Optimizing Autism Prediction Models: A Comprehensive Approach to Boost accuracy and Outperform existing application result

DA5030

Shalini_Karthyk

FALL2023

Autism Screening on Adults

Autism, often known as autism spectrum disorder (ASD), is a group of developmental disorders characterized by difficulties with social skills, repetitive behaviors, speech, and nonverbal communication.

Adapting the CRISP-DM method and with the help of Data Minning and Machine Learning and trailoring the algorithms to **Predict the likelihood of having autism to aid in early intervention prioritization_**.Early intervention is crucial for individuals with autism and predicting the likelihood can help prioritize resources for those who may need intervention. Here we are trying to develop a classification algorithm to predict autism based on the survey response and other variable information to outperform the application's current **prediction accuracy**.

Libraries Required

```r
# Common packages for efficient coding
library(dplyr)
library(tidyr)
library(ggplot2)

# Correlation and chi-squared
library(corrplot)
library(MASS)

# Association rules
library("arules")
library("arulesViz")
library(stringr)

# Sampling
library(caret)

# Randomforest
library("randomForest")

# svm
library(e1071)
```

```
# bagging
library(ipred)
```

1 Data Acquisition

The dataset is collected from the kaggle and it is from survey responses from an app form. [**Autism-Screening]https://www.kaggle.com/datasets/andrewmvd/autism-screening-on-adults**.With this dataset we are trying to understand the structure, explore features, analyse with with multiple machine learning technique and to check if we can outperform the current application result.

```
# Data is in archive form
zip_name <- "archive.zip"
csv_name <- "autism_screening.csv"

# dataset archive link
screen_link <- "https://www.kaggle.com/datasets/andrewmvd
/autism-screening-on-adults/download?datasetVersionNumber=1"

# data extraction and unzip to get csv
data_unzip <- unz(zip_name, csv_name)
autism_data <- read.csv(data_unzip, header = T, stringsAsFactors = F)
```

This dataset used in autism screening in adults, capturing various demographic information, answers to specific questions, and the classification result regarding autism. Each row represents an individual, used for exploring patterns and relationships between different features and the likelihood of being classified with autism.

The brief description of each columns are given below:

A1_Score to A10_Score : Scores marked as 1 or 0 based on the response (1 indicates Autism positive)

age: Age of the individual.

gender: Gender of the individual.

ethnicity: Ethnicity of the individual.

jundice: Presence of a yellowish tinge to the skin and the whites of the eye caused by a buildup of bilirubin.

austim: Whether the person was diagnosed with autism or not (by physician).

contry_of_res: Country of residence.

used_app_before: Whether the person used the app before.

result: Sum of scores.

age_desc: Age description.

relation: Who made the answers.

Class/ASD: Whether the app classified them with autism or not.

2 Data Exploration

```r
# Inspecting elements of the CSV
# Dimension if the data
dimensions <- dim(autism_data)

# Str of the data
str_data <- str(autism_data)

# printing first 5 lines of the data
head_data <- head(autism_data)

# Columns list
cols_data <- colnames(autism_data)

# Summary of the dataset
summary_data <- summary(autism_data)

# Changing unknown("?") to NA's in the dataset
autism_data <- autism_data %>%
  mutate_all(~ifelse(. == "?", NA, .))

# Changing the case sensitive "others" to Others in ethinicity column
autism_data["ethnicity"] <- autism_data %>% dplyr::select(ethnicity) %>%
  mutate_all(~ifelse(. == "others", "Others", .))

# Changing the case sensitive yes no in Class.ASD similar to autism target
autism_data["Class.ASD"] <- autism_data %>% dplyr::select(Class.ASD) %>%
  mutate_all(~ifelse(. == "YES", "yes", "no"))

# na check
missing_nas <- apply(is.na(autism_data),2,sum)

# Changing the spell error in autism column
colnames(autism_data)[15] <- "autism"

# checking class info from the dataset
class_info <- sapply(autism_data, class)

info_table <- knitr::kable(data.frame(cbind(class_info, missing_nas)))
```

The dataset has **704** rows and **21** columns. It has total of **192** and the information table shows the column list with their respective class and number of the missing values for each column.

Information Table

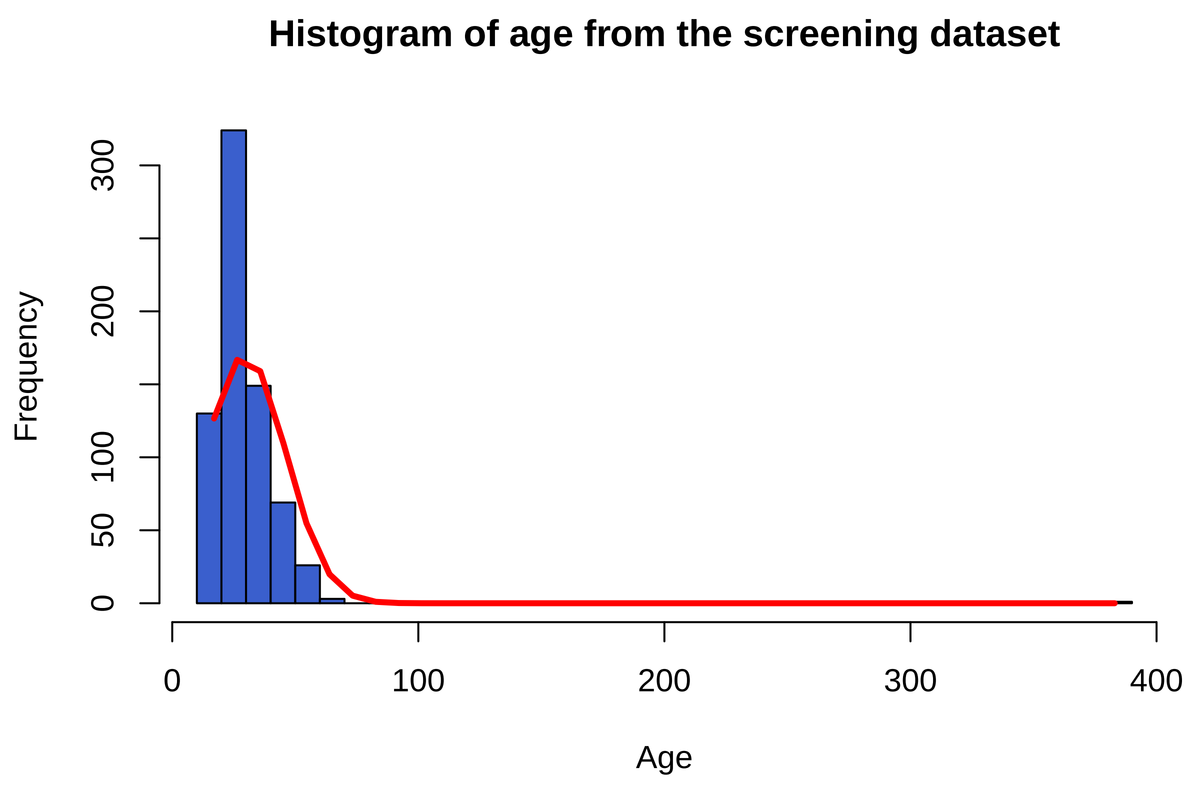
	class_info	missing_nas
A1_Score	integer	0
A2_Score	integer	0
A3_Score	integer	0
A4_Score	integer	0
A5_Score	integer	0
A6_Score	integer	0
A7_Score	integer	0
A8_Score	integer	0

	class_info	missing_nas
A9_Score	integer	0
A10_Score	integer	0
age	numeric	2
gender	character	0
ethnicity	character	95
jundice	character	0
autism	character	0
contry_of_res	character	0
used_app_before	character	0
result	numeric	0
age_desc	character	0
relation	character	95
Class.ASD	character	0

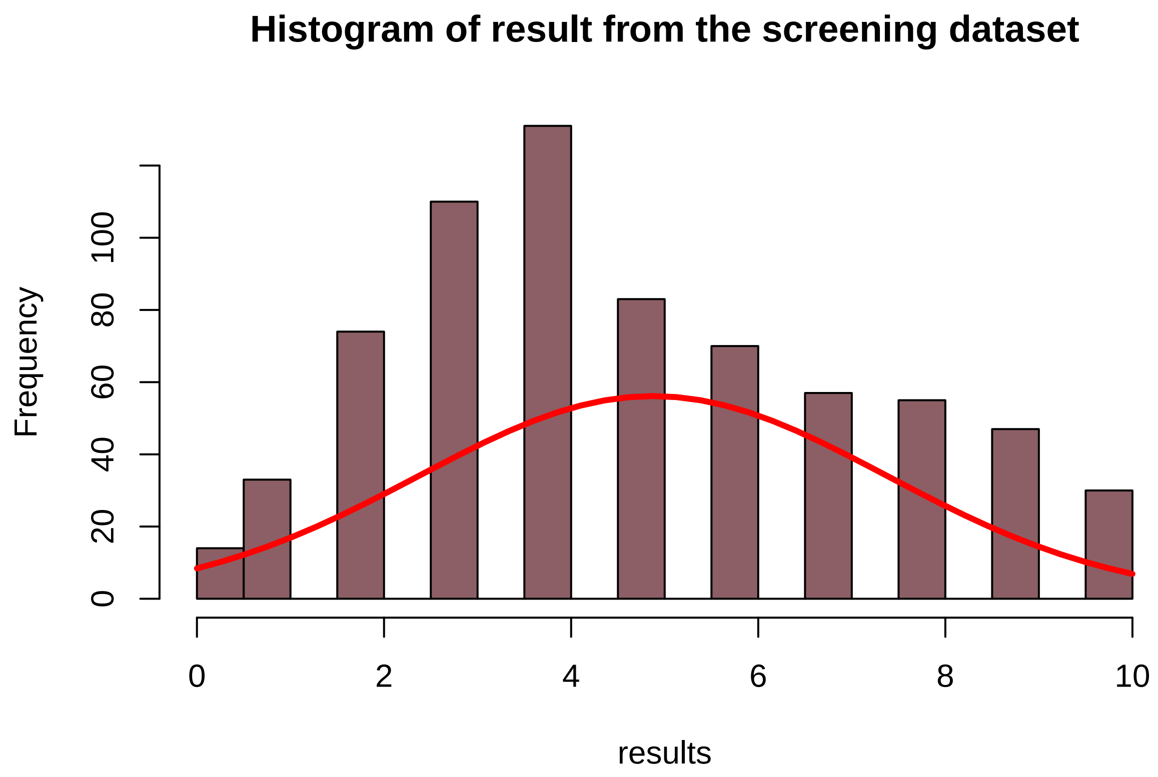
2.1 Exploratory dataplots

There are many exploratory dataplots that can be used to understand this data set, since we haven't removed the outliers for the continuous variable there might be some off-distribution in the plot which we will fix later. Also some plots including the gender ratio and the autism frequency will help us to understand the data better and to address the class imbalance issues

```r
# Histogram with the distribution line for age
h<-hist(autism_data$age, breaks=30, col="royalblue3",
        xlab="Age",
   main="Histogram of age from the screening dataset")
xfit<-seq(min(autism_data$age, na.rm = TRUE)
          ,max(autism_data$age, na.rm = TRUE),length=40)
yfit<-dnorm(xfit,mean=mean(autism_data$age, na.rm =TRUE),
            sd=sd(autism_data$age, na.rm = TRUE))
yfit <- yfit*diff(h$mids[1:2])*length(autism_data$age)
lines(xfit, yfit, col="red", lwd=3)
```

