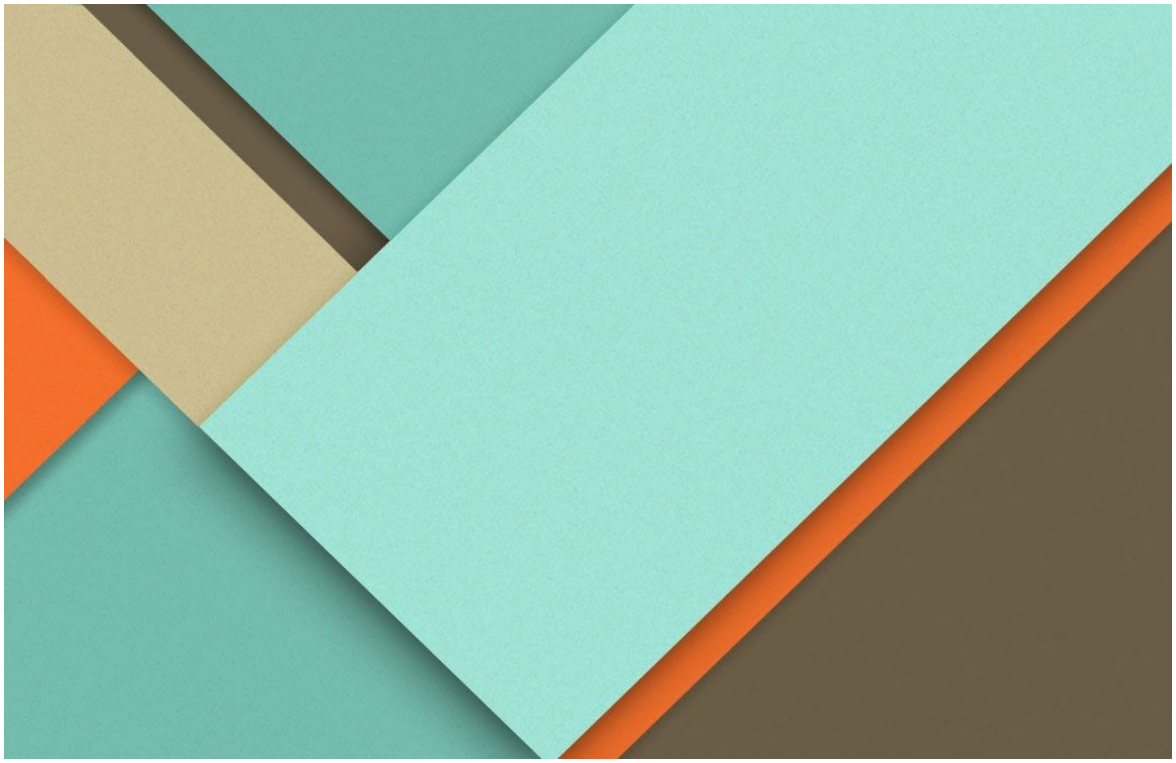


ML Classifier: Autism Screening Adult Data Set

Building an ML classifier on Autism Screening Adult Data Set to predict if Autism Spectrum Disorder is present or not.

Topic Chosen :

B.
Applying ML Classification algorithms on the data set and getting inferences from the data. You may use the appropriate ML algorithm packages available in Python but must know which algorithm you use and the concept behind it.

Overview :

In this IDS project report, we analyze, describe and document the important observations and inferences drawn from the Autism Screening Adult dataset. This report not only elucidates the descriptive stats and summarization of the dataset chosen but also builds and trains an ML Classifier using **Logistic Regression Algorithm** The crucial highlights of this report are:

- Defining the problem statement
- Preliminary & Exploratory Data Analysis
- Cleaning and Preprocessing the Data
- Building & Training the model

About the Dataset: Source & Metainformation

Autism Screening Adult Dataset was chosen from UCI Machine Learning Repository . A zip folder was downloaded from the Data Folder which had the data file and data description file. The data file was in .arff format , so we had converted it to .csv format.

The extraction of this data was done by-

Source :

Fadi Fayez Thabtah

Department of Digital Technology

Manukau Institute of Technology,

Auckland, New Zealand

fadi.fayez '@' manukau.ac.nz

The entire dataset consists of 704 instances and 21 attributes. The data contains a good blend of Categorical, continuous, binary and missing values. Hence, it provided a good learning experience in dealing and appropriately handling the data to make it meaningful and extract inferences by analyzing from the data.

Data Set Characteristics:	N/A	**Number of Instances:**	704	**Area:**	Social
Attribute Characteristics:	Integer	**Number of Attributes:**	21	**Date Donated**	2017-12-24
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	46682

The language used for this project: **Python 3**

Attributes of the Data Set:

The Data set consists of 21 columns including 20 predictive (or non-class attributes) columns or features and 1 class-attribute. There are 11 numerical values, 9 categorical values and, 1 continuous value. Following is the description of attributes :

Class Attribute : ASD : 1 or 0

1.**A1_Score:** *Binary (0, 1) :* The answer code of the question based on the screening method used

2.**A2_Score**: *Binary (0, 1)* : The answer code of the question based on the screening method used

3.**A3_Score:** *Binary (0, 1):* The answer code of the question based on the screening method used

4.**A4_Score**: *Binary (0, 1):* The answer code of the question based on the screening method used

5.**A5_Score**: *Binary (0, 1):* The answer code of the question based on the screening method used

6.**A6_Score**: *Binary (0, 1):*The answer code of the question based on the screening method used

7.**A7_Score**: *Binary (0, 1):* The answer code of the question based on the screening method used

8.**A8_Score**: *Binary (0, 1):*The answer code of the question based on the screening method used

9.**A9_Score**: *Binary (0, 1):* The answer code of the question based on the screening method used

10.**A10_Score**: *Binary (0, 1):* The answer code of the question based on the screening method used

11.**age :** *Numerical:* Age in years

12.**Gender:** *Categorical* :f,m

13.**ethnicity** : *Categorical* : White-European,Latino, Others,Black,Asian,'Middle Eastern ',Pasifika,'South Asian',Hispanic,Turkish, others.

