autism_detection

April 8, 2025

1 Importing required Libraries and the Dataset.

[6]:
```python
import numpy as np
import pandas as pd
from time import time
from IPython.display import display # Allows use of Display() for Dataframes.

# Import supplementary visualization code visuals.py
import visuals as vs

%matplotlib inline

data = pd.read_csv("autism_data.csv")
display(data.head(5))
```

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

   A8_Score  A9_Score  A10_Score  …  gender       ethnicity jundice austim  \
0         1         0          0  …       f  White-European      no     no
1         1         0          1  …       m          Latino      no    yes
2         1         1          1  …       m          Latino     yes    yes
3         1         0          1  …       f  White-European      no    yes
4         1         0          0  …       f               ?      no     no

   contry_of_res used_app_before result     age_desc relation Class/ASD
0  United States              no    6.0  18 and more     Self        NO
1         Brazil              no    5.0  18 and more     Self        NO
2          Spain              no    8.0  18 and more   Parent       YES
3  United States              no    6.0  18 and more     Self        NO
4          Egypt              no    2.0  18 and more        ?        NO

[5 rows x 21 columns]
```

```python
[7]: # Total number of records:
n_records = len(data.index)

# Total number of records with ASD
n_asd_yes = len(data[data['Class/ASD'] == 'YES'])

# Total number of records without ASD
n_asd_no = len(data[data['Class/ASD'] == 'NO'])

# Percentage of individuals with ASD
yes_percentage = float((n_asd_yes) / n_records * 100)

# Printing the outputs
print(f'Total number of records  : {n_records}')
print(f'Number of individuals with ASD : {n_asd_yes}')
print(f'Number of individuals without ASD : {n_asd_no}')
print("Percentage of individuals with ASD : {:.2f}%".format(yes_percentage))
```

```
Total number of records  : 704
Number of individuals with ASD : 189
Number of individuals without ASD : 515
Percentage of individuals with ASD : 26.85%
```

1.1 Featureset Exploration

```python
[8]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 704 entries, 0 to 703
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   A1_Score         704 non-null    int64
 1   A2_Score         704 non-null    int64
 2   A3_Score         704 non-null    int64
 3   A4_Score         704 non-null    int64
 4   A5_Score         704 non-null    int64
 5   A6_Score         704 non-null    int64
 6   A7_Score         704 non-null    int64
 7   A8_Score         704 non-null    int64
 8   A9_Score         704 non-null    int64
 9   A10_Score        704 non-null    int64
 10  age              702 non-null    float64
 11  gender           704 non-null    object
 12  ethnicity        704 non-null    object
 13  jundice          704 non-null    object
 14  austim           704 non-null    object
 15  contry_of_res    704 non-null    object
```

```
 16  used_app_before  704 non-null    object
 17  result           704 non-null    float64
 18  age_desc         704 non-null    object
 19  relation         704 non-null    object
 20  Class/ASD        704 non-null    object
dtypes: float64(2), int64(10), object(9)
memory usage: 115.6+ KB
```

[9]:
```
data.describe()
```

[9]:
```
         A1_Score    A2_Score    A3_Score    A4_Score    A5_Score    A6_Score  \
count  704.000000  704.000000  704.000000  704.000000  704.000000  704.000000
mean     0.721591    0.453125    0.457386    0.495739    0.498580    0.284091
std      0.448535    0.498152    0.498535    0.500337    0.500353    0.451301
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
50%      1.000000    0.000000    0.000000    0.000000    0.000000    0.000000
75%      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

         A7_Score    A8_Score    A9_Score   A10_Score         age      result
count  704.000000  704.000000  704.000000  704.000000  702.000000  704.000000
mean     0.417614    0.649148    0.323864    0.573864   29.698006    4.875000
std      0.493516    0.477576    0.468281    0.494866   16.507465    2.501493
min      0.000000    0.000000    0.000000    0.000000   17.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000   21.000000    3.000000
50%      0.000000    1.000000    0.000000    1.000000   27.000000    4.000000
75%      1.000000    1.000000    1.000000    1.000000   35.000000    7.000000
max      1.000000    1.000000    1.000000    1.000000  383.000000   10.000000
```

1.2 Preparing the Data

[10]:
```
data.isna().sum()
```

[10]:
```
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                2
gender             0
ethnicity          0
```

