# AUTISM PREDICTION USING MACHINE LEARNING

# SOURCE CODE:

1. **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 importaccuracy\_score, confusion\_matrix, classification\_rImporting the dependencies

eport

import pickle

**2.Data Loading & Understanding**

# read the csv data to a pandas dataframe

df = pd.read\_csv("/content/train.csv")

**Initial Inspection**

df.shape

df.tail()

# display all columns of a dataframe

pd.set\_option('display.max\_columns', None)

df.info()

# convert age column datatype to integer

df["age"] = df["age"].astype(int)

df.head(2)

# dropping ID & age\_desc column

df = df.drop(columns=["ID", "age\_desc"])

df.shape

df.head(2)

df.columns

df["contry\_of\_res"].unique()

# define the mapping dictionary for country names

mapping ={"Viet Nam": "Vietnam","AmericanSamoa": "United States","Hong Kong": "China"

}

# repalce value i    n the country column

df["contry\_of\_res"] = df["contry\_of\_res"].replace(mapping)

df["contry\_of\_res"].unique()

# taget class distribution

df["Class/ASD"].value\_counts()

**3.Exploratory Data Analysis (EDA)**

df.shape

df.columns

df.head(2)

df.describe()

**Univariate Analysis**

**Numerical Columns:**

**age result**

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)

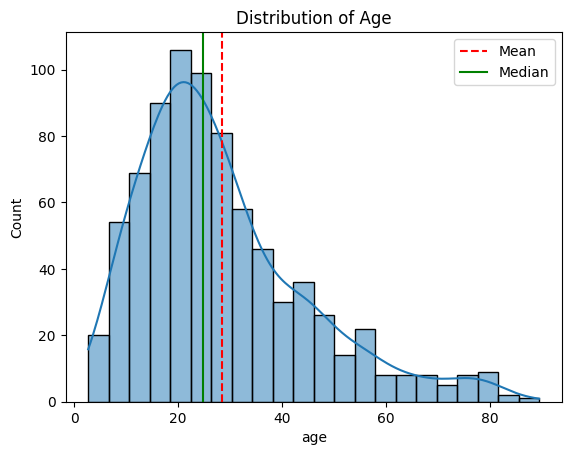
# Add vertical lines for mean and median

plt.axvline(age\_mean, color="red", linestyle="--", label="Mean")

plt.axvline(age\_median, color="green", linestyle="-", label="Median")

plt. legend()

plt.show()



# 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()

**Box plots for identifying outliers in the numerical columns**

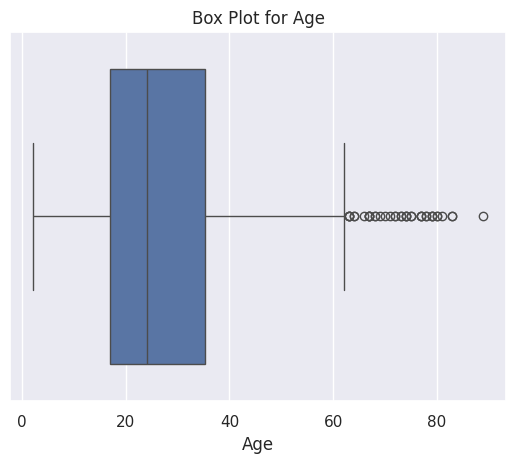
# box plot

sns.boxplot(x=df["age"])

plt.title("Box Plot for Age")

plt.xlabel("Age")

plt.show()



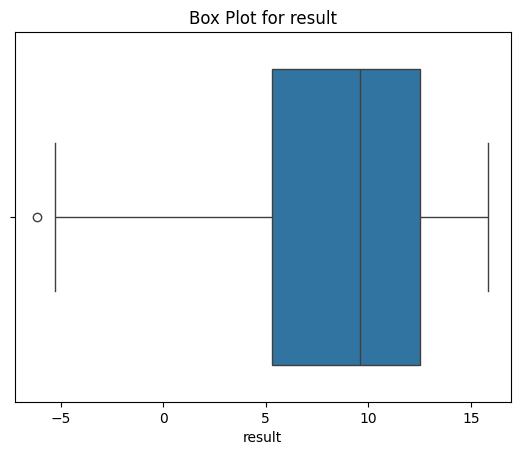
# box plot

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)

# 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)

**Univariate analysis of Categorical columns**

df.columns

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()

# countplot for target column (Class/ASD)

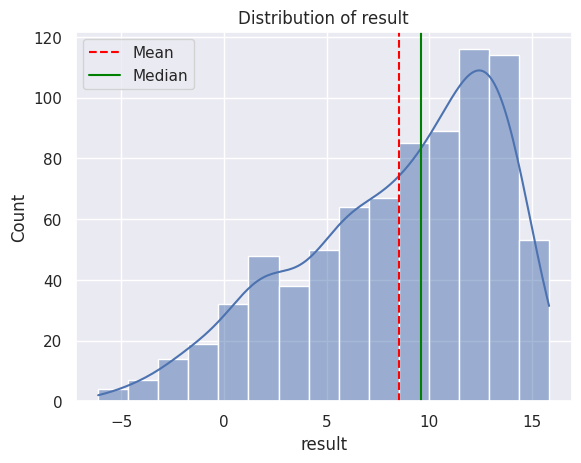
sns.countplot(x=df["Class/ASD"])

plt.title("Count Plot for Class/ASD")

plt.xlabel("Class/ASD")

plt.ylabel("Count")

plt.show()



df["Class/ASD"].value\_counts()