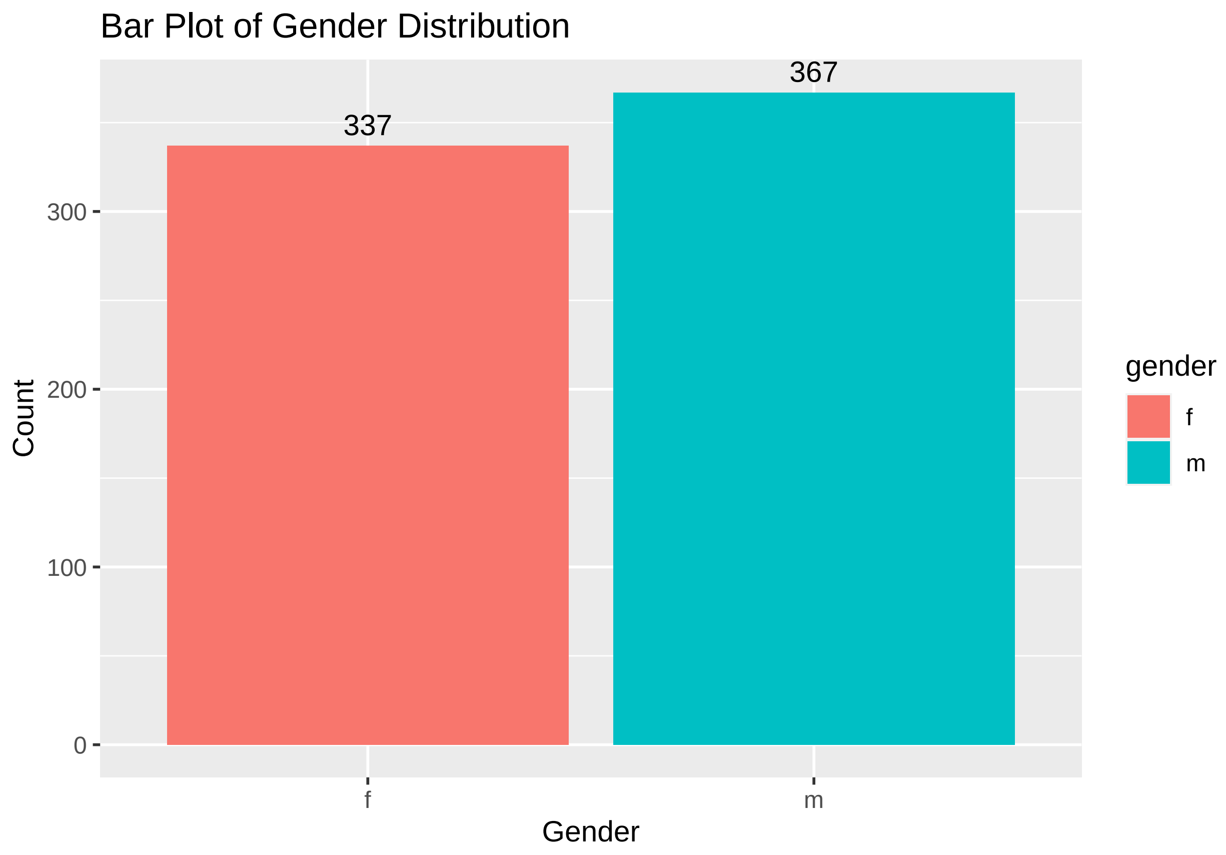


```
# Histogram with the distribution line for result
h<-hist(autism_data$result, breaks=30, col="lightpink4",
        xlab="results",
   main="Histogram of result from the screening dataset")
xfit<-seq(min(autism_data$result, na.rm = TRUE)
          ,max(autism_data$result, na.rm = TRUE),length=40)
yfit<-dnorm(xfit,mean=mean(autism_data$result, na.rm =TRUE),
            sd=sd(autism_data$result, na.rm = TRUE))
yfit <- yfit*diff(h$mids[1:2])*length(autism_data$result)
lines(xfit, yfit, col="red", lwd=3)
```



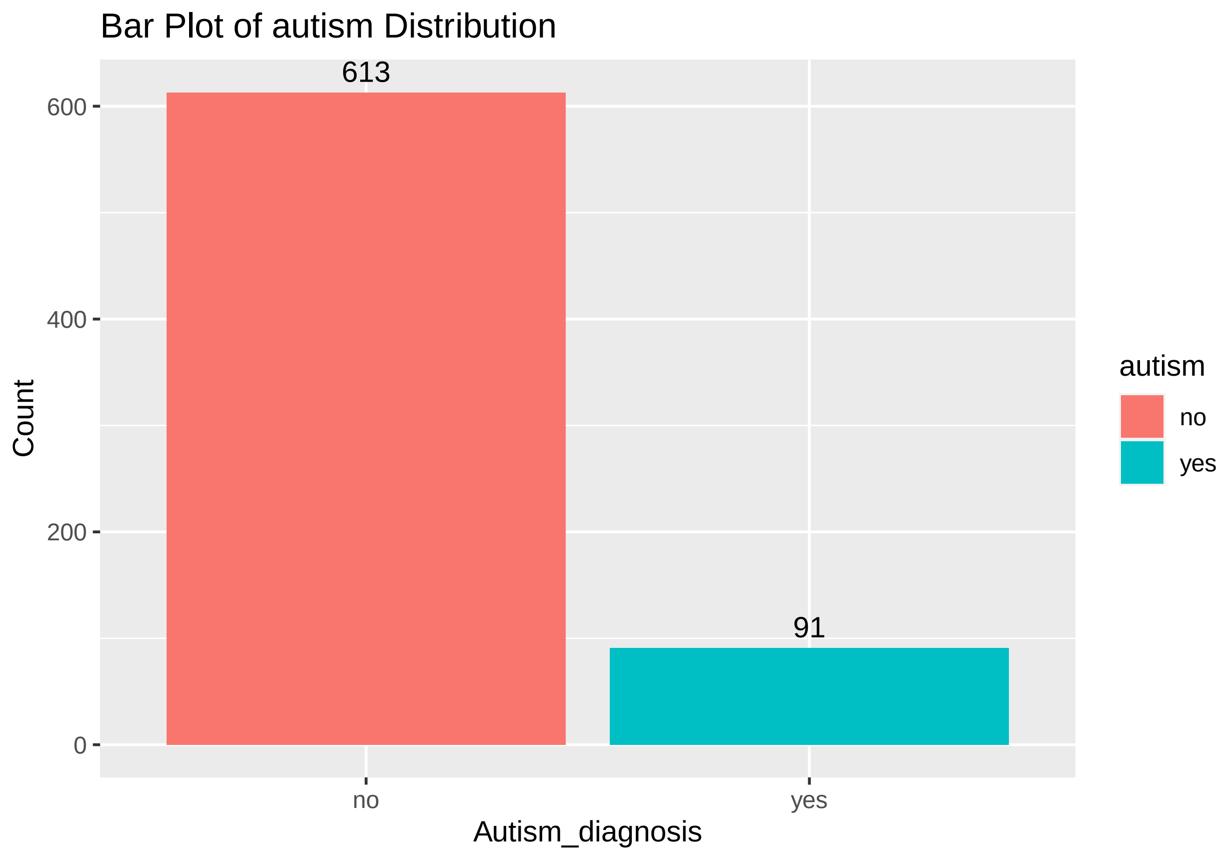
This plot helps us to understand the distribution of ages and result in our dataset. Knowing the age distribution is important because certain age groups may have different prevalence rates of autism. This histogram here clearly shows that there are some outliers that has to be handled proceeding the analysis. The plot also shows right skewed data that is the peak of the graph lies to the left side of the center which can be resolved with the normalization. As for the result it shows normal diatribution though we will be doing feture engineering on the result column we can sill see that its normally distributed for the data understanding purpose.

```r
# Bar plot for Gender Distribution
ggplot(autism_data, aes(x = gender, fill = gender)) +
  geom_bar() +
  labs(title = "Bar Plot of Gender Distribution", x = "Gender", y = "Count") +
  geom_text(stat = "count", aes(label = after_stat(count)), vjust = -0.5)
```



Understanding the gender distribution is important for considering potential gender-related factors in autism. It will also let us know if there are any bias over the survey data as we might need to decide of we should include the gender as one of the factor for modelling. With the above plot show there are **337** female and **367**, we can clearly see that there are not much difference in gender and number of observations has almost equal percentage of the gender distribution so model will be exposed to a similar number of instances from each class, preventing potential biases.

```
# Bar plot for autism(target variable)
ggplot(autism_data, aes(x = autism, fill = autism)) +
  geom_bar() +
  labs(title = "Bar Plot of autism Distribution",
       x = "Autism_diagnosis", y = "Count") +
  geom_text(stat = "count", aes(label = after_stat(count)), vjust = -0.5)
```



The plot shows the total number of **91** are predicted as positive for the diagnosis of autism and **613** diagnosed as no. Since the data is imbalanced and there is more possibilities are presented for the the negative diagnosis. This might be a issue when the new cases are predicted to land in the minority "yes" class.

2.2 Detection of outliers for continuous features

```
# Outliers detection and removal in continuous variable (age)
# calculating total number of unique outliers in age
z_scores_cols <- abs(autism_data$age-
                       mean(autism_data$age, na.rm = TRUE)
                     )/sd(autism_data$age, na.rm = TRUE)

nrow_outlier <- numeric(0)

# index of the outliers
nrow_outlier <- c(nrow_outlier, unlist(which(z_scores_cols > 3)))

# Percentage of the outliers.
len_outliers <- length(unique(nrow_outlier))
percent_out <- (len_outliers/ nrow(autism_data)) * 100

# Removing outliers using the index
autism_no_outlier <- autism_data[-nrow_outlier, ]
```

We are handling outliers in the "age" variable as can be considered relevant to the prediction of autism, as there could be age-related patterns or factors that influence the likelihood of an autism diagnosis. The total outliers present in the dataset are **1** which covers **0.1420455** percentage in the total number of observations. Since it is very less we are removing the oulier observation from the dataset.

Here we can also see that results is also a continuous column but the outliers are not checked for the result variable as "result" is a total sum of binary scores from other variables (A1_Score to A10_Score) nd it is always between 1 to 10, including both individual scores and the total sum might introduce redundancy and potential multicollinearity issues in the analysis. An we will be introducing weighted scores in future to avoid give more appropriate analysis for the binary scores.

2.3 Correlation/ Collinearity/ chi-squared test

Exploring the relationships between variables helps us to understand the dataset's structure. Correlation analysis can reveal if there's any linear relationship between numerical variables, potentially indicating patterns or trends, If there are any association for the categorical variables.

```r
# Taking age and result for correlation check
numeric_varibles <- data.frame(autism_no_outlier$age, autism_no_outlier$result)
# Correlation between numeric variables age and result
cor_matrix <- cor(na.omit(numeric_varibles))
```

The correlation between the age and result give a good insight on the data if the different age will affect the results(sum of scores). Understanding the correlation between age and result can be informative about how age might influence the overall result score, which could be relevant for feature engineering or interpretation. Here the correlation between age and result is **0.0977992** , very close to 0 showing there is a weak corelation between the these to variables. That is Increasing or decreasing one of them will not affect the overall result.

