EMPLOYEE ATTRITION ANALYSIS

IDENTIFYING THE BEST ML MODEL TO PREDICT ATTRITION USING R

Prepared by Manveen Kaur

Table of Contents

Brief Introduction

This predictive analytics project focuses on developing a machine learning model in R to predict employee attrition for ABC Company in the best possible manner.

Employee Attrition refers to the process of employees leaving an organisation voluntarily, either through resignation, retirement, or other reasons, resulting in a reduction in the organisation's workforce size. Attrition can have significant implications for organisational performance, including increased recruitment costs, loss of institutional knowledge, and decreased morale among remaining employees. Understanding the drivers of employee attrition and implementing strategies to mitigate its impact is essential for organisations to maintain a stable and productive workforce.

By employing various ML models, we aim to identify the best model to predict employee attrition.

Problem Statement

ABC company is facing problems of high employee attrition. The company understands that employees are the most important part of an organisation, and successful employees meet deadlines, make sales, and build the brand through positive customer interactions. Hence, the company wants to understand which segments of employees are likely to resign. The company's Human Resources department has surveyed and collected data on several variables. Based on the available data, the primary objective is to create an ML model that can predict the attrition of employees in the best possible manner.

Understanding the Dataset and the Variables

The dataset provided for this analysis contains 17 variables, each capturing specific attributes related to employee demographics, employment history, and work-related factors. Below is a brief overview of the variables included in the dataset:

S. NO.	VARIABLE NAME	DESCRIPTION
1	Age	The age of the employee
2	Attrition	Indicates whether an employee has left the company (Yes) or is still employed (No)
3	BusinessTravel	Frequency of business travel among employees (Travel_Rarely, Travel_Frequently, Non-Travel)
4	Department	The department in which the employee works (Sales, Human Resources, Research & Development)
5	DistanceFromHome	The distance of the employee's home from the workplace.
6	EducationField	The education background of the employee. (Human Resources, Life Sciences, Marketing, Medical, Technical Degree, Other)
7	Employee ID	Unique identifier for each employee
8	Gender	Gender of the employee. (Male, Female)
9	JobLevel	Level of the employee's job within the organizational hierarchy (1, 2, 3, 4, 5)
10	MaritalStatus	Marital status of the employee (Married, Single, Divorced)
11	MonthlyIncome	The monthly income of the employee
12	NumCompaniesWorked	Number of companies the employee has worked for previously.
13	PercentSalaryHike	Percentage increase in salary during the most recent salary hike.
14	TotalWorkingYears	Total number of years the employee has been working.
15	YearsAtCompany	Number of years the employee has been with the company.
16	YearsSinceLastPromotion	Number of years since the employee's last promotion.
17	YearsWithCurrManager	Number of years the employee has been working under the current manager.

Exploratory Data Analysis/ Data Cleaning

Data Cleaning is the process of transforming raw data into consistent data that can be easily analysed. It is very important for any type of data analysis and especially for creating predictive models, as the quality of the model heavily depends on the quality of the data. This portion of our document will discuss our data-cleaning process for the Employee Attrition dataset.

Tasks and Techniques Applied for Data Cleaning

Below are the methods and steps we undertook to attain a refined dataset suitable for applying diverse machine-learning models -

- Converting the Data Set into a Data Frame
- Correcting Data Types
- Removal of Unwanted data
- Dealing with Missing Values
- Checking and Treating Normality and Outliers

1. Converting the Data Set into a Data Frame

The first step we took while transforming our raw data was to convert the data file Employee Attrition stored in the variable EA into a data frame. It is crucial to convert the data set into a data frame because many packages only work on data frames, and if it's not converted, the data analysis will be faulty**.** This conversion is performed by using the function **as.data.frame**

```
# loading data
EA <- EmployeeAttrition
EA <- as.data.frame(EA)
```

2. Correcting Data Types

In this particular step, we correct the data types of various columns by converting the categorical fields into factors to apply the statistical models correctly. In the same step, there are two particular columns, **NumCompaniesWorked and TotalWorkingYears,** where there are missing values, but instead of those missing values being null values, they are the characters “NA”, so the data type of these columns are characters instead of the data type being numeric. To correct this issue, we use the function **as.numeric**.

```
EA[,c(2,3,4,6,8,9,10)] <- lapply(EA[,c(2,3,4,6,8,9,10)], factor)#categorical to factors
EA[,c(12,14)] <- lapply(EA[,c(12,14)],as.numeric)#Char data type to numeric data type
```

3. Replacing Appropriate Values

Here, we are removing all the columns which are not important while making predictive machine learning models so that our outcomes are not faulty. In this particular instance, there was only one unnecessary column, the **EmployeeID** column**.**

```
# Subsetting into required dataset/ Removing unwanted column (Employee ID)
EA <- EA[,c(-7)]
str(EA)
```

4. Dealing with Missing Values

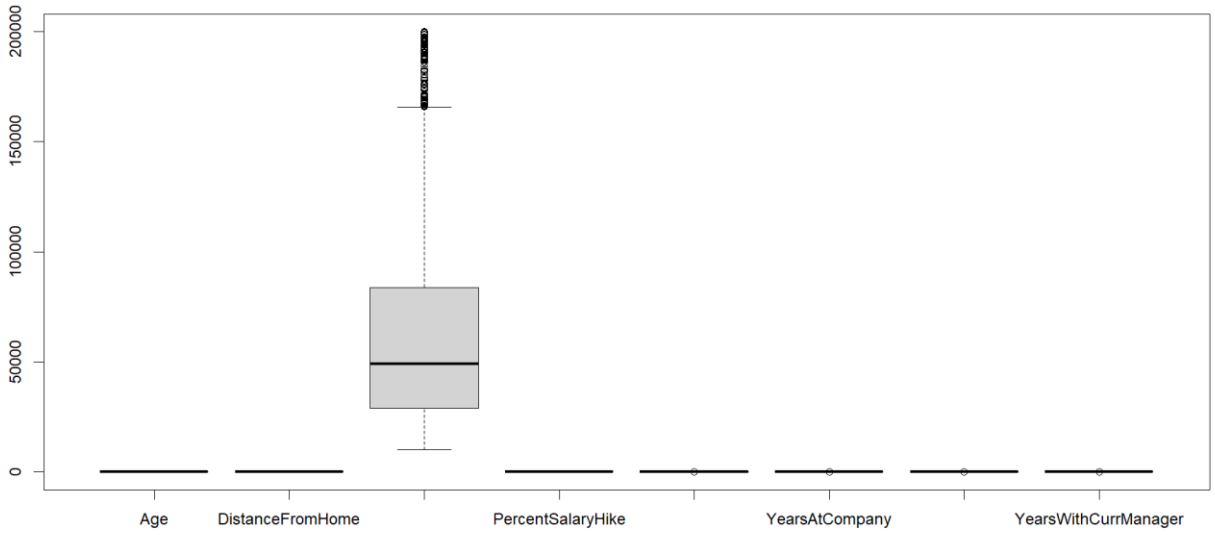
While making a model, it is imperative that there aren't any missing values in the data set, as missing values decrease the predictive power of the model. Missing values were only found in two columns, **NumCompaniesWorked and TotalWorkingYears.** There are two ways to deal with missing values: Deletion of the missing values and Imputation of the missing values. The method we chose to go with is the **imputation of missing values** and not the deletion of missing values, as that step is only applied when a particular column has more than 70-75% missing values. While imputing the data of the two mentioned columns, the central tendency measure we use to impute is the median, as the data in those columns are numeric.

```
#Dealing with Missing Data - Imputing the missing values
map(EA,~sum(is.na(.)))
EA[11]<-impute(EA[11],fun = median) #NumCompaniesWorked
EA[13]<-impute(EA[13],fun = median) #TotalWorkingYears
```

