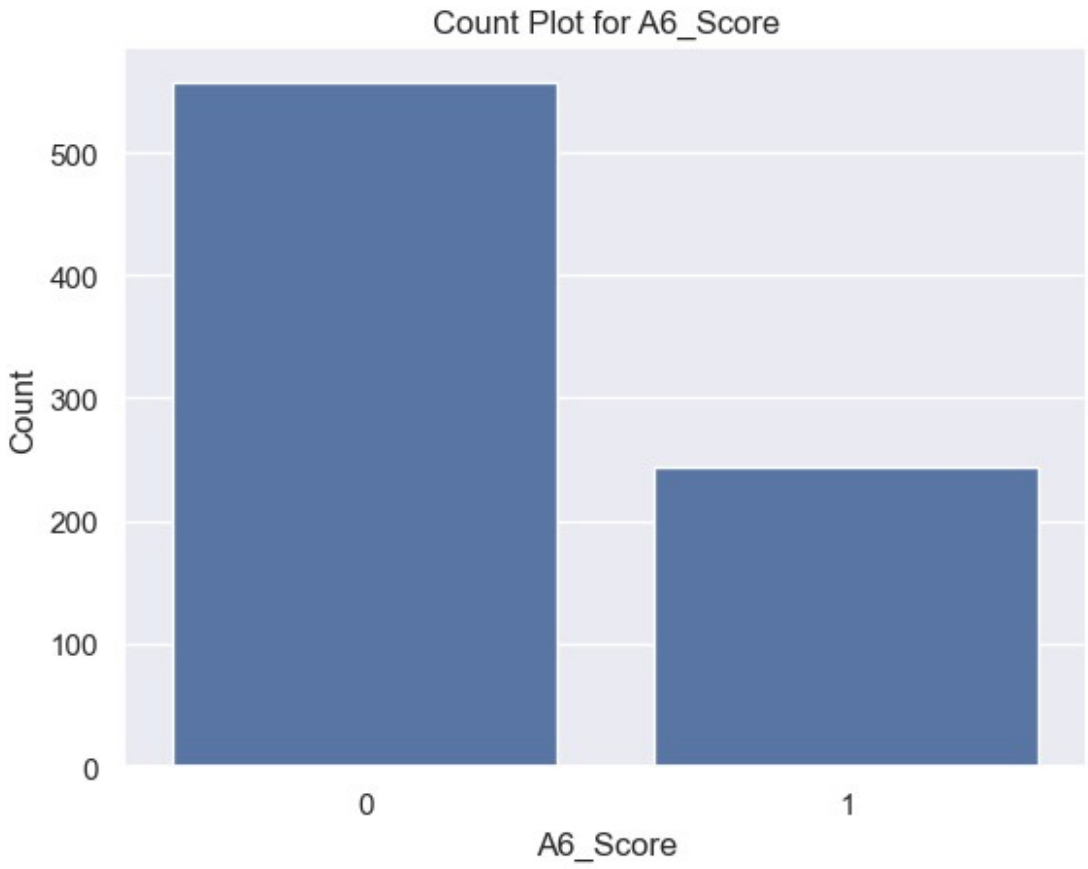

Count Plot for A1_Score
Count
400
300
200
100
0
0
1
A1_Score

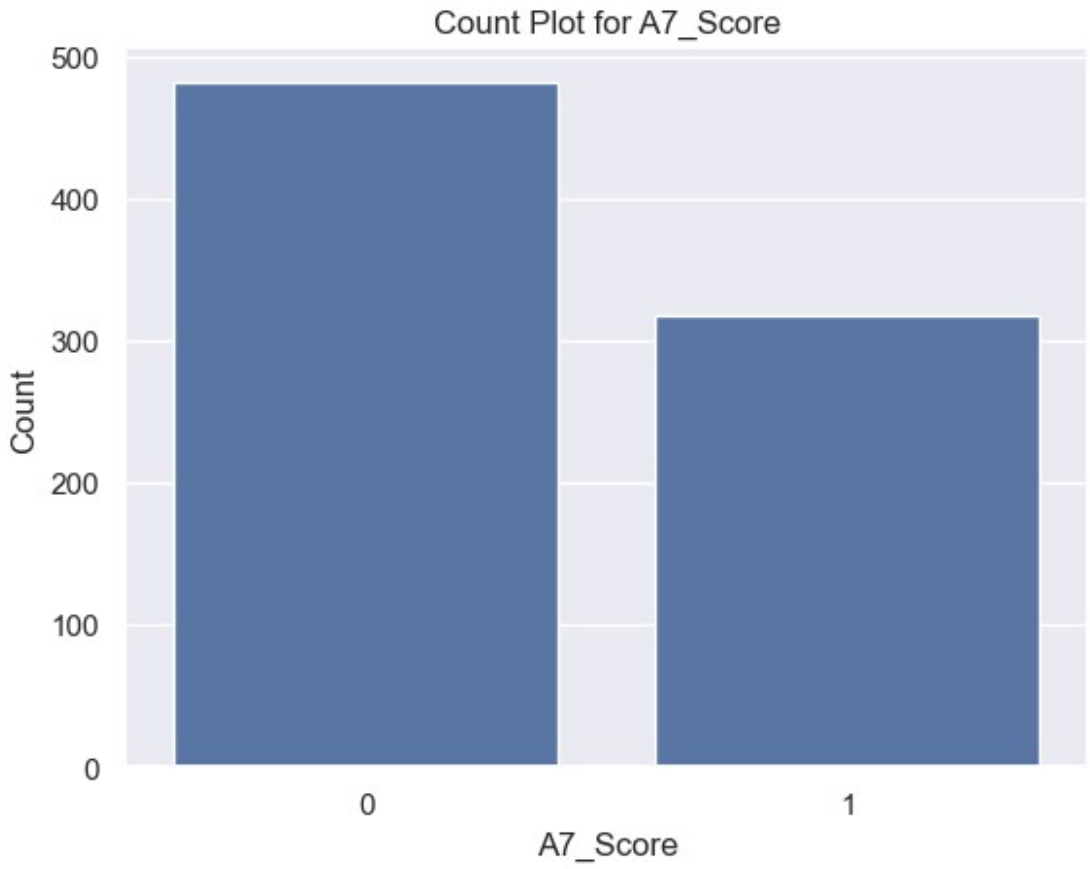

Count Plot for A2_Score
Count
400
350
300
250
200
150
100
50
0
0
1
A2_Score

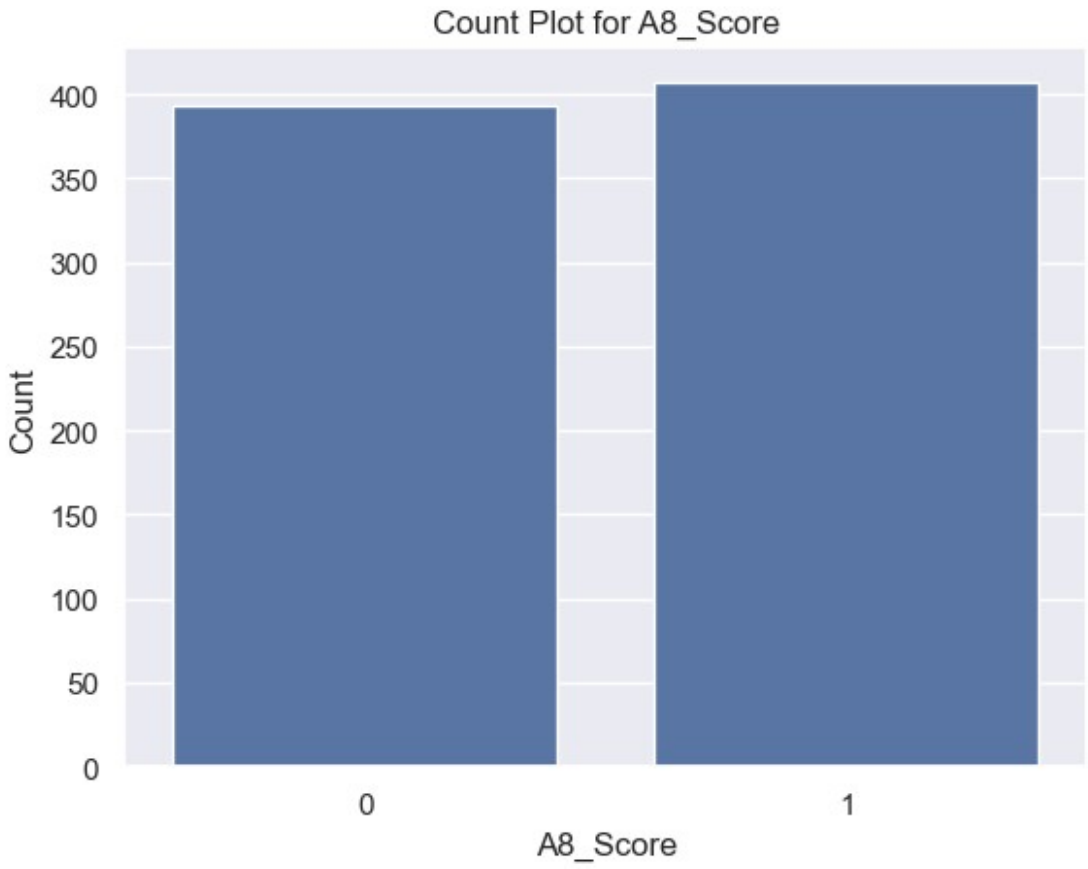

Count Plot for A3_Score
Count
400
300
200
100
0
0
1
A3_Score

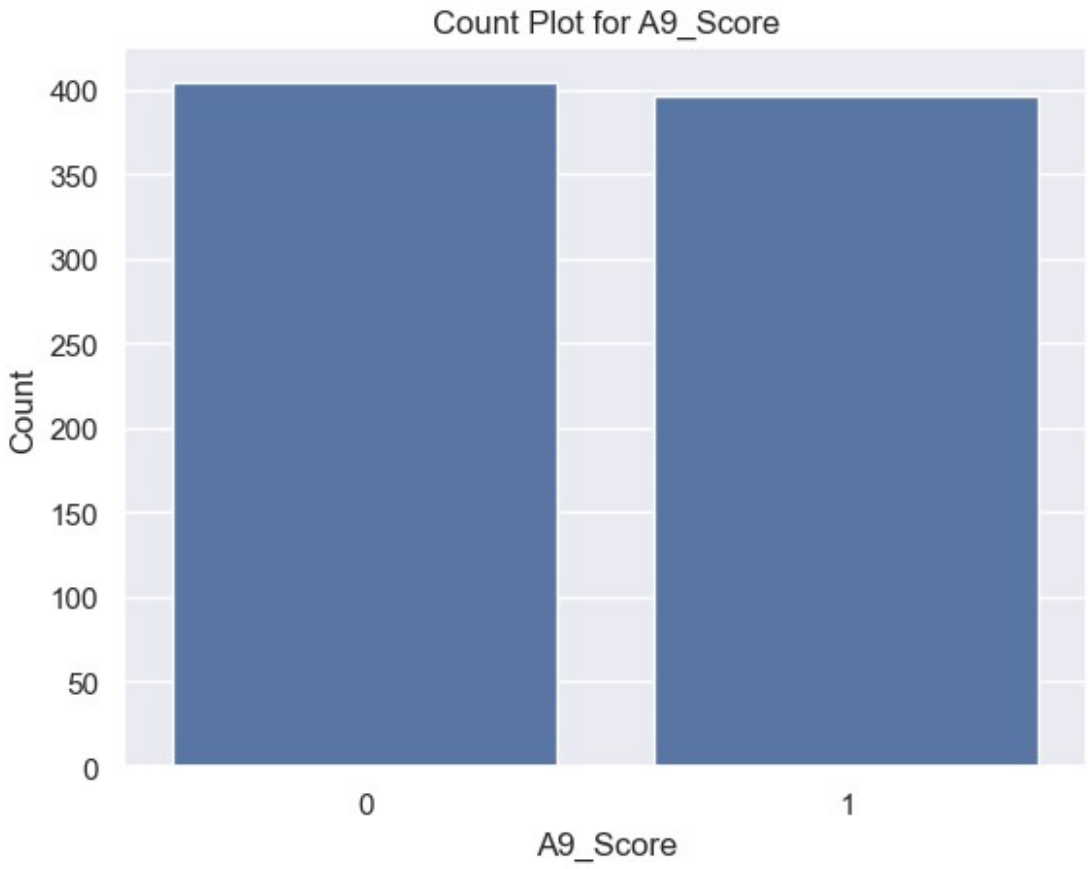

Count Plot for A4_Score
Count
400
300
200
100
0
0
1
A4_Score

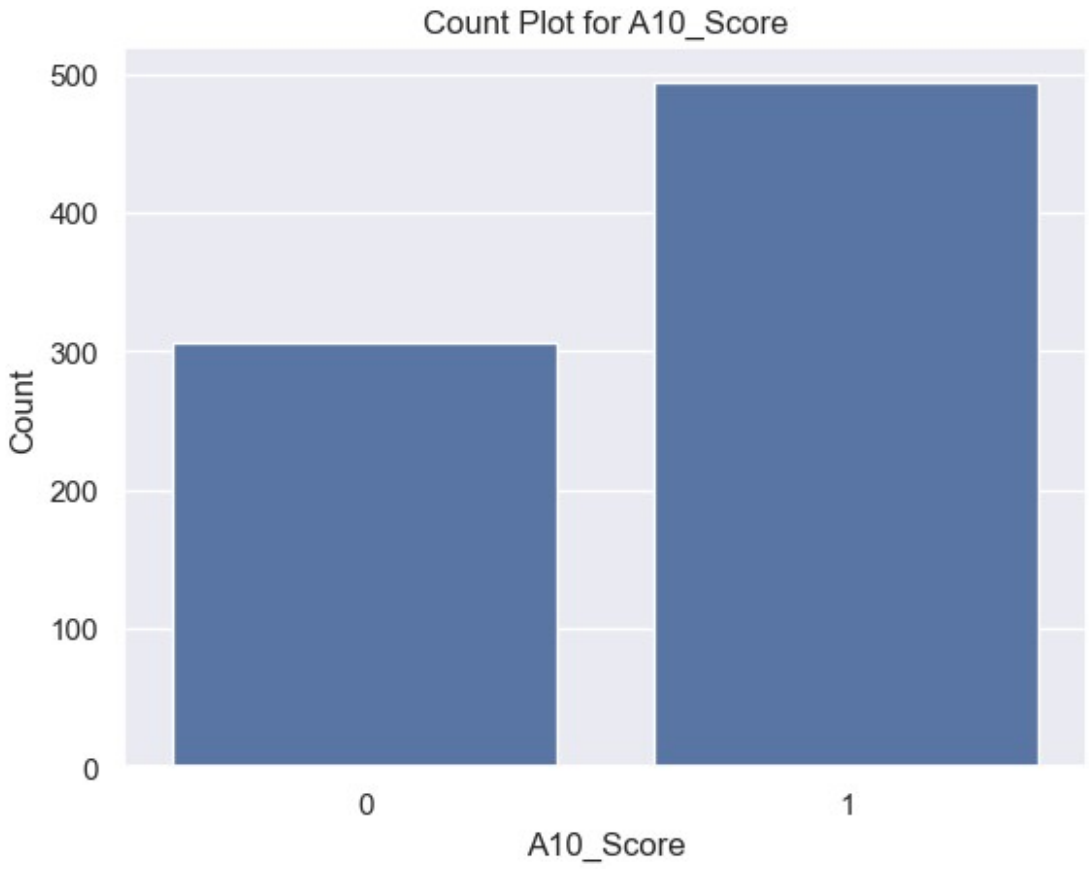

Count Plot for A5_Score
500
400
300
200
100
0
Count
0
1
A5_Score

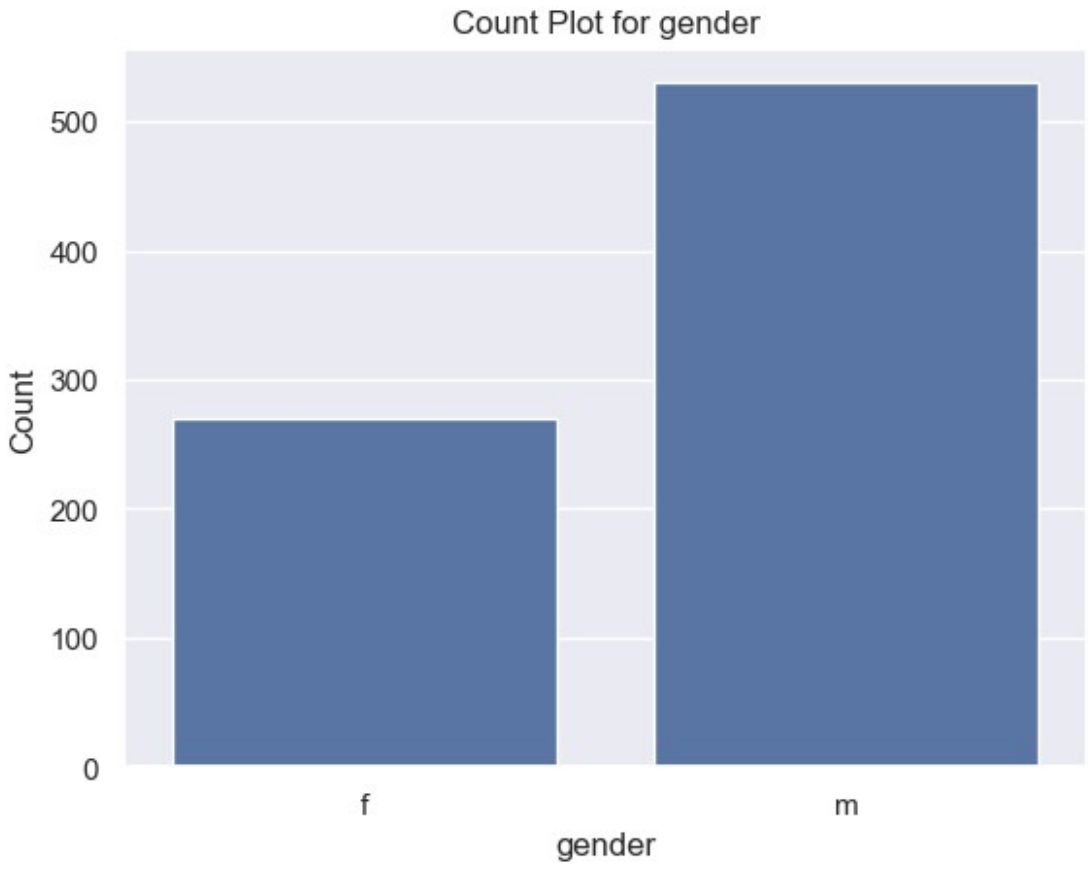

Count Plot for A6_Score
Count
500
400
300
200
100
0
0
1
A6_Score

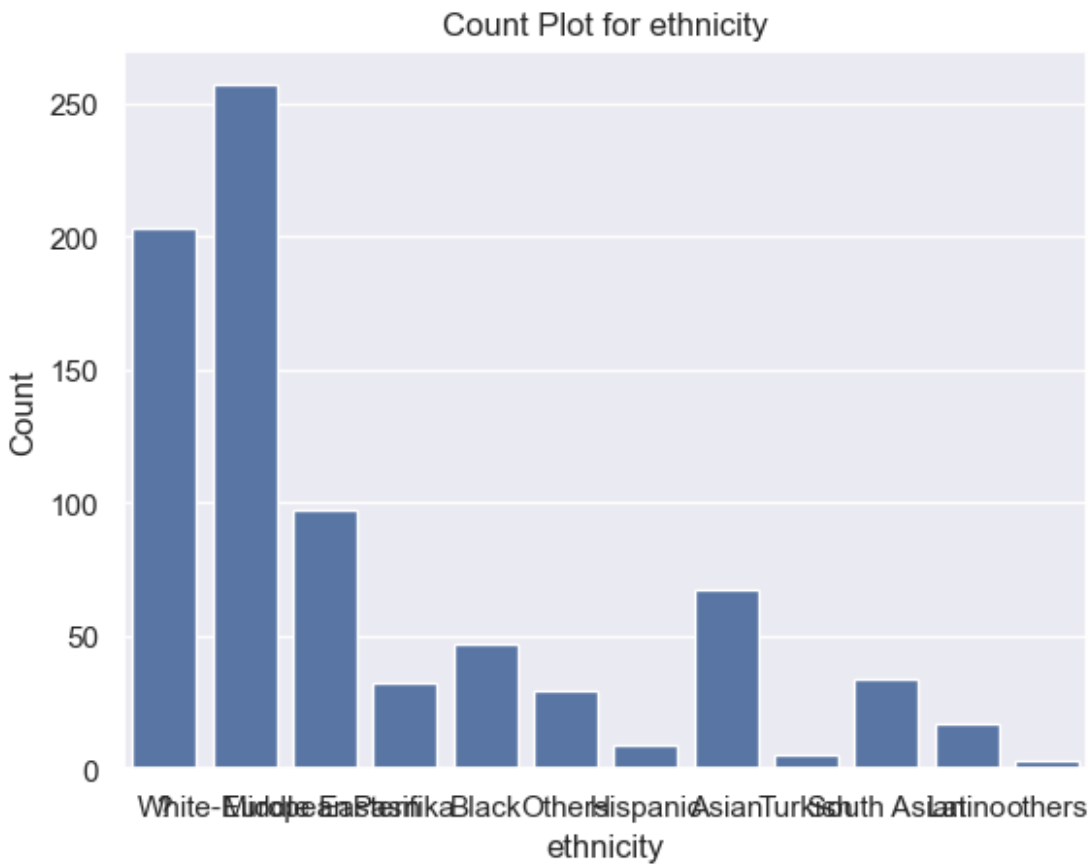

Count Plot for A7_Score
500
400
300
200
100
0
Count
0
1
A7_Score

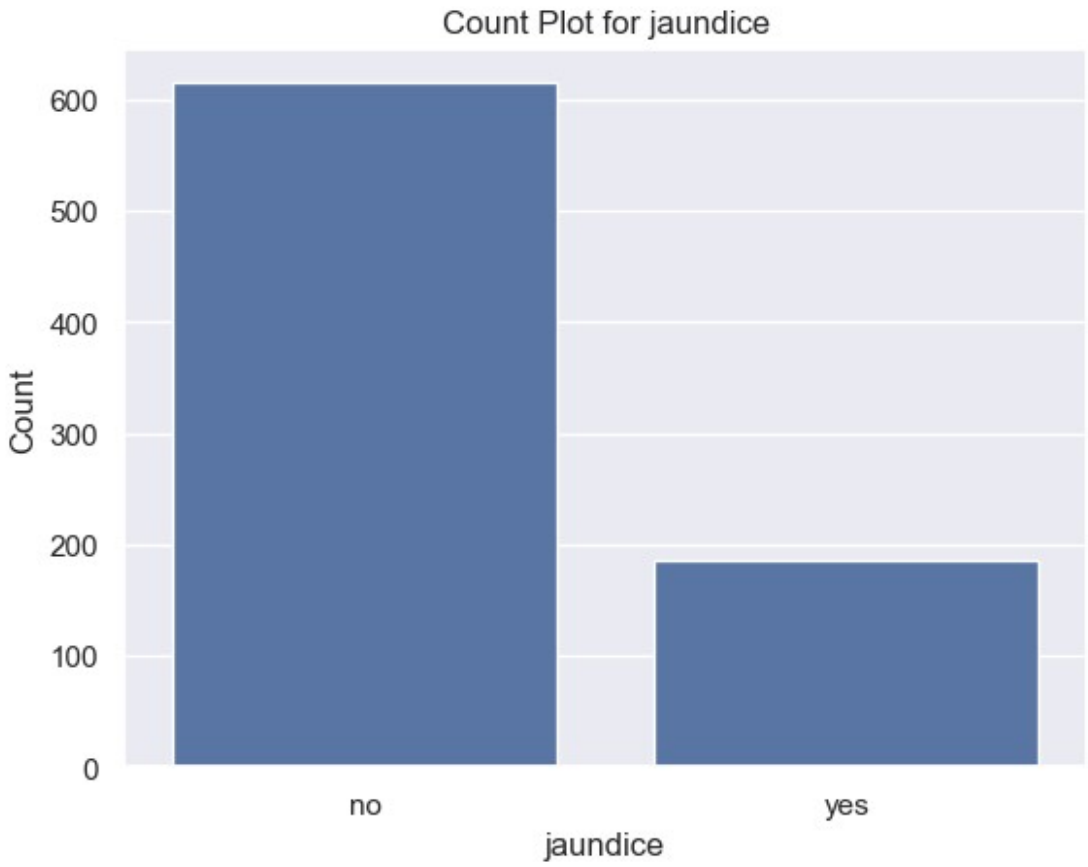

Count Plot for A8_Score
Count
400
350
300
250
200
150
100
50
0
0
1
A8_Score

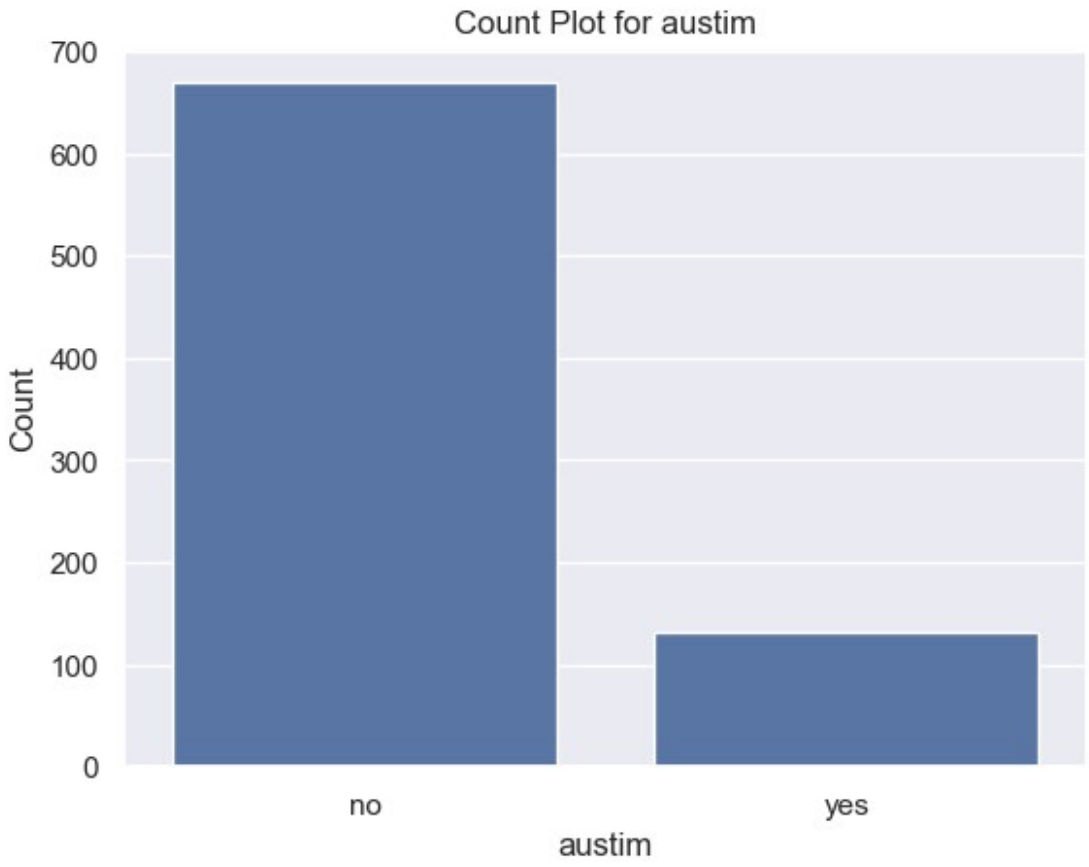

Count Plot for A9_Score
Count
400
350
300
250
200
150
100
50
0
0
1
A9_Score

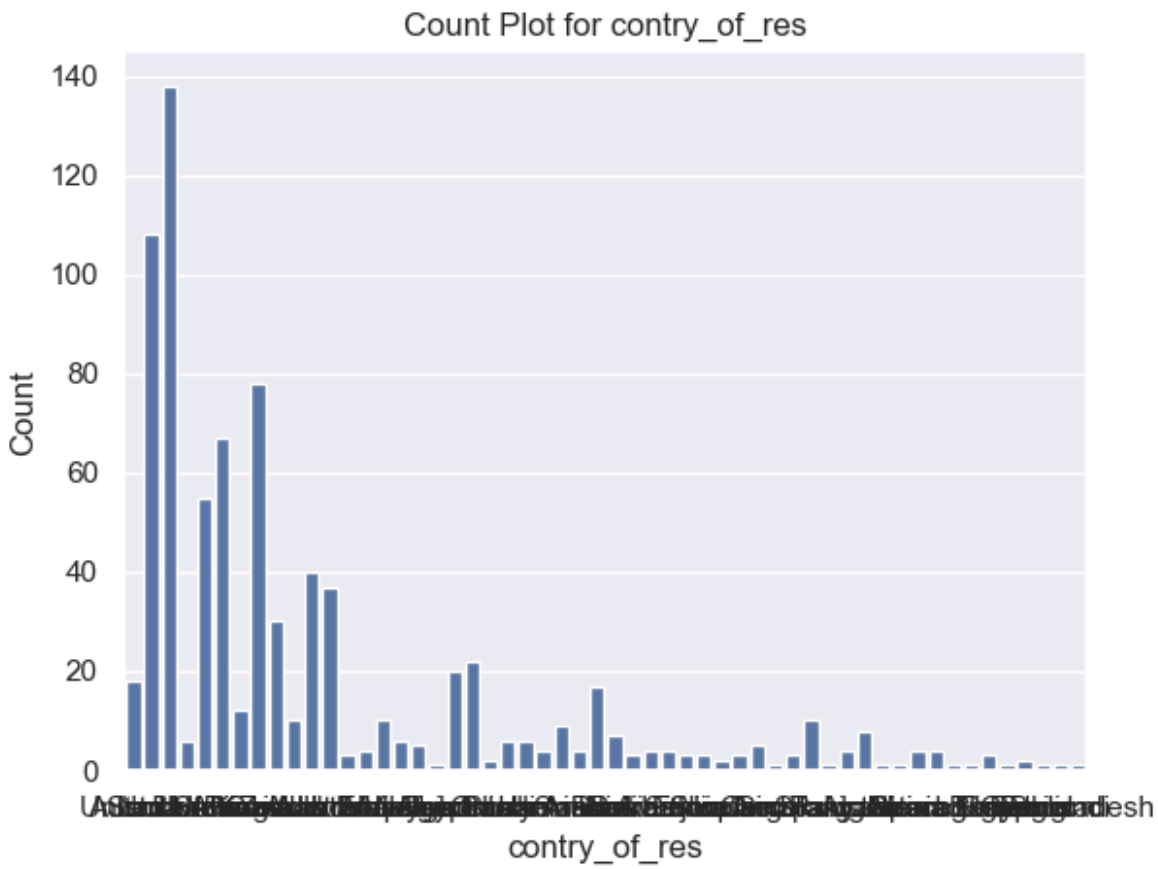

Count Plot for A10_Score
500
400
300
200
100
0
Count
0
1
A10_Score

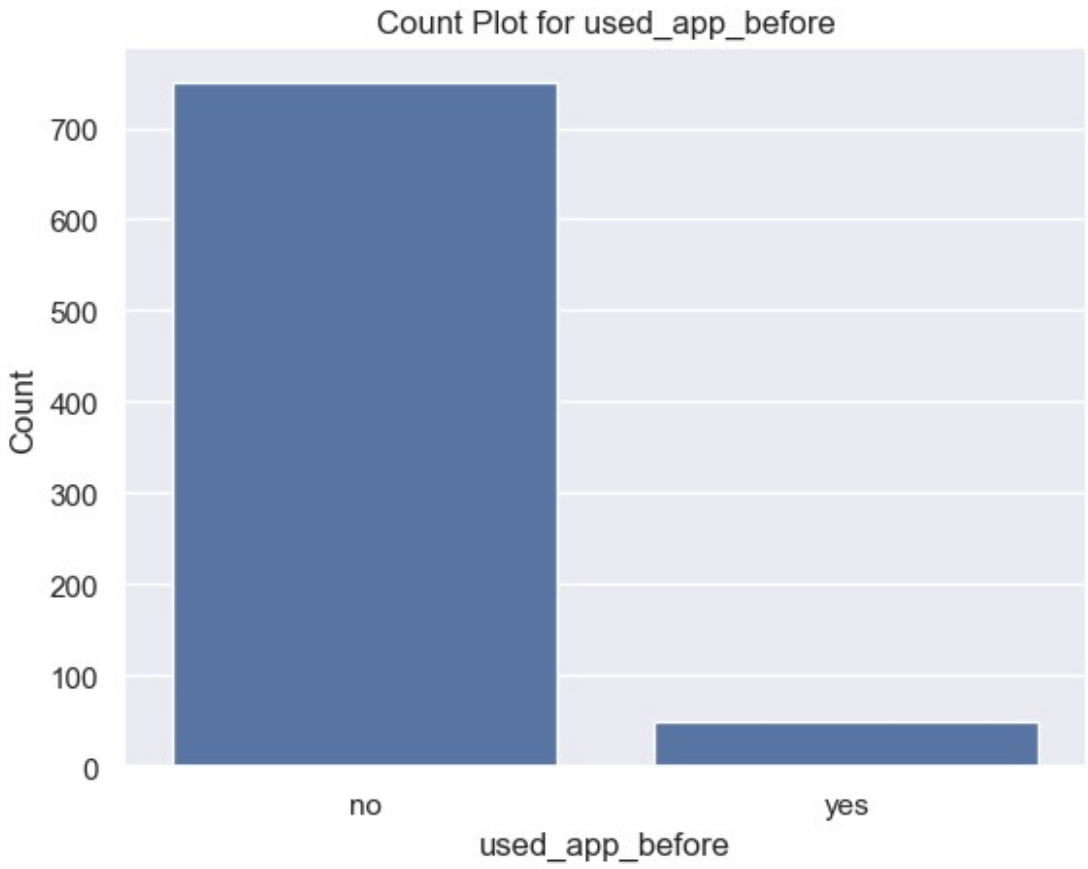

Count Plot for gender
500
400
300
200
100
0
Count
f
m
gender

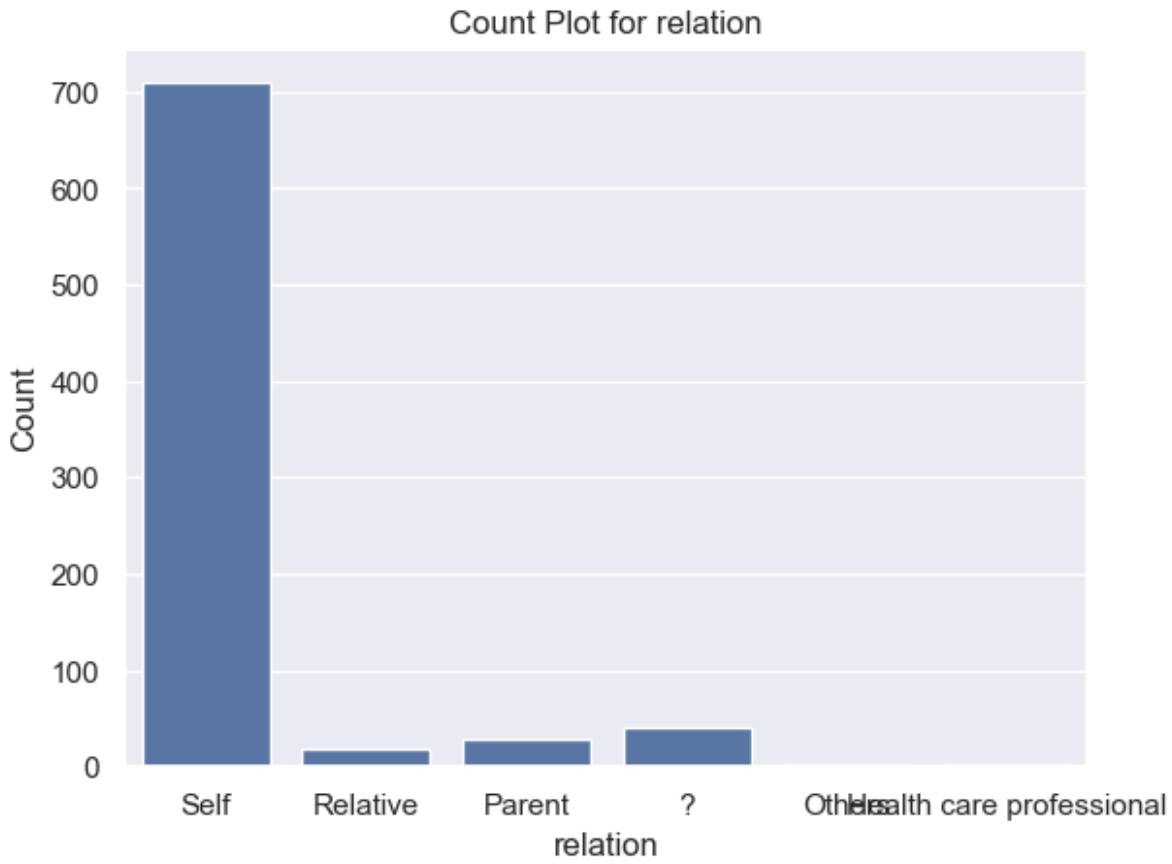

Count Plot for ethnicity
Count
250
200
150
100
50
0
White-
Black
Hispanic
Asian
Turkish
South Asian
Latino
others
ethnicity


Count Plot for jaundice
600
500
400
300
200
