```
#uploading the data.csv using pandas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
file_path = 'data.csv'
df = pd.read_csv(file_path)
```

```
df_copy = df.copy()
```

```
#trying to add or remove the columns to see if it will improve the model
df['total_autism_score'] = df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', '

df.drop('age_desc', axis=1, inplace=True)
```

```
#seeing how many null values are in each column
print(df.isnull().sum())
```

```
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                   4
gender                0
ethnicity            43
jaundice              0
autism                0
country_of_res        0
used_app_before       0
result                0
relation             43
class                 0
total_autism_score    0
dtype: int64
```

```
#in the 4 age columns, there are 4 missing values. I will fill them with the mean of the column
df['age'].fillna(df['age'].mean(), inplace=True)
#in ethnicity, there are 43 missing values. I will fill them with the mode of the column, since they are categorical, i will ad as "unknown"
df['ethnicity'].fillna('unknown', inplace=True)
#in relation, there are 6\43 missing values. I will fill them with the mode of the column, since they are categorical, i will add as "unknown
df['relation'].fillna('unknown', inplace=True)
```

We used mean for age null values. There were only 4 null values so we used mean. For the other categorical values we just put in unknown as there isnt a way to just guesstimate the ethnicity and relation.

```
df['age_autism_interaction'] = df['age'] * df['total_autism_score']
```

```
df['class'] = df['class'].map({'NO': 0, 'YES': 1})
y = df['class']

X = df.drop('class', axis=1)
X['Gender'] = X['gender'].map({'M': 0, 'F': 1})
X['jaundice'] = X['jaundice'].map({'no': 0, 'yes': 1})
X['autism'] = X['autism'].map({'no': 0, 'yes': 1})
X['used_app_before'] = X['used_app_before'].map({'no': 0, 'yes': 1})
```

For categorical data types which are binary, we are just using binary encoding. The columns like gender, jaundice, autism, used_app_before,class only have two categorical values in them. So we used 1 and 0 for the values.

```
#one hot encoding categorical columns remaining
categorical_cols = X.select_dtypes(include=['object']).columns
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True,dtype=int)
```

One hot encoding was used for all the other categorical values as they have more than 2 categories in them.

```
categorical_cols = X.select_dtypes(include=['object']).columns
```

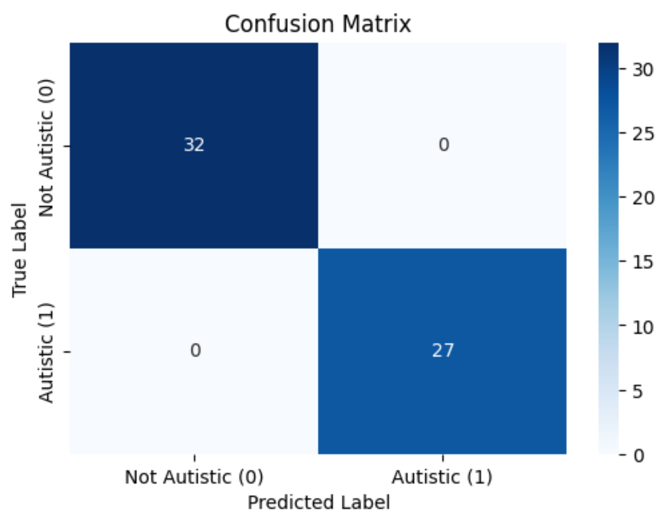
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)
```

For Validation purposes we are using 20% of data for testing our model.

```
import xgboost as xgb
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
```

```
model = xgb.XGBClassifier(subsample=0.6, n_estimators=300, min_child_weight=3, max_depth=6, learning_rate=0.1, gamma=2.5, colsample_bytree=0.
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
f1 = f1_score(y_test, y_pred)
print(f"F1 Score: {f1:.4f}")
cm = confusion_matrix(y_test, y_pred)

# Displaying a confusion matrix as a heatmap
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Not Autistic (0)', 'Autistic (1)'], yticklabels=['Not Autistic (0)', 'Autist
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

```
Accuracy: 1.0000
F1 Score: 1.0000
```



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

           0       1.00      1.00      1.00        32
           1       1.00      1.00      1.00        27

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59
```

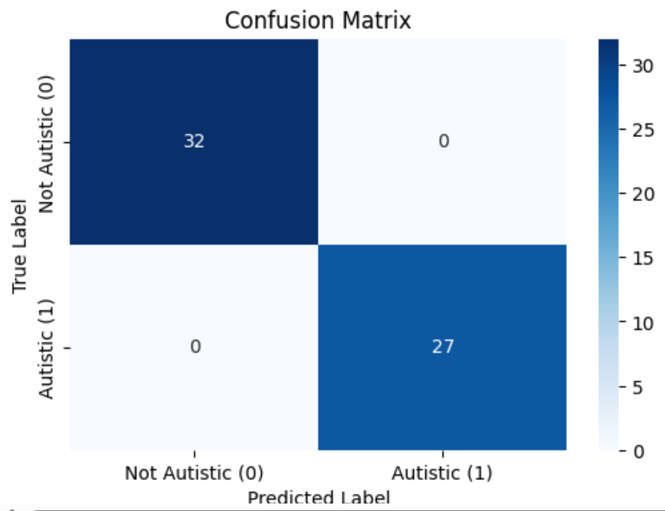
The hyperparameters used are found by using the tuning data from the code box result below.

We first used xgBoost because when Olivier did its presentation last week we thought it would be a really good model to try out. Since we knew that autism cases are not prevalent in real world. We wanted to emphasize on F1 score than on accuracy.

```
#for hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import f1_score, confusion_matrix

xgb_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='logloss', random_state=42)

param_dist = {
    'n_estimators': [100,200,300,400,500],
    'learning_rate': [0.025,0.05,0.1,0.2],
    'max_depth': [5,6,7,8],
    'min_child_weight': np.arange(1, 6, 1),
    'gamma': np.linspace(0, 5, 5),
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.3, 0.5, 0.7, 1.0],
}


random_search = RandomizedSearchCV(
    estimator=xgb_model,
    param_distributions=param_dist,
    n_iter=400,
    scoring='f1',
    cv=3,
    verbose=2,
    n_jobs=-1,
    random_state=42
)


random_search.fit(X_train, y_train)
print("Best Parameters:", random_search.best_params_)

best_model = random_search.best_estimator_
y_pred = best_model.predict(X_test)



y_pred = best_model.predict_proba(X_test)
```

```
f1 = f1_score(y_test, y_pred)
print(f"F1 Score: {f1:.4f}")

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Not Autistic (0)', 'Autistic (1)'],
            yticklabels=['Not Autistic (0)', 'Autistic (1)'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

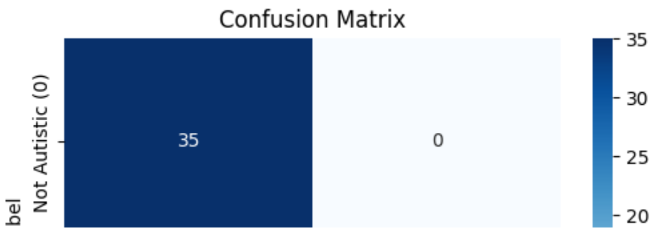
```
Fitting 3 folds for each of 400 candidates, totalling 1200 fits
Best Parameters: {'subsample': 0.6, 'n_estimators': 300, 'min_child_weight': np.int64(3), 'max_depth': 6, 'learning_rate': 0.1, 'gamma':
F1 Score: 1.0000
```



When doing the hyperparameters tuning we could see that the given details does give really good answers. It really does have accuracy of 100% and this is validation on test dataset. So this already is a really good score so i didnt have any other reason to try another model. We might argue that this is because of overfitting but since we are looking at different data to validate our model and the model having a good score even on that model means that this is a good thing and sufficient model

```
#now using random forest
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

f1 = f1_score(y_test, y_pred)
print(f"F1 Score: {f1:.4f}")

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Not Autistic (0)', 'Autistic (1)'],
            yticklabels=['Not Autistic (0)', 'Autistic (1)'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 1.0000
F1 Score: 1.0000
```


Confusion Matrix
Not Autistic (0)
bel