5. Checking and Treating Normality and Outliers

In this particular step, we check the normality of the data along with the fact if there are any outliers in the numeric data. To check the normality of the data, we check the skewness and kurtosis, which should be in the range of -1 to +1 for skewness and -4 to +4 for kurtosis. To check the outliers in the data, we made a box plot. After checking the skewness and kurtosis, we realise the data is not normal, and after creating the boxplot of the numeric data, we see that the data isn't normal and there are outliers. The columns where we found outliers were **MonthlyIncome, TotalWorkingYears, YearsSinceLastPromotion and YearsWithCurrManager.**

```
> skewness(EA[c(1,5,10,12,13,14,15,16)])
                      Age          DistanceFromHome             MonthlyIncome
                0.4128645                 0.9571400                 1.3684185
      PercentSalaryHike         TotalWorkingYears             YearsAtCompany
                0.8202899                 1.1184985                 1.7627284
YearsSinceLastPromotion     YearsWithCurrManager
                1.9822646                 0.8326003
> kurtosis(EA[c(1,5,10,12,13,14,15,16)])
                      Age          DistanceFromHome             MonthlyIncome
                2.593149                  2.771852                  3.997738
      PercentSalaryHike         TotalWorkingYears             YearsAtCompany
                2.696344                  3.919605                  6.918057
YearsSinceLastPromotion     YearsWithCurrManager
                6.596318                  3.166398
```



6. Treating Normality and Outliers

➔ **Step 1:** Storing the outliers in vectors

The first step we take while treating our outliers is storing the outliers in vectors, as it is important for the methods of treating outliers where we remove outliers and the method where we impute outliers.

```
#Storing Outliers in Vectors
out10<-boxplot(EA$MonthlyIncome)$out
out13<-boxplot(EA$TotalWorkingYears)$out
out14<-boxplot(EA$YearsAtCompany)$out
out15<-boxplot(EA$YearsSinceLastPromotion)$out
out16<-boxplot(EA$YearsWithCurrManager)$out

# Num of Outliers in each vector
length(out10)
length(out13)
length(out14)
length(out15)
length(out16)
```

➔ **Step 2**: Creating Data Copies

We are creating three data copies of EA named EA1, EA2 and EA3 so that we can apply the three methods for treating normality and outliers without hampering the original data.

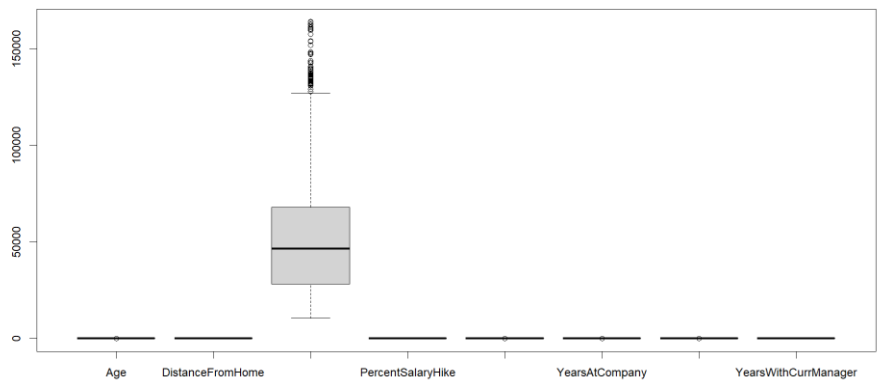
```
# Creating Data Copies
EA1<-EA
EA2<-EA
EA3<-EA
```

➔ **Step 3:** Applying Method 1 (Removing Outliers in EA1)

The first method for treating normality and outliers is the removal of outliers. For this, in data copy EA1, we remove the outliers stored in vectors out 10, out 13, out 14, out 15, and out 16. After applying this method, we see that the data is still not normal, as the skewness and kurtosis are not within the aforementioned range. Even the outlier issue is not solved, as almost all the previous columns still have outliers, and a new column, **Age,** also has outliers.

```
#Method 1: Removing Outliers in EA1 --------------------------
EA1<-EA1[-which(EA1$MonthlyIncome %in% out10),]
EA1<-EA1[-which(EA1$TotalWorkingYears %in% out13),]
EA1<-EA1[-which(EA1$YearsAtCompany %in% out14),]
EA1<-EA1[-which(EA1$YearsSinceLastPromotion %in% out15),]
EA1<-EA1[-which(EA1$YearsWithCurrManager %in% out16),]

boxplot(EA1[c(1,5,10,12,13,14,15,16)])
skewness(EA1[c(1,5,10,12,13,14,15,16)])
kurtosis(EA1[c(1,5,10,12,13,14,15,16)])
```

```
> skewness(EA1[c(1,5,10,12,13,14,15,16)])
                Age        DistanceFromHome           MonthlyIncome
          0.5159519               0.9856691               1.1943969
     PercentSalaryHike       TotalWorkingYears          YearsAtCompany
          0.8086118               0.9332259               0.7711648
YearsSinceLastPromotion    YearsWithCurrManager
          1.6794656               0.7831050
> kurtosis(EA1[c(1,5,10,12,13,14,15,16)])
                Age        DistanceFromHome           MonthlyIncome
           2.859878                2.860137                3.849712
     PercentSalaryHike       TotalWorkingYears          YearsAtCompany
           2.662499                3.753822                3.229953
YearsSinceLastPromotion    YearsWithCurrManager
           4.890072                2.714210
```

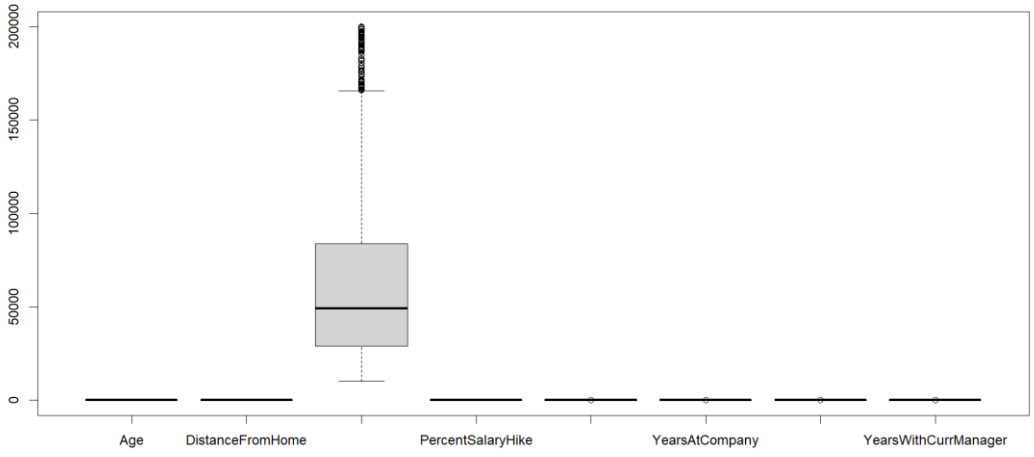


➔ **Step 4:** Applying Method 2 (Imputing Outliers in EA2)

The second method for treating normality and outliers is the imputation of outliers. For this data copy, EA2 is used. In this particular copy, we will replace the values in the vectors with NA. After replacing the outliers with NA, we will impute those values, and the central tendency measure we will use here is the median, as the data is numerical. Even after imputing the outliers, we still don't have normal data, and there are outliers.

```
#Method 2: Imputing Outliers in EA2 ----------------------------------------
# Replacing Outliers with NAs
EA2[EA2$MonthlyIncome %in% out10, "MonthlyIncome"]= NA
EA2[EA2$TotalWorkingYears %in% out13, "TotalWorkingYears"]= NA
EA2[EA2$YearsAtCompany %in% out14, "YearsAtCompany"]= NA
EA2[EA2$YearsSinceLastPromotion %in% out15, "YearsSinceLastPromotion"]= NA
EA2[EA2$YearsWithCurrManager %in% out16, "YearsWithCurrManager"]= NA

# Imputing those NAs
EA2[10]<-impute(EA2[10],fun = median)
EA2[13]<-impute(EA2[13],fun = median)
EA2[14]<-impute(EA2[14],fun = median)
EA2[15]<-impute(EA2[15],fun = median)
EA2[16]<-impute(EA2[16],fun = median)

boxplot(EA2[c(1,5,10,12,13,14,15,16)])
skewness(EA2[c(1,5,10,12,13,14,15,16)])
kurtosis(EA2[c(1,5,10,12,13,14,15,16)])
```

```
> skewness(EA2[c(1,5,10,12,13,14,15,16)])
                     Age         DistanceFromHome           MonthlyIncome
               0.4128645                0.9571400               1.3684185
      PercentSalaryHike       TotalWorkingYears          YearsAtCompany
               0.8202899                1.1184985               1.7627284
YearsSinceLastPromotion    YearsWithCurrManager
               1.9822646                0.8326003
> kurtosis(EA2[c(1,5,10,12,13,14,15,16)])
                     Age         DistanceFromHome           MonthlyIncome
                2.593149                2.771852               3.997738
      PercentSalaryHike       TotalWorkingYears          YearsAtCompany
                2.696344                3.919605               6.918057
YearsSinceLastPromotion    YearsWithCurrManager
                6.596318                3.166398
```



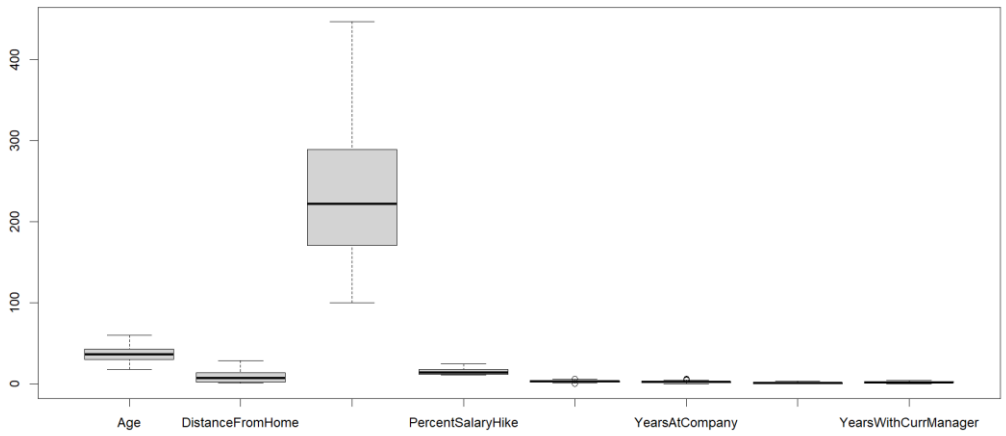
➔ **Step 5:** Applying Method 3 (Transformation of Outliers in EA2)

The third method for treating normality and outliers is the transformation of variables in data copy EA3. Transformation refers to the replacement of a variable by a function, which consists of square root transformation and lock transformation. As there are two methods in transformation, we created two data copies of EA3, named EA3a and EA3b.

We apply square root transformation on data copy EA3a and log transformation on data copy EA3b. After applying the square root transformation version, we see that the issue of normality is resolved, but there are still outliers in two columns.

```
# Creating copies of EA3
EA3a<-EA3
EA3b<-EA3

# Square Root Transformation in EA3a
EA3a$MonthlyIncome <- sqrt(EA3a$MonthlyIncome)
EA3a$TotalWorkingYears <- sqrt(EA3a$TotalWorkingYears)
EA3a$YearsAtCompany <- sqrt(EA3a$YearsAtCompany)
EA3a$YearsSinceLastPromotion <- sqrt(EA3a$YearsSinceLastPromotion)
EA3a$YearsWithCurrManager <- sqrt(EA3a$YearsWithCurrManager)
skewness(EA3a[c(1,5,10,12,13,14,15,16)])
kurtosis(EA3a[c(1,5,10,12,13,14,15,16)])
boxplot(EA3a[c(1,5,10,12,13,14,15,16)])
```

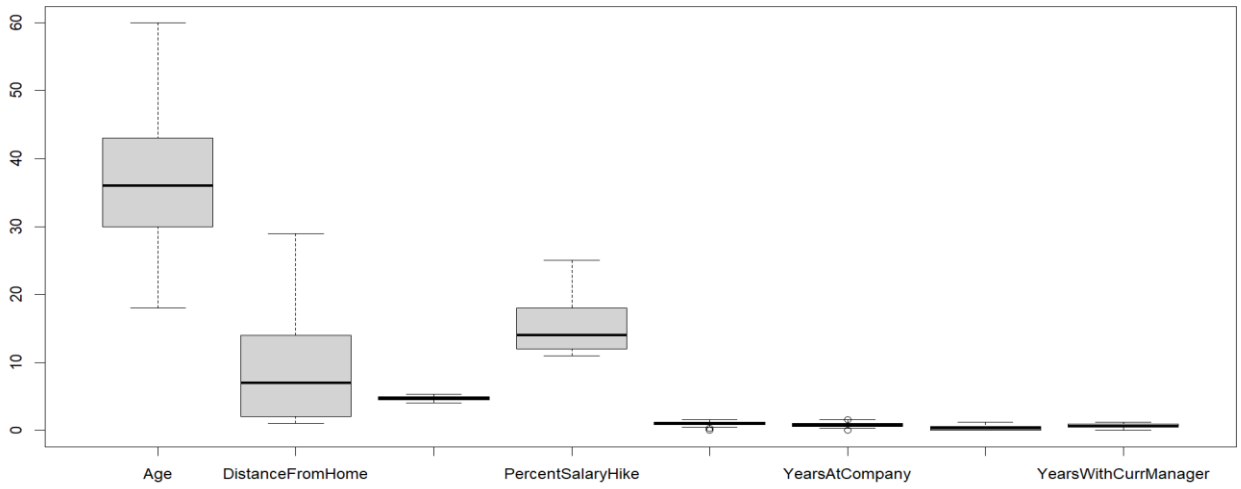
```
> skewness(EA3a[c(1,5,10,12,13,14,15,16)])
                     Age         DistanceFromHome            MonthlyIncome
               0.4128645                0.9571400                0.8610798
       PercentSalaryHike       TotalWorkingYears            YearsAtCompany
               0.8202899                0.1772671                0.4263989
YearsSinceLastPromotion    YearsWithCurrManager
               0.7383711               -0.2543509
> kurtosis(EA3a[c(1,5,10,12,13,14,15,16)])
                     Age         DistanceFromHome            MonthlyIncome
                2.593149                2.771852                2.882922
       PercentSalaryHike       TotalWorkingYears            YearsAtCompany
                2.696344                2.988924                3.380276
YearsSinceLastPromotion    YearsWithCurrManager
                2.652369                2.225705
```



While applying log transformation on data copy EA3b, we use the formula log10(x+1) as some values are equal to 0. After applying this transformation, we see that the issue of normality has been solved, but there are still some outliers.

```
# Log Transformation in EA3b
EA3b$MonthlyIncome <- log10(EA3b$MonthlyIncome)
EA3b$TotalWorkingYears <- log10(EA3b$TotalWorkingYears+1)
EA3b$YearsAtCompany <- log10(EA3b$YearsAtCompany+1)
EA3b$YearsSinceLastPromotion <- log10(EA3b$YearsSinceLastPromotion+1)
EA3b$YearsWithCurrManager <- log10(EA3b$YearsWithCurrManager+1)
skewness(EA3b[c(1,5,10,12,13,14,15,16)])
kurtosis(EA3b[c(1,5,10,12,13,14,15,16)])
boxplot(EA3b[c(1,5,10,12,13,14,15,16)])
```

```
> skewness(EA3b[c(1,5,10,12,13,14,15,16)])
                     Age         DistanceFromHome           MonthlyIncome       PercentSalaryHike
               0.4128645                0.9571400               0.2859002               0.8202899
       TotalWorkingYears           YearsAtCompany YearsSinceLastPromotion    YearsWithCurrManager
              -0.6217767               -0.2074959               0.7180709              -0.3573206
> kurtosis(EA3b[c(1,5,10,12,13,14,15,16)])
                     Age         DistanceFromHome           MonthlyIncome       PercentSalaryHike
               2.593149                2.771852               2.300765                2.696344
       TotalWorkingYears           YearsAtCompany YearsSinceLastPromotion    YearsWithCurrManager
               3.515201                2.713848               2.398191                2.090585
```

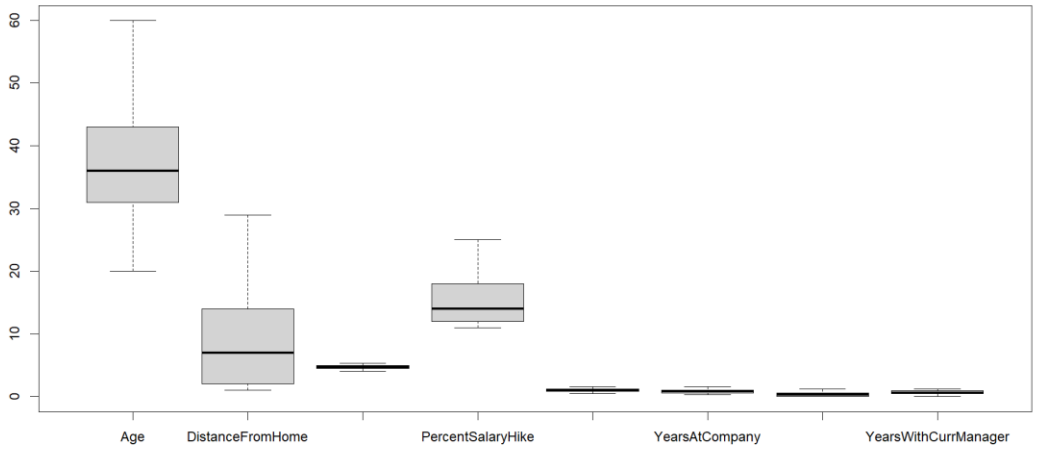


➔ **Step 6:** Performing Log Transformation & then Removing Outliers

In this particular step, we combined two different methods of treating normality and outliers, as the methods were not enough individually. We created a data copy of the log transformed data set (EA3b) and named it EA4. After creating the data copy, we stored the outliers left in columns **(TotalWorkingYears & YearsAtCompany)** into vectors out 13a and out 14a. We later remove these vectors of outliers from the dataset EA4. After combining the two methods, we finally have a normally distributed data set with no outliers, which means this is our final data set based on which we will be making all our machine learning models.

```
# Performing Log transformation & then removing outliers ----------------

# creating copy of log transformed data set
EA4<-EA3b

# storing outliers in vectors & checking number of outliers in each
boxplot(EA4[c(1,5,10,12,13,14,15,16)])
out13a<-boxplot(EA4$TotalWorkingYears)$out
length(out13a)
out14a<-boxplot(EA4$YearsAtCompany)$out
length(out14a)

# Removing the records with outliers
EA4<-EA4[-which(EA4$TotalWorkingYears %in% out13a),]
EA4<-EA4[-which(EA4$YearsAtCompany %in% out14a),]

# Checking normality and outliers
skewness(EA4[c(1,5,10,12,13,14,15,16)])
kurtosis(EA4[c(1,5,10,12,13,14,15,16)])
boxplot(EA4[c(1,5,10,12,13,14,15,16)])
```

```
> skewness(EA4[c(1,5,10,12,13,14,15,16)])
                    Age        DistanceFromHome           MonthlyIncome       PercentSalaryHike
             0.46487194              0.92276118              0.28791248              0.82791983
      TotalWorkingYears          YearsAtCompany YearsSinceLastPromotion    YearsWithCurrManager
             0.01013163              0.04576743              0.60871682             -0.47079453
> kurtosis(EA4[c(1,5,10,12,13,14,15,16)])
                    Age        DistanceFromHome           MonthlyIncome       PercentSalaryHike
               2.549793                2.679714                2.307532                2.702307
      TotalWorkingYears          YearsAtCompany YearsSinceLastPromotion    YearsWithCurrManager
               2.594497                2.587616                2.275235                2.493079
```

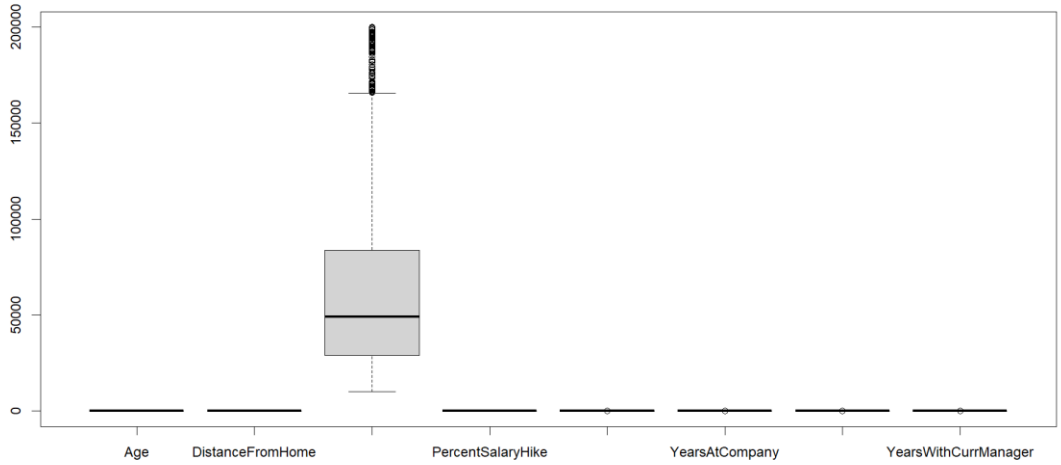


➔ **Step 7:** Performing Square Root Transformation & then Removing Outliers

In this particular step, we combined two different methods of treating normality and outliers, as the methods were not enough individually. We created a data copy of the sqrt transformed data set (EA3a) and named it EA5. After creating the data copy, we stored the outliers left in columns **(TotalWorkingYears & YearsAtCompany)** into vectors out13b and out14b. We later remove these vectors of outliers from the dataset EA5. After combining the two methods, we still have outliers; therefore, this data set isn't a fit for us.

```
# Performing Square root transformation & then removing outliers ----------
# creating copy of sqrt transformed data set
EA5<-EA3a
# storing outliers in vectors & checking number of outliers in each
boxplot(EA5[c(1,5,10,12,13,14,15,16)])
out13b<-boxplot(EA5$TotalWorkingYears)$out
length(out13b)
out14b<-boxplot(EA5$YearsAtCompany)$out
length(out14b)
# Removing the records with outliers
EA5<-EA5[-which(EA5$TotalWorkingYears %in% out13b),]
EA5<-EA5[-which(EA5$YearsAtCompany %in% out14b),]
# Checking normality and outliers
skewness(EA5[c(1,5,10,12,13,14,15,16)])
kurtosis(EA5[c(1,5,10,12,13,14,15,16)])
boxplot(EA[c(1,5,10,12,13,14,15,16)])
```

```
> skewness(EA5[c(1,5,10,12,13,14,15,16)])
                    Age        DistanceFromHome           MonthlyIncome
              0.4701691               0.9540519               0.8472972
      PercentSalaryHike       TotalWorkingYears          YearsAtCompany
              0.8141553               0.2056042               0.2047476
YearsSinceLastPromotion    YearsWithCurrManager
              0.7138918              -0.2569286
> kurtosis(EA5[c(1,5,10,12,13,14,15,16)])
                    Age        DistanceFromHome           MonthlyIncome
               2.679665               2.765764               2.859281
      PercentSalaryHike       TotalWorkingYears          YearsAtCompany
               2.683875               2.823275               2.866658
YearsSinceLastPromotion    YearsWithCurrManager
               2.616190               2.265447
```



Splitting the Data for Machine Learning

Data splitting is a fundamental step in the machine learning pipeline that involves dividing the available dataset into separate subsets for training and testing purposes. This process ensures that the model's performance is evaluated on unseen data, objectively assessing its capabilities.

The training dataset is used to create and train the machine-learning model. The model learns patterns and relationships between input features (independent variables) and the target variable (attrition in this case) using the data in the training set.

The testing dataset is used to evaluate the performance of the trained model. The model's predictions on the testing set are compared against the actual values to assess its accuracy and effectiveness in predicting new, unseen data.

```
# Splitting the final dataset into testing & traning
set.seed(100)
intrain<-createDataPartition(y = EA4$Attrition, p = 0.8, list = FALSE)
training<-EA4[intrain,]
testing<-EA4[-intrain,]
str(training)
str(testing)
```

1. **Setting Seed:** The set.seed() function in R initialises the random number generator with a specific seed value; here, we have taken 100. This allows for reproducibility in the analysis, meaning that the same observations are assigned to the training and testing sets every time we run the code.

2. **Data Partitioning:** The createDataPartition() function from the caret package splits the dataset into training and testing sets. The y argument specifies the target variable (Attrition), and the p argument determines the proportion of data allocated to the training set (80% in this case).

3. **Training and Testing Sets:** The intrain vector contains the indexes of the records in the training set, which are then used to subset the original dataset (EA4). The training dataset contains the observations for training the machine learning models, while the testing dataset contains the remaining observations for model evaluation.

If we look at the testing dataset, most instances fall into the negative class (No), while the positive class (Yes) is significantly smaller. Out of 809 observations in the testing dataset, 699 fall into the negative class and 110 fall into the positive class. This leads to data imbalance, leading to biased model performance, as the model may prioritise accuracy on the majority class at the expense of the minority class. Hence, we will also look at the balanced accuracy and precision of various models to compare them.

Creating Machine Learning Models Using various Algorithms

Since our dependent variable (Attrition) is categorical with two categories (Yes or No), we will employ ML techniques like logistic regression, Naive Bayes Classifier, Decision Tree and Random Forest.

Logistic Regression

About Binomial Logistic Regression:

Binomial logistic regression is a type of logistic regression used when the dependent variable has two categories or levels. Binomial logistic regression aims to model the probability that an observation falls into one of the two categories based on the values of the predictor variables.

Assumptions:

```
# ASSUMPTIONS FOR LOGISTIC REGRESSION
skewness(EA4[c(1,5,10,12,13,14,15,16)])
kurtosis(EA4[c(1,5,10,12,13,14,15,16)])
X <- EA4[c(1,5,10,12,13,14,15,16)]
rcorr(as.matrix(X))
```

- ➔ Normality of Data: We checked the skewness and kurtosis of the continuous numeric variables in the dataset - Age, DistanceFromHome, MonthlyIncome, NumCompaniesWorked, PercentSalaryHike, TotalWorkingYears, YearsAtCompany, and YearsSinceLastPromotion. The skewness of all the variables was within the recommended range of +-1, and the kurtosis of all the variables was within the recommended range of +- 4. Hence, the dataset is normally distributed.

- ➔ No serious outliers: Since we already used the treated dataset, no outliers were found.

- ➔ No multicollinearity: The rcorr() function from the Hmisc package is used to calculate the correlation matrix for the selected variables. When calculated, the correlation coefficient between each pair of the independent variables (continuous numeric variables) is less than 0.9 hence, there is no multicollinearity.

Model 1: Binomial Logistic Regression with all the Independent variables

```
# Model 1 - logistic regression
M1<-train(data=training,Attrition~.,method="glm", family="binomial")
summary(M1)
predAttritionLR1<-predict(M1,newdata=testing)
confusionMatrix(predAttritionLR1,testing$Attrition, positive = "Yes") #Positive Class is Yes
confusionMatrix(predAttritionLR1,testing$Attrition)
```

Creation of Model

- The train() function from the caret package is used to train/create the model.
- Attrition~. indicating that Attrition is the dependent variable and all other variables in the dataset are independent variables.
- method="glm" specifies that logistic regression is used as the modelling method.
- family="binomial" specifies the binomial family for logistic regression since the DV had two categories.

Summary

- summary() function is used to obtain a summary of the trained logistic regression model.
- The summary provides information about the model's coefficients, significance levels, and goodness-of-fit statistics based on the training data itself.

Checking the model's accuracy on the testing dataset

- predict() function is used to generate predictions for employee attrition on the testing dataset using the trained logistic regression model. These predicted values are stored in a vector called 'predAttritionLR1'
- confusionMatrix() function from the caret package is used to compute the confusion matrix and evaluate the performance of the logistic regression model.

Interpretation

This interpretation is in accordance with the confusion matrix where the positive class is taken as "Yes" for Attrition.

- ➔ Accuracy: The overall proportion of correct predictions made by the model. In this case, the accuracy is 86.4%, indicating that the model correctly classified 86.4% of all instances.
- ➔ Sensitivity (Recall): The proportion of true positives (TP) among all actual positives (TP +FN). It measures the model's ability to identify attrition cases correctly. In this case, the sensitivity is low at 3.6%, indicating that the model struggles to identify employees who will leave the company correctly.

- ➔ Specificity: The proportion of true negatives (TN) among all actual negatives (TN + FP). It measures the model's ability to identify non-attrition cases correctly. In this case, the specificity is high at 99.4%, indicating that the model correctly identifies employees who will not leave the company.
- ➔ Precision: The proportion of true positives (TP) among all instances predicted as positive (TP + FP). It measures the accuracy of positive predictions made by the model. In this case, the precision is moderate at 50%, indicating that when the model predicts attrition, it is correct 50% of the time.
- ➔ Balanced Accuracy: The average of sensitivity and specificity. It provides a balanced assessment of the model's performance across both classes. In this case, the balanced accuracy is low at 51.5%, reflecting the imbalance in performance between the two classes.

Model 2: Binomial Logistic Regression with only significant variables

When we ran the summary command in the previous model, we learned that only a few variables significantly impact the decision variables. We check this using the individual p-value associated with the variables. The p-value must be less than 0.05 for a variable to be a significant predictor. The significant variables in our dataset are:
BusinessTravel, Department, EducationField, MaritalStatus, NumCompaniesWorked, TotalWorkingYears, and YearsSinceLastPromotion.

Hence, in Model 2, we used only the significant predictors to predict attrition.

```
# Model 2 - logistic regression with only significant IVs
M2<-train(data=training,Attrition~BusinessTravel+Department+EducationField+MaritalStatus+NumCompaniesWorked+TotalWorkingYears+YearsSinceLastPromotion,method="glm", family="binomial")
summary(M2)
predAttritionLR2<-predict(M2,newdata=testing)
confusionMatrix(predAttritionLR2,testing$Attrition,positive = "Yes")
confusionMatrix(predAttritionLR2,testing$Attrition)
```

Creation of Model

- The train() function from the caret package is used to train/create the model.
- "Attrition ~ BusinessTravel + Department + EducationField + MaritalStatus + NumCompaniesWorked + TotalWorkingYears + YearsSinceLastPromotion" specifies the predictors used in the model, while Attrition is the DV.
- method="glm" specifies that logistic regression is used as the modelling method.
- family="binomial" specifies the binomial family for logistic regression since the DV had two categories.

Summary

- summary() function is used to obtain a summary of the trained logistic regression model.
- The summary provides information about the model's coefficients, significance levels, and goodness-of-fit statistics based on the training data itself.

Checking the model's accuracy on the testing dataset

- predict() function is used to generate predictions for employee attrition on the testing dataset using the trained logistic regression model. These predicted values are stored in a vector called 'predAttritionLR2'
- confusionMatrix() function from the caret package is used to compute the confusion matrix and evaluate the performance of the logistic regression model.

Interpretation

This interpretation is in accordance with the confusion matrix where the positive class is taken as "Yes" for Attrition.

- ➔ Accuracy: The overall proportion of correct predictions made by the model. In this case, the accuracy is 86.65%, indicating that the model correctly classified 86.65% of all instances.
- ➔ Sensitivity (Recall): The proportion of true positives (TP) among all actual positives (TP +FN). It measures the model's ability to identify attrition cases correctly. In this case, the sensitivity is low at 3.64%, indicating that the model struggles to identify employees who will leave the company correctly.
- ➔ Specificity: The proportion of true negatives (TN) among all actual negatives (TN + FP). It measures the model's ability to identify non-attrition cases correctly. In this case, the specificity is high at 99.71%, indicating that the model correctly identifies employees who will not leave the company.
- ➔ Precision: The proportion of true positives (TP) among all instances predicted as positive (TP + FP). It measures the accuracy of positive predictions made by the model. In this case, the precision is moderate at 66.67%, indicating that when the model predicts attrition, it is correct 66.67% of the time.
- ➔ Balanced Accuracy: The average of sensitivity and specificity. It provides a balanced assessment of the model's performance across both classes. In this case, the balanced accuracy is low at 51.68%, reflecting the imbalance in performance between the two classes.

Naive Bayes Algorithm

About Naive Bayes Algorithm:

It is based on Bayes theorem and is used to solve classification problems. It predicts the conditional probability of the categories of the DV given the values of the IVs (using the Bayes theorem). The "naive" assumption of independence between variables allows Naive Bayes to scale well and make predictions quickly, even with limited training data. However, this assumption may not always hold true in real-world datasets, leading to potential limitations in predictive performance.

Assumptions:

- ➔ It assumes that the independent variables are conditionally independent or unrelated to any of the other variables in the model. We checked for multicollinearity in the data and found the absence of multicollinearity. Hence, to proceed with this technique, we will consider this assumption to be checked based on multicollinearity itself.

Model 3: Using Naive Bayes Algorithm

```
# Model 3 - Naive Bayes
M3<- train(Attrition~., data=training, method="naive_bayes")
predAttritionNB1<-predict(M3, newdata = testing)
confusionMatrix(predAttritionNB1, testing$Attrition)
confusionMatrix(predAttritionNB1, testing$Attrition, positive = "Yes")
```

Creation of Model

- Model 3 uses the Naive Bayes algorithm, implemented through the train() function from the caret package.
- The formula Attrition ~. specifies that all independent variables (excluding the dependent variable) in the dataset will be used to predict the target variable Attrition.
- The method = "naive_bayes" argument specifies the Naive Bayes method for modelling.

Checking the accuracy of the model on the testing dataset

- The predict() function is used to generate predictions (predAttritionNB1) for employee attrition on the testing dataset (testing) using the trained Naive Bayes model (M3).
- confusionMatrix() function from the caret package is used to compute the confusion matrix and evaluate the model's performance.

Interpretation

This interpretation is in accordance with the confusion matrix where the positive class is taken as “Yes” for Attrition.

- ➔ Accuracy: The overall proportion of correct predictions made by the model. In this case, the accuracy is 86.4%, indicating that the model correctly classified 86.4% of all instances.
- ➔ Sensitivity (Recall): The proportion of true positives (TP) among all actual positives (TP +FN). It measures the model's ability to identify attrition cases correctly. In this case, the sensitivity is 0%, indicating that the model fails to identify any employees who will leave the company correctly.
- ➔ Specificity: The proportion of true negatives (TN) among all actual negatives (TN + FP). It measures the model's ability to identify non-attrition cases correctly. In this case, the specificity is high at 100%, indicating that the model correctly identifies employees who will not leave the company.
- ➔ Precision: The proportion of true positives (TP) among all instances predicted as positive (TP + FP). It measures the accuracy of positive predictions made by the model. Since there are no true positives (TP = 0), the positive predictive value is not a valid metric in this case (NaN).
- ➔ Balanced Accuracy: The average of sensitivity and specificity. It provides a balanced assessment of the model's performance across both classes. In this case, the balanced accuracy is low at 50%, reflecting the model's poor performance in predicting attrition cases.

This model, Model 3, based on Naive Bayes, shows perfect specificity but fails to identify any attrition cases (0 sensitivity) correctly. This indicates that while the model effectively predicts non-attrition cases, it completely misses identifying employees likely to leave the company. This imbalance in sensitivity and specificity suggests that the model may not be suitable for accurate attrition prediction in this context.

Decision Tree

About Decision Tree:

The Decision Tree algorithm is a powerful tool for classification tasks, as it creates a flowchart-like structure (tree) where each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label. A decision tree simply asks a question, and based on the answer (Yes/No), it further splits the treeinto subtrees.

Attribute selection measure (ASM) is used to select the variables (attributes) for decision nodes
Two popular techniques for ASM are:

1. Information Gain: It calculates how much information a variable provides about the DV. A decision tree algorithm always tries to maximise the information gain value, and a variable with the highest information gain is selected first.
2. Gini Index: The Gini index is a measure of impurity/uncertainty in a variable while predicting the DV. A decision tree algorithm always tries to minimise the value of the Gini Index, and a variable with the lowest value of the Gini Index is selected first.

Model 4: Using Decision Tree and Gini Index as the Attribute Selection Measure

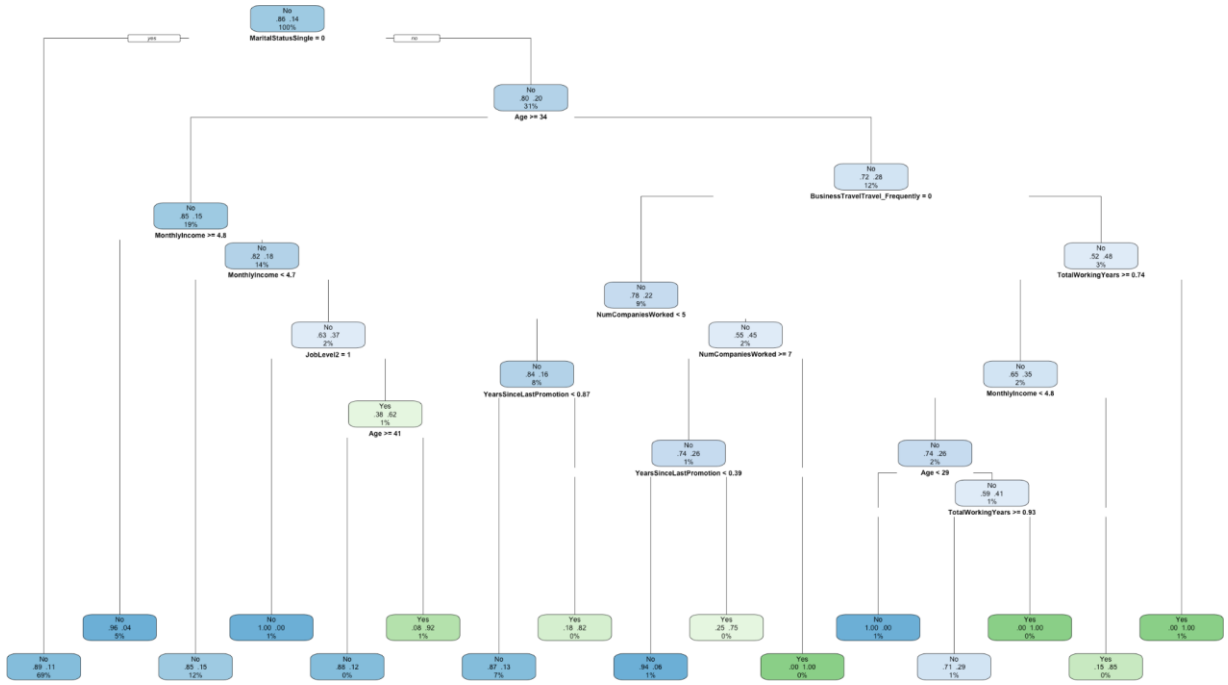
```
# Model 4 - Decision Tree - Gini Index
M4<-train(data=training,Attrition~., method="rpart")
rpart.plot(M4$finalModel, extra=104)
predAttritionDT1<-predict(M4, newdata = testing)
confusionMatrix(predAttritionDT1, testing$Attrition)
confusionMatrix(predAttritionDT1, testing$Attrition, positive = "Yes")
```

Creation of Model

- Model 4 uses the Decision Tree algorithm implemented through the train() function from the caret package.
- The formula Attrition ~. specifies that all independent variables (excluding the dependent variable) in the dataset will be used to predict the target variable Attrition.
- The method = "rpart" argument specifies the Decision Tree algorithm with the Gini Index as the Attribute Selection Measure for splitting nodes.

Visualisation

- The rpart.plot() function visualises the trained Decision Tree model. This function generates a plot of the decision tree, providing insights into how the model makes predictions based on different variables.



- Root Node:
  - MaritalStatusSingle=0: The root node is if the employee isn't single and then bifurcates into further decision nodes
- Decision Nodes:
  - In total, there are 16 decision nodes, including the first decision node, i.e. root node
- Leaf Nodes:
  - Yes: The employee is leaving the organisation, and there is attrition
  - No: The employee is not leaving the organisation, and there is no attrition
  - There are 16 leaf nodes with some labels as yes and some as no.

Checking the accuracy of the model on the testing dataset

- The predict() function is used to generate predictions (predAttritionDT1) for employee attrition on the testing dataset using the trained Decision Tree model (M4).

- confusionMatrix() function from the caret package is used to compute the confusion matrix and evaluate the model's performance.

Interpretation

This interpretation is in accordance with the confusion matrix where the positive class is taken as “Yes” for Attrition.

- ➔ Accuracy: The overall proportion of correct predictions made by the model. In this case, the accuracy is 87.52%, indicating that the model correctly classified 87.52% of all instances.
- ➔ Sensitivity (Recall): The proportion of true positives (TP) among all actual positives (TP +FN). It measures the model's ability to identify attrition cases correctly. In this case, the sensitivity is 13.64%, indicating that the model struggles to identify employees who will leave the company correctly.
- ➔ Specificity: The proportion of true negatives (TN) among all actual negatives (TN + FP). It measures the model's ability to identify non-attrition cases correctly. In this case, the specificity is high at 99.14%, indicating that the model correctly identifies employees who will not leave the company.
- ➔ Precision: The proportion of true positives (TP) among all instances predicted as positive (TP + FP). It measures the accuracy of positive predictions made by the model. In this case, the precision is moderate at 71.43%, indicating that when the model predicts attrition, it is correct 71.43% of the time.
- ➔ Balanced Accuracy: The average of sensitivity and specificity. It provides a balanced assessment of the model's performance across both classes. In this case, the balanced accuracy is moderate at 56.39%.

Model 5: Using Decision Tree and Information Gain as the Attribute Selection Measure

```
# Model 5 - Decision Tree - Information Gain
M5<-train(data=training,Attrition~., method="rpart",parms=list(split="information"))
rpart.plot(M5$finalModel)
predAttritionDT2<-predict(M5, newdata = testing)
confusionMatrix(predAttritionDT2, testing$Attrition)
confusionMatrix(predAttritionDT2, testing$Attrition, positive = "Yes")
```

Creation of Model

- Model 5 uses the Decision Tree algorithm implemented through the train() function from the caret package.
- The formula Attrition ~. specifies that all independent variables (excluding the dependent variable) in the dataset will be used to predict the target variable, Attrition.
- The method = "rpart" argument specifies the Decision Tree algorithm, and parms = list(split = "information") specifies Information Gain as the Attribute Selection Measure for splitting nodes.

Visualisation

- The rpart.plot() function visualises the trained Decision Tree model. This function generates a plot of the decision tree, providing insights into how the model makes predictions based on different variables.

  Note: the diagram was intense and unclear when we tried to take a screenshot. Hence, no picture is attached. Kindly refer to the R file.

- Root Node:
  - MaritalStatusSingle=1: The root node is if the employee is single
- Decision Nodes:
  - In total, there are 17 decision nodes, including the first decision node, i.e. root node
- Leaf Nodes:
  - Yes: The employee is leaving the organisation, and there is attrition
  - No: The employee is not leaving the organisation, and there is no attrition
  - In total, there are 18 leaf nodes with some labels as yes and some as no

Checking the accuracy of the model on the testing dataset

- The predict() function is used to generate predictions (predAttritionDT2) for employee attrition on the testing dataset using the trained Decision Tree model (M5).
- confusionMatrix() function from the caret package is used to compute the confusion matrix and evaluate the model's performance.

Interpretation

This interpretation is in accordance with the confusion matrix where the positive class is taken as “Yes” for Attrition.

➔ Accuracy: The overall proportion of correct predictions made by the model. In this case, the accuracy is 87.27%, indicating that the model correctly classified 87.27% of all instances.

- ➔ Sensitivity (Recall): The proportion of true positives (TP) among all actual positives (TP +FN). It measures the model's ability to identify attrition cases correctly. In this case, the sensitivity is 12.73%, indicating that the model struggles to identify employees who will leave the company correctly.
- ➔ Specificity: The proportion of true negatives (TN) among all actual negatives (TN + FP). It measures the model's ability to identify non-attrition cases correctly. In this case, the specificity is high at 98.99%, indicating that the model correctly identifies employees who will not leave the company.
- ➔ Precision: The proportion of true positives (TP) among all instances predicted as positive (TP + FP). It measures the accuracy of positive predictions made by the model. In this case, the precision is moderate at 66.67%, indicating that when the model predicts attrition, it is correct 66.67% of the time.
- ➔ Balanced Accuracy: The average of sensitivity and specificity. It provides a balanced assessment of the model's performance across both classes. In this case, the balanced accuracy is moderate at 55.86%.

Random Forest

About Random Forest:

Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. The "forest" it builds is an ensemble (collection) of decision trees, usually trained with the “bagging” method. The random forest algorithm aggregates the predictions of multiple decision trees of varying depth. Every decision tree in the forest is trained on a subset of the dataset called the bootstrapped (randomly selected) dataset.

Model 6: Using Random Forest

```
# Model 6 - Random Forest
M6 <- train(data=training,Attrition~., method="rf")
M6$finalModel
predAttritionRF <- predict(M6, newdata=testing)
confusionMatrix(predAttritionRF, testing$Attrition)
confusionMatrix(predAttritionRF, testing$Attrition, positive="Yes")
```

Creation of Model

- Model 6 uses the Random Forest algorithm implemented through the train() function from the caret package.

- The formula Attrition ~. specifies that all independent variables (excluding the dependent variable) in the dataset will be used to predict the target variable Attrition.
- The method = "rf" argument specifies the Random Forest algorithm for classification.

Checking the accuracy of the model on the testing dataset

- The trained Random Forest model (M6$finalModel) is displayed, providing information about the ensemble of decision trees created during training.
- It says that the model consists of 500 decision trees.
- At each split in the decision trees, the Random Forest model randomly selects a subset of variables to consider. In this case, 13 variables were tried at each split on average.
- The predict() function is used to generate predictions (predAttritionRF) for employee attrition on the testing dataset using the trained Random Forest model (M6).
- confusionMatrix() function from the caret package is used to compute the confusion matrix and evaluate the model's performance.

Interpretation

This interpretation is in accordance with the confusion matrix where the positive class is taken as “Yes” for Attrition.

- ➔ Accuracy: The overall proportion of correct predictions made by the model. In this case, the accuracy is exceptionally high at 99.63%, indicating that the model correctly classified 99.63% of all instances.
- ➔ Sensitivity (Recall): The proportion of true positives (TP) among all actual positives (TP + FN). It measures the model's ability to identify attrition cases correctly. In this case, the sensitivity is high at 97.27%, indicating that the model performs well in identifying employees who will leave the company.
- ➔ Specificity: The proportion of true negatives (TN) among all actual negatives (TN + FP). It measures the model's ability to identify non-attrition cases correctly. In this case, the specificity is perfect at 100%, indicating that the model correctly identifies employees who will not leave the company.
- ➔ Precision: The proportion of true positives (TP) among all instances predicted as positive (TP + FP). It measures the accuracy of positive predictions made by the model. In this case, the precision is perfect at 100%, indicating that it is always correct when the model predicts attrition.
- ➔ Balanced Accuracy: The average of sensitivity and specificity. It provides a balanced assessment of the model's performance across both classes. In this case, the balanced accuracy is very high at 98.64%.

Making ML models on Non-Transformed data using Non-Parametric algorithms

What is a non-parametric algorithm?

Non-parametric algorithms are those that do not assume any specific form or distribution for the underlying data. Unlike parametric algorithms, which make strong assumptions about the data distribution, non-parametric methods can adapt to any structure present in the data. Decision trees are a classic example of non-parametric algorithms because they can model complex relationships without requiring the data to fit a predefined shape.

Dataset Preparation

```
# Let's create a copy of the original dataset
EA_copy <- EA

# Splitting the final dataset into testing & traning
set.seed(100)
intrainC <-createDataPartition(y = EA_copy$Attrition, p = 0.8, list = FALSE)
trainingC <-EA_copy[intrain,]
testingC <-EA_copy[-intrain,]
str(trainingC)
str(testingC)
```

- **EA_copy:** A copy of the original dataset is created to ensure that any transformations or manipulations do not affect the original data.
- **Data Splitting:** The data is split into training (80%) and testing (20%) sets using a random seed for reproducibility. `createDataPartition` ensures that the splitting is stratified based on the `Attrition` variable, maintaining the proportion of attrition cases in both training and testing sets.

Model 7: Using Decision Tree and Gini Index as the Attribute Selection Measure

```
# Model 7 - Decision Tree - Gini Index
M7<-train(data=trainingC,Attrition~., method="rpart")
rpart.plot(M7$finalModel)
predAttritionDT3<-predict(M7, newdata = testingC)
confusionMatrix(predAttritionDT3, testingC$Attrition)
confusionMatrix(predAttritionDT3, testingC$Attrition, positive = "Yes")
```

Creation of Model

- Model 7 uses the Decision Tree algorithm with Gini Index as the attribute selection measure, implemented through the train() function from the caret package.
- The formula Attrition ~ . specifies that all independent variables (excluding the dependent variable) in the dataset will be used to predict the target variable Attrition.
- The method = "rpart" argument specifies the Decision Tree method for modelling.
- The decision tree is plotted using the rpart.plot() function to visualise the splits and decision paths.

Checking the accuracy of the model on the testing dataset

- The predict() function is used to generate predictions (predAttritionDT3) for employee attrition on the testing dataset (testingC) using the trained Decision Tree model (M7).
- Confusion matrices are generated to evaluate the performance of the Decision Tree model. The confusionMatrix() call with positive = "Yes" focuses on evaluating the model's performance specifically for predicting attrition cases (where the positive class is "Yes").

Interpretation

This interpretation is in accordance with the confusion matrix where the positive class is taken as “Yes” for Attrition.

- ➔ Accuracy: The overall proportion of correct predictions made by the model. In this case, the model achieves an accuracy of 83.8%, indicating that it correctly predicts 83.8% of the instances in the testing dataset.
- ➔ Sensitivity (Recall): The sensitivity is 0.000, meaning the model fails to correctly identify any actual attrition cases (True Positives). This is a critical issue as it indicates the model's inability to detect attrition.
- ➔ Specificity: The specificity is 1.000, meaning the model correctly identifies all non-attrition cases. This indicates a perfect prediction for the negative class (No attrition).
- ➔ Precision: The proportion of true positives (TP) among all instances predicted as positive (TP + FP). It measures the accuracy of positive predictions made by the model. In this case, this value is NaN because the model never predicted the positive class (attrition), resulting in undefined precision.
- ➔ Balanced Accuracy: The average of sensitivity and specificity. It provides a balanced assessment of the model's performance across both classes. In this case, the balanced accuracy is 0.500. The low balanced accuracy reflects the model's poor performance in predicting the positive class (attrition).

Model 8: Using Decision Tree and Information Gain as the Attribute Selection Measure

```
# Model 8 - Decision Tree - Information Gain
M8<-train(data=trainingC,Attrition~., method="rpart",parms=list(split="information"))
rpart.plot(M8$finalModel, extra=104)
predAttritionDT4<-predict(M8, newdata = testingC)
confusionMatrix(predAttritionDT4, testingC$Attrition, positive = "Yes")
```

Creation of Model

- Model 8 uses the Decision Tree algorithm with Information Gain as the attribute selection measure. The model is created using the train() function from the caret package.
- The formula Attrition ~ . specifies that all independent variables (excluding the dependent variable) in the dataset will be used to predict the target variable Attrition.
- The method="rpart" specifies the use of Decision Tree method, and parms=list(split="information") sets the attribute selection measure to Information Gain.

Checking the accuracy of the model on the testing dataset

- The predict() function generates predictions (predAttritionDT4) for employee attrition on the testing dataset (testingC) using the trained Decision Tree model (M8).
- The confusionMatrix() function evaluates the performance of the Decision Tree model with a focus on the positive class "Yes" for attrition.

Interpretation

This interpretation is in accordance with the confusion matrix where the positive class is taken as “Yes” for Attrition.

- ➔ Accuracy: The overall proportion of correct predictions made by the model. In this case, the model achieves an overall accuracy of 83.8%, meaning it correctly predicts 83.8% of the instances in the testing dataset.
- ➔ Sensitivity (Recall): The sensitivity is 0.000, meaning the model fails to correctly identify any actual attrition cases (True Positives). This is a critical issue as it indicates the model's inability to detect attrition.

- ➔ Specificity: The specificity is 1.000, meaning the model correctly identifies all non-attrition cases. This indicates a perfect prediction for the negative class (No attrition).
- ➔ Precision: The proportion of true positives (TP) among all instances predicted as positive (TP + FP). It measures the accuracy of positive predictions made by the model. In this case, this value is NaN because the model never predicted the positive class (attrition), resulting in undefined precision.
- ➔ Balanced Accuracy: The average of sensitivity and specificity. It provides a balanced assessment of the model's performance across both classes. In this case, the balanced accuracy is 0.500. The low balanced accuracy reflects the model's poor performance in predicting the positive class (attrition).

Model 9: Using Random Forest

```
# Model 9 - Random Forest
M9 <- train(data=trainingC,Attrition~., method="rf")
M9$finalModel
predAttritionRF2 <- predict(M9, newdata=testingC)
confusionMatrix(predAttritionRF2, testingC$Attrition, positive="Yes")
```

Creation of Model

- Model 9 uses the Random Forest algorithm. The model is created using the train() function from the caret package.
- The formula Attrition ~ . specifies that all independent variables (excluding the dependent variable) in the dataset will be used to predict the target variable Attrition.
- The method="rf" specifies the use of the Random Forest method for modelling.

Checking the accuracy of the model on the testing dataset

- The trained Random Forest model (M9$finalModel) is displayed, providing information about the ensemble of decision trees created during training.
- It says that the model consists of 500 decision trees.
- At each split in the decision trees, the Random Forest model randomly selects a subset of variables to consider. In this case, 13 variables were tried at each split on average.
- The predict() function is used to generate predictions (predAttritionRF2) for employee attrition on the testing dataset using the trained Random Forest model (M9).
- confusionMatrix() function from the caret package is used to compute the confusion matrix and evaluate the model's performance.

Interpretation

This interpretation is in accordance with the confusion matrix where the positive class is taken as “Yes” for Attrition.

- ➔ Accuracy: The overall proportion of correct predictions made by the model. In this case, the model achieves an accuracy of 84.34%, indicating that it correctly predicts 84.34% of the instances in the testing dataset.
- ➔ Sensitivity (Recall): The sensitivity is 0.041727, meaning the model correctly identifies only 4.17% of actual attrition cases (True Positives). This is quite low, showing that the model struggles to detect attrition.
- ➔ Specificity: The specificity is 0.998331, indicating the model correctly identifies 99.83% of non-attrition cases.
- ➔ Precision: The proportion of true positives (TP) among all instances predicted as positive (TP + FP). It measures the accuracy of positive predictions made by the model. In this case, the precision is 0.828571, meaning that when the model predicts attrition, it is correct 82.86% of the time.
- ➔ Balanced Accuracy: The average of sensitivity and specificity. It provides a balanced assessment of the model's performance across both classes. In this case, the balanced accuracy is 0.520029 - 52%, which is the average of sensitivity and specificity. The low balanced accuracy reflects the model's poor performance in predicting the positive class (attrition).

Evaluation & Comparison of various models

MODEL	ACCURACY	SENSITIVITY	SPECIFICITY	PRECISION	BALANCED ACCURACY
Model 1	86.4%	3.64%	99.43%	50.00%	51.42%
Model 2	86.65%	3.64%	99.71%	66.67%	51.68%
Model 3	86.4%	0.00%	100.00%	N/A	50.00%
Model 4	87.52%	13.64%	99.14%	71.43%	56.39%
Model 5	87.27%	12.73%	98.99%	66.67%	55.86%
Model 6	99.63%	97.27%	100.00%	100.00%	98.64%
Model 7	83.8%	0.00%	100.00%	N/A	50.00%
Model 8	83.8%	0.00%	100.00%	N/A	50.00%
Model 9	84.34%	4.17%	99.83%	82.85%	52.00%

- Accuracy: Model 6 (Random Forest) achieved the highest accuracy of 99.63%, indicating the proportion of correct predictions made by the model across all classes.
- Sensitivity: Model 6 (Random Forest) achieved the highest sensitivity of 97.27%, indicating its ability to correctly identify employees who left the company (attrition cases) among all actual attrition cases.
- Specificity: Model 6 (Random Forest) achieved a perfect specificity of 100.00%, indicating its ability to correctly identify employees who did not leave the company (non-attrition cases) among all actual non-attrition cases.
- Precision: Model 6 (Random Forest) achieved a perfect precision of 100.00%, indicating that when it predicts attrition, it is always correct.
- Balanced Accuracy: Model 6 (Random Forest) achieved the highest balanced accuracy of 98.64%, the average of sensitivity and specificity, providing a balanced assessment of the model's performance across both classes.

Result

Based on the comparison of performance metrics, it is evident that Model 6, which utilises the Random Forest algorithm, outperforms the other models regarding accuracy, sensitivity, specificity, precision, and balanced accuracy. With a nearly perfect accuracy of 99.63% and excellent sensitivity and specificity, Model 6 demonstrates superior predictive performance for attrition prediction in the context of ABC Company. The Random Forest algorithm's ability to handle complex relationships in the data and mitigate overfitting using an ensemble of decision trees makes it the best choice for accurately identifying employees at risk of leaving the company. Therefore, Model 6 is recommended as the most suitable model for predicting employee attrition and informing HR decision-making processes in ABC Company.