**handle missing values in ethnicity and relation column**

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

df["ethnicity"].unique()

df["relation"].unique()

df["relation"].unique()

df.head()

**Label Encoding**

# identify columns with "object" data type

object\_columns = df.select\_dtypes(include=["object"]).columns

print(object\_columns)

# 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

df.head()

**Bivariate Analysis**

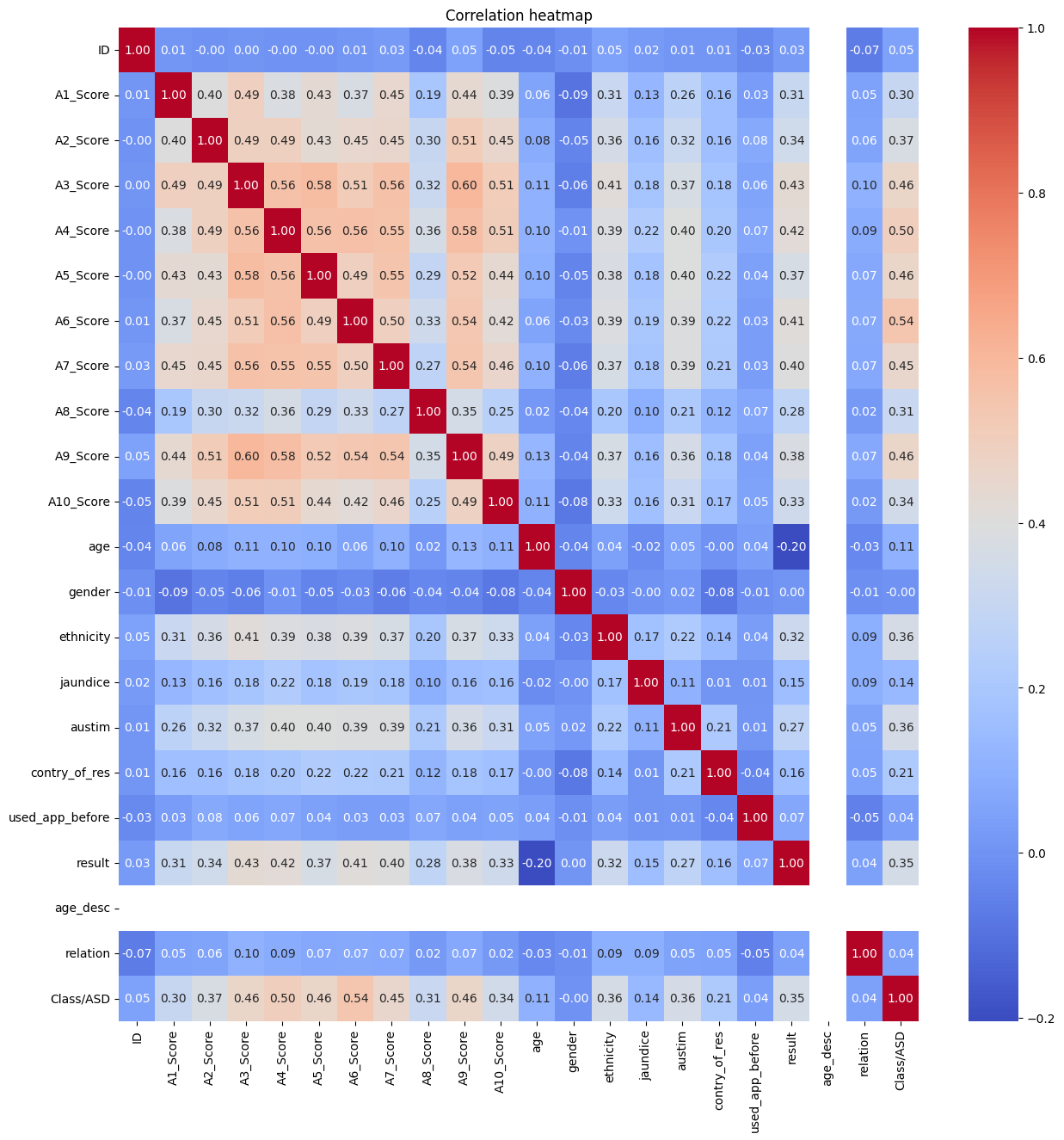
# correlation matrix

plt.figure(figsize=(15, 15))

sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")

plt.title("Correlation heatmap")

plt.show()



**4.Data preprocessing**

# 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")

# re# evaluate on test data

**Train Test Split**

df.columns

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

y = df["Class/ASD"]

print(X)

print(y)

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)

y\_train.value\_counts()

y\_test.value\_counts()

**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)

print(y\_train\_smote.shape)

**5. 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)

cv\_scores

1. **Model Selection & Hyperparameter Tuning**

cv\_scores

# 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

# the below steps can be automated by using a for loop or by using a pipeline

# 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\_i

# 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)

# 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}")

# save the best model

with open("best\_model.pkl", "wb") as f:

pickle.dump(best\_model, f)

**7.Evaluation**

# evaluate on test data

y\_test\_pred = best\_model.predict(X\_test)

("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))