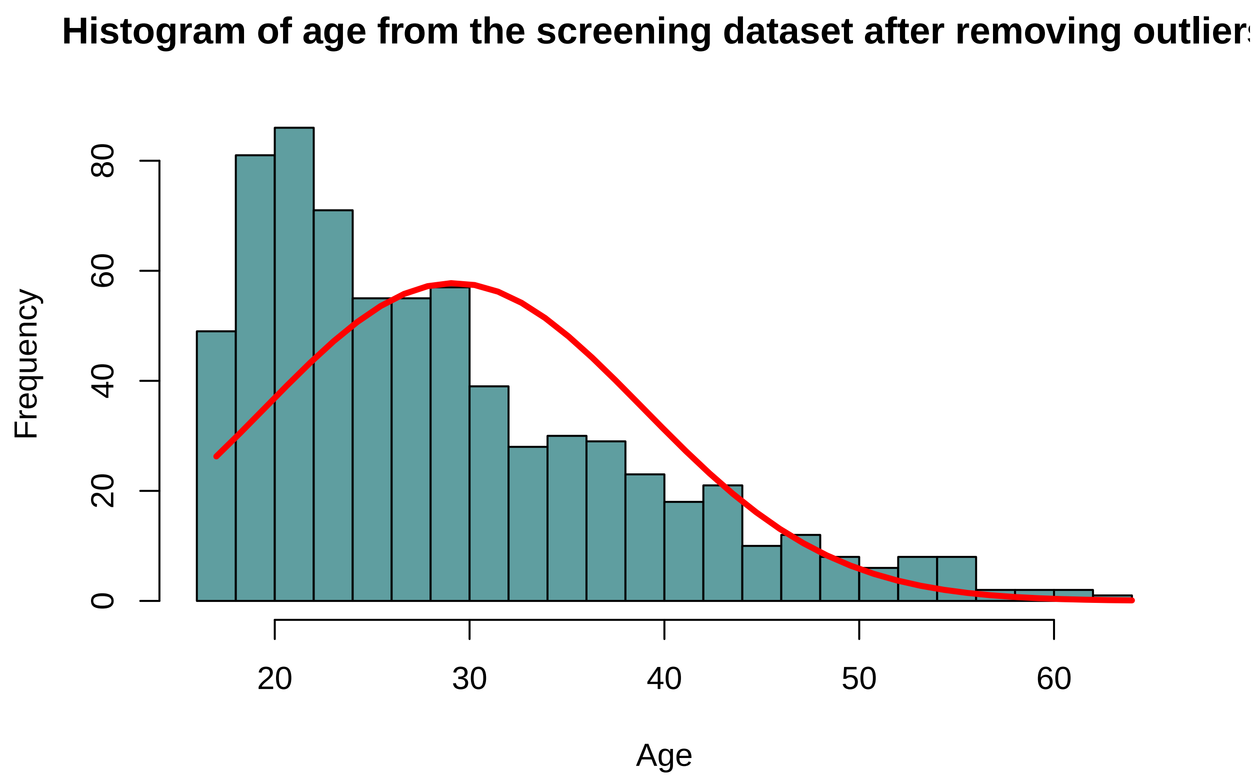
```r
# No na variables
no_NA_variables <- na.omit(autism_no_outlier)

# Categorical variables association with chi_squared
categorical_variables <- c("gender", "ethnicity", "jundice",
                           "contry_of_res", "used_app_before", "relation")

# List to store pvalues
chi_squared_results <- list()

for (variable in categorical_variables) {
  suppressWarnings(chi_squared_test_result <- chisq.test(
    no_NA_variables[,variable], no_NA_variables$autism))

  # Storing the pvalues
  chi_squared_results[variable] <- chi_squared_test_result$p.value
}
```

The chisq results shows **gender, ethnicity, jundice, contry__of__res** all these variables have pvalues less than 0.05 showing significant association between the variables and the target autism. The **used__app__before, relation** are above the 0.05, the variable used__apps__before has pvalue of **0.9252972** and is no significant association between whether the person used an app before and the likelihood of having autism. But the variable relation, pvalue of **0.0586583** has slightly above the 0.05 and shows borderline association and can you used in our model.

2.4 Evaluation of distribution

As we have removed the outlier in the age we can now evaluate the distribution and the histogram for the type of distribution in the variable.

```
# Shapiro-Wilk Test for Age
sw_age <- shapiro.test(autism_no_outlier$age)

# Histogram with the distribution line for age
h<-hist(autism_no_outlier$age, breaks=30, col="cadetblue",
        xlab="Age",
   main="Histogram of age from the screening dataset after removing outliers")
xfit<-seq(min(autism_no_outlier$age, na.rm = TRUE)
          ,max(autism_no_outlier$age, na.rm = TRUE),length=40)
yfit<-dnorm(xfit,mean=mean(autism_no_outlier$age, na.rm =TRUE),
            sd=sd(autism_no_outlier$age, na.rm = TRUE))
yfit <- yfit*diff(h$mids[1:2])*length(autism_no_outlier$age)
lines(xfit, yfit, col="red", lwd=3)
```



To evaluate the distribution of age after removing the outliers, running the shapiro wilk test to check if the datset is normally distributed. The pvalue from the test is $2.6617025 \times 10^{-20}$ less the 0.05 show the age is not normally distributed. For the visual representation of the distribution we are using histogram which shows the age range between **17** and **64** with right skewed data after the ouliters been removed.

3 Data Cleaning and Shaping

The Data Cleaning and Shaping includes the missing value identification, imputation, normalization transformation and feature engineering, We are also going to add association mining to some feature which can help us to improve the model by giving weightage to the binary scores based on their association with the target variable.

3.1 Identification of missing values

```
# Identify missing values
missing_values <- is.na(autism_no_outlier)

# Column count for missing values
missing_col_count <- colSums(missing_values)

# columns with the missing values
missing_cols <- missing_col_count[missing_col_count > 0]
```

The column that has the missing values are **age, ethnicity, relation**. The column age has total of **2**, ethnicity has **95** and relation has **95** missing columns.

3.2 Data imputation and handling missing data

```
# Evaluating the distribution of relation and ethinicity
# Frequency table for relation
relation_table <-  table(autism_no_outlier$relation)

# Frequency table for ethinicity
ethnicity_table <-  table(autism_no_outlier$ethnicity)

# Missing values in age
# imputing missing values with median
autism_no_outlier$age[is.na(autism_no_outlier$age)] <-
  median(autism_no_outlier$age, na.rm = TRUE)

# Missing values in ethinicity and relation
# imputing missing values with mode
mode <- function(col) {
  col <- na.omit(col)
  uniq <- unique(col)
  match_tab <- tabulate(match(col, uniq))
  uniq[match_tab == max(match_tab)]
}


clean_autism_data <- autism_no_outlier %>%
  mutate(
    relation = if_else(is.na(relation), mode(autism_no_outlier$relation), relation),
    ethnicity = if_else(is.na(ethnicity), mode(autism_no_outlier$ethnicity), ethnicity)
  )
```

```
# Evaluating the distribution of relation and ethinicity after imputation
# Frequency table for relation
relation_table <-  table(clean_autism_data$relation)

# Frequency table for ethinicity
ethinicity_table <-  table(clean_autism_data$ethnicity)
```

The number of missing values in the age are handle by imputing it with the median of the age. Imputation by mean can potentially reduce data variability and introduce bias. To avoid this and to maintain the central tendency of the data, imputation by median is used for the provided data. For the Categorical variables like relation and ethnicity mode function is used to impute and this helps in maintaining the distribution of the data.

3.3 Feature engineering and newly derived features

We are opting to do the feature engineering and by removing, adding and manipulating features at this stage before proceeding with the normalization and transformation of the features to get the good model.

```
# Removing the age_desc feature with 1 level observation
clean_autism_data <- clean_autism_data  %>% dplyr::select(-age_desc)

# Removing the result feature
# Since the column is just the sum of scores
clean_autism_data <- clean_autism_data  %>% dplyr::select(-result)

# Removing the used_apps_before feature
# Found it has no association with the autism after chisq test
clean_autism_data <- clean_autism_data  %>% dplyr::select(-used_app_before)

# Removing the Class.ASD feature
# Will be using it to compare the accuracy with the whole dataset
clean_autism_data <- clean_autism_data  %>% dplyr::select(-Class.ASD)
```

The **"age__desc"** feature is removed because it may not provide valuable information for predicting autism, especially if it has only one level of observation. Variables with no variability don't contribute to the predictive power of the model. The **"result"** feature is removed because it is a sum of scores from other features, and when modeling, it might introduce multicollinearity issues (high correlation with individual scores). Removing it can help avoid redundancy and potential model instability. The decision to remove **"used__app__before"** is based on a prior chi-squared test, indicating that this feature doesn't show a significant association with the target variable (autism). If a variable doesn't contribute significantly to predicting the target, removing it can simplify the model and to improve predictive accuracy. We will be removing the **"Class.ASD"** from the model as these are the predicted outputs from the app which we will comparing and expecting to outperform.

```
# Too-many factors in countries
# Function to map countries to regions

region_map <- function(country)
  {
  # Approximate mapping with the overall region
  region_list <- list(
    Africa = c("Burundi", "South Africa", "Sierra Leone", "Ethiopia",
                "Niger", "Angola"),
    Asia = c("Jordan", "United Arab Emirates", "Afghanistan", "Lebanon",
             "Saudi Arabia", "Pakistan", "Bangladesh", "China", "India",
             "Philippines", "Sri Lanka", "Viet Nam", "Iran", "Japan",
             "Malaysia", "Hong Kong", "Kazakhstan", "American Samoa",
             "Armenia", "Turkey", "Nepal", "Indonesia", "Azerbaijan", "Iraq",
             "Cyprus"),
    Europe = c("Spain", "Austria", "Ireland", "United Kingdom",
               "Italy", "Romania", "Sweden", "Netherlands", "France",
               "Germany", "Russia", "Iceland", "Belgium", "Finland",
               "Czech Republic", "Portugal"),
    North_America = c("United States", "Canada", "Mexico"),
    South_America = c("Brazil", "Argentina", "Chile", "Costa Rica",
                      "Ecuador", "Uruguay", "Bolivia", "Aruba"),
    Oceania = c("New Zealand", "Australia", "Tonga"),
    Others = c("Egypt", "Bahamas", "Oman", "Nicaragua")
  )

  # Map each country
  for (region in names(region_list))
    {
    if (country %in% region_list[[region]]) {
      return(region)
    }
  }

  # If country is not found in mappings, return "Others"
  return("Others")
}

# Apply the function to create a new variable 'region'
clean_autism_data["contry_of_res"]<- sapply(clean_autism_data$contry_of_res, region_map)
```

3.4 Association Mining for binary features

The features from A1_score to A10_score are the scores that are given by the survey in the app to measure the social and cognitive trait. All the questions have equal weightage (1,0) in the dataset, one being likely measure for the autism. The following are the question that are used to get the survey from the app.

A1_Score: I often notice small sounds when others do not.

A2_Score: I usually concentrate more on the whole picture, rather than the small details.

A3_Score: I find it easy to do more than one thing at once.

A4_Score: If there is an interruption, I can switch back to what I was doing very quickly.

A5_Score: I find it easy to ‘read between the lines’ when someone is talking to me.

A6_Score: I know how to tell if someone listening to me is getting bored.

A7_Score: When I’m reading a story, I find it difficult to work out the characters’ intentions.