100
0
Count
no
yes
jaundice

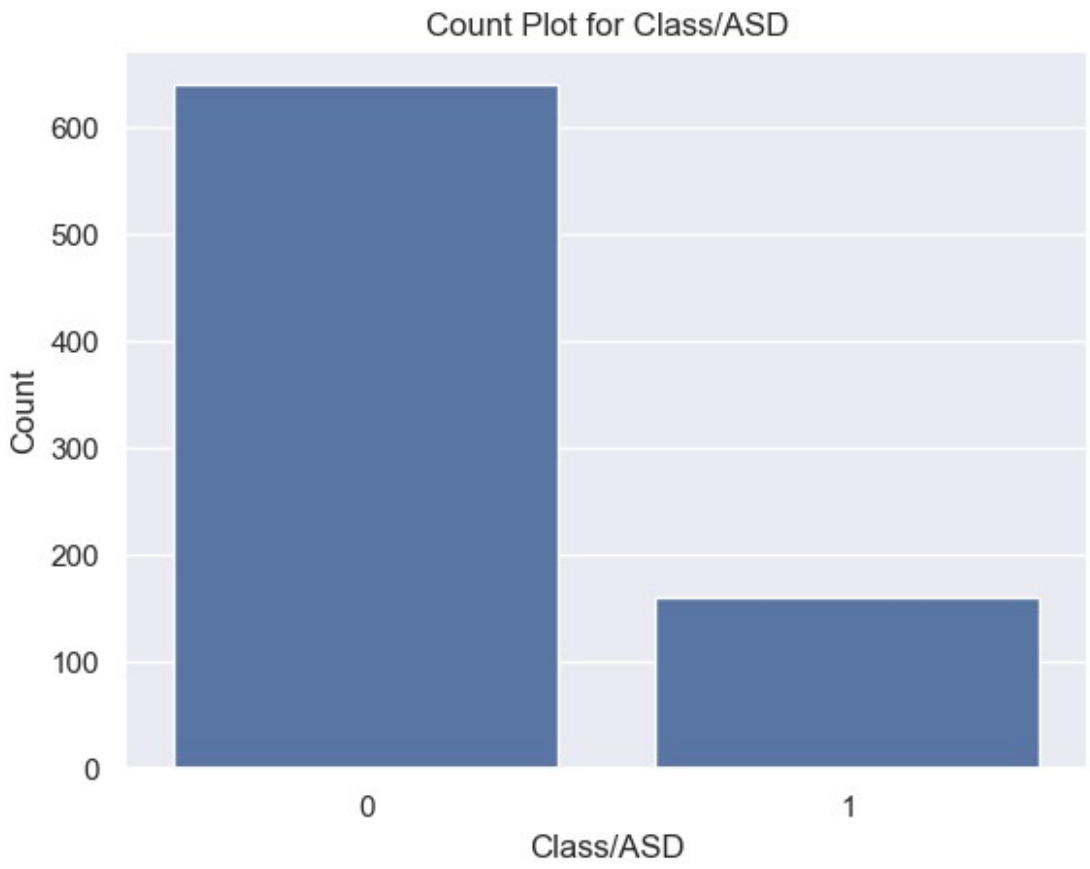

Count Plot for austim
700
600
500
400
300
200
100
0
Count
no
yes
austim


Count Plot for contry_of_res
140
120
100
80
60
40
20
0
Count
contry_of_res


Count Plot for used_app_before
Count
700
600
500
400
300
200
100
0
no
yes
used_app_before




```
# countplot for target column (Class/ASD)
sns.countplot(x=df["Class/ASD"])
plt.title("Count Plot for Class/ASD")
plt.xlabel("Class/ASD")
plt.ylabel("Count")
plt.show()
```




```
df["Class/ASD"].value_counts()
```

```
Class/ASD
0    639
1    161
Name: count, dtype: int64
```

handle missing values in ethnicity and relation column

```
df["ethnicity"] = df["ethnicity"].replace({"?": "Others", "others":
