Leveraging Machine Learning for Autism Spectrum Disorder(ASD) Detection

Data

Dataset for autism disorder typically comprises structured information from individuals diagnosed with autism spectrum disorder (ASD) alongside a control group. It encompasses demographic details such as age, gender, and family history, coupled with clinical features like diagnostic scores from standardized tests such as the Autism Diagnostic Observation Schedule (ADOS) or behavioural traits. Additionally, medical history, genetic factors, environmental exposures, intervention records, and outcome measures contribute to a comprehensive understanding. Ethical collection practices and privacy safeguards are paramount, while addressing biases ensures the dataset's reliability for training ML models aimed at furthering our comprehension of autism spectrum disorder.

The dataset provided appears to contain information related to autism spectrum disorder (ASD). Here's a breakdown of the column

1. A1_Score to A10_Score: Scores representing responses to ten different questions or items. These scores may be related to specific behaviours, symptoms, or characteristics associated with ASD. Each score likely indicates the severity or frequency of a particular behaviour or trait .
2. age: Age of the individual.
3. gender: Gender of the individual m' for Male and 'f' for Female.
4. ethnicity: Ethnicity of the individual Ethnicities in text form ['White-European', 'Latino', '?', 'Others', 'Black', 'Asian','Middle-Eastern ', 'Pasifika', 'South Asian', 'Hispanic', 'Turkish', 'others'].
5. jundice: Binary variable indicating whether the individual was born with jaundice (a medical condition involving yellowing of the skin and eyes) or not
6. autism:no and yes for whether or not anyone in the immediate family has been diagnosed with autism not.
7. country_of_res: Country of residence of the individual.
8. used_app_before: Binary variable indicating whether the individual has used an app related to autism before or not.
9. result: Result or outcome of the assessment or evaluation conducted. This could be related to the presence or severity of ASD symptoms.
10. age_desc: Description of the age group the individual belongs to.
11. relation:Relation of person who completed the test.
12. Class/ASD: Binary variable indicating whether the individual has ASD or not.

Data loading

Importing Dataset

In [1]:
```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv('Autism-Child-Data.csv')  #importing data
data.head(n=10)         #viewing first 10 rows
```

Out[1]:

	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score	...	gender	ethnicity	jundic
0	1	1	1	1	0	0	1	1	0	0	...	f	White-European	n
1	1	1	0	1	0	0	0	1	0	1	...	m	Latino	n
2	1	1	0	1	1	0	1	1	1	1	...	m	Latino	ye
3	1	1	0	1	0	0	1	1	0	1	...	f	White-European	n
4	1	0	0	0	0	0	0	1	0	0	...	f	?	n
5	1	1	1	1	1	0	1	1	1	1	...	m	Others	ye
6	0	1	0	0	0	0	0	1	0	0	...	f	Black	n
7	1	1	1	1	0	0	0	0	1	0	...	m	White-European	n
8	1	1	0	0	1	0	0	1	1	1	...	m	White-European	n
9	1	1	1	1	0	1	1	1	1	0	...	m	Asian	ye

10 rows × 21 columns

Data Exploring

Numerical Features are:

'A1_Score' 'A2_Score' 'A3_Score' 'A4_Score' 'A5_Score' 'A6_Score' 'A7_Score' 'A8_Score' 'A9_Score' 'A10_Score' 'age' 'result'

Non Numeric Features (which needs to be encoding) are:

'gender' 'ethnicity' 'jundice' 'austim' 'contry_of_res' 'used_app_before' 'age_desc' 'relation' 'Class/ASD'

In [2]:
```
total_records = len(data.index) #calculating total records
total_asd_yes = len(data[data['Class/ASD'] == 'YES'])  #total individuals diagonised with ASD
total_asd_no = len(data[data['Class/ASD'] == 'NO'])    #total individuals with no ASD
yes_percent = float(total_asd_yes) / total_records *100   # percentage of individuals diagonised with ASD
print ("Total number of records: {}".format(total_records))
print ("Individuals diagonised with ASD: {}".format(total_asd_yes))
print ("Individuals with no ASD: {}".format(total_asd_no))
print ("Percentage of individuals diagonised with ASD: {:.2f}%".format(yes_percent))
```

```
Total number of records: 704
Individuals diagonised with ASD: 189
Individuals with no ASD: 515
Percentage of individuals diagonised with ASD: 26.85%
```

Data Cleaning

This step involves handling missing values, outliers, and inconsistencies in the data. Common techniques include imputation (filling missing values), outlier detection and removal, and normalization or standardization. Unfortunately, for this dataset, there are many invalid or missing entries(?) we must deal with, moreover, there are some qualities about certain features that must be adjusted.

Cleaning Numerical Features are:

In [3]:
```
data['age'].value_counts()  # counts the occurances of each data (age) values.
```

Out[3]:

21.0	49
20.0	46
23.0	37
22.0	37
19.0	35
24.0	34
27.0	31
18.0	31
30.0	30
26.0	28
25.0	27
29.0	27
28.0	24
31.0	21
32.0	18
17.0	18
35.0	17
37.0	17
40.0	16
33.0	16
42.0	15
36.0	13
38.0	12
34.0	12
43.0	11
44.0	10
47.0	8
39.0	7
46.0	6
53.0	6
55.0	6
50.0	5
52.0	5
45.0	4
48.0	4
41.0	3
49.0	3
54.0	2
61.0	2
56.0	2

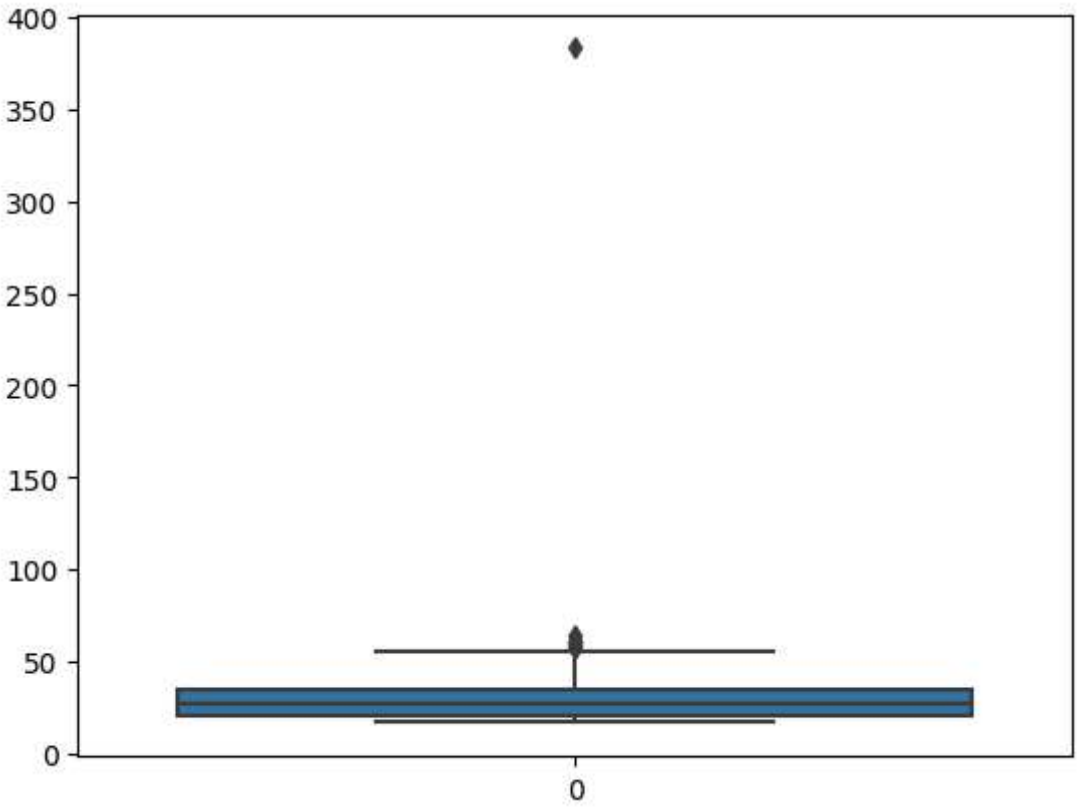
```
58.0      2
64.0      1
59.0      1
60.0      1
51.0      1
383.0     1
Name: age, dtype: int64
```

In [4]:
```
#calculating the maximum and minimum value of age
max_value = data['age'].max()
min_value = data['age'].min()
print("Maximum age found:", max_value)
print("Minimum age found:", min_value)
```

```
Maximum age found: 383.0
Minimum age found: 17.0
```

In [5]:
```
sns.boxplot(data['age']) #plotting age
```

Out[5]:
```
<AxesSubplot:>
```



we found some usual age for one of the record which is 383, either it might be wrong entry(incorrect value). we can either remove it or change the value to 38 or 83

In [6]:
```
data['age'] = data['age'].replace(383, 33) #replacing age value (as human with approx 116 age is the highest aged person)
```

In [7]:
```
data['age'].isnull().sum() #analysing null values in age column
```

Out[7]:
```
2
```

In [8]:
```
data['age'] = data['age'].fillna(round(data['age'].mean())) #filling the null values with mean of the age column data
data['age'].value_counts()
```

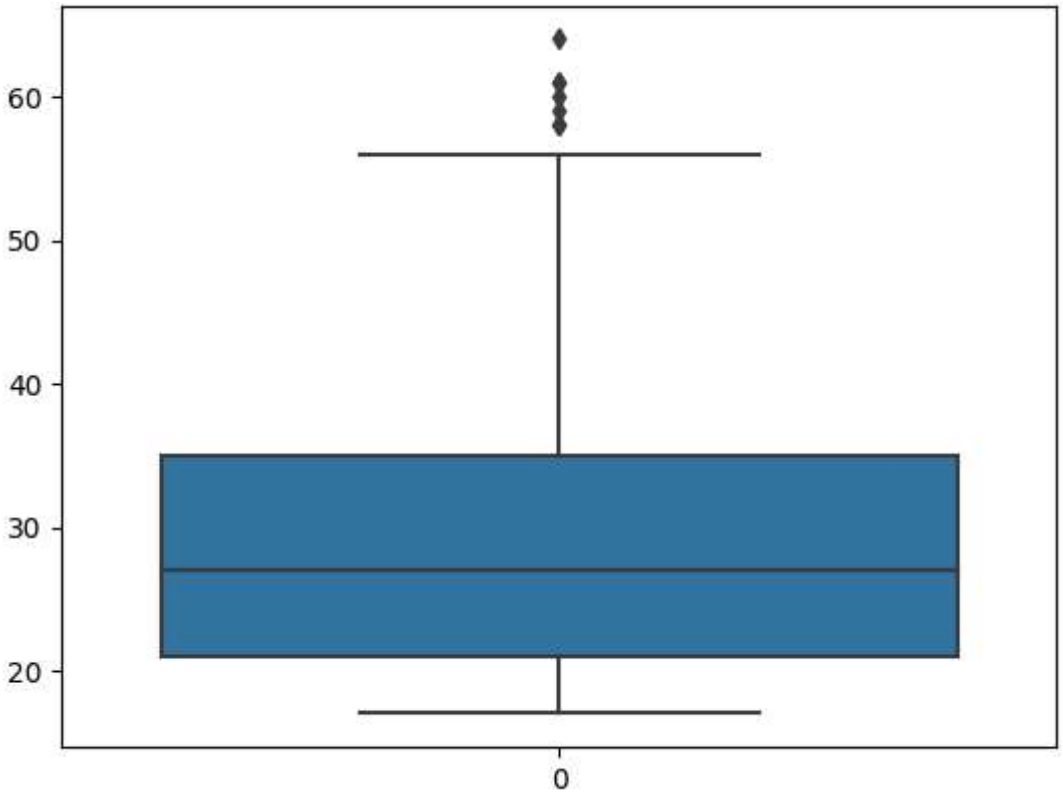
Out[8]:

21.0	49
20.0	46
23.0	37
22.0	37
19.0	35
24.0	34
27.0	31
18.0	31
30.0	30
29.0	29
26.0	28
25.0	27
28.0	24
31.0	21
17.0	18
32.0	18
37.0	17
35.0	17
33.0	17
40.0	16
42.0	15
36.0	13
34.0	12
38.0	12
43.0	11
44.0	10
47.0	8
39.0	7
46.0	6
55.0	6
53.0	6
50.0	5
52.0	5
48.0	4
45.0	4
49.0	3
41.0	3
56.0	2
61.0	2
58.0	2

```
54.0     2
64.0     1
60.0     1
59.0     1
51.0     1
Name: age, dtype: int64
```

In [9]:
```
sns.boxplot(data['age']) #plotting age
```

Out[9]: <AxesSubplot:>



In [10]:
```
#calculating the maximum and minimum value of age
max_value = data['age'].max()
min_value = data['age'].min()
```

```
print("Maximum age found:", max_value)
print("Minimum age found:", min_value)
```

```
Maximum age found: 64.0
Minimum age found: 17.0
```

Cleaning Categorical Features are:

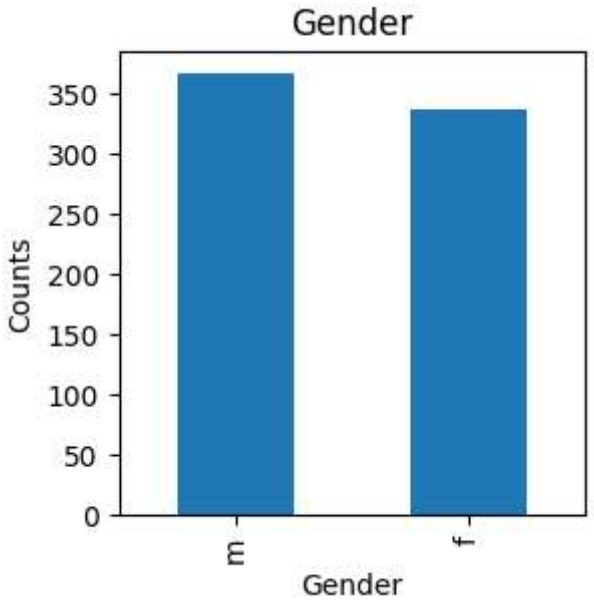
In [11]:
```
# calculating gender count.
data['gender'].value_counts()
```

Out[11]:
```
m    367
f    337
Name: gender, dtype: int64
```

In [12]:
```
# finding null values.
data['gender'].isnull().sum()
```

Out[12]:
```
0
```

In [13]:
```
gender_counts = data['gender'].value_counts()
plt.figure(figsize=(3, 3))
gender_counts.plot(kind='bar')
plt.title('Gender')
plt.xlabel('Gender')
plt.ylabel('Counts')
plt.show()
```



In [14]:
```
# calculating ethinicity count
data['ethnicity'].value_counts()
```

Out[14]:
```
White-European     233
Asian              123
?                   95
Middle Eastern      92
Black               43
South Asian         36
Others              30
Latino              20
Hispanic            13
Pasifika            12
Turkish              6
others               1
Name: ethnicity, dtype: int64
```

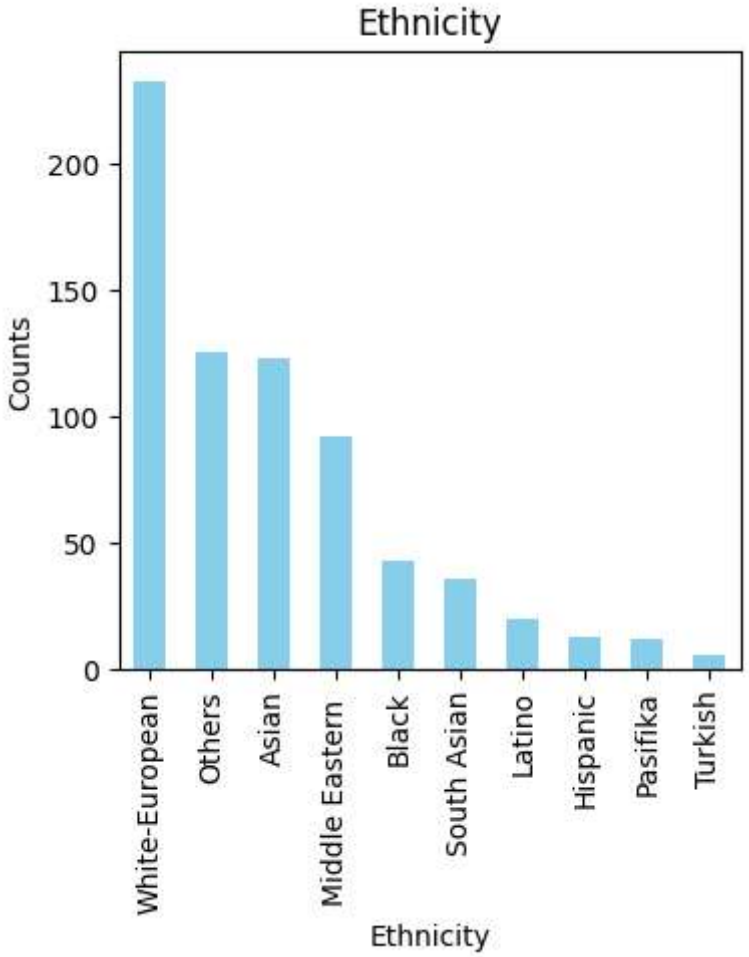
In [15]:
```
# replacing unknown with others
data['ethnicity'] = data['ethnicity'].replace('?', 'others')
#replacing all others with Others as both are same
```

```
data['ethnicity'] = data['ethnicity'].replace('others', 'Others')
data['ethnicity'].value_counts()
```

Out[15]:
```
White-European     233
Others             126
Asian              123
Middle Eastern      92
Black               43
South Asian         36
Latino              20
Hispanic            13
Pasifika            12
Turkish              6
Name: ethnicity, dtype: int64
```

In [16]:
```
ethnicity_counts = data['ethnicity'].value_counts()
plt.figure(figsize=(4, 4))
ethnicity_counts.plot(kind='bar', color='skyblue')
plt.title('Ethnicity')
plt.xlabel('Ethnicity')
plt.ylabel('Counts')
```

Out[16]:
```
Text(0, 0.5, 'Counts')
```



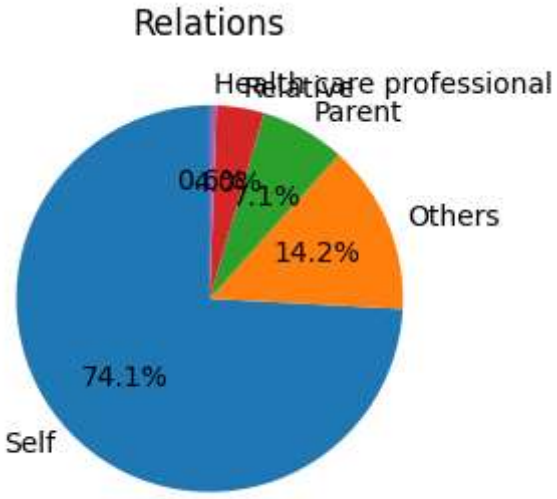
In [17]:
```
# calculating no.of relations existing
data['relation'].value_counts()
```

Out[17]:
```
Self                        522
?                            95
Parent                       50
Relative                     28
Others                        5
Health care professional      4
Name: relation, dtype: int64
```

In [18]:
```
# replacing unknown relation values with others.
data['relation'] = data['relation'].replace('?', 'Others')
data['relation'].value_counts()
```

Out[18]:
```
Self                        522
Others                      100
Parent                       50
Relative                     28
Health care professional      4
Name: relation, dtype: int64
```

In [19]:
```
relation_counts = data['relation'].value_counts()
plt.figure(figsize=(3, 3))
plt.pie(relation_counts, labels=relation_counts.index, autopct='%1.1f%%', startangle=90, pctdistance=0.60)
centre_circle = plt.Circle((0,0),0.50,fc='white')
fig = plt.gcf()
plt.axis('equal')
plt.title('Relations')
plt.tight_layout()
plt.show()
```



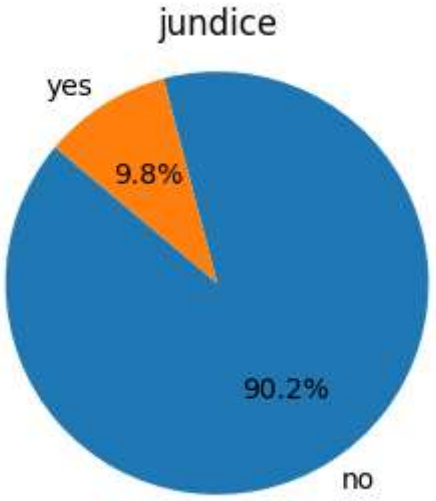
In [20]:
```
# calculating jundice counts
data['jundice'].value_counts()
```

Out[20]:
```
no     635
yes     69
Name: jundice, dtype: int64
```

In [21]:
```
#analysing for any null values in jundice column.
data['jundice'].isnull().sum()
```

Out[21]:
```
0
```

In [22]:
```
value_counts = data['jundice'].value_counts()
plt.figure(figsize=(3,3))
plt.pie(value_counts, labels=value_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('jundice')
plt.axis('equal')
plt.show()
```



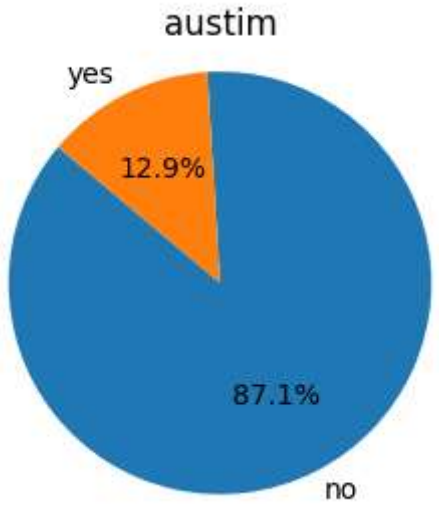
In [23]:
```
#calculating autism counts
data['austim'].value_counts()
```

Out[23]:
```
no     613
yes     91
Name: austim, dtype: int64
```

In [24]:
```
# checking for null values.
data['austim'].isnull().sum()
```

Out[24]:
```
0
```

In [25]:
```
value_counts = data['austim'].value_counts()
plt.figure(figsize=(3,3))
plt.pie(value_counts, labels=value_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('austim')
plt.axis('equal')
plt.show()
```



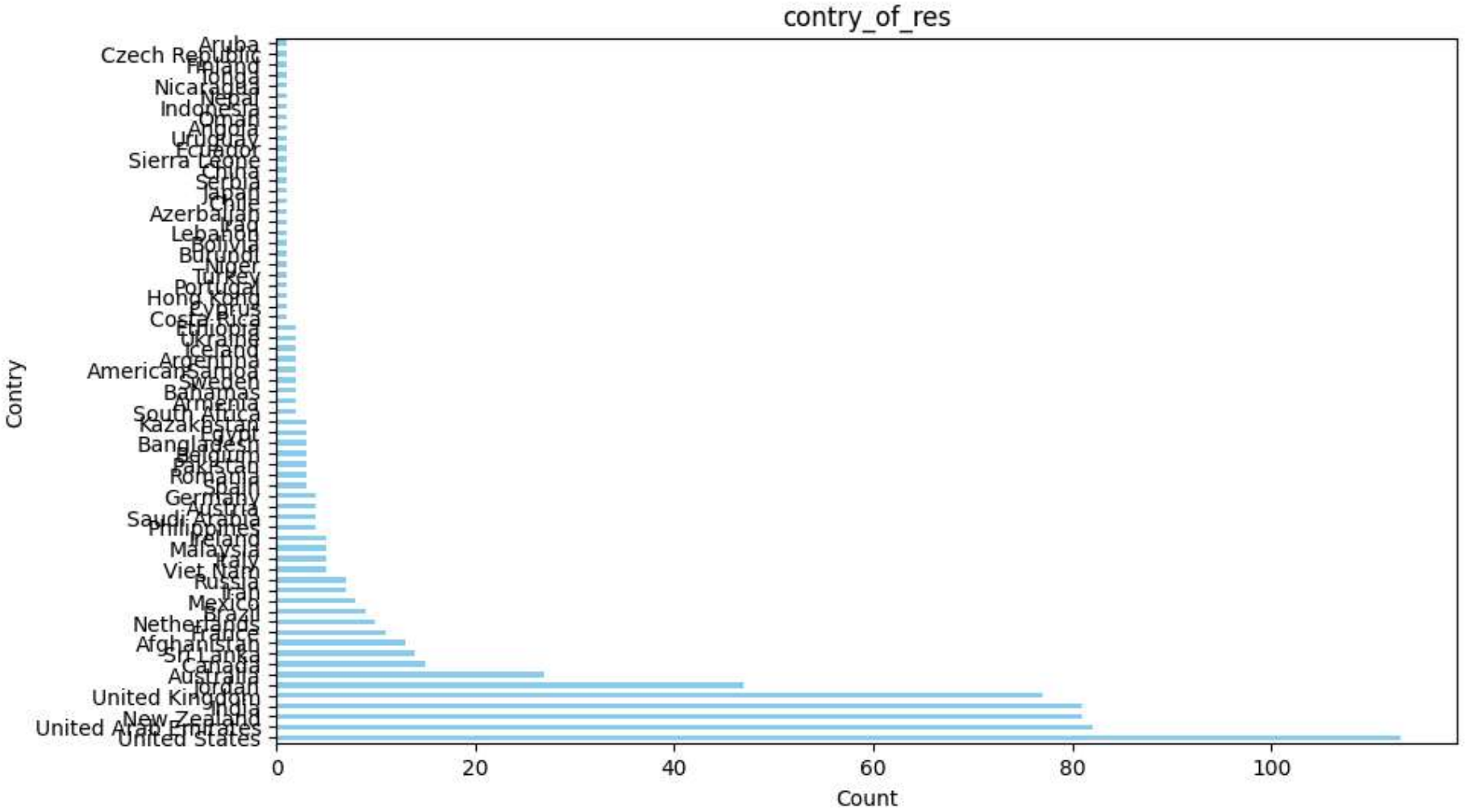
In [26]:
```
#calculating contry_of_res count
data['contry_of_res'].value_counts()
```

Out[26]:
```
United States           113
United Arab Emirates     82
New Zealand              81
India                    81
United Kingdom           77
                        ...
Nicaragua                 1
Tonga                     1
Finland                   1
Czech Republic            1
Aruba                     1
Name: contry_of_res, Length: 67, dtype: int64
```

In [27]:
```
# checking for any null values in country of residence column.
data['contry_of_res'].isnull().sum()
```

Out[27]:
```
0
```

In [28]:
```
value_counts = data['contry_of_res'].value_counts()
plt.figure(figsize=(10, 6))
value_counts.plot(kind='barh', color='skyblue')
plt.xlabel('Count')
plt.ylabel('Contry')
plt.title('contry_of_res')
plt.show()
```

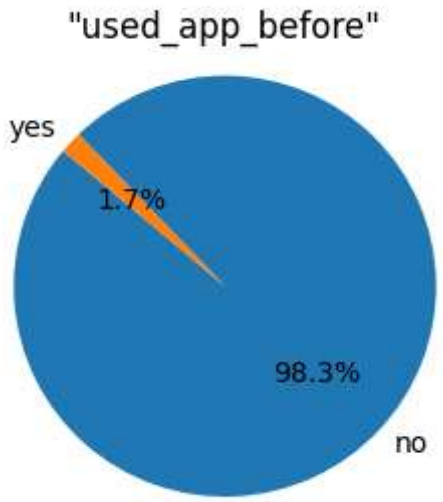


In [29]:
```
data['used_app_before'].value_counts()
```

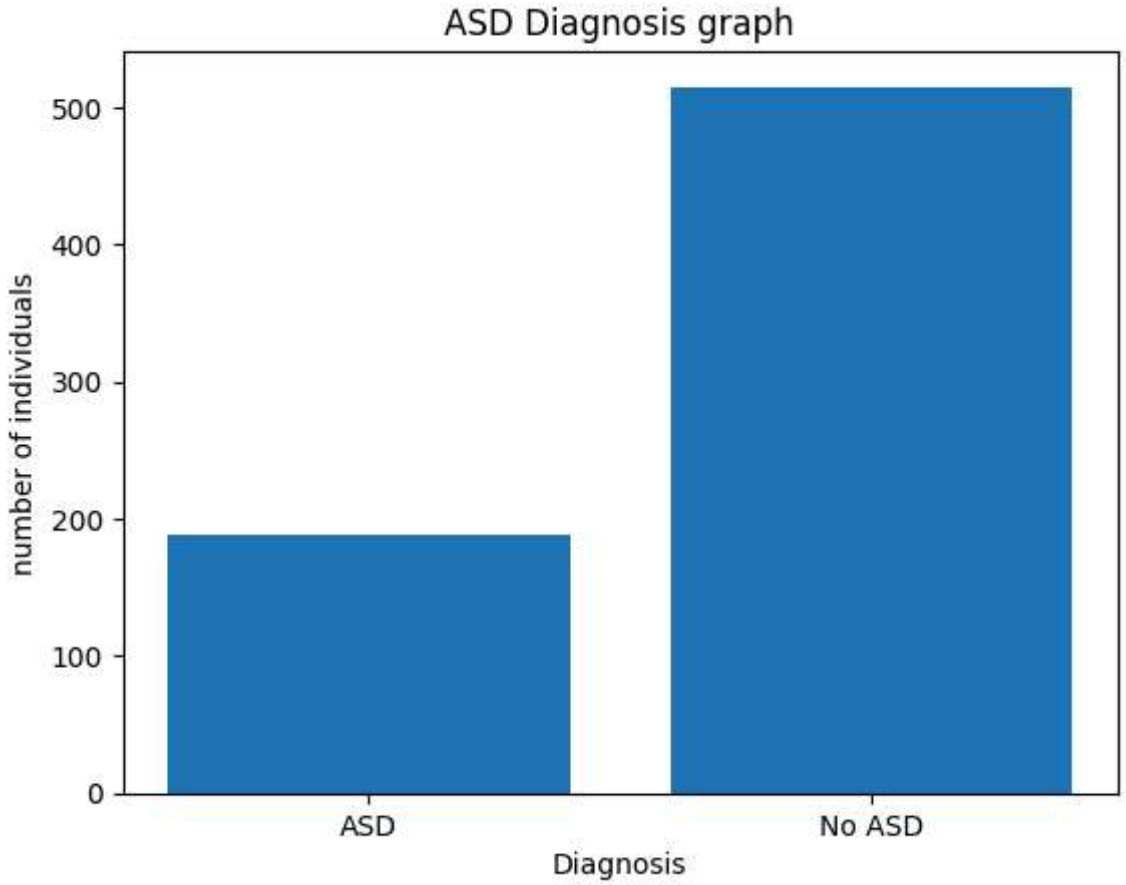
Out[29]:
```
no     692
yes     12
Name: used_app_before, dtype: int64
```

In [30]:
```
value_counts = data['used_app_before'].value_counts()
plt.figure(figsize=(3,3))
plt.pie(value_counts, labels=value_counts.index, autopct='%1.1f%%', startangle=140)
```

```
plt.title('"used_app_before"')
plt.axis('equal')
plt.show()
```



In [31]:
```
#Visualization for ASD diagnosis in existing data.
categories = ['ASD', 'No ASD']
values = [total_asd_yes, total_asd_no]
plt.bar(categories, values)
plt.xlabel('Diagnosis')
plt.ylabel('number of individuals')
plt.title('ASD Diagnosis graph')
plt.show()
```



Visualization with seaborn and matplotlib

Before data can be used as input for machine learning algorithms, it must be cleaned, formatted, and maybe even restructured — this is typically known as preprocessing. This preprocessing can help tremendously with the outcome and predictive power of nearly all learning algorithms.

In [32]:
```
#saving data after preprocessing into csv file.
data.to_csv('preprocessed.csv', index=False)
c_data= pd.read_csv('preprocessed.csv')
c_data.head(5)
```

Out[32]:

	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score	...	gender	ethnicity	jundic
0	1	1	1	1	0	0	1	1	0	0	...	f	White-European	n
1	1	1	0	1	0	0	0	1	0	1	...	m	Latino	n
2	1	1	0	1	1	0	1	1	1	1	...	m	Latino	ye
3	1	1	0	1	0	0	1	1	0	1	...	f	White-European	n
4	1	0	0	0	0	0	0	1	0	0	...	f	Others	n

5 rows × 21 columns

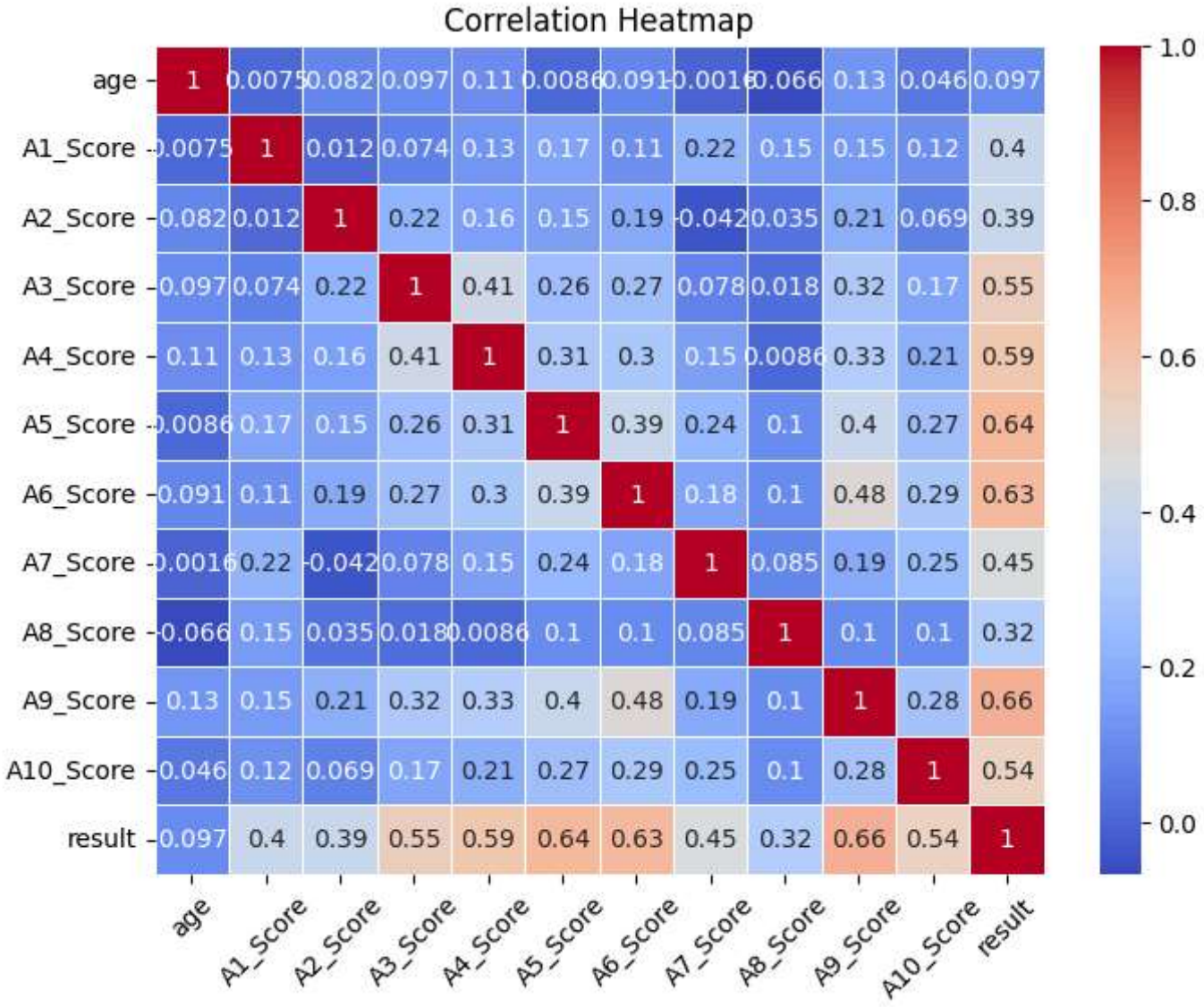
In [33]:
```
c_data.describe() # describing preprocessed data
```

Out[33]:

	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score	age
count	704.000000	704.000000	704.000000	704.000000	704.000000	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	0.417614	0.649148	0.323864	0.573864	29.198864
std	0.448535	0.498152	0.498535	0.500337	0.500353	0.451301	0.493516	0.477576	0.468281	0.494866	9.691850
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	17.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	21.000000
50%	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.000000	27.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	35.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	64.000000

Let's check out the data types of all our features including the target feature. Moreover, lets count the total number of instances and the target-class distribution.

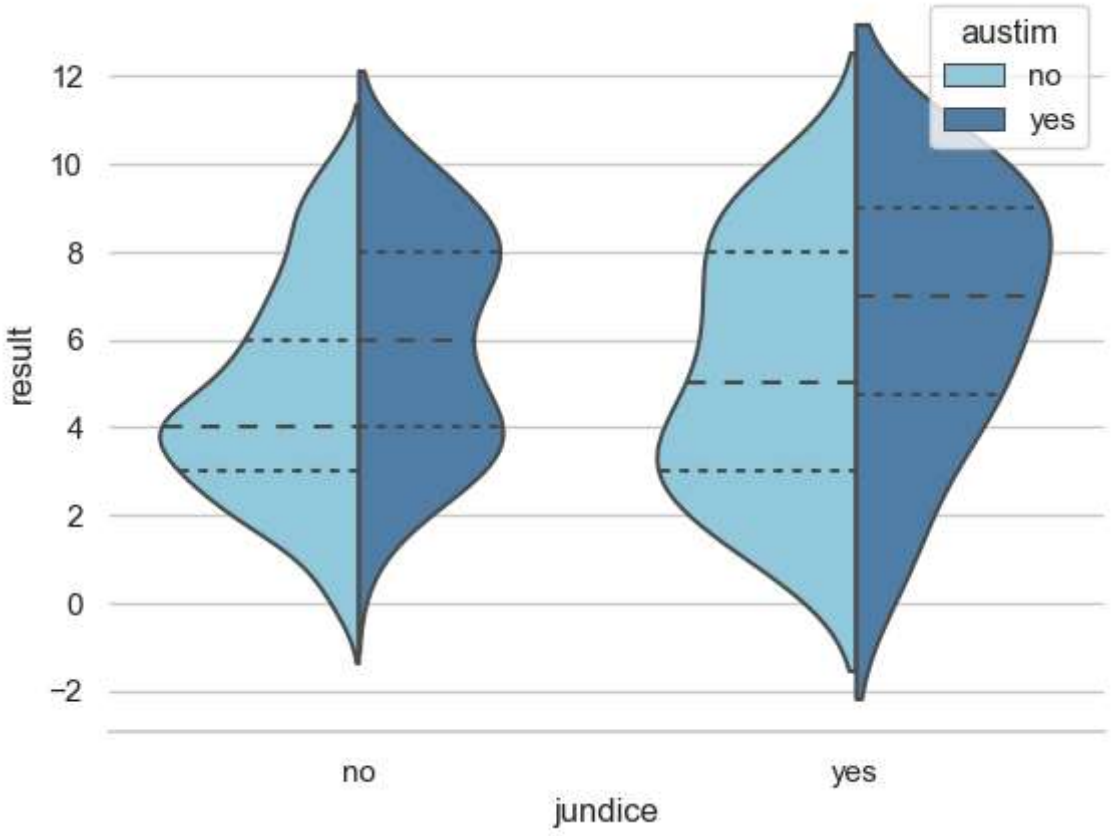
In [34]:
```
### Finding co relation between the features so we can elimanate the features which are highly co related and does not contrib
c_data_correction= c_data[['age','A1_Score','A2_Score','A3_Score','A4_Score','A5_Score','A6_Score','A7_Score','A8_Score',
                      'A9_Score','A10_Score', 'result']]
correlation_matrix = c_data_correction.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.show()
```



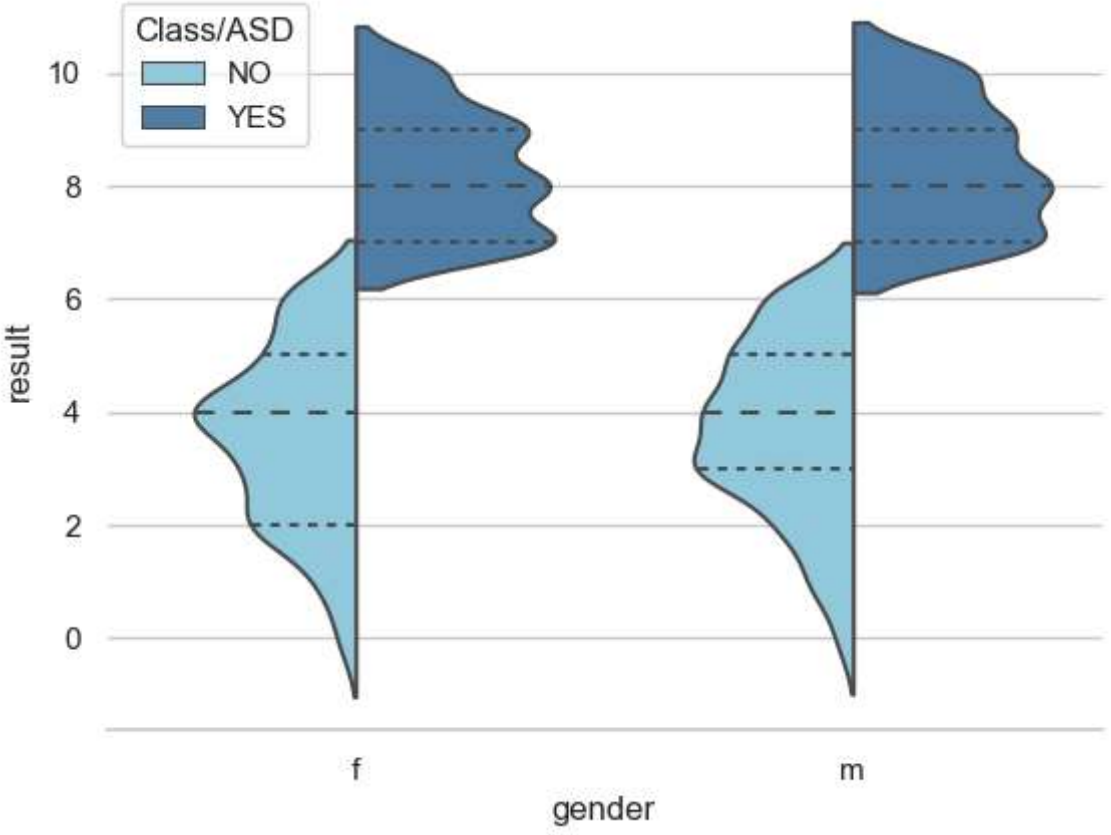
In [35]:
```
#identifying data types
print(c_data.dtypes)
```

```
A1_Score             int64
A2_Score             int64
A3_Score             int64
A4_Score             int64
A5_Score             int64
A6_Score             int64
A7_Score             int64
A8_Score             int64
A9_Score             int64
A10_Score            int64
age                float64
gender              object
ethnicity           object
jundice             object
austim              object
contry_of_res       object
used_app_before     object
result             float64
age_desc            object
relation            object
Class/ASD           object
dtype: object
```

In [36]:
```
#visualizing results scores corresponding to jundice and autism.
sns.set(style="whitegrid", color_codes=True)
sns.violinplot(x="jundice", y="result", hue="austim", data=c_data, split=True,
               inner="quart", palette={'yes': "steelblue", 'no': "skyblue"})
sns.despine(left=True)
```



In [37]:
```
#visualizing results scores corresponding to gender and Class/ASD
sns.violinplot(x="gender", y="result", hue="Class/ASD", data=c_data, split=True,
               inner="quart", palette={'YES': "steelblue", 'NO': "skyblue"})
sns.despine(left=True)
```



In [38]:
```
#visualizing results scores corresponding to relation and Class/ASD
sns.violinplot(x="relation", y="result", hue="Class/ASD", data=c_data, split=True,
               inner="quart", palette={'YES': "steelblue", 'NO': "skyblue"})
sns.despine(left=True)
```

Feature Engineering

In [39]:
```
#identifying relevent features based on visualizations and data analysis.
asd_raw = c_data['Class/ASD']
features_raw = c_data[['age', 'gender', 'jundice', 'contry_of_res', 'relation','austim',
                       'A1_Score','A2_Score','A3_Score','A4_Score','A5_Score','A6_Score','A7_Score','A8_Score',
                       'A9_Score','A10_Score']]
```

One-Hot-Coding for non-numeric features

In [40]:
```
#handling and scaling non numeric features.
def min_max_scaling(series):
    min_val = series.min()
    max_val = series.max()
    scaled_series = (series - min_val) / (max_val - min_val)
    return scaled_series
features_minmax_transform = features_raw.copy()
features_minmax_transform['age'] = min_max_scaling(features_raw['age'])
print(features_minmax_transform.head(5))
```

```
        age gender jundice  contry_of_res relation austim  A1_Score  A2_Score  \
0  0.191489      f      no  United States     Self     no         1         1
1  0.148936      m      no         Brazil     Self    yes         1         1
2  0.212766      m     yes          Spain   Parent    yes         1         1
3  0.382979      f      no  United States     Self    yes         1         1
4  0.489362      f      no          Egypt   Others     no         1         0

   A3_Score  A4_Score  A5_Score  A6_Score  A7_Score  A8_Score  A9_Score  \
0         1         1         0         0         1         1         0
1         0         1         0         0         0         1         0
2         0         1         1         0         1         1         1
3         0         1         0         0         1         1         0
4         0         0         0         0         0         1         0

   A10_Score
0          0
1          1
2          1
3          1
4          0
```

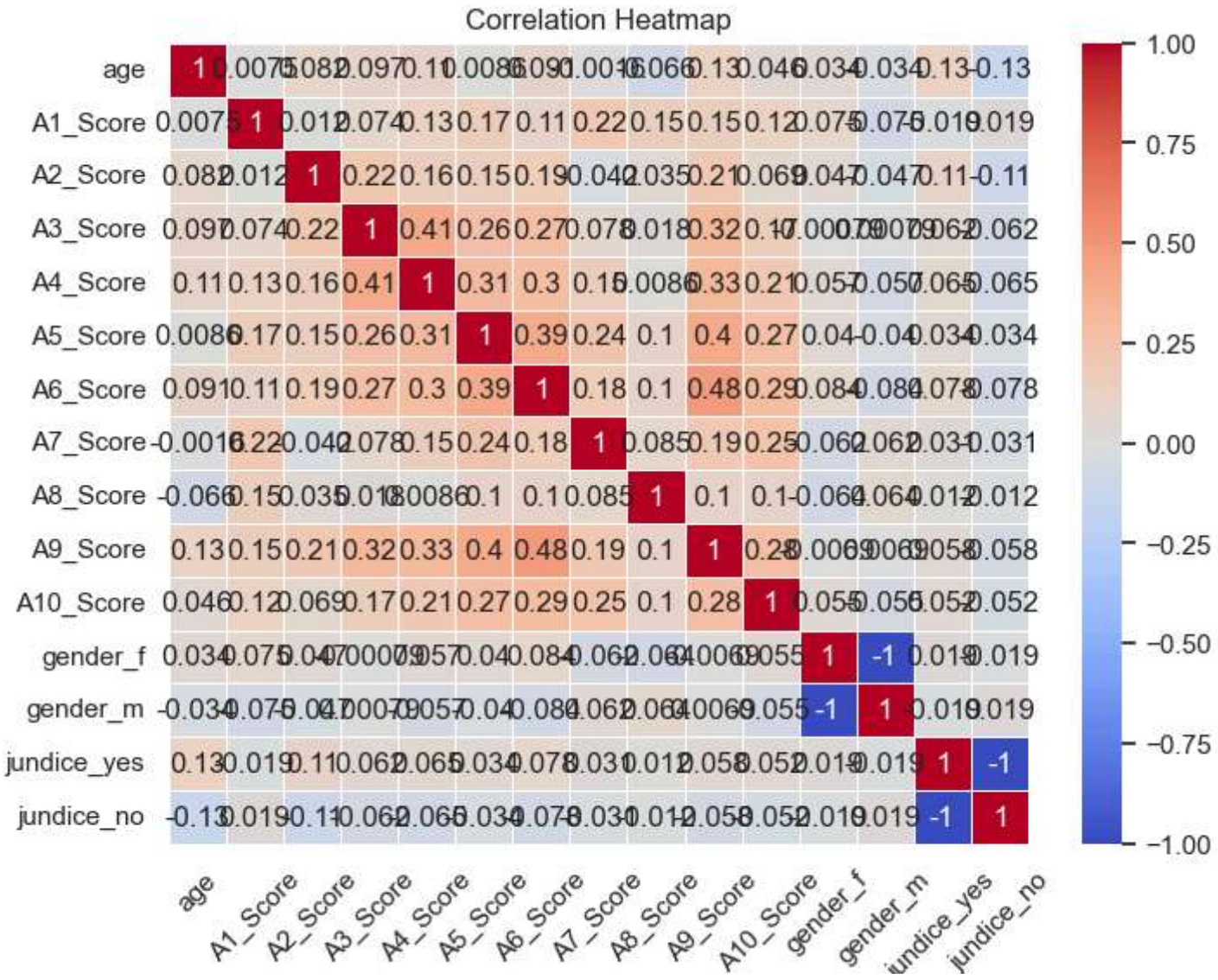
In [41]:
```
# performing one-hot encoding.
features_final = pd.get_dummies(features_minmax_transform, dtype='int')
display(features_final.head(100))
asd_classes = asd_raw.apply(lambda x: 1 if x == 'YES' else 0)
encoded = list(features_final.columns)
print ("total features after one-hot encoding.".format(len(encoded)))
print (encoded)

### Finding co relation between the features so we can elimanate the features which are highly co related and does not contrib
```

```
c_data_correction= features_final[['age','A1_Score','A2_Score','A3_Score','A4_Score','A5_Score','A6_Score','A7_Score','A8_Scor
                      'A9_Score','A10_Score', 'gender_f', 'gender_m', 'jundice_yes', 'jundice_no']]
correlation_matrix = c_data_correction.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.show()
```

	age	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	...	contry_of_res_United States	contry
0	0.191489	1	1	1	1	0	0	1	1	0	...	1	
1	0.148936	1	1	0	1	0	0	0	1	0	...	0	
2	0.212766	1	1	0	1	1	0	1	1	1	...	0	
3	0.382979	1	1	0	1	0	0	1	1	0	...	1	
4	0.489362	1	0	0	0	0	0	0	1	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
95	0.425532	1	1	0	0	0	0	0	1	0	...	0	
96	0.234043	1	0	0	0	1	0	1	1	0	...	0	
97	0.106383	1	0	0	1	1	0	1	0	0	...	0	
98	0.191489	1	1	0	0	0	0	0	1	0	...	0	
99	0.085106	0	0	0	0	0	0	0	0	1	...	0	

100 rows × 89 columns

```
total features after one-hot encoding.
['age', 'A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Scor
e', 'gender_f', 'gender_m', 'jundice_no', 'jundice_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_re
s_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_r
es_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_Kazakhsta
n', 'contry_of_res_Lebanon', 'contry_of_res_Malaysia', 'contry_of_res_Mexico', 'contry_of_res_Nepal', 'contry_of_res_Netherland
s', 'contry_of_res_New Zealand', 'contry_of_res_Nicaragua', 'contry_of_res_Niger', 'contry_of_res_Oman', 'contry_of_res_Pakista
n', 'contry_of_res_Philippines', 'contry_of_res_Portugal', 'contry_of_res_Romania', 'contry_of_res_Russia', 'contry_of_res_Saud
i Arabia', 'contry_of_res_Serbia', 'contry_of_res_Sierra Leone', 'contry_of_res_South Africa', 'contry_of_res_Spain', 'contry_o
f_res_Sri Lanka', 'contry_of_res_Sweden', 'contry_of_res_Tonga', 'contry_of_res_Turkey', 'contry_of_res_Ukraine', 'contry_of_re
s_United Arab Emirates', 'contry_of_res_United Kingdom', 'contry_of_res_United States', 'contry_of_res_Uruguay', 'contry_of_res
_Viet Nam', 'relation_Health care professional', 'relation_Others', 'relation_Parent', 'relation_Relative', 'relation_Self', 'a
ustim_no', 'austim_yes']
```



Spliting the Data

In [42]:
```
#splitting data into train and test sets.
def data_spliting( X , y) :
```

```
    if isinstance(X, pd.Series):
        X = X.to_frame()  # Convert Series to DataFrame
    if isinstance(y, pd.DataFrame) and y.shape[1] == 1:
        y = y.iloc[:, 0]  # Convert DataFrame to Series

    X = (X - X.mean()) / X.std()
    split_index = int(0.50 * len(X))
    X_train, X_test = X.iloc[:split_index], X.iloc[split_index:]
    y_train, y_test = y.iloc[:split_index], y.iloc[split_index:]
    return X_train, X_test, y_train, y_test
```

In [43]:
```
#performance metric calculations.
def calculate_metrics(y_true, y_pred):
    y_pred = np.array(y_pred)  # Convert list to numpy array
    tp = np.sum((y_true == 1) & (y_pred == 1))
    tn = np.sum((y_true == 0) & (y_pred == 0))
    fp = np.sum((y_true == 0) & (y_pred == 1))
    fn = np.sum((y_true == 1) & (y_pred == 0))
    TPR = tp / (tp + fn)
    FPR = fp / (fp + tn)
    precision = tp / (tp + fp) if (tp + fp) else 0
    recall = tp / (tp + fn) if (tp + fn) else 0
    f1_score = 2 * (precision * recall) / (precision + recall) if (precision + recall) else 0
    confusion_matrix = np.array([[tp, fp], [fn, tn]])
    return precision, recall, f1_score, confusion_matrix, tp, fp
```

Models Implementation

Logistic Regression without cross-validation

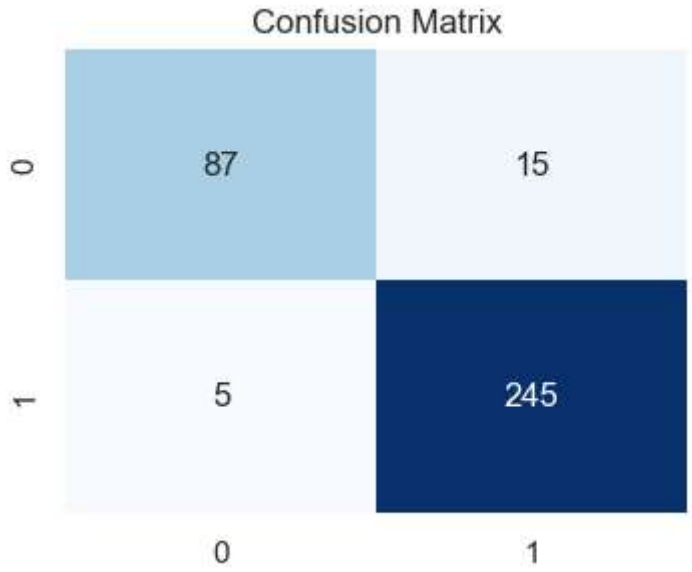
In [44]:
```
import numpy as np
import pandas as pd

class LogisticRegression:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.weights = None
```

```
        self.bias = None

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        num_samples, num_features = X.shape
        self.weights = np.zeros(num_features)
        self.bias = 0

        for _ in range(self.num_iterations):
            linear_model = np.dot(X, self.weights) + self.bias
            y_predicted = self.sigmoid(linear_model)

            dw = (1 / num_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / num_samples) * np.sum(y_predicted - y)

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self.sigmoid(linear_model)
        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
        return y_predicted_cls

#splitting data
X_train, X_test, y_train, y_test = data_spliting(features_final, asd_classes)

#logistic regression
model = LogisticRegression(learning_rate=0.1, num_iterations=1000)
#training
model.fit(X_train, y_train)

#predicting
predictions = model.predict(X_test)

#metric results
precision, recall, f1_score, confusion_matrix, tp, fp = calculate_metrics(y_test, predictions)
logistic_regression_accuracy = np.mean(predictions == y_test)
print("Logistic Regression without cross validation:")
```

```
print("Accuracy:", logistic_regression_accuracy)
print("F1 Score:", f1_score)
print("Confusion Matrix:\n", confusion_matrix)
```

```
Logistic Regression without cross validation:
Accuracy: 0.9431818181818182
F1 Score: 0.8969072164948452
Confusion Matrix:
 [[ 87  15]
 [  5 245]]
```

In [45]:
```
#confusion matrix for logistic regression visualization
plt.figure(figsize=(4, 3))
plt.title('Confusion Matrix')
sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.show()
```



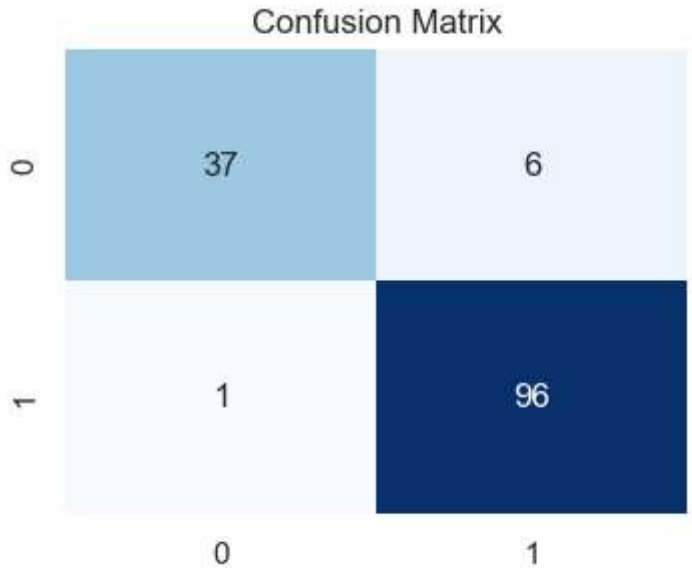
Logistic Regression with cross-validation

In [46]:
```
import numpy as np
import pandas as pd
```

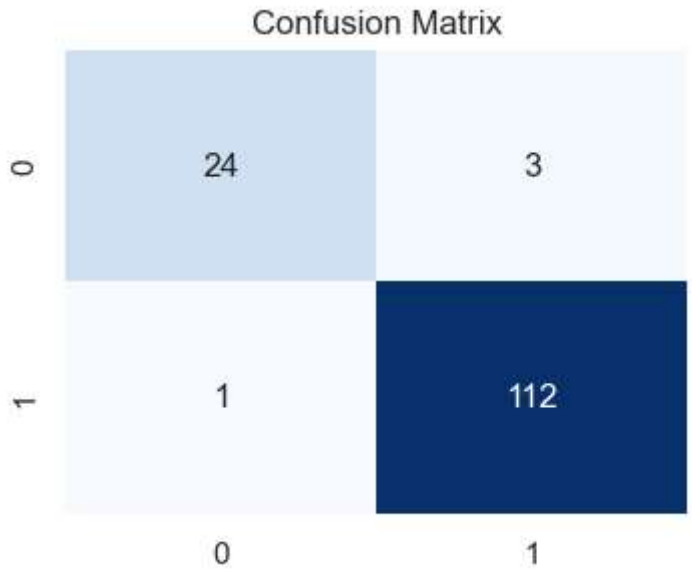
```
class LogisticRegression:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.weights = None
        self.bias = None

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        num_samples, num_features = X.shape
        self.weights = np.zeros(num_features)
        self.bias = 0

        for _ in range(self.num_iterations):
            linear_model = np.dot(X, self.weights) + self.bias
            y_predicted = self.sigmoid(linear_model)

            dw = (1 / num_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / num_samples) * np.sum(y_predicted - y)

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self.sigmoid(linear_model)
        return [1 if i > 0.5 else 0 for i in y_predicted]

metrics = {
        "log_cv_mean_accuracy": [],
        "f1_score": [],
        "confusion_matrix": []
    }

def k_fold_cross_validation(X, y, k=5):
    fold_size = len(X) // k
    indices = np.random.permutation(len(X))
    logistic_regression_cv_accuracy = []
    for i in range(k):
```

```
        test_indices = indices[i*fold_size:(i+1)*fold_size]
        train_indices = np.concatenate([indices[:i*fold_size], indices[(i+1)*fold_size:]])

        X_train, X_test = X.iloc[train_indices], X.iloc[test_indices]
        y_train, y_test = y.iloc[train_indices], y.iloc[test_indices]

        model = LogisticRegression(learning_rate=0.1, num_iterations=1000)
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)

        acc = np.mean(predictions == y_test)
        prec, rec, f1, confusion_matrix, TPR, FPR = calculate_metrics(y_test, predictions)

        metrics["log_cv_mean_accuracy"] = (acc)
        metrics["f1_score"] = (f1)
        metrics["confusion_matrix"] = (confusion_matrix)
        logistic_regression_cv_accuracy.append(acc)
        print("Result of Logistic Regression with 5-Fold Cross-Validation:", metrics)
        plt.figure(figsize=(4, 3))
        plt.title('Confusion Matrix')
        sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
        plt.show()


    return metrics, np.mean(logistic_regression_cv_accuracy)

X, y = features_final, asd_classes

#converting series to dataframe
if isinstance(X, pd.Series):
    X = X.to_frame()

#converting dataframe to series
if isinstance(y, pd.DataFrame) and y.shape[1] == 1:
    y = y.iloc[:, 0]

X = (X - X.mean()) / X.std()

metrics, logistic_regression_cv_accuracy = k_fold_cross_validation(X, y, k=5)
print("Mean Accuracy of Logistic Regression with K fold cross validation:", (logistic_regression_cv_accuracy))
```

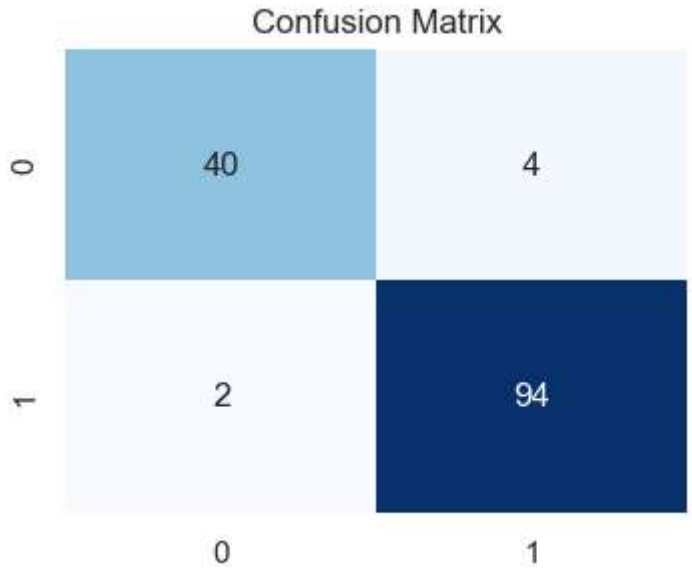
```
Result of Logistic Regression with 5-Fold Cross-Validation: {'log_cv_mean_accuracy': 0.95, 'f1_score': 0.9135802469135803, 'con
fusion_matrix': array([[37,  6],
       [ 1, 96]], dtype=int64)}
```



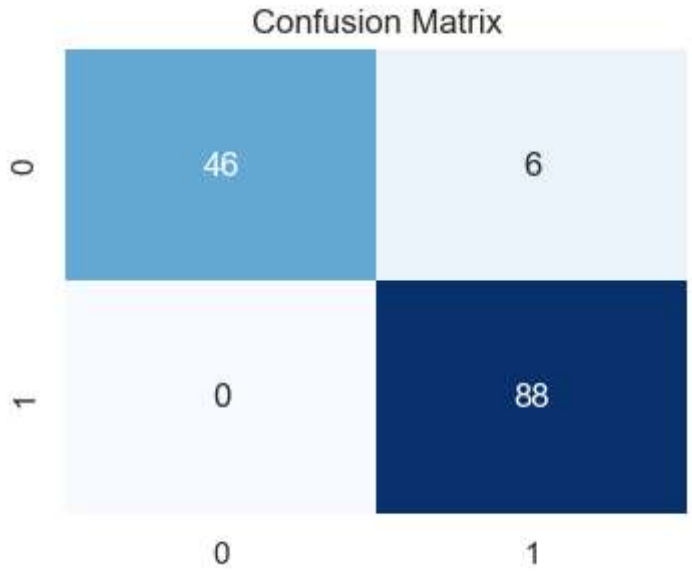

```
Result of Logistic Regression with 5-Fold Cross-Validation: {'log_cv_mean_accuracy': 0.9714285714285714, 'f1_score': 0.92307692
3076923, 'confusion_matrix': array([[ 24,   3],
       [  1, 112]], dtype=int64)}
```



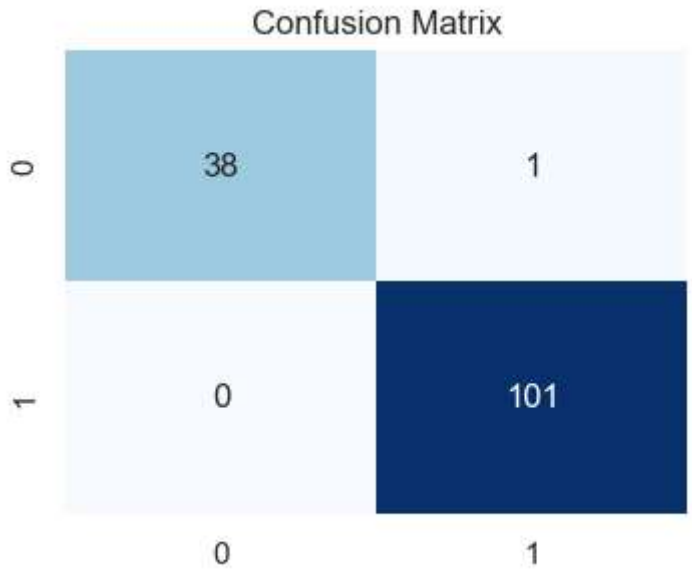

```
Result of Logistic Regression with 5-Fold Cross-Validation: {'log_cv_mean_accuracy': 0.9571428571428572, 'f1_score': 0.93023255
81395349, 'confusion_matrix': array([[40,  4],
       [ 2, 94]], dtype=int64)}
```



```
Result of Logistic Regression with 5-Fold Cross-Validation: {'log_cv_mean_accuracy': 0.9571428571428572, 'f1_score': 0.93877551
02040816, 'confusion_matrix': array([[46,  6],
       [ 0, 88]], dtype=int64)}
```




```
Result of Logistic Regression with 5-Fold Cross-Validation: {'log_cv_mean_accuracy': 0.9928571428571429, 'f1_score': 0.98701298
70129869, 'confusion_matrix': array([[ 38,   1],
       [  0, 101]], dtype=int64)}
```




```
Mean Accuracy of Logistic Regression with K fold cross validation: 0.9657142857142859
```

Decision Tree without Cross-Validation

In [47]:

```python
import numpy as np
import pandas as pd

class DecisionTree:
    def __init__(self, max_depth=None):
        self.max_depth = max_depth

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def _entropy(self, y):
        _, counts = np.unique(y, return_counts=True)
        probabilities = counts / len(y)
        entropy = -np.sum(probabilities * np.log2(probabilities))
        return entropy

    def _split(self, X, y, feature_index, threshold):
        left_mask = X.iloc[:, feature_index] <= threshold
```

```
        right_mask = ~left_mask
        return X[left_mask], X[right_mask], y[left_mask], y[right_mask]

    def _find_best_split(self, X, y):
        best_entropy = np.inf
        best_feature_index = None
        best_threshold = None

        for feature_index in range(X.shape[1]):
            thresholds = np.unique(X.iloc[:, feature_index])
            for threshold in thresholds:
                X_left, X_right, y_left, y_right = self._split(X, y, feature_index, threshold)

                if len(y_left) == 0 or len(y_right) == 0:
                    continue

                total_instances = len(y_left) + len(y_right)
                left_weight = len(y_left) / total_instances
                right_weight = len(y_right) / total_instances

                entropy = (left_weight * self._entropy(y_left)) + (right_weight * self._entropy(y_right))

                if entropy < best_entropy:
                    best_entropy = entropy
                    best_feature_index = feature_index
                    best_threshold = threshold

        return best_feature_index, best_threshold

    def _build_tree(self, X, y, depth):
        if depth == 0 or len(np.unique(y)) == 1:
            return np.bincount(y).argmax()

        best_feature_index, best_threshold = self._find_best_split(X, y)

        if best_feature_index is None:
            return np.bincount(y).argmax()

        X_left, X_right, y_left, y_right = self._split(X, y, best_feature_index, best_threshold)

        left_subtree = self._build_tree(X_left, y_left, depth - 1)
```

```
        right_subtree = self._build_tree(X_right, y_right, depth - 1)

        return (best_feature_index, best_threshold, left_subtree, right_subtree)

    def fit(self, X, y):
        self.tree = self._build_tree(X, y, self.max_depth)

    def _predict_instance(self, instance, tree):
        if isinstance(tree, np.int64):
            return tree

        feature_index, threshold, left_subtree, right_subtree = tree
        if instance.iloc[feature_index] <= threshold:  # Use iloc for position-based indexing
            return self._predict_instance(instance, left_subtree)
        else:
            return self._predict_instance(instance, right_subtree)

    def predict(self, X):
        return np.array([self._predict_instance(instance, self.tree) for _, instance in X.iterrows()])


X_train, X_test, y_train, y_test = data_spliting(features_final, asd_classes)

dt = DecisionTree(max_depth=2)
dt.fit(X_train, y_train)

predictions = dt.predict(X_test)

def accuracy(y_true, y_pred):
    correct = sum(y_true == y_pred)
    return correct / len(y_true)

decision_tree_accuracy_value = accuracy(y_test, predictions)
precision, recall, f1_score, confusion_matrix,TPR, FPR = calculate_metrics(y_test, predictions)

print("Result of Decision tree without cross validation:")
print("Accuracy:", decision_tree_accuracy_value)
print("F1 Score:", f1_score)
plt.figure(figsize=(4, 3))
plt.title('Confusion Matrix')
```

```
sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.show()
```

```
Result of Decision tree without cross validation:
Accuracy: 0.8295454545454546
F1 Score: 0.625
```

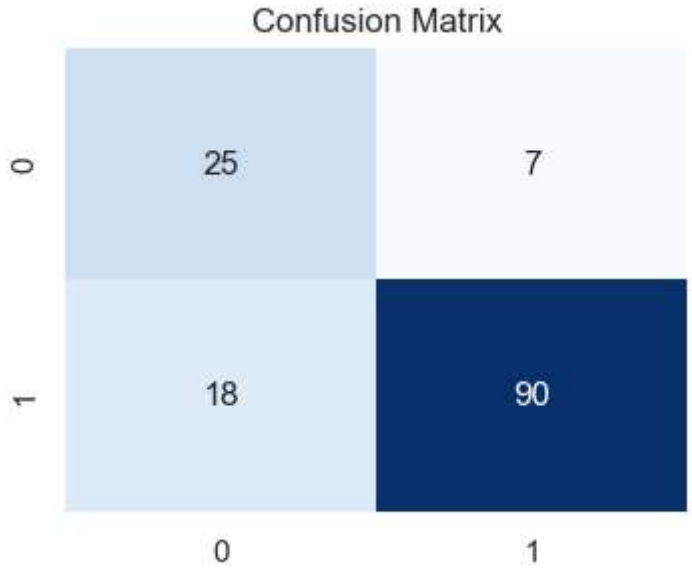
Decision Tree With Cross-Validation

In [48]:

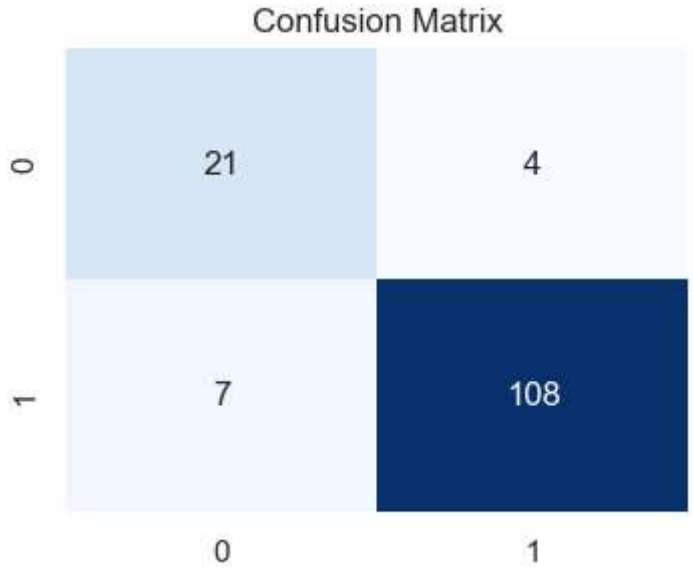
```
import numpy as np
k = 5
data = pd.concat([features_final, asd_classes], axis=1)
data_shuffled = data.sample(frac=1, random_state=0).reset_index(drop=True)
fold_size = len(data) // k
fold_indices = [range(i * fold_size, (i + 1) * fold_size) for i in range(k)]

accuracy_scores = []
for i in range(k):
    validation_indices = fold_indices[i]
    training_indices = [idx for idx in range(len(data)) if idx not in validation_indices]
    X_train = data_shuffled.iloc[training_indices, :-1]
    y_train = data_shuffled.iloc[training_indices, -1]
```

```
    X_test = data_shuffled.iloc[validation_indices, :-1]
    y_test = data_shuffled.iloc[validation_indices, -1]
    dt_classifier = DecisionTree(max_depth=2)
    dt_classifier.fit(X_train, y_train)
    predictions = dt_classifier.predict(X_test)
    accuracy = np.mean(predictions == y_test)
    precision, recall, f1_score, confusion_matrix,TP, FP = calculate_metrics(y_test, predictions)
    print("confusion_matrix:", confusion_matrix)
    plt.figure(figsize=(4, 3))
    plt.title('Confusion Matrix')
    sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.show()
    accuracy_scores.append(accuracy)

decision_tree_cv_accuracy = np.mean(accuracy_scores)

print("Result of Decision tree with cross validation:")
print("F1 Score:", f1_score)
print("Cross-Validation Accuracy Scores:", accuracy_scores)
print("Mean Accuracy of Decision tress with k-fold cross validation:", decision_tree_cv_accuracy),
```

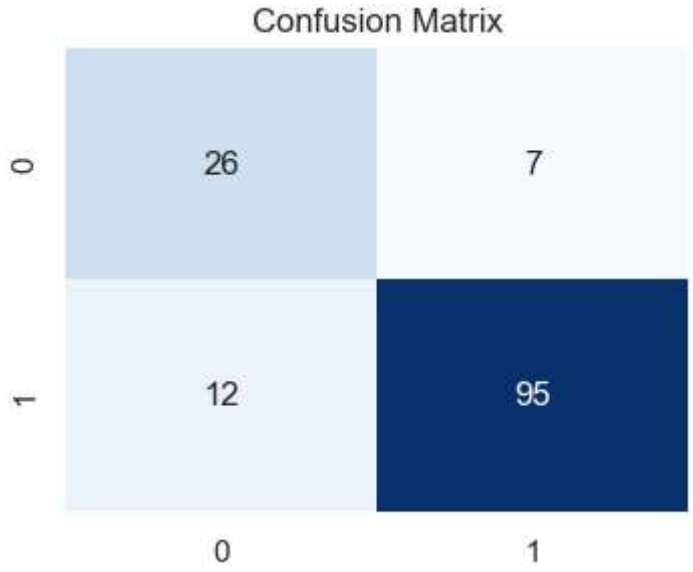
```
confusion_matrix: [[25  7]
 [18 90]]
```



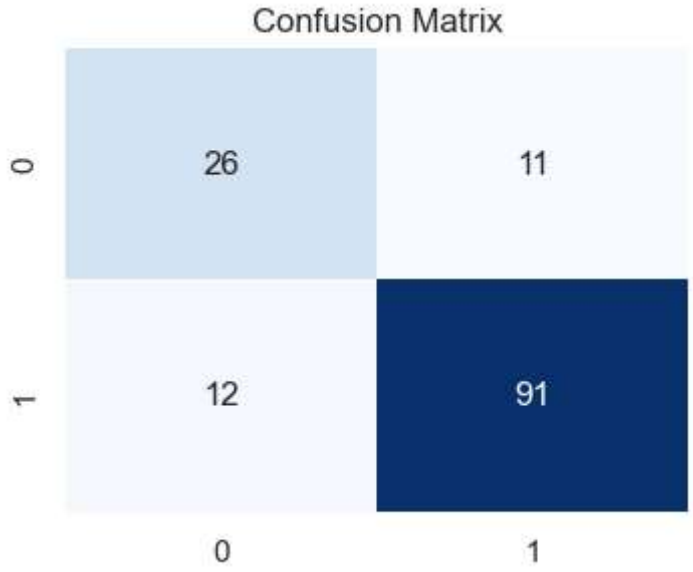
```
confusion_matrix: [[ 21   4]
 [  7 108]]
```



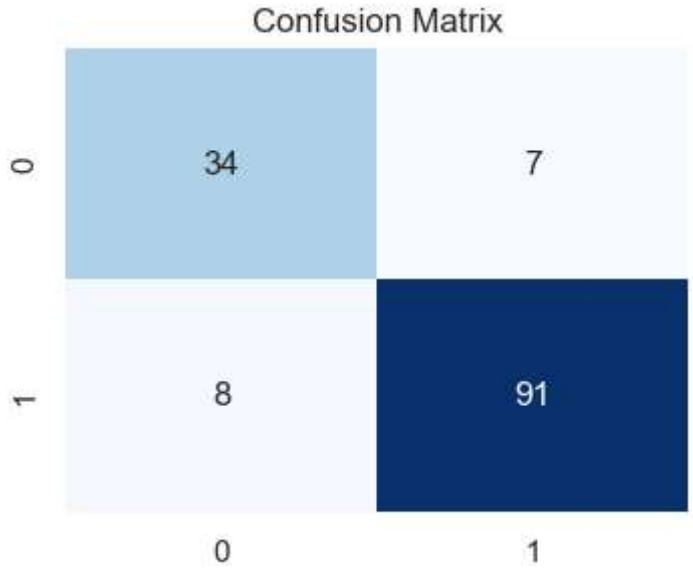

```
confusion_matrix: [[26  7]
 [12 95]]
```



```
confusion_matrix: [[26 11]
 [12 91]]
```




```
confusion_matrix: [[34  7]
 [ 8 91]]
```



```
Result of Decision tree with cross validation:
F1 Score: 0.8192771084337348
Cross-Validation Accuracy Scores: [0.8214285714285714, 0.9214285714285714, 0.8642857142857143, 0.8357142857142857, 0.8928571428
571429]
Mean Accuracy of Decision tress with k-fold cross validation: 0.8671428571428572
```

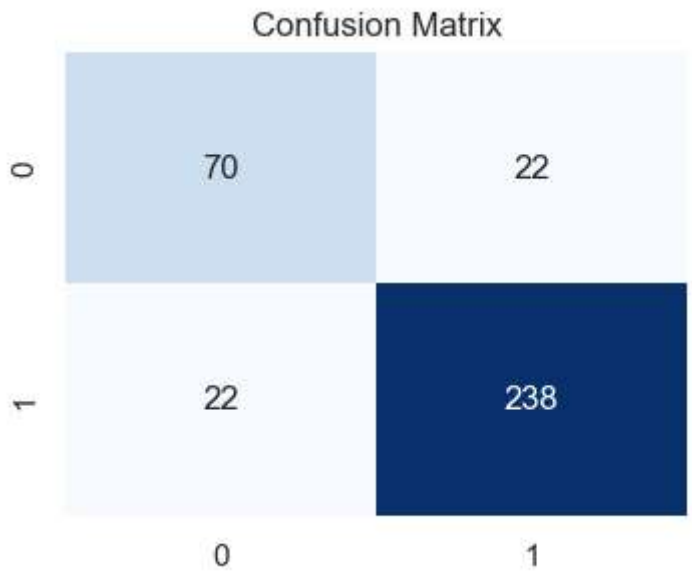
Out[48]: (None,)

Random Forest Without Cross-Validation

In [49]:
```python
import numpy as np
import pandas as pd
class RandomForest:
    def __init__(self, n_trees=None, max_depth=None, sample_size=None, n_features=None, random_state=None):
        self.n_trees = n_trees
        self.max_depth = max_depth
        self.sample_size = sample_size
        self.n_features = n_features
        self.random_state = random_state
        self.trees = []

    def fit(self, X, y):
        np.random.seed(self.random_state)
        self.trees = []
        for _ in range(self.n_trees):
            if self.sample_size is None:
                sample_size = len(X)
            else:
                sample_size = self.sample_size
            if self.n_features is None:
                n_features = X.shape[1]
            else:
                n_features = self.n_features
            indices = np.random.choice(X.index, size=sample_size, replace=True)
            sample_X = X.loc[indices].sample(n=n_features, axis=1, replace=False)
            sample_y = y.loc[indices]
            tree = DecisionTree(max_depth=self.max_depth)
            tree.fit(sample_X, sample_y)
            self.trees.append(tree)
```

```
    def predict(self, X):
        tree_preds = pd.DataFrame({i: tree.predict(X) for i, tree in enumerate(self.trees)})
        return tree_preds.mode(axis=1)[0].values

X_train, X_test, y_train, y_test = data_spliting(features_final, asd_classes)

rf = RandomForest(n_trees=20, max_depth=5, sample_size=80, n_features=int(np.sqrt(features_final.shape[1])), random_state=0)
rf.fit(X_train, y_train)
predictions_rf = rf.predict(X_test)
def accuracy(y_true, y_pred):
    correct = sum(y_true == y_pred)
    return correct / len(y_true)

accuracy_rf = accuracy(y_test, predictions_rf)
precision, recall, f1_score, confusion_matrix, TPR, FPR = calculate_metrics(y_test, predictions_rf)

print("Result Random Forest tree without cross validation:")
print("Accuracy:", accuracy_rf)
print("F1 Score:", f1_score)
plt.figure(figsize=(4, 3))
plt.title('Confusion Matrix')
sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.show()
```

```
Result Random Forest tree without cross validation:
Accuracy: 0.875
F1 Score: 0.7608695652173914
```

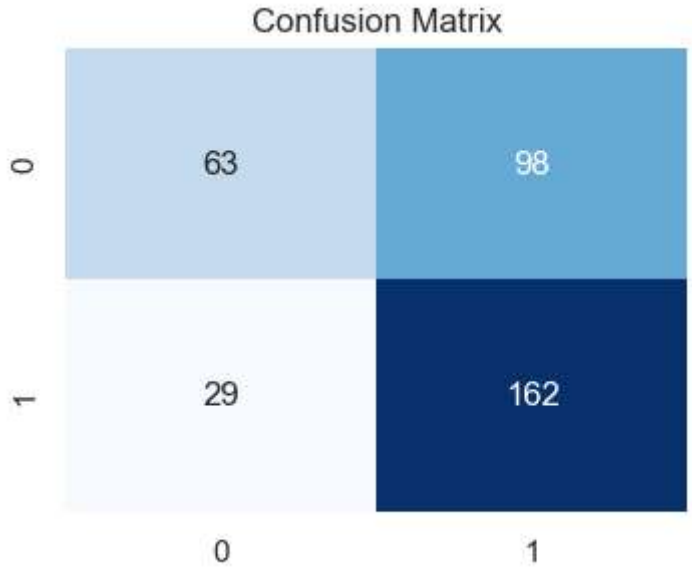


Parameter Tuning for random forest

In [50]:
```
rf = RandomForest(n_trees=8, max_depth=5, sample_size=90, n_features=int(np.sqrt(features_final.shape[1])), random_state=1)
rf.fit(X_train, y_train)

# Making predictions
predictions_rf = rf.predict(X_test)

# Calculate accuracy and other metrics using the defined functions
def accuracy(y_true, y_pred):
    correct = sum(y_true == y_pred)
    return correct / len(y_true)

rf_accuracy = accuracy(y_test, predictions_rf)
precision, recall, f1_score, confusion_matrix, TPR, FPR = calculate_metrics(y_test, predictions_rf)

print("Random Forest tree without cross validation:")
print("Accuracy:", rf_accuracy)
print("F1 Score:", f1_score)
plt.figure(figsize=(4, 3))
plt.title('Confusion Matrix')
```

```
sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.show()
```

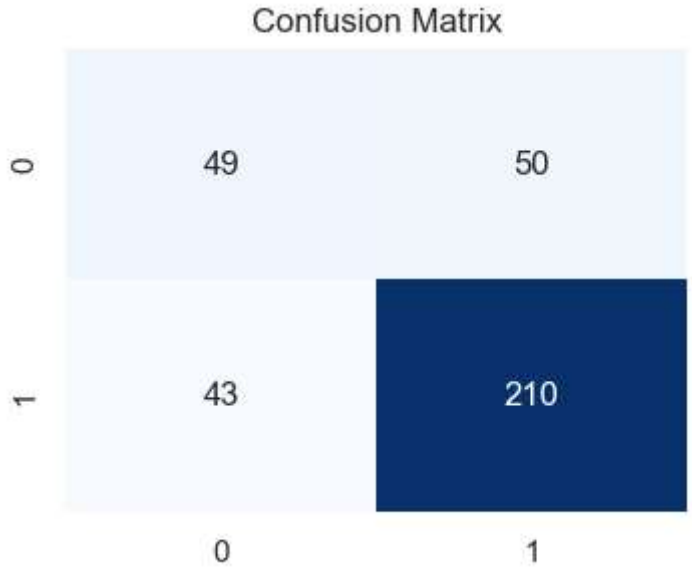
```
Random Forest tree without cross validation:
Accuracy: 0.6392045454545454
F1 Score: 0.4980237154150197
```



In [51]:
```
rf = RandomForest(n_trees=10, max_depth=3, sample_size=100, n_features=int(np.sqrt(features_final.shape[1])), random_state=1)
rf.fit(X_train, y_train)

predictions_rf = rf.predict(X_test)
def accuracy(y_true, y_pred):
    correct = sum(y_true == y_pred)
    return correct / len(y_true)

accuracy_rf = accuracy(y_test, predictions_rf)
precision, recall, f1_score, confusion_matrix, TPR, FPR = calculate_metrics(y_test, predictions_rf)

print("Random Forest Metrics:")
print("Accuracy:", accuracy_rf)
print("F1 Score:", f1_score)
plt.figure(figsize=(4, 3))
plt.title('Confusion Matrix')
```

```
sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.show()
```

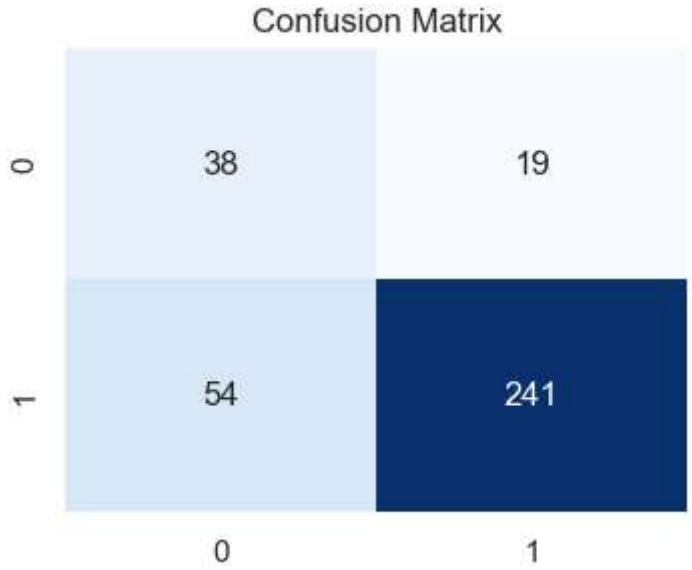
```
Random Forest Metrics:
Accuracy: 0.7357954545454546
F1 Score: 0.5130890052356021
```



In [52]:
```
rf = RandomForest(n_trees=20, max_depth=5, sample_size=100, n_features=int(np.sqrt(features_final.shape[1])), random_state=1)
rf.fit(X_train, y_train)

predictions_rf = rf.predict(X_test)
def accuracy(y_true, y_pred):
    correct = sum(y_true == y_pred)
    return correct / len(y_true)

accuracy_rf = accuracy(y_test, predictions_rf)
precision, recall, f1_score, confusion_matrix, TPR, FPR = calculate_metrics(y_test, predictions_rf)

print("Random Forest Metrics:")
print("Accuracy:", accuracy_rf)
print("F1 Score:", f1_score)
plt.figure(figsize=(4, 3))
plt.title('Confusion Matrix')
```

```
sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.show()
```

```
Random Forest Metrics:
Accuracy: 0.7926136363636364
F1 Score: 0.5100671140939598
```



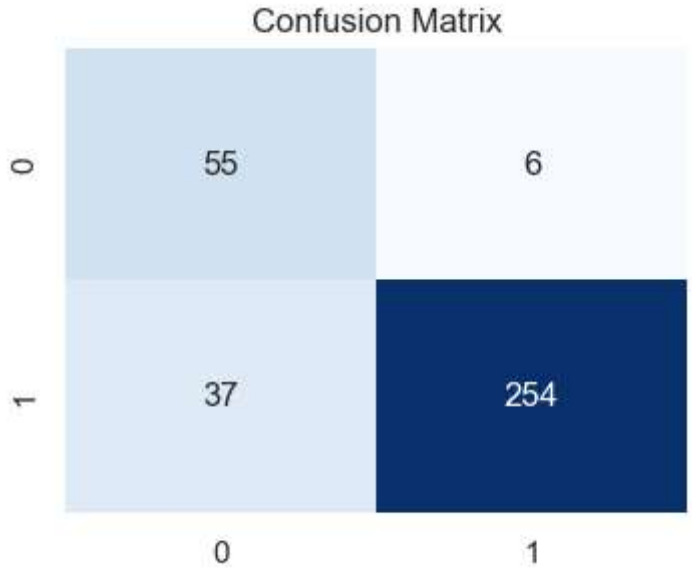
Random Forest With Cross-Validation

In [53]:
```
import numpy as np
import pandas as pd
class RandomForest:
    def __init__(self, n_trees=None, max_depth=None, sample_size=None, n_features=None, random_state=None):
        self.n_trees = n_trees
        self.max_depth = max_depth
        self.sample_size = sample_size
        self.n_features = n_features
        self.random_state = random_state
        self.trees = []

    def fit(self, X, y):
        np.random.seed(self.random_state)
```

```
        self.trees = []
        for _ in range(self.n_trees):
            if self.sample_size is None:
                sample_size = len(X)
            else:
                sample_size = self.sample_size
            if self.n_features is None:
                n_features = X.shape[1]
            else:
                n_features = self.n_features
            indices = np.random.choice(X.index, size=sample_size, replace=True)
            sample_X = X.loc[indices].sample(n=n_features, axis=1, replace=False)
            sample_y = y.loc[indices]
            tree = DecisionTree(max_depth=self.max_depth)
            tree.fit(sample_X, sample_y)
            self.trees.append(tree)

    def predict(self, X):
        tree_preds = pd.DataFrame({i: tree.predict(X) for i, tree in enumerate(self.trees)})
        return tree_preds.mode(axis=1)[0].values

X_train, X_test, y_train, y_test = data_spliting(features_final, asd_classes)

rf = RandomForest(n_trees=30, max_depth=5, sample_size=60, n_features=int(np.sqrt(features_final.shape[1])), random_state=1)
rf.fit(X_train, y_train)

predictions_rf = rf.predict(X_test)

def accuracy(y_true, y_pred):
    correct = sum(y_true == y_pred)
    return correct / len(y_true)

accuracy_cv_rf = accuracy(y_test, predictions_rf)
precision, recall, f1_score, confusion_matrix, TPR, FPR = calculate_metrics(y_test, predictions_rf)

print("Random Forest tree without cross validation:")
print("Accuracy:", accuracy_cv_rf)
print("F1 Score:", f1_score)
plt.figure(figsize=(4, 3))
plt.title('Confusion Matrix')
```

```
sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.show()
```

```
Random Forest tree without cross validation:
Accuracy: 0.8778409090909091
F1 Score: 0.7189542483660131
```

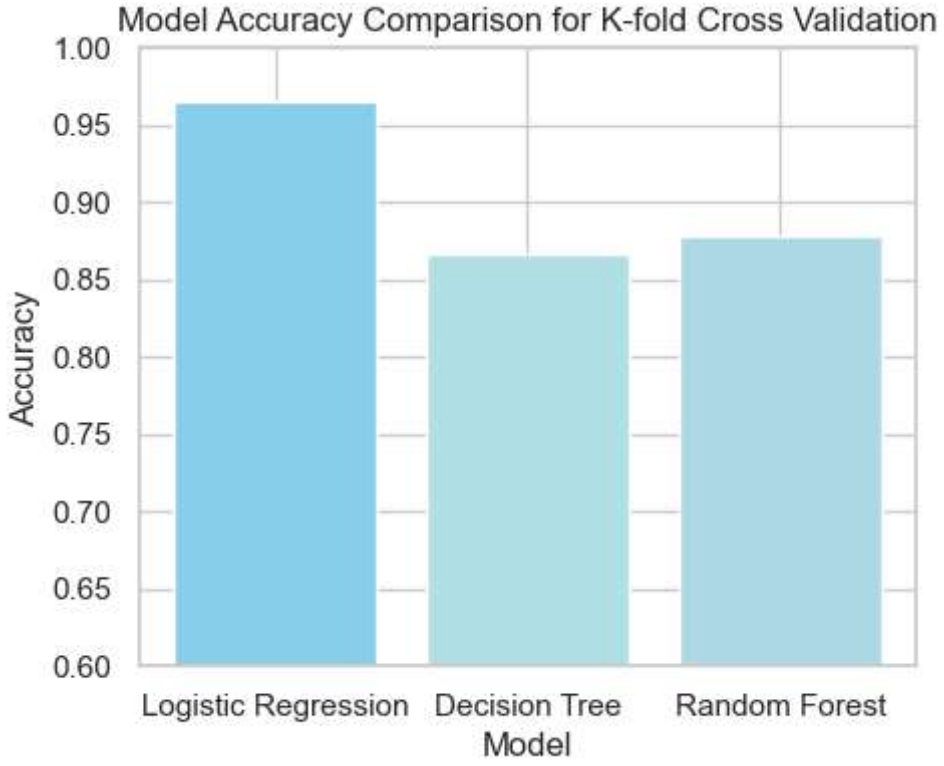


Model Accuracy Comparison

In [54]:
```
model_names = ['Logistic Regression', 'Decision Tree', 'Random Forest']
accuracies = [logistic_regression_accuracy, decision_tree_accuracy_value, accuracy_rf]

plt.figure(figsize=(5, 4))
plt.bar(model_names, accuracies, color=['skyblue', 'powderblue', 'lightblue'])
plt.title('Model Accuracy Comparison')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0.6, 1)
plt.show()
```

Model Accuracy Comparison for K-fold Cross Validation

In [55]:
```
model_names = ['Logistic Regression', 'Decision Tree', 'Random Forest']
accuracies = [logistic_regression_cv_accuracy, decision_tree_cv_accuracy, accuracy_cv_rf]  # Example accuracy values

plt.figure(figsize=(5, 4))
plt.bar(model_names, accuracies, color=['skyblue', 'powderblue', 'lightblue'])
plt.title('Model Accuracy Comparison for K-fold Cross Validation')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0.6, 1)
plt.show()
```


Model Accuracy Comparison for K-fold Cross Validation
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
Accuracy
Logistic Regression
Decision Tree
Random Forest
Model