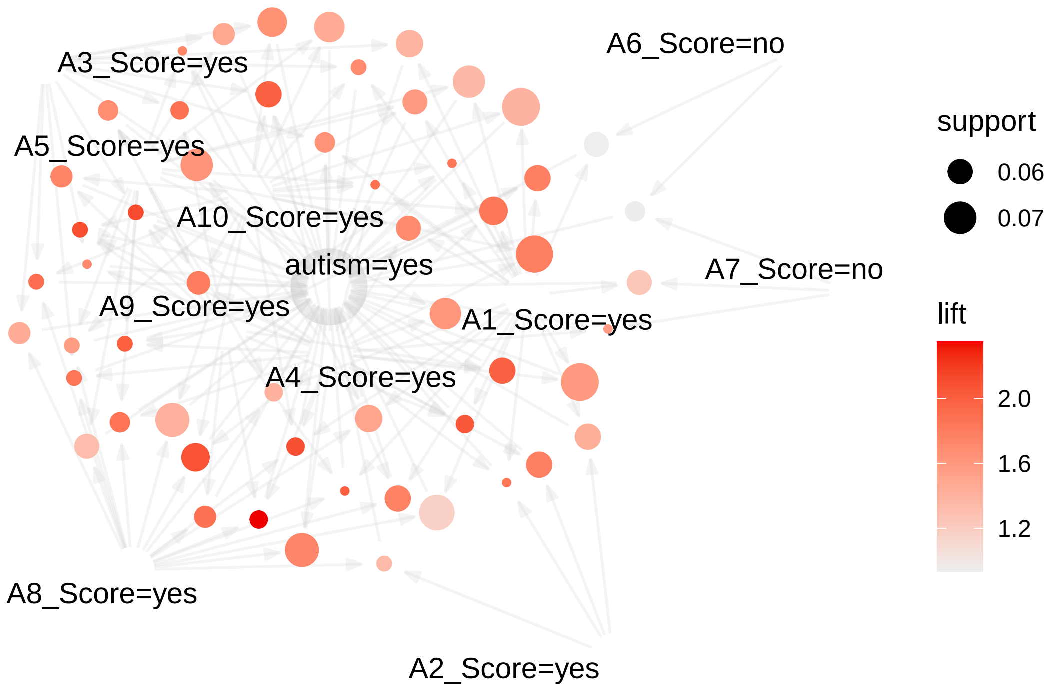
A8_Score: I like to collect information about categories of things (e.g., types of car, types of bird, types of train, types of plant, etc.).

A9_Score: I find it easy to work out what someone is thinking or feeling just by looking at their face.

A10_Score: I find it difficult to work out people's intentions.

SCORING on the dataset : Score 1 point for Definitely or Slightly Agree on each of items 1, 7, 8, and 10. Score 1 point for Definitely or Slightly Disagree on each of items 2, 3, 4, 5, 6, and 9. If the individual scores 6 or above, consider referring them for a specialist diagnostic assessment.

```r
# Taking all the binary data and autism target varible
ar_rule_cols <- clean_autism_data[, c(1:10, 15)]

# Converting all binaries to yes or no and changing the table to transactions
ar_rule_cols[1:10]<- ifelse(ar_rule_cols[1:10] == 1, "yes", "no")
suppressWarnings(transactions <- as(ar_rule_cols, "transactions"))

# Assocuation rules
rules <- apriori(transactions,
                             parameter =
                   list(support = 0.05, confidence = 0.10, minlen=3),
                 appearance =
                   list(rhs=c("autism=no", "autism=yes"),default="lhs"),
                             control = list(verbose = TRUE))

filtered_rules <- subset(rules, rhs %in% "autism=yes")

# Sort the rules by coverage
rules.sorted <- sort(filtered_rules, by = "coverage")

# Plot the rules for better understanding of the interactions
plot(rules.sorted, method="graph", control=list(type="items"))
```



We are selecting the columns used to derive the association rule, selected columns include binary data related to answers (A1_Score to A10_Score) and the target variable (autism). The entire dataset is transformed into a transactions format and Apriori algorithm is applied to mine association rules from the transactions.

Rules are filtered to include only those with **'autism=yes'** in the RHS. The filtered rules are sorted based on coverage the total rules are **55** . Coverage is preioritized as it is the proportion of instances in the dataset with the rhs (autism=yes). it helps to analyse on rules that cover a larger portion of your data and eventually helping the model, and a graph is plotted for better visualization. This plot indicated the support with the size of the circle and the lift with the brightness. Based on this plot we can check most relatable and frequest support like A1_score and A5 score etc and A7 and A10 have very low lift.

```
# Creating an extract function to get list of all elements
extract_rule<- function(rule) {
  lhs <- labels(lhs(rule))
  lhs <- unlist(str_extract_all(lhs, "\\b\\w+\\b"))
  if(any(sapply("no", grepl, lhs)) == FALSE)
  {
    return(lhs)
  }
  else{
    return("")
  }
}

# Empty character list to keep the unique rule elements
all_rules_vars <- character(0)
```

```r
# Loop to get the unlised rule variables
for( i in 1:20)
{
  all_rules_vars <- c(all_rules_vars, extract_rule(rules.sorted[i]))
}

# Removing all "yes" representations and empty strings
all_rules_vars <- paste0(all_rules_vars [!all_rules_vars  %in% c("yes","")])
all_rules_vars <- sort(table(all_rules_vars),decreasing=F)

# Assigning weights to the majority rule elements
weights <- rep(2, length(all_rules_vars))
weight_data <- data.frame(coln = names(all_rules_vars))
weight_data <- weight_data %>% mutate(all_rules_vars, weights)

# loop to add the weights to the columns
for(i in 2:nrow(weight_data))
{
  if(weight_data[i, "all_rules_vars"] > weight_data[i-1, "all_rules_vars"])
  {
    weight_data[i, "weights"]  <-  weight_data[i-1, "weights"] + 1
  }
  else
  {
    weight_data[i, "weights"] <- weight_data[i-1, "weights"]
  }
}

Weight_data_table <- knitr::kable(data.frame
                                  (columns = weight_data$coln, weights =
                                      weight_data$weights))
```

Unique rule elements are collected, and weights are assigned based on their frequency. The following tables shows the rule elements with their respective weight.

Weight_data_table

columns	weights
A2_Score	2
A3_Score	3
A5_Score	3
A4_Score	4
A10_Score	5
A1_Score	6
A8_Score	6

```r
# Creating new Variable to have a backup clean data
ar_clean_data <- clean_autism_data

# changing weights to the dataset
for(i in 1:nrow(weight_data))
{
```

```
ar_clean_data[weight_data$coln[i]] <- ar_clean_data %>%
  dplyr::select(col = weight_data$coln[i]) %>%
  mutate(col = ifelse(col == 1, weight_data$weights[i], 0))
}
```

Weights are applied to the original dataset based on the assigned weights for each column, after using these rules in model we can adjust the number of rules and criteria to filter based on the model performance.

3.5 Normalization and Standardization of data

```
# Normalizing the data to maintain data in one scale
normalized <- ar_clean_data %>% dplyr::select_if(is.numeric) %>%
  scale(center = TRUE, scale = TRUE)
normalized <- data.frame(normalized)
```

All the numeric columns in the datset are normalized with with scale and centre. These values helps us to get the the standardized values without affecting range and difference of one observation to other.

```
# Testing for the normality before transformation
# Shapiro-Wilk normality check
no_norm_col <- c()
for (c in 1:ncol(normalized))
  { #repeat for all columns
  sw_test <- shapiro.test(normalized[,c])
  #test for normality
  if (sw_test[["p.value"]]<= 0.05)
    {
    no_norm_col[length(no_norm_col)+1] <- c
    }
  }

# Printing all the no normal columns
no_normal_names <- (colnames(normalized[no_norm_col]))


# checking if all columns are not normally distributed
nor_check <- ifelse((ncol(normalized) == length(no_norm_col)),
                    "All the columns are not normally distributed",
                    "some columns are not normally distribute" )
```

The Normalized data is checked for the normality before the transformation steps to know if there are any variables that are already in normal distribution. The result from the normality check is **All the columns are not normally distributed** .

3.6 Tranformation of features to adjust distribution

```
# Checkinf if there are any 0's in the columns
zero <- colSums(normalized==0)
support_constant <- abs(min(normalized)) + 1
```

```r
Transform <- normalized
# log transformation
log_normality <- numeric()
Transform <- log(normalized + support_constant)
log_normality <- c(log_normality, lapply(Transform,
                                         function(x) shapiro.test(x)$p.value))

Transform <- normalized
# Inverse transformation
inverse_normality <- numeric()
Transform <- 1/(normalized + support_constant)
inverse_normality <- c(inverse_normality, lapply(Transform,
                                         function(x) shapiro.test(x)$p.value))

Transform <- normalized
# Square root transformation
sq_normality <- numeric()
Transform <- sqrt(normalized + support_constant)
sq_normality <- c(sq_normality, lapply(Transform,
                                         function(x) shapiro.test(x)$p.value))

# Check all the methods for pvalue > 0.05
max_val <- max(c(sum(log_normality > 0.05), sum(inverse_normality > 0.05)
      , sum(sq_normality > 0.05)))

# Checking highest pvalue method
max_method <- which.max(c(max(unlist(log_normality)),
                          max(unlist(inverse_normality)),
                          max((unlist(sq_normality)))))


Transform <- normalized
# Choosing inverse tranformation to do the futher analysis
# Inverse transformation
inverse_normality <- numeric()
Transform <- 1/(normalized + support_constant)
inverse_normality <- c(inverse_normality, lapply(Transform,
                                         function(x) shapiro.test(x)$p.value))
```

After finding that none of the columns in the dataset follow a normal distribution, we are attempting to normalize them using log, inverse, and square root transformation methods. The selection of the best method involves two criteria and We use the Shapiro-Wilk test to obtain the p-value to check the normality. First, we consider the number of columns that become normally distributed after transformation and has p_value greater than 0.05.

However, since none of the columns meet this criterion in any of the methods, we proceed with the second option. This involves identifying the best transformation by selecting the method with the highest p-value, (considering they are most normalized) for the practicum purpose. In this case the method **2** , the inverse method, yielded the highest p-value, so we have decided to perform data transformation using this method.

3.7 Dummy Coding and adding all categorical features to the dataset

Moving froward we will be using our **original data** and the **scaled and Transformed data** , both to check in the model and whichever has highest accuracy and F1 score can be used for future predictions