"Others"})

df["ethnicity"].unique()
```

```
array(['Others', 'White-European', 'Middle Eastern ', 'Pasifika',
'Black',
       'Hispanic', 'Asian', 'Turkish', 'South Asian', 'Latino'],
      dtype=object)
```

```
df["relation"].unique()
```

```
array(['Self', 'Relative', 'Parent', '?', 'Others',
       'Health care professional'], dtype=object)
```

```
df["relation"] = df["relation"].replace(
    {"?": "Others",
     "Relative": "Others",
     "Parent": "Others",
     "Health care professional": "Others"}
)
```

```
df["relation"].unique()
```

```
array(['Self', 'Others'], dtype=object)
```

```
df.head()
```

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

   A8_Score  A9_Score  A10_Score  age gender       ethnicity jaundice
austim  \
0         0         1          1   38      f          Others       no
no
1         0         0          0   47      m          Others       no
no
2         1         1          1    7      m  White-European       no
yes
3         0         0          0   23      f          Others       no
no
4         0         0          0   43      m          Others       no
no

   contry_of_res used_app_before     result relation  Class/ASD
0        Austria              no   6.351166     Self          0
1          India              no   2.255185     Self          0
2  United States              no  14.851484     Self          1
3  United States              no   2.276617     Self          0
4   South Africa              no  -4.777286     Self          0
```

Label Encoding

```
# identify columns with "object" data type
object_columns = df.select_dtypes(include=["object"]).columns

print(object_columns)
```

```
Index(['gender', 'ethnicity', 'jaundice', 'austim', 'contry_of_res',
       'used_app_before', 'relation'],
      dtype='object')
```

```
# initialize a dictionary to store the encoders
encoders = {}

# apply label encoding and store the encoders
for column in object_columns:
  label_encoder = LabelEncoder()
  df[column] = label_encoder.fit_transform(df[column])
  encoders[column] = label_encoder   # saving the encoder for this
column