14. **jaundice :** *Categorical :* no,yes

15. **autism** : *Categorical :* no,yes

16. contry_of_res *Categorical :* United States',Brazil,Spain,Egypt,'NewZealand',Bahamas,Burundi,Austria,Argentina,Jordan,Ireland,United Arab Emirates',Afghanistan,Lebanon,'UnitedKingdom','SouthAfrica',Italy,Pakistan,Bangladesh,Chile,France,China,Australia,Canada,'Saudi Arabia',Netherlands,Romania,Sweden,Tonga,Oman,India,Philippines,'Sri Lanka','Sierra Leone',Ethiopia,'Viet Nam',Iran,'Costa Rica',Germany,Mexico,Russia,Armenia,Iceland,Nicaragua,'Hong Kong',Japan,Ukraine,Kazakhstan,AmericanSamoa,Uruguay,Serbia,Portugal,Malaysia,Ecuador,Niger,Belgium,Bolivia,Aruba,Finland,Turkey,Nepal,Indonesia,Angola,Azerbaijan,Iraq,'Czech Republic',Cyprus

17.**used_app_before** *Categorical:* yes,no

18: **result** : *numeric* : 1,2,3,4,5,6,7,8,9,10

19: **age_desc :** *Categorical:* 18 and more

20. **relation:** *Categorical:* Self,Parent,Health care professional,Relative,Others

Defining the problem statement

The problem statement to be dealt with is a Binary Classification Problem. The data contains anonymous information such as age, gender,jaundice,autism,etc. The goal is to train a binary classifier to predict whether a person has Autism Spectrum Disorder or not.

Loading the Data

● Python Library: Pandas has been mainly used in order to handle and explore the dataset. So, dataset has been loaded into pandas dataframe using the read_csv() function.
● Data Set doesn't contain any headers or column-names. So headers have been assigned while reading the .csv files using names=columns, where columns is a python list of header names.
● The data consists of some whitespaces before and after the data values. These have been removed while loading the data using the ' *, *' separator.
● The data consists of many missing values represented by '?', these have been loaded into the dataframe as na_values. (i.e not available data = '?')
● Python engine is used as engine='python' in order to avoid a compiler warning.

●data_head() shows headers alongwith first few instances.

●data_info() shows information about different attributes - number of non-null values and attribute type.

These operations are represented by following screenshot of code

```
In [385]: columns = ["A1_Score", "A2_Score","A3_Score", "A4_Score","A5_Score","A6_Score" ,"A7_Score", "A8_Score","A9_Score", "A10_Score","
          data=pd.read_csv('project_autism.csv', names=columns, sep=' *, *', na_values='?', engine='python')
```

```
In [387]: data.head()
```

Out[387]:

	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score	...	gender	ethnicity	jundice	austim	contr
0	1	1	1	1	0	0	1	1	0	0	...	f	White-European	no	no	'Unite
1	1	1	0	1	0	0	0	1	0	1	...	m	Latino	no	yes	
2	1	1	0	1	1	0	1	1	1	1	...	m	Latino	yes	yes	
3	1	1	0	1	0	0	1	1	0	1	...	f	White-European	no	yes	'Unite
4	1	0	0	0	0	0	0	1	0	0	...	f	NaN	no	no	

5 rows × 21 columns

```
In [389]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 704 entries, 0 to 703
Data columns (total 21 columns):
A1_Score            704 non-null int64
A2_Score            704 non-null int64
A3_Score            704 non-null int64
A4_Score            704 non-null int64
A5_Score            704 non-null int64
A6_Score            704 non-null int64
A7_Score            704 non-null int64
A8_Score            704 non-null int64
A9_Score            704 non-null int64
A10_Score           704 non-null int64
Age                 702 non-null float64
gender              704 non-null object
ethnicity           609 non-null object
jundice             704 non-null object
austim              704 non-null object
contry_of_res       704 non-null object
used_app_before     704 non-null object
result              704 non-null int64
age_desc            704 non-null object
relation            609 non-null object
ASD                 704 non-null object
```

```
In [392]: numerical_attributes = data.select_dtypes(include=['int64'])
          numerical_attributes.head()
```

Out[392]:

	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score	result
0	1	1	1	1	0	0	1	1	0	0	6
1	1	1	0	1	0	0	0	1	0	1	5
2	1	1	0	1	1	0	1	1	1	1	8
3	1	1	0	1	0	0	1	1	0	1	6
4	1	0	0	0	0	0	0	1	0	0	2

Observations:

● Data set contains 704 instances
● 3 Attributes namely,ethnicity,age and relation have less than 704 non-null values. This means that these attributes have missing values.
● Data set contains numeric attributes (int64) as well as categorical attributes (object).
● numerical_attributes.columns give names of all the numerical attributes.

Exploratory Data Analysis (EDA)

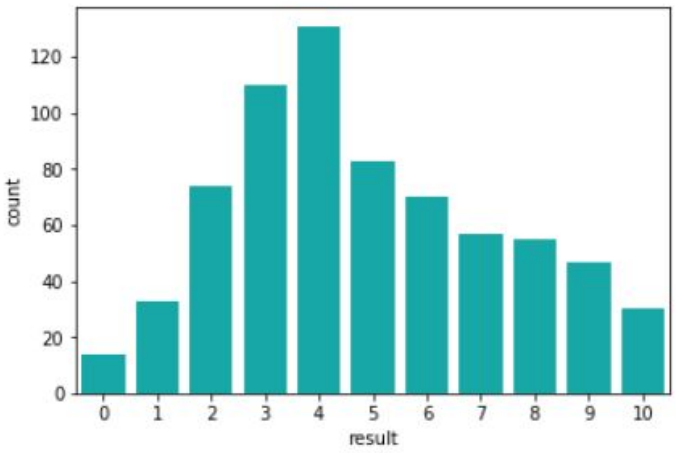
Data Visualization

Having identified which attributes are numeric and which ones are categorical, now we engage in various visualization methods to analyze and represent the data meaningfully. Following is the code for data visualizations using count plots , histograms and heatmap in continuation with the previous code. We import pyplot from **matplotlib** for graphs and **numpy** for matching object data types and axes_style from **seaborn**