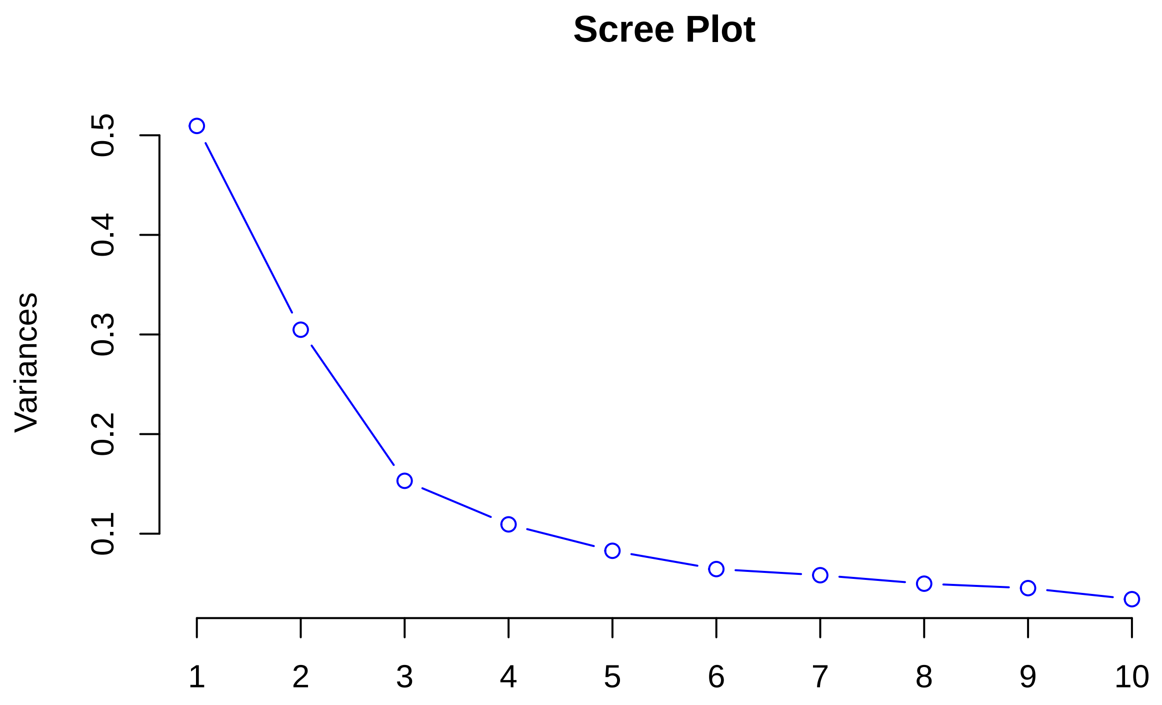
```
# Before Adding dummy coding to the dataset
# Adding all the categorical variables to the transformed data
Transform <- cbind(Transform, gender = clean_autism_data$gender,
                   ethnicity = clean_autism_data$ethnicity,
                   jundice = clean_autism_data$jundice,
                   autism = clean_autism_data$autism,
                   contry_of_res = clean_autism_data$contry_of_res,
                   relation = clean_autism_data$relation)


# All categorical variables can be dummy coded
# Picking gender and ethinicity to dummy code
# duplicating gender to encode
encoded_transform_data <- Transform

# Creating 2 new columns
encoded_transform_data$f <- 0
encoded_transform_data$m <- 0

# adding binary numbers by matching the columns
for (i in 1:nrow(encoded_transform_data)) {
  gen <- encoded_transform_data[i, "gender"]
  encoded_transform_data[i, gen] <- 1
}

# Removing gender column
encoded_transform_data <- encoded_transform_data %>% dplyr::select(-"gender")


# Encoding Ethnicity
# Creating 10 new columns
encoded_transform_data$Asian <- 0
encoded_transform_data$Black <- 0
encoded_transform_data$Hispanic <- 0
encoded_transform_data$Latino <- 0
encoded_transform_data$Middle_Eastern <- 0
encoded_transform_data$Others <- 0
encoded_transform_data$Pasifika <- 0
encoded_transform_data$South_Asian <- 0
encoded_transform_data$Turkish <- 0
encoded_transform_data$White_European <- 0

# adding binary numbers by matching the columns
for (i in 1:nrow(encoded_transform_data))
  {
  gen <- encoded_transform_data[i, "ethnicity"]

  if (gen == "Middle Eastern "){
    encoded_transform_data[i, "Middle_Eastern"] <- 1
  }
  else if (gen == "South Asian"){
    encoded_transform_data[i, "South_Asian"] <- 1
  }
  else if (gen == "White-European"){
    encoded_transform_data[i, "White_European"] <- 1
  }
  else {
    encoded_transform_data[i, gen] <- 1
  }
}

# Removing ethnicity column
encoded_transform_data <- encoded_transform_data %>% dplyr::select(-"ethnicity")
```

Selecting 2 variables to dummy coding (gender and ethnicity). The ethnicity has the lowest chisq value and has the strongest association with the target variable. The gender also has strong association with the target autism variable and also based on this analysis there might evidence to suggest that autism may present differently in males and females. By dummy coding gender, the analysis can capture potential gender-specific patterns in autism characteristics. Though our aim is to outperform the current application algorithm, giving gender in dummy coding might be a good training for the model and could welcome some interesting patterns.

There are total of **2** elements in gender (Male, Female) and **10** in ethnicity (Asian, Black, Hispanic, Latino, Middle Eastern, Others, Pasifika, South Asian, Turkish, White-European). We created dummy code for all of them which increased the size of the dataset. we will be doing PCA analysis to understand if it is need to do the feature reduction.

3.8 Identification of principal component

```
# Checking PC's for the data
pca_components <- prcomp(encoded_transform_data %>%
                           dplyr::select(where(is.numeric)))

# Summary
pca_sum <- summary(pca_components)

# Scree plot to check the number of components to use
screeplot(pca_components, col = "blue", type = "line", main = "Scree Plot")
```



The PCA is done in all the numerical variables, including binary variables and the components are tested for the proportion of variance. Proportion of Variance indicates the fraction of the total variance in the data explained by each principal component. Cumulative Proportion shows the accumulated proportion of variance explained up to each principal component. PC1 is the most important component capturing **31.753** percentage of the data. and the cummulative propotion shows upto PC5 it captures **72.275** of the data at it eventually stabilizes with small differences. We can look at the Scree plot to see that the components after PC5 are very close and doesn't contribute much to the data.

Though PCA gave us good insight to the variances and number if features, We are not using the dimensionaly reduced features for this screening data analysis as the first component explains very less variance, principal components might not capture a substantial amount of the total variance in the data.

4 Model Construction

Sampling is done to all three datasets (original. Transformed, Dummy_coded) to check the accuracy. We are using 3 models including (logistic regression, Random forest and SVM). All these are efficient and can handle all numeric and categorical variable. Our target variable here is going to be autism and the Class.ASD (previously predicted output from the app) will be calculated for its accuracy and kept in the separate variable while creating training and testing variable.

4.1 Creation of training and validation subsets

```
# Before doing sampling we are converting all the character to factors
# Original
```

```r
col_names <- c("gender", "ethnicity", "jundice", "autism",
               "contry_of_res", "relation")
ar_clean_data[, col_names] <- lapply(ar_clean_data
                                            [, col_names], as.factor)

# Transform
col_names <- c("gender", "ethnicity", "jundice", "autism",
               "contry_of_res", "relation")
Transform[, col_names] <- lapply(Transform[, col_names], as.factor)

# Coded
col_names <- c("jundice", "autism", "contry_of_res", "relation")
encoded_transform_data[, col_names] <- lapply(encoded_transform_data
                                              [, col_names], as.factor)


# Subsetting to training and validation dataset
set.seed(111)

# Using caret to create partition to maintain the target distribution
train_index <- createDataPartition(ar_clean_data$autism, p = 0.7, list = FALSE)
original_train_data <- ar_clean_data[train_index, ]
original_valid_data <- ar_clean_data[-train_index, ]

# Using same index to separate Tranform data
transformed_train_data <- Transform[train_index, ]
transformed_valid_data <- Transform[-train_index, ]

# Using same index to separate Dummy coded data
coded_train_data <- encoded_transform_data[train_index, ]
coded_valid_data <- encoded_transform_data[-train_index, ]

prop_table <- knitr::kable(data.frame
                           (rbind(Train = prop.table(table(original_train_data$autism)),
                                  Validation = prop.table(table(original_valid_data$autism))
                                  )))
```

Splitting the data into training and validation sets with the ratio of 70 and 30 using random sampling. The training set is used to build classification models, and the validation set is used to assess the models' performance. The fixed seed ensures us to obtain consistent results when splitting the data. We are checking for all the three datasets and all the character fetures are converted to categorical before modelling.

The "Caret" package is used to created the partition as it creates equal distribution of the traget variable in training and validation. The prop table is used to check the proportion of the target variable in training and validation data.

Proportion Table

	no	yes
Train	0.8701826	0.1298174
Validation	0.8714286	0.1285714

4.2 Creation of Logistic regression model

```r
# For original
# create logistic regression model for original
lr_classf <- glm(autism~ .,
                 data=original_train_data, family=binomial)

# Checking the summary
summary_lr <- summary(lr_classf)


# Prediction and confusion matrix
# make predictions on training data
lrm_pred <- predict(lr_classf,
                          newdata = original_valid_data %>%
                      dplyr::select(-"autism")
                          ,type = "response", positive="yes")

lrm_pred <- ifelse(lrm_pred > 0.2,"yes", "no")
lrm_predicted <- (as.factor(unname(lrm_pred)))

# Confusion matrix for the validation data
lr_conf_matrix <- confusionMatrix(table(predicted = lrm_predicted,
                                        actual = original_valid_data$autism),
                                  positive="yes")

# information from confusion matrix
lr_TN <- lr_conf_matrix[2][[1]][1]
lr_FN <- lr_conf_matrix[2][[1]][3]
lr_FP <- lr_conf_matrix[2][[1]][2]
lr_TP <- lr_conf_matrix[2][[1]][4]

# Accuracy
lr_accuracy <- (lr_TP + lr_TN )/ (lr_TN + lr_TP + lr_FN + lr_FP)

# Precision
lr_precision <- lr_TP/(lr_TP + lr_FP)

print(lr_conf_matrix$table)
```

```
         actual
predicted  no yes
      no  147  15
      yes  36  12
```

The Confusion matrix table for the actual and predicted diagnosis of the validation dataset is derived. The table shows that there are **147** True Negative (TN) observations were correctly predicted as “no”. **36** False Positive (FP) observations were predicted as “yes” but actually “no”. **15** False Negative (FN) observations were predicted as “no” but actually “yes”. **12** True Positive (TP) observations were correctly predicted as “yes”. The model here has high tendency to choose “no” so we are adjust the and give higher probabilty to choose “yes” as this is the best case in diagnostic data.

The overall accuracy, that is the proportion of correct predictions out of the total predictions. In this case, it is **0.7571429**, the precision for the model from the confusion matrix is **0.25** While the accuracy and precision of our model may seem to be good, given our models over reliance on predicting no (which is a majority of our data) The accuracy cannot be completely relied on.

```r
# For transformed
# create logistic regression model
lr_classf_transformed <- glm(autism~ .,
                data=transformed_train_data, family=binomial)

# Checking the summary
summary_lr_trans <- summary(lr_classf_transformed)

# Prediction and confusion matrix
# make predictions on training data
lrm_pred <- predict(lr_classf_transformed,
                          newdata = transformed_valid_data %>%
                      dplyr::select(-"autism")
                          ,type = "response", positive="yes")

lrm_pred <- ifelse(lrm_pred > 0.2,"yes", "no")
lrm_predicted <- (as.factor(unname(lrm_pred)))

# Confusion matrix for the validation data
lr_conf_matrix <- confusionMatrix(table(predicted = lrm_predicted,
                                        actual = transformed_valid_data$autism),
                                  positive="yes")

# information from confusion matrix
lr_TN <- lr_conf_matrix[2][[1]][1]
lr_FN <- lr_conf_matrix[2][[1]][3]
lr_FP <- lr_conf_matrix[2][[1]][2]
lr_TP <- lr_conf_matrix[2][[1]][4]

# Accuracy
lr_accuracy <- (lr_TP + lr_TN )/ (lr_TN + lr_TP + lr_FN + lr_FP)

# Precision
lr_precision <- lr_TP/(lr_TP + lr_FP)

print(lr_conf_matrix$table)
```

```
         actual
predicted  no yes
      no  145  16
      yes  38  11
```

The Confusion matrix table for the actual and predicted diagnosis of the validation dataset is derived. The table shows that there are **145** True Negative (TN) observations were correctly predicted as "no". **38** False Positive (FP) observations were predicted as "yes" but actually "no". **16** False Negative (FN) observations were predicted as "no" but actually "yes". **11** True Positive (TP) observations were correctly predicted as "yes". The model here has high tendency to choose "no" so we are adjust the and give higher probabilty to choose "yes" as this is the best case in diagnostic data.

The overall accuracy, that is the proportion of correct predictions out of the total predictions. In this case, it is **0.7428571**, the precision for the model from the confusion matrix is **0.2244898**. Though this model has the transformed data, it has less accuracy and precision compared to the original data. So we will not be using this for our ensemble model.

```
# For dummy coded
# create logistic regression model for coded
lr_classf_coded <- glm(autism~ .,
                 data=coded_train_data, family=binomial)

# Checking the summary
summary_lr_coded <- summary(lr_classf_coded)


# Prediction and confusion matrix
# make predictions on training data
suppressWarnings(lrm_pred <- predict(lr_classf_coded,
                          newdata = coded_valid_data %>%
                      dplyr::select(-"autism")
                          ,type = "response", positive="yes"))

lrm_pred <- ifelse(lrm_pred > 0.2,"yes", "no")
lrm_predicted <- (as.factor(unname(lrm_pred)))

# Confusion matrix for the validation data
lr_conf_matrix <- confusionMatrix(table(predicted = lrm_predicted,
                                        actual = coded_valid_data$autism),
                                  positive="yes")

# information from confusion matrix
lr_TN <- lr_conf_matrix[2][[1]][1]
lr_FN <- lr_conf_matrix[2][[1]][3]
lr_FP <- lr_conf_matrix[2][[1]][2]
lr_TP <- lr_conf_matrix[2][[1]][4]

# Accuracy
lr_accuracy <- (lr_TP + lr_TN )/ (lr_TN + lr_TP + lr_FN + lr_FP)

# Precision
lr_precision <- lr_TP/(lr_TP + lr_FP)

print(lr_conf_matrix$table)
```

```
         actual
predicted  no yes
      no  145  16
      yes  38  11
```

The coded data has all the properties with similar with the transformed logistic regression. The overall accuracy, that is the proportion of correct predictions out of the total predictions. In this case, it is **0.7428571**, the precision for the model from the confusion matrix is **0.2244898**. As the both coded and the transformed data both have less accuracy compared to the original data we will be picking original data for the ensemble model.

4.3 Creation of Randomforest model

```r
set.seed(111)
# For randomforest original
# create  randomforest model
rf_classf <- randomForest(autism~ .,
                 data=original_train_data,
                 )

# Checking the summary
summary_rf <- summary(rf_classf)


# Prediction and confusion matrix
# make predictions on training data
rfm_pred <- predict(rf_classf,
                        newdata = original_valid_data %>%
                      dplyr::select(-"autism")
                        ,type = "response", positive="yes")


# Confusion matrix for the validation data
rf_conf_matrix <- confusionMatrix(table(predicted = rfm_pred,
                                        actual = original_valid_data$autism),
                                  positive="yes")

# information from confusion matrix
rf_TN <- rf_conf_matrix[2][[1]][1]
rf_FN <- rf_conf_matrix[2][[1]][3]
rf_FP <- rf_conf_matrix[2][[1]][2]
rf_TP <- rf_conf_matrix[2][[1]][4]

# Accuracy
rf_accuracy <- (rf_TP + rf_TN )/ (rf_TN + rf_TP + rf_FN + rf_FP)

# Precision
rf_precision <- rf_TP/(rf_TP + rf_FP)

print(rf_conf_matrix$table)
```

```
         actual
predicted  no yes
      no  179  25
      yes   4   2
```

The Confusion table shows that there are **179** True Negative (TN) observations were correctly predicted as "no". **4** False Positive (FP) observations were predicted as "yes" but actually "no". **25** False Negative (FN) observations were predicted as "no" but actually "yes". **2** True Positive (TP) observations were correctly predicted as "yes". Here the false negative are very high compared to true positives and it might not be the ideal for the diagnosis dataset like this. The overall accuracy, that is the proportion of correct predictions out of the total predictions. In this case, it is **0.8619048**, the precision for the model from the confusion matrix is **0.3333333**.

```
set.seed(123)
# For transformed
# create randomforest model
rf_classf_transformed <- randomForest(autism~ .,
                  data=transformed_train_data,
                  )

# Checking the summary
summary_rf_trans <- summary(rf_classf_transformed)

# Prediction and confusion matrix
# make predictions on training data
rfm_pred <- predict(rf_classf_transformed,
                          newdata = transformed_valid_data %>%
                      dplyr::select(-"autism")
                          ,type = "response", positive="yes")


# Confusion matrix for the validation data
rf_conf_matrix <- confusionMatrix(table(predicted = rfm_pred,
                                        actual = transformed_valid_data$autism),
                                  positive="yes")

# information from confusion matrix
rf_TN <- rf_conf_matrix[2][[1]][1]
rf_FN <- rf_conf_matrix[2][[1]][3]
rf_FP <- rf_conf_matrix[2][[1]][2]
rf_TP <- rf_conf_matrix[2][[1]][4]

# Accuracy
rf_accuracy <- (rf_TP + rf_TN )/ (rf_TN + rf_TP + rf_FN + rf_FP)

# Precision
rf_precision <- rf_TP/(rf_TP + rf_FP)

print(rf_conf_matrix$table)
```

```
         actual
predicted  no yes
      no  178  26
      yes   5   1
```

With minor changes in the confusion matrix the transformed data doesn't have any improvement to the model. The overall accuracy, that is the proportion of correct predictions out of the total predictions. In this case, it is **0.852381**, the precision for the model from the confusion matrix is **0.1666667**. The precision is very less and also the accuracy is little less compared to the original data.

```
set.seed(111)
# For dummy coded
# create randomforest model for coded
rf_classf_coded <- randomForest(autism~ .,
                  data=coded_train_data,
                  )
```

```
# Checking the summary
summary_rf_coded <- summary(rf_classf_coded)


# Prediction and confusion matrix
# make predictions on training data
rfm_pred <- predict(rf_classf_coded,
                         newdata = coded_valid_data %>%
                      dplyr::select(-"autism")
                         ,type = "response", positive="yes")

# Confusion matrix for the validation data
rf_conf_matrix <- confusionMatrix(table(predicted = rfm_pred,
                                     actual = coded_valid_data$autism),
                                 positive="yes")

# information from confusion matrix
rf_TN <- rf_conf_matrix[2][[1]][1]
rf_FN <- rf_conf_matrix[2][[1]][3]
rf_FP <- rf_conf_matrix[2][[1]][2]
rf_TP <- rf_conf_matrix[2][[1]][4]

# Accuracy
rf_accuracy <- (rf_TP + rf_TN )/ (rf_TN + rf_TP + rf_FN + rf_FP)

# Precision
rf_precision <- rf_TP/(rf_TP + rf_FP)

print(rf_conf_matrix$table)
```

```
         actual
predicted  no yes
      no  179  25
      yes   4   2
```

The confusion matrix doesn't show any changes from the original data as well. The accuracy is very similar to the original data. **0.8619048** is the accuracy the precision for the model from the confusion matrix is **0.3333333**. This might be because The features that were transformed or dummy coded may not have contributed much to the model's decision-making process. If the original features already contained the necessary information for the model, additional preprocessing might not change the outcome.

4.4 Creation of svm model

```
# changing the target as binary output in train
svm_data <- original_train_data %>% mutate(autism
                                         = ifelse(autism =="yes", 1, 0))

# For svm original
# create svm model
svm_classf <- svm(autism~ .,
                 data=svm_data, cost = 10,
                 kernal = "linear")


# Checking the summary
summary_svm <- summary(svm_classf)


# Prediction and confusion matrix
# make predictions on training data
svmm_pred <- predict(svm_classf,
                          newdata = original_valid_data %>%
                      dplyr::select(-"autism")
                          ,type = "response", positive=1)

svmm_pred <- ifelse(svmm_pred > 0.1,"yes", "no")

# Confusion matrix for the validation data
svm_conf_matrix <- confusionMatrix(table(predicted = svmm_pred,
                                      actual = original_valid_data$autism),
                                 positive="yes")

# information from confusion matrix
svm_TN <- svm_conf_matrix[2][[1]][1]
svm_FN <- svm_conf_matrix[2][[1]][3]
svm_FP <- svm_conf_matrix[2][[1]][2]
svm_TP <- svm_conf_matrix[2][[1]][4]

# Accuracy
svm_accuracy <- (svm_TP + svm_TN )/ (svm_TN + svm_TP + svm_FN + svm_FP)

# Precision
svm_precision <- svm_TP/(svm_TP + svm_FP)

print(svm_conf_matrix$table)
```

```
         actual
predicted  no yes
      no  131  15
      yes  52  12
```

The Confusion table shows that there are **131** True Negative (TN) observations were correctly predicted as “no”. **52** False Positive (FP) observations were predicted as “yes” but actually “no”. **15** False Negative (FN) observations were predicted as “no” but actually “yes”. **12** True Positive (TP) observations were correctly predicted as “yes”.

We are changing the svm target variable to binary so that we can alter the probability more towards “Yes” for autism than “no” as our data has high tendency to move towards “no”. The cost id set to 10 for every misclassification. while tuning the data we could find the best cost to fit the model better. Before predicion the Kernals are check for the best model and the linear is chosen.

The overall accuracy, that is the proportion of correct predictions out of the total predictions. In this case, it is **0.6809524**, the precision for the model from the confusion matrix is **0.1875**.

```r
# changing the target as binary output in train
svm_data <- transformed_train_data %>% mutate(autism
                                          = ifelse(autism =="yes", 1, 0))

# For transformed
# create svm model
svm_classf_transformed <- svm(autism~ .,
                 data=svm_data, cost = 10,
                 kernal = "linear")


# Checking the summary
summary_svm_trans <- summary(svm_classf_transformed)

# Prediction and confusion matrix
# make predictions on training data
svmm_pred <- predict(svm_classf_transformed,
                          newdata = transformed_valid_data %>%
                      dplyr::select(-"autism")
                          ,type = "response", positive=1)

svmm_pred <- ifelse(svmm_pred > 0.1,"yes", "no")

# Confusion matrix for the validation data
svm_conf_matrix <- confusionMatrix(table(predicted = svmm_pred,
                                     actual = transformed_valid_data$autism),
                                  positive="yes")

# information from confusion matrix
svm_TN <- svm_conf_matrix[2][[1]][1]
svm_FN <- svm_conf_matrix[2][[1]][3]
svm_FP <- svm_conf_matrix[2][[1]][2]
svm_TP <- svm_conf_matrix[2][[1]][4]

# Accuracy
svm_accuracy <- (svm_TP + svm_TN )/ (svm_TN + svm_TP + svm_FN + svm_FP)

# Precision
svm_precision <- svm_TP/(svm_TP + svm_FP)

print(svm_conf_matrix$table)
```

```
         actual
predicted  no yes
      no  133  14
      yes  50  13
```

The Confusion table shows that there are **133** True Negative (TN) observations were correctly predicted as “no”. **50** False Positive (FP) observations were predicted as “yes” but actually “no”. **14** False Negative (FN) observations were predicted as “no” but actually “yes”. **13** True Positive (TP) observations were correctly predicted as “yes”. The overall accuracy, that is the proportion of correct predictions out of the total predictions. In this case, it is **0.6952381**, the precision for the model from the confusion matrix is **0.2063492**.

In SVM the transformed data seems to have a better accuracy and than the original. It is also converted to binaries and the probabilities are adjusted more toward the "yes", since the percentage of Yes in the autism data is very small.

```
set.seed(111)
# changing the target as binary output in train
svm_data <- coded_train_data %>% mutate(autism
                                          = ifelse(autism =="yes", 1, 0))

# For coded
# create svm model
svm_classf_coded <- svm(autism~ .,
                 data=svm_data, cost = 10,
                 kernal = "linear")


# Checking the summary
summary_svm_coded <- summary(svm_classf_coded)

# Prediction and confusion matrix
# make predictions on training data
svmm_pred <- predict(svm_classf_coded,
                          newdata = coded_valid_data %>%
                      dplyr::select(-"autism")
                          ,type = "response", positive=1)

svmm_pred <- ifelse(svmm_pred > 0.1,"yes", "no")

# Confusion matrix for the validation data
svm_conf_matrix <- confusionMatrix(table(predicted = svmm_pred,
                                        actual = coded_valid_data$autism),
                                  positive="yes")

# information from confusion matrix
svm_TN <- svm_conf_matrix[2][[1]][1]
svm_FN <- svm_conf_matrix[2][[1]][3]
svm_FP <- svm_conf_matrix[2][[1]][2]
svm_TP <- svm_conf_matrix[2][[1]][4]

# Accuracy
svm_accuracy <- (svm_TP + svm_TN )/ (svm_TN + svm_TP + svm_FN + svm_FP)

# Precision
svm_precision <- svm_TP/(svm_TP + svm_FP)

print(svm_conf_matrix$table)
```

```
         actual
predicted  no yes
      no  124  12
      yes  59  15
```

The Confusion table shows that there are **124** True Negative (TN) observations were correctly predicted as "no". **59** False Positive (FP) observations were predicted as "yes" but actually "no". **12** False Negative

(FN) observations were predicted as “no” but actually “yes”. **15** True Positive (TP) observations were correctly predicted as “yes”. The overall accuracy, that is the proportion of correct predictions out of the total predictions. In this case, it is **0.6619048**, the precision for the model from the confusion matrix is **0.2027027**. The dummy coding with the linear kernal has improved this model which gives us less Faalse negative values so we are choosing this coded svn for future analysis.

4.5 Appropriеteness of the model

The three models that is been used are Logistic Regression, Random Forest and SVM.

Logistic Regression: Logistic regression is suitable for binary classification problems which can both categorical and numerical variables. There is a possibility of adjusting the probabilities based on the domain knowledge and it is less prone to overfitting.

Random Forest : Creates ensemble of trees and are effective for capturing complex relationships and interactions in the data. It can handle both categorical and numerical and can work will with dataset with many features.

SVM : Can handle high dimensional data works very well with binary classification. It also has kernal’s and some hyperparameters that can be adjusted to get good classification.

5 Model Evaluation

There are many ways to do the model evaluation to estimate and understand which model works better, The following are some of the techniques we are using to do the evaluation.

5.1 Evaluation of fit of the model with holdout method

The holdout method were we take the models separate it to training and testing with 70 to 30 ratio and the prediction is done for the validation data. All the model are test with three different dataset and based on the accuracy, false negative, True positive values and some other parameters we are choosing to work with original dataset for both logistic and randomforest and Tranformed data for the SVM model respectively

```
# Logistic regression on original data
lrm_pred <- predict(lr_classf,
                         newdata = original_valid_data %>%
                     dplyr::select(-"autism")
                         ,type = "response", positive="yes")

lrm_pred <- ifelse(lrm_pred > 0.2,"yes", "no")
lrm_predicted <- (as.factor(unname(lrm_pred)))

# Confusion matrix for the validation data
lr_conf_matrix <- confusionMatrix(table(predicted = lrm_predicted,
                                        actual = original_valid_data$autism),
                                  positive="yes")

# information from confusion matrix
lr_TN <- lr_conf_matrix[2][[1]][1]
lr_FN <- lr_conf_matrix[2][[1]][3]
lr_FP <- lr_conf_matrix[2][[1]][2]
lr_TP <- lr_conf_matrix[2][[1]][4]
```

```r
# Accuracy
lr_accuracy <- (lr_TP + lr_TN )/ (lr_TN + lr_TP + lr_FN + lr_FP)

# Precision
lr_precision <- lr_TP/(lr_TP + lr_FP)

# Recall
lr_recall <- lr_TP/(lr_TP + lr_FN)

logistic<- knitr::kable(lr_conf_matrix$table)
```

The confusion matrix for the logistic regression:

Logistic Regression

	no	yes
no	147	15
yes	36	12

Accuracy : **0.7571429** Precision : **0.25** Recall : **0.4444444**

```r
set.seed(111)
# make predictions on training data original
rfm_pred <- predict(rf_classf,
                          newdata = original_valid_data %>%
                      dplyr::select(-"autism")
                          ,type = "response", positive="yes")


# Confusion matrix for the validation data
rf_conf_matrix <- confusionMatrix(table(predicted = rfm_pred,
                                        actual = original_valid_data$autism),
                                  positive="yes")

# information from confusion matrix
rf_TN <- rf_conf_matrix[2][[1]][1]
rf_FN <- rf_conf_matrix[2][[1]][3]
rf_FP <- rf_conf_matrix[2][[1]][2]
rf_TP <- rf_conf_matrix[2][[1]][4]

# Accuracy
rf_accuracy <- (rf_TP + rf_TN )/ (rf_TN + rf_TP + rf_FN + rf_FP)

# Precision
rf_precision <- rf_TP/(rf_TP + rf_FP)

# Recall
rf_recall <- rf_TP/(rf_TP + rf_FN)

# confusion table
random <- knitr::kable(rf_conf_matrix$table)
```

The confusion matrix for the Random forest:

Random Forest

	no	yes
no	179	25
yes	4	2

Accuracy : **0.8619048** Precision : **0.3333333** Recall : **0.0740741**

```r
# SVM for  coded data
# make predictions on training data
svmm_pred <- predict(svm_classf_coded,
                          newdata = coded_valid_data %>%
                      dplyr::select(-"autism")
                          ,type = "response", positive=1)

svmm_pred <- ifelse(svmm_pred > 0.1,"yes", "no")

# Confusion matrix for the validation data
svm_conf_matrix <- confusionMatrix(table(predicted = svmm_pred,
                                         actual = coded_valid_data$autism),
                                   positive="yes")

# information from confusion matrix
svm_TN <- svm_conf_matrix[2][[1]][1]
svm_FN <- svm_conf_matrix[2][[1]][3]
svm_FP <- svm_conf_matrix[2][[1]][2]
svm_TP <- svm_conf_matrix[2][[1]][4]

# Accuracy
svm_accuracy <- (svm_TP + svm_TN )/ (svm_TN + svm_TP + svm_FN + svm_FP)

# Precision
svm_precision <- svm_TP/(svm_TP + svm_FP)

# Recall
svm_recall <- svm_TP/(svm_TP + svm_FN)

# confusion table
svm_table <- knitr::kable(svm_conf_matrix$table)
```

The confusion matrix for the SVM:

SVM Model

	no	yes
no	124	12
yes	59	15

Accuracy : **0.6619048** Precision : **0.2027027** Recall : **0.5555556_**

Based on all the model outputs, accuracy, precision the best model in the these three models are “SVM”, logistic regression is very close comes next to the SVM. We are choosing SVM as the best model, as the the recall which is the model predicted higher correct “YES” diagnosis, and the false negatives are less compared to the other models. The precision is also not very small compared to logistic. So opting to choose SVM. but we can further validate the model with K Fold cross validation to evaluate and get better understanding best model decision.

5.2 Evaluation with K Fold cross validation

```r
set.seed(111)
# creating control parameters for k fold validation
t_ctrl <- trainControl(method = "cv", number = 10)

# Kfold logistic regression for original model
suppressWarnings(lr_kfold <- train(autism~ .,
                 data=ar_clean_data, family=binomial,
                 method = "glm",trControl = t_ctrl))

print(lr_kfold)
```

The model has the accuracy of **87.34** percentage The accuracy is high and can be used as the good indicator for the diagnosis of autism, though its is towards one side is understandable based on the percentage of “yes” diagnosis in autism is very less.

```r
set.seed(111)

# Kfold random forest for original model
rf_kfold <- train(autism~ .,
                 data=ar_clean_data,
                 method = "rf",trControl = t_ctrl)

print(rf_kfold)
```

The model has accuracy the mtry **2** of with accuracy of **87.06** percentage While tuning hyper parameters we could adjust the mtry based on this. Since the Accuracy is similar to the logistic regression and kappa is less than the previous model we are not using this as the best model for the autism screening prediction.

```r
set.seed(111)

# Kfold random forest for original model
svm_kfold <- train(autism~ .,
                 data= encoded_transform_data,
                 method = "svmLinear",
                 cost = 10,
                 trControl = t_ctrl)

print(svm_kfold)
```

This svm has the accuracy of **87.06** While tuning, while tuning we could work on the best cost selection for the model.

All the models have very less values for Kappa and it might be because of that Kappa might be less informative in binary classification problems, especially when there is a class imbalance.

5.3 Tuning of model parameters

Tuning the model parameter to get the best model for the prediction involves finding the optimal values for hyperparameters that control the training method. We will try multiple hyper parameters in all there models and tune the model to get right classification in the diagnosis.

```
# Tuning Random forest
set.seed(111)
rf_classf_tuned <- randomForest(autism~ .,
                  data=original_train_data,
                  ntree = 500,
                  mtry = sqrt(length(original_train_data)))

# Checking the summary
summary_rf_tuned <- summary(rf_classf_tuned)


# Prediction and confusion matrix
# make predictions on training data
rfm_pred_tuned <- predict(rf_classf_tuned,
                           newdata = original_valid_data %>%
                      dplyr::select(-"autism")
                           ,type = "response", positive="yes")



# Confusion matrix for the validation data
rf_conf_matrix_tuned <- confusionMatrix(table(predicted = rfm_pred_tuned,
                                        actual = original_valid_data$autism),
                                  positive="yes")

print(rf_conf_matrix_tuned$table)
```

```
         actual
predicted  no yes
      no  179  25
      yes   4   2
```

Tuned Random Forest

Tuning the random forest model by including the mtry and ntree parameters. Random forest is a robust algorithm although changing the mtry and ntree parameters doesn't affect the prediction. The overall accuracy is maintain at **86.19** .

```
set.seed(111)
# changing the target as binary output in train
svm_data <- coded_train_data %>% mutate(autism
                                            = ifelse(autism =="yes", 1, 0))

# tuning SVM with sequence of costs
costvalues <- 10^seq(-3,2,1)
tuned.svm <- tune(svm, autism~ ., data = svm_data,
                  ranges = list(cost = costvalues), kernel = "linear")

# Taking the best model
```

```r
best_cost <- tuned.svm$best.parameters$cost

# tuned best cost
svmm_pred_Tmodel <- predict(svm_classf_coded,
                         newdata = coded_valid_data %>%
                       dplyr::select(-"autism"), cost = best_cost
                         ,type = "response", positive=1)


svmm_pred_tuned <- ifelse(svmm_pred_Tmodel > 0.2,"yes", "no")

# Confusion matrix for the validation data
svm_conf_matrix_tuned <- confusionMatrix(table(predicted = svmm_pred_tuned,
                                        actual = coded_valid_data$autism),
                                  positive="yes")


#confusion matrix table
print(svm_conf_matrix_tuned$table)
```

```
         actual
predicted  no yes
      no  145  19
      yes  38   8
```

Tuned Confusion Matrix

Tuning the SVM with best cost hyperparameter does increase the Accuracy. The overall accuracy of the tuned svm model is **72.86** . However for this diagnosis dataset even with less accuracy if the dataset has lot of false negative compared to the un tuned classification so we can still select the SVM model for with untuned classification.

5.4 Comparison of Models and interpretations

```r
# Comparing 3 models acuracy precision recall and F1 score
# Logistic regression
log_accuracy <- round(lr_conf_matrix$overall["Accuracy"]*100, digits = 2)
log_precision <- round(lr_conf_matrix$byClass["Precision"], digits = 3)
log_recall <- round(lr_conf_matrix$byClass["Recall"], digits = 3)
log_F1 <- round(lr_conf_matrix$byClass["F1"], digits = 3)

# Randomforest with tuned hyperparameters
rf_accuracy <- round(rf_conf_matrix_tuned$overall["Accuracy"]*100, digits = 2)
rf_precision <- round(rf_conf_matrix_tuned$byClass["Precision"], digits = 3)
rf_recall <- round(rf_conf_matrix_tuned$byClass["Recall"], digits = 3)
rf_F1 <- round(rf_conf_matrix_tuned$byClass["F1"], digits = 3)

# SVM
svm_accuracy <- round(svm_conf_matrix$overall["Accuracy"]*100, digits = 2)
svm_precision <- round(svm_conf_matrix$byClass["Precision"], digits = 3)
svm_recall <- round(svm_conf_matrix$byClass["Recall"], digits = 3)
svm_F1 <- round(svm_conf_matrix$byClass["F1"], digits = 3)
```

```r
# adding all the elements to a dataframe
stats = c("Model","Accuracy_Percet", "Precision", "Recall", "F1_score")
evaluations = data.frame(matrix(nrow = 0, ncol = length(stats)))
colnames(evaluations) = stats
evaluations[1,] <- c("Logistic Regression",log_accuracy,
                     log_precision, log_recall, log_F1)
evaluations[2,] <- c("Random Forest",rf_accuracy,
                     rf_precision, rf_recall, rf_F1)
evaluations[3,] <- c("SVM",svm_accuracy,
                     svm_precision, svm_recall, svm_F1)
```

Compared to other models Random forest has highest accuracy of **86.19** percentage compared to the logistic regression **75.71** and **66.19**. The F1 Score for logistic regression is high **0.32** than the random forest with **0.121** and SVM **0.297** . So the better model with above 70 percent accuracy and comparitively higher F1 score is **"Logistic Regression"**. But We are yet to compare the accuracy of the current application which is present in Class.ASD in the Actual Data.

```r
# Checking all the models with the current algorithms result
# we are trying to outperform the current algorithms result
# Taking Class.ASD from the autism dataset and comparing
class_AD_conf <- confusionMatrix(table
                                 (predicted = autism_data$Class.ASD,
                                   actual = autism_data$autism))

class_Accuracy <- round(class_AD_conf$overall["Accuracy"]*100, digits = 2)
class_precision <- round(class_AD_conf$byClass["Precision"], digits = 3)
class_recall <- round(class_AD_conf$byClass["Recall"], digits = 3)
class_F1 <- round(class_AD_conf$byClass["F1"], digits = 3)


# Adding App detected accuracy to the table
evaluations[4,] <- c("Class.ASD(App)",class_Accuracy,
                     class_precision, class_recall, class_F1)

model_table <- knitr::kable(data.frame(cbind(Models = evaluations$Model,
                                             AccuracyPer = evaluations$Accuracy_Percet)))

print(class_AD_conf$table)
```

```
         actual
predicted  no yes
      no  467  48
      yes 146  43
```

Model Accuracy Table

Models	AccuracyPer
Logistic Regression	75.71
Random Forest	86.19
SVM	66.19

Models	AccuracyPer
Class.ASD(App)	72.44

Within the accuracy of our model 2 of them has higher accuracy than the current applications results. However the F1 Score and the precision recall are all very low. To stabilize this and to get a good model prediction with high F1 score and prediction, we can use a bagging technique.

6 Model Tuning and Performance Improvement

We have already tuned the models in the steps before but it can be improved further with other ensemble techniques.Usually in datasets like this autism screening having a good precision and F1 score is very important can increase the accuracy with the good precision and F1 score can be improved. We can use multiple techniques like ensemble prediction and bagging to increase the F1 score and accuracy through this.

6.1 Ensemble model construction

```r
#write function that predicts if the autism presence

predict_autismClass <- function(train_data, train_data_svm,
                                 test_data, test_data_svm,
                                 lr_classf, rf_classf, svm_classf_coded)
  {

  # initializing vote variable for the result
  vote = data.frame(lr=rep(NA,nrow(test_data)),
                    rf=rep(NA,nrow(test_data)),
                    svm=rep(NA,nrow(test_data)),
                    sum=rep(NA,nrow(test_data)))


    # Logistic regression prediction
    preds_lr <- predict(lr_classf,
                        newdata = test_data
                        ,type = "response")

    preds_lr <- ifelse(preds_lr > 0.2,"yes", "no")
    preds_lr <- (as.factor(unname(preds_lr)))

    vote$lr <- ifelse(preds_lr=="yes",1,0)

    # Random forest prediction
    preds_rf <- predict(rf_classf,
                  newdata = test_data)

    vote$rf <- ifelse(preds_rf=="yes",1,0)


    # svm prediction
    preds_svm <- predict(svm_classf_coded,
                  newdata = test_data_svm)
    vote$svm <- ifelse(preds_svm=="yes",1,0)


    # Taking the majority vote
    vote$sum <- vote$lr + vote$rf + vote$svm

    #change voted numbers to predictions
    vote$lr <- ifelse(vote$lr==1,"yes","no")
    vote$rf <- ifelse(vote$rf==1,"yes","no")
    vote$svm <- ifelse(vote$svm==1,"yes","no")
    vote$sum <- ifelse(vote$sum>=2,"yes","no")

    # return vote
    return(vote)
}
```

The ensemble function is created with the 3 models and the the sum of each predictions are voted and the highest predicted values will be selected as the predicted value of the observation. The individual predictions are also collected to get the comparison of emsemble model to the individual models.

6.2 Application of Ensembles to make the predictions

```
# function call to ensemble
ensemble_predict <- predict_autismClass(original_train_data, coded_train_data,
                                        original_valid_data %>%
                      dplyr::select(-"autism"),
                                        coded_valid_data %>%
                      dplyr::select(-"autism"),
                                        lr_classf, rf_classf, svm_classf_coded)

# Confusion Matrix
ensemble_conf <- confusionMatrix(table(ensemble_predict$sum,
                                        original_valid_data$autism),positive="yes")

print(ensemble_conf$table)
```

```
       no yes
  no  179  25
  yes   4   2
```

The Ensemble takes all the models and their respective train and test for the autism screening data. We combined all the the scores and made the predictions for the The F1 score is a statistic that combines precision and recall to provide a single value that summarises a classification model's performance.Comparing the three models (Logistic regression, Random forest and), with the ensemble score,The ensemble model had a F1-score of **0.121**. with the accuracy of **86.19**.

6.3 Comparison of ensemble to individual models

```
# Elements in ensemble
en_accuracy <- round(ensemble_conf$overall["Accuracy"]*100, digits = 2)
en_precision <- round(ensemble_conf$byClass["Precision"], digits = 3)
en_recall <- round(ensemble_conf$byClass["Recall"], digits = 3)
en_F1 <- round(ensemble_conf$byClass["F1"], digits = 3)

# Adding the ensemble values to the evaluation table
evaluations[5,] <- c("Ensemble", en_accuracy , en_precision, en_recall, en_F1)

# Printing Model table for accuracy
model_table <- knitr::kable(data.frame(cbind(Models = evaluations$Model,
                                         AccuracyPer = evaluations$Accuracy_Percet)))
```

Model Accuracy Table

Models	AccuracyPer
Logistic Regression	75.71
Random Forest	86.19
SVM	66.19
Class.ASD(App)	72.44
Ensemble	86.19

The Random Forest model has the highest accuracy among the individual models **86.19** The Ensemble model has the same accuracy as the Random Forest, suggesting that the ensemble leverages the strengths of the Random Forest. The SVM model has the lowest accuracy but has relatively higher recall **0.556** , indicating that it identifies more true positive cases but also has more false positives. The Logistic Regression model provides a balance between precision and recall with F1 score **0.32** precision and recall is balance in the the Ensemble model with F1 score of **0.121** The F1 scores are small due to th imbalance in the screening data and can be solved with plackages like "Smote" to overfit the minor class.

6.4 Bagging with homogeneous learners

Bagging the models can drastically improve the algorithms efficiency with their F1 score and the precision eventually the accuracy. we have two homogeneous model with the same hyperparameters. We can bag logistic regression and random forest algorithm and get the prediction.

```
set.seed(111)
# Bagging with the models for training set
# List of model to bag
bag_models <- list(lr_classf, rf_classf)

# bagging with all the homogeneous learners
bagged_samp <- bagging(autism ~ ., data = original_train_data, nbagg = 10,
                       coob = TRUE, predictors = bag_models)
predict_samp <- predict(bagged_samp , original_valid_data)

# Confusion matric
bag_conf_matrix_samp <- confusionMatrix(table
                                 (predicted = predict_samp,
                                   actual = original_valid_data$autism))
```

```r
# Printing the confusion matrix
print(bag_conf_matrix_samp$table)
```

```
         actual
predicted  no yes
      no  174  25
      yes   9   2
```

```r
# Bagging the models for the complete dataset
bagged_all_data <- bagging(autism ~ ., data = ar_clean_data, nbagg = 10,
                     coob = TRUE, predictors = bag_models)
predict_all_data <- predict(bagged_all_data, ar_clean_data)

# Confusion matric
bag_confM_all <- confusionMatrix(table(predicted = predict_all_data,
                                   actual = ar_clean_data$autism))


# Printing the confusion matrix
print(bag_confM_all$table)
```

```
         actual
predicted  no yes
      no  611   9
      yes   1  82
```

After Bagging both the Logistic regression and the Random forest our models the accuracy of the model increased and the F1 scores, Precision, Recall are also high.

Attributes for Bagged train-valid data Accuracy : **83.81** percentage

Precision : **0.874**

Recall : **0.951**

F1 score : **0.911**

The above results are for the data which are separated as training and testing. Bagging the models with the complete dataset gave the values as follows

Attributes for the complete dataset :

Accuracy : **98.58** Percentage

Precision : **0.985**

Recall : **0.998**

F1 score : **0.992**

overall accuracy is relatively high, the imbalanced nature of the classes is reflected in low specificity and balanced accuracy. The model seems to be biased towards predicting the majority class “no” , resulting in low sensitivity for the minority class “yes”. The complete dataset bagging method gave us a good separtion of class for the classification.

Conclusion

The main goal was to predict the likelihood of autism based on various features with the increased accuracy with and testing to outperform the current application's accuracy. Explored a dataset with features like scores, age, gender, ethnicity, and other factors. Checked distribution, summary statistics, and visualized relationships between variables, handled outliers, explored association rules to assign weights to certain features.

Implemented three models—Logistic Regression, Random Forest, and SVM for classification. Tuned model parameters for better performance. Compared individual models and an ensemble model to understand their strengths and weaknesses. The accuracy is coppared with the application results and the accuracy of randomforest and logistic regress is dericed high. Used Bagging technique tot improve the model and the F1 score. In conclusion, The project successfully followed CRISP DM through the different stages of the data mining process, providing valuable insights and predictive models for autism detection.

References

1 Autism screening on adults. (2020, August 17). Kaggle. https://www.kaggle.com/datasets/andrewmvd/autism-screening-on-adults

2 Lantz, B. (n.d.). Machine Learning with R - Third Edition. O'Reilly Online Learning. https://learning.oreilly.com/library/view/machine-learning-with/9781788295864/ch01s04.html

3 Brownlee, J. (2020, July 16). Framework for data preparation techniques in Machine Learning. MachineLearningMastery.com. https://machinelearningmastery.com/framework-for-data-preparation-for-machine-learning/

4 Wesley. (2012, September 27). Association rule learning and the apriori Algorithm | R-Bloggers. R-bloggers. https://www.r-bloggers.com/2012/09/association-rule-learning-and-the-apriori-algorithm/

5 Thabtah, F. (2018). Machine learning in autistic spectrum disorder behavioral research: A review and ways forward. Informatics for Health & Social Care, 44(3), 278–297. https://doi.org/10.1080/17538157.2017.1399132

6 Classification: precision and recall. (n.d.). Google for Developers. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

7 Pandian, S. (2021, January 10). Exploring ensemble learning in machine learning world! Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/01/exploring-ensemble-learning-in-machine-learning-world/

8 Grolemund, H. W. M. Ç. a. G. (n.d.). R for Data Science (2E) - 16 factors. https://r4ds.hadley.nz/factors.html

9 Zach. (2022, April 19). How to use SMOTE for imbalanced data in R (With example). Statology. https://www.statology.org/smote-in-r/