# save the encoders as a pickle file
with open("encoders.pkl", "wb") as f:
  pickle.dump(encoders, f)

encoders
```

```
{'gender': LabelEncoder(),
 'ethnicity': LabelEncoder(),
 'jaundice': LabelEncoder(),
 'austim': LabelEncoder(),
 'contry_of_res': LabelEncoder(),
 'used_app_before': LabelEncoder(),
 'relation': LabelEncoder()}

df.head()
```

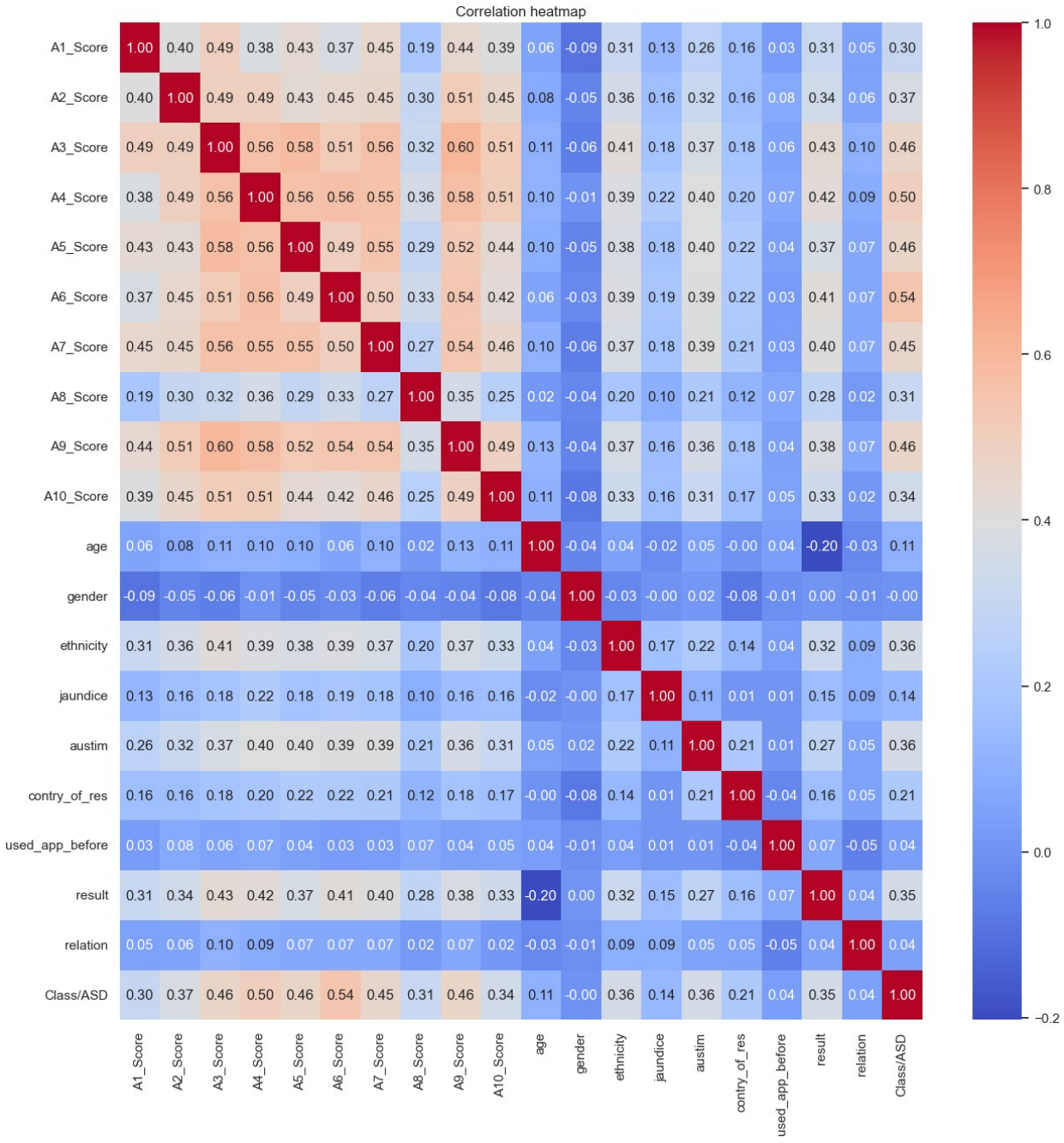
```
   A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score
A7_Score  \
0         1         0         1         0         1         0
1
1         0         0         0         0         0         0
0
2         1         1         1         1         1         1
1
3         0         0         0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

   A8_Score  A9_Score  A10_Score  age  gender  ethnicity  jaundice
```

```
austim  \
0          0          1           1   38        0           5          0
0
1          0          0           0   47        1           5          0
0
2          1          1           1    7        1           9          0
1
3          0          0           0   23        0           5          0
0
4          0          0           0   43        1           5          0
0
```

	contry_of_res	used_app_before	result	relation	Class/ASD
0	6	0	6.351166	1	0
1	23	0	2.255185	1	0
2	52	0	14.851484	1	1
3	52	0	2.276617	1	0
4	44	0	-4.777286	1	0

Bivariate Analysis

```
# correlation matrix
plt.figure(figsize=(15, 15))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation heatmap")
plt.show()
```



Insights from EDA:

Data preprocessing

Handling teh outliers

```
# function to replace the outliers with median
def replace_outliers_with_median(df, column):
  Q1 = df[column].quantile(0.25)
  Q3 = df[column].quantile(0.75)
  IQR = Q3 - Q1

  lower_bound = Q1 - 1.5 * IQR
  upper_bound = Q3 + 1.5 * IQR

  median = df[column].median()

  # replace outliers with median value
  df[column] = df[column].apply(lambda x: median if x < lower_bound or
x > upper_bound else x)

  return df

# replace outliers in the "age" column
df = replace_outliers_with_median(df, "age")

# replace outliers in the "result" column
df = replace_outliers_with_median(df, "result")

df.head()
```

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

   A8_Score  A9_Score  A10_Score   age  gender  ethnicity  jaundice
austim  \
0         0         1          1  38.0       0          5         0
0
1         0         0          0  47.0       1          5         0
0
2         1         1          1   7.0       1          9         0
1
3         0         0          0  23.0       0          5         0
```

```
0
4         0         0          0  43.0       1          5         0
0

   contry_of_res  used_app_before     result  relation  Class/ASD
0              6                0   6.351166         1          0
1             23                0   2.255185         1          0
2             52                0  14.851484         1          1
3             52                0   2.276617         1          0
4             44                0  -4.777286         1          0
```

```
df.shape
```

```
(800, 20)
```

Train Test Split

```
df.columns
```

```
Index(['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score',
'A6_Score',
       'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'age',
'gender',
       'ethnicity', 'jaundice', 'austim', 'contry_of_res',
'used_app_before',
       'result', 'relation', 'Class/ASD'],
      dtype='object')
```

```
X = df.drop(columns=["Class/ASD"])
y = df["Class/ASD"]
```

```
print(X)
```

```
     A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score
A7_Score  \
0           1         0         1         0         1         0
1
1           0         0         0         0         0         0
0
2           1         1         1         1         1         1
1
3           0         0         0         0         0         0
0
4           0         0         0         0         0         0
0
..        ...       ...       ...       ...       ...       ...
...
795         0         1         0         0         0         0
0
796         0         1         1         0         0         1
0
```

```
797          0          0          0          0          0          0
0
798          0          0          0          0          0          0
0
799          0          1          0          0          0          0
0

     A8_Score  A9_Score  A10_Score   age  gender  ethnicity  jaundice
austim  \
0           0          1           1  38.0       0          5          0
0
1           0          0           0  47.0       1          5          0
0
2           1          1           1   7.0       1          9          0
1
3           0          0           0  23.0       0          5          0
0
4           0          0           0  43.0       1          5          0
0
..        ...        ...         ...   ...     ...        ...        ...
...
795         0          1           1  16.0       1          2          0
0
796         1          1           1  20.0       1          9          0
0
797         0          0           0   5.0       1          7          1
0
798         0          0           0  16.0       0          5          0
0
799         0          0           0  46.0       0          5          0
0

     contry_of_res  used_app_before     result  relation
0                6                0   6.351166         1
1               23                0   2.255185         1
2               52                0  14.851484         1
3               52                0   2.276617         1
4               44                0  -4.777286         1
..             ...              ...        ...       ...
795             34                0  12.999501         1
796             16                0  13.561518         1
797             34                0   2.653177         1
798             14                0   9.069342         1
799             50                1   2.243304         1

[800 rows x 19 columns]
```

```
print(y)
```

```
0      0
1      0
2      1
3      0
4      0
      ..
795    0
796    0
797    0
798    0
799    0
Name: Class/ASD, Length: 800, dtype: int64
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

print(y_train.shape)
print(y_test.shape)
```

```
(640,)
(160,)
```

```
y_train.value_counts()
```

```
Class/ASD
0    515
1    125
Name: count, dtype: int64
```

```
y_test.value_counts()
```

```
Class/ASD
0    124
1     36
Name: count, dtype: int64
```

SMOTE (Synthetic Minority Oversampling technique)

```
smote = SMOTE(random_state=42)

X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

print(y_train_smote.shape)
```

```
(1030,)
```

```
print(y_train_smote.value_counts())
```

```
Class/ASD
1    515
```

```
0    515
Name: count, dtype: int64
```

Model Training

```
# dictionary of classifiers
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42)
}

# dictionary to store the cross validation results
cv_scores = {}

# perform 5-fold cross validation for each model
for model_name, model in models.items():
  print(f"Training {model_name} with default parameters...")
  scores = cross_val_score(model, X_train_smote, y_train_smote, cv=5,
scoring="accuracy")
  cv_scores[model_name] = scores
  print(f"{model_name} Cross-Validation Accuracy:
{np.mean(scores):.2f}")
  print("-"*50)
```

```
Training Decision Tree with default parameters...
Decision Tree Cross-Validation Accuracy: 0.86
--------------------------------------------------
Training Random Forest with default parameters...
Random Forest Cross-Validation Accuracy: 0.92
--------------------------------------------------
Training XGBoost with default parameters...
XGBoost Cross-Validation Accuracy: 0.90
--------------------------------------------------
```

```
cv_scores
```

```
{'Decision Tree': array([0.7961165 , 0.87864078, 0.87378641, 0.8592233
, 0.87378641]),
 'Random Forest': array([0.90776699, 0.92718447, 0.9223301 ,
0.91747573, 0.9223301 ]),
 'XGBoost': array([0.87378641, 0.9223301 , 0.89320388, 0.91262136,
0.91747573])}
```

Model Selection & Hyperparameter Tuning

```
# Initializing models
decision_tree = DecisionTreeClassifier(random_state=42)
random_forest = RandomForestClassifier(random_state=42)
xgboost_classifier = XGBClassifier(random_state=42)

# Hyperparameter grids for RandomizedSearchCV

param_grid_dt = {
    "criterion": ["gini", "entropy"],
    "max_depth": [None, 10, 20, 30, 50, 70],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}


param_grid_rf = {
    "n_estimators": [50, 100, 200, 500],
    "max_depth": [None, 10, 20, 30],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "bootstrap": [True, False]
}


param_grid_xgb = {
    "n_estimators": [50, 100, 200, 500],
    "max_depth": [3, 5, 7, 10],
    "learning_rate": [0.01, 0.1, 0.2, 0.3],
    "subsample": [0.5, 0.7, 1.0],
    "colsample_bytree": [0.5, 0.7, 1.0]
}



# hyperparameter tunig for 3 tree based models

# perform RandomizedSearchCV for each model
random_search_dt = RandomizedSearchCV(estimator=decision_tree,
param_distributions=param_grid_dt, n_iter=20, cv=5,
scoring="accuracy", random_state=42)
random_search_rf = RandomizedSearchCV(estimator=random_forest,
param_distributions=param_grid_rf, n_iter=20, cv=5,
scoring="accuracy", random_state=42)
random_search_xgb = RandomizedSearchCV(estimator=xgboost_classifier,
param_distributions=param_grid_xgb, n_iter=20, cv=5,
scoring="accuracy", random_state=42)

# fit the models
random_search_dt.fit(X_train_smote, y_train_smote)
```

```
random_search_rf.fit(X_train_smote, y_train_smote)
random_search_xgb.fit(X_train_smote, y_train_smote)

RandomizedSearchCV(cv=5,
                   estimator=XGBClassifier(base_score=None,
booster=None,
                                           callbacks=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
device=None,
                                           early_stopping_rounds=None,
                                           enable_categorical=False,
                                           eval_metric=None,
feature_types=None,
                                           gamma=None,
grow_policy=None,
                                           importance_type=None,

interaction_constraints=None,
                                           learning_rate...
                                           min_child_weight=None,
missing=nan,
                                           monotone_constraints=None,
                                           multi_strategy=None,
                                           n_estimators=None,
n_jobs=None,
                                           num_parallel_tree=None,
                                           random_state=42, ...),
                   n_iter=20,
                   param_distributions={'colsample_bytree': [0.5, 0.7,
1.0],
                                        'learning_rate': [0.01, 0.1,
0.2, 0.3],
                                        'max_depth': [3, 5, 7, 10],
                                        'n_estimators': [50, 100, 200,
500],
                                        'subsample': [0.5, 0.7, 1.0]},
                   random_state=42, scoring='accuracy')


# Get the model with best score

best_model = None
best_score = 0

if random_search_dt.best_score_ > best_score:
  best_model = random_search_dt.best_estimator_
  best_score = random_search_dt.best_score_
```

```
if random_search_rf.best_score_ > best_score:
  best_model = random_search_rf.best_estimator_
  best_score = random_search_rf.best_score_

if random_search_xgb.best_score_ > best_score:
  best_model = random_search_xgb.best_estimator_
  best_score = random_search_xgb.best_score_


print(f"Best Model: {best_model}")
print(f"Best Cross-Validation Accuracy: {best_score:.2f}")
```

```
Best Model: RandomForestClassifier(bootstrap=False, max_depth=20,
n_estimators=50,
                       random_state=42)
Best Cross-Validation Accuracy: 0.93
```

```
# save the best model
with open("best_model.pkl", "wb") as f:
  pickle.dump(best_model, f)
```

Evaluation

```
# evaluate on test data
y_test_pred = best_model.predict(X_test)
print("Accuracy score:\n", accuracy_score(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Classification Report:\n", classification_report(y_test,
y_test_pred))
```

```
Accuracy score:
 0.81875
Confusion Matrix:
 [[108  16]
 [ 13  23]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.87      0.88       124
           1       0.59      0.64      0.61        36

    accuracy                           0.82       160
   macro avg       0.74      0.75      0.75       160
weighted avg       0.82      0.82      0.82       160
```