MD2201 Data Science
Course Project

S.No	Div	Batch No	Group No	Roll No	Gr.No	Name of Student
1	B	2	5	35	12210570	Pratik Mandalkar
2				36	12210569	Pratik Mane
3				38	12210540	Netra Mohekar
4				55	12210864	Pranav Patel
5				60	12211463	Harshwardhan Patil

Science

1. **Project Title:** Employee Retention

2. **Data Set Name:** IBM HR Analytics Employee Attrition & Performance

3. **Data Set Source:** Kaggle

4. **Data set Link:** https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset/

5. **Data Set Description:**

Number of Rows: 1470
Number of Columns: 35
No missing values (all columns have zero missing values)
Class imbalance exists (Attrition column seems to have imbalanced classes)

Data Types of Each Variable:

Age: Integer
Attrition: Categorical (Yes/No)
BusinessTravel: Categorical (Travel Rarely, Travel Frequently, Non-Travel)
DailyRate: Integer
Department: Categorical (Research & Development, Sales, Human Resources)
DistanceFromHome: Integer
Education: Integer
EducationField: Categorical (Life Sciences, Medical, Marketing, Technical Degree, Other)
EmployeeCount: Integer (seems constant, possibly insignificant)
EmployeeNumber: Integer (possibly an identifier)
EnvironmentSatisfaction: Integer
Gender: Categorical (e.g., Male, Female)
HourlyRate: Integer
JobInvolvement: Integer
JobLevel: Integer
JobRole: Categorical (Sales Executive, Research Scientist, Laboratory Technician, Manufacturing Director, Healthcare Representative, Manager, Sales Representative, Research Director, Human Resources)
JobSatisfaction: Integer
MaritalStatus: Categorical (Single, Married, Divorced)
MonthlyIncome: Integer
MonthlyRate: Integer
NumCompaniesWorked: Integer
Over18: Categorical (e.g., Y)
OverTime: Categorical (e.g., Yes, No)
PercentSalaryHike: Integer
PerformanceRating: Integer

RelationshipSatisfaction: Integer
StandardHours: Integer (seems constant, insignificant)
StockOptionLevel: Integer
TotalWorkingYears: Integer
TrainingTimesLastYear: Integer
WorkLifeBalance: Integer
YearsAtCompany: Integer
YearsInCurrentRole: Integer
YearsSinceLastPromotion: Integer
YearsWithCurrManager: Integer

Categories of Variables:

- Categorical Variables: BusinessTravel, Department, EducationField, Gender, JobRole, MaritalStatus, Over18, OverTime
- Binary Variables: Attrition, Over18, OverTime
- Numeric Variables: Age, DailyRate, DistanceFromHome, Education, EmployeeCount, EmployeeNumber, EnvironmentSatisfaction, HourlyRate, JobInvolvement, JobLevel, JobSatisfaction, MonthlyIncome, MonthlyRate, NumCompaniesWorked, PercentSalaryHike, PerformanceRating, RelationshipSatisfaction, StandardHours, StockOptionLevel, TotalWorkingYears, TrainingTimesLastYear, WorkLifeBalance, YearsAtCompany, YearsInCurrentRole, YearsSinceLastPromotion, YearsWithCurrManager

Missing Values:

No missing values in any column.

Class Imbalance:

The Attrition column indicates class imbalance, with "Yes" (indicating attrition) and "No" classes. The "Attrition" column indicates a significant class imbalance:
"Yes" values: 237 instances.
"No" values: 1233 instances.

6. Description of Work Done:

The project's workflow was meticulously designed to prepare and optimize a dataset for predictive modeling. It began with comprehensive data collection efforts to acquire a dataset suitable for analysis. Subsequent steps focused on stringent data preprocessing techniques, which included identifying and addressing null or NA values and managing class imbalance through various methodologies such as random oversampling, random undersampling, and their combinations, alongside the utilization of OVUM techniques. Additionally, outlier detection and removal were conducted using the z-score method to enhance dataset integrity. Through normalization procedures, the data distribution across features was standardized, ensuring uniformity and facilitating accurate model training.

The dataset was then partitioned into training and testing subsets using multiple splits, namely 70/30, 75/25, and 80/20, to accommodate diverse modeling strategies. Following this, six distinct algorithms—kNN, logistic regression, naive Bayes, decision tree, random forest, and SVM—were implemented for model training. Furthermore, various feature selection techniques, including CFS, chi-square, Lasso, RFE, importance score, genetic algorithm, and information gain, were systematically investigated to identify the most discriminative features for modeling purposes.

Finally, to optimize model performance and enhance predictive accuracy, hyperparameter tuning was conducted across all algorithms. This methodical approach ensured the meticulous preparation and refinement of the dataset, laying a solid foundation for robust predictive modeling and informed decision-making processes.

7. Literature Survey:

i. Intelligent Employee Retention System for Attrition Rate Analysis and Churn Prediction:
Journal: JGIM
Dataset used: Consumer goods (FMCG) company.
Data preprocessing: Employ CTC level, promotion, No NA values, Fuzzy-Analytical Hierarchy Process
Feature Selection: Employee Satisfaction, Appraisal_rating, Number of projects/tasks assigned per quarter, Time spent per project per quarter, Promotion, Employee_CTC_level, Safety measure
Algorithm: Random forest and gradient boosting, Deep neural network (DNN)

Evaluation Parameters: Accuracy
Findings: Employee turnover status

ii. HR Analytics: Employee Attrition Analysis using Random Forest:
Journal: IJPE
Dataset used: A real dataset provided by IBM analytics
Data preprocessing: SMOTE
Feature Selection: Current role, years with current manager, monthly income & total working years, age and total working years, percent salary hike, and performance rating
Algorithm: Random forest
Evaluation Parameters: Accuracy, CV score, Sensitivity, Specificity, ROC score

iii. Predicting Employee Attrition Using Machine Learning Techniques:
Journal: MDPI
Dataset used: A real dataset provided by IBM analytics
Data preprocessing: Not specified
Feature Selection: Age, Monthly income, Attrition, Monthly rate Business travel, Number of previous employers
Algorithm: Gaussian Naïve Bayes, Naïve Bayes classifier for multivariate Bernoulli models, Logistic Regression classifier, K-nearest neighbours (K-NN)
Evaluation Parameters: Gaussian Naïve Bayes confusion matrix
Findings: Monthly income, age, overtime, and distance from home were important predictors, with Gaussian Naïve Bayes classifier showing the best performance.

iv. From Big Data to Deep Data to Support People Analytics for Employee Attrition Prediction:
Journal: IEEE Access
Dataset used: IBM HR analytics attrition dataset, Human resource analytics Kaggle Dataset
Data preprocessing: RFE, SelectKBest
Feature Selection: Not specified
Algorithm: DT, LR, SVM, DNN, LSTM, CNN, RF, XGB, VC, Stacked
Evaluation Parameters: Accuracy
Findings: VC achieved the highest accuracy of 98%.

v. Predicting Employee Attrition Using Machine Learning Approaches:
Journal: Applied Sciences
Dataset used: IBM HR analytics attrition dataset
Data preprocessing: Not specified
Feature Selection: Max-out
Algorithm: SVM, LR, ETC, DTC
Evaluation Parameters: SVM-88%, ETC-93%, LR-74%, DTC-84%

vi. A Machine Learning Application for Human Resource:
Journal: Verlag Berlin Heidelberg
Dataset used: Not mentioned
Data preprocessing: No outliers
Feature Selection: Distributed algorithm, linear programming model, goal programming model, Fuzzy Model
Algorithm: Fuzzy modeling
Evaluation Parameters: Task-Arrange or Hungarian Algorithm gives the best allocation.

vii. Employee attrition prediction using logistic regression:
Journal: IJRASET
Dataset used: Kaggle (IBM HR Analytics)
Data preprocessing: Not specified
Feature Selection: Age, job satisfaction, monthly income, years
Algorithm: Naive Bayes, K-NN, logistic regression

Evaluation Parameters: Accuracy rate

viii. Study and Prediction Analysis of the Employee Turnover using Machine Learning Approaches:
Journal: IEEE Xplore
Dataset used: IBM Watson Analytics1
Data preprocessing: "dropna()" in pandas
Feature Selection: Not specified
Algorithm: Gradient Booster, KNN, Logistic, Naïve Bays, Random F., SVC
Evaluation Parameters: Precision and Accuracy

ix. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities:
Journal: Journal of Internet Services and Applications
Dataset used: NSF TeraGrid dataset, Internet traffic collected at the POP of an ISP network
Data preprocessing: Outliers different in different dataset
Feature Selection: Max, Min, Avg load observed in past 10 s ∼30 s, Link load observed at τ time scale, N past days hourly traffic volume, Past measurements
Algorithm: Supervised MLP-NN
Evaluation Parameters: Confusion matrix, accuracy, precision, recall, and f measure

x. A Study on Employee Retention as a Tool for Improving Organizational Effectiveness:
Journal: IJMTS
Dataset used: Not mentioned
Data preprocessing: No outliers
Feature Selection: Pearson Chi-Square, Continuity Correction, Likelihood Ratio, Fisher's Exact Test, Linear-by-Linear, Association N of Valid Cases
Algorithm: Not specified
Evaluation Parameters: Accuracy

xi. Employee retention prediction in corporate organizations using machine learning methods:
Journal: Academy of Entrepreneurship Journal
Dataset used: Employment history of 613 employees (specific name is not mentioned)
Data preprocessing: Not specified
Feature Selection: Not specified
Algorithm: Random Forest, Support Vector Mechanism, Decision Trees, K-Nearest Neighbour, and Logistic Regression
Evaluation Parameters: Accuracy, precision, recall, and F1 score
Findings: The OOB estimate error rate = 0.36%, AUC = 0.9927, confidence level for the model = 95%

xii. Predictive Analytics of Employee Attrition using K-Fold Methodologies:
Journal: ijmsc
Dataset used: Kaggle (no specific name mentioned)
Data preprocessing: Not specified
Feature Selection: Not specified
Algorithm: Cat Boost, Random Forest (RF), Light GBM
Evaluation Parameters: ROC curves
Findings: Accuracy of 83.67% for LightBoost, 70-30 split dataset

xiii. Predict Employee Retention Using Data Science:
Journal: ijeecse
Dataset used: Not mentioned
Data preprocessing: Deleting the entire row
Feature Selection: Not specified
Algorithm: Regression analysis
Evaluation Parameters: Confusion matrix and accuracy score
Findings: Predict employee retention with an outcome of 97%

xiv. Intelligent Employee Retention System for Attrition Rate Analysis and Churn Prediction:
Journal: Journal of Global Information Management
Dataset used: Not mentioned
Data preprocessing: Not mentioned
Feature Selection: Random Forest, Gradient Boost, and Deep Neural Network (DNN)
Algorithm: Fuzzy Analytical Hierarchy Process, confusion matrix, outcome of NNM
Evaluation Parameters: Adjusted R-squared value of 0.79 suggests that 79% of the variance

xv. Predicting and explaining employee turnover intention:
Journal: International Journal of Data Science and Analytics
Dataset used: IBM HR analytics attrition dataset, Human resource analytics Kaggle Dataset
Data preprocessing: Mode imputation
Feature Selection: Not specified
Algorithm: LR, LGBM, XAI, DT, KNN
Evaluation Parameters: Not specified

xvi. An employee retention model using organizational network analysis for voluntary turnover:
Journal: Social Network Analysis and Mining
Dataset used: Not mentioned
Data preprocessing: Not specified
Feature Selection: Not specified
Algorithm: Not specified
Evaluation Parameters: Not specified

xvii. Employee Performance Prediction Using EPP Framework:
Journal: IJSRCSEIT
Dataset used: Not mentioned
Data preprocessing: Not specified
Feature Selection: Not specified
Algorithm: Naive Bayes
Evaluation Parameters: Not specified
A Study on Employee Retention as a Tool for Improving Organizational Effectiveness:

xviii. Journal: IJMTS
Dataset used: Not mentioned
Data preprocessing: Not specified
Feature Selection: Not specified
Algorithm: Not specified
Evaluation Parameters: Accuracy

xix. Identifying factors for employee retention using computational techniques: an approach to assist the decision-making process:
Journal: SN Applied Sciences
Dataset used: Not mentioned
Data preprocessing: Not specified
Feature Selection: Not specified
Algorithm: Data mining
Evaluation Parameters: Not specified

8. Data Preprocessing:

I. Handle Missing Values

In the datasets utilized for this analysis, there are no missing values present. Despite the absence of missing data, understanding methods for handling them is still crucial for future datasets where missing values may arise. This preemptive

knowledge ensures the ability to effectively manage and analyze data, maintaining the integrity and reliability of results. Common strategies for handling missing data include data imputation, where missing values are estimated based on existing data, and removal of missing values, though unnecessary in this particular dataset. By being prepared to address missing data challenges, researchers can ensure robust and accurate analyses, even in the absence of missing values in the initial dataset.

II. Encode Categorical Variables

In the process of preparing the dataset for analysis, one crucial step involves encoding categorical variables. This process aims to convert categorical data into a numerical format that can be easily interpreted by analytical models. Initially, the categorical columns within the dataset are identified, including attributes such as "Department," "BusinessTravel," "EducationField," "Gender," "JobRole," "MaritalStatus," and "OverTime." Subsequently, binary responses in the "Attrition" column are transformed into numeric values to facilitate analysis; "Yes" is encoded as 1, while "No" is encoded as 0. Additionally, all specified categorical columns, including "Attrition," are converted into factors. This conversion ensures that each categorical variable is represented as a discrete factor with distinct levels, enabling the dataset to be effectively utilized in subsequent analytical processes. Through the encoding of categorical variables, the dataset becomes structured and ready for further exploration, enabling the discovery of insights and patterns within the data.

III. Handle Class Imbalance

Handling class imbalance in machine learning datasets is crucial for ensuring model performance isn't skewed by disproportionate class representation. Various techniques address this challenge:

Random oversampling involves duplicating minority class instances randomly until balance is achieved. This boosts minority class representation without losing information, mitigating bias toward the majority class. Conversely, random undersampling involves randomly discarding majority class instances to restore balance. While computationally efficient, it risks discarding valuable data and reducing model accuracy.

A combined approach merges oversampling of the minority class with undersampling of the majority class. This method aims to balance the dataset while minimizing the drawbacks of individual oversampling and undersampling techniques. Specialized oversampling techniques like SMOTE synthesize new minority class instances by interpolating between existing ones, reducing overfitting risks associated with random oversampling. Meanwhile, undersampling methods like NearMiss select majority class instances closest to minority class instances, reducing the imbalance ratio while preserving majority class structure. By implementing these methods, practitioners can effectively tackle class imbalance, enhancing model performance and ensuring more accurate predictions across all classes.

IV. Normalize Data

In the project, the normalization technique used is Z-score normalization, also known as standardization. This technique is applied to numerical features within the dataset to ensure that they are on a comparable scale, with a mean of 0 and a standard deviation of 1.
The normalization process involves the following steps:

a. Calculation of Z-scores:

For each numerical feature, the Z-score is calculated. This is done by subtracting the mean of the feature from each data point and then dividing by the standard deviation of the feature. The formula for calculating the Z-score of a data point x in a feature is:

$$Z = \frac{x - mean}{Standard\ Deviation}$$

The Z-score represents the number of standard deviations away from the mean each data point is.

b. Normalization of Data:

By applying the Z-score formula to each numerical feature, the values are transformed to have a mean of 0 and a standard deviation of 1. This standardizes the numerical features and brings them to a consistent scale.

c. Normalization Impact:

Normalization using Z-score ensures that numerical features with different units or scales are treated equally in the analysis and modeling process. It prevents features with larger magnitudes from dominating the analysis and helps avoid biases in model performance.

d. Implementation in the Project:

In the project, Z-score normalization is applied to the dataset's numerical features using the scale function in R. This function

calculates the Z-score for each numerical column and standardizes the data accordingly.
By normalizing the numerical features using Z-score normalization, the project ensures that the data is appropriately scaled and ready for analysis and modeling. This preprocessing step enhances the performance and interpretability of machine learning models by eliminating potential biases arising from differences in feature scales.
The effectiveness of Z-score normalization can be observed through the reduction in the number of rows after outlier handling. For the over-sampled dataset, the number of rows decreased from 2418 to 2260 after applying the Z-score method. Similarly, for both over and under-sampled datasets, the number of rows reduced from 1000 to 938. This reduction indicates that outliers, identified using Z-score normalization, were successfully handled, leading to a more refined dataset for subsequent analysis and modeling.

V. Validate Data Integrity

Validating data integrity is a crucial aspect of data analysis, ensuring that the dataset is reliable, accurate, and free from errors. This process involves several key steps to confirm the completeness, consistency, and correctness of the data. Firstly, a completeness check is performed to ensure that all expected data fields are present and populated within the dataset, guarding against missing or incomplete information that could skew analysis results. Next, consistency checks are conducted to verify that data values adhere to predefined rules or constraints, such as valid formats for numeric values or correct categorization for categorical variables. Moreover, accuracy checks are carried out to validate data entries against external sources or known standards, identifying any discrepancies or outliers that may impact analysis outcomes. Finally, cross-field validation ensures that relationships between different data fields are logical and consistent, further enhancing the reliability of the dataset. By systematically validating data integrity through these measures, analysts can trust the quality of the dataset and the validity of insights derived from it, thereby bolstering the credibility and usefulness of subsequent analyses or models.

9. Feature Selection:

I. *C*orrelation-Based Feature Selection (CFS)

The Correlation-based Feature Selection (CFS) method is a feature selection technique that evaluates the relevance and redundancy of features based on their correlation with the target variable and with each other. CFS is implemented to select the most informative features for predicting employee attrition. It calculates the correlation of each feature with the target variable ("Attrition") and evaluates the redundancy among features. The selected features, determined to be the most relevant and least redundant, are then printed as the final output. This process aids in identifying key predictors of employee attrition while minimizing overfitting and maximizing predictive accuracy.

Features Selected:
"Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel", "MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager"

II. Chi Square (χ^2) test

The provided code implements the chi-square test for feature selection. In short, the chi-square test assesses the independence between categorical variables, making it suitable for identifying relevant features in classification tasks. Specifically, the chi_square function calculates the chi-square statistic for each feature, measuring its association with the target variable (Attrition). The features are then sorted based on their chi-square scores, and the top features are selected for further analysis. Finally, the selected features are printed as the output. This approach helps in identifying the most informative features for predicting employee attrition in the dataset.

Features Selected:
"MonthlyRate", "MonthlyIncome", "DailyRate", "Age", "TotalWorkingYears", "JobRole", "YearsAtCompany", "OverTime", "HourlