Examining Gender & Ethnic Differences in ASD Screening Assessments

Chana Sherrington

Business Analytics and Insights

Professor David Parent

August 18, 2022

Executive Summary:

Autism (ASD) is a neurological disorder that affects 1% of the world's population. Historically, research on this disorder has not taken gender differences into account when creating screening assessments for autism. Because of this, many autistic women have been incorrectly diagnosed with ADHD, anxiety, or depression. The purpose of this analysis is to determine whether ASD assessment responses differ between males and females, as well as ethnic groups, and whether we need to create curated assessments for autism diagnosis. To answer these questions we utilized Decision Trees, Random Forests, Neural Networks, Logistic Regressions, and Association Rules. The output of our analytical models showed that there are significant differences between male and female responses to autism assessments and further differences between ethnic groups. Clinical experts should therefore create specialized screening assessments that take these differences into account to improve diagnosis accuracy.

1.0 Introduction:

1.1 Background:

Autism Spectrum Disorder (ASD) is a neurological and developmental disorder th characterized by difficulties in communication, learning, and social interaction, makingt challenging for these individuals to navigate society. These difficulties are exacerbated if the individual is unaware of their disorder. Getting a diagnosis and learning about how your brain differs from neurotypical individuals allows people to learn special techniques to deal with their problems and utilize their differences in a positive way.

1.2 Problem:

About 2.21% of adults in the U.S. have been diagnosed with this disorder, but this number is likely to be greatly underestimated as it is becoming clearer that many women with ASD have been incorrectly diagnosed with depression or anxiety, and it is theorized that autism manifests differently between males and females. According to some reports, women typically have high-functioning autism, allowing them to operate independently in the world, which is why their symptoms largely go unnoticed. Many of these women are highly successful and intelligent yet struggle to function in their everyday lives due to their undiagnosed disorders.

1.3 Objective:

This analysis aims to determine whether we can develop more accurate methods of autism screening assessment if we consider possible differences in behaviour between genders or other demographics. To answer this, we must explore the following questions:

1. Which behaviors measured in the AQ-10-Adult are most indicative of ASD?

2. Do females display different combinations of behaviors than men?
3. If there are gender differences, can they further be differentiated by ethnicity?
4. If there are no gender differences, can they still be differentiated ethnicity?

1.4 Assumptions:

The dataset for this analysis was collected from a mobile autism screening application. We must assume that users responded honestly when answering questions and providing demographic information. Because there is no ID column provided by the initial dataset, we must also assume that all records are unique, even if they may seem like duplicates.

1.5 Limitations:

Data may not be representative of the entire population, as it was collected from a downloadable app. The sample only includes individuals who can download and operate the application.

2.0 Data Sources:

2.1 Data Introduction:

This dataset contains records of ten behavioural features taken from the Autism Quotient assessment for adults (AQ-10-Adult) and ten demographic characteristics that have proven to be effective in detecting ASD cases in behavioural science.

Task: Classification

Attribute Type: Categorical, continuous and binary

Area: Medical, health and social science

Format Type: Non-Matrix

Does your data set contain missing values? Yes

Number of Records: 704

Number of Attributes (fields within each record): 21

Target Variable: Class/ASD is a binary variable. 0 indicates that the participant is unlikely to have ASD and requires no further assessment, while 1 indicates the individual may have ASD and requires further assessment.

2.2 Data Dictionary:

Attribute	Type	Role	Description
age	Number	Input	Age in years
age_desc	String	Rejected	Description of age category (18 and above)
gender	String	Input	Male or Female
ethnicity	String	Input	List of common ethnicities in text format
austim	Boolean (yes or no)	Input	Whether there is a history of ASD diagnosis in the family.
jundice	Boolean (yes or no)	Input	Whether the case was born with jaundice
Relation	String	Input	Who is competing the test (Parent, self, caregiver, medical staff, clinician ,etc.)
contry_of_residence	String	Input	List of countries in text format
Used_app_before	Boolean (yes or no)	Input	Whether the user has used a screening app
Question 1 Answer (A1_Score)	Binary (0, 1)	Input	I often notice small sounds when others do not
Question 2 Answer (A2_Score)	Binary (0, 1)	Input	I usually concentrate more on the whole picture, rather than the small details
Question 3 Answer (A3_Score)	Binary (0, 1)	Input	I find it easy to do more than one thing at once
Question 4 Answer (A4_Score)	Binary (0, 1)	Input	If there is an interruption, I can switch back to what I was doing very quickly
Question 5 Answer (A5_Score)	Binary (0, 1)	Input	I find it easy to 'read between the lines' when someone is talking to me
Question 6 Answer (A6_Score)	Binary (0, 1)	Input	I know how to tell if someone listening to me is getting bored
Question 7 Answer (A7_Score)	Binary (0, 1)	Input	When I'm reading a story I find it difficult to work out the characters' intentions
Question 8 Answer (A8_Score)	Binary (0, 1)	Input	I like to collect information about categories of things
Question 9 Answer (A9_Score)	Binary (0, 1)	Input	I find it easy to work out what someone is thinking or feeling just by looking at their face
Question 10 Answer (A10_Score)	Binary (0, 1)	Input	I find it difficult to work out people's intentions
Screening Score (Result)	Integer	Rejected	The final score obtained based on the scoring algorithm of the screening method used. This was computed in an automated manner. Score of 7+ generates 'YES' for target variable.
Class/ASD	String	Target	'YES' or 'NO' ASD Classification

2.3 Rejected Variables:

- **Age Description:** a redundant variable of the Age variable.
- **Result:** this is the participant's final score that generates the output of the target variable. It is too highly correlated with the target variable and the _Score variables to include in the analysis.

3.0 Data Exploration:

Data Exploration is the first step of data analysis in which data visualization tools and statistical techniques are used to understand the characteristics of the data.

The first step was to import the necessary packages to conduct the data exploration.

```
# Import packages
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Next, I checked the dimensions of the dataset. This dataset contains 704 records and has 21 unique variables.

```
# Check dataset dimensions
print(autism_df.shape)
```

```
(704, 21)
```

I checked the dataset column names for each variable. We can see that there are several column names incorrectly spelled (eg. jaundice and country_of_res). These will need to be corrected during the data cleansing process.

```
# Check variables
autism_df.columns
```

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

I ran the info() function to see the properties of each variable in the dataset. The majority of variables are integers or objects. There are no Null values in this dataset.

```
# Check variable properties
print(autism_df.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
None
```

Checked the number of each data type in the dataset.

```
# Count number of each data type
pd.value_counts(autism_df.dtypes)
```

```
int64      10
object      9
float64     2
dtype: int64
```

I selected the categorical variables to examine independently from the numeric variables. Half of the variables are binary except for ethnicity, country_of_res, age_desc, and relation.

```
# Check categorical variables only
autism_df.select_dtypes(exclude="number").head()
```

	gender	ethnicity	jundice	austim	contry_of_res	used_app_before	age_desc	relation	Class/ASD
0	f	White-European	no	no	United States	no	18 and more	Self	NO
1	m	Latino	no	yes	Brazil	no	18 and more	Self	NO
2	m	Latino	yes	yes	Spain	no	18 and more	Parent	YES
3	f	White-European	no	yes	United States	no	18 and more	Self	NO
4	f	?	no	no	Egypt	no	18 and more	?	NO

Next I looked at the unique values for each numeric variable. The _Score variables are binary, result has 11 unique values (0-10), and age is the only continuous variable in the dataset.

```
# Check unique values for numeric variables
autism_df.select_dtypes(include='number').nunique().sort_values()
```

```
A1_Score      2
A2_Score      2
A3_Score      2
A4_Score      2
A5_Score      2
A6_Score      2
A7_Score      2
A8_Score      2
A9_Score      2
A10_Score     2
result       11
age          46
dtype: int64
```

I used the describe() function to find the count, mean, standard deviation, minimum value, maximum value, and quartile ranges for each numeric variable. The standard deviation range is small between each variable. Looking at the ‘age’ column we can see that there are two missing values and an extreme outlier as the maximum value.

```
# Get descriptive statistics summary of dataset
autism_df.describe()
```

	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score	age	result
count	704.000000	704.000000	704.000000	704.000000	704.000000	704.000000	704.000000	704.000000	704.000000	704.000000	702.000000	704.000000
mean	0.721591	0.453125	0.457386	0.495739	0.498580	0.284091	0.417614	0.649148	0.323864	0.573864	29.698006	4.875000
std	0.448535	0.498152	0.498535	0.500337	0.500353	0.451301	0.493516	0.477576	0.468281	0.494866	16.507465	2.501493
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	17.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	21.000000	3.000000
50%	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.000000	27.000000	4.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	35.000000	7.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	383.000000	10.000000

To confirm the missing values I ran the isnull() function to check for missing values in each column. As we can see there are two missing values in the ‘age’ column. No other columns have missing variables.

```
# Check for missing values
autism_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                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
```

I checked to see if there are any records with the exact same values. There are five records that could be duplicates. However, since it is possible that two participants could have the same information and there are no identification variables we must assume that these records are unique.

```
# Check for duplicates
autism_df.duplicated().sum()
# It is possible for two different participants to have the same info so we will be treating all records as unique
```

```
5
```

Next, I examined the skewness of each variable. Age is the most skewed, which will likely be corrected once we remove the extreme maximum value. We can do nothing to reduce skewness for the other variables since they are set categorical values that cannot be transformed or scaled. The range of skewness for the numeric variables is acceptable for this analysis.

```
# Check for skewed variables
skewed = autism_df.skew(axis = 0, skipna = True)
skewed
# Age is significanlty skewed, probably an extreme outlier
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py
```

```
A1_Score     -0.990881
A2_Score      0.188732
A3_Score      0.171443
A4_Score      0.017082
A5_Score      0.005694
A6_Score      0.959556
A7_Score      0.334826
A8_Score     -0.626382
A9_Score      0.754410
A10_Score    -0.299370
age          14.204194
result        0.323436
dtype: float64
```

I checked the unique value count for our target variable 'Class/ASD'. There are significantly more records with a 'NO' classification than a 'YES' classification. If the accuracy of our models during data modeling is not good enough then we may consider upsampling to equalize the target outcomes.

```
# Count participants by diagnosis
autism_df['Class/ASD'].value_counts()
```

```
NO     515
YES    189
Name: Class/ASD, dtype: int64
```

Here is the ratio of males to females in the dataset. The distribution between the genders is pretty even with only 30 more males than females.

```
# Count participants by gender
autism_df['gender'].value_counts()
```

```
m    367
f    337
Name: gender, dtype: int64
```

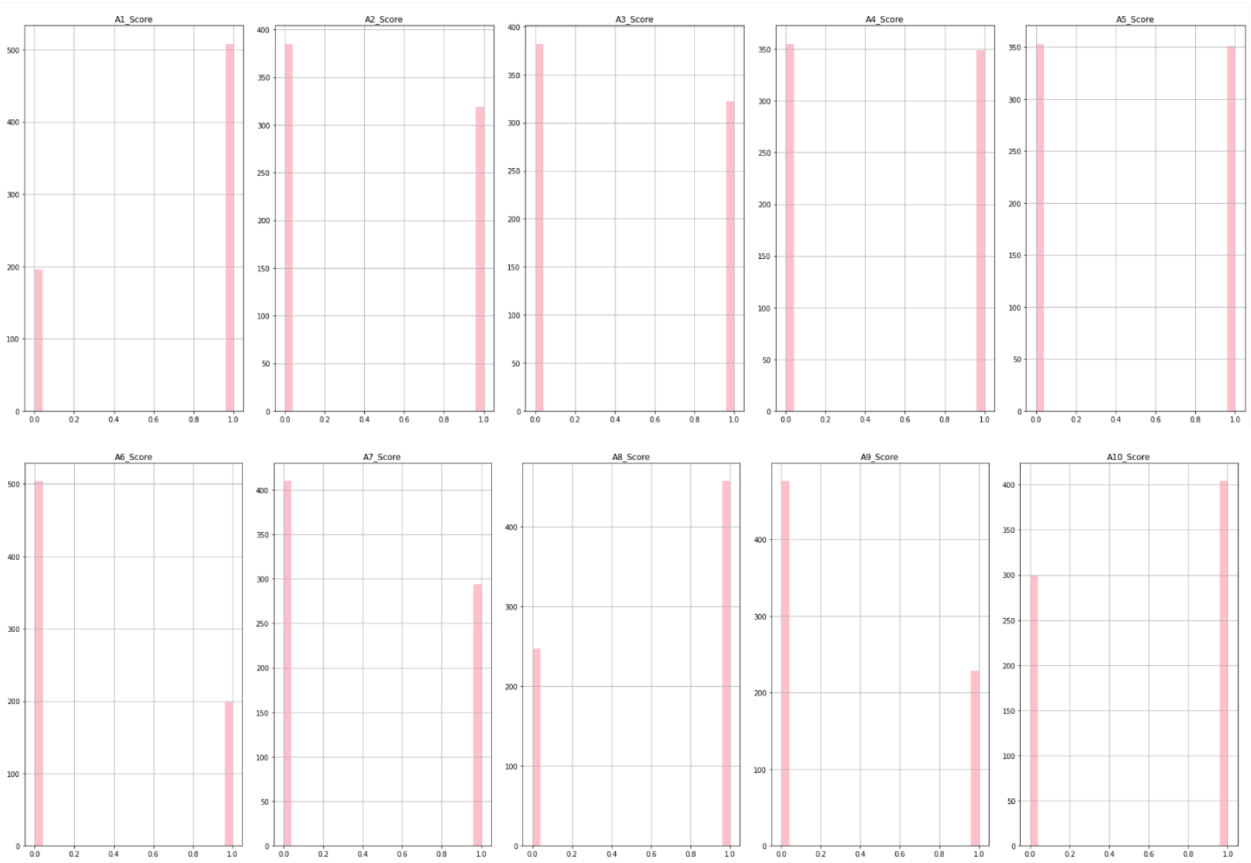
I checked the count for each unique value in the 'ethnicity' column. White-European is significantly higher than the other ethnicities. We can also see some invalid values that will need to be replaced during data cleaning (eg. '?' and 'others').

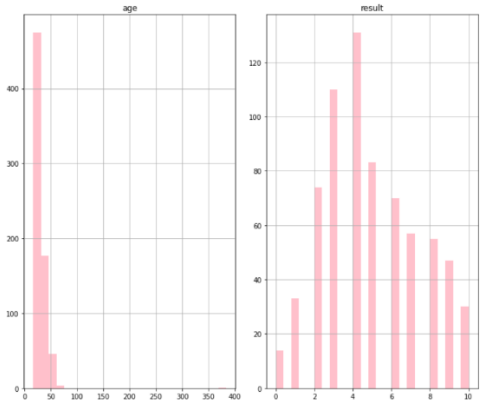
```
# Count participants by ethnicity
autism_df['ethnicity'].value_counts()
# Some invalid variables will need to be changed in data cleaning
```

```
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
```

I used a histogram plot to see the count of unique values for each numeric variable. Most responses are evenly distributed except for A1_Score, A6_Score, and A9_Score. The most common overall score in the ‘result’ column is 4.0 and the most common age range of participants is 20-30.

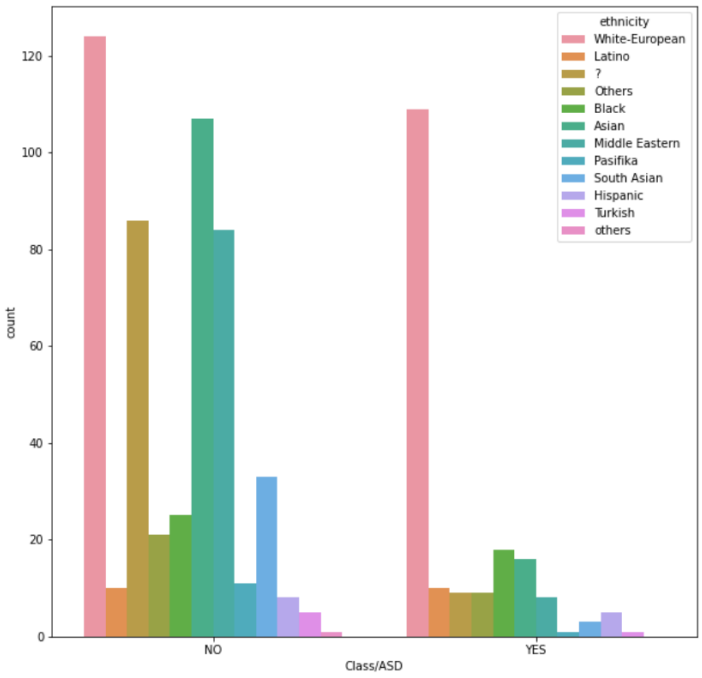
```
# Plot value count of numeric variables.
autism_df.hist(bins=25, figsize=(25, 25), layout=(-1, 5), color='pink')
plt.tight_layout()
# Most differences in responses for A1, A6, A8, and A9
```





I used a count plot to check the number of participants with a ‘YES’ or ‘NO’ classification for each ethnicity. It looks like the distribution of ‘YES’ and ‘NO’ classifications are evenly divided for each ethnicity except for Asian, Middle Eastern, and South Asian.

```
# Diagnosis by ethnicity.
plt.figure(figsize = (10, 10))
sns.countplot(x = 'Class/ASD', hue = 'ethnicity', data = autism_df)
plt.show()
# Likelihood of ethnic bias
```

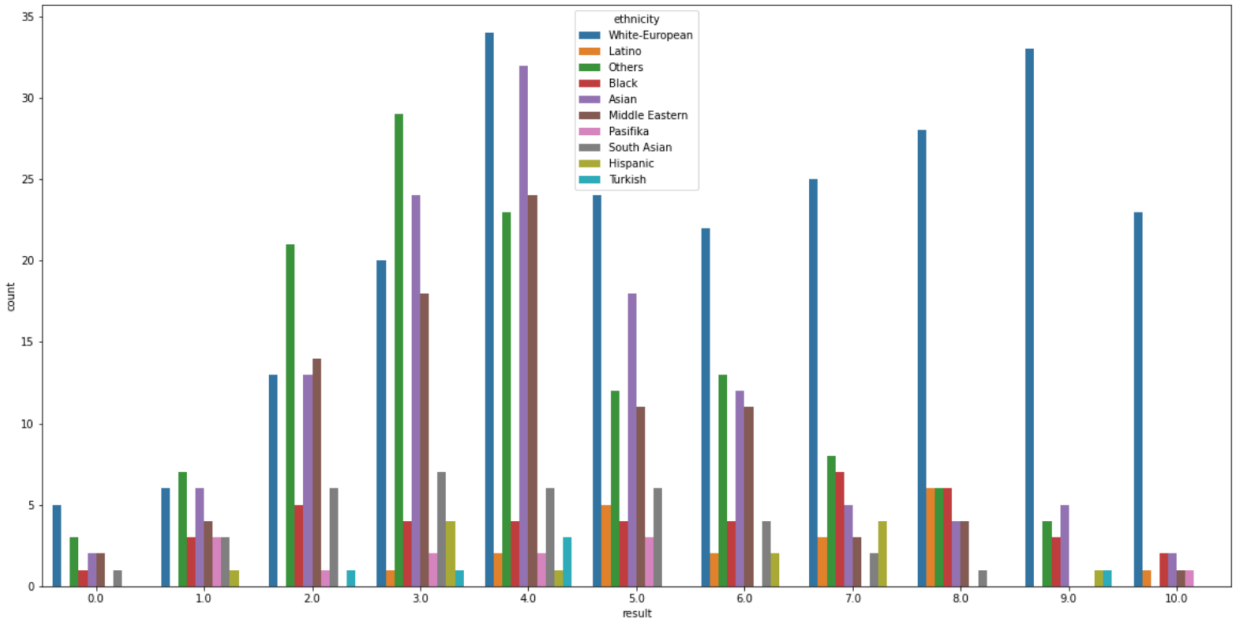


I replaced the values of ‘others’ and ‘?’ as ‘Others’, merging them into a single value.

```
# Replace '?' and 'others' as 'Others'
autism_df['ethnicity'] = autism_df['ethnicity'].replace('?', 'Others')
autism_df['ethnicity'] = autism_df['ethnicity'].replace('others', 'Others')
```

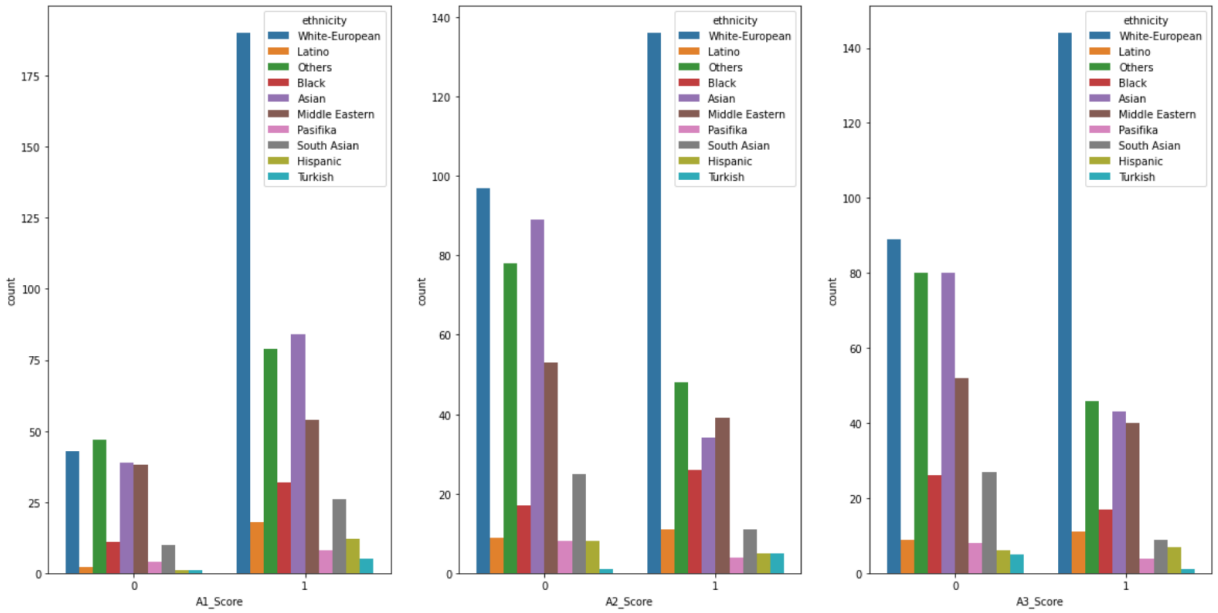
I used a count plot to check the typical overall score each ethnicity is getting. From looking at this plot it seems like there is a real possibility that the AQ-10 is biased towards White-European culture.

```
# Plot result by ethnicity
plt.figure(figsize = (20, 10))
sns.countplot(x = 'result', hue = 'ethnicity', data = autism_df)
plt.show()
# Black ethnicity and Latino ethnicity score more in the 8.0 result than Asian or Middle Eastern
```

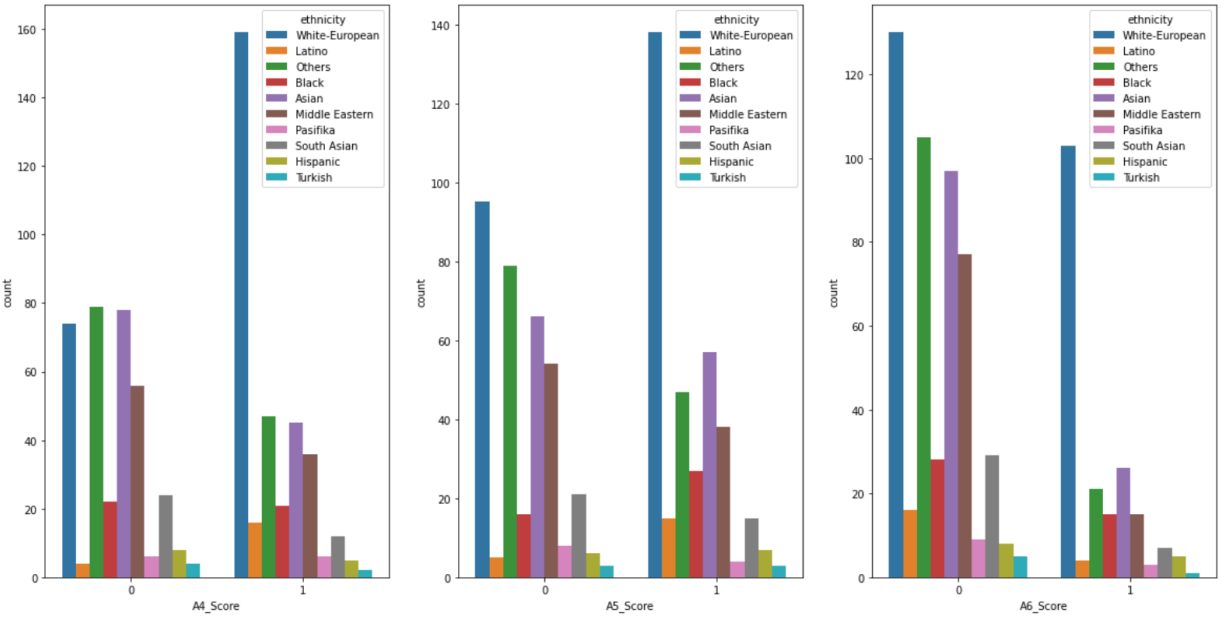


I then used a count plot to check which questions each ethnicity is typically responding to that contributed to their overall score. There are differences in the number of people getting scored for each question for each ethnicity, requiring further investigation in our analysis.

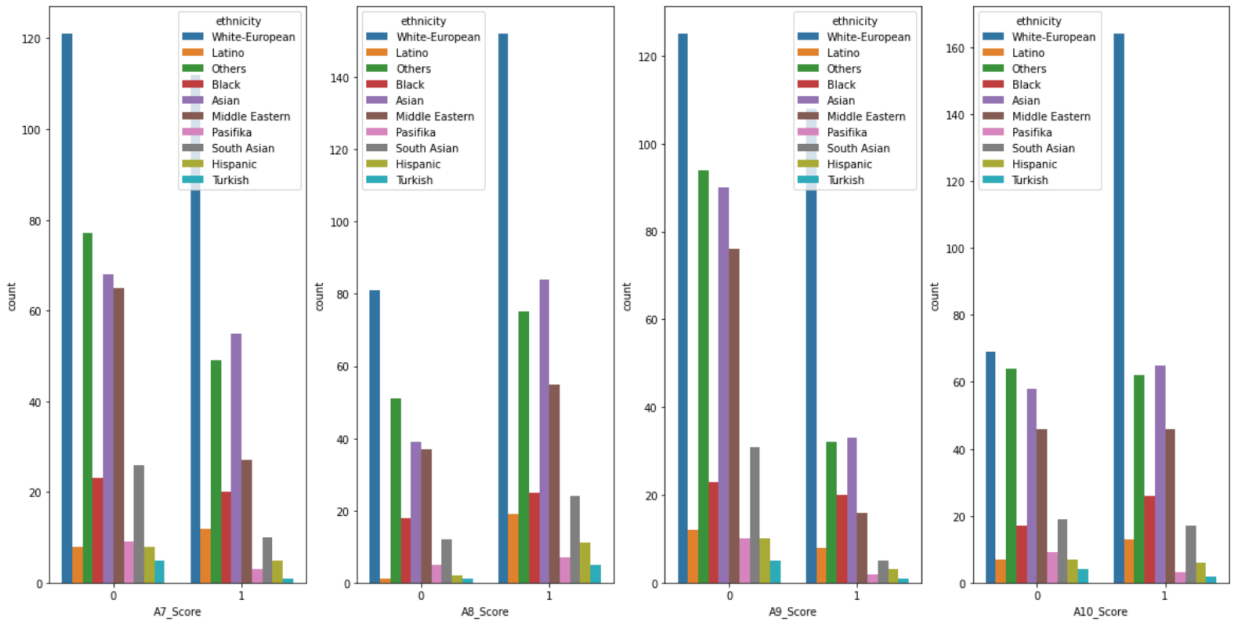
```
# Check for differences in question responses by ethnicity. White-European will have the most records for all so we will focus on other ethnicities.
fig, ax =plt.subplots(1,3, figsize=(20,10))
sns.countplot(x = 'A1_Score', hue = 'ethnicity', data = autism_df, ax=ax[0])
sns.countplot(x = 'A2_Score', hue = 'ethnicity', data = autism_df, ax=ax[1])
sns.countplot(x = 'A3_Score', hue = 'ethnicity', data = autism_df, ax=ax[2])
plt.show()
# Asian ethnicity scores most for A1, Middle Eastern ethnicity for A2.
```




```
fig, ax =plt.subplots(1,3, figsize=(20,10))
sns.countplot(x = 'A4_Score', hue = 'ethnicity', data = autism_df, ax=ax[0])
sns.countplot(x = 'A5_Score', hue = 'ethnicity', data = autism_df, ax=ax[1])
sns.countplot(x = 'A6_Score', hue = 'ethnicity', data = autism_df, ax=ax[2])
plt.show()
# Much lower scores for other ethnicities in A6
```

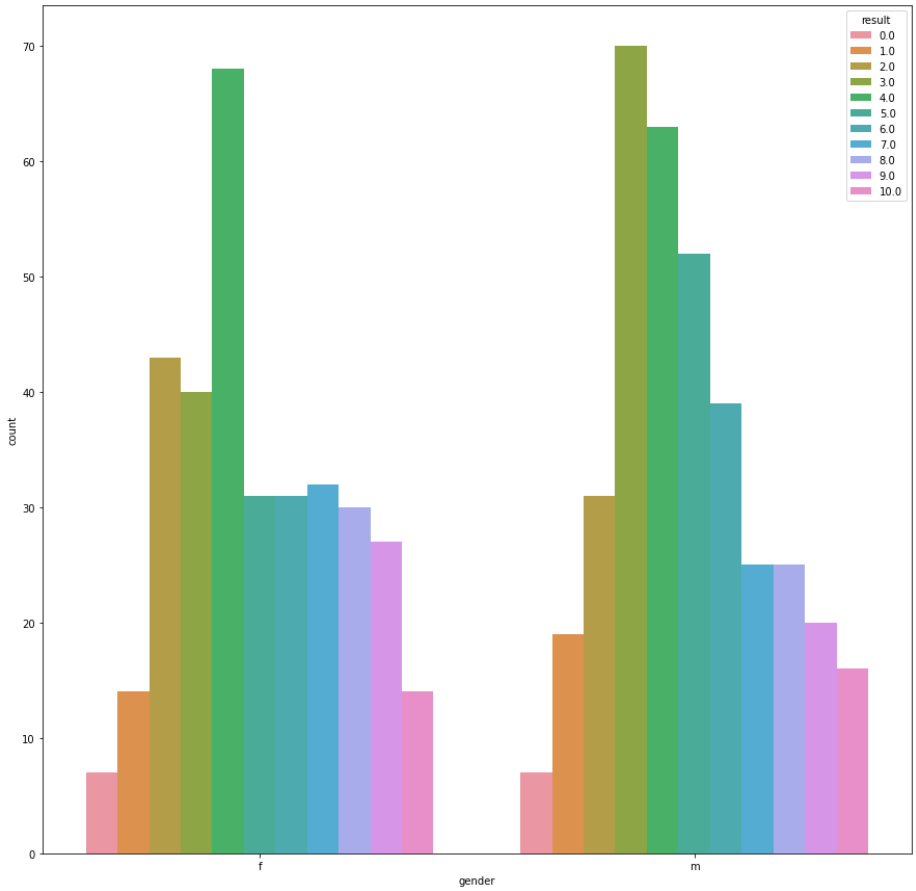


```
fig, ax =plt.subplots(1,4, figsize=(20,10))
sns.countplot(x = 'A7_Score', hue = 'ethnicity', data = autism_df, ax=ax[0])
sns.countplot(x = 'A8_Score', hue = 'ethnicity', data = autism_df, ax=ax[1])
sns.countplot(x = 'A9_Score', hue = 'ethnicity', data = autism_df, ax=ax[2])
sns.countplot(x = 'A10_Score', hue = 'ethnicity', data = autism_df, ax=ax[3])
plt.show()
# Higher scores for A8
# Likely possibility of ethnic bias
```



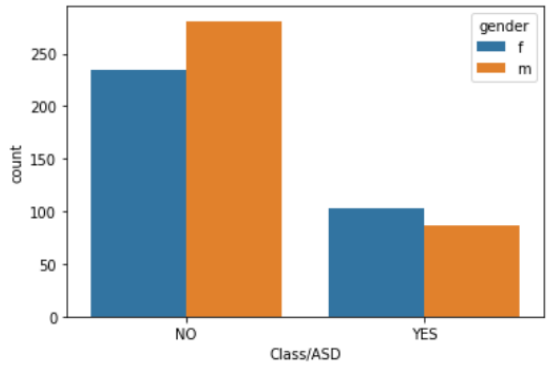
I used another count plot to check the typical overall score for each gender. There are more females scoring between 7-9 points than males, but more males get the full 10 points than females.

```
# Plot result by gender.
plt.figure(figsize = (15, 15))
sns.countplot(x = 'gender', hue = 'result', data = autism_df)
plt.show()
# Males are getting the highest score more than females, but in other scores above 0.7 females are more.
```



I used a count plot to check how many women and men are being diagnosed with ASD in this dataset. There are more females being classified as having ASD than males.

```
# Diagnosis by gender
sns.countplot(x = 'Class/ASD', hue = 'gender', data = autism_df)
plt.show()
# More females being diagnoses with ASD than males in this assessment
```



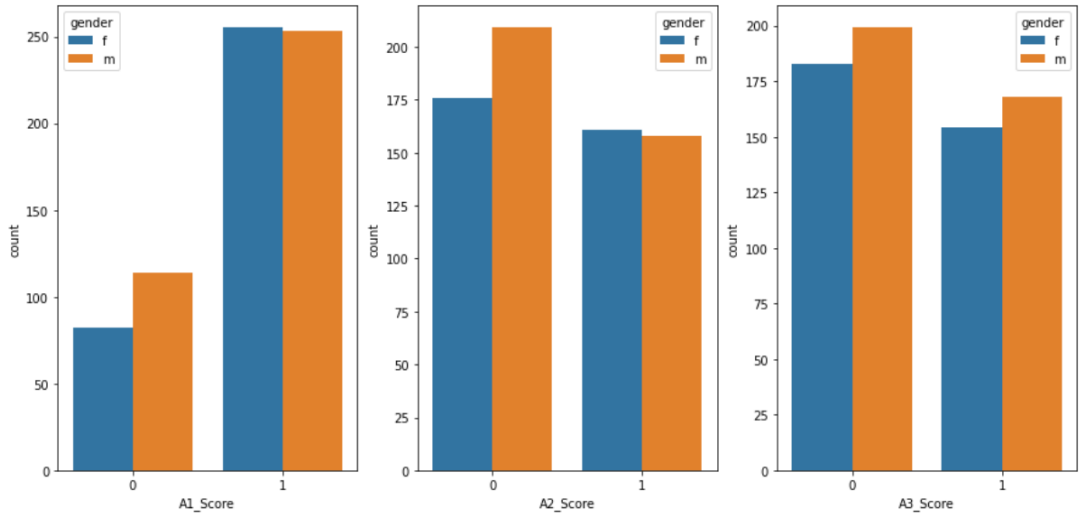
I checked the percentage for each classification for both males and females. 30.56% of females are classified with ASD and 23.43% of males are classified with ASD.

```
# Diagnosis by gender in percentages.
ASD_per_gender = autism_df.groupby('gender')['Class/ASD'].value_counts()
ASD_per_gender.groupby(level=[0]).apply(lambda g: 100* g / g.sum()).round(2)
```

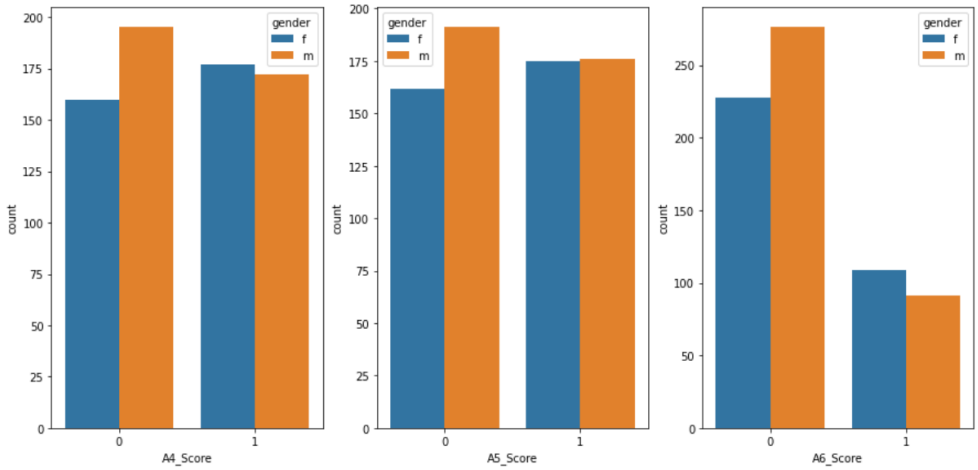
```
gender  Class/ASD
f       NO          69.44
        YES         30.56
m       NO          76.57
        YES         23.43
Name: Class/ASD, dtype: float64
```

Next, I looked at how males and females are responding to each question in the dataset. There are more males that scored in questions A3, A7, A8, & A9 than females.

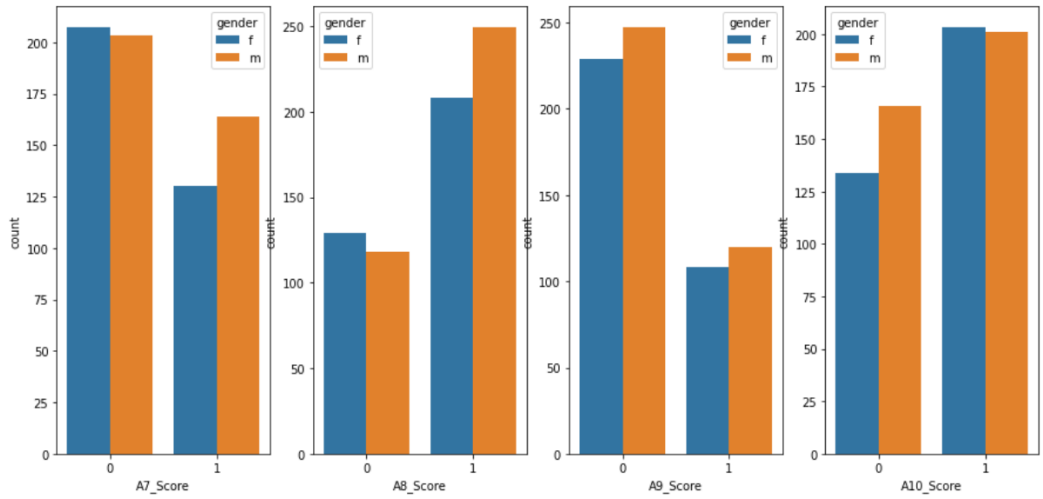
```
# Differences in question responses between genders
fig, ax =plt.subplots(1,3, figsize=(15,7))
sns.countplot(x = 'A1_Score', hue = 'gender', data = autism_df, ax=ax[0])
sns.countplot(x = 'A2_Score', hue = 'gender', data = autism_df, ax=ax[1])
sns.countplot(x = 'A3_Score', hue = 'gender', data = autism_df, ax=ax[2])
plt.show()
# Responses are mostly the same for A1 and A2, but differ slightly in A3
```



```
fig, ax =plt.subplots(1,3, figsize=(15,7))
sns.countplot(x = 'A4_Score', hue = 'gender', data = autism_df, ax=ax[0])
sns.countplot(x = 'A5_Score', hue = 'gender', data = autism_df, ax=ax[1])
sns.countplot(x = 'A6_Score', hue = 'gender', data = autism_df, ax=ax[2])
plt.show()
# Males giving more 'Yes' responses for A6 than females, but overall response is typically 'No'.
```



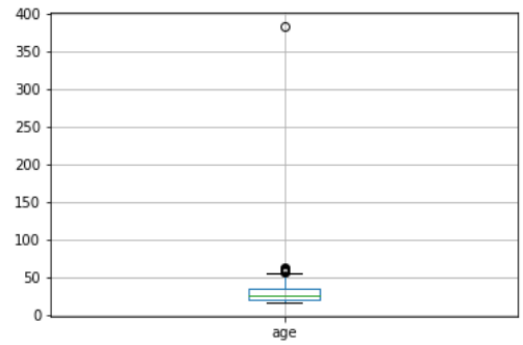

```
fig, ax =plt.subplots(1,4, figsize=(15,7))
sns.countplot(x = 'A7_Score', hue = 'gender', data = autism_df, ax=ax[0])
sns.countplot(x = 'A8_Score', hue = 'gender', data = autism_df, ax=ax[1])
sns.countplot(x = 'A9_Score', hue = 'gender', data = autism_df, ax=ax[2])
sns.countplot(x = 'A10_Score', hue = 'gender', data = autism_df, ax=ax[3])
plt.show()
# Slight differences in A8 and A9.
```



Using boxplots I looked at the distribution of age in the dataset. As indicated previously, we have an extreme outlier of 383 years of age.

```
# Participant age distribution. Extreme value to be removed in data cleaning.
autism_df.boxplot(column = 'age')
```

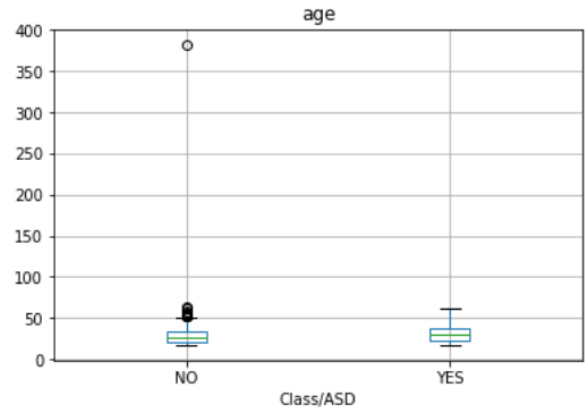
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47f79b2b50>
```



I checked the age of participants for each ASD classification. The ages were evenly distributed between the classifications.

```
# Age of respondants by diagnosis.
autism_df.boxplot(column = 'age', by = 'Class/ASD')
plt.suptitle('')
# Similar range for both.
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cboo
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) e
Text(0.5, 0.98, '')
```

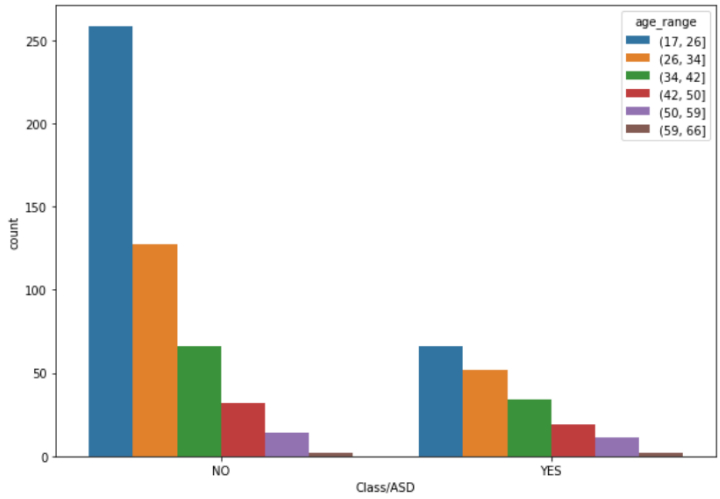


I sorted age ranges into six bins to simplify the analysis.

```
# Group ages into bins
autism_df['age_range'] = pd.cut(x=autism_df['age'], bins=[17, 26, 34, 42, 50, 59, 66])
```

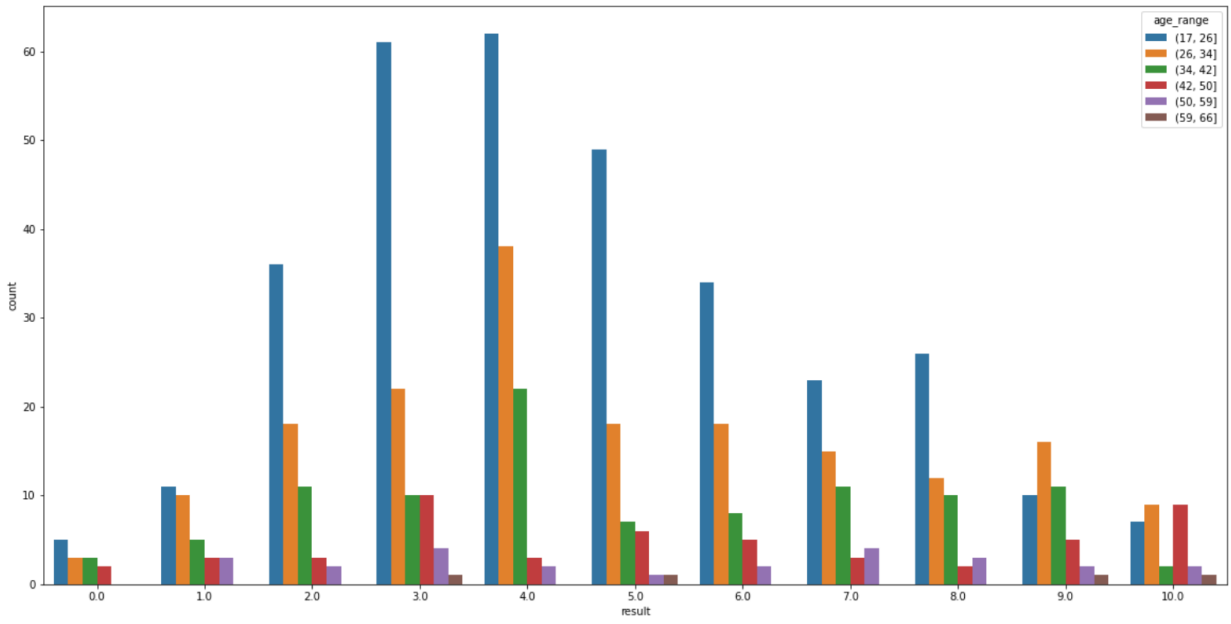
Looking at the classifications for each age group we can see that most of the records are between the ages of 17-26.

```
# Plot bins by diagnosis
plt.figure(figsize = (10, 7))
sns.countplot(x = 'Class/ASD', hue = 'age_range', data = autism_df)
plt.show()
# Distribution is similar for both outcomes.
```



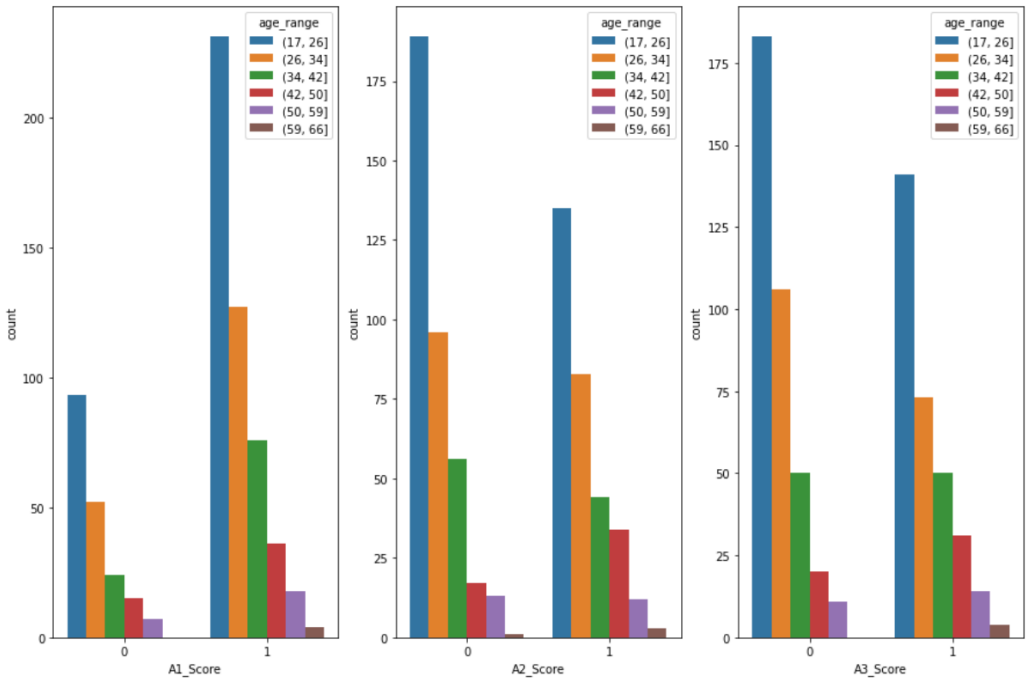
I looked at the score distribution for each age range.

```
# Plot bins by result
plt.figure(figsize = (20, 10))
sns.countplot(x = 'result', hue = 'age_range', data = autism_df)
plt.show()
# Interesting that more people with the highest scores are 26-34 and 42-50
```

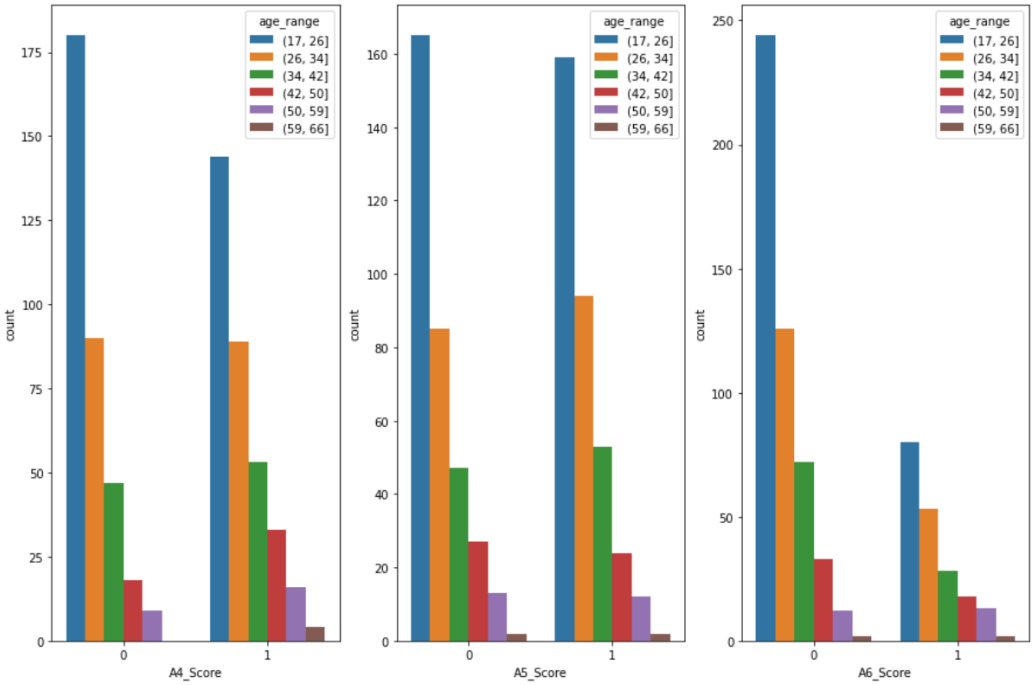


I then looked at what questions people from different age ranges were typically responding to.

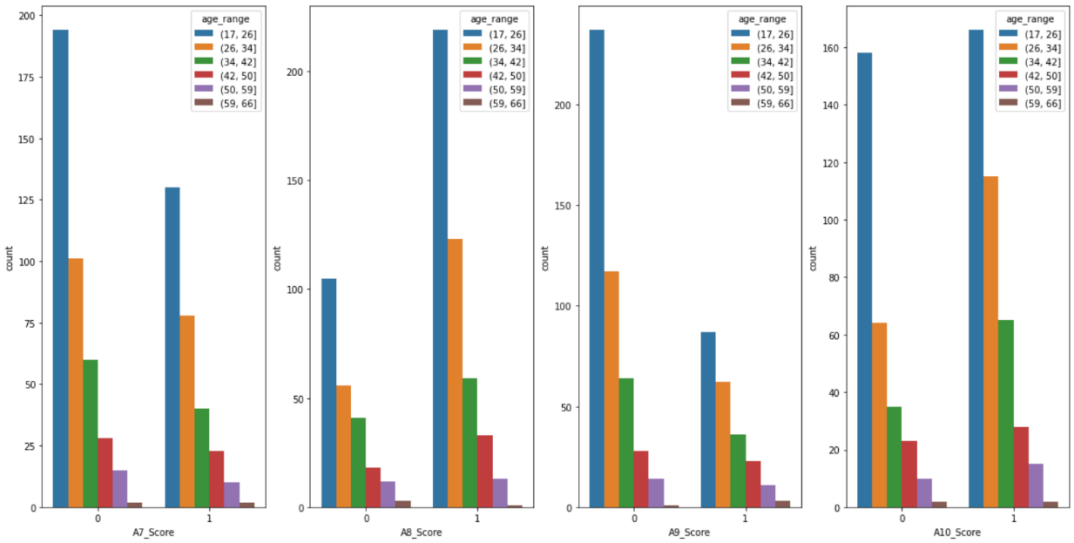
```
# Plot differences in question responses by age
fig, ax =plt.subplots(1,3, figsize=(15,10))
sns.countplot(x = 'A1_Score', hue = 'age_range', data = autism_df, ax=ax[0])
sns.countplot(x = 'A2_Score', hue = 'age_range', data = autism_df, ax=ax[1])
sns.countplot(x = 'A3_Score', hue = 'age_range', data = autism_df, ax=ax[2])
plt.show()
```




```
fig, ax =plt.subplots(1,3, figsize=(15,10))
sns.countplot(x = 'A4_Score', hue = 'age_range', data = autism_df, ax=ax[0])
sns.countplot(x = 'A5_Score', hue = 'age_range', data = autism_df, ax=ax[1])
sns.countplot(x = 'A6_Score', hue = 'age_range', data = autism_df, ax=ax[2])
plt.show()
# All participants ages 59+ respond 'Yes' to A4.
```

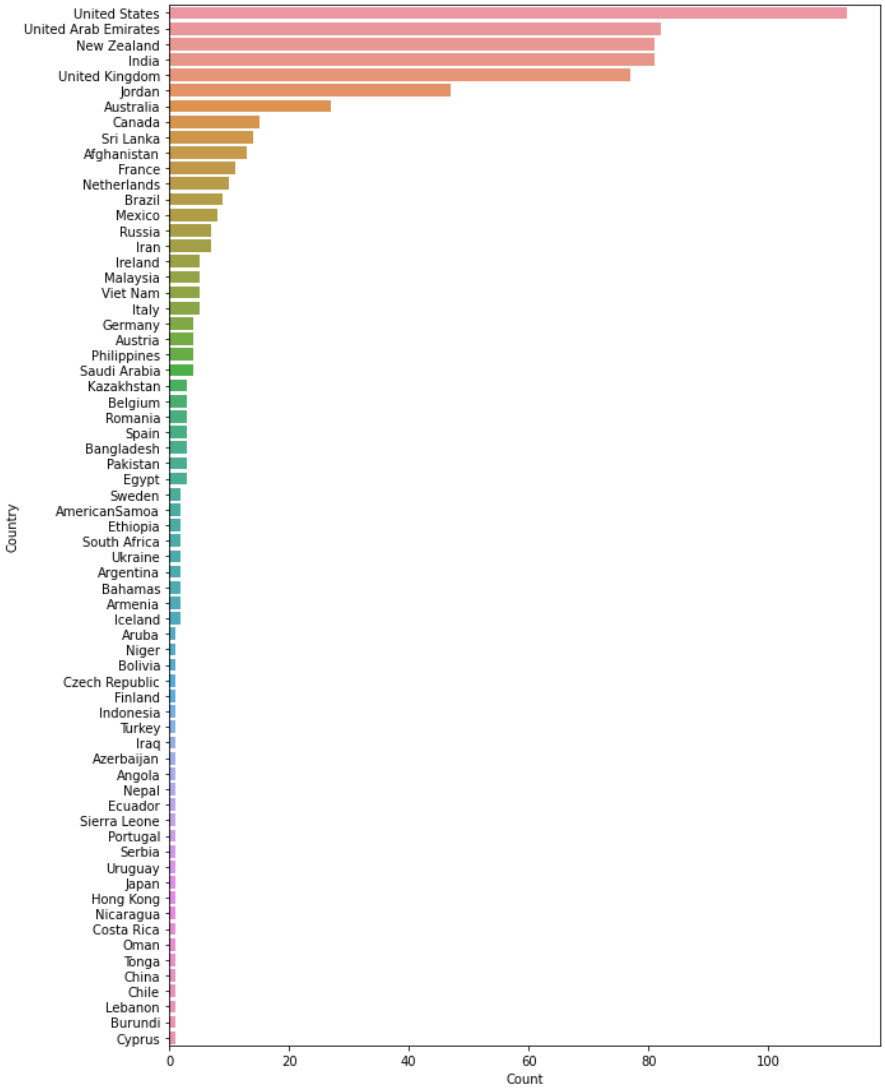


```
fig, ax =plt.subplots(1,4, figsize=(20,10))
sns.countplot(x = 'A7_Score', hue = 'age_range', data = autism_df, ax=ax[0])
sns.countplot(x = 'A8_Score', hue = 'age_range', data = autism_df, ax=ax[1])
sns.countplot(x = 'A9_Score', hue = 'age_range', data = autism_df, ax=ax[2])
sns.countplot(x = 'A10_Score', hue = 'age_range', data = autism_df, ax=ax[3])
plt.show()
# age range 17-26 has high scores in A8 and A10. 'Yes' responses for age range 42-50 are pretty consistant.
```



I looked at the number of records by country of residence. The majority of participants in the dataset are from the United States.

```
# Number of app users by country of residence
plt.figure(figsize = (10,15))
fig = sns.barplot(y=autism_df['contry_of_res'].value_counts().index, x=autism_df['contry_of_res'].value_counts().values, data=autism_df)
fig.set(xlabel='Count', ylabel='Country')
plt.show()
# Country with the most participants is the USA
```



I wanted to look at the number of participants by continent, so I imported a new dataset that contained records of each continent of residence.

```
# Import new dataset with continents for each country in autism_df dataset
cc_df = pd.read_csv('country_continent.csv')
cc_df.head()
```

	contry_of_res	continent_of_res
0	United States	North America
1	Brazil	South America
2	Spain	Europe
3	United States	North America
4	Egypt	Africa

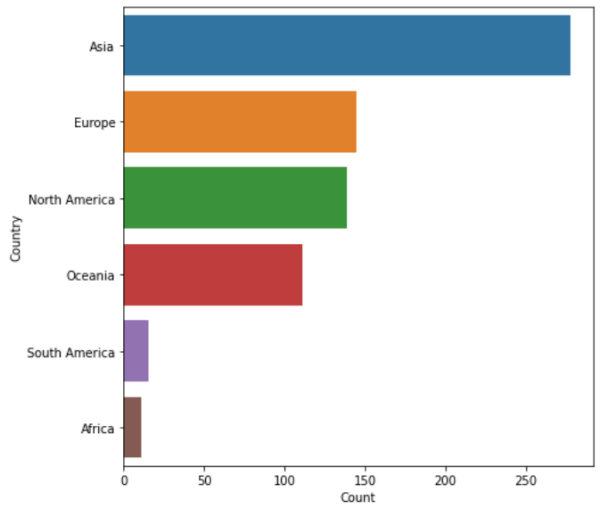
I merged the new dataset with my autism dataset using an inner join and merging on ‘contry_of_res’.

```
# Merge the datasets and remove duplicate rows
autism = pd.merge(autism_df, cc_df, how='inner', on=['contry_of_res']).drop_duplicates()
autism.shape
```

```
(699, 22)
```

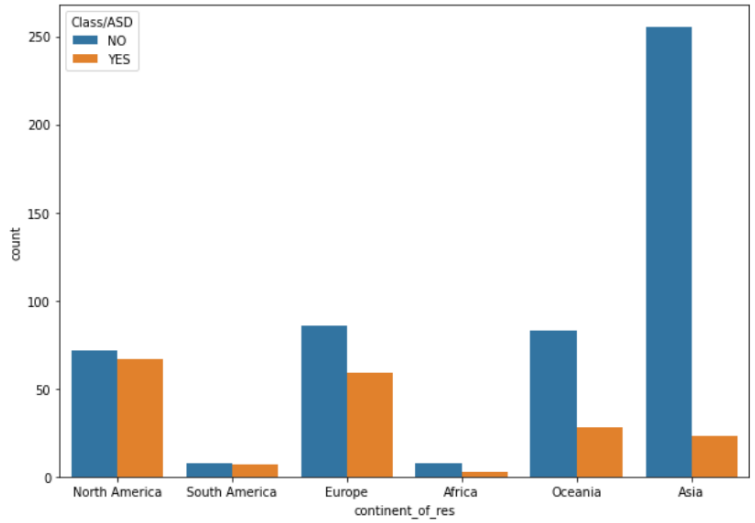
I created a bar plot to look at the number of participants by each continent. Most of the participants come from Asia, despite the most common country of residence being the United States.

```
# Plot number of participants by continent
plt.figure(figsize = (7,7))
fig = sns.barplot(y=autism['continent_of_res'].value_counts().index, x=autism['continent_of_res'].value_counts().values, data=autism)
fig.set(xlabel='Count', ylabel='Country')
plt.show()
# Most participants live in Asia
```



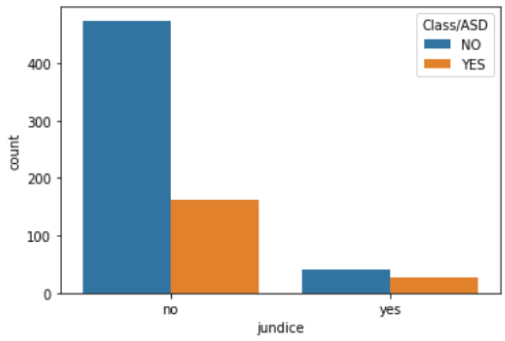
I used a count plot to look at ASD classification by continent of residency. Europeans and North Americans have more positive classifications that other continents.

```
# Plot diagnosis by continent of residency
plt.figure(figsize=(10,7))
sns.countplot(x = 'continent_of_res', hue = 'Class/ASD', data = autism)
plt.show()
# Europe and North America have the most positive number of diagnosis.
```



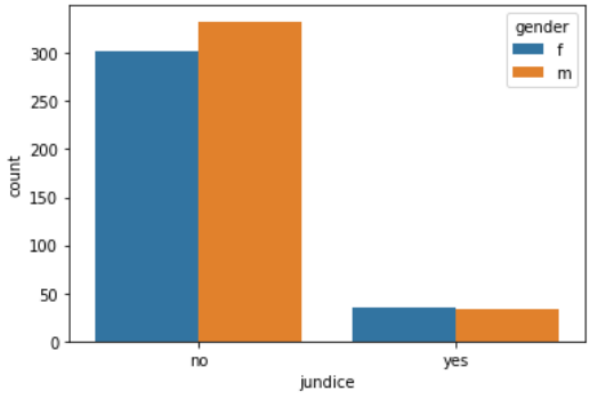
I looked at the impact of having had jaundice as a baby on an autistic diagnosis. In this dataset there does not seem to be a strong relationship between the two variables.

```
# Plot relation between previous diagnosis of jaundice and autism.
sns.countplot(x = 'jundice', hue = 'Class/ASD', data = autism_df)
plt.show()
# Having jaundice does not seem to increase likelihood of ASD
```



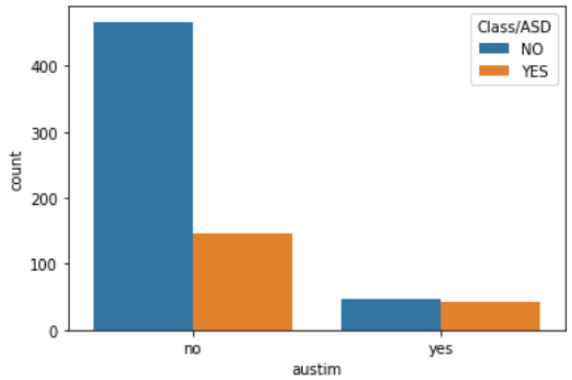
I looked to see if jaundice is more prominent in males than females.

```
# Is jaundice more typical in a particular gender?
sns.countplot(x = 'jundice', hue = 'gender', data = autism_df)
plt.show()
# Answer: NO
```



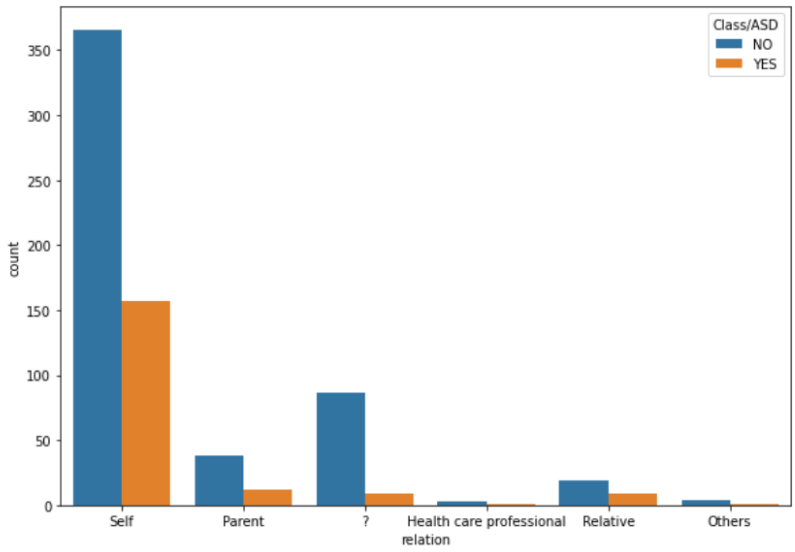
I looked at the genetic impact of autism in the dataset. A previous history of autism in the family does not seem to have a dramatic impact on diagnosis for this dataset.

```
sns.countplot(x = 'austim', hue = 'Class/ASD', data = autism_df)
plt.show()
# Genetic componant does not seem significant in this case
```



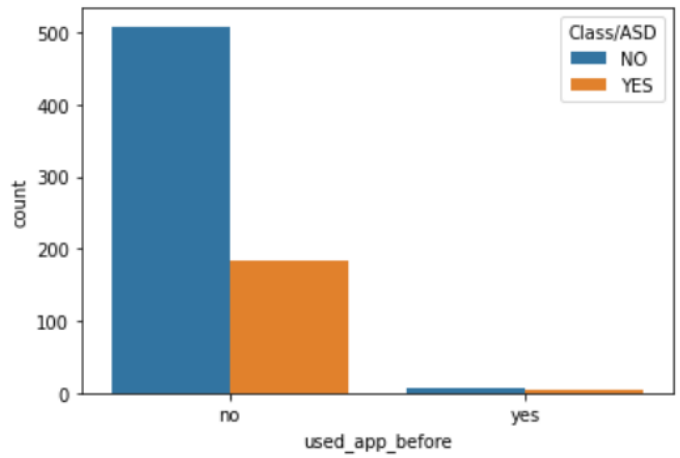
I also wanted to check if the person filling out the questionnaire had an impact on the target outcome. There is an unknown category where those who filled it out had significantly more ‘NO’ classifications than ‘YES’ classification. There are also some invalid variables that need to be replaced.

```
# Plot Class/ASD by who completed the app questionnaire
# 'relation' has an invalid value that will need to be changed in data cleaning.
plt.figure(figsize=(10,7))
sns.countplot(x = 'relation', hue = 'Class/ASD', data = autism_df)
plt.show()
# Majority of people completed the questionnaire themselves
```



Looking at the count plot we can see that the majority of the participants had not used the app before.

```
# Plot Class/ASD by whether participants have used the app before
sns.countplot(x = 'used_app_before', hue = 'Class/ASD', data = autism_df)
plt.show()
# Most participants have not used the app before and it does not seem to affect diagnosis.
```

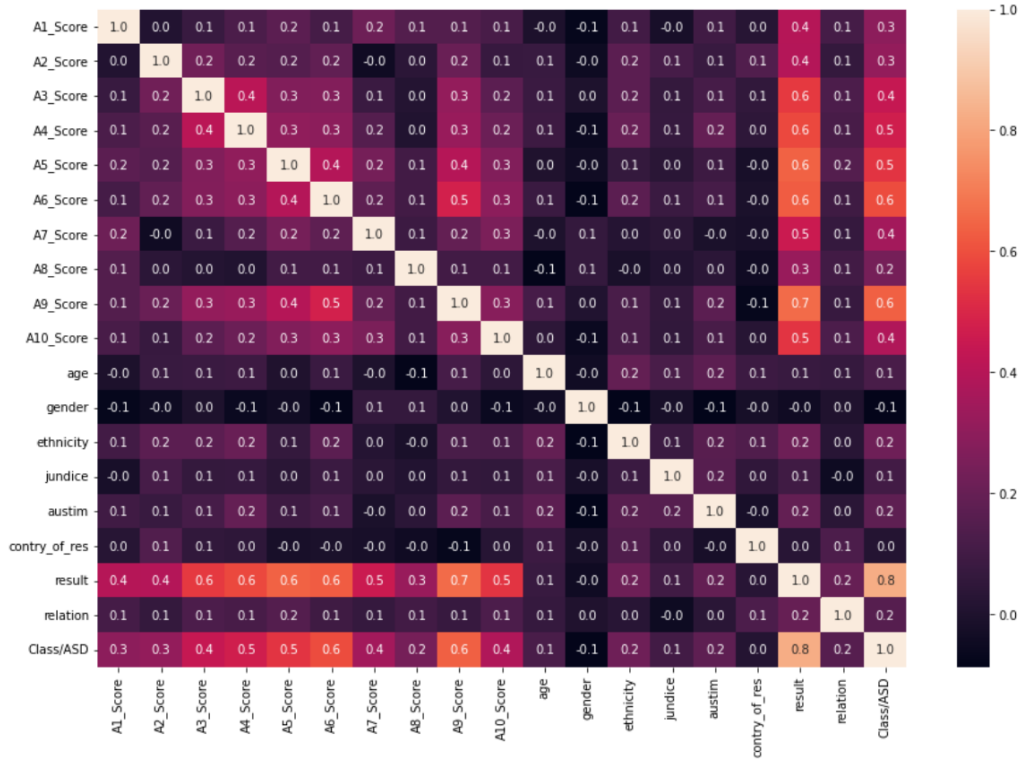


To transform the categorical variables into numeric variables I imported LabelEncoder from sklearn and fit_transformed the variable into numeric form.

```
# Transform categorical variables to numerical for full heatmap
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
autism_df['ethnicity'] = le.fit_transform(autism_df['ethnicity'])
autism_df['jundice'] = le.fit_transform(autism_df['jundice'])
autism_df['austim'] = le.fit_transform(autism_df['austim'])
autism_df['contry_of_res'] = le.fit_transform(autism_df['contry_of_res'])
autism_df['relation'] = le.fit_transform(autism_df['relation'])
autism_df['Class/ASD'] = le.fit_transform(autism_df['Class/ASD'])
autism_df['gender'] = le.fit_transform(autism_df['gender'])
autism_df['age'] = le.fit_transform(autism_df['age'])
```

I created a correlation heatmap for all the variables. Most of the variables have little to no correlation with the target except for the _Score variables and the result variable.

```
# Create heatmap
corr2 = autism_df.corr()
plt.figure(figsize = (15, 10))
sns.heatmap(corr2, annot=True, fmt='.1f')
plt.show()
# No strong correlations for categorigal variables.
```



3.1 EDA Conclusion:

From our Exploratory Data Analysis, it seems the majority of our participants are between the ages of 17-26 and primarily reside in the United States. Almost all of the participants are using the app

for the first tim in this analysis. The dataset has an equal distribution of gender with 367 males and 337 females. Females are shown to be classified as having ASD more than males and more frequently score between 7-9 points, while males score the full 10 points more than females do. For ethnicity, White-Europeans significantly outnumber the other ethnic groups. The demographic variables of 'relation', 'jaundice', and 'austim' do not seem to have a significant impact on whether or not a participant is classified with ASD.

4.0 Data Cleansing & Preparation

Data cleansing and preparation is the process of detecting and correcting corrupt or inaccurate variables and values from a dataset and transforming raw data into a form that can easily be analyzed. For this analysis, we will be using the methods of:

- Renaming
- Exclusion
- Deletion
- Imputation

Replace the columns names of incorrectly spelled variables.

```
#Replace incorrectly spelled column names
#'Austim' is a feature that describes a history of autism in the family. We will replace it as 'autism_in_family'.
autism_df.rename(columns={'jundice':'jaundice', 'austim':'autism_in_family', 'contry_of_res': 'country_of_res'}, inplace=True)
autism_df.columns
```

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

Check the index of our extreme variable.

```
# Maximum age value is an extreme outlier. Drop the value.
print(autism_df[(autism_df['age']==383.0)].index)
```

```
Int64Index([52], dtype='int64')
```

Delete the record.

```
# Drop record 52
autism_df.drop(index=52, inplace=True)

#Reset the index
autism_df.reset_index(inplace=True)
```

The skewness of this variable has been reduced to a normal range now that we have deleted the extreme value.

```
# Check skewness after removing extreme variable in age
skewed = autism_df.skew(axis = 0, skipna = True)
skewed
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher
  """Entry point for launching an IPython kernel.
A1_Score    -0.988682
A2_Score     0.186115
A3_Score     0.168807
A4_Score     0.014256
A5_Score     0.002851
A6_Score     0.957352
A7_Score     0.332346
A8_Score    -0.630645
A9_Score     0.752161
A10_Score   -0.302755
age          1.036616
result       0.323562
dtype: float64
```

The age column also has a decimal value that we want to change to match the other ages.

```
autism_df['age'].unique()
```

```
array([26.        , 24.        , 27.        , 35.        , 40.        ,
       36.        , 17.        , 64.        , 29.        , 33.        ,
       18.        , 31.        , 30.        , 34.        , 38.        ,
       42.        , 43.        , 48.        , 37.        , 55.        ,
       50.        , 53.        , 20.        , 28.        , 21.        ,
       47.        , 32.        , 44.        , 29.19400856, 19.        ,
       58.        , 45.        , 22.        , 39.        , 25.        ,
       23.        , 54.        , 60.        , 41.        , 46.        ,
       56.        , 61.        , 59.        , 52.        , 49.        ,
       51.        ])
```

Round the decimal value to 29.

```
# Replace weird value with 29
autism_df['age'] = autism_df['age'].replace(29.1940085592011, 29)
```

```
autism_df['age'].unique()
```

```
array([26., 24., 27., 35., 40., 36., 17., 64., 29., 33., 18., 31., 30.,
       34., 38., 42., 43., 48., 37., 55., 50., 53., 20., 28., 21., 47.,
       32., 44., nan, 19., 58., 45., 22., 39., 25., 23., 54., 60., 41.,
       46., 56., 61., 59., 52., 49., 51.])
```

‘Age’ has two missing values. We will impute these missing values with the mean of ‘Age’.

```
#Impute missing values in age with the mean of age
autism_df['age'] = autism_df['age'].fillna(autism_df['age'].mean())
```

```
# Check that missing values have been replaced
autism_df.isnull().sum()
```

```
index               0
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                 0
gender              0
ethnicity           0
jaundice            0
autism_in_family    0
country_of_res      0
used_app_before     0
result              0
relation            0
Class/ASD           0
dtype: int64
```

Replace the '?' value in relation with 'Others'.

```
# Replace '?' as 'Others'
autism_df['relation'] = autism_df['relation'].replace('?', 'Others')
autism_df['relation'].unique()
```

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

An index column was created when we uploaded the dataset. Delete this column.

```
#drop the index column
autism_df.drop(columns='index', inplace=True)
```

5.0 Modeling Exploration:

For this analysis we used a top-down approach, beginning with a broader analysis of the entire dataset. The purpose of this was to eliminate possible bias in the analysis and allow me to compare submodels with the overall dataset to see whether model accuracy improves when segmenting the data by gender. We explored the dataset using 4 different sets of predictors:

1. All of the dataset variables
2. Only the demographic variables
3. All of the dataset variables with ‘contry_of_res’ replaced by ‘continent_of_res’
4. Only using the _Score variables

The next step in the analysis was to split the full dataset into two smaller datasets by gender. After running a full analysis on these two datasets I further segmented the data by ethnicity for both genders and reran the analysis for ethnic subgroups. For both the male and female datasets the _Score variables were used exclusively as the predictors.

Models:

- **Decision Trees:**

  Decision trees are one of the best predictive modeling tools used. Input selection is conducted by a split search algorithm that rejects any variables with a log worth below 0.7. The complexity of decision trees is reduced by pruning so that the resulting tree only includes variables above the p-value threshold. The initial split is the Root Node, and the final splits are the Leaf Nodes. For this analysis, we will be using two types of decision trees: Maximal Trees and Optimized Trees with hyperparameter tuning.

- **Random Forests:**

  Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees and then combining their output to achieve a single result. This model uses a bootstrapping method to generate random samples of decision trees with low correlation to one another and singles out common features across all the trees.

- **Regressions:**

A regression model is a statistical model that measures the relationship between the independent variables and the dependent variables. For this analysis, I will primarily be using Logistic Regression models. Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative distribution function of the logistic distribution. I will also run a Linear Regression Model which is used to predict continuous variables.

- **Neural Networks:**

  A neural network is a set of connected input/output variables where each connection has a given weight that determines the outcome. Neural networks take non-linear functions of linear combinations of input variables. Neural networks are a black box method, meaning that you cannot see what is happening in the system that generates the given output.

- **Association Rules:**

  Association Rule Learning is a machine learning method used to discover relationships between variables in a dataset. It identifies frequent if-then associations where the 'if' is the antecedent and the 'then' is the consequent. It then uses the criteria of support and confidence to identify the most important relationships.

Full Dataset Analysis:

Import the necessary packages to run the models and the functions to assess model accuracy.

```
# Import libraries
%matplotlib inline

from pathlib import Path

import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LinearRegression, LogisticRegression, LogisticRegressionCV
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, roc_curve, auc, roc_auc_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
import matplotlib.pylab as plt
import seaborn as sns
import statsmodels.api as sm

!pip install mord
from mord import LogisticIT

!pip install scikit-plot
import scikitplot as skplt
!pip install dmba
from dmba import plotDecisionTree, classificationSummary, regressionSummary, gainsChart, adjusted_r2_score, AIC_score, BIC_score
from dmba.metric import AIC_score

import heapq
from collections import defaultdict
from mlxtend.frequent_patterns import apriori, association_rules
!pip install surprise
from surprise import Dataset, Reader, KNNBasic
```

Rename Class/ASD as Class_ASD.

```
# Rename Class/ASD as Class_ASD
autism_df.rename(columns={'Class/ASD': 'Class_ASD'}, inplace=True)
```

5.1 Full Dataset Analysis with All Variables as Predictors:

Dummies:

Many of the models we are running cannot handle categorical variables. Before partitioning our dataset, we must use the get_dummies method to convert the categorical variables into numeric form.

Data Partitioning:

The first step is to partition the dataset into training sets and validation sets. Splitting data, or data partitioning, is a standard procedure for honest model performance assessment when running predictive models. We will split our data into two parts: Training data (60%) which is used for fitting the data and Validation data (40%) which is used for monitoring and modifying

the data to create better generalizations. I set random_state to 1 to ensure that every time I run the model, I get the same results.

```
# Split the data into training and validation
x = autism_df.drop(columns=['ID', 'result', 'Class_ASD'])
y = autism_df['Class_ASD']
# Convert categorical variables to dummies
X = pd.get_dummies(x, drop_first=True)
Y = pd.get_dummies(y, drop_first=True)

train_X, valid_X, train_Y, valid_Y = train_test_split(X, Y, test_size=0.4, random_state=1)
```

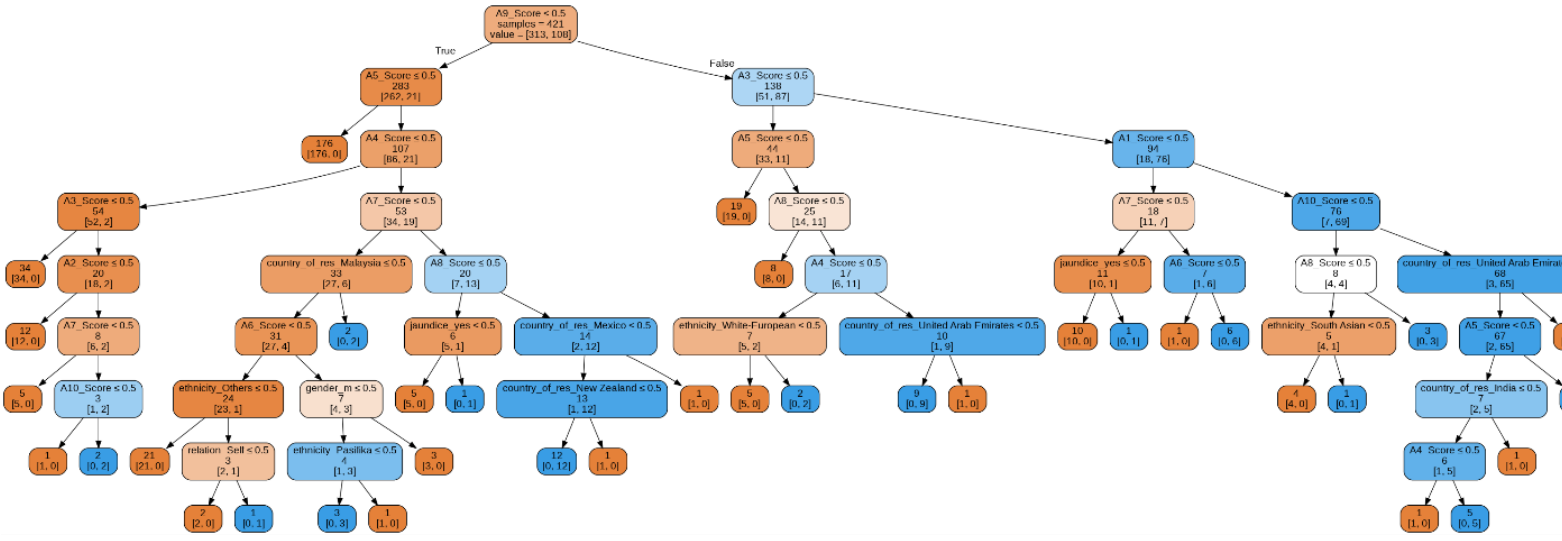
5.1.0 Decision Trees:

Create a maximal decision tree.

```
# Create a maximal tree
fullClassTree = DecisionTreeClassifier(random_state=1)
fullClassTree.fit(train_X, train_Y)
```

```
DecisionTreeClassifier(random_state=1)
```

```
# Plot the tree
plotDecisionTree(fullClassTree, feature_names=train_X.columns)
```



Check the model accuracy.

```
# Check accuracy
classificationSummary(valid_Y,fullClassTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9220)

       Prediction
Actual   0   1
     0 191  10
     1  12  69
```

Create parameters to optimize the decision tree.

```
# Set Decision Tree Paremeters
param_grid = {
    'max_depth':[2, 3, 5, 8],
    'min_samples_split': [0.07, 0.05, 0.01, 0.005],
    'min_impurity_decrease': [0.02, 0.01, 0.001, 0.005, 0.0001]
}
```

Run a gridsearch. This is a tuning technique that attempts to find the optimum parameters for a model by performing an exhaustive search through each of the parameter combinations. For our decision tree models we will use a cross-validation (cv) value of 5, which is the default.

```
# Perform a grid search for the best parameters
gridsearch = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid, cv=5, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

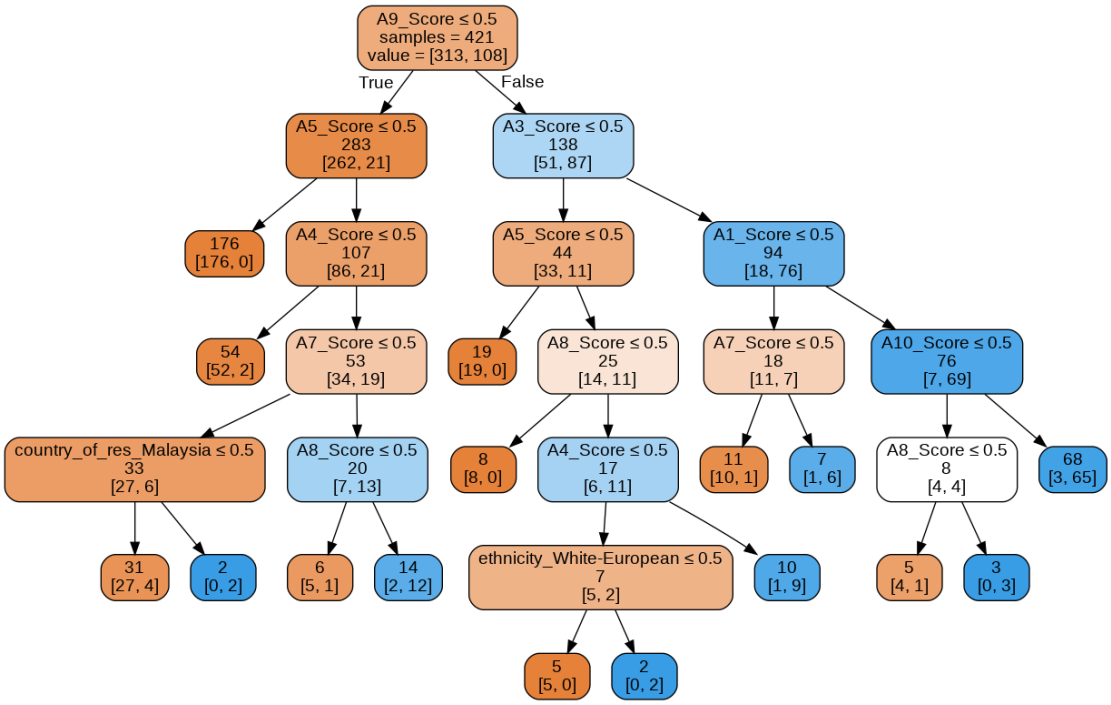
Check the best parameters selected by the gridsearch.

```
# Check for best parameters
gridsearch.best_params_
```

```
{'max_depth': 8, 'min_impurity_decrease': 0.001, 'min_samples_split': 0.005}
```

Optimize the decision tree and run the model. The first split is on the A9_Score, which splits into A5_Score and A3_Score.

```
# Create a new decision tree using optimal parameters
gridClassTree = DecisionTreeClassifier(max_depth=8,
                                       min_impurity_decrease=0.005,
                                       min_samples_split=0.01,
                                       random_state=1)
gridClassTree.fit(train_X, train_Y)
plotDecisionTree(gridClassTree, feature_names = train_X.columns)
```



Check the accuracy of the model. Model accuracy decreases slightly after optimization.

```
# Check Accuracy
classificationSummary(valid_Y, gridClassTree.predict(valid_X))
# Accuracy is lower than maximal tree but it is a better model because it is not overfitting
```

```
Confusion Matrix (Accuracy 0.9113)

       Prediction
Actual   0   1
     0 190  11
     1  14  67
```

5.1.1 Random Forest:

Create parameters to optimize the random forest model.

```
# Set random forest parameters
param_grid = {
    'max_depth':[5, 8, 10],
    'min_samples_split': [0.01, 0.001, 0.0001],
    'min_impurity_decrease': [0.01, 0.001, 0.005],
    'n_estimators':[300, 500, 700]
}
```

Run the parameters through the gridsearch.

```
# Perform a grid search for the best parameters
gridsearch = GridSearchCV(RandomForestClassifier(random_state=1), param_grid, cv=3, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

Check the best parameters.

```
# Check for best parameters
gridsearch.best_params_
```

```
{'max_depth': 10,
 'min_impurity_decrease': 0.001,
 'min_samples_split': 0.01,
 'n_estimators': 700}
```

Optimize the random forest and run the model.

```
# Create a random forest model using 500 samples
randomForest = RandomForestClassifier(random_state=1, n_estimators=700,
                                      min_impurity_decrease=0.001,
                                      min_samples_split=0.01,
                                      max_depth=10)
randomForest.fit(train_X, train_Y)
```

Create variables for feature importance and standard deviation.

```
# Calculate the importance of each variable and the standard deviation
importance = randomForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in randomForest.estimators_], axis=0)
```

Create a dataframe for the feature importance and standard deviation variables.

```
# Create dataframe of importance and standard deviation for each variable
randomForest_df = pd.DataFrame({'feature': train_X.columns,
                  'importance': importance,
                  'std': std})
print(randomForest_df.sort_values('importance', ascending=False))
```

```
                     feature  importance       std
8                   A9_Score    0.144486  0.173427
5                   A6_Score    0.143848  0.174352
4                   A5_Score    0.105866  0.121106
3                   A4_Score    0.091749  0.117589
2                   A3_Score    0.078869  0.101111
..                       ...         ...       ...
38      country_of_res_Chile    0.000000  0.000000
36    country_of_res_Burundi    0.000000  0.000000
33    country_of_res_Belgium    0.000000  0.000000
27      country_of_res_Aruba    0.000000  0.000000
59  country_of_res_Kazakhstan    0.000000  0.000000

[94 rows x 3 columns]
```

The accuracy for the random forest is greater than the decision trees.

```
# Check for accuracy
classificationSummary(valid_Y, randomForest.predict(valid_X))
# Accuracy is superior to decision trees
```

```
Confusion Matrix (Accuracy 0.9291)

       Prediction
Actual   0   1
     0 201   0
     1  20  61
```

Create a new variable for the features with importance greater than 5%.

```
# Create dataframe of variables with importance greater than 0
important = randomForest_df[(randomForest_df['importance'] >= 0.05)]
important.value_counts()
```

```
feature    importance  std
A10_Score  0.059460    0.083715    1
A3_Score   0.078869    0.101111    1
A4_Score   0.091749    0.117589    1
A5_Score   0.105866    0.121106    1
A6_Score   0.143848    0.174352    1
A7_Score   0.052393    0.067893    1
A9_Score   0.144486    0.173427    1
dtype: int64
```

5.1.2 Regressions:

Linear Regression:

Create a linear regression model and fit it to the training data.

```
# Run a linear regression to predict Class_ASD
lin_reg = LinearRegression()
lin_reg.fit(train_X, train_Y)
```

```
LinearRegression()
```

Check the coefficients and odds ratio for each variable. This dataframe has some unexpected values with high impact on the target according to the model.

```
# Check the intercept, coefficients and calculate odds ratio
print('intercept', lin_reg.intercept_)
coef = pd.DataFrame({'coef': lin_reg.coef_[0], 'odds': np.e**lin_reg.coef_[0]}, index=X.columns)
print(coef.sort_values('odds', ascending = False))
```

```
intercept [-0.73002671]
                                coef      odds
country_of_res_Sierra Leone    0.569060  1.766605
country_of_res_Nepal           0.527094  1.694002
country_of_res_Sweden          0.441237  1.554629
country_of_res_Viet Nam        0.417596  1.518307
country_of_res_Malaysia        0.413807  1.512566
...                                 ...       ...
country_of_res_Bolivia        -0.321020  0.725409
country_of_res_Czech Republic -0.453387  0.635472
country_of_res_Ecuador        -0.502330  0.605119
country_of_res_Hong Kong      -0.530698  0.588195
country_of_res_Bahamas        -0.569363  0.565886

[94 rows x 2 columns]
```

Check the accuracy of the model.

```
# Check model accuracy
regressionSummary(valid_Y, lin_reg.predict(valid_X))
```

```
Regression statistics

              Mean Error (ME) : 0.0108
Root Mean Squared Error (RMSE) : 0.2742
     Mean Absolute Error (MAE) : 0.2145
```

Import shap to visualize the importance of individual features.

```
# Install shap for visualization
!pip install shap
import shap
```

The shap values place the _Score variables as having the most impact on the target output, which does not matching our coefficient dataframe.

```
# Use shap to vizualize feature value and impact on the target
explainer = shap.Explainer(lin_reg, masker=shap.maskers.Impute(data= train_X),
                           feature_names= train_X.columns, algorithm="linear")
shap_values = explainer.shap_values(X)
shap.summary_plot(shap_values, X)
```

Reset the valid_Y dataframe index.

```
# Reset the index of valid_Y
validY_reset = valid_Y.reset_index()
# Remove the index column
validY_reset.drop(columns='index', inplace=True)
validY_reset.head()
```

	YES
0	0
1	0
2	1
3	1
4	0

Create a variable for the model predictions.

```
# Create a variable for model prediction
pred = lin_reg.predict(valid_X)
```

Create a dataframe from the pred variable.

```
# Create a dataframe for model predicitons
result = pd.DataFrame(pred, columns=['predicted'])
```

Concatenate the new dataframe with the valid_Y dataframe.

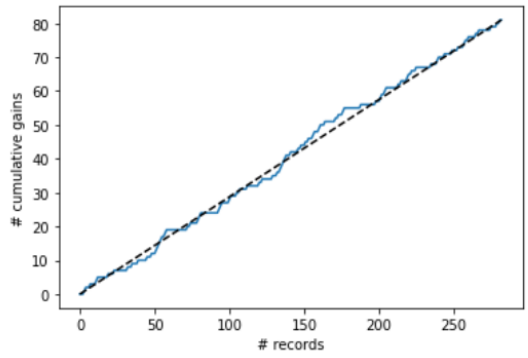
```
# Concatenate the prediction and valid_Y dataframes
df1 = pd.concat([result, validY_reset], axis=1)
df1.head()
```

	predicted	YES
0	0.081759	0
1	-0.068162	0
2	0.684777	1
3	0.881592	1
4	0.027152	0

Create a gains chart to see how well the model is predicting the target. The larger the curve of the line representing model predictions the better the model. We can see from this gains chart that the line has no curve at all. The linear regressions are performing so poorly that I will not run them again in the analysis.

```
# Plot a gains chart
plt.figure(figsize=(7,5))
gainsChart(df1.YES)
plt.show()

# Linear regression model is terrible
```

```
<Figure size 504x360 with 0 Axes>
```



Logistic Regression:

```
# Run a logistic regression model
log_reg = LogisticRegression(solver='liblinear', C=1e42, random_state=1)
log_reg.fit(train_X, train_Y)
```

Create a dataframe to check the coefficients and odds ratio. In this odds ratio the variables are extremely and unusually high. This may be due to the many variables predicting the target.

```
# Check the coefficients, intercept, and calcualte odds ratio
log_result = pd.DataFrame({'coef': log_reg.coef_[0], 'odds': np.e**log_reg.coef_[0]}, index=X.columns)
print('intercept', lin_reg.intercept_)
print(log_result.sort_values('odds', ascending=False))
```

```
intercept [-0.73002671]
                              coef           odds
A4_Score                 12.913161  405615.323854
A1_Score                 12.685488  323026.124497
A5_Score                 11.372498   86898.651784
A7_Score                 11.319376   82402.899214
A8_Score                 10.065906   23527.059044
...                            ...            ...
country_of_res_Pakistan  -3.909906       0.020042
relation_Parent          -4.095743       0.016643
relation_Self            -4.261023       0.014108
country_of_res_Hong Kong -5.124486       0.005949
country_of_res_Jordan    -6.194366       0.002041

[94 rows x 2 columns]
```

Check the accuracy of the model. The accuracy is extremely high and based on our coefficients it is likely that the model is overfitting to the data.

```
# Check model accuracy
classificationSummary(valid_Y, log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9894)

       Prediction
Actual   0   1
     0 201   0
     1   3  78
```

5.1.3 Neural Network:

Run a neural network with 2 hidden layers.

```
# Run a Neural Network model with 2 hidden layers. Fit the model to training data
clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=2, random_state=1)
clf.fit(train_X, train_Y)
```

Model accuracy is higher than the accuracy of the other models.

```
# Check for accuracy
classificationSummary(valid_Y, clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9574)

       Prediction
Actual   0   1
     0 195   6
     1   6  75
```

```
# Look at model probability prediction
pd.DataFrame(clf.predict_proba(valid_X))
```

	0	1
0	1.000000	3.772214e-10
1	1.000000	3.772214e-10
2	1.000000	3.930357e-10
3	0.017072	9.829276e-01
4	1.000000	3.772214e-10
...	...	...
277	1.000000	4.201857e-10
278	0.017072	9.829276e-01
279	1.000000	3.772214e-10
280	0.017073	9.829272e-01
281	1.000000	3.772214e-10

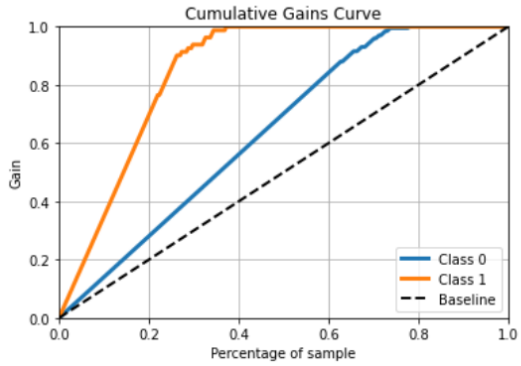
282 rows × 2 columns

Create a cumulative gains chart. Here we can see that the model predictions of ‘YES’ ASD classification optimize at 40% while predictions for ‘NO’ ASD classifications optimize at 70%.

```
# Create a variable for model prediction probabilities
pred = clf.predict_proba(valid_X)
```

```
# Run a cumulative gains chart
import scikitplot as skplt
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, pred)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



5.2 Analysis with only demographic variables as predictors:

Set only the demographic variables as the predictors. Use the get_dummies method to convert all the variables into numeric form then split the dataset.

```
# Split the dataset with Class/ASD as the target and only demographic variables as predictors
predictors = ['age', 'gender', 'ethnicity', 'jaundice', 'autism_in_family', 'country_of_res', 'used_app_before', 'relation']
x = autism_df[predictors]
y = autism_df['Class_ASD']
# Convert categorical variables to dummies
X = pd.get_dummies(x, drop_first=True)
Y = pd.get_dummies(y, drop_first=True)

train_X, valid_X, train_Y, valid_Y = train_test_split(X, Y, test_size=0.4, random_state=1)
```

5.2.0 Decision Trees:

Create a maximal decision tree.

```
# Run a maximal tree
fullClassTree = DecisionTreeClassifier(random_state=1)
fullClassTree.fit(train_X, train_Y)
```

```
DecisionTreeClassifier(random_state=1)
```

Check the accuracy of the model. The accuracy is significantly lower than when the numeric variables are included in the analysis. This is typical for diagnosis datasets.

```
# Check accuracy
classificationSummary(valid_Y, fullClassTree.predict(valid_X))
# Accuracy drops without the _Score variables
```

```
Confusion Matrix (Accuracy 0.7340)

       Prediction
Actual   0   1
     0 173  28
     1  47  34
```

Create parameters to optimize the decision tree and run a gridsearch.

```
# Create parameters to optimize the tree
param_grid = {
    'max_depth':[5, 7, 8, 10, 11],
    'min_samples_split': [0.05, 0.01, 0.005, 0.001],
    'min_impurity_decrease': [0.02, 0.01, 0.001, 0.005, 0.0001]
}
```

```
# Run a grid search to find the best parameters
gridsearch = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid, cv=5, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

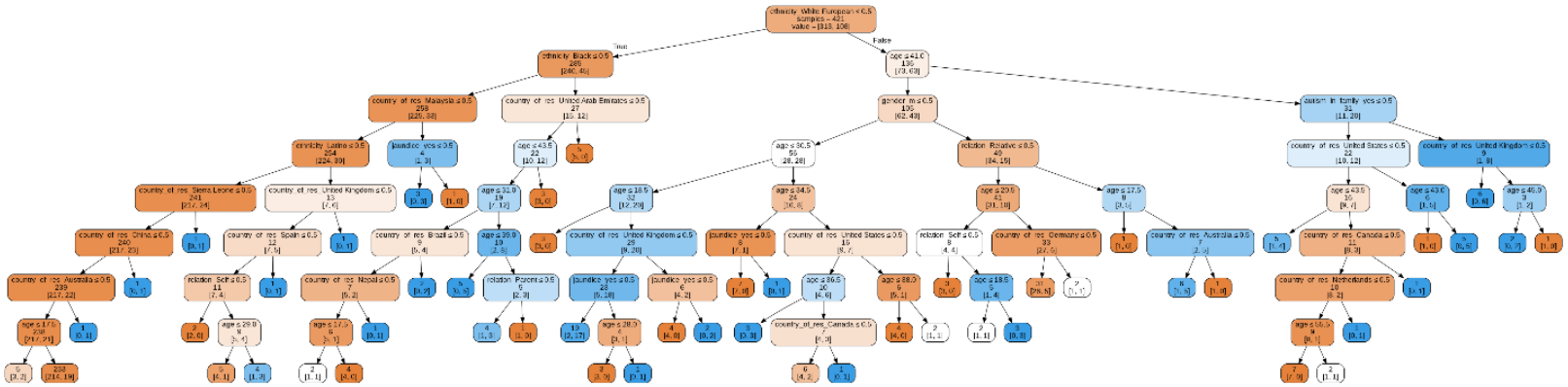
Check for the best parameters.

```
# Check for best parameters
gridsearch.best_params_
```

```
{'max_depth': 8, 'min_impurity_decrease': 0.001, 'min_samples_split': 0.005}
```

Optimize the decision tree and run the model.

```
# Optimize the decision tree and plot
gridClassTree = DecisionTreeClassifier(max_depth=8,
                                       min_impurity_decrease=0.001,
                                       min_samples_split=0.005,
                                       random_state=1)
gridClassTree.fit(train_X, train_Y)
plotDecisionTree(gridClassTree, feature_names = train_X.columns)
```



Checking the accuracy of the model we can see that it has not improved with optimization.

```
# Check model accuracy
classificationSummary(valid_Y, gridClassTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.7340)

       Prediction
Actual   0   1
     0 181  20
     1  55  26
```

5.3 Using continent_of_res instead of country_of_res:

Upload the dataset containing the continent of residency for each record.

```
# Upload dataset containing continent of residence
continent = pd.read_csv('country_continent.csv')
continent.head()
```

	contry_of_res	continent_of_res
0	United States	North America
1	Brazil	South America
2	Spain	Europe
3	United States	North America
4	Egypt	Africa

Merge the two datasets.

```
# Merge the autism dataset with the continent dataset using an inner join. Drop any duplicates
autism_df = pd.merge(autism_df, continent, how='inner', left_on=['country_of_res'], right_on=['contry_of_res']).drop_duplicates()
autism_df.shape
```

```
(703, 23)
```

Delete the ‘contry_of_res’ column from the continent dataset.

```
# Drop duplicate column
autism_df.drop(columns=['contry_of_res'], inplace=True)
autism_df.shape
```

```
(703, 22)
```

Split the dataset and replace country_of_res with continent_of_res in the predictors. Convert categorical variables to numeric form using get_dummies and split the dataset into training and validations sets.

```
# Split the dataset and use continent_or_res instead of country_of_res as a predictor
x = autism_df.drop(columns=['ID', 'country_of_res', 'result', 'Class_ASD'])
y = autism_df['Class_ASD']
# Convert categorical variables to dummies
X = pd.get_dummies(x, drop_first=True)
Y = pd.get_dummies(y, drop_first=True)

train_X, valid_X, train_Y, valid_Y = train_test_split(X, Y, test_size=0.4, random_state=1)
```

5.3.0 Decision Trees:

Create a maximal tree.

```
# Run a maximal tree
fullContinentTree = DecisionTreeClassifier(random_state=1)
fullContinentTree.fit(train_X, train_Y)
```

```
DecisionTreeClassifier(random_state=1)
```

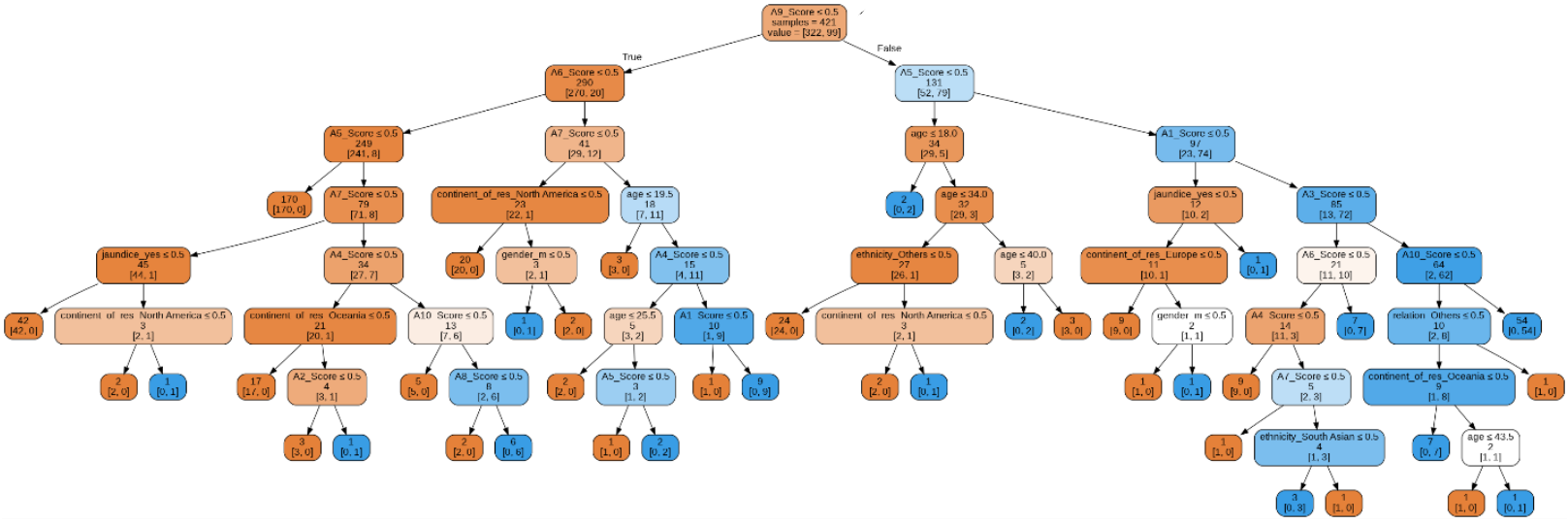
Model accuracy is much higher when eliminating country_of_res from the predictors.

```
# Check model accuracy
classificationSummary(valid_Y, fullContinentTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8794)

       Prediction
Actual   0   1
     0 179  13
     1  21  69
```

Plot the decision tree. The initial split is on the A9_Score, which splits into A5_Score and A6_Score.

```
# Plot the tree
plotDecisionTree(fullContinentTree, feature_names=train_X.columns)
```



Create parameters to optimize the decision tree and run a gridsearch.

```
# Set parameters to optimize the tree
param_grid = {
    'max_depth':[3, 5, 7, 8, 9],
    'min_samples_split': [0.05, 0.01, 0.005, 0.001],
    'min_impurity_decrease': [0.02, 0.01, 0.001, 0.005, 0.0001]
}
```

```
# Run a grid search to find the best optimizers
gridsearch = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid, cv=5, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

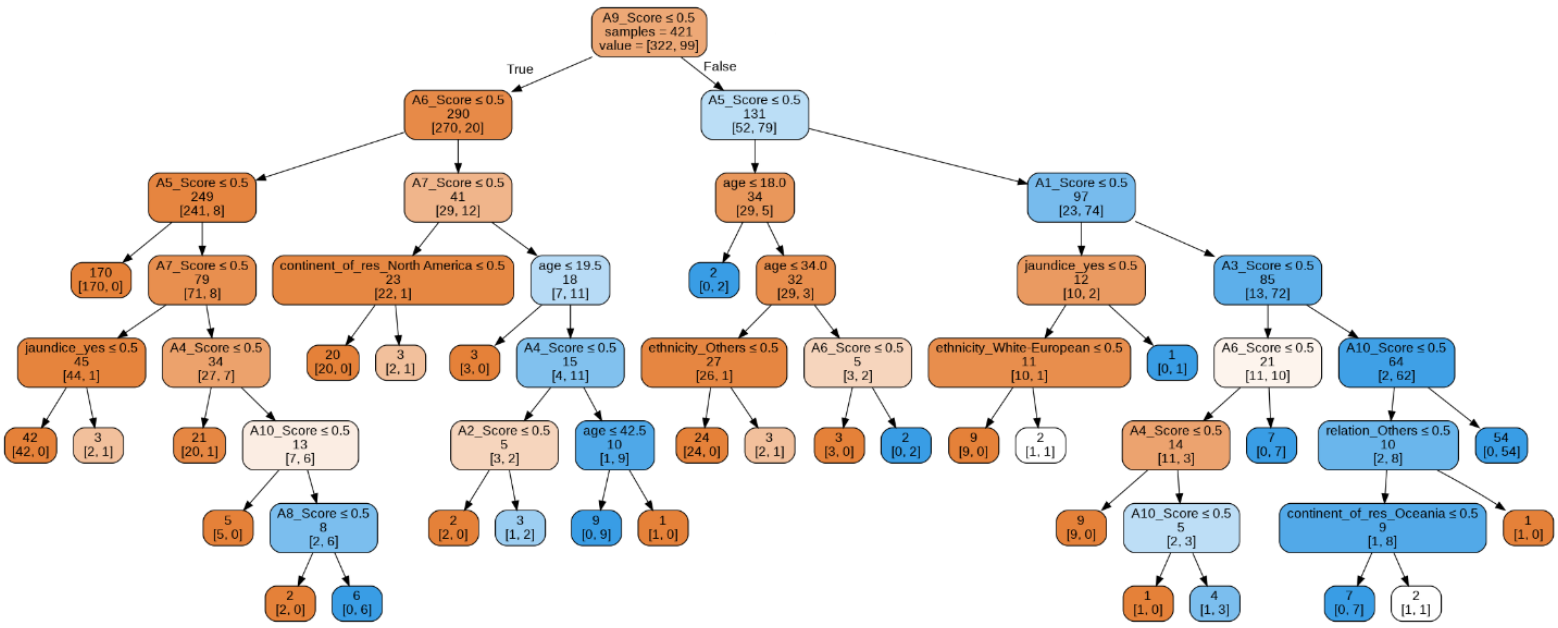
Check for the best parameters.

```
# Check for best parameters
gridsearch.best_params_
```

```
{'max_depth': 9, 'min_impurity_decrease': 0.001, 'min_samples_split': 0.01}
```

Optimize the decision tree and run the model.

```
# Optimize the tree and plot the model
gridContinentTree = DecisionTreeClassifier(max_depth=9,
                                           min_impurity_decrease=0.001,
                                           min_samples_split=0.01,
                                           random_state=1)
gridContinentTree.fit(train_X, train_Y)
plotDecisionTree(gridContinentTree, feature_names = train_X.columns)
```



Check the accuracy of the model. Model accuracy has improved with optimization.

```
# Check model accuracy
classificationSummary(valid_Y, gridContinentTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9113)

       Prediction
Actual   0   1
     0 183   9
     1  16  74
```

5.4 Analysis on _Score variables only:

This is the analysis I will be using to compare the male and female subgroups. Set the predictors as the _Score variables. Convert the target variable to numeric form and split the dataset.

```
# Split the dataset with only _Score variables as predictors
predictors = ['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score']
x = autism_df[predictors]
y = autism_df['Class_ASD']
# Convert categorical variables to dummies
X = x
Y = pd.get_dummies(y, drop_first=True)

train_X, valid_X, train_Y, valid_Y = train_test_split(X, Y, test_size=0.4, random_state=1)
```

5.4.0 Decision Trees:

Run a maximal tree.

```
# Run a maximal tree
fullScoreTree = DecisionTreeClassifier(random_state=1)
fullScoreTree.fit(train_X, train_Y)
```

```
DecisionTreeClassifier(random_state=1)
```

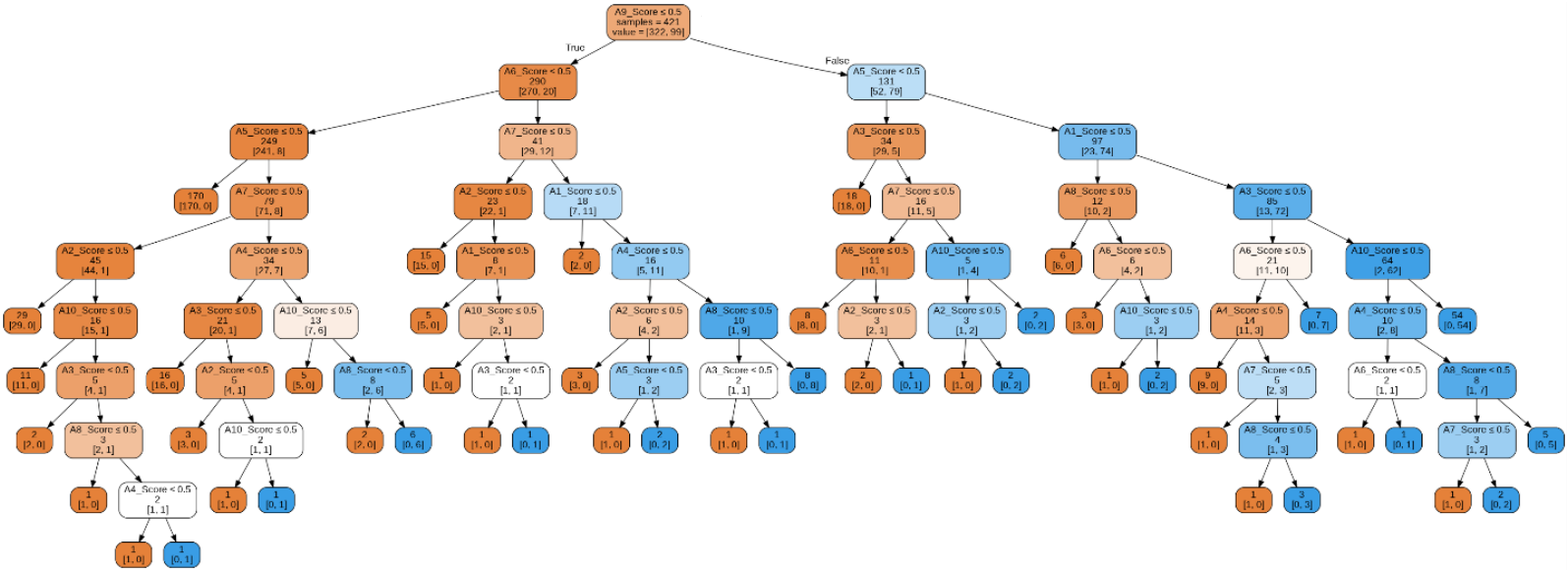
Check the accuracy of the model. Model accuracy is the highest so far from all the models from have run on the other predictors.

```
# Check model accuracy
classificationSummary(train_Y, fullScoreTree.predict(train_X))
classificationSummary(valid_Y, fullScoreTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual   0   1
     0 322   0
     1   0  99
Confusion Matrix (Accuracy 0.9397)

       Prediction
Actual   0   1
     0 186   6
     1  11  79
```

Plot the tree. The initial split is on A9_Score which splits into A6_Score and A5_Score.

```
# Plot the tree
plotDecisionTree(fullScoreTree, feature_names=train_X.columns)
```



Set parameters to run through a gridsearch.

```
# Create parameters to optimize the decision tree
param_grid = {
    'max_depth':[3, 4, 5, 7, 8],
    'min_samples_split': [0.05, 0.01, 0.001, 0.005],
    'min_impurity_decrease': [0.05, 0.02, 0.01, 0.001, 0.005]
}
```

```
# Run a grid search to find the best parameters
gridsearch = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid, cv=5, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

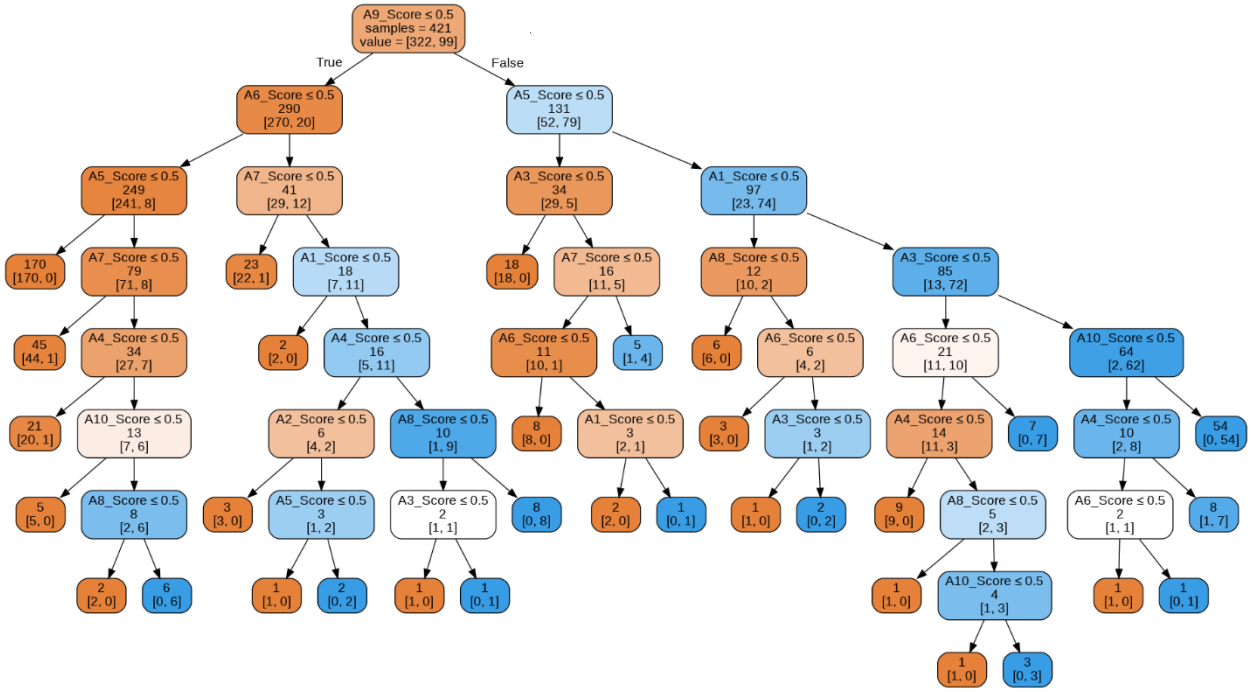
Check for the best parameters.

```
# Check for best parameters
gridsearch.best_params_
```

```
{'max_depth': 8, 'min_impurity_decrease': 0.001, 'min_samples_split': 0.001}
```

Optimize the decision tree and run the model.

```
# Optimize the tree and plot the model
gridScoreTree = DecisionTreeClassifier(max_depth=8,
                                       min_impurity_decrease=0.001,
                                       min_samples_split=0.001,
                                       random_state=1)
gridScoreTree.fit(train_X, train_Y)
plotDecisionTree(gridScoreTree, feature_names = train_X.columns)
```



The accuracy of the model has decreased slightly after optimization.

```
# Check model accuracy
classificationSummary(train_Y, gridScoreTree.predict(train_X))
classificationSummary(valid_Y, gridScoreTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9881)

       Prediction
Actual   0   1
     0 320   2
     1   3  96
Confusion Matrix (Accuracy 0.9326)

       Prediction
Actual   0   1
     0 186   6
     1  13  77
```

5.4.1 Random Forest:

Create parameters to run through a gridsearch.

```
# Set random forest parameters
param_grid = {
    'max_depth':[5, 6, 8, 9],
    'min_samples_split': [0.01, 0.001, 0.005],
    'min_impurity_decrease': [0.01, 0.001, 0.005],
    'n_estimators':[300, 400, 500]
}
```

```
# Perform a grid search for the best parameters
gridsearch = GridSearchCV(RandomForestClassifier(random_state=1), param_grid, cv=3, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

Check for the best parameters.

```
# Check for best parameters
gridsearch.best_params_
```

```
{'max_depth': 8,
 'min_impurity_decrease': 0.001,
 'min_samples_split': 0.01,
 'n_estimators': 400}
```

Optimize the random forest and run the model.

```
# Run the random forest
scoreForest = RandomForestClassifier(random_state=1, n_estimators=400,
                                     max_depth=8, min_impurity_decrease=0.001,
                                     min_samples_split=0.001)
scoreForest.fit(train_X, train_Y)
```

Create variables for feature importance and standard deviation.

```
# Create variables for feature importance and standard deviation
importance = scoreForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in scoreForest.estimators_], axis=0)
```

Create a dataframe to look at feature importance and standard deviation.

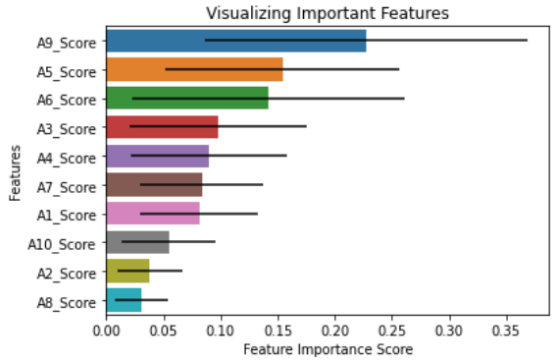
```
# Check feature importance and standard deviation
scoreForest_df = pd.DataFrame({'feature': train_X.columns,
                  'importance': importance,
                  'std': std})
print(scoreForest_df.sort_values('importance', ascending=False))
```

```
     feature  importance       std
8   A9_Score    0.227616  0.141566
4   A5_Score    0.154065  0.102864
5   A6_Score    0.142026  0.119462
2   A3_Score    0.097954  0.078088
3   A4_Score    0.090108  0.068261
6   A7_Score    0.083610  0.053694
0   A1_Score    0.081173  0.051162
9  A10_Score    0.054517  0.041359
1   A2_Score    0.037943  0.028337
7   A8_Score    0.030988  0.022953
```

Create a barplot to visualize feature importance in descending order. A9_Score, A5_Score, and A6_Score have the highest feature importance.

```
# Plot feature importance
value_plot = scoreForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```

```
Text(0.5, 1.0, 'Visualizing Important Features')
```



Accuracy of the model is greater than the accuracy of the decision trees.

```
# Check model accuracy
classificationSummary(train_Y, scoreForest.predict(train_X))
classificationSummary(valid_Y, scoreForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9905)

       Prediction
Actual   0   1
     0 321   1
     1   3  96
Confusion Matrix (Accuracy 0.9610)

       Prediction
Actual   0   1
     0 190   2
     1   9  81
```

5.4.2 Regression:

Run a logistic regression.

```
# Run a logistic regression with C=1
score_log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
score_log_reg.fit(train_X, train_Y)
```

The accuracy for the model is greater than the accuracy for decision trees and random forests.

```
# Check model accuracy
classificationSummary(train_Y, score_log_reg.predict(train_X))
classificationSummary(valid_Y, score_log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9786)

       Prediction
Actual   0   1
     0 319   3
     1   6  93
Confusion Matrix (Accuracy 0.9681)

       Prediction
Actual   0   1
     0 190   2
     1   7  83
```

Create a dataframe for the variable coefficients and odds ratio. The features with the strongest impact on the target outcome match the previous model output.

```
# Check variable coefficients and odds ratio
score_result = pd.DataFrame({'coef': score_log_reg.coef_[0], 'odds': np.e**score_log_reg.coef_[0]}, index=X.columns)
print(score_result.sort_values('odds', ascending=False))
```

```
               coef      odds
A9_Score   1.463563  4.321329
A6_Score   1.402261  4.064379
A5_Score   1.351028  3.861394
A7_Score   1.314564  3.723128
A3_Score   1.042488  2.836264
A4_Score   1.013283  2.754629
A1_Score   0.907248  2.477496
A2_Score   0.786157  2.194945
A10_Score  0.690719  1.995149
A8_Score   0.543986  1.722861
```

5.4.3 Neural Network:

Create optional hidden layer sizes to run through a gridseach.

```
# Create optional hidden layer sizes for neural network
neural_param = {'hidden_layer_sizes': [1, 2, 3, 4, 5, 6, 7, 8]}
```

```
# Run a gridsearch to find the best hidden layer size
gridsearch= GridSearchCV(MLPClassifier(activation='logistic', solver='lbfgs', random_state=1, max_iter=500),
                         param_grid=neural_param, cv=5, n_jobs=-1, return_train_score=True)
gridsearch.fit(train_X, train_Y)
```

The best hidden layer size is one hidden layer.

```
# Check for best parameters
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 1}
```

Run the optimized neural network.

```
# Run a neural network with 3 hidden layers
clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=3, random_state=1)
clf.fit(train_X, train_Y)
```

This is the highest model accuracy out of all the models I have run.

```
# Check model accuracy
classificationSummary(valid_Y, clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9965)

       Prediction
Actual   0   1
     0 192   0
     1   1  89
```

```
# Look at model probability prediction
pd.DataFrame(clf.predict_proba(valid_X))
```

	0	1
0	1.000000	2.936072e-30
1	1.000000	1.716808e-25
2	1.000000	1.587530e-26
3	1.000000	8.167031e-25
4	1.000000	8.258144e-27
...	...	...
277	1.000000	3.059181e-29
278	1.000000	4.819816e-30
279	0.987030	1.297030e-02
280	1.000000	3.059475e-30
281	0.000001	9.999988e-01

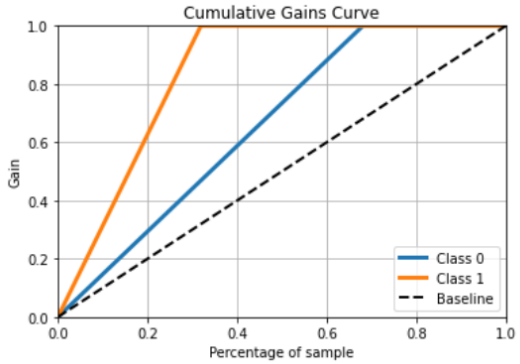
282 rows × 2 columns

In this cumulative gains chart predictions for ‘YES’ ASD classifications optimize at 30%, while predictions for ‘NO’ ASD classifications optimize at 70%.

```
# Create a variable for model prediction
pred = clf.predict_proba(valid_X)
```

```
# Run a cumulative gains chart
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, pred)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



5.5 Model Comparison for Full Dataset Analysis:

Datasets	Maximal Trees	Optimized Trees	Random Forest	Logistic Regression	Neural Network
Full Dataset	92.2%	91.1%	92.9%	98.9%	95.7%
Categorical Variables	73.0%	73.0%			
Continent instead of Country	87.9%	91.1%			
_Score variables only	93.9%	93.2%	96.1%	96.8%	99.6%

For the full dataset analysis and the _Score variable analysis the best model was the neural network. Looking at these neural networks we can see that the model is better at prediction a positive diagnosis of ASD than a negative diagnosis. For the analysis on demographic variables and the analysis using continent_of_res the best model was the optimized decision tree. Model accuracy does not improve when using continent_of_res as a predictor instead of countr_of_res. Throughout the models in all of our analyses the _Score variables were the most predictive of the target variable, particularly A9_Score, A6_Score, and A5_Score. When they were excluded from the analysis accuracy drops dramatically.

5.6 Conclusion:

Through this analysis I have verified that the best predictors for the target variable are the _Score variables obtained from the AQ-10. This supports our approach to the analysis for our female subgroups and ethnic subgroups where I will be assessing responses to the AQ-10 represented by

the _Score variables. By running an analysis of the full dataset on all the possible predictors I have hopefully reduced bias in my future steps.

6.0 Modelling Gender Differences (Females):

Create a new dataset that only contains female records.

```
# Create a Female-only dataset from the autism dataset.
female_df = autism_df[(autism_df['gender']=='f')]
female_df.shape
```

```
(336, 21)
```

Set the predictors and target then create dummies for the target variable. Split the dataset into training and validation sets.

```
# Split the dataset with Class/ASD as the target variable and _Score variables as predictors.
predictors = ['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score']
outcome = 'Class/ASD'

x= female_df[predictors]
y= female_df[outcome]
# Convert categorical vairables to dummies.
X = x
Y = pd.get_dummies(y, drop_first=True)

train_X, valid_X, train_Y, valid_Y = train_test_split(X, Y, test_size=0.4, random_state=1)
```

6.0.1 Decision Trees:

Create a maximal decision tree.

```
# Run a maximal tree
fullClassTree = DecisionTreeClassifier(random_state=1)
fullClassTree.fit(train_X, train_Y)
```

```
DecisionTreeClassifier(random_state=1)
```

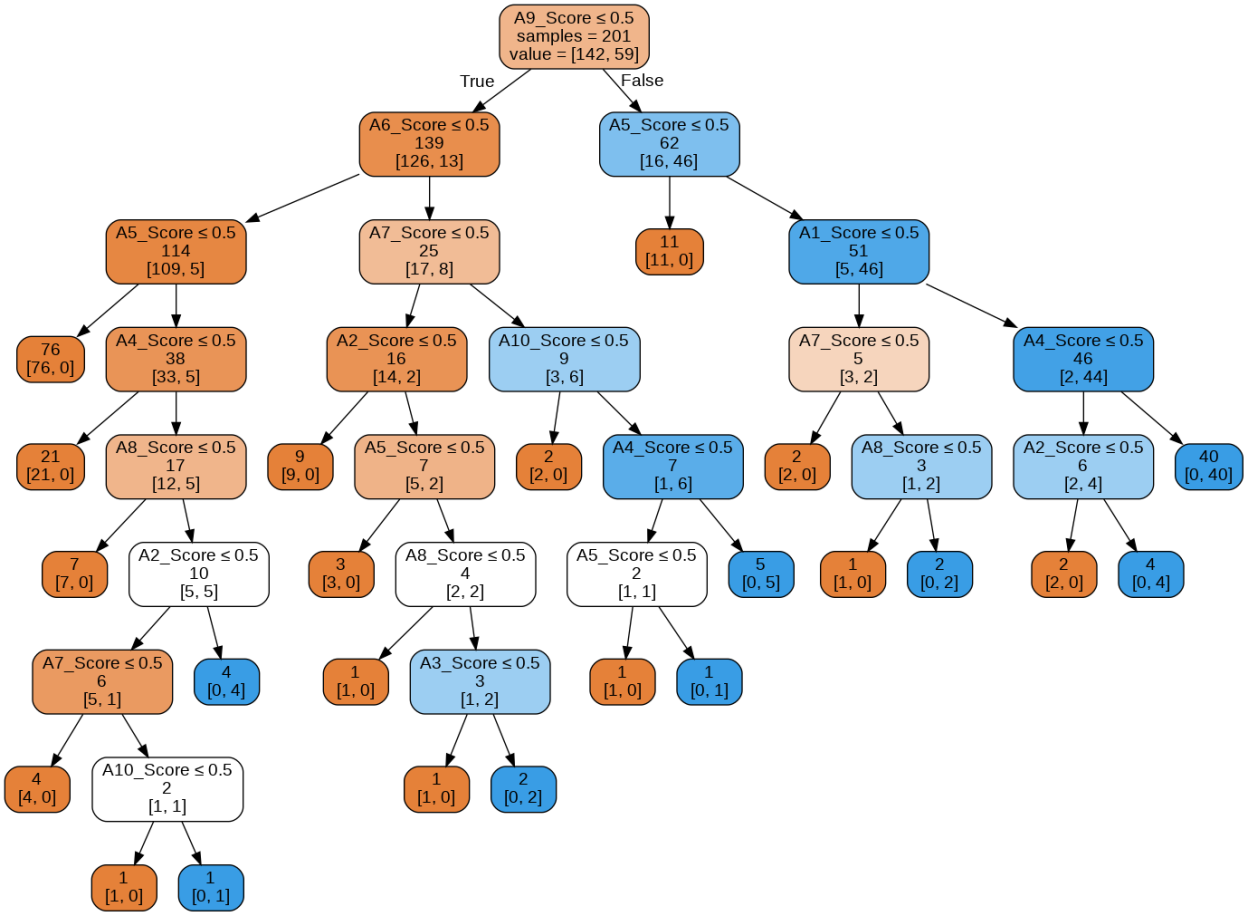
Check the model accuracy.

```
# Check model accuracy
classificationSummary(train_Y,fullClassTree.predict(train_X))
classificationSummary(valid_Y,fullClassTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual   0   1
     0 142   0
     1   0  59
Confusion Matrix (Accuracy 0.9185)

       Prediction
Actual  0  1
     0 85  6
     1  5 39
```

Plot the Decision Tree. The first split is on A9_Score which splits into A5_Score and A6_Score.

```
# Plot the tree
plotDecisionTree(fullClassTree, feature_names=train_X.columns)
```



Next, I am going to create an optimized decision tree. Set optional parameters for maximum depth of the tree, the minimum number of splits, and the minimum log worth for each variable in the tree.

```
# Create parameters to optimize the tree
param_grid = {
    'max_depth':[2, 3, 5, 7, 9],
    'min_samples_split': [0.07, 0.05, 0.01, 0.005, 0.001],
    'min_impurity_decrease': [0.02, 0.01, 0.001, 0.005, 0.0001]
}
```

Using the gridsearch method run all combinations of the parameters. Set cross-validation to 5.

```
# Run a grid search to find the best parameters
gridsearch = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid, cv=5, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=1), n_jobs=-1,
             param_grid={'max_depth': [2, 3, 5, 7, 9],
                         'min_impurity_decrease': [0.02, 0.01, 0.001, 0.005,
                                                   0.0001],
                         'min_samples_split': [0.07, 0.05, 0.01, 0.005, 0.001]})
```

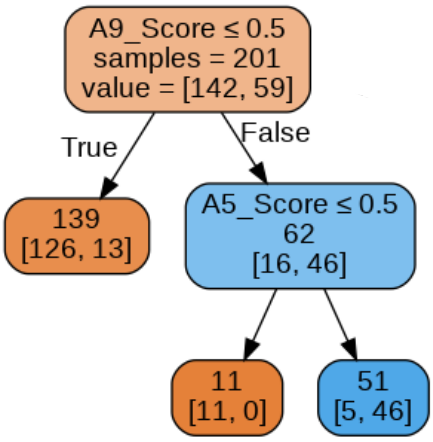
Check the grid search for the best parameters. Our best parameters to optimize the decision tree are a maximum depth of 2, a minimum logworth threshold of 0.02, and minimum splits of 0.07.

```
# Check for the best parameters
gridsearch.best_params_
```

```
{'max_depth': 2, 'min_impurity_decrease': 0.02, 'min_samples_split': 0.07}
```

Create the optimized tree using the parameters selected by the grid search. Plot the tree. A9_Score and A5_Score are the most significant variables.

```
# Run the optimized tree and plot.
gridClassTree = DecisionTreeClassifier(max_depth=2,
                                       min_impurity_decrease=0.02,
                                       min_samples_split=0.07,
                                       random_state=1)
gridClassTree.fit(train_X, train_Y)
plotDecisionTree(gridClassTree, feature_names = train_X.columns)
```



Check the model accuracy. The accuracy for this model is lower than the accuracy for the maximal tree. This is likely because the maximal tree was overfitting to the data.

```
# Check model accuracy
classificationSummary(train_Y, gridClassTree.predict(train_X))
classificationSummary(valid_Y, gridClassTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9104)

       Prediction
Actual   0   1
     0 137   5
     1  13  46
Confusion Matrix (Accuracy 0.8519)

       Prediction
Actual  0  1
     0 83  8
     1 12 32
```

6.0.2 Random Forest:

Create parameters to optimize the random forest model for maximum depth, minimum splits, minimum logworth, and the number of estimators.

```
# Set random forest parameters
param_grid = {
    'max_depth':[6, 7, 8],
    'min_samples_split': [0.05, 0.01, 0.001],
    'min_impurity_decrease': [0.01, 0.001, 0.005],
    'n_estimators':[100, 200, 300]
}
```

Use the grid search method to check for the best parameters. Set the cross-validation to 3.

```
# Perform a grid search for the best parameters
gridsearch = GridSearchCV(RandomForestClassifier(random_state=1), param_grid, cv=3, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

Check the best parameters to optimize the model.

```
# Check for best variables
gridsearch.best_params_
```

```
{'max_depth': 6,
 'min_impurity_decrease': 0.001,
 'min_samples_split': 0.01,
 'n_estimators': 200}
```

Create the random forest with the optimal parameters.

```
# Run a random forest
randomForest = RandomForestClassifier(random_state=1, n_estimators=200,
                                      max_depth=6, min_impurity_decrease=0.001,
                                      min_samples_split=0.01)
randomForest.fit(train_X, train_Y)
```

Create variables to check the feature importance and standard deviation for each variable.

```
# Calculate feature importance and standard deviation
importance = randomForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in randomForest.estimators_], axis=0)
```

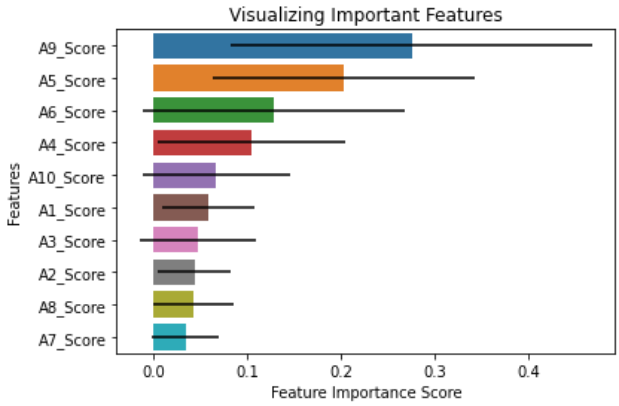
Create a dataframe for each feature, their importance, and standard deviation.

```
# Check feature importance and standard deviation
randomForest_df = pd.DataFrame({'feature': train_X.columns,
                   'importance': importance,
                   'std': std})
print(randomForest_df.sort_values('importance', ascending=False))
```

```
     feature  importance       std
8   A9_Score    0.275067  0.193068
4   A5_Score    0.202490  0.139482
5   A6_Score    0.128100  0.139457
3   A4_Score    0.104006  0.100614
9  A10_Score    0.066508  0.078986
0   A1_Score    0.057989  0.048592
2   A3_Score    0.046824  0.062337
1   A2_Score    0.043036  0.038498
7   A8_Score    0.041818  0.043289
6   A7_Score    0.034162  0.035681
```

Create a barplot that shows the feature importance of each variable, ranked from high to low importance. Like the decision tree models, A9_Score, A6_Score, and A5_Score.

```
# Plot feature importance
value_plot = randomForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```



Check the accuracy of the model. The accuracy is higher than it was for the decision trees.

```
# Check model accuarcy
classificationSummary(train_Y, randomForest.predict(train_X))
classificationSummary(valid_Y, randomForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9950)

       Prediction
Actual   0   1
     0 142   0
     1   1  58
Confusion Matrix (Accuracy 0.9407)

       Prediction
Actual  0  1
     0 87  4
     1  4 40
```

6.0.3 Logistic Regression:

Create a logistic regression model. Inverse regularization of strength (C) is set to 1.

```
# Run a logistic regression with C=1
log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
log_reg.fit(train_X, train_Y)
```

Check the accuracy of the model. Accuracy is lower than the random forest, but higher than the decision trees.

```
# Check model accuracy
classificationSummary(train_Y, log_reg.predict(train_X))
classificationSummary(valid_Y, log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9602)

       Prediction
Actual   0   1
     0 139   3
     1   5  54
Confusion Matrix (Accuracy 0.9333)

       Prediction
Actual  0  1
     0 84  7
     1  2 42
```

Create a dataframe to check the coefficients and odds ratio for each variable. A positive response to A9_Score makes a person 4.6x more likely to be diagnosed with autism. A positive response to A5_Score and A6_Score makes a person 3.8x and 3.1x more likely to be diagnose with autism respectively.

```
# Check variable coefficients and odds ratio
log_result = pd.DataFrame({'coef': log_reg.coef_[0], 'odds': np.e**log_reg.coef_[0]}, index=X.columns)
print(log_result.sort_values('coef', ascending=False))
```

```
              coef      odds
A9_Score   1.533162  4.632801
A5_Score   1.355721  3.879556
A6_Score   1.133214  3.105622
A4_Score   0.835397  2.305730
A3_Score   0.645770  1.907454
A1_Score   0.605536  1.832235
A2_Score   0.589131  1.802421
A8_Score   0.511200  1.667291
A10_Score  0.495990  1.642124
A7_Score   0.458409  1.581555
```

6.0.4 Neural Network:

Create optional hidden layer sizes to optimize the neural network.

```
# Create optional hidden layers for neural network
clf_param = {'hidden_layer_sizes': [1, 2, 3, 4, 5, 6, 7, 8]}
```

Use the gridsearch method to find the best hidden layer size for the network.

```
# Run a grid search to find the best number of hidden layers
gridsearch= GridSearchCV(MLPClassifier(activation='logistic', solver='lbfgs', random_state=1, max_iter=500),
                         param_grid=clf_param, cv=5, n_jobs=-1, return_train_score=True)
gridsearch.fit(train_X, train_Y)
```

The best hidden layer size for the neural network is 7 layers.

```
# Find the best hidden layer size
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 7}
```

Run the neural network with the optimal hidden layers.

```
# Run a neural network with 7 hidden layers
clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=7, random_state=1)
clf.fit(train_X, train_Y)
```

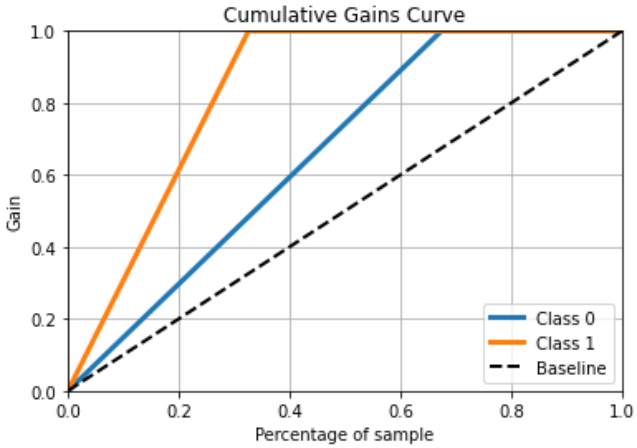
Check the accuracy of the model. Accuracy is 100%.

```
# Check model accuracy
classificationSummary(valid_Y, clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual  0  1
     0 91  0
     1  0 44
```

Create a variable for the predicted probabilities. Import the skplt package and create a cumulative gains chart. For this model, predictions of ‘YES’ classifications optimize at 30% and predictions of ‘NO’ classifications optimize at 65%.

```
pred = clf.predict_proba(valid_X)
# Run a cumulative gains chart
import scikitplot as skplt
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, pred)
plt.show()
```



6.0.5 Association Rules:

Create a dataset to run the association rules for _Score variables. Set the ID column as the index.

```
# Create dataset using _Score variables and ID
scores_df = female_df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'ID']]
# Set ID as index
scores_df.set_index('ID', inplace=True)
scores_df.head()
```

ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score
0	1	1	1	1	0	0	1	1	0	0
3	1	1	0	1	0	0	1	1	0	1
4	1	0	0	0	0	0	0	1	0	0
6	0	1	0	0	0	0	0	1	0	0
11	0	1	0	1	1	1	1	0	0	1

Set parameters for apriori rules that will be used to create the association rules. The minimum support for itemsets is 25% frequency.

```
# Check itemset frequency with a minimum of 15% support using apriori
questionsets = apriori(scores_df, min_support=.25, use_colnames=True)
questionsets.head()
```

	support	itemsets
0	0.755952	(A1_Score)
1	0.479167	(A2_Score)
2	0.458333	(A3_Score)
3	0.526786	(A4_Score)
4	0.520833	(A5_Score)

Run the association rules using a confidence level of 50% as the threshold for important itemsets.

Sort the associations by lift.

```
# Use association rules to check relations between _Scores. Use minimum 50% confidence as metric.
rules = association_rules(questionsets, metric='confidence', min_threshold=0.5)
rules.sort_values(by=['lift'], ascending=False).head()
```

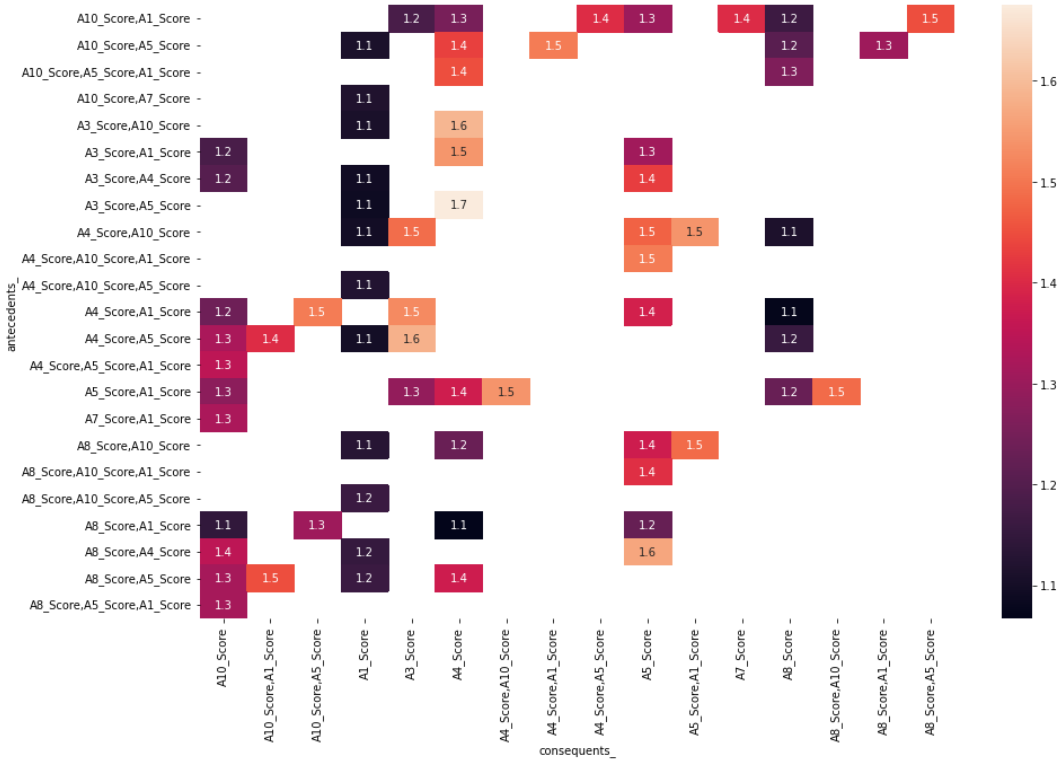
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
103	(A4_Score)	(A3_Score, A5_Score)	0.526786	0.303571	0.267857	0.508475	1.674975	0.107940	1.416872
100	(A3_Score, A5_Score)	(A4_Score)	0.303571	0.526786	0.267857	0.882353	1.674975	0.107940	4.022321
42	(A5_Score)	(A9_Score)	0.520833	0.321429	0.270833	0.520000	1.617778	0.103423	1.413690
43	(A9_Score)	(A5_Score)	0.321429	0.520833	0.270833	0.842593	1.617778	0.103423	3.044118
106	(A3_Score, A10_Score)	(A4_Score)	0.312500	0.526786	0.261905	0.838095	1.590960	0.097284	2.922794

Create a heatmap that maps the associations between variables according to lift.

```
# Create a heatmap to plot association between _Scores
rules['lhs items'] = rules['antecedents'].apply(lambda x:len(x) )
rules[rules['lhs items']>1].sort_values('lift', ascending=False).head()

# Replace frozen sets with strings
rules['antecedents_'] = rules['antecedents'].apply(lambda a: ','.join(list(a)))
rules['consequents_'] = rules['consequents'].apply(lambda a: ','.join(list(a)))

# Transform the DataFrame of rules into a matrix using the lift metric
pivot = rules[rules['lhs items']>1].pivot(index = 'antecedents_',
                    columns = 'consequents_', values= 'lift')


# Replace frozen sets with strings
rules['antecedents_'] = rules['antecedents'].apply(lambda a: ','.join(list(a)))
rules['consequents_'] = rules['consequents'].apply(lambda a: ','.join(list(a)))

# Transform the DataFrame of rules into a matrix using the lift metric
pivot = rules[rules['lhs items']>1].pivot(index = 'antecedents_',
                    columns = 'consequents_', values= 'lift')

# Generate a heatmap with annotations on and the colorbar off
plt.figure(figsize = (15, 10))
sns.heatmap(pivot, annot = True, fmt='.1f')
plt.yticks(rotation=0)
plt.xticks(rotation=90)
plt.show()
```

For females, the three strongest associations are:

- A3_Score & A5_Score -> A4_Score
- A8_Score & A4_Score -> A5_Score
- A3_Score & A10_Score -> A4_Score



7.0 Modeling Ethnic Groups (Females):

I will be examining 5 ethnic groups: White females, Black females, Asian females, Middle Eastern females, and the unidentified Others group.

Change the value name of ‘White-European to ‘White’.

```
# Change value 'White-European' to 'White'
female_df['ethnicity'] = female_df['ethnicity'].replace(['White-European'], 'White')
```

Create a new dataframe for each of the ethnicities.

```
# Create a dataset for White females
white_df = female_df[(female_df['ethnicity']=='White')]
white_df.shape
```

```
(124, 21)
```

```
#Create dataset for Black females
black_df = female_df[(female_df['ethnicity']=='Black')]
```

```
# Create dataset for Asian ethnicity
asian_df = female_df[(female_df['ethnicity']=='Asian')]
```

Remove the space from 'Middle Eastern ' to 'Middle Eastern'.

```
# Replace value 'Middle Eastern ' with 'Middle Eastern'
female_df['ethnicity'] = female_df['ethnicity'].replace(['Middle Eastern '], 'Middle Eastern')

# Create dataset for Middle Eastern ethnicity
eastern_df = female_df[(female_df['ethnicity']=='Middle Eastern')]
```

```
# Create a dataset for unidentified ethnicity females.
others_df = female_df[(female_df['ethnicity']=='Others')]
```

Set the predictors as the _Score variables and Class/ASD as the target. Convert categorical variables to numeric with get_dummies and split the dataset into training and validation sets. This image shows the example from the Others subgroup. The same process was followed for all the other subgroups and the data was split on each unique dataset.

```
# Split the dataset with Class/ASD as the target variable, using only _Scores as perdictors.
predictors = ['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score']
outcome = 'Class/ASD'

x= others_df[predictors]
y= others_df[outcome]

X = x
Y = pd.get_dummies(y, drop_first=True)

train_X, valid_X, train_Y, valid_Y = train_test_split(X, Y, test_size=0.4, random_state=1)
```

7.1 Decision Trees:

7.1.0 Maximal Tree- White Females:

```
# Run a maximal tree
whiteFullTree = DecisionTreeClassifier(random_state=1)
whiteFullTree.fit(train_X, train_Y)
```

```
DecisionTreeClassifier(random_state=1)
```

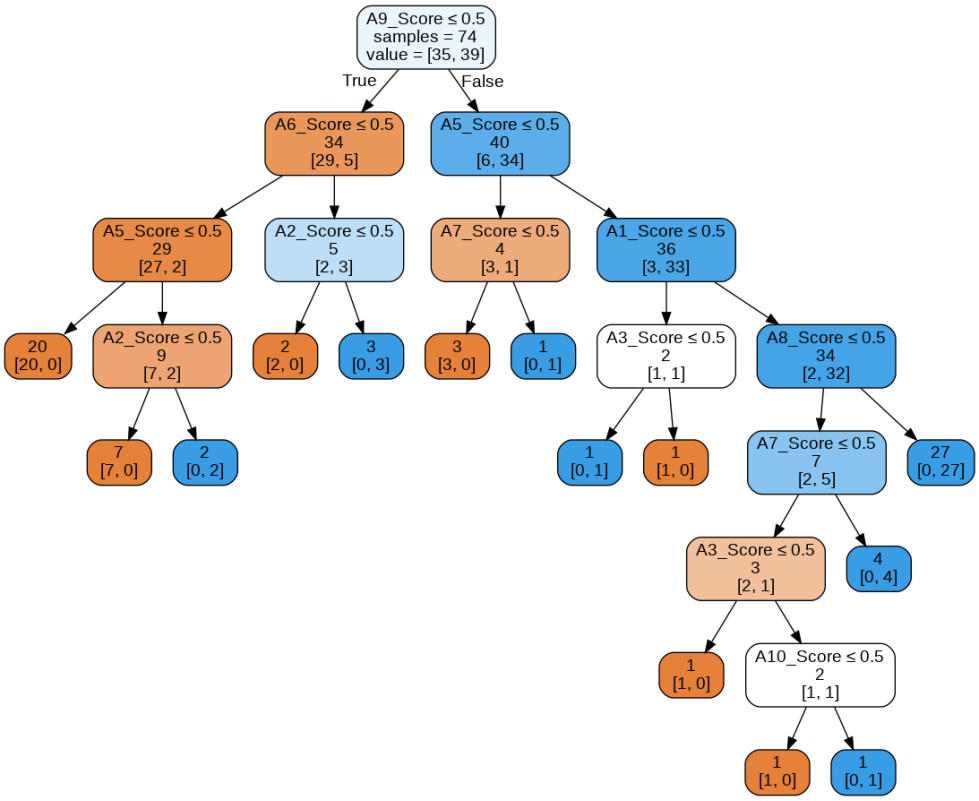
Model accuracy is 88%, which is lower than the accuracy for the full female dataset maximal tree.

```
# Check model accuracy
classificationSummary(valid_Y, whiteFullTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8800)

       Prediction
Actual  0  1
     0 22  3
     1  3 22
```

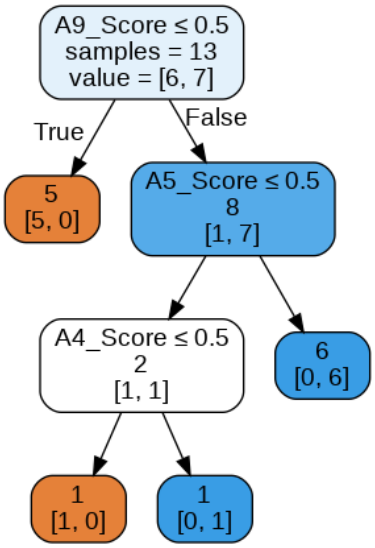
Like the full female dataset, the maximal tree for white females splits on the A9_Score into A6_Score and A5_Score.

```
# Plot the tree
plotDecisionTree(whiteFullTree, feature_names=train_X.columns)
```



7.1.1 Maximal Tree- Black Females:

The maximal tree for Black females differs from the White female tree, with A4_Score replacing A6_Score in importance.



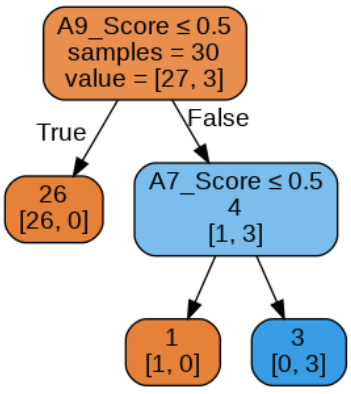
The accuracy for this tree is 70%, significantly lower than the White female tree.

```
# Check accuracy
classificationSummary(valid_Y, blackFullTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.7000)

       Prediction
Actual 0 1
     0 4 2
     1 1 3
```

7.1.2 Maximal Tree- Asian Females:

Asian females differ from the previous two groups, with A7_Score being the second split instead of A5_Score.



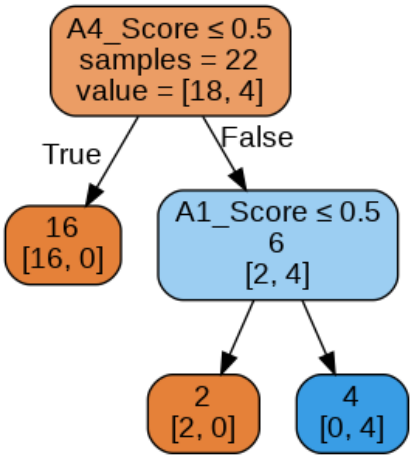
Accuracy for Asian females is higher than for Black females but still lower than for White females.

```
# Check accuracy
classificationSummary(valid_Y, asianFullTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8095)

       Prediction
Actual  0  1
     0 16  2
     1  2  1
```

7.1.3 Maximal Tree- Middle Eastern Females:

This decision tree is the most different from the others with A4_Score as the first split and A1_Score as the second split.



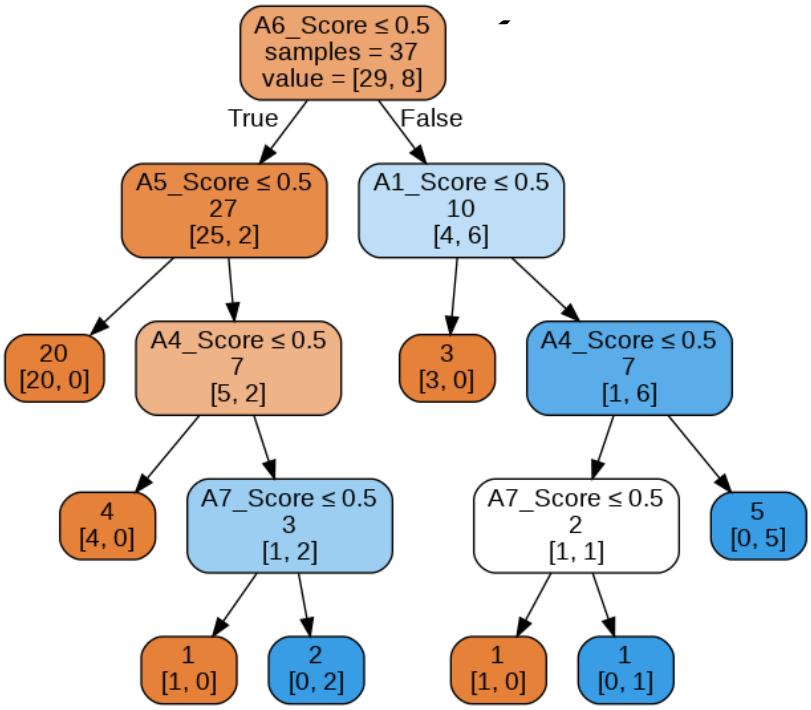
Accuracy for Middle Eastern females is the second highest after the White female tree.

```
# Check for accuracy
classificationSummary(valid_Y, easternFullTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8125)

       Prediction
Actual  0  1
     0 13  3
     1  0  0
```

7.1.4 Maximal Tree- Other Ethnicity:

For the Others subgroup, A6_Score is the first split, and A1_Score and A5_Score are the second splits.



The accuracy for this model is the highest accuracy out of all the subgroups.

```
# Check model accuracy
classificationSummary(valid_Y, othersFullTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9231)

       Prediction
Actual  0  1
     0 21  2
     1  0  3
```

7.1.5 Optimized Tree- White Females:

Some of our maximal trees require further optimization. Set parameters to be selected by a gridsearch for White females.

```
# Set parameters to optimize the tree
param_grid = {
    'max_depth':[3, 5, 7, 8, 9],
    'min_samples_split': [0.05, 0.01, 0.005, 0.001, 0.0001],
    'min_impurity_decrease': [0.05, 0.02, 0.01, 0.001, 0.005]
}
```

Run the grid search.

```
# Run a grid search to find the best parameters
gridsearch = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid, cv=5, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=1), n_jobs=-1,
             param_grid={'max_depth': [3, 5, 7, 8, 9],
                         'min_impurity_decrease': [0.05, 0.02, 0.01, 0.001,
                                                   0.005],
                         'min_samples_split': [0.05, 0.01, 0.005, 0.001,
                                               0.0001]})
```

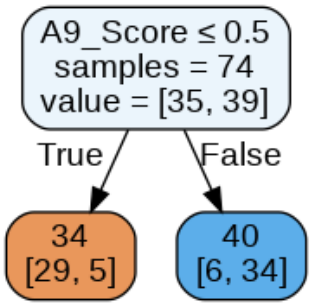
Check for the best parameters.

```
# Check for the best parameters
gridsearch.best_params_
```

```
{'max_depth': 3, 'min_impurity_decrease': 0.05, 'min_samples_split': 0.05}
```

Optimize the tree and run the model. The optimized tree only contains the initial A9_Score split.

```
# Run the optimized tree and plot
whiteClassTree = DecisionTreeClassifier(max_depth=3,
                                        min_impurity_decrease=0.05,
                                        min_samples_split=0.05,
                                        random_state=1)
whiteClassTree.fit(train_X, train_Y)
plotDecisionTree(whiteClassTree, feature_names = train_X.columns)
```



The accuracy of the model actually decreases from 88% to 84%.

```
# Check model accuracy
classificationSummary(valid_Y, whiteClassTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8400)

       Prediction
Actual  0  1
     0 24  1
     1  7 18
```

```
# Check accuracy
classificationSummary(valid_Y, asianClassTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8095)

       Prediction
Actual  0  1
     0 16  2
     1  2  1
```

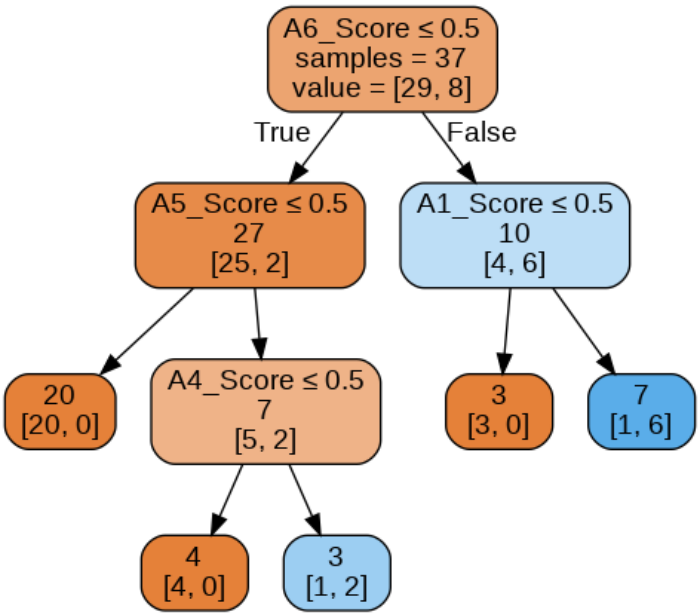
7.1.6 Optimized Tree- Others:

```
# Check for best parameters
gridsearch.best_params_
```

```
{'max_depth': 3, 'min_impurity_decrease': 0.02, 'min_samples_split': 0.07}
```

```
# Plot the optimized tree
othersClassTree = DecisionTreeClassifier(max_depth=3,
                                         min_impurity_decrease=0.02,
                                         min_samples_split=0.07,
                                         random_state=1)
othersClassTree.fit(train_X, train_Y)
plotDecisionTree(othersClassTree, feature_names = train_X.columns)
```

The optimized tree only contains the first 4 splits of A6_Score, A5_Score, A1_Score, and A4_Score.



Accuracy for the optimized tree is lower than the accuracy for the maximal tree.

```
# Check model accuracy
classificationSummary(valid_Y, othersClassTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8846)

       Prediction
Actual  0  1
     0 20  3
     1  0  3
```

7.2 Random Forests:

7.2.0 Random Forest- White Females:

Create parameters to run through the gridsearch.

```
# Set random forest parameters
param_grid = {
    'max_depth':[5, 6, 7, 8],
    'min_samples_split': [0.05, 0.02, 0.01],
    'min_impurity_decrease': [0.01, 0.001, 0.005],
    'n_estimators':[100, 200, 300]
}
```

```
# Perform a grid search for the best parameters
gridsearch = GridSearchCV(RandomForestClassifier(random_state=1), param_grid, cv=3, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

```
# Check best parameters
gridsearch.best_params_
```

```
{'max_depth': 5,
 'min_impurity_decrease': 0.001,
 'min_samples_split': 0.05,
 'n_estimators': 100}
```

Optimize the Random Forest with the best parameters and run the model.

```
# Run a random forest
whiteForest = RandomForestClassifier(random_state=1, n_estimators=100,
                                     max_depth=5, min_impurity_decrease=0.001,
                                     min_samples_split=0.05)
whiteForest.fit(train_X, train_Y)
```

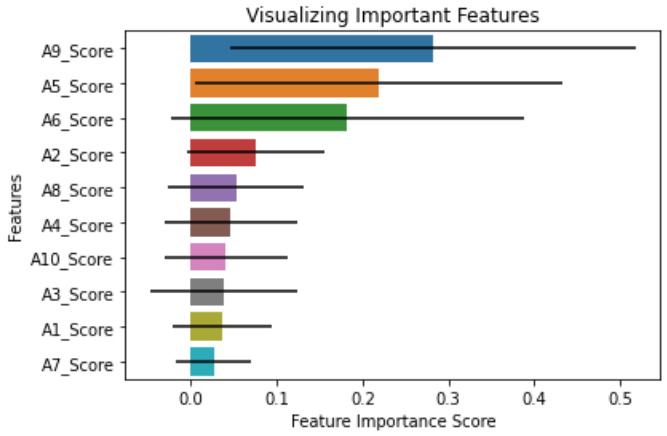
Create a dataframe to check the variable importance and standard deviation.

```
# Check feature importance and standard deviation
importance = whiteForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in whiteForest.estimators_], axis=0)

whiteForest_df = pd.DataFrame({'feature': train_X.columns,
                               'importance': importance,
                               'std': std})
print(whiteForest_df.sort_values('importance', ascending=False))
```

```
     feature  importance       std
8   A9_Score    0.281884  0.236822
4   A5_Score    0.218816  0.214353
5   A6_Score    0.182414  0.205197
1   A2_Score    0.076252  0.080108
7   A8_Score    0.052216  0.079046
3   A4_Score    0.046456  0.077251
9  A10_Score    0.040788  0.071473
2   A3_Score    0.037924  0.085968
0   A1_Score    0.036691  0.057224
6   A7_Score    0.026558  0.043371
```

Run a barplot of feature importance ranked from highest to lowest. For White females, the first 3 feature importances match the full female dataset, however A2_Score and A8_Score have much higher feature importance.

```
# Plot feature importance
value_plot = whiteForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```



The accuracy for this model is much higher than the accuracy for the decision trees.

```
# Check model accuracy
classificationSummary(valid_Y, whiteForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9200)

       Prediction
Actual  0  1
     0 22  3
     1  1 24
```

7.2.1 Random Forest – Black Females:

The best parameters for Black females differ from those for the White females.

```
# Check for best parameters
gridsearch.best_params_
```

```
{'max_depth': 3,
 'min_impurity_decrease': 0.01,
 'min_samples_split': 0.05,
 'n_estimators': 300}
```

```
# Run a random forest
blackForest = RandomForestClassifier(random_state=1, n_estimators=300,
                                     max_depth=3, min_impurity_decrease=0.01,
                                     min_samples_split=0.05)
blackForest.fit(train_X, train_Y)
```

Check the feature importance and standard deviation for each variable.

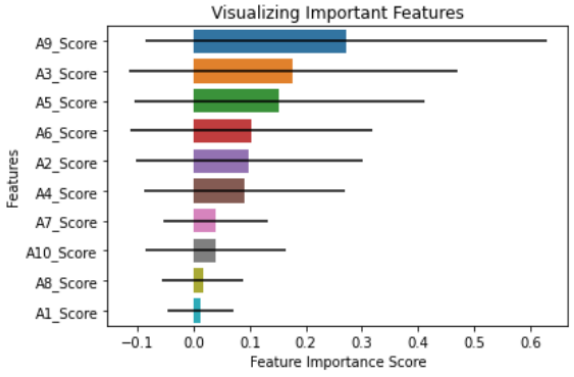
```
# Check importance and standard deviation for features
importance = blackForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in blackForest.estimators_], axis=0)

blackForest_df = pd.DataFrame({'feature': train_X.columns,
                               'importance': importance,
                               'std': std})
print(blackForest_df.sort_values('importance', ascending=False))
```

```
     feature  importance       std
8   A9_Score    0.271758  0.357815
2   A3_Score    0.176914  0.292845
4   A5_Score    0.152766  0.258560
5   A6_Score    0.102678  0.215260
1   A2_Score    0.098706  0.202712
3   A4_Score    0.090564  0.178296
6   A7_Score    0.039104  0.093972
9  A10_Score    0.038944  0.124698
7   A8_Score    0.016473  0.072279
0   A1_Score    0.012094  0.059218
```

Run a barplot to check feature importance, ranked from highest to lowest importance. Unlike White females, A3_Score is the second most important feature, while A8_Score has low feature importance.

```
# Plot feature importance
value_plot = blackForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```

```
Text(0.5, 1.0, 'Visualizing Important Features')
```



Model accuracy for the Random Forest is higher than model accuracy for the decision trees.

```
# Check accuracy
classificationSummary(valid_Y, blackForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8000)

       Prediction
Actual 0 1
     0 4 2
     1 0 4
```

7.2.2 Random Forest- Asian Females:

Asian female Random Forest has different parameters than Black and White females.

```
# Check the best parameters
gridsearch.best_params_
```

```
{'max_depth': 3,
 'min_impurity_decrease': 0.01,
 'min_samples_split': 0.05,
 'n_estimators': 100}
```

Optimize the random forest and run the model.

```
# Run a random forest
asianForest = RandomForestClassifier(random_state=1, n_estimators=100,
                                     max_depth=3, min_impurity_decrease=0.01,
                                     min_samples_split=0.05)
asianForest.fit(train_X, train_Y)
```

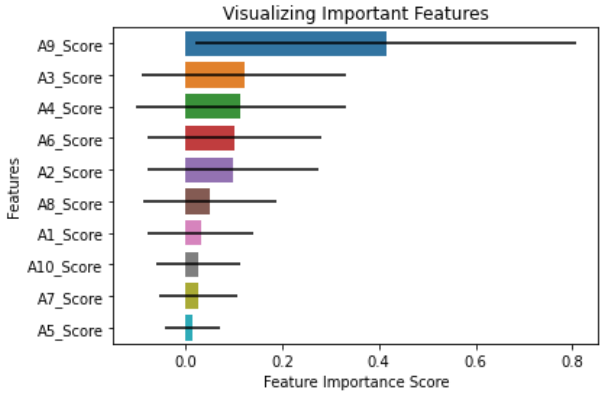
Check the feature importance and standard deviation for each variable.

```
# Check feature importance and standard deviation
importance = asianForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in asianForest.estimators_], axis=0)

asianForest_df = pd.DataFrame({'feature': train_X.columns,
                               'importance': importance,
                               'std': std})
print(asianForest_df.sort_values('importance', ascending=False))
```

```
     feature  importance       std
8   A9_Score    0.414556  0.393491
2   A3_Score    0.121364  0.212006
3   A4_Score    0.114074  0.217054
5   A6_Score    0.102022  0.179625
1   A2_Score    0.097956  0.176649
7   A8_Score    0.050499  0.137293
0   A1_Score    0.031412  0.108647
9  A10_Score    0.027799  0.086716
6   A7_Score    0.026234  0.081163
4   A5_Score    0.014083  0.056238
```

Create a barplot that shows feature importance ranked from highest to lowest. Like Black females, A3_Score is the second highest feature, but A4_Score is also ranked high in importance while A5_Score is the least important.

```
# Plot feature importance
value_plot = asianForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```



Model accuracy for the random forest is higher than the accuracy for the decision trees.

```
# Check accuracy
classificationSummary(valid_Y, asianForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9524)

       Prediction
Actual  0  1
     0 18  0
     1  1  2
```

7.2.3 Random Forest- Middle Eastern Females:

Parameters for Middle Eastern females are the same as for Asian females.

```
# Check best parameters
gridsearch.best_params_
```

```
{'max_depth': 3,
 'min_impurity_decrease': 0.01,
 'min_samples_split': 0.05,
 'n_estimators': 100}
```

Optimize the random forest and run the model.

```
# Run a random forest
easternForest = RandomForestClassifier(random_state=1, n_estimators=100,
                                       max_depth=100, min_impurity_decrease=0.01,
                                       min_samples_split=0.05)
easternForest.fit(train_X, train_Y)
```

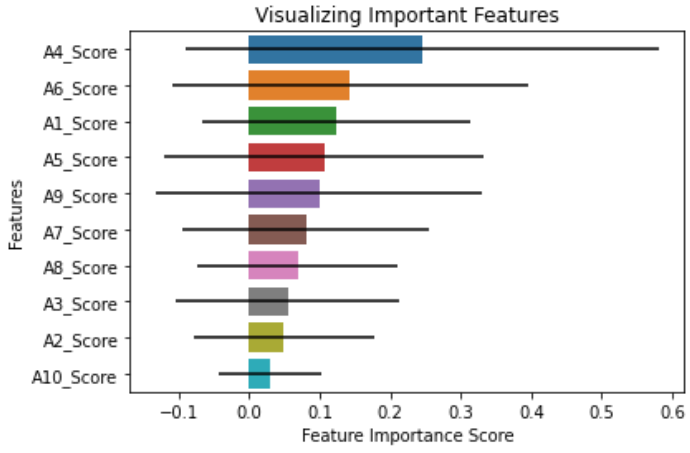
Check feature importance and standard deviation for each variable.

```
# Check variable importance and standard deviation
importance = easternForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in easternForest.estimators_], axis=0)

easternForest_df = pd.DataFrame({'feature': train_X.columns,
                                 'importance': importance,
                                 'std': std})
print(easternForest_df.sort_values('importance', ascending=False))
```

```
     feature  importance       std
3   A4_Score    0.245795  0.335497
5   A6_Score    0.143140  0.252164
0   A1_Score    0.123708  0.190009
4   A5_Score    0.105881  0.226572
8   A9_Score    0.098781  0.232090
6   A7_Score    0.080330  0.174127
7   A8_Score    0.069153  0.141821
2   A3_Score    0.054588  0.158233
1   A2_Score    0.049117  0.127720
9  A10_Score    0.029508  0.073131
```

Use a barplot to check the feature importance for each variable. A4_Score is the most important for this subgroup and A1_Score is the third in importance. Unlike Asian and Black females, A3_Score has low feature importance.

```
# Plot feature importance
value_plot = easternForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```



Model accuracy for the random forest is extremely high

```
# Check accuracy
classificationSummary(valid_Y, easternForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual  0
     0 16
```

7.2.4 Random Forest- Others:

The best parameters for the Others subgroup differ from all the other previous groups.

```
# Check for best paramters
gridsearch.best_params_
```

```
{'max_depth': 6,
 'min_impurity_decrease': 0.01,
 'min_samples_split': 0.05,
 'n_estimators': 200}
```

Optimize the random forest and run the model.

```
# Run a random forest
othersForest = RandomForestClassifier(random_state=1, n_estimators=200,
                                      max_depth=6, min_impurity_decrease=0.01,
                                      min_samples_split=0.05)
othersForest.fit(train_X, train_Y)
```

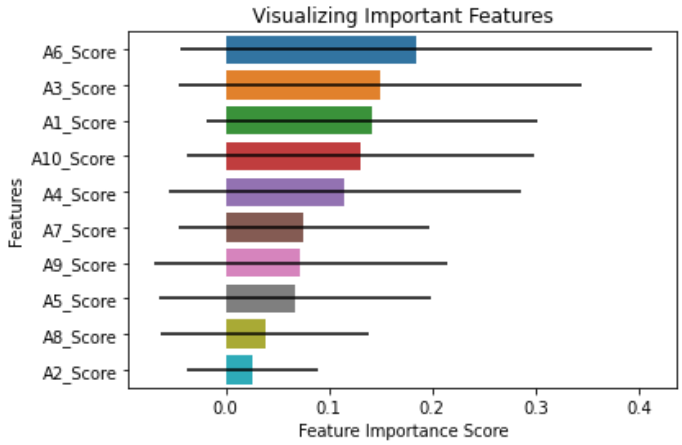
Check the feature importance and standard deviation for each variable.

```
# Check feature importance and standard deviation
importance = othersForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in othersForest.estimators_], axis=0)

othersForest_df = pd.DataFrame({'feature': train_X.columns,
                                'importance': importance,
                                'std': std})
print(othersForest_df.sort_values('importance', ascending=False))
```

```
     feature  importance       std
5   A6_Score    0.184331  0.228697
2   A3_Score    0.149925  0.195401
0   A1_Score    0.141121  0.160155
9  A10_Score    0.130710  0.167918
3   A4_Score    0.115434  0.171144
6   A7_Score    0.075438  0.121132
8   A9_Score    0.072099  0.141851
4   A5_Score    0.066481  0.131708
7   A8_Score    0.038319  0.100929
1   A2_Score    0.026143  0.063714
```

Create a barplot to check feature importance ranked in descending order. A6_Score has the highest feature importance and, like the Middle Eastern subgroup, A3_Score and A1_Score also have high importance.

```
# Plot feature importance
value_plot = othersForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```



Model accuracy for the random forest is higher than the accuracy for the decision trees.

```
# Check accuracy
classificationSummary(valid_Y, othersForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9615)

       Prediction
Actual  0  1
     0 23  0
     1  1  2
```

7.3 Neural Networks:

7.3.0 Neural Network- White Females:

Create optional hidden layer sizes to run through a gridsearch.

```
# Create optional hidden layer sizes for neural network
whiteCLF_param = {'hidden_layer_sizes': [1, 2, 3, 4, 5, 6, 7, 8]}
```

```
# Run a grid search to find optimal hidden layer size
gridsearch= GridSearchCV(MLPClassifier(activation='logistic', solver='lbfgs', random_state=1, max_iter=500),
                         param_grid=whiteCLF_param, cv=5, n_jobs=-1, return_train_score=True)
gridsearch.fit(train_X, train_Y)
```

The best hidden layer size for this model is 4 layers.

```
# Check for best parameters
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 4}
```

Run the optimized neural network.

```
# Run a neural network with 4 hidden layers
white_clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=4, random_state=1)
white_clf.fit(train_X, train_Y)
```

Accuracy for this model is the same as the random forest.

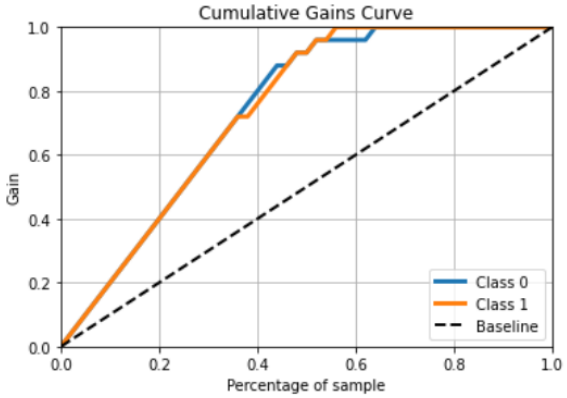
```
# Check model accuracy
classificationSummary(valid_Y, white_clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9200)

       Prediction
Actual  0  1
     0 21  4
     1  0 25
```

Create a cumulative gains chart for the probabilities calculated by the model. For 'YES' ASD classifications, the neural network optimizes at 55% and for the 'NO' ASD classification it optimizes at 63%.

```
# Create a variable for model prediction probabilites
white_clf_pred = white_clf.predict_proba(valid_X)
# Plot a cumulative gains chart
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, white_clf_pred)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



7.3.1 Neural Network- Black Females:

The optimal hidden layer size is 2.

```
# Find best layer size
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 2}
```

Optimize the neural network and run the model.

```
# Run a neural network with 2 hidden layers
black_clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=2, random_state=1)
black_clf.fit(train_X, train_Y)
```

The accuracy for this model is the same as the random forest.

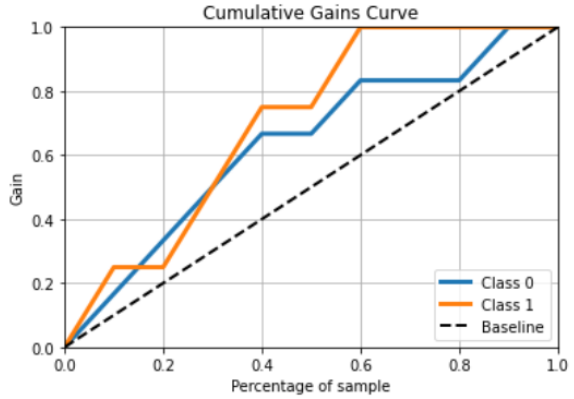
```
# Check model accuracy
classificationSummary(valid_Y, black_clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8000)

       Prediction
Actual 0 1
     0 4 2
     1 0 4
```

Looking at the cumulative gains chart we can see that the neural network optimizes at 60% for 'YES' ASD classifications and 90% for 'NO' ASD classifications.

```
# Create a variable for model prediction probabilities
black_clf_pred = black_clf.predict_proba(valid_X)

# Plot cumulative gains chart
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, black_clf_pred)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



7.3.2 Neural Network- Asian Females:

Best hidden layer size for Asian female neural network is 1 hidden layer.

```
# Check optimal hidden layer size
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 1}
```

Optimize the neural network and run the model.

```
# Run a neural network with 1 hidden layer
asian_clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=1, random_state=1)
asian_clf.fit(train_X, train_Y)
```

Model accuracy is lower than the random forest model.

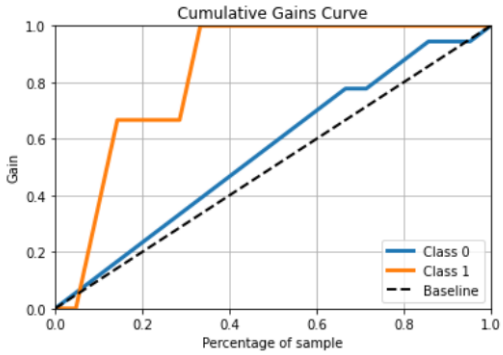
```
# Check model accuracy
classificationSummary(valid_Y, asian_clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8571)

       Prediction
Actual  0  1
     0 16  2
     1  1  2
```

In this cumulative gains chart the neural network optimizes at 35% for ‘YES’ ASD classifications and at 100% for ‘NO’ ASD classifications.

```
# Create a variable for model prediction probabilites
asian_clf_pred = asian_clf.predict_proba(valid_X)

# Plot the cumulative gains chart
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, asian_clf_pred)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



7.3.3 Neural Network- Middle Eastern Females:

The optimal hidden layer size for this subgroup is 5 hidden layers.

```
# Find optimal hidden layer size
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 5}
```

Optimize the neural network and run the model.

```
# Run a neural network with 5 hidden layers
eastern_clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=5, random_state=1)
eastern_clf.fit(train_X, train_Y)
```

Model accuracy is lower than the accuracy for the random forest.

```
# Check model accuracy
classificationSummary(valid_Y, eastern_clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9375)

       Prediction
Actual  0  1
     0 15  1
     1  0  0
```

7.3.4 Neural Network- Others:

The optimal hidden layer size for this subgroup is 7 hidden layers.

```
# Check optimal hidden layer size
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 7}
```

Optimize the neural network and run the model.

```
# Run a neural network with 7 hidden layers
others_clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=7, random_state=1)
others_clf.fit(train_X, train_Y)
```

Model accuracy is the same as the random forest accuracy.

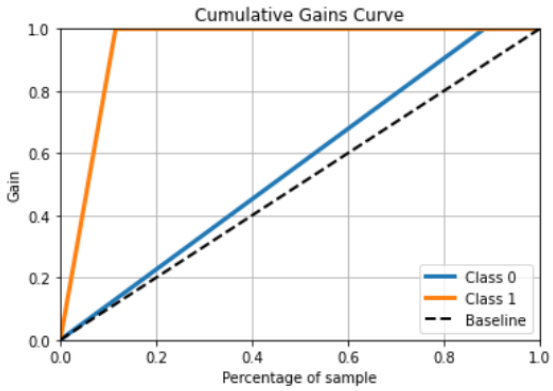
```
# Check model accuracy
classificationSummary(valid_Y, others_clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9615)

       Prediction
Actual  0  1
     0 23  0
     1  1  2
```

In the cumulative gains chart, ‘YES’ ASD classification optimizes at 10% and ‘NO’ ASD classification optimizes at 90%.

```
# Create a variable for model prediction probabilities
others_clf_pred = others_clf.predict_proba(valid_X)

# Plot the cumulative gains chart
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, others_clf_pred)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



7.4 Logistic Regressions:

7.4.0 Logistic Regression- White Females:

Run the model using the default inverse of regularization strength.

```
# Run a logistic regression
white_log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
white_log_reg.fit(train_X, train_Y)
```

Model accuracy is the same as the previous models.

```
# Check model accuracy
classificationSummary(valid_Y, white_log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9200)

       Prediction
Actual  0  1
     0 21  4
     1  0 25
```

Check the model coefficients and odds ratio. A9_Score, A6_Score, and A5_Score have the strongest impact on being classified with ASD.

```
# Check variable coefficients and odds ratio
white_result = pd.DataFrame({'coef': white_log_reg.coef_[0], 'odds': np.e**white_log_reg.coef_[0]}, index=X.columns)
print(white_result.sort_values('coef', ascending=False))
```

```
               coef      odds
A9_Score   1.371763  3.942294
A6_Score   1.330846  3.784242
A5_Score   1.199204  3.317477
A2_Score   0.858799  2.360325
A8_Score   0.402007  1.494822
A7_Score   0.375811  1.456172
A10_Score  0.278772  1.321507
A4_Score   0.189611  1.208779
A3_Score   0.091208  1.095497
A1_Score  -0.294455  0.744937
```

7.4.1 Logistic Regression-Black Females:

```
# Run a logistic regression
black_log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
black_log_reg.fit(train_X, train_Y)
```

Check model coefficients and odds ratio. A9_Score, A4_Score, and A5_Score have the strongest impact on a ‘YES’ ASD classification.

```
# Check variable coefficients and odds ratio
print(pd.DataFrame({'coef': black_log_reg.coef_[0], 'odds': np.e**black_log_reg.coef_[0]}, index=X.columns))
```

```
               coef      odds
A1_Score  -0.255627  0.774430
A2_Score   0.282787  1.326822
A3_Score   0.744271  2.104907
A4_Score   0.496684  1.643263
A5_Score   0.590129  1.804221
A6_Score   0.442558  1.556685
A7_Score  -0.333171  0.716648
A8_Score  -0.170340  0.843378
A9_Score   0.878171  2.406495
A10_Score -0.042026  0.958845
```

The accuracy of this model is the same as the previous models.

```
# Check accuracy
classificationSummary(valid_Y, black_log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8000)

       Prediction
Actual 0 1
     0 4 2
     1 0 4
```

7.4.2 Logistic Regression- Asian Females:

```
# Run a logistic regression
asian_log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
asian_log_reg.fit(train_X, train_Y)
```

For the odds ratio, A6_Score, A4_Score, and A3_Score have the strongest impact on the target outcome.

```
# Check variable coefficients and odds ratio
print(pd.DataFrame({'coef': asian_log_reg.coef_[0], 'odds': np.e**asian_log_reg.coef_[0]}, index=X.columns))
```

```
              coef      odds
A1_Score  -1.241365  0.288989
A2_Score   0.302150  1.352764
A3_Score   0.431721  1.539905
A4_Score   0.415891  1.515721
A5_Score   0.012819  1.012901
A6_Score   0.504595  1.656314
A7_Score   0.054174  1.055668
A8_Score  -0.149180  0.861414
A9_Score   0.968079  2.632883
A10_Score  0.003966  1.003974
```

```
# Check accuracy
classificationSummary(valid_Y, asian_log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8095)

       Prediction
Actual  0  1
     0 17  1
     1  3  0
```

7.4.3 Logistic Regression- Middle Eastern Females:

```
# Run a logistic Regression
eastern_log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
eastern_log_reg.fit(train_X, train_Y)
```

In this odds ratio, A4_Score has the highest impact on the target variable, followed by A6_Score and A5_Score.

```
# Check variable coefficients and odds ratio
print(pd.DataFrame({'coef': eastern_log_reg.coef_[0], 'odds': np.e**eastern_log_reg.coef_[0]}, index=X.columns))
```

```
              coef      odds
A1_Score   0.146552  1.157835
A2_Score  -0.587248  0.555855
A3_Score   0.199677  1.221008
A4_Score   0.950982  2.588250
A5_Score   0.424133  1.528265
A6_Score   0.477727  1.612405
A7_Score   0.350436  1.419686
A8_Score   0.051084  1.052412
A9_Score   0.137483  1.147382
A10_Score -0.036840  0.963831
```

Model accuracy is lower than the random forest and neural network.

```
# Check accuracy
classificationSummary(valid_Y, eastern_log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9375)

       Prediction
Actual  0  1
     0 15  1
     1  0  0
```

7.4.4 Logistic Regression- Others:

```
# Run a logistic regression
others_log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
others_log_reg.fit(train_X, train_Y)
```

For this odds ratio, A6_Score, A10_Score, and A3_Score have the most impact on the target outcome.

```
# Check variable coefficients and odds ratio
print(pd.DataFrame({'coef': others_log_reg.coef_[0], 'odds': np.e**others_log_reg.coef_[0]}, index=X.columns))
```

```
              coef      odds
A1_Score   0.272381  1.313088
A2_Score  -0.282600  0.753821
A3_Score   0.613614  1.847095
A4_Score   0.303012  1.353931
A5_Score   0.223514  1.250463
A6_Score   0.862661  2.369456
A7_Score   0.191211  1.210714
A8_Score   0.001703  1.001704
A9_Score   0.081361  1.084762
A10_Score  0.593402  1.810135
```

```
# Check accuracy
classificationSummary(valid_Y, others_log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9615)

       Prediction
Actual  0  1
     0 23  0
     1  1  2
```

7.5 Association Rules:

7.5.0 Associations- White Female Responses:

Create a dataset to examine associations between _Score variables. Set the ID column as the index.

```
# Create a dataset with _Score variables and ID.
white_scores = white_df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'ID']]
# Set ID as index
white_scores.set_index('ID', inplace=True)
white_scores.head()
```

ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score
0	1	1	1	1	0	0	1	1	0	0
3	1	1	0	1	0	0	1	1	0	1
31	1	0	0	1	1	1	1	1	0	1
33	1	1	1	1	1	1	1	1	1	1
34	1	1	1	1	1	1	1	1	1	1

Set the minimum support to 20% association frequency.

```
# Check for frequent itemsets with 20% minimum support using aprior
questionsets = apriori(white_scores, min_support=.2, use_colnames=True)
questionsets.head()
```

	support	itemsets
0	0.862903	(A1_Score)
1	0.548387	(A2_Score)
2	0.612903	(A3_Score)
3	0.741935	(A4_Score)
4	0.645161	(A5_Score)

Set a minimum confidence threshold of 50% and check for most frequent itemsets.

```
# Check for relations between _Score variables using associatio rules. Use minimum 50% confidence as metric.
white_rules = association_rules(questionsets, metric='confidence', min_threshold=0.5)
white_rules.sort_values(by=['lift'], ascending=False).head()
```

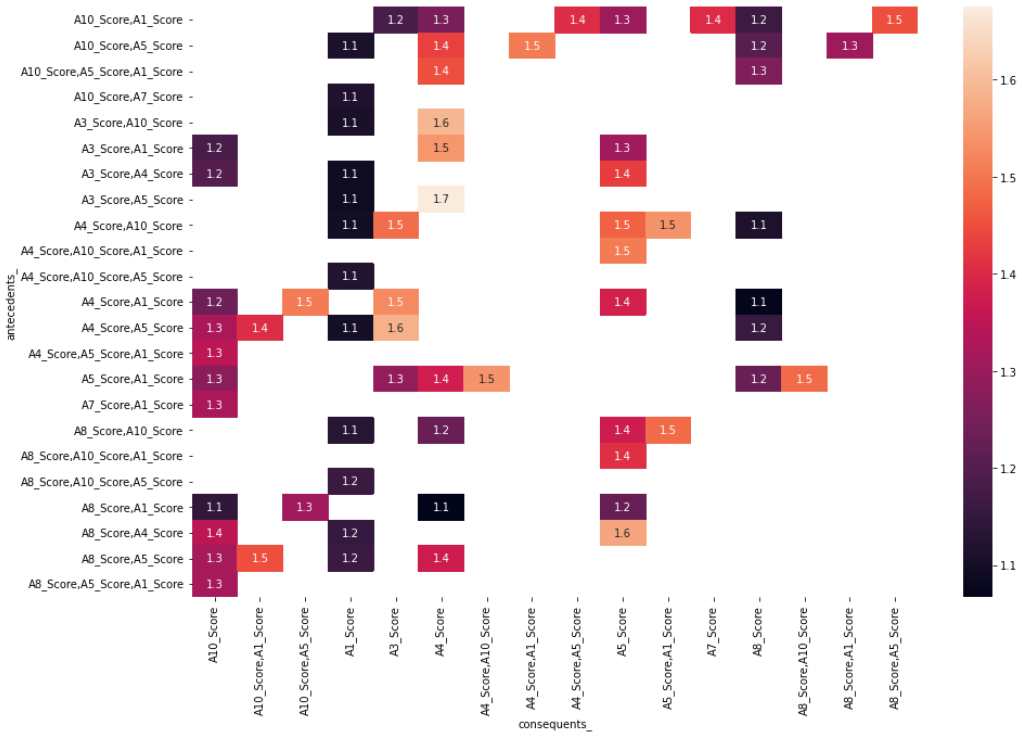
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
6801	(A4_Score, A10_Score, A1_Score, A6_Score)	(A7_Score, A9_Score)	0.330645	0.266129	0.201613	0.609756	2.291205	0.113619	1.880544
6829	(A7_Score, A9_Score)	(A4_Score, A10_Score, A1_Score, A6_Score)	0.266129	0.330645	0.201613	0.757576	2.291205	0.113619	2.761089
7127	(A3_Score, A7_Score, A5_Score)	(A4_Score, A10_Score, A6_Score)	0.250000	0.354839	0.201613	0.806452	2.272727	0.112903	3.333333
7119	(A4_Score, A10_Score, A6_Score)	(A3_Score, A7_Score, A5_Score)	0.354839	0.250000	0.201613	0.568182	2.272727	0.112903	1.736842
7040	(A3_Score, A10_Score, A5_Score)	(A2_Score, A4_Score, A6_Score)	0.387097	0.241935	0.209677	0.541667	2.238889	0.116025	1.653959

Create a heatmap to visualize the association strength between variables.

```
# Create a heatmap to plot association between _Scores
rules['lhs items'] = rules['antecedents'].apply(lambda x:len(x) )
rules[rules['lhs items']>1].sort_values('lift', ascending=False).head()

# Replace frozen sets with strings
rules['antecedents_'] = rules['antecedents'].apply(lambda a: ','.join(list(a)))
rules['consequents_'] = rules['consequents'].apply(lambda a: ','.join(list(a)))

# Transform the DataFrame of rules into a matrix using the lift metric
pivot = rules[rules['lhs items']>1].pivot(index = 'antecedents_',
                    columns = 'consequents_', values= 'lift')


# Replace frozen sets with strings
rules['antecedents_'] = rules['antecedents'].apply(lambda a: ','.join(list(a)))
rules['consequents_'] = rules['consequents'].apply(lambda a: ','.join(list(a)))

# Transform the DataFrame of rules into a matrix using the lift metric
pivot = rules[rules['lhs items']>1].pivot(index = 'antecedents_',
                    columns = 'consequents_', values= 'lift')

# Generate a heatmap with annotations on and the colorbar off
plt.figure(figsize = (15, 10))
sns.heatmap(pivot, annot = True, fmt='.1f')
plt.yticks(rotation=0)
plt.xticks(rotation=90)
plt.show()
```

The three strongest associations for White females are:

- A3_Score & A5_Score → A4_Score
- A8_Score & A4_Score → A5_Score
- A3_Score & A10_Score → A4_Score



7.5.1 Associations- Black Female Responses:

```
# Create dataset with _Score variables and ID
black_scores = black_df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'ID']]
# Set ID column as index
black_scores.set_index('ID', inplace=True)
black_scores.head()
```

ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score
6	0	1	0	0	0	0	0	1	0	0
21	0	0	0	0	0	0	0	0	0	0
51	1	1	1	1	1	1	1	1	1	1
76	1	1	0	1	1	1	1	1	1	0
121	1	0	0	0	0	0	0	0	0	0

Set the minimum support of itemset frequency to 20%.

```
# Identify frequent itemsets using apriori
questionsets = apriori(black_scores, min_support=.2, use_colnames=True) # min_support for itemsets is 20%.
questionsets.head()
```

	support	itemsets
0	0.782609	(A1_Score)
1	0.782609	(A2_Score)
2	0.347826	(A3_Score)
3	0.478261	(A4_Score)
4	0.695652	(A5_Score)

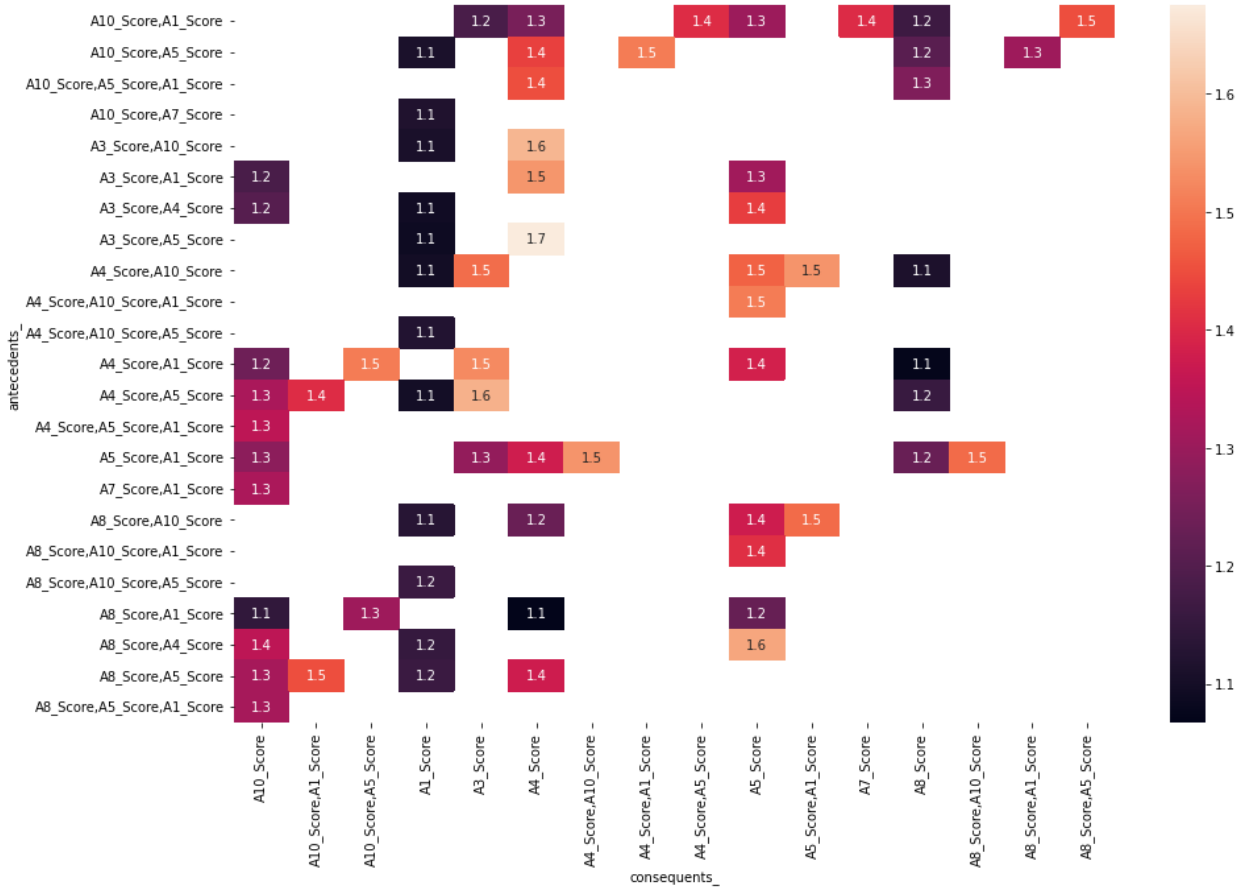
Set the minimum confidence level for associations to 50%.

```
# Check for relationships between _Scores using association rules. Set metric as 'confidence' and min_threshold at 50%.
black_rules = association_rules(questionsets, metric='confidence', min_threshold=0.5)
# Sort values by lift
black_rules.sort_values(by=['lift'], ascending=False).head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2629	(A3_Score, A1_Score, A9_Score)	(A2_Score, A4_Score, A10_Score)	0.217391	0.304348	0.217391	1.000000	3.285714	0.151229	inf
1535	(A3_Score, A1_Score)	(A2_Score, A4_Score, A10_Score)	0.260870	0.304348	0.260870	1.000000	3.285714	0.181474	inf
2578	(A8_Score, A3_Score, A1_Score)	(A2_Score, A4_Score, A10_Score)	0.217391	0.304348	0.217391	1.000000	3.285714	0.151229	inf
2241	(A2_Score, A4_Score, A10_Score)	(A8_Score, A3_Score)	0.304348	0.217391	0.217391	0.714286	3.285714	0.151229	2.73913
2551	(A2_Score, A4_Score, A10_Score, A1_Score)	(A8_Score, A3_Score)	0.304348	0.217391	0.217391	0.714286	3.285714	0.151229	2.73913

The three strongest associations for Black females are the same as those for White females:

- A3_Score & A5_Score → A4_Score
- A8_Score & A4_Score → A5_Score
- A3_Score & A10_Score → A4_Score



7.5.2 Associations- Asian Female Responses:

```
# Create a dataset using _Score variables and ID. Set ID as index.
asian_scores = asian_df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'ID']]
asian_scores.set_index('ID', inplace=True)
asian_scores.head()
```

ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score
41	0	0	0	0	1	1	0	0	0	0
49	1	1	0	0	0	1	1	1	0	1
54	1	1	1	1	1	1	0	1	1	1
100	1	0	0	0	0	0	0	1	0	0
107	1	0	0	1	1	1	1	0	0	0

Set the minimum support frequency for itemsets to 20%.

```
# Check for frequent itemsets using apriori
questionsets = apriori(asian_scores, min_support=.2, use_colnames=True) # Set minimum support to 20%.
questionsets.head()
```

	support	itemsets
0	0.764706	(A1_Score)
1	0.313725	(A2_Score)
2	0.372549	(A3_Score)
3	0.333333	(A4_Score)
4	0.490196	(A5_Score)

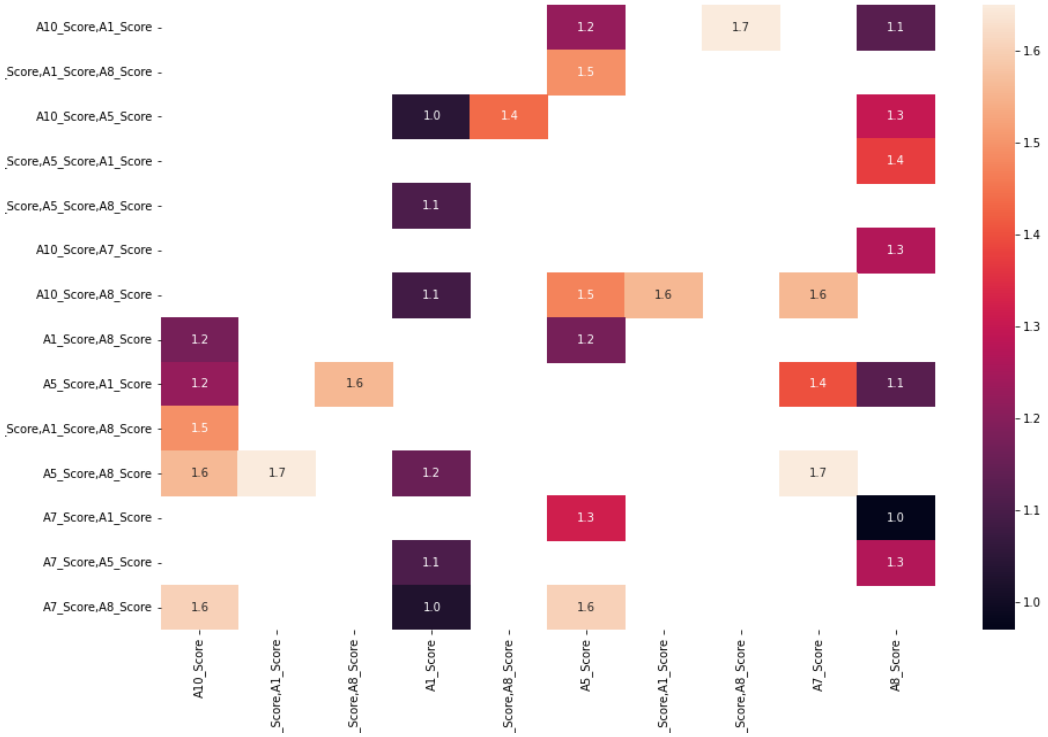
Set the confidence level for variable associations to a 50% minimum threshold.

```
# Check for relations between _Scores using association rules
rules = association_rules(questionsets, metric='confidence', min_threshold=0.5) # Use confidence as the metric with 50% minimum threshold.
rules.sort_values(by=['lift'], ascending=False).head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
175	(A2_Score, A5_Score)	(A4_Score, A3_Score)	0.176471	0.215686	0.156863	0.888889	4.121212	0.118800	7.058824
178	(A4_Score, A3_Score)	(A2_Score, A5_Score)	0.215686	0.176471	0.156863	0.727273	4.121212	0.118800	3.019608
177	(A4_Score, A2_Score)	(A5_Score, A3_Score)	0.196078	0.235294	0.156863	0.800000	3.400000	0.110727	3.823529
176	(A5_Score, A3_Score)	(A4_Score, A2_Score)	0.235294	0.196078	0.156863	0.666667	3.400000	0.110727	2.411765
171	(A4_Score, A5_Score, A3_Score)	(A2_Score)	0.176471	0.313725	0.156863	0.888889	2.833333	0.101499	6.176471

The strongest associations for Asian females are:

- A10_Score & A1_Score → A8_Score & A5_Score
- A5_Score & A8_Score → A7 Score



7.5.3 Associations- Middle Eastern Female Responses:

```
# Create dataset with _Score variables and ID. Set ID as the index
eastern_scores = eastern_df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'ID']]
eastern_scores.set_index('ID', inplace=True)
eastern_scores.head()
```

ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score
11	0	1	0	1	1	1	1	0	0	1
18	0	0	1	0	1	1	0	0	0	0
42	0	0	1	1	0	0	0	0	0	1
82	0	0	1	0	1	0	0	1	0	1
84	1	0	0	0	0	0	0	1	0	1

Set the minimum frequency for itemset associations to 12%.

```
# Check for frequent itemsets using apriori.
questionsets = apriori(eastern_scores, min_support=.12, use_colnames=True) # Set min frequency to 12%.
questionsets.head()
```

	support	itemsets
0	0.605263	(A1_Score)
1	0.447368	(A2_Score)
2	0.368421	(A3_Score)
3	0.342105	(A4_Score)
4	0.315789	(A5_Score)

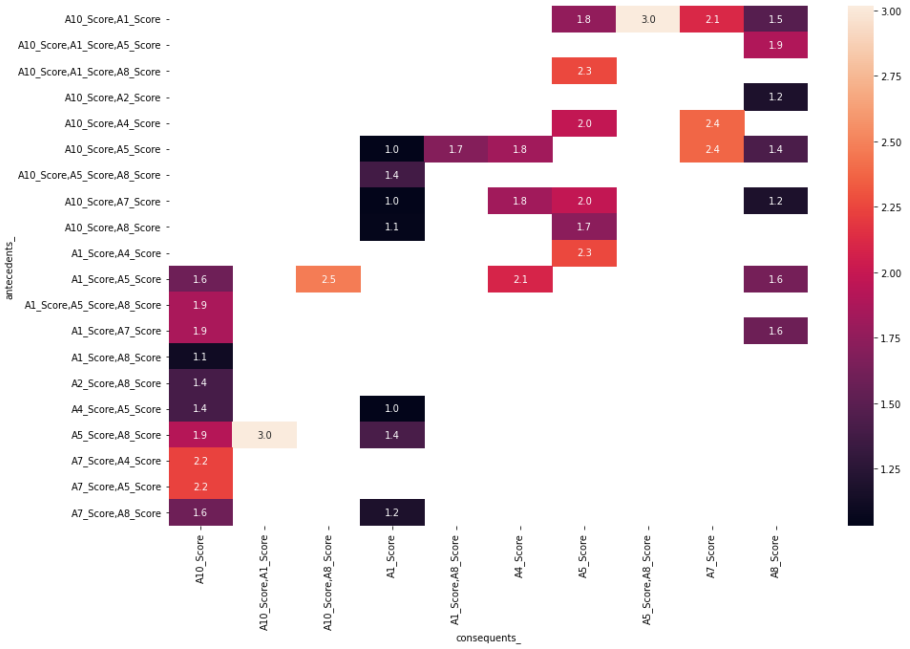
Set the minimum confidence level to 50% and check for most important associations.

```
# Check for relations between _Score variables using association rules.
rules = association_rules(questionsets, metric='confidence', min_threshold=0.5) # Use confidence as metric and set threshold to 50%
rules.sort_values(by=['lift'], ascending=False).head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
76	(A5_Score, A8_Score)	(A10_Score, A1_Score)	0.184211	0.236842	0.131579	0.714286	3.015873	0.087950	2.671053
73	(A10_Score, A1_Score)	(A5_Score, A8_Score)	0.236842	0.184211	0.131579	0.555556	3.015873	0.087950	1.835526
75	(A1_Score, A5_Score)	(A10_Score, A8_Score)	0.184211	0.289474	0.131579	0.714286	2.467532	0.078255	2.486842
61	(A7_Score)	(A10_Score, A5_Score)	0.263158	0.210526	0.131579	0.500000	2.375000	0.076177	1.578947
57	(A7_Score)	(A10_Score, A4_Score)	0.263158	0.210526	0.131579	0.500000	2.375000	0.076177	1.578947

The most frequent associations for Middle Eastern females are:

- A10_Score & A1_Score ←→ A5_Score & A8_Score
- A1_Score & A5_Score → A10_Score & A8_Score



7.5.4 Associations- Others Responses:

```
# Create a dataset using _Score variables and ID. Set ID as index.
others_scores = others_df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'ID']]
others_scores.set_index('ID', inplace=True)
others_scores.head()
```

ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score
4	1	0	0	0	0	0	0	1	0	0
12	0	1	1	1	1	1	0	0	1	0
14	1	0	0	0	0	0	1	1	0	1
25	0	1	1	0	0	0	0	1	0	0
32	1	1	0	0	0	0	1	0	0	1

Set the minimum frequency for itemsets to 15%.

```
# Check for frequent itemsets using apriori
questionsets = apriori(others_scores, min_support=.15, use_colnames=True) # Set minimum support to 15%.
questionsets.head()
```

	support	itemsets
0	0.619048	(A1_Score)
1	0.349206	(A2_Score)
2	0.380952	(A3_Score)
3	0.412698	(A4_Score)
4	0.380952	(A5_Score)

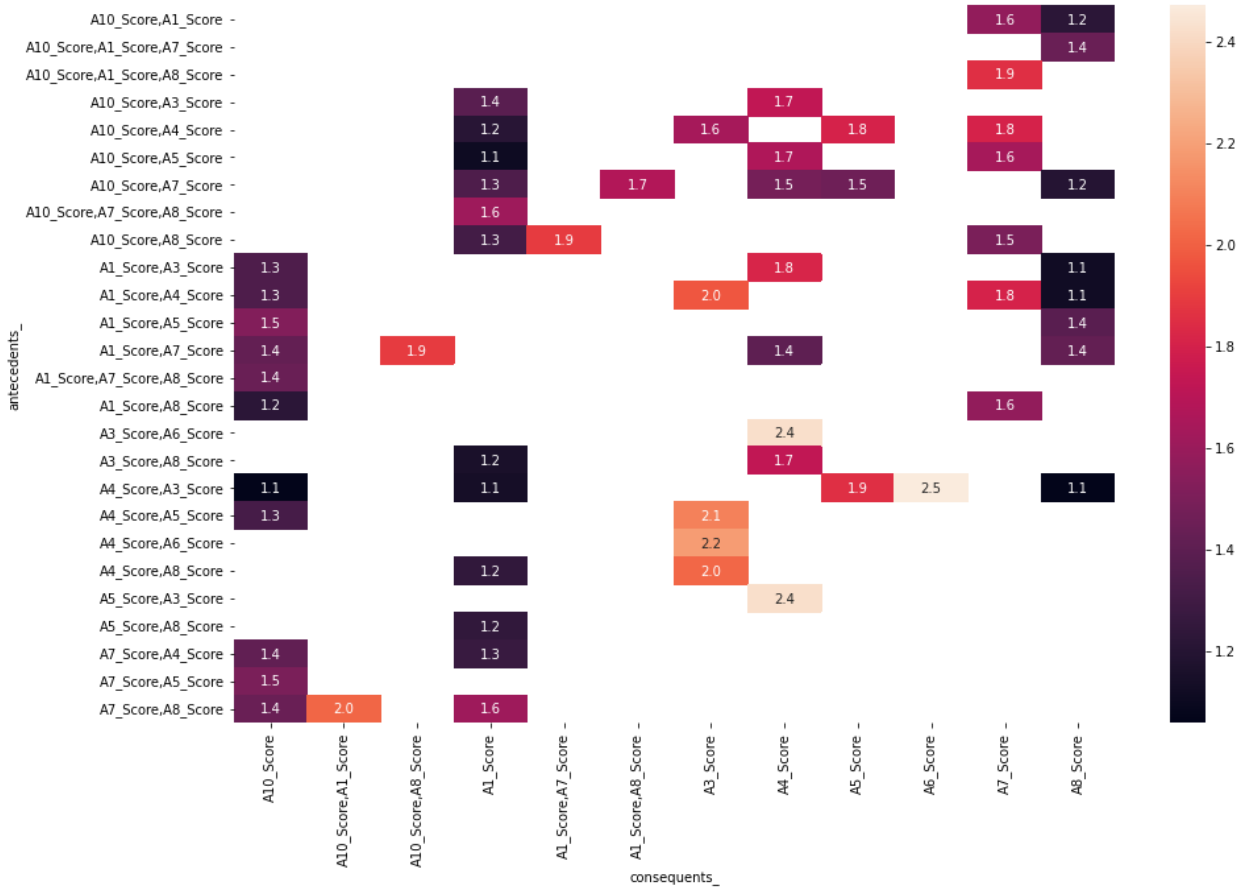
Set minimum confidence threshold to 50%.

```
# Check for relations between _Scores using association rules
rules = association_rules(questionsets, metric='confidence', min_threshold=0.5) # Use confidence as the metric with 50% minimum threshold.
rules.sort_values(by=['lift'], ascending=False).head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
79	(A6_Score)	(A4_Score, A3_Score)	0.238095	0.269841	0.158730	0.666667	2.470588	0.094482	2.190476
76	(A4_Score, A3_Score)	(A6_Score)	0.269841	0.238095	0.158730	0.588235	2.470588	0.094482	1.850340
78	(A3_Score, A6_Score)	(A4_Score)	0.158730	0.412698	0.158730	1.000000	2.423077	0.093222	inf
73	(A5_Score, A3_Score)	(A4_Score)	0.190476	0.412698	0.190476	1.000000	2.423077	0.111867	inf
77	(A4_Score, A6_Score)	(A3_Score)	0.190476	0.380952	0.158730	0.833333	2.187500	0.086168	3.714286

The most frequent itemset associations for Others females are:

- A4_Score & A3_Score → A6_Score
- A3_Score & A6_Score → A4_Score
- A5_Score & A3_Score → A4_Score



7.6 Model Comparison for Female dataset and Female Ethnic Subgroups:

Datasets	Maximal Trees	Optimized Trees	Random Forest	Logistic Regression	Neural Network
Full Female	91.8%	85.2%	94.0%	93.3%	100.0%
White Female	88.0%	84.0%	92.0%	92.0%	92.0%
Black Female	70.0%	70.0%	80.0%	80.0%	80.0%
Asian Female	81.0%	81.0%	95.0%	81.0%	85.7%
Middle Eastern Female	81.3%		100.0%	93.8%	93.8%
Others Female	92.3%	88.0%	96.0%	96.2%	96.2%

The best model for the entire female dataset is the neural network. When we looked at the output of this model we can see that it is better at predicting a positive ASD diagnosis than a negative ASD diagnosis. Additionally, the accuracy for this model is higher than the accuracy of the full dataset where we analyzed the _Score variables without distinguishing gender. This indicated that there are gender differences that make the models more accurate when analyzed separately.

The best model for the ethnic subgroups is the Random Forest model. Looking at the feature importances for each of the variables in our barplots we saw that there are differences in importance ranking for each of the ethnicities. This supports our theory that there are ethnic differences in autistic behavior, indicated by responses to the AQ-10.

8.0 Modelling Gender Differences (Males):

Create a new dataset for male only records.

```
# Create new dataset from autism_df for male-only records
male_df = autism_df[(autism_df['gender']=='m')]
male_df.shape
```

```
(367, 21)
```

Set the _Score variables as the predictors and Class/ASD as the target variable. Convert Class/ASD to numeric using get_dummies and partition the dataset into training and validation sets.

```
# Split the dataset with Class/ASD as the target variable.
predictors = ['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score']
outcome = 'Class/ASD'

x= male_df[predictors]
y= male_df[outcome]
# Transform categorical varialbes to dummies
X = x
Y = pd.get_dummies(y, drop_first=True)

train_X, valid_X, train_Y, valid_Y = train_test_split(X, Y, test_size=0.4, random_state=1)
```

8.0.1 Decision Trees:

Create a maximal tree with no hyperparameter tuning.

```
# Create a maximal tree
fullClassTree = DecisionTreeClassifier(random_state=1)
fullClassTree.fit(train_X, train_Y)
```

```
DecisionTreeClassifier(random_state=1)
```

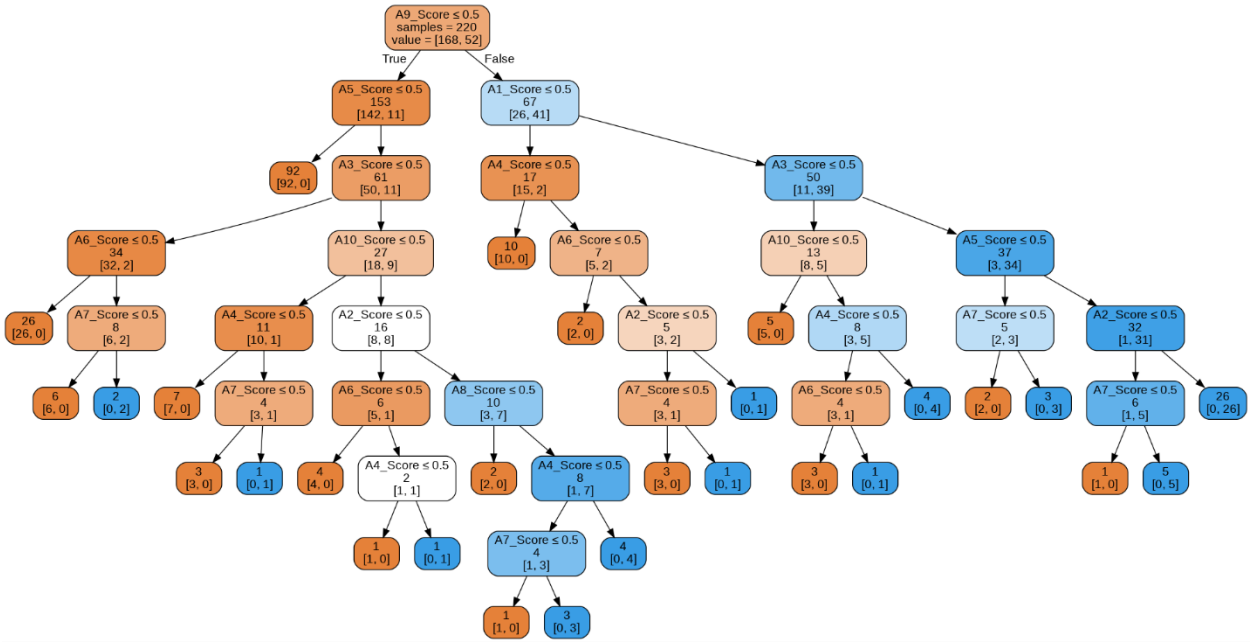
The accuracy for this model is 91.8%.

```
# Check model accuracy
classificationSummary(train_Y, fullClassTree.predict(train_X))
classificationSummary(valid_Y, fullClassTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual   0   1
     0 168   0
     1   0  52
Confusion Matrix (Accuracy 0.9184)

       Prediction
Actual   0   1
     0 107   6
     1   6  28
```

The first split for this tree is on the A9_Score, which splits into the A5_Score and A1_Score.

```
# Plot the tree
plotDecisionTree(fullClassTree, feature_names=train_X.columns)
```



Set optional parameters to optimize the decision tree.

```
# Create parameters to optimize the tree.
param_grid = {
    'max_depth':[3, 5, 6, 7, 9],
    'min_samples_split': [0.07, 0.05, 0.01, 0.005, 0.001],
    'min_impurity_decrease': [0.02, 0.01, 0.001, 0.005, 0.0001]
}
```

Use a gridsearch to run through all possible combinations of the parameters.

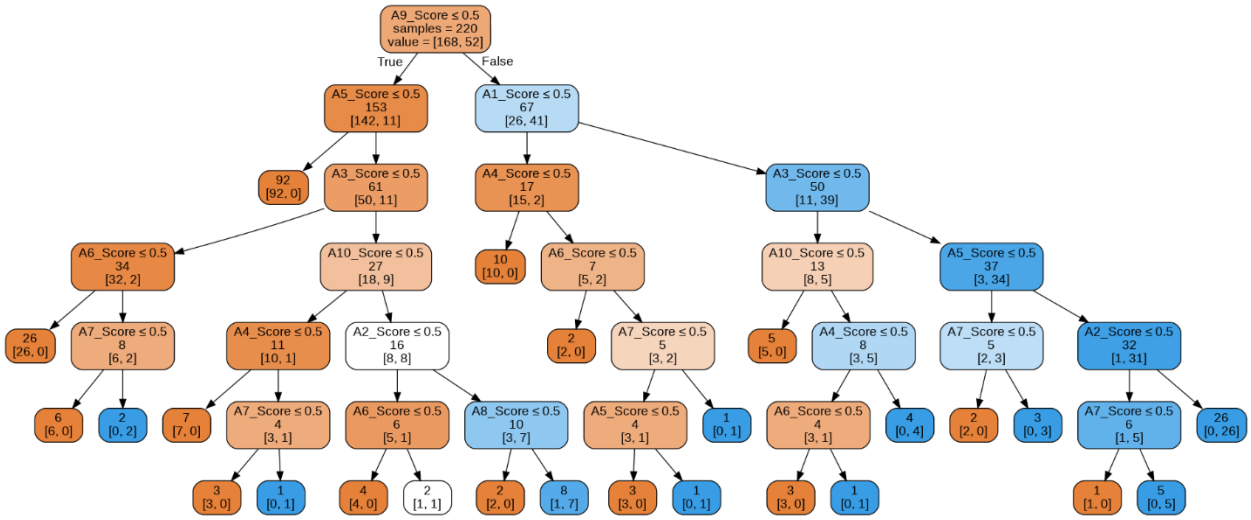
```
# Run a grid search to find the best parameters.
gridsearch = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid, cv=5, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

```
# Check for best parameters
gridsearch.best_params_
```

```
{'max_depth': 6, 'min_impurity_decrease': 0.001, 'min_samples_split': 0.01}
```

Create the optimized tree and run the model.

```
# Run the optimized tree and plot.
gridClassTree = DecisionTreeClassifier(max_depth=6,
                                       min_impurity_decrease=0.001,
                                       min_samples_split=0.01,
                                       random_state=1)
gridClassTree.fit(train_X, train_Y)
plotDecisionTree(gridClassTree, feature_names = train_X.columns)
```



The accuracy for this model is the same as the accuracy for the maximal tree.

```
# Check model accuracy
classificationSummary(train_Y, gridClassTree.predict(train_X))
classificationSummary(valid_Y, gridClassTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9909)

       Prediction
Actual   0   1
     0 167   1
     1   1  51
Confusion Matrix (Accuracy 0.9184)

       Prediction
Actual   0   1
     0 107   6
     1   6  28
```

8.0.2 Random Forest:

Create parameters to optimize the random forest.

```
# Set random forest parameters
param_grid = {
    'max_depth':[6, 7, 8],
    'min_samples_split': [0.05, 0.01, 0.001],
    'min_impurity_decrease': [0.01, 0.001, 0.005],
    'n_estimators':[300, 400, 500]
}
```

Run a gridsearch on all combinations of the parameters to find the best one.

```
# Perform a grid search for the best parameters
gridsearch = GridSearchCV(RandomForestClassifier(random_state=1), param_grid, cv=3, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

```
# Check for the best parameters
gridsearch.best_params_
```

```
{'max_depth': 7,
 'min_impurity_decrease': 0.001,
 'min_samples_split': 0.01,
 'n_estimators': 300}
```

Optimize the random forest and run the model.

```
# Run the optimized random forest
randomForest = RandomForestClassifier(random_state=1, n_estimators=300,
                                      max_depth=7, min_impurity_decrease=0.001,
                                      min_samples_split=0.01)
randomForest.fit(train_X, train_Y)
```

Create variables for the standard deviation and feature importance.

```
# Create variables for feature importance and standard deviation
importance = randomForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in randomForest.estimators_], axis=0)
```

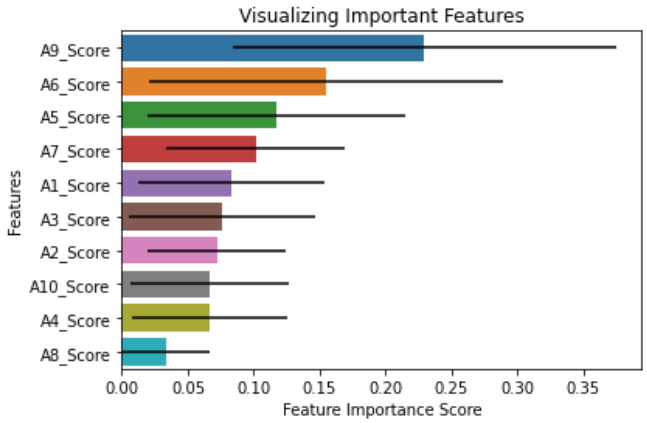
Check the standard deviation and feature importance of each variable.

```
# Check feature importance and standard deviation
randomForest_df = pd.DataFrame({'feature': train_X.columns,
                   'importance': importance,
                   'std': std})
print(randomForest_df.sort_values('importance', ascending=False))
```

```
     feature  importance       std
8   A9_Score    0.229473  0.145736
5   A6_Score    0.154952  0.134307
4   A5_Score    0.117172  0.097255
6   A7_Score    0.101293  0.067790
0   A1_Score    0.082952  0.070572
2   A3_Score    0.075834  0.070981
1   A2_Score    0.071954  0.052670
9  A10_Score    0.066632  0.059966
3   A4_Score    0.066248  0.058795
7   A8_Score    0.033489  0.032619
```

Use a barplot to check the feature importance of each variable ranked in descending order. Like the female dataset, A9_Score, A6_Score, and A5_Score have the highest feature importance. However, A7_Score which had the lowest feature importance for females has the fourth ranking of importance for males. A10_Score and A4_Score have a low ranking for males, but had a high ranking for females.

```
# Plot feature importance
value_plot = randomForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```



The accuracy for this model is higher than the accuracy for the decision trees.

```
# Check model accuracy
classificationSummary(train_Y, randomForest.predict(train_X))
classificationSummary(valid_Y, randomForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9955)

       Prediction
Actual   0   1
     0 168   0
     1   1  51
Confusion Matrix (Accuracy 0.9796)

       Prediction
Actual   0   1
     0 112   1
     1   2  32
```

8.0.3 Logistic Regression:

Run the model with the default inverse strength regularization of 1.

```
# Run a logistic regression with C=1
log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
log_reg.fit(train_X, train_Y)
```

The accuracy for the logistic regression is higher than the accuracy for the random forest and decision trees.

```
# Check model accuracy
classificationSummary(train_Y, log_reg.predict(train_X))
classificationSummary(valid_Y, log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9591)

       Prediction
Actual   0   1
     0 167   1
     1   8  44
Confusion Matrix (Accuracy 0.9864)

       Prediction
Actual   0   1
     0 113   0
     1   2  32
```

Check the coefficients and odds ratio. A9_Score, A6_Score, and A7_Score have the strongest impact on the target variable. A positive response to these questions makes it 2.8-4x more likely that you will be classified with ASD.

```
# Check variable coefficients and odds ratio
log_result = pd.DataFrame({'coef': log_reg.coef_[0], 'odds': np.e**log_reg.coef_[0]}, index=X.columns)
print(log_result.sort_values('coef', ascending=False))
```

```
               coef      odds
A9_Score   1.424485  4.155717
A6_Score   1.125707  3.082396
A7_Score   1.055870  2.874475
A5_Score   0.909666  2.483493
A2_Score   0.804351  2.235245
A3_Score   0.697851  2.009429
A4_Score   0.649822  1.915200
A1_Score   0.639447  1.895432
A10_Score  0.477392  1.611865
A8_Score   0.309754  1.363090
```

8.0.4 Neural Network:

Create optional hidden layer sizes to optimize the neural network.

```
# Create optional hidden layer sizes for neural network
clf_param = {'hidden_layer_sizes': [1, 2, 3, 4, 5, 6, 7, 8]}
```

Run a gridsearch on each hidden layer size to find the best parameter.

```
# Run a gridsearch to find the best hidden layer size
gridsearch= GridSearchCV(MLPClassifier(activation='logistic', solver='lbfgs', random_state=1, max_iter=500),
                         param_grid=clf_param, cv=5, n_jobs=-1, return_train_score=True)
gridsearch.fit(train_X, train_Y)
```

The best number of hidden layers for this model is 1 hidden layer.

```
# Check for best params
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 1}
```

Run the optimized neural network.

```
# Run a neural network with 1 hidden layer
clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=1, random_state=1)
clf.fit(train_X, train_Y)
```

The accuracy for this model is 100%.

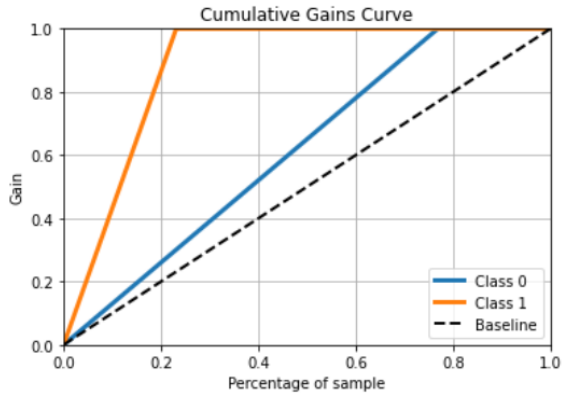
```
# Check model accuracy
classificationSummary(valid_Y, clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual   0   1
     0 113   0
     1   0  34
```

Create a cumulative gains chart to check the model predictions of both classifications. For ‘YES’ ASD classifications the model optimizes at 22% and for ‘NO’ ASD classifications the model optimizes at 76%.

```
# Create a variable for model prediction probabilities
pred = clf.predict_proba(valid_X)

# Plot cumulative gains chart
import scikitplot as skplt
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, pred)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



8.0.5 Association Rules:

Create a new dataset to check associations between _Score variables. Set the ID column as the index.

```
# Create a new dataset using _Score variables and ID
scores_df = male_df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'ID']]
# Set ID as the index
scores_df.set_index('ID', inplace=True)
scores_df.head()
```

ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score
1	1	1	0	1	0	0	0	1	0	1
2	1	1	0	1	1	0	1	1	1	1
5	1	1	1	1	1	0	1	1	1	1
7	1	1	1	1	0	0	0	0	1	0
8	1	1	0	0	1	0	0	1	1	1

Set the minimum frequency of itemset associations to 20%.

```
# Check for frequent itemsets using apriori. Minimum frequency is 20%
questionsets = apriori(scores_df, min_support=.2, use_colnames=True)
questionsets.head()
```

	support	itemsets
0	0.689373	(A1_Score)
1	0.430518	(A2_Score)
2	0.457766	(A3_Score)
3	0.468665	(A4_Score)
4	0.479564	(A5_Score)

Set the minimum confidence threshold to 50% and check for the most important associations.

```
# Use association rules to check for relations between variables
rules = association_rules(questionsets, metric='confidence', min_threshold=0.5) # Use minimum 50% confidence as metric
rules.sort_values(by=['lift'], ascending=False).head()
```

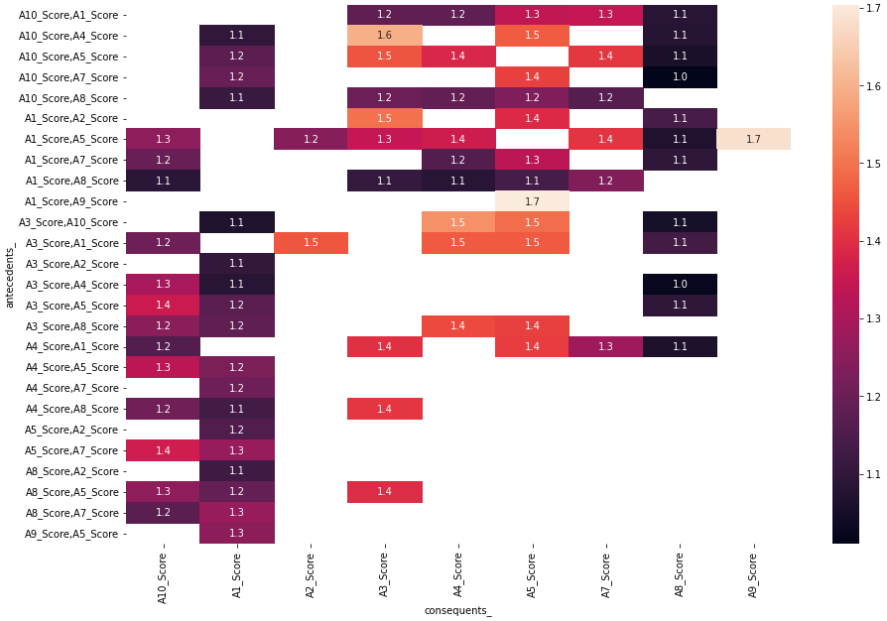
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
100	(A1_Score, A9_Score)	(A5_Score)	0.253406	0.479564	0.207084	0.817204	1.704057	0.085560	2.847091
101	(A1_Score, A5_Score)	(A9_Score)	0.376022	0.326975	0.207084	0.550725	1.684300	0.084135	1.498022
103	(A9_Score)	(A1_Score, A5_Score)	0.326975	0.376022	0.207084	0.633333	1.684300	0.084135	1.701759
125	(A10_Score, A4_Score)	(A3_Score)	0.294278	0.457766	0.215259	0.731481	1.597939	0.080549	2.019355
123	(A3_Score, A10_Score)	(A4_Score)	0.297003	0.468665	0.215259	0.724771	1.546458	0.076064	1.930518

Create a heatmap to visualize the associations between variables.

```
# Create a heatmap to plot association between _Scores
rules['lhs items'] = rules['antecedents'].apply(lambda x:len(x) )
rules[rules['lhs items']>1].sort_values('lift', ascending=False).head()

# Replace frozen sets with strings
rules['antecedents_'] = rules['antecedents'].apply(lambda a: ','.join(list(a)))
rules['consequents_'] = rules['consequents'].apply(lambda a: ','.join(list(a)))

# Transform the DataFrame of rules into a matrix using the lift metric
pivot = rules[rules['lhs items']>1].pivot(index = 'antecedents_',
                    columns = 'consequents_', values= 'lift')


# Replace frozen sets with strings
rules['antecedents_'] = rules['antecedents'].apply(lambda a: ','.join(list(a)))
rules['consequents_'] = rules['consequents'].apply(lambda a: ','.join(list(a)))

# Transform the DataFrame of rules into a matrix using the lift metric
pivot = rules[rules['lhs items']>1].pivot(index = 'antecedents_',
                    columns = 'consequents_', values= 'lift')

# Generate a heatmap with annotations on and the colorbar off
plt.figure(figsize = (15, 10))
sns.heatmap(pivot, annot = True, fmt='.1f')
plt.yticks(rotation=0)
plt.xticks(rotation=90)
plt.show()
```

The most important associations for Males are:

- A1_Score & A5_Score → A9_Score
- A1_Score & A9_Score → A5_Score
- A10_Score & A4_Score → A3_Score



9.0 Modeling Ethnic Groups (Males):

Like I did for the female dataset I will be examining 5 ethnic groups: White males, Black males, Asian males, Middle Eastern males, and an unidentified Others group.

```
# Create a new dataset for White males.
white_df = male_df[(male_df['ethnicity']=='White')]
```

```
#Create dataset for Black females
black_df = male_df[(male_df['ethnicity']=='Black')]
```

```
# Create dataset for Asian ethnicity
asian_df = male_df[(male_df['ethnicity']=='Asian')]
```

```
# Create dataset for Middle Eastern ethnicity
eastern_df = male_df[(male_df['ethnicity']=='Middle Eastern')]
```

```
# Create a dataset for others
others_df = male_df[(male_df['ethnicity']=='Others')]
```

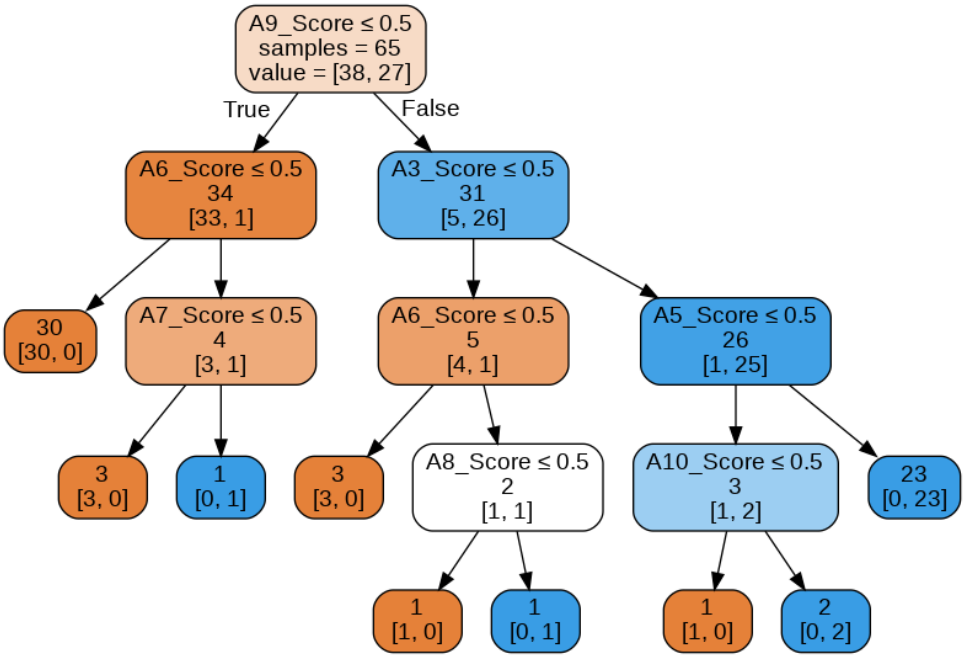
Set the _Score variables as the predictors and Class/ASD as the target variable. Convert Class/ASD to numeric form using get_dummies and split the dataset into training and validation sets. This image shows an example from the Others subgroup. For the rest of the subgroups the data was split on their individual datasets.

```
# Split the dataset with Class/ASD as the target variable, using only _Scores as perdictors.
predictors = ['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score']
outcome = 'Class/ASD'

x= others_df[predictors]
y= others_df[outcome]
# Convert categorical varialbes to dummies
X = x
Y = pd.get_dummies(y, drop_first=True)

train_X, valid_X, train_Y, valid_Y = train_test_split(X, Y, test_size=0.4, random_state=1)
```

9.1 Decision Trees:

9.1.0 Maximal Tree- White Males:

The first split in this decision tree is on A9_Score which splits into A6_Score and A3_Score.



The accuracy for this model is lower than the accuracy for the maximal tree in the full male dataset.

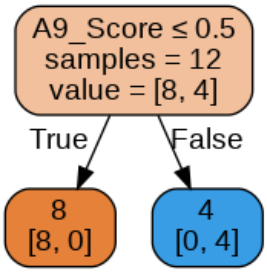
```
# Check model accuracy
classificationSummary(valid_Y, whiteFullTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8636)

       Prediction
Actual  0  1
     0 24  2
     1  4 14
```

9.1.2 Maximal Tree- Black Males:

This is a small maximal tree, most likely because the number of records for Black males is small.

This is significantly higher than the accuracy for Black females (70%).



Accuracy for this model is extremely high, again most likely because of how small the dataset is.

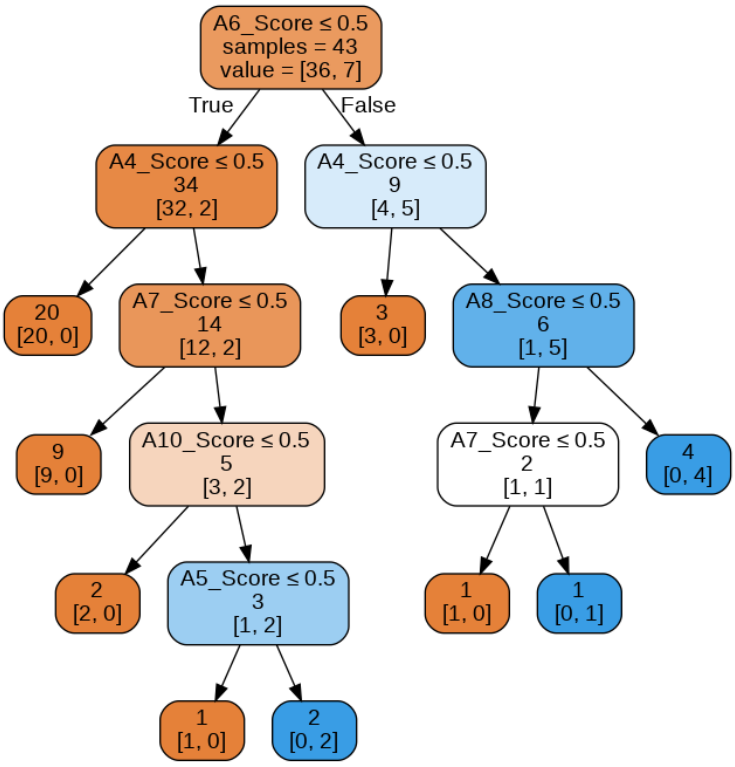
```
# Check accuracy
classificationSummary(valid_Y, blackFullTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual 0 1
     0 5 0
     1 0 3
```

9.1.3 Maximal Tree- Asian Males:

This model splits from A9_Score into A4_Score, which did not appear in the previous models

for White and Black males.



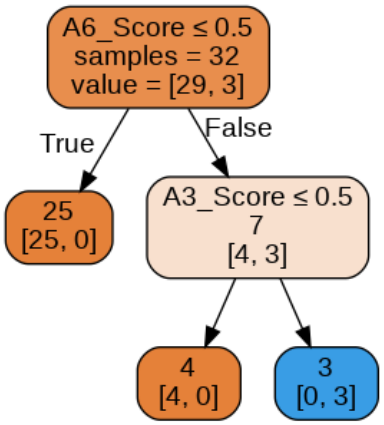
Model accuracy is higher than the accuracy for White Males.

```
# Check accuracy
classificationSummary(valid_Y, asianFullTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9310)

       Prediction
Actual  0  1
     0 25  1
     1  1  2
```

9.1.4 Maximal Tree- Middle Eastern Males:

Like Asian males, the primary split for this model is on A6_Score which splits into A3_Score.



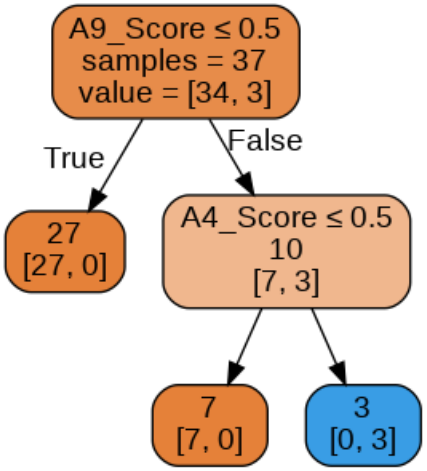
This is the second highest model accuracy out of all the ethnic subgroups.

```
# Check for accuracy
classificationSummary(valid_Y, easternFullTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9545)

       Prediction
Actual  0  1
     0 21  0
     1  1  0
```

9.1.5 Maximal Tree- Others:

This model splits on A9_Score into A4_Score.



This is the second lowest model accuracy out of all the subgroups.

```
# Check model accuracy
classificationSummary(valid_Y, othersFullTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9231)

       Prediction
Actual  0  1
     0 22  0
     1  2  2
```

9.1.6 Optimized Tree- White Males:

I am going to try and optimize the decision tree for White males as it was significantly lower than the accuracy for other ethnicities. I created optional parameters to run through a grid search.

```
# Create parameters to optimize the tree
param_grid = {
    'max_depth':[3, 5, 7, 8, 9],
    'min_samples_split': [0.05, 0.01, 0.005, 0.001, 0.0001],
    'min_impurity_decrease': [0.05, 0.02, 0.01, 0.001, 0.005]
}
```

Use a gridsearch to run through all the combinations of the parameters.

```
# Run a grid search to find the best parameters.
gridsearch = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid, cv=5, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

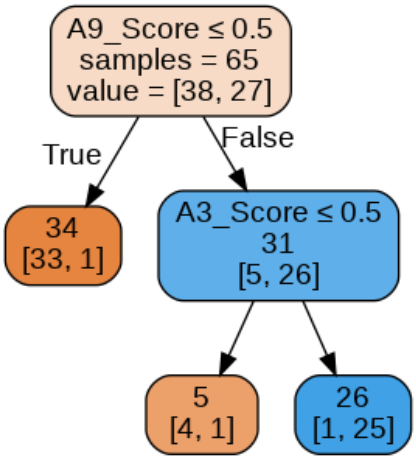
Find the best set of parameters.

```
# Check for best parameters.
gridsearch.best_params_
```

```
{'max_depth': 3, 'min_impurity_decrease': 0.05, 'min_samples_split': 0.05}
```

Optimize the decision tree and run the model.

```
# Run the optimized tree and plot.
whiteClassTree = DecisionTreeClassifier(max_depth=3,
                                        min_impurity_decrease=0.05,
                                        min_samples_split=0.05,
                                        random_state=1)
whiteClassTree.fit(train_X, train_Y)
plotDecisionTree(whiteClassTree, feature_names = train_X.columns)
```



Accuracy for the model actually gets worse when optimized using a gridsearch.

```
# Check accuracy
classificationSummary(valid_Y, whiteClassTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8409)

       Prediction
Actual  0  1
     0 24  2
     1  5 13
```

9.2 Random Forests:

9.2.0 Random Forest- White Males:

Create optional parameters to run through the gridsearch.

```
# Set random forest parameters
param_grid = {
    'max_depth':[6, 7, 8],
    'min_samples_split': [0.05, 0.01, 0.001],
    'min_impurity_decrease': [0.01, 0.001, 0.005],
    'n_estimators':[100, 200, 300]
}
```

Use the gridsearch to run through all combinations of the parameters.

```
# Perform a grid search for the best parameters
gridsearch = GridSearchCV(RandomForestClassifier(random_state=1), param_grid, cv=3, n_jobs=-1)
gridsearch.fit(train_X, train_Y)
```

```
# Check for best parameters
gridsearch.best_params_
```

```
{'max_depth': 6,
 'min_impurity_decrease': 0.01,
 'min_samples_split': 0.05,
 'n_estimators': 300}
```

Optimize the random forest and run the model.

```
# Run a random forest
whiteForest = RandomForestClassifier(random_state=1, n_estimators=300,
                                     max_depth=6, min_impurity_decrease=0.01,
                                     min_samples_split=0.05)
whiteForest.fit(train_X, train_Y)
```

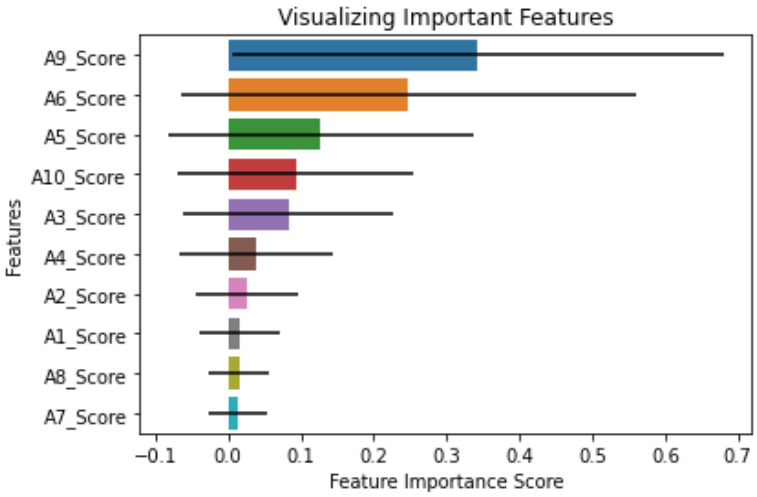
Check the feature importance and standard deviation for each variable.

```
# Check variable importance and standard deviation
importance = whiteForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in whiteForest.estimators_], axis=0)

whiteForest_df = pd.DataFrame({'feature': train_X.columns,
                               'importance': importance,
                               'std': std})
print(whiteForest_df.sort_values('importance', ascending=False))
```

```
     feature  importance       std
8   A9_Score    0.342878  0.337865
5   A6_Score    0.247640  0.313141
4   A5_Score    0.126992  0.209975
9  A10_Score    0.092578  0.161183
2   A3_Score    0.082540  0.145127
3   A4_Score    0.037802  0.105161
1   A2_Score    0.026167  0.069943
0   A1_Score    0.015607  0.055966
7   A8_Score    0.014667  0.041931
6   A7_Score    0.013129  0.040283
```

Create a barplot to visualize feature importance in descending order. A9_Score, A6_Score, and A5_Score have the highest feature importance. Unlike the full male dataset where A7_Score was the fourth most important feature, it is the least important for White males.

```
# Plot feature importance
value_plot = whiteForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```



The accuracy for this model is higher than the accuracy for the decision trees.

```
# Check accuracy
classificationSummary(valid_Y, whiteForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8846)

       Prediction
Actual  0  1
     0 21  1
     1  2  2
```

9.2.1 Random Forest- Black Males:

The best parameters for Black males are different than those for White males.

```
# Check for best parameters
gridsearch.best_params_
```

```
{'max_depth': 3,
 'min_impurity_decrease': 0.05,
 'min_samples_split': 0.07,
 'n_estimators': 200}
```

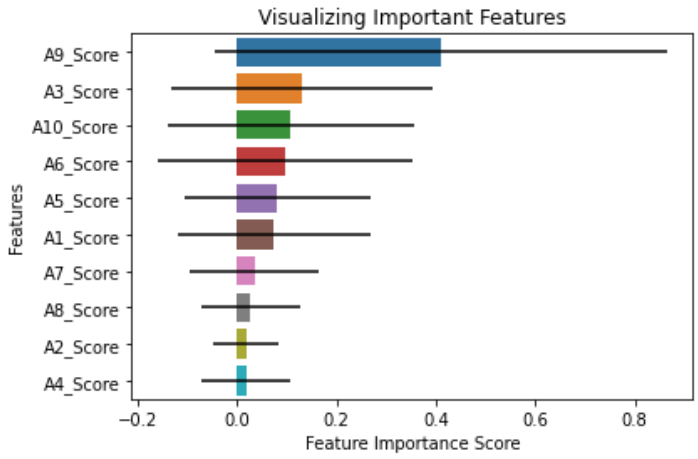
```
# Run a random forest
blackForest = RandomForestClassifier(random_state=1, n_estimators=200,
                                     max_depth=3, min_impurity_decrease=0.05,
                                     min_samples_split=0.07)
blackForest.fit(train_X, train_Y)
```

```
# Check importance and standard deviation for features
importance = blackForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in blackForest.estimators_], axis=0)

blackForest_df = pd.DataFrame({'feature': train_X.columns,
                               'importance': importance,
                               'std': std})
print(blackForest_df.sort_values('importance', ascending=False))
```

	feature	importance	std
8	A9_Score	0.410252	0.454856
2	A3_Score	0.130832	0.263082
9	A10_Score	0.108169	0.247743
5	A6_Score	0.097091	0.257298
4	A5_Score	0.081340	0.187244
0	A1_Score	0.073948	0.194287
6	A7_Score	0.034683	0.128296
7	A8_Score	0.027393	0.100314
1	A2_Score	0.018276	0.066062
3	A4_Score	0.018017	0.090016

Create a barplot to visualize feature importance in descending order. A6_Score and A5_Score have a lower ranking than the White males, while A3_Score and A10_Score have a much higher ranking.

```
# Plot feature importance
value_plot = blackForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```



The accuracy for this model is the same as the decision tree.

```
# Check accuracy
classificationSummary(valid_Y, blackForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual 0 1
     0 5 0
     1 0 3
```

9.2.2 Random Forest- Asian Males:

This model has the same parameters as the random forest for White males.

```
# Check for best parameters
gridsearch.best_params_
```

```
{'max_depth': 6,
 'min_impurity_decrease': 0.001,
 'min_samples_split': 0.05,
 'n_estimators': 300}
```

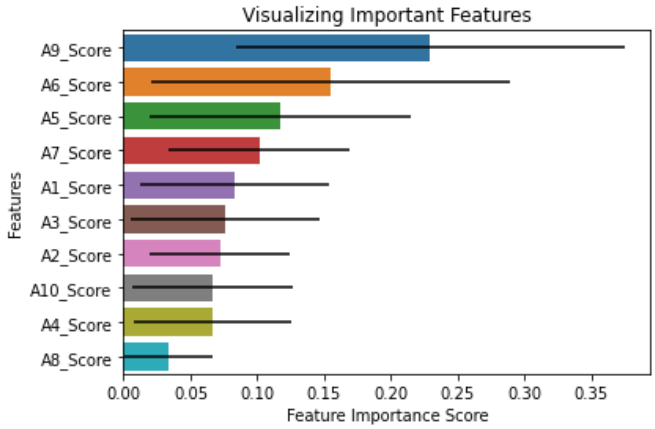
```
# Run a random forest
asianForest = RandomForestClassifier(random_state=1, n_estimators=300,
                                     max_depth=6, min_impurity_decrease=0.001,
                                     min_samples_split=0.05)
asianForest.fit(train_X, train_Y)
```

```
# Check feature importance and standard deviation
importance = randomForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in randomForest.estimators_], axis=0)

asianForest_df = pd.DataFrame({'feature': train_X.columns,
                               'importance': importance,
                               'std': std})
print(asianForest_df.sort_values('importance', ascending=False))
```

```
     feature  importance       std
8   A9_Score    0.229473  0.145736
5   A6_Score    0.154952  0.134307
4   A5_Score    0.117172  0.097255
6   A7_Score    0.101293  0.067790
0   A1_Score    0.082952  0.070572
2   A3_Score    0.075834  0.070981
1   A2_Score    0.071954  0.052670
9  A10_Score    0.066632  0.059966
3   A4_Score    0.066248  0.058795
7   A8_Score    0.033489  0.032619
```

Create a barplot to visualize feature importance in descending order. The feature importance ranking for this subgroup is an exact replica of the model created for the full male dataset.

```
# Plot feature importance
value_plot = asianForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```



The accuracy for this model is higher than the accuracy for the decision tree.

```
# Check accuracy
classificationSummary(valid_Y, asianForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9655)

       Prediction
Actual  0  1
     0 26  0
     1  1  2
```

9.2.3 Random Forest- Middle Eastern Males:

The best parameters for this subgroup differs from the other subgroups.

```
# Check best parameters
gridsearch.best_params_
```

```
{'max_depth': 3,
 'min_impurity_decrease': 0.01,
 'min_samples_split': 0.05,
 'n_estimators': 100}
```

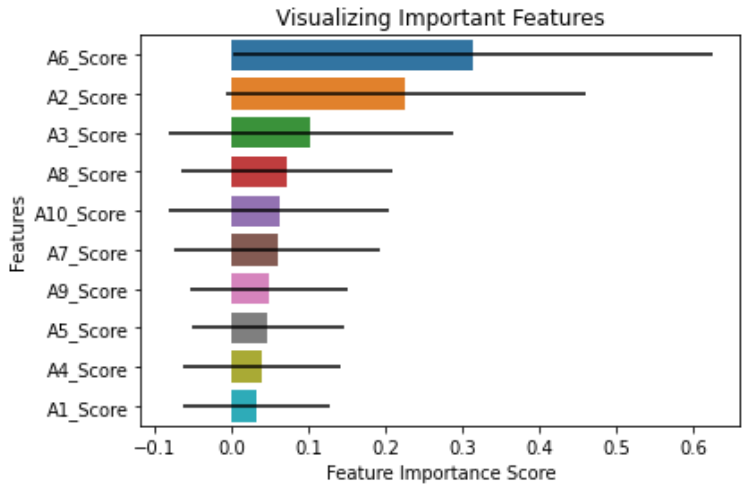
```
# Run a random forest
easternForest = RandomForestClassifier(random_state=1, n_estimators=100,
                                       max_depth=3, min_impurity_decrease=0.01,
                                       min_samples_split=0.05)
easternForest.fit(train_X, train_Y)
```

```
# Check variable importance and standard deviation
importance = easternForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in easternForest.estimators_], axis=0)

easternForest_df = pd.DataFrame({'feature': train_X.columns,
                                 'importance': importance,
                                 'std': std})
print(easternForest_df.sort_values('importance', ascending=False))
```

```
     feature  importance       std
5   A6_Score    0.313220  0.312323
1   A2_Score    0.226023  0.234442
2   A3_Score    0.102182  0.185085
7   A8_Score    0.071504  0.137689
9  A10_Score    0.061536  0.143769
6   A7_Score    0.059361  0.134062
8   A9_Score    0.048914  0.102591
4   A5_Score    0.046637  0.098629
3   A4_Score    0.039700  0.102714
0   A1_Score    0.030922  0.095400
```

Create a barplot to visualize feature importance in descending order. A6_Score has the highest feature importance while A9_Score has a much lower ranking. A2_Score, A3_Score, and A8_Score are also significant for this subgroup.

```
# Plot feature importance
value_plot = easternForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```



The accuracy for this model is the same as the decision tree accuracy.

```
# Check accuracy
classificationSummary(valid_Y, easternForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9545)

       Prediction
Actual  0  1
     0 21  0
     1  1  0
```

9.2.4 Random Forest- Others:

```
# Check for best paramters
gridsearch.best_params_
```

```
{'max_depth': 6,
 'min_impurity_decrease': 0.01,
 'min_samples_split': 0.05,
 'n_estimators': 100}
```

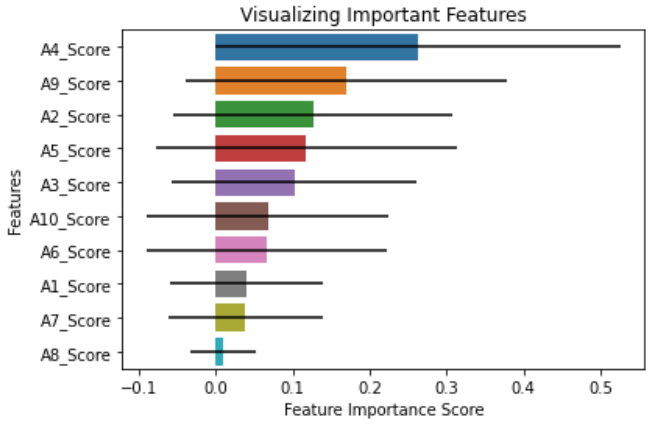
```
# Run a random forest
othersForest = RandomForestClassifier(random_state=1, n_estimators=100,
                                      max_depth=6, min_impurity_decrease=0.01,
                                      min_samples_split=0.05)
othersForest.fit(train_X, train_Y)
```

```
# Check feature importance and standard deviation
importance = othersForest.feature_importances_
std = np.std([tree.feature_importances_ for tree in othersForest.estimators_], axis=0)

othersForest_df = pd.DataFrame({'feature': train_X.columns,
                                'importance': importance,
                                'std': std})
print(othersForest_df.sort_values('importance', ascending=False))
```

```
     feature  importance       std
3   A4_Score    0.262752  0.263922
8   A9_Score    0.170225  0.208991
1   A2_Score    0.126066  0.182013
4   A5_Score    0.117362  0.196207
2   A3_Score    0.102436  0.159427
9  A10_Score    0.067173  0.157927
5   A6_Score    0.066175  0.156394
0   A1_Score    0.040072  0.099865
6   A7_Score    0.038114  0.100418
7   A8_Score    0.009626  0.042413
```

Create a barplot to visualize feature importance in descending order. A4_Score is the most important feature. A6_Score has a much lower feature importance for this subgroup than all the other subgroups.

```
# Plot feature importance
value_plot = othersForest_df.sort_values('importance', ascending = False)
sns.barplot(x = value_plot['importance'], y = value_plot['feature'], xerr = value_plot['std'])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```




```
# Check accuracy
classificationSummary(valid_Y, othersForest.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8462)

       Prediction
Actual  0  1
     0 22  0
     1  4  0
```

9.3 Neural Networks:

9.3.0 Neural Network- White Males:

Create optional hidden layer sizes to run through the gridsearch.

```
# Create optional hidden layer sizes for neural network
whiteCLF_param = {'hidden_layer_sizes': [1, 2, 3, 4, 5, 6, 7, 8]}
```

```
# Run a grid search to find optimal hidden layer size
gridsearch= GridSearchCV(MLPClassifier(activation='logistic', solver='lbfgs', random_state=1, max_iter=500),
                         param_grid=whiteCLF_param, cv=5, n_jobs=-1, return_train_score=True)
gridsearch.fit(train_X, train_Y)
```

The best hidden layer size is 4 hidden layers.

```
# Check for best parameters
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 4}
```

Optimize the neural network and run the model.

```
# Run a neural network with 4 hidden layers
white_clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=4, random_state=1)
white_clf.fit(train_X, train_Y)
```

The accuracy for this model is lower than the accuracy for the random forest.

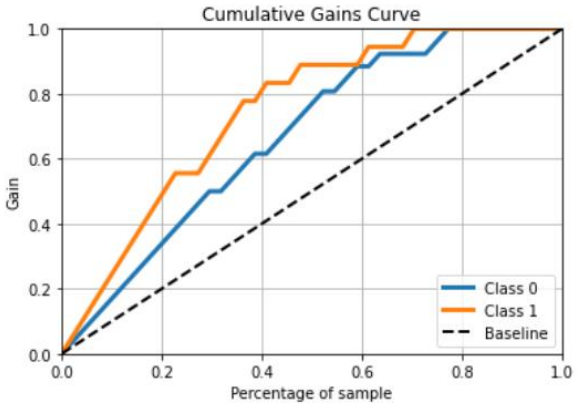
```
# Check model accuracy
classificationSummary(valid_Y, white_clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8636)

       Prediction
Actual  0  1
     0 23  3
     1  3 15
```

This cumulative gains chart shows that the model optimizes for ‘YES’ ASD classifications at 70% and optimizes at 75% for ‘NO’ ASD classifications.

```
white_clf_pred = white_clf.predict_proba(valid_X)

# Plot cumulative gains chart
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, white_clf_pred)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



9.3.1 Neural Network- Black Males:

The best hidden layer size for this neural network is 2 hidden layers.

```
# Find best layer size
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 2}
```

```
# Run a neural network with 2 hidden layers
black_clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=2, random_state=1)
black_clf.fit(train_X, train_Y)
```

The accuracy for this model is lower than the accuracy for the decision trees and random forest.

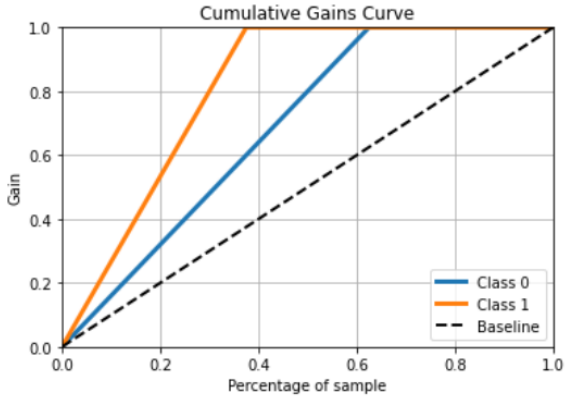
```
# Check model accuracy
classificationSummary(valid_Y, black_clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8750)

       Prediction
Actual 0 1
     0 4 1
     1 0 3
```

In this cumulative gains chart, predictions of 'YES' ASD classifications optimize at 36% and predictions of 'NO' ASD classifications optimize at 62%.

```
black_clf_pred = black_clf.predict_proba(valid_X)

# Plot cumulative gains chart
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, black_clf_pred)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



9.3.2 Neural Network- Asian Males:

The best hidden layer size is one hidden layer.

```
# Check optimal hidden layer size
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 1}
```

```
# Run a neural network with 1 hidden layer
asian_clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=1, random_state=1)
asian_clf.fit(train_X, train_Y)
```

Accuracy for this model is lower than the accuracy for the decision tree and random forest.

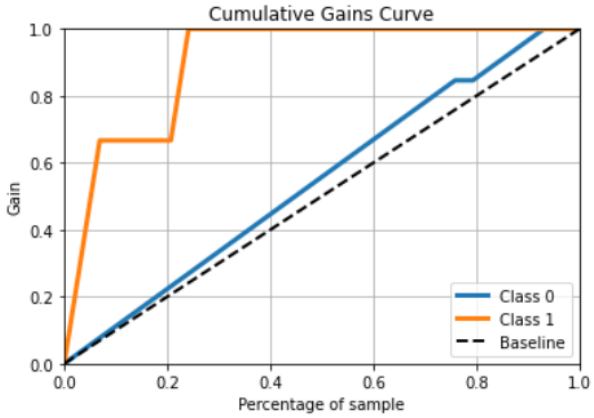
```
# Check model accuracy
classificationSummary(valid_Y, asian_clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8966)

       Prediction
Actual  0  1
     0 24  2
     1  1  2
```

In this cumulative gains chart, predictions of 'YES' ASD classifications optimize at 22%, while predictions of 'NO' ASD classifications optimize at 90%.

```
asian_clf_pred = asian_clf.predict_proba(valid_X)

# Plot the cumulative gains chart
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, asian_clf_pred)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



9.3.3 Neural Network- Middle Eastern Males:

The best hidden layer size for this model is 7 hidden layers.

```
# Find optimal hidden layer size
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 7}
```

```
# Run a neural network with 7 hidden layers
eastern_clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=7, random_state=1)
eastern_clf.fit(train_X, train_Y)
```

Accuracy for this model is the same as the other models.

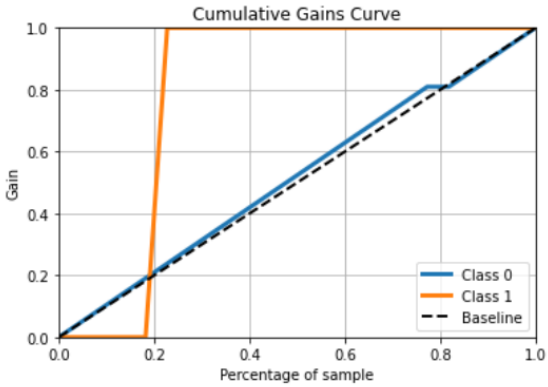
```
# Check model accuracy
classificationSummary(valid_Y, eastern_clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9545)

       Prediction
Actual  0  1
     0 21  0
     1  1  0
```

In this cumulative gains chart, predictions of 'YES' ASD classifications optimize at 22% and predictions of 'NO' ASD classifications optimize at 100%.

```
eastern_clf_pred = eastern_clf.predict_proba(valid_X)

# Plot cumulative gains chart
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, eastern_clf_pred)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



9.3.4 Neural Network- Others:

The best hidden layer size for this model is 2 hidden layers.

```
# Check optimal hidden layer size
gridsearch.best_params_
```

```
{'hidden_layer_sizes': 2}
```

```
# Run a neural network with 2 hidden layers
others_clf = MLPClassifier(activation = 'logistic', solver='lbfgs', hidden_layer_sizes=2, random_state=1)
others_clf.fit(train_X, train_Y)
```

Model accuracy is lower for neural networks.

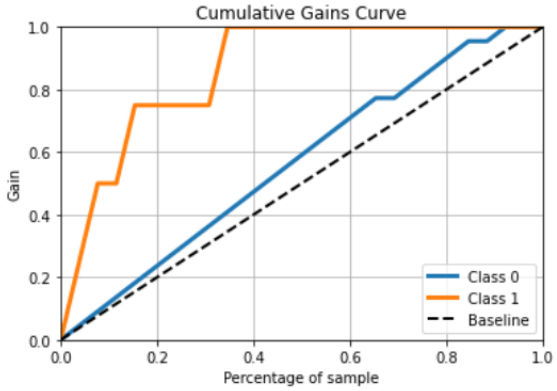
```
# Check model accuracy
classificationSummary(valid_Y, others_clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8846)

       Prediction
Actual  0  1
     0 21  1
     1  2  2
```

In this cumulative gains chart predictions of 'YES' ASD classifications optimize at 35% and predictions of 'NO' ASD classifications optimize at 90%.

```
others_clf_pred = others_clf.predict_proba(valid_X)

# Plot the cumulative gains chart
plt.figure(figsize=(10,10))
skplt.metrics.plot_cumulative_gain(valid_Y, others_clf_pred)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



9.4 Logistic Regressions:

9.4.0 Logistic Regression- White Males:

```
# Run a logistic regression
white_log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
white_log_reg.fit(train_X, train_Y)
```

Accuracy for this model is even lower than the neural network model.

```
# Check accuracy
classificationSummary(valid_Y, white_log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8409)

       Prediction
Actual  0  1
     0 22  4
     1  3 15
```

Looking at the odds ratio of this model we can see that A9_Score, A6_Score, and A10_Score have the strongest impact on the target outcome.

```
# Check variable coefficients and odds ratios
white_result = pd.DataFrame({'coef': white_log_reg.coef_[0], 'odds': np.e**white_log_reg.coef_[0]}, index=X.columns)
print(white_result.sort_values('odds', ascending=False))
```

```
               coef      odds
A9_Score   1.607627  4.990951
A6_Score   1.353457  3.870785
A10_Score  0.643937  1.903963
A5_Score   0.624552  1.867410
A3_Score   0.617115  1.853572
A7_Score   0.259470  1.296243
A4_Score   0.252409  1.287122
A2_Score   0.201085  1.222729
A8_Score   0.093979  1.098536
A1_Score  -0.568070  0.566618
```

9.4.1 Logistic Regression- Black Males:

```
# Run a logistic regression
black_log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
black_log_reg.fit(train_X, train_Y)
```

The odds ratio for this model shows that A9_Score, A6_Score, and A3_Score have the strongest impact on the target outcome.

```
# Check variable coefficients and odds ratio
print(pd.DataFrame({'coef': black_log_reg.coef_[0], 'odds': np.e**black_log_reg.coef_[0]}, index=X.columns))
```

```
               coef      odds
A1_Score   0.212778  1.237109
A2_Score  -0.442746  0.642270
A3_Score   0.396077  1.485983
A4_Score  -0.098976  0.905764
A5_Score   0.184026  1.202047
A6_Score   0.415306  1.514834
A7_Score   0.127250  1.135701
A8_Score  -0.201792  0.817265
A9_Score   1.044013  2.840592
A10_Score  0.328076  1.388295
```

The accuracy for this model is quite low in comparison to the decision tree and random forest.

```
# Check accuracy
classificationSummary(valid_Y, black_log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8750)

       Prediction
Actual 0 1
     0 4 1
     1 0 3
```

9.4.2 Logistic Regression- Asian Males:

```
# Run a logistic regression
asian_log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
asian_log_reg.fit(train_X, train_Y)
```

In the odds ratio for this model A6_Score, A4_Score, and A9_Score have the strongest impact on the target outcome.

```
# Check variable coefficients and odds ratio
print(pd.DataFrame({'coef': asian_log_reg.coef_[0], 'odds': np.e**asian_log_reg.coef_[0]}, index=X.columns))
```

```
               coef      odds
A1_Score  -0.328343  0.720116
A2_Score  -0.396814  0.672459
A3_Score   0.336654  1.400254
A4_Score   0.784441  2.191182
A5_Score   0.245782  1.278621
A6_Score   1.016027  2.762198
A7_Score   0.670832  1.955864
A8_Score  -0.175521  0.839020
A9_Score   0.603510  1.828526
A10_Score -0.298877  0.741650
```

The accuracy for this model is higher than the accuracy for the neural network.

```
# Check accuracy
classificationSummary(valid_Y, asian_log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9310)

       Prediction
Actual  0  1
     0 26  0
     1  2  1
```

9.4.3 Logistic Regression- Middle Eastern Males:

```
# Run a logistic Regression
eastern_log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
eastern_log_reg.fit(train_X, train_Y)
```

A6_Score, A2_Score, and A7_Score have the strongest impact on the target in this odds ratio.

```
# Check variable coefficients and odds ratio
print(pd.DataFrame({'coef': eastern_log_reg.coef_[0], 'odds': np.e**eastern_log_reg.coef_[0]}, index=X.columns))
```

```
               coef      odds
A1_Score  -0.688278  0.502441
A2_Score   0.333134  1.395334
A3_Score   0.104790  1.110477
A4_Score  -0.175119  0.839357
A5_Score   0.156537  1.169454
A6_Score   1.132651  3.103874
A7_Score   0.224857  1.252143
A8_Score  -0.039395  0.961371
A9_Score   0.091852  1.096203
A10_Score  0.040537  1.041370
```

The accuracy for this model is the same as the accuracy for all the other models.

```
# Check accuracy
classificationSummary(valid_Y, eastern_log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9545)

       Prediction
Actual  0  1
     0 21  0
     1  1  0
```

9.4.4 Logistic Regression- Others:

```
# Run a logistic regression
others_log_reg = LogisticRegression(solver='liblinear', C=1, random_state=1)
others_log_reg.fit(train_X, train_Y)
```

A4_Score, A9_Score, and A3_Score have the strongest impact on the target in this odds ratio.

```
# Check variable coefficients and odds ratio
print(pd.DataFrame({'coef': others_log_reg.coef_[0], 'odds': np.e**others_log_reg.coef_[0]}, index=X.columns))
```

```
               coef      odds
A1_Score  -0.104043  0.901186
A2_Score   0.206849  1.229796
A3_Score   0.391276  1.478867
A4_Score   0.711848  2.037753
A5_Score   0.443744  1.558531
A6_Score  -0.077395  0.925525
A7_Score  -0.058000  0.943650
A8_Score  -0.644209  0.525078
A9_Score   0.573496  1.774459
A10_Score -0.032099  0.968410
```

Model accuracy is worse for this model than the rest of the Others models.

```
# Check accuracy
classificationSummary(valid_Y, others_log_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8462)

       Prediction
Actual  0  1
     0 22  0
     1  4  0
```

9.5 Association Rules:

9.5.0 Associations- White Male Responses:

Create a new dataset to assess the associations between _Score variables. Set the ID column as the index.

```
# Create a dataset using _Score variables an ID.
white_scores = white_df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'ID']]
# Set ID as index.
white_scores.set_index('ID', inplace=True)
white_scores.head()
```

ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score
7	1	1	1	1	0	0	0	0	1	0
8	1	1	0	0	1	0	0	1	1	1
10	1	1	1	1	1	1	1	1	1	1
17	0	0	0	0	0	0	0	1	0	1
30	0	0	0	0	0	0	0	0	0	0

Set the minimum itemset frequency to 25%.

```
# Check itemset frequency using apriori
questionsets = apriori(white_scores, min_support=.25, use_colnames=True) # Set minimum threshold to 25% frequency.
questionsets.head()
```

	support	itemsets
0	0.761468	(A1_Score)
1	0.623853	(A2_Score)
2	0.623853	(A3_Score)
3	0.614679	(A4_Score)
4	0.532110	(A5_Score)

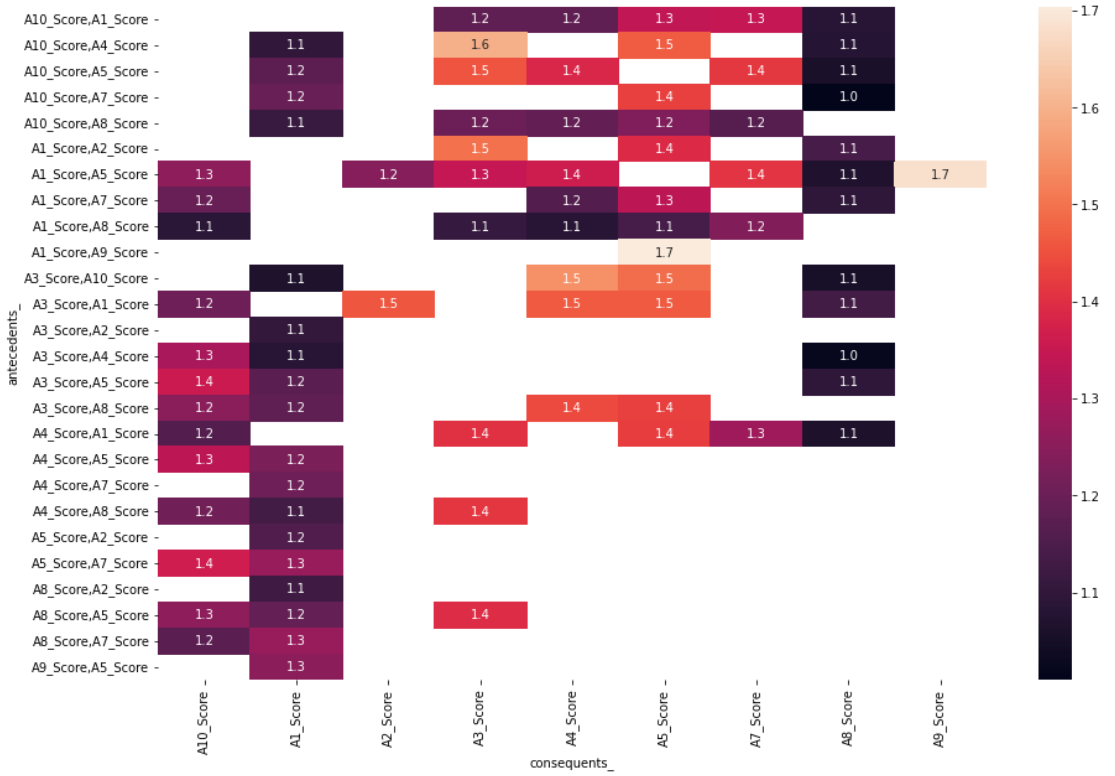
Set the minimum confidence level to 50%. Sort the values by lift.

```
# Check variable relationships using association rules.
white_rules = association_rules(questionsets, metric='confidence', min_threshold=0.5) # Set metric as 'confidence' with a threshold of 50%.
white_rules.sort_values(by=['lift'], ascending=False).head() # Order by Lift
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2745	(A6_Score, A4_Score, A8_Score)	(A10_Score, A9_Score)	0.266055	0.385321	0.256881	0.965517	2.505747	0.154364	17.825688
2758	(A10_Score, A9_Score)	(A6_Score, A4_Score, A8_Score)	0.385321	0.266055	0.256881	0.666667	2.505747	0.154364	2.201835
2028	(A9_Score, A2_Score)	(A6_Score, A3_Score, A1_Score)	0.330275	0.311927	0.256881	0.777778	2.493464	0.153859	3.096330
2009	(A6_Score, A3_Score, A1_Score)	(A9_Score, A2_Score)	0.311927	0.330275	0.256881	0.823529	2.493464	0.153859	3.795107
2027	(A3_Score, A9_Score)	(A6_Score, A1_Score, A2_Score)	0.376147	0.275229	0.256881	0.682927	2.481301	0.153354	2.285815

The most important associations for White males are:

- A1_Score & A5_Score → A9_Score
- A1_Score & A9_Score → A5_Score
- A10_Score & A4_Score → A3_Score.



9.5.1 Associations- Black Male Responses:

```
# Create dataset with _Score variables and ID
black_scores = black_df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'ID']]
# Set ID column as index
black_scores.set_index('ID', inplace=True)
black_scores.head()
```

ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score
28	0	0	0	0	0	0	0	1	0	0
68	0	0	0	0	1	0	0	0	0	0
130	1	0	0	0	0	0	0	1	0	1
233	0	1	1	0	0	0	0	0	0	0
439	1	0	1	1	1	0	1	0	0	1

Set the minimum itemset frequency to 20%.

```
# Identify frequent itemsets using apriori
questionsets = apriori(black_scores, min_support=.2, use_colnames=True) # min_support for itemsets is 20%.
questionsets.head()
```

	support	itemsets
0	0.70	(A1_Score)
1	0.40	(A2_Score)
2	0.45	(A3_Score)
3	0.50	(A4_Score)
4	0.55	(A5_Score)

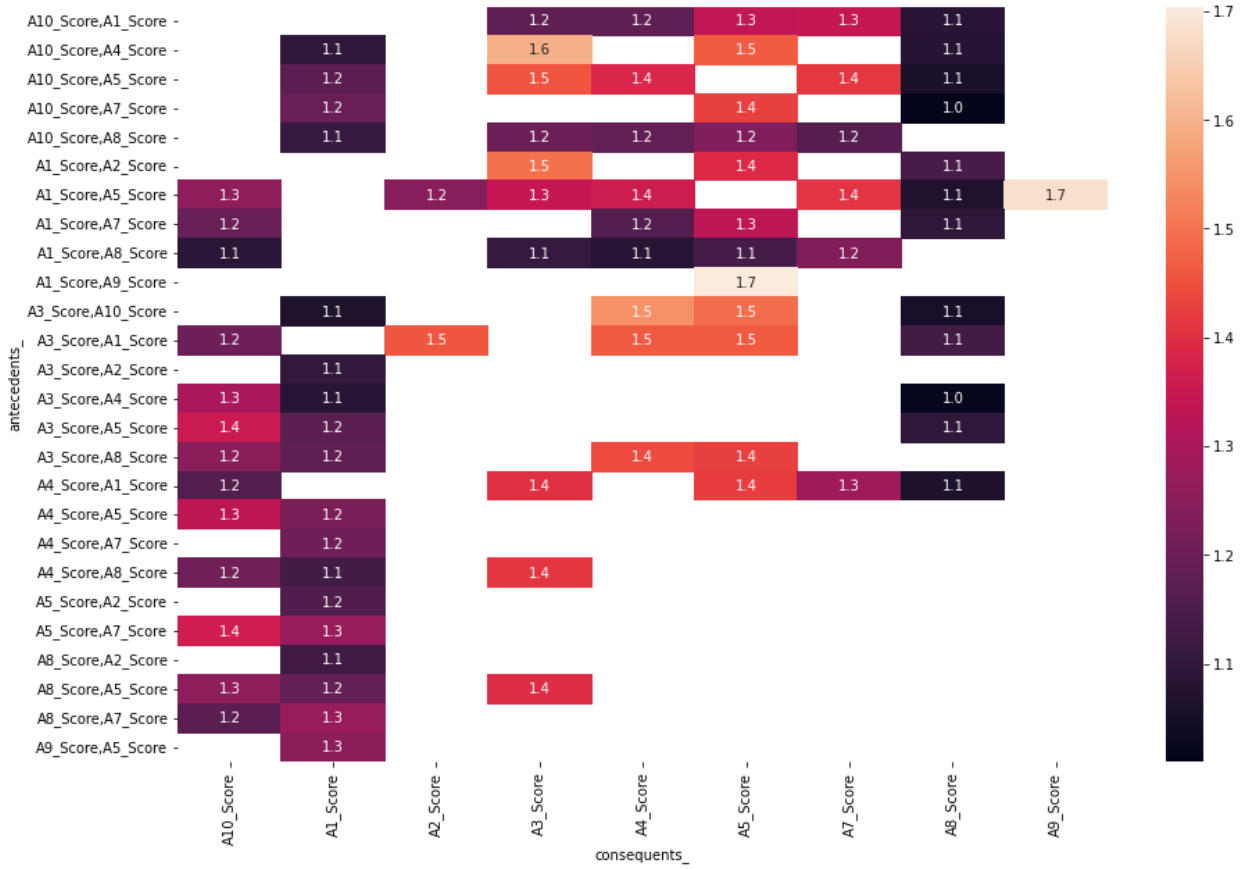
Set the minimum confidence level to 50%.

```
# Check for relationships between _Scores using association rules. Set metric as 'confidence' and min_threshold at 50%.
black_rules = association_rules(questionsets, metric='confidence', min_threshold=0.5)
# Sort values by lift
black_rules.sort_values(by=['lift'], ascending=False).head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
1967	(A6_Score, A9_Score)	(A10_Score, A8_Score, A5_Score)	0.25	0.25	0.2	0.8	3.2	0.1375	3.75
1960	(A10_Score, A8_Score, A5_Score)	(A6_Score, A9_Score)	0.25	0.25	0.2	0.8	3.2	0.1375	3.75
2333	(A6_Score, A1_Score, A9_Score)	(A10_Score, A8_Score, A5_Score)	0.25	0.25	0.2	0.8	3.2	0.1375	3.75
2339	(A10_Score, A8_Score, A5_Score)	(A6_Score, A1_Score, A9_Score)	0.25	0.25	0.2	0.8	3.2	0.1375	3.75
2350	(A6_Score, A9_Score)	(A10_Score, A1_Score, A8_Score, A5_Score)	0.25	0.25	0.2	0.8	3.2	0.1375	3.75

The most important associations for Black males are the same as those for White Males:

- A1_Score & A5_Score → A9_Score
- A1_Score & A9_Score → A5_Score
- A10_Score & A4_Score → A3_Score.



9.5.2 Associations- Asian Male Responses:

```
# Create a dataset using _Score variables and ID. Set ID as index.
asian_scores = asian_df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'ID']]
asian_scores.set_index('ID', inplace=True)
asian_scores.head()
```

ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score
9	1	1	1	1	0	1	1	1	1	0
70	0	0	1	1	0	0	0	1	0	0
101	1	0	0	0	0	0	0	0	1	1
102	1	1	0	0	1	0	1	1	0	0
104	0	0	0	0	0	0	0	0	0	0

Set the minimum itemset frequency threshold to 15%.

```
# Check for frequent itemsets using apriori
questionsets = apriori(asian_scores, min_support=.15, use_colnames=True) # Set minimum support to 15%.
questionsets.head()
```

	support	itemsets
0	0.625000	(A1_Score)
1	0.250000	(A2_Score)
2	0.333333	(A3_Score)
3	0.388889	(A4_Score)
4	0.444444	(A5_Score)

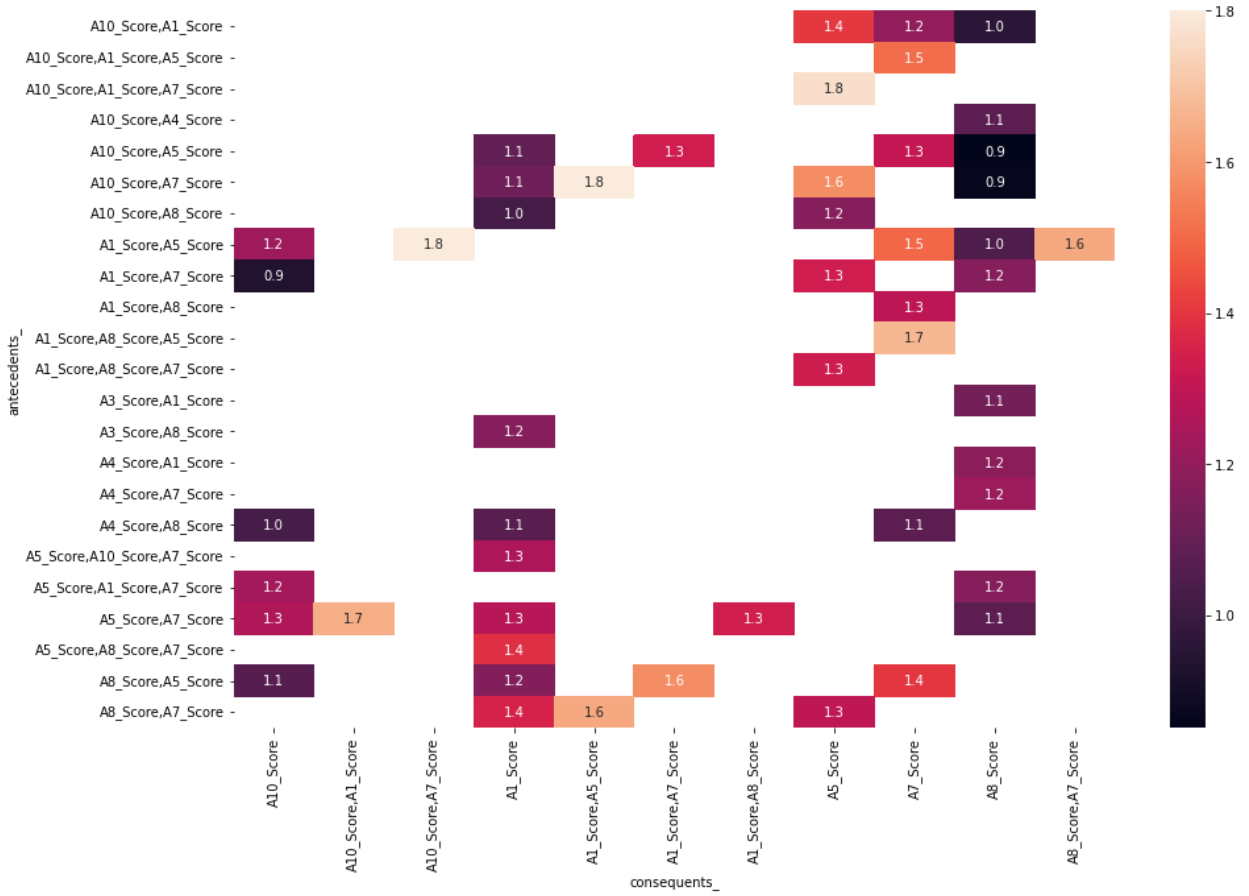
Set the minimum confidence level to 50%.

```
# Check for relations between _Scores using association rules
rules = association_rules(questionsets, metric='confidence', min_threshold=0.5) # Use confidence as the metric with 50% minimum threshold.
rules.sort_values(by=['lift'], ascending=False).head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
85	(A10_Score, A7_Score)	(A1_Score, A5_Score)	0.277778	0.305556	0.152778	0.550000	1.800000	0.067901	1.543210
88	(A1_Score, A5_Score)	(A10_Score, A7_Score)	0.305556	0.277778	0.152778	0.500000	1.800000	0.067901	1.444444
81	(A10_Score, A1_Score, A7_Score)	(A5_Score)	0.194444	0.444444	0.152778	0.785714	1.767857	0.066358	2.592593
76	(A1_Score, A8_Score, A5_Score)	(A7_Score)	0.222222	0.486111	0.180556	0.812500	1.671429	0.072531	2.740741
86	(A5_Score, A7_Score)	(A10_Score, A1_Score)	0.277778	0.333333	0.152778	0.550000	1.650000	0.060185	1.481481

The most important associations for Asian males are:

- A1_Score & A5_Score ←→ A10_Score & A7_Score
- A1_Score & A8_Score → A7_Score



9.5.3 Associations- Middle Eastern Male Responses:

```
# Create dataset with _Score variables and ID. Set ID as the index
eastern_scores = eastern_df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'ID']]
eastern_scores.set_index('ID', inplace=True)
eastern_scores.head()
```

ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score
15	1	1	0	1	1	0	0	1	0	1
16	1	0	0	0	0	0	1	1	1	1
22	0	0	0	1	0	0	1	1	1	1
23	0	0	0	0	0	0	0	1	0	1
27	0	0	0	0	0	0	0	1	0	0

Set the minimum itemset frequency to 15%.

```
# Check for frequent itemsets using apriori.
questionsets = apriori(eastern_scores, min_support=.15, use_colnames=True) # Set min frequency to 15%.
questionsets.head()
```

	support	itemsets
0	0.574074	(A1_Score)
1	0.407407	(A2_Score)
2	0.481481	(A3_Score)
3	0.425926	(A4_Score)
4	0.481481	(A5_Score)

Set the minimum confidence level to 15%.

```
# Check for relations between _Score variables using association rules.
rules = association_rules(questionsets, metric='confidence', min_threshold=0.5) # Use confidence as metric and set threshold to 50%
rules.sort_values(by=['lift'], ascending=False).head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
89	(A8_Score, A5_Score)	(A3_Score, A10_Score)	0.314815	0.277778	0.166667	0.529412	1.905882	0.079218	1.534722
85	(A3_Score, A10_Score)	(A8_Score, A5_Score)	0.277778	0.314815	0.166667	0.600000	1.905882	0.079218	1.712963
80	(A8_Score, A5_Score)	(A3_Score, A1_Score)	0.314815	0.277778	0.166667	0.529412	1.905882	0.079218	1.534722
77	(A3_Score, A1_Score)	(A8_Score, A5_Score)	0.277778	0.314815	0.166667	0.600000	1.905882	0.079218	1.712963
81	(A3_Score, A10_Score, A8_Score)	(A5_Score)	0.185185	0.481481	0.166667	0.900000	1.869231	0.077503	5.185185

The most important associations for Middle Eastern Males are:

- A4_Score & A8_Score → A3_Score
- A4_Score & A3_Score → A1_Score

9.5.4 Associations- Others Responses:

```
# Create a dataset using _Score variables and ID. Set ID as index.
others_scores = others_df[['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'ID']]
others_scores.set_index('ID', inplace=True)
others_scores.head()
```

ID	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score
5	1	1	1	1	1	0	1	1	1	1
13	1	0	0	0	0	0	1	1	0	1
19	0	0	0	0	0	0	1	1	0	1
20	0	1	1	1	0	0	0	0	0	0
24	1	1	1	1	0	0	0	1	0	0

Set the minimum itemset frequency to 15%.

```
# Check for frequent itemsets using apriori
questionsets = apriori(others_scores, min_support=.15, use_colnames=True) # Set minimum support to 15%.
questionsets.head()
```

	support	itemsets
0	0.634921	(A1_Score)
1	0.412698	(A2_Score)
2	0.349206	(A3_Score)
3	0.333333	(A4_Score)
4	0.365079	(A5_Score)

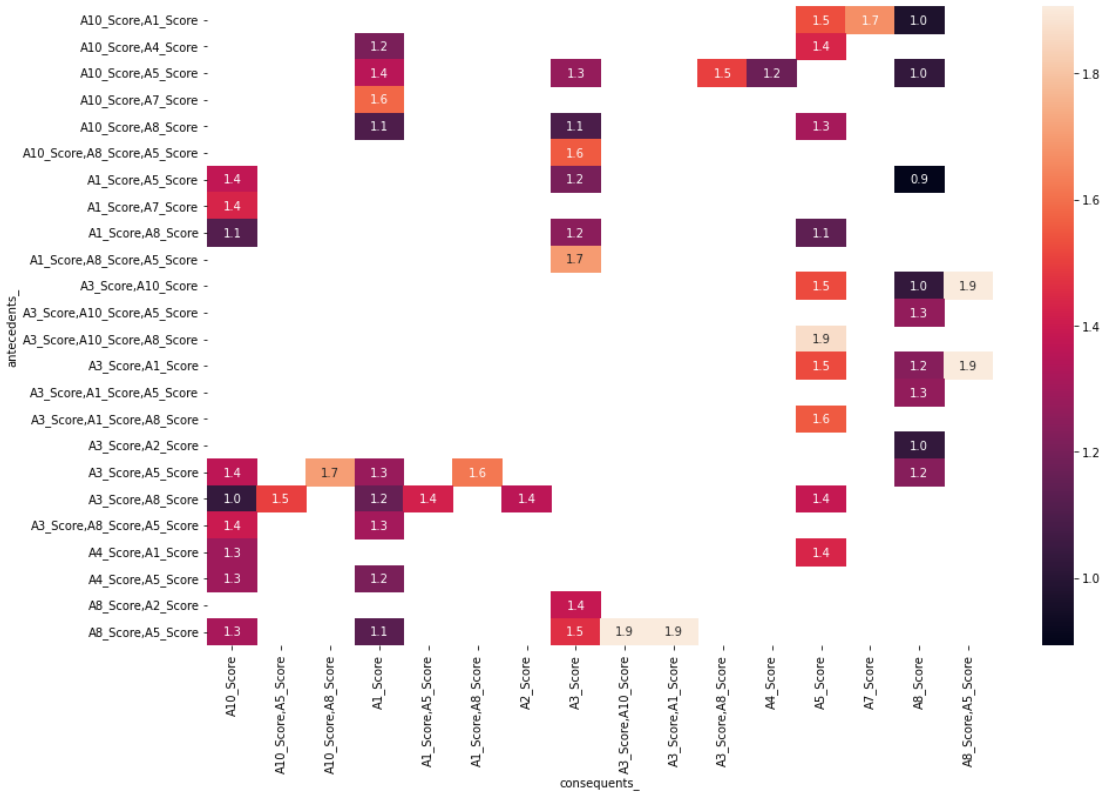
Set the minimum confidence level to 50%.

```
# Check for relations between _Scores using association rules
rules = association_rules(questionsets, metric='confidence', min_threshold=0.5) # Use confidence as the metric with 50% minimum threshold.
rules.sort_values(by=['lift'], ascending=False).head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
71	(A4_Score, A8_Score)	(A3_Score)	0.222222	0.349206	0.158730	0.714286	2.045455	0.081129	2.277778
39	(A4_Score, A1_Score)	(A3_Score)	0.238095	0.349206	0.158730	0.666667	1.909091	0.075586	1.952381
16	(A4_Score)	(A3_Score)	0.333333	0.349206	0.222222	0.666667	1.909091	0.105820	1.952381
15	(A3_Score)	(A4_Score)	0.349206	0.333333	0.222222	0.636364	1.909091	0.105820	1.833333
40	(A3_Score, A1_Score)	(A4_Score)	0.253968	0.333333	0.158730	0.625000	1.875000	0.074074	1.777778

The strongest associations for the Others subgroup are:

- A8_Score & A5_Score ←→ A3_Score & A10_Score
- A8_Score & A5_Score ←→ A3_Score & A1_Score



9.6 Model Comparison for Entire Male Dataset and Male Ethnic Subgroups:

Datasets	Maximal Trees	Optimized Trees	Random Forest	Logistic Regression	Neural Network
Full Male	91.8%	91.8%	98.0%	98.6%	100.0%
White Male	86.4%	84.0%	88.5%	84.1%	86.4%
Black Male	100.0%		100.0%	87.5%	87.5%
Asian Male	93.1%	89.7%	96.6%	93.1%	89.7%
Middle Eastern Male	95.5%		95.5%	95.5%	95.5%
Others Male	92.3%		84.6%	84.6%	88.5%

For the analysis of the full male subgroup, the neural network is the best model. Like the female dataset, this accuracy for this model is higher than the neural models I ran for the analysis on the full dataset using _Score variables as predictors. This further proves that there are gender differences between male and female responses to the questions on the AQ-10 that improve model accuracy when analyzed separately. Looking at the accuracies across all the models for the full male dataset, they are greater than all the accuracies for the full female dataset. This indicates that the dataset is geared more towards males than females.

For our ethnic subgroups, the best models were the Random Forest, except for the Others subgroup where the maximal tree was the best model. Comparing the splits of the maximal trees across all the subgroups we saw that they split on different variables. For the Random Forest feature importance, each of the subgroups had its own unique ranking for feature importance. This supports the hypothesis that there are ethnic differences in responses to the AQ-10.

10.0 Conclusion:

The purpose of this analysis was to determine whether we can improve ASD screening by accounting for gender and ethnic differences in autistic behaviors. We have found that:

- Model accuracy improved when subgrouping the data by gender.
- Model accuracy for males tends to be higher than model accuracy for females.
- Feature importance differs between males and females.
- Questionnaire response itemsets differ for males and females.
- Model accuracy does not tend to improve when subgrouping the dataset by ethnicity for both males and females.
- Feature importance differs between ethnic groups for both genders.
- Questionnaire response itemsets differ by ethnic group except for Black & White participants who share the same response associations.

NOTE: The accuracy for all of our models is extremely high, which in most cases would indicate that there is something wrong with the models. However, in this case we are using variables that are highly predictive of the target since each variable contributes to an overall score that determines the target outcome, so it makes sense that our models are extremely accurate.

11.0 Recommendations:

It is clear from this analysis that there are gender and ethnic differences in autistic behavior, made evident by differences in responses to the AQ-10-Adult assessment. We recommend that clinical experts:

- Expand their research on female manifestations of ASD.
- Research the differences in ASD symptoms and behaviors for ethnic groups.
- Create more nuanced screening assessments for ASD that are specialized for gender and ethnic groups.