```
jundice            0
austim             0
contry_of_res      0
used_app_before    0
result             0
age_desc           0
relation           0
Class/ASD          0
dtype: int64
```

Dropping missing values

[11]:
```python
data.dropna(inplace=True)
data.describe()
```

[11]:
```
         A1_Score    A2_Score    A3_Score    A4_Score    A5_Score    A6_Score  \
count  702.000000  702.000000  702.000000  702.000000  702.000000  702.000000
mean     0.723647    0.452991    0.458689    0.497151    0.498575    0.284900
std      0.447512    0.498140    0.498646    0.500348    0.500354    0.451689
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
50%      1.000000    0.000000    0.000000    0.000000    0.000000    0.000000
75%      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

         A7_Score    A8_Score    A9_Score   A10_Score         age      result
count  702.000000  702.000000  702.000000  702.000000  702.000000  702.000000
mean     0.417379    0.650997    0.324786    0.574074   29.698006    4.883191
std      0.493478    0.476995    0.468629    0.494835   16.507465    2.498051
min      0.000000    0.000000    0.000000    0.000000   17.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000   21.000000    3.000000
50%      0.000000    1.000000    0.000000    1.000000   27.000000    4.000000
75%      1.000000    1.000000    1.000000    1.000000   35.000000    7.000000
max      1.000000    1.000000    1.000000    1.000000  383.000000   10.000000
```

[12]:
```python
# After Data Cleaning

# Total number of records:
n_records = len(data.index)

# Total number of records with ASD
n_asd_yes = len(data[data['Class/ASD'] == 'YES'])

# Total number of records without ASD
n_asd_no = len(data[data['Class/ASD'] == 'NO'])

# Printing the outputs
print("AFTER REMOVING NULL VALUES : ")
```

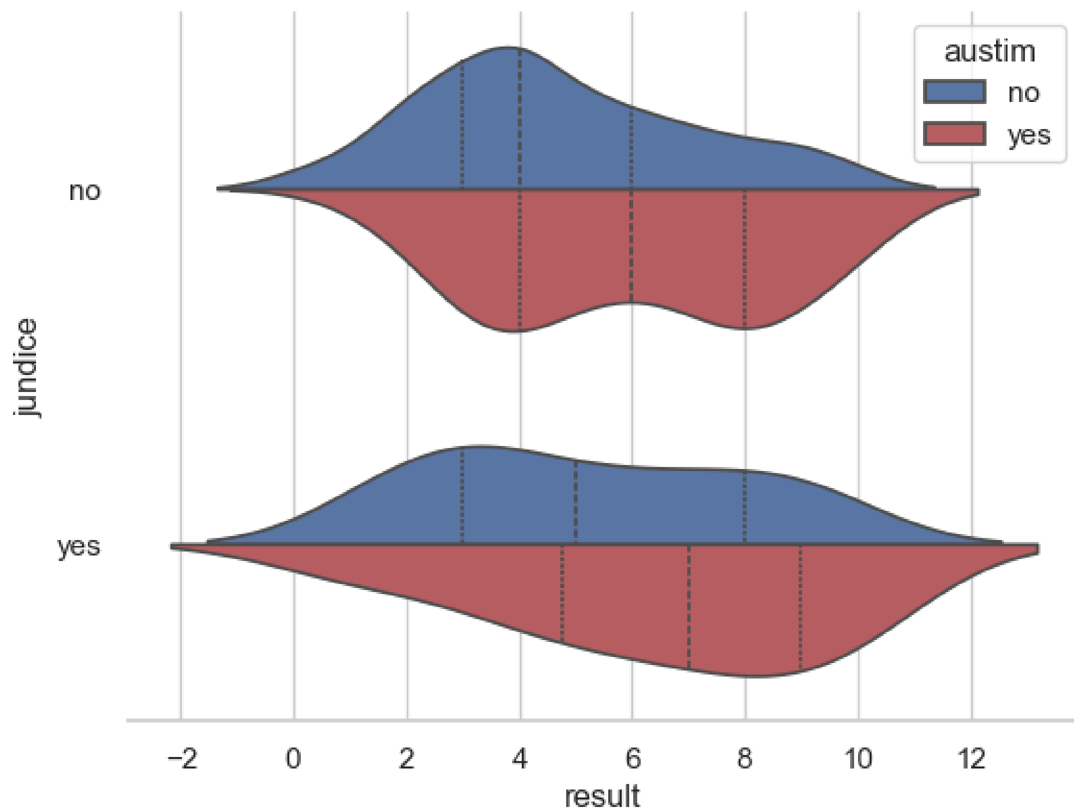
```
print(f'Total number of records  : {n_records}')
print(f'Number of individuals with ASD : {n_asd_yes}')
print(f'Number of individuals without ASD : {n_asd_no}')
```

```
AFTER REMOVING NULL VALUES :
Total number of records  : 702
Number of individuals with ASD : 189
Number of individuals without ASD : 513
```

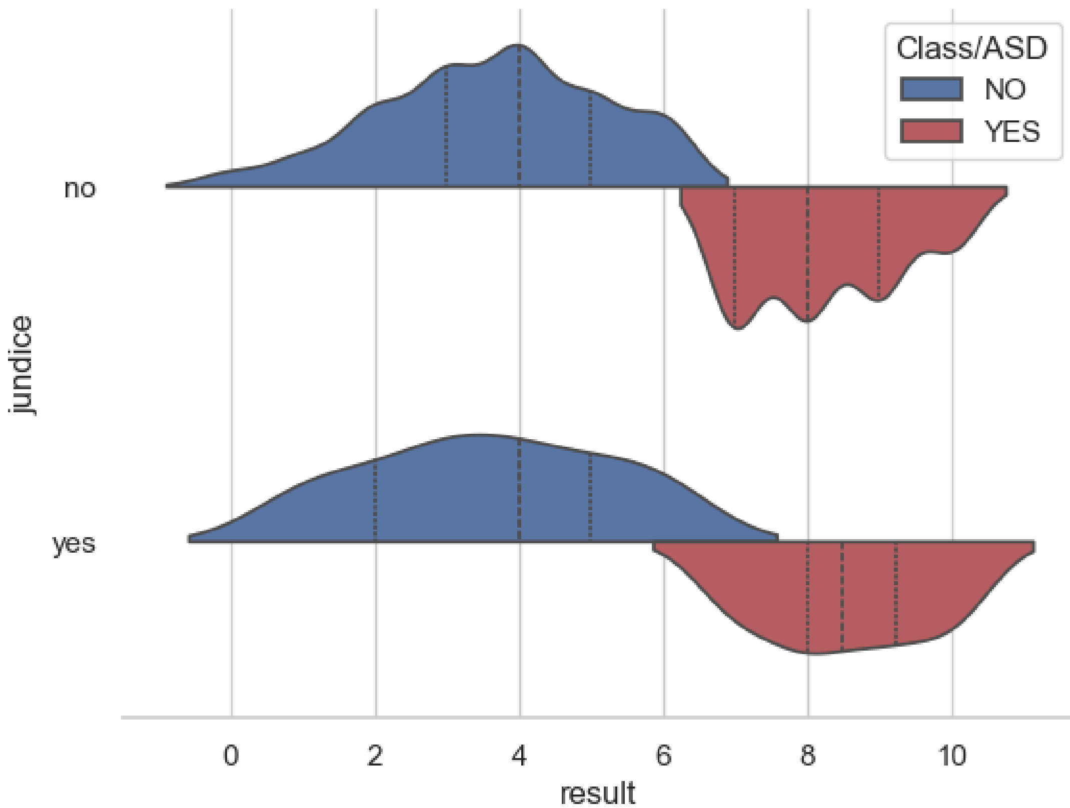
1.2.1 Visualizations with Seaborn

[13]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="whitegrid", color_codes=True)
```

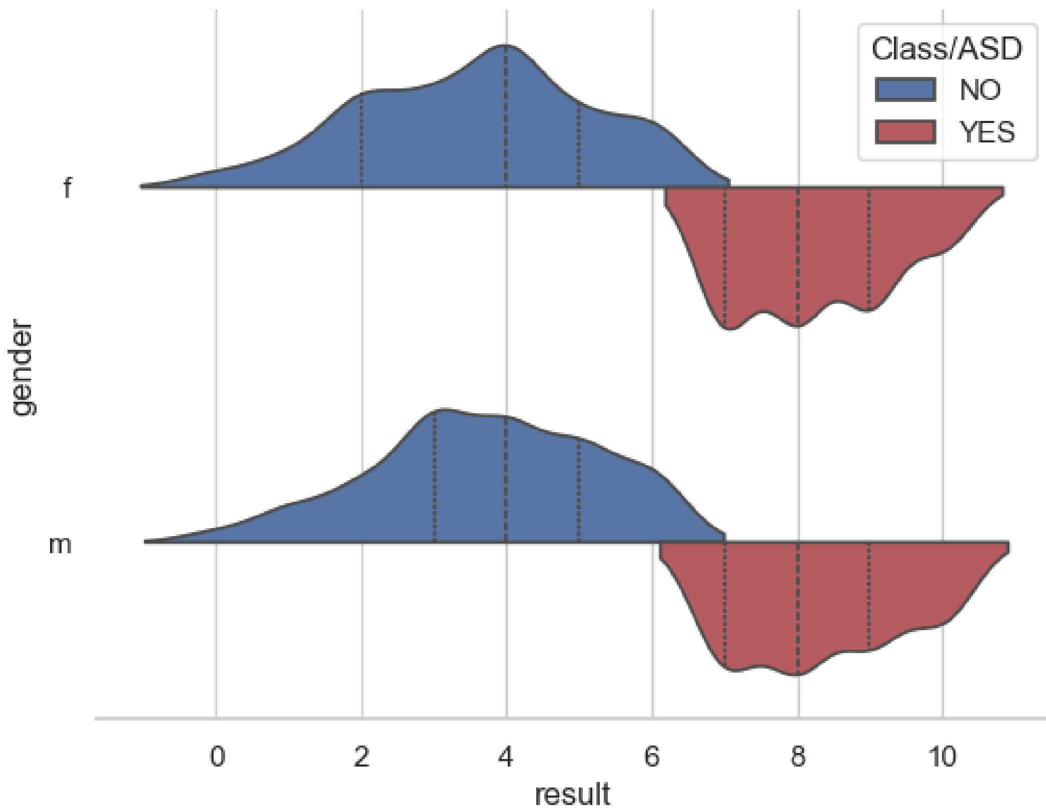
[14]:
```
# Draw a nested violinplot and split the violins for easier comparison
sns.violinplot(x="result", y="jundice", hue="austim", data=data,␣
 ↪split=True,inner="quart", palette={'yes': "r", 'no': "b"})
sns.despine(left=True)
```



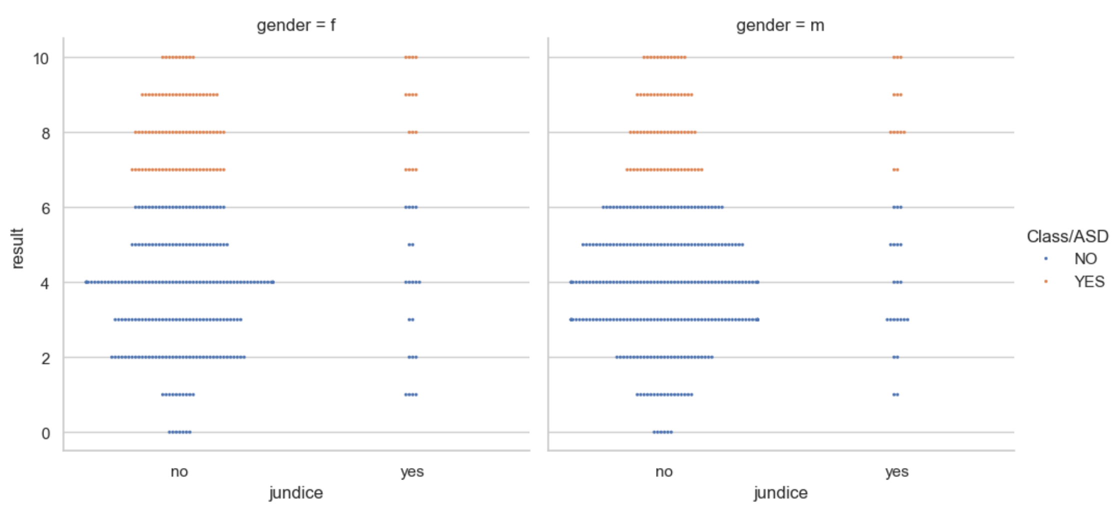
[15]:
```
# Draw a nested violinplot and split the violins for easier comparison
sns.violinplot(x="result", y="jundice", hue="Class/ASD", data=data, split=True,
               inner="quart", palette={'YES': "r", 'NO': "b"})
sns.despine(left=True)
```



[16]:
```
# Draw a nested violinplot and split the violins for easier comparison
sns.violinplot(x="result", y="gender", hue="Class/ASD", data=data, split=True,
               inner="quart", palette={'YES': "r", 'NO': "b"})
sns.despine(left=True)
```



[17]:
```
sns.catplot(x="jundice", y="result", hue="Class/ASD", s = 5, col="gender",␣
↪data=data, kind="swarm");
```



Convert the Pandas dataframes into numpy arrays that can be used by scikit_learn. Let's create an array that extracts only the feature data we want to work with and another array that contains the classes (class/ASD).

[18]:
```
data_raw = data['Class/ASD']
features_raw = data[['age', 'gender', 'ethnicity', 'jundice', 'austim',␣
 ↪'contry_of_res', 'result',
                                ␣
 ↪'relation','A1_Score','A2_Score','A3_Score','A4_Score','A5_Score','A6_Score','A7_Score','A8
                                  'A9_Score','A10_Score']]
```

Data Preprocessing : using MinMaxScaler()

[19]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
num = ['age','result']
features_minmax_transform = pd.DataFrame(data = features_raw)
features_minmax_transform[num] = scaler.fit_transform(features_raw[num])
```

[20]:
```
display(features_minmax_transform.head(5))
```

```
        age gender      ethnicity jundice austim  contry_of_res  result  \
0  0.024590      f  White-European      no     no  United States     0.6
1  0.019126      m         Latino      no    yes         Brazil     0.5
2  0.027322      m         Latino     yes    yes          Spain     0.8
3  0.049180      f  White-European      no    yes  United States     0.6
4  0.062842      f               ?      no     no          Egypt     0.2

  relation  A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score  \
0     Self         1         1         1         1         0         0
1     Self         1         1         0         1         0         0
2   Parent         1         1         0         1         1         0
3     Self         1         1         0         1         0         0
4        ?         1         0         0         0         0         0

   A7_Score  A8_Score  A9_Score  A10_Score
0         1         1         0          0
1         0         1         0          1
2         1         1         1          1
3         1         1         0          1
4         0         1         0          0
```

1.2.2 One-Hot Encoding on features__minmax__transform

[21]:
```
features_final = pd.get_dummies(features_minmax_transform)
features_final.head(5)
```

[21]:
```
        age  result  A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  \
0  0.024590     0.6         1         1         1         1         0
```

```
1  0.019126     0.5         1         1         0         1         0
2  0.027322     0.8         1         1         0         1         1
3  0.049180     0.6         1         1         0         1         0
4  0.062842     0.2         1         0         0         0         0

   A6_Score  A7_Score  A8_Score  …  contry_of_res_United Kingdom  \
0         0         1         1  …                         False
1         0         0         1  …                         False
2         0         1         1  …                         False
3         0         1         1  …                         False
4         0         0         1  …                         False

   contry_of_res_United States  contry_of_res_Uruguay  contry_of_res_Viet Nam  \
0                         True                  False                   False
1                        False                  False                   False
2                        False                  False                   False
3                         True                  False                   False
4                        False                  False                   False

   relation_?  relation_Health care professional  relation_Others  \
0       False                              False            False
1       False                              False            False
2       False                              False            False
3       False                              False            False
4        True                              False            False

   relation_Parent  relation_Relative  relation_Self
0            False              False           True
1            False              False           True
2             True              False          False
3            False              False           True
4            False              False          False

[5 rows x 103 columns]
```

1.2.3 Encode all classes data to numerical values

[22]:
```python
data_classes = data_raw.apply(lambda x : 1 if x == 'YES' else 0)
```

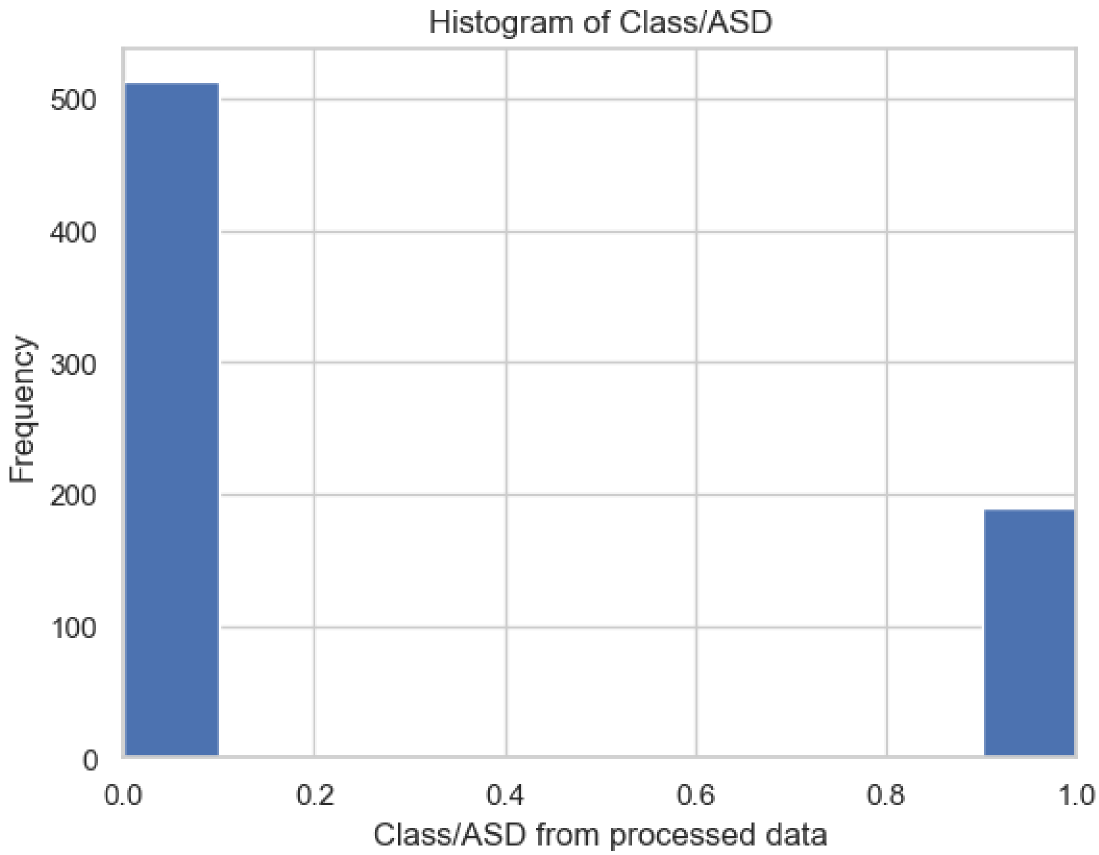
[23]:
```python
encoded = list(features_final.columns)
print("{} total features after one-hot encoding".format(len(encoded)))
print(encoded)
```

```
103 total features after one-hot encoding
['age', 'result', 'A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score',
'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'gender_f',
'gender_m', 'ethnicity_?', 'ethnicity_Asian', 'ethnicity_Black',
'ethnicity_Hispanic', 'ethnicity_Latino', 'ethnicity_Middle Eastern ',
```

```
'ethnicity_Others', 'ethnicity_Pasifika', 'ethnicity_South Asian',
'ethnicity_Turkish', 'ethnicity_White-European', 'ethnicity_others',
'jundice_no', 'jundice_yes', 'austim_no', 'austim_yes',
'contry_of_res_Afghanistan', 'contry_of_res_AmericanSamoa',
'contry_of_res_Angola', 'contry_of_res_Argentina', 'contry_of_res_Armenia',
'contry_of_res_Aruba', 'contry_of_res_Australia', 'contry_of_res_Austria',
'contry_of_res_Azerbaijan', 'contry_of_res_Bahamas', 'contry_of_res_Bangladesh',
'contry_of_res_Belgium', 'contry_of_res_Bolivia', 'contry_of_res_Brazil',
'contry_of_res_Burundi', 'contry_of_res_Canada', 'contry_of_res_Chile',
'contry_of_res_China', 'contry_of_res_Costa Rica', 'contry_of_res_Cyprus',
'contry_of_res_Czech Republic', 'contry_of_res_Ecuador', 'contry_of_res_Egypt',
'contry_of_res_Ethiopia', 'contry_of_res_Finland', 'contry_of_res_France',
'contry_of_res_Germany', 'contry_of_res_Hong Kong', 'contry_of_res_Iceland',
'contry_of_res_India', 'contry_of_res_Indonesia', 'contry_of_res_Iran',
'contry_of_res_Iraq', 'contry_of_res_Ireland', 'contry_of_res_Italy',
'contry_of_res_Japan', 'contry_of_res_Jordan', 'contry_of_res_Kazakhstan',
'contry_of_res_Lebanon', 'contry_of_res_Malaysia', 'contry_of_res_Mexico',
'contry_of_res_Nepal', 'contry_of_res_Netherlands', 'contry_of_res_New Zealand',
'contry_of_res_Nicaragua', 'contry_of_res_Niger', 'contry_of_res_Oman',
'contry_of_res_Pakistan', 'contry_of_res_Philippines', 'contry_of_res_Portugal',
'contry_of_res_Romania', 'contry_of_res_Russia', 'contry_of_res_Saudi Arabia',
'contry_of_res_Serbia', 'contry_of_res_Sierra Leone', 'contry_of_res_South
Africa', 'contry_of_res_Spain', 'contry_of_res_Sri Lanka',
'contry_of_res_Sweden', 'contry_of_res_Tonga', 'contry_of_res_Turkey',
'contry_of_res_Ukraine', 'contry_of_res_United Arab Emirates',
'contry_of_res_United Kingdom', 'contry_of_res_United States',
'contry_of_res_Uruguay', 'contry_of_res_Viet Nam', 'relation_?',
'relation_Health care professional', 'relation_Others', 'relation_Parent',
'relation_Relative', 'relation_Self']
```

[24]:
```
plt.hist(data_classes, bins=10)
plt.xlim(0,1)
plt.title('Histogram of Class/ASD')
plt.xlabel('Class/ASD from processed data')
plt.ylabel('Frequency')
```

[24]:
```
Text(0, 0.5, 'Frequency')
```



1.3 Shuffle and Split the data

All the categorical variables have been converted to numerical variables and have been normalized, we split the data into train and test set, test set will be 20% of the total data.

[25]:
```
from sklearn.model_selection import train_test_split
np.random.seed(123)
X_train, X_test, y_train, y_test =␣
 ↪train_test_split(features_final,data_classes,test_size=0.2, random_state=1)
print("Train set has {} enteries.".format(X_train.shape[0]))
print("Test set has {} enteries.".format(X_test.shape[0]))
```

```
Train set has 561 enteries.
Test set has 141 enteries.
```

1.4 Models :

1.4.1 1. Decision Tress

[26]:
```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
dec_model = DecisionTreeClassifier()
dec_model.fit(X_train.values, y_train)
```

[26]:
```
DecisionTreeClassifier()
```

[27]:
```
y_pred = dec_model.predict(X_test.values)
print('True : ', y_test.values[0:25])
print('False :', y_pred[0:25])
```

```
True :  [1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0]
False : [1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0]
```

[28]:
```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, y_pred)
print(cm)
TP = cm[1,1]
FP = cm[0,1]
TN = cm[0,0]
FN = cm[1,0]
```

```
[[101   0]
 [  0  40]]
```

[29]:
```
print('Accuracy:')
print((TN+TP)/float(TP+TN+FP+FN))
print('Error:')
print((FP+FN)/float(TP+TN+FP+FN))
print('Precision:')
print(metrics.precision_score(y_test,y_pred))
print('Score:')
print(dec_model.score(X_test.values, y_test))
```

```
Accuracy:
1.0
Error:
0.0
Precision:
1.0
Score:
1.0
```

1.4.2 2. Random Forest

[30]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
rndm_model = RandomForestClassifier(n_estimators=5, random_state=1)
cv_score = cross_val_score(rndm_model, features_final, data_classes, cv =10)
cv_score.mean()
```

[30]: np.float64(0.9900603621730383)

[31]:
```
# F-beta Score
rndm_model.fit(X_train.values, y_train)
from sklearn.metrics import fbeta_score
y_pred = rndm_model.predict(X_test.values)
fbeta_score(y_test, y_pred, average='binary', beta=0.5)
```

[31]: 1.0

1.4.3 3. Support Vector Machine

[32]:
```
from sklearn import svm
svm_model = svm.SVC(kernel='linear', C=1, gamma=2)
cv_score = cross_val_score(svm_model, features_final, data_classes, cv =10)
cv_score.mean()
```

[32]: np.float64(1.0)

[33]:
```
#F-beta Score
svm_model.fit(X_train.values, y_train)
from sklearn.metrics import fbeta_score
y_pred = svm_model.predict(X_test.values)
fbeta_score(y_test, y_pred, average='binary', beta=0.5)
```

[33]: 1.0

1.4.4 4. K-Nearest-Neighbors(KNN)

[34]:
```
from sklearn import neighbors
knn_model = neighbors.KNeighborsClassifier(n_neighbors=10)
cv_score = cross_val_score(knn_model, features_final, data_classes, cv =10)
cv_score.mean()
```

[34]: np.float64(0.9458752515090543)

[35]:
```
#F-beta Score
knn_model.fit(X_train.values, y_train)
from sklearn.metrics import fbeta_score
y_pred = knn_model.predict(X_test.values)
```

```
fbeta_score(y_test, y_pred, average='binary', beta=0.5)
```

[35]: 0.9183673469387755

[36]:
```
for n in range(10,30):
    knn_model = neighbors.KNeighborsClassifier(n_neighbors=n)
    cv_scores = cross_val_score(knn_model, features_final, data_classes, cv=10)
    print (n, cv_scores.mean())
```

```
10 0.9458752515090543
11 0.9473239436619719
12 0.9444869215291751
13 0.9501609657947686
14 0.9458953722334005
15 0.9458953722334004
16 0.951569416498994
17 0.951549295774648
18 0.9529778672032194
19 0.9572635814889336
20 0.9529778672032194
21 0.9529778672032194
22 0.9486921529175051
23 0.9472635814889336
24 0.9486921529175051
25 0.9486720321931589
26 0.9515090543259557
27 0.9501006036217303
28 0.9486720321931589
29 0.9472434607645874
```

Hence, K is not making any significant difference on accuracy of our predictions.

1.4.5 5. Naive Bayes

[37]:
```
from sklearn.naive_bayes import MultinomialNB
nb_model = MultinomialNB()
cv_score = cross_val_score(nb_model, features_final, data_classes, cv =10)
cv_score.mean()
```

[37]: np.float64(0.8746277665995976)

[38]:
```
#F-beta Score
nb_model.fit(X_train.values, y_train)
from sklearn.metrics import fbeta_score
y_pred = nb_model.predict(X_test.values)
fbeta_score(y_test, y_pred, average='binary', beta=0.5)
```

[38]: 0.7675438596491229

1.4.6 6. Logistic Regression

[39]:
```python
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression()
cv_score = cross_val_score(lr_model, features_final, data_classes, cv =10)
cv_score.mean()
```

[39]:
```
np.float64(0.9971428571428571)
```

[40]:
```python
#F-beta Score
lr_model.fit(X_train.values, y_train)
from sklearn.metrics import fbeta_score
y_pred = lr_model.predict(X_test.values)
fbeta_score(y_test, y_pred, average='binary', beta=0.5)
```

[40]:
```
0.9948979591836735
```

1.5 Model Tuning

[43]:
```python
from sklearn.metrics import fbeta_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.svm import SVC
```

[44]:
```python
def f_beta_score(y_true, y_predict):
    return fbeta_score(y_true, y_predict, beta = 0.5)
clf = SVC(random_state = 1)
parameters = {'C':range(1,6),'kernel':
 ↪['linear','poly','rbf','sigmoid'],'degree':range(1,6)}
scorer = make_scorer(f_beta_score)
```

[45]:
```python
grid_obj = GridSearchCV(estimator = clf, param_grid = parameters, scoring =␣
 ↪scorer)
grid_fit = grid_obj.fit(X_train.values, y_train)
best_clf = grid_fit.best_estimator_
```

[46]:
```python
predictions = (clf.fit(X_train.values, y_train)).predict(X_test.values)
best_predictions = best_clf.predict(X_test.values)
```

[47]:
```python
print ("Unoptimized model\n------")
print ("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test,␣
 ↪predictions)))
print ("F-score on testing data: {:.4f}".format(fbeta_score(y_test,␣
 ↪predictions, beta = 0.5)))
print ("\nOptimized Model\n------")
```

```
print ("Final accuracy score on the testing data: {:.4f}".
 ↪format(accuracy_score(y_test, best_predictions)))
print ("Final accuracy score on the testing data: {:.4f}".
 ↪format(accuracy_score(y_test, best_predictions)))
print ("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test,␣
 ↪best_predictions, beta = 0.5)))
```

```
Unoptimized model
------
Accuracy score on testing data: 0.9645
F-score on testing data: 0.9574

Optimized Model
------
Final accuracy score on the testing data: 1.0000
Final accuracy score on the testing data: 1.0000
Final F-score on the testing data: 1.0000
```

[ ]:

```

```