1. Count plot for Numerical Attributes:

To understand the relationship of some intuitively related attributes with the class attribute, we plotted the count-plots as follows to derive important observations. We have used **seaborn** library to draw count-plots.

```
In [394]: sns.countplot(x="result", data=data, color="c");
```



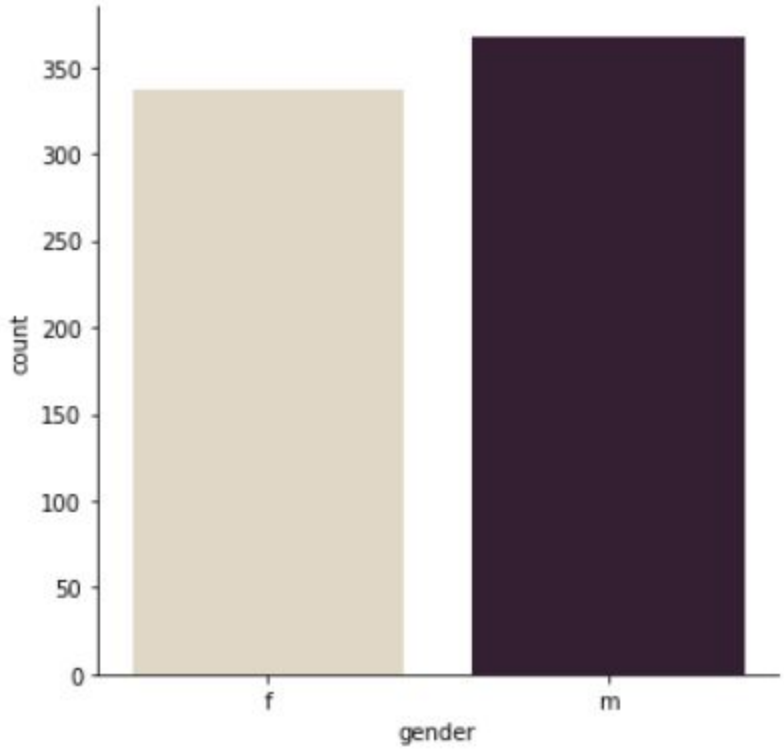
2. Catplot for Categorical Attributes:

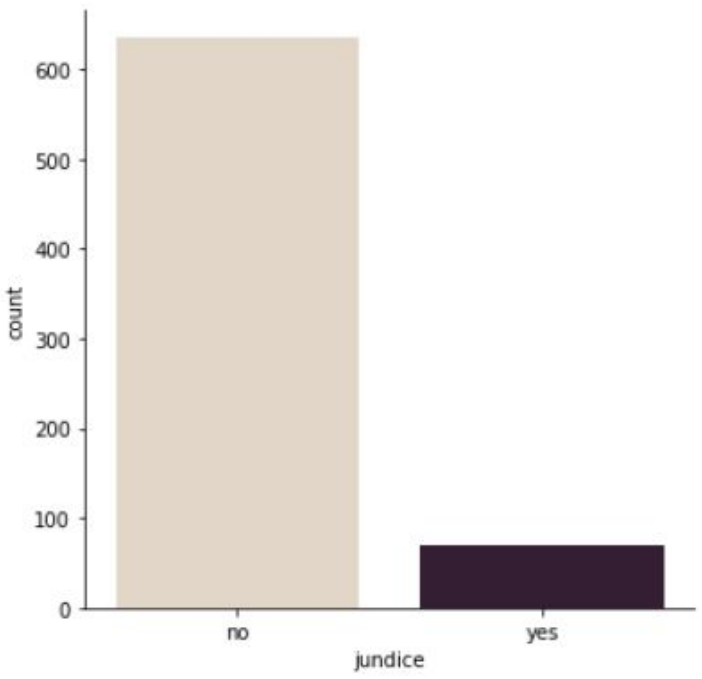
We represent each of the categorical attributes using catlpot as follows:

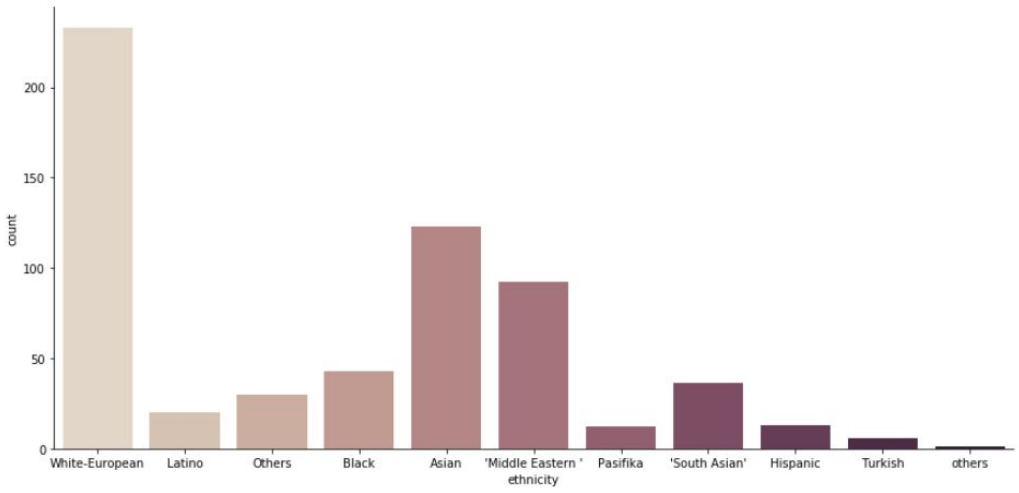
CODE:

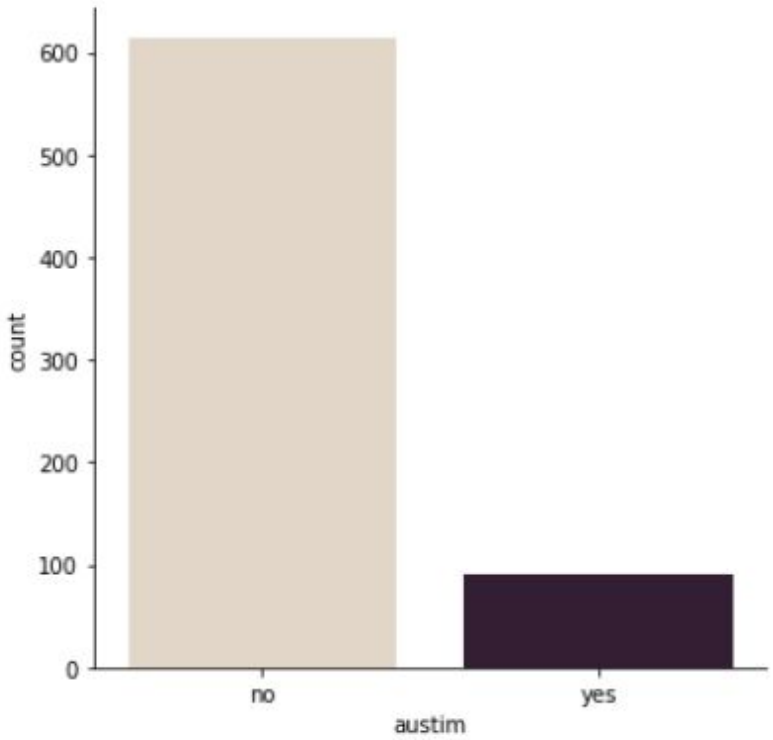
```
In [396]: sns.catplot(x="gender", kind="count", palette="ch:.28", data=data);
          sns.catplot(x="ASD", kind="count", palette="ch:.25", data=data);
          sns.catplot(x="relation", kind="count", palette="ch:.25", data=data, height=8, aspect=2);
          sns.catplot(x="age_desc", kind="count", palette="ch:.28", data=data);
          sns.catplot(x="used_app_before", kind="count", palette="ch:.25", data=data);
          sns.catplot(x="contry_of_res", kind="count", palette="ch:.25", data=data, height=10, aspect=5);
          sns.catplot(x="austim", kind="count", palette="ch:.25", data=data);
          sns.catplot(x="ethnicity", kind="count", palette="ch:.25", data=data, height=6, aspect=2);
          sns.catplot(x="jundice", kind="count", palette="ch:.25", data=data);
```

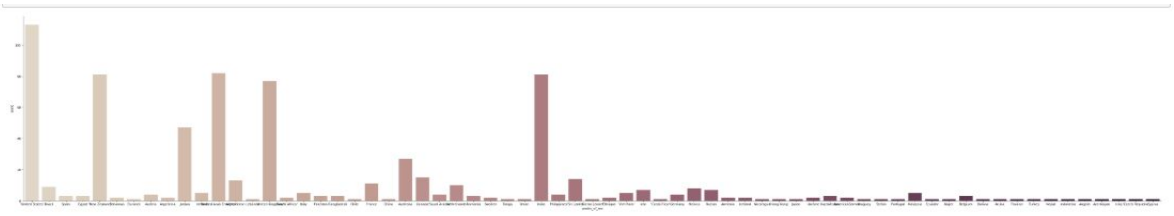
We represent each of the categorical attributes using as follows:

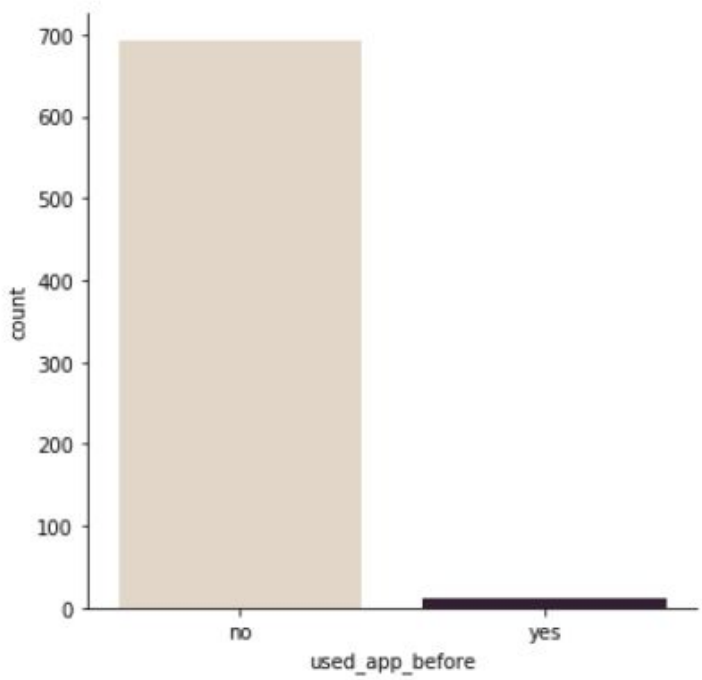

350
300
250
200
150
100
50
0
count
f
m
gender

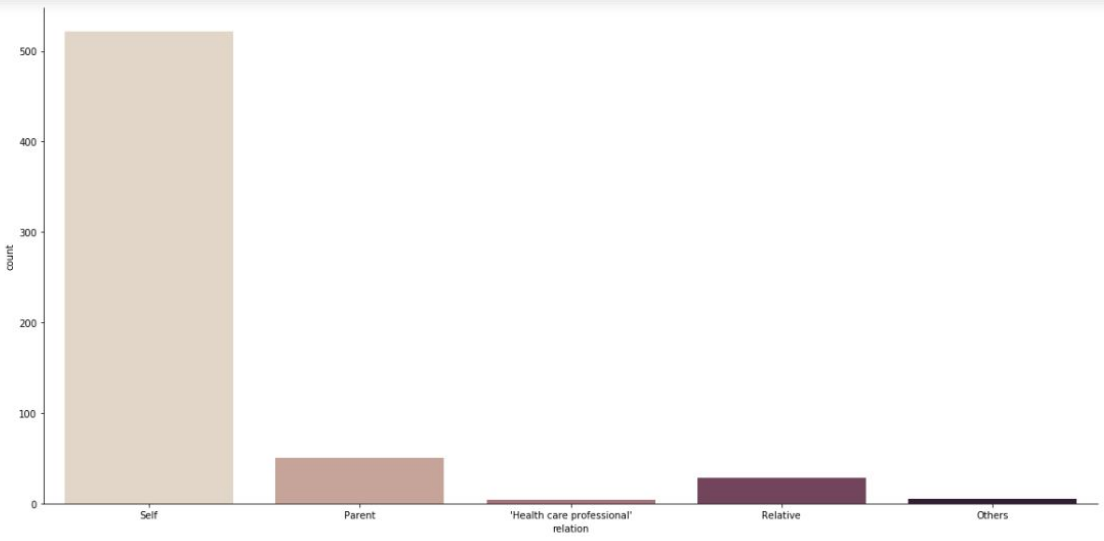

600
500
400
300
200
100
0
count
no
yes
jundice

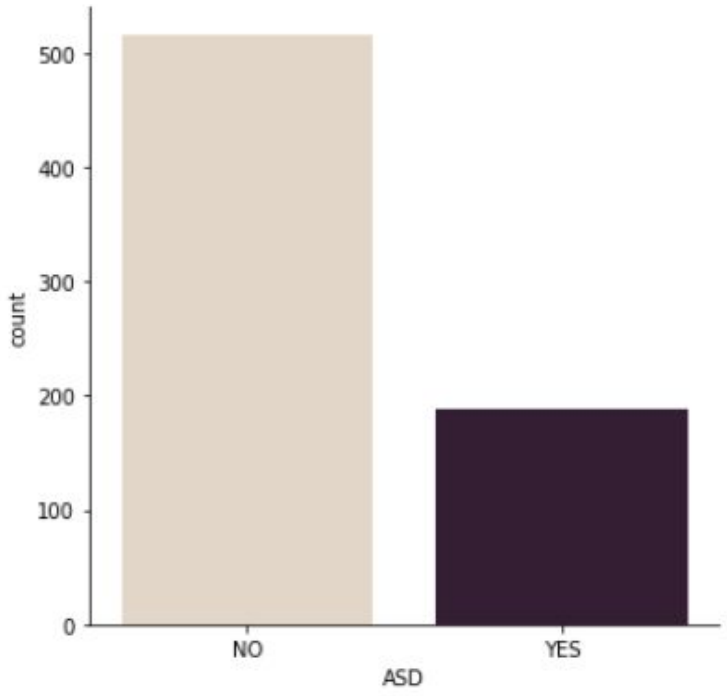

200
150
100
50
0
count
White-European
Latino
Others
Black
Asian
'Middle Eastern '
Pasifika
'South Asian'
Hispanic
Turkish
others
ethnicity

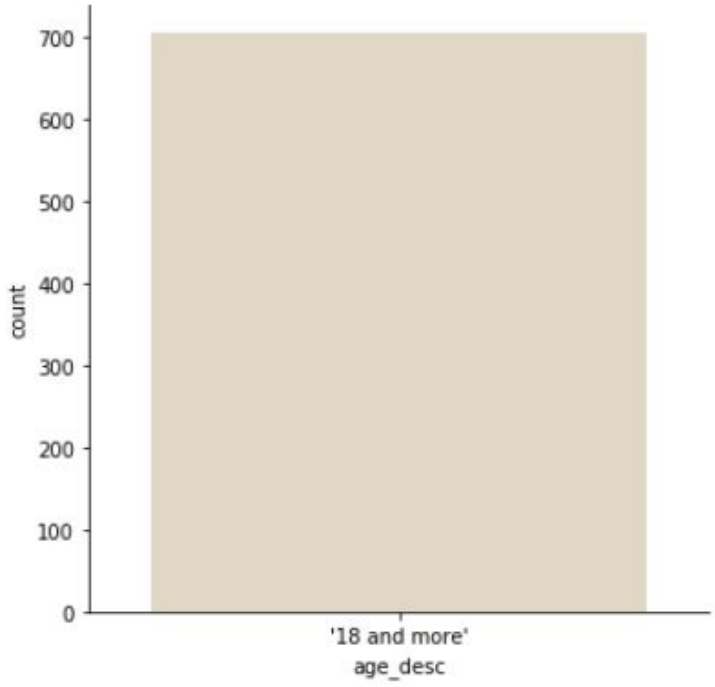

600
500
400
300
200
100
0
count
no
yes
austim




700
600
500
400
300
200
100
0
count
no
yes
used_app_before


500
400
300
200
100
0
count
Self
Parent
'Health care professional'
Relative
Others
relation


500
400
300
200
100
0
count
NO
YES
ASD


700
600
500
400
300
200
100
0
count
'18 and more'
age_desc

Observations:

- Most people are not born with jaundice
- The majority of people are White -European in our data.
- The highest number of people in our data belong to the US.
- Almost all the people in our data have not used the app before.
- Age_desc gives detailed information about age. In our data, all the people are above 18 years old, so this attribute doesn't make any extra contribution towards are the prediction.So, we can drop this attribute.
- In our data approx 28.5% of people are diagnosed with ASD.

Summary Statistics

Describing data in terms of its mean, median, mode, standard deviation and other such related parameters aid in better analysis and draw conclusion in terms of around which value the data is concentrated, how much is the spread of the data, what is the representative value of the data and other such questions, which are easily answerable using descriptive analysis.

data.describe() is used to describe such parameters of the numerical data, since such parameters doesn't carry any meaning for the categorical data.

```
In [405]: data.describe()
```

Out[405]:

	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

The above description of the data gives count of instances of each numerical attribute, its mean, standard deviation, minimum and maximum values as well as quartile information (25th percentile, median and 75th percentile).

Data Preprocessing

Data Cleaning

This section is aimed at handling the **missing** and **duplicat**e values to produce a very clean dataset on which further preprocessing and analysis can be carried out. It was observed in EDA that only categorical features have missing values and that too only three of them while all the numerical attributes are perfectly clean.

Basic cleaning has already been applied while loading the data into the data frame. This includes replacing '?' mark values with NA and removing unnecessary whitespaces. These NA filled values are missing values that now can be easily replaced with appropriate values to produce clean data.

1. Handling Missing Values

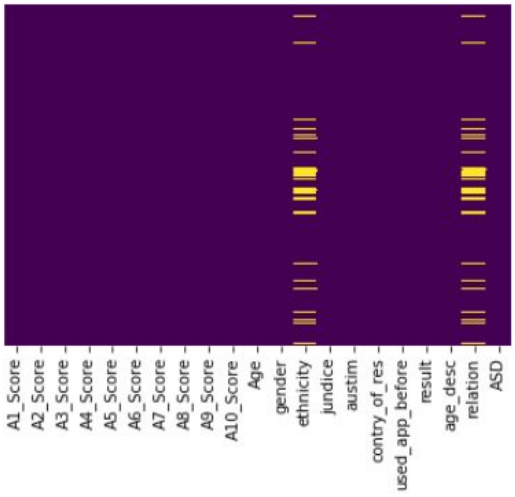
We first use .isnull().sum() to find the count of missing values in each column and then observe which columns have null count > 0.

```
In [391]: data.isnull().sum()

Out[391]: 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          95
          jundice             0
          austim              0
          contry_of_res       0
          used_app_before     0
          result              0
          age_desc            0
          relation           95
          ASD                 0
          dtype: int64
```

★ As stated earlier, we have got missing values in categorical attributes namely, age, ethnicity, and relation having 2, 95 and 95 respectively.

The missing values can also be represented through the graph using heatmap as follows:

```
In [390]: sns.heatmap(data.isnull(),yticklabels=False ,cbar=False,cmap='viridis')

Out[390]: <matplotlib.axes._subplots.AxesSubplot at 0x17a23033a20>
```



★ The yellow marks in the graph are for missing values.

We replace missing values with mode (i.e label with the maximum number of instances) using .fillna() function :

1. **age**

   The mode is found out using .value_counts() function.

```
In [407]: data['Age'].value_counts()

Out[407]: 21.0    49
          20.0    46
          23.0    37
          22.0    37
          19.0    35
          24.0    34
          27.0    31
          18.0    31
          30.0    30
          26.0    28
          25.0    27
          29.0    27
          28.0    24
          31.0    21
          32.0    18
          17.0    18
          35.0    17
          37.0    17
          40.0    16
          33.0    16
          42.0    15
          36.0    13
          38.0    12
          34.0    12
          43.0    11
          44.0    10
```

2. **ethnicity** and **relation**

   The mode can be observed from their respective graphs above.

   The missing values are replaced by the mode of respective attribute.

```
In [408]: data['Age'].fillna(21, inplace=True)
          data['ethnicity'].fillna('White-European', inplace=True)
          data['relation'].fillna('Self', inplace=True)
```

2.Handling Duplicate Values

Two rows are duplicate if all of their input attribute values, as well as output attribute value, match exactly. We have used .***duplicated()*** to find all the duplicate instances. ***Keep="False"*** marks 1st occurrence of an instance as well as all succeeding exactly matching occurrences of that instance as duplicate or false. If an instance has only 1 occurrence, then it'll be marked true and not displayed.

As observed by the output before dropping duplicates, row count was 704, and after performing drop_duplicates we have a total of 699 rows left. So our data had 5 duplicate instances.

Duplicate data

```
In [411]: data.shape
Out[411]: (704, 21)

In [412]: data =data.drop_duplicates(keep='first')

In [413]: data.shape
Out[413]: (699, 21)
```

Attribute Conversion: Categorical to Numerical

For classification, we are going to use Logistic Regression. This classifier only works for the numerical data, therefore we need to convert all our categorical attributes to numerical attributes.

1. ***gender*** :

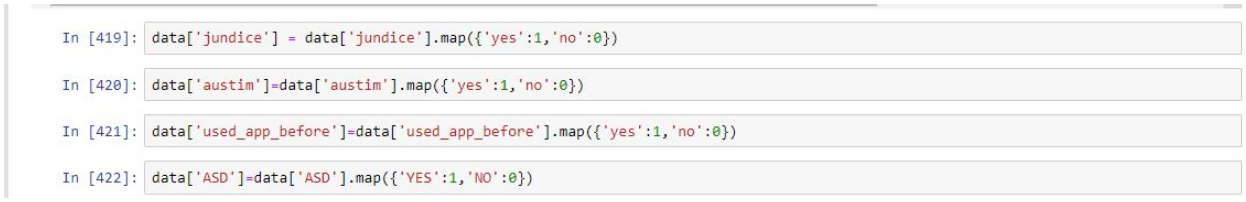
Gender has only 2 possible attribute values or categories - "***m***" and ***"f"*** The values do not have any order since its a nominal attribute. So we can directly map them to '0' and '1', where 0 is denoting 'f' and 1 is denoting

```
In [416]: data['gender'] = data['gender'].map({'m':1,'f':0})
```

'm'.

2. ***jaundice, autism, used_app_before, ASD*** :

   They are categorical variables with only two categories '***yes***' and ***'no'***, and the values do not have any order since its a nominal attribute. So we can directly map '***yes'*** to 1 and ***'no'*** to 0.

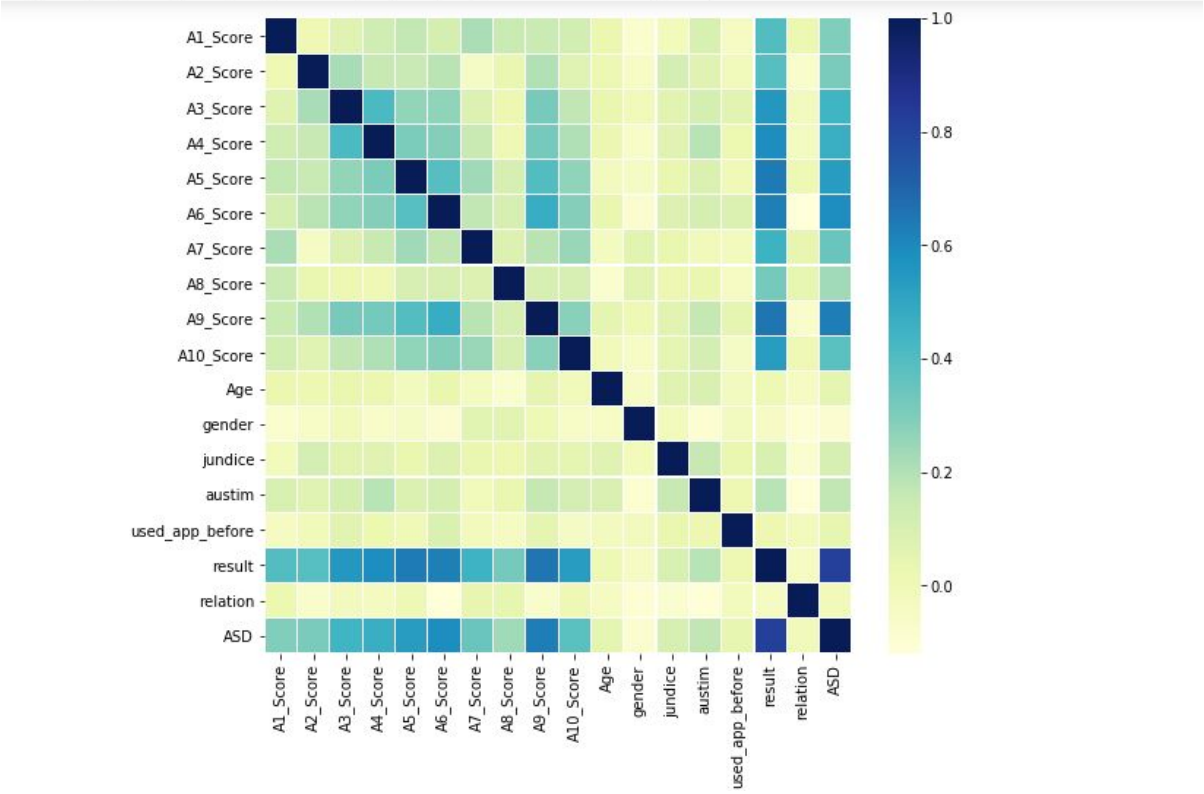


3. ***relation*** :

It is originally a column consisting of 5 unique categories. We assigned numeric labels to these categories by analyzing the relatedness between these columns.

We find the relation between all the numerical attributes using the Correlation Matrix made

using **seaborn** library



- We find out that class ***relation*** and ASD are very less related to each other, so we drop this attribute relation.
- We found the same analysis for attributes ***ethnicity*** and ***contry_of_res,*** so we will drop these too.
- The attribute age_desc is of no use as discussed earlier. So, it will be dropped too.

```
In [424]: #dropping age_desc because it is of no use
          data.drop(['age_desc'], axis=1, inplace=True)
          data.drop(['contry_of_res'], axis=1, inplace=True)
          data.drop(['ethnicity'], axis=1, inplace=True)
          data.drop(['relation'], axis=1, inplace=True)
```

Hence, after all these steps, we have converted all the categorical attributes to numerical attributes.

Following is the data type info of all attributes

```
In [432]: data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 699 entries, 0 to 703
Data columns (total 17 columns):
A1_Score            699 non-null int64
A2_Score            699 non-null int64
A3_Score            699 non-null int64
A4_Score            699 non-null int64
A5_Score            699 non-null int64
A6_Score            699 non-null int64
A7_Score            699 non-null int64
A8_Score            699 non-null int64
A9_Score            699 non-null int64
A10_Score           699 non-null int64
Age                 699 non-null float64
gender              699 non-null int64
jundice             699 non-null int64
austim              699 non-null int64
used_app_before     699 non-null int64
result              699 non-null int64
ASD                 699 non-null int64
dtypes: float64(1), int64(16)
memory usage: 98.3 KB
```

Feature Selection

Hence, it's important to select important features out of this dataset and drop the irrelevant ones. The reason for dropping some of the columns had already been mentioned previously, for example, ***"relation"*** is to be dropped.

MODEL TRAINING

Train Data & Test Data

The dataset available in the **UCI Machine Learning Repository** did not contain separate files for training and test data, so we have to do it manually.

Before splitting the dataset the independent and dependent variables should be separated because we will fit our classifier to the training data and use it to predict the dependent variable of test set.

We have randomly divided our data set using **train_test_split()** classifier method imported for ***sklearn*** model.

CODE:

```
In [441]: Y = data.ASD
          X = data.drop('ASD', axis=1)

In [442]: X.head()

Out[442]:
```

ore	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score	Age	gender	jundice	austim	used_app_before	result
1	1	1	1	0	0	1	1	0	0	26.0	0	0	0	0	6
1	1	0	1	0	0	0	1	0	1	24.0	1	0	1	0	5
1	1	0	1	1	0	1	1	1	1	27.0	1	1	1	0	8
1	1	0	1	0	0	1	1	0	1	35.0	0	0	1	0	6
1	0	0	0	0	0	0	1	0	0	40.0	0	0	0	0	2

```
In [444]: X_train, X_test, y_train, y_test = train_test_split(X, Y,test_size=0.2)

In [445]: X_train.shape

Out[445]: (559, 16)

In [447]: X_test.shape

Out[447]: (140, 16)
```

The line ***test_size=0.2*** suggests that the test data should be 20% of the dataset and the rest should be train data. With the outputs of the ***shape()*** functions, you can see that we have 559 rows in the test data and 140 in the training data.

The ratio of data in training and test data is kept 80:20 because we want to train our model with many inputs so that the prediction is done with higher accuracy.

Deciding Classifier: Logistic Regression Classifier

(A model is like a pair of goggles. It puts certain things into focus.)

We thought of choosing the Logistic Regression Algorithm for our Binary Classification Problem. However, we were in a great dilemma about choosing the appropriate algorithm for the type of data that we are handling. The type of data preprocessing techniques and analysis that we do on the data in-hand largely affects the type of Classification Algorithm that we choose for our Classification problem.

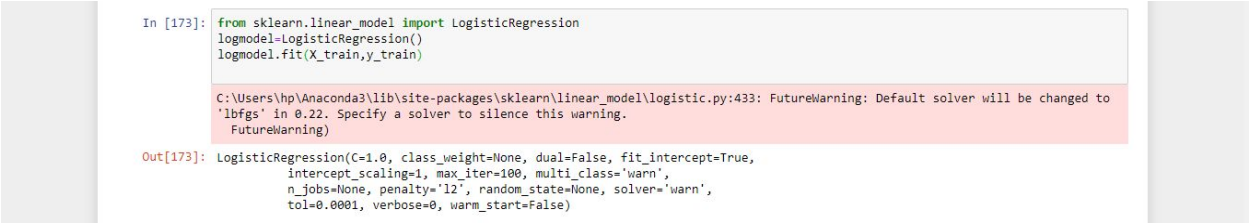
Keeping this thing in mind, choosing the appropriate classification algorithm becomes a crucial step in the entire Machine Learning Process.

So, by looking at the data set, right from the beginning, we had chosen Logistic Regression as an appropriate classification algorithm for our dataset. This decision was taken due to the following reasons:

- It is less prone to overfitting but it can overfit in high dimensional datasets
- It not only gives a measure of how relevant a predictor (coefficient size) is, but also its direction of association (positive or negative).
- It is easier to implement, interpret and very efficient to train.
- It does work better when you remove attributes that are unrelated to the output variable as well as attributes that are very similar (correlated) to each other.
- Because of its simplicity and the fact that it can be implemented relatively easy and quick.
- It is also a good baseline that you can use to measure the performance of other more complex Algorithms.

Training Classifier

The classifier ***logmodel*** built is trained using .fit() function. Scaled ***X_train*** and ***y_train*** are passed as arguments to train the Logistic Regression model



```
In [173]: from sklearn.linear_model import LogisticRegression
          logmodel=LogisticRegression()
          logmodel.fit(X_train,y_train)

          C:\Users\hp\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed to
          'lbfgs' in 0.22. Specify a solver to silence this warning.
            FutureWarning)

Out[173]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='warn',
                    n_jobs=None, penalty='l2', random_state=None, solver='warn',
                    tol=0.0001, verbose=0, warm_start=False)
```

Predictions

After training the model it is used to predict the ***X_test*** values of the test set and stored in a separate array ***predictions***

```
In [174]: predictions=logmodel.predict(X_test)

In [175]: predictions
Out[175]: array([1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0,
                 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
                 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0,
                 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0,
                 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0,
                 0, 0, 0, 1, 0, 0, 0, 0]], dtype=int64)
```

Evaluation methods:

1.Confusion Matrix

It is a table that describes the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

- **true positives (TP):** These are cases in which we predicted yes and they do have the ASD.
- **true negatives (TN):** We predicted no, and they don't have ASD.
- **false positives (FP):** We predicted yes, but they don't actually haveASD. (Also known as a "Type I error.")
- **false negatives (FN):** We predicted no, but they actually do have ASD. (Also known as a "Type II error.")

The score is to calculate the accuracy value between 0-1

```
In [176]: data1 = {'ASD_pred': predictions}
          df2 = pd.DataFrame(data=data1)

In [177]: from sklearn.metrics import confusion_matrix
          confusion_matrix(y_test,df2.ASD_pred)
Out[177]: array([[99,  1],
                 [ 2, 38]], dtype=int64)

In [179]: logmodel.score(X_test, y_test)
Out[179]: 0.9785714285714285
```

2.Accuracy Measures

Precision, Recall, the F1 score is calculated for the given prediction

```
In [183]: from sklearn.metrics import accuracy_score, roc_auc_score, classification_report

In [184]: print('\nLogistic Regression Accuracy: {:.2f}%'.format(accuracy_score(y_test, predictions) * 100))
          print('Logistic Regression AUC: {:.2f}%'.format(roc_auc_score(y_test,predictions ) * 100))
          print('Logistic Regression Classification report:\n\n', classification_report(y_test, predictions))

          Logistic Regression Accuracy: 97.86%
          Logistic Regression AUC: 97.00%
          Logistic Regression Classification report:

                        precision    recall  f1-score   support

                     0       0.98      0.99      0.99       100
                     1       0.97      0.95      0.96        40

              micro avg       0.98      0.98      0.98       140
              macro avg       0.98      0.97      0.97       140
           weighted avg       0.98      0.98      0.98       140
```

OBSERVATIONS

We get an accuracy rate of **97.86 %,** which is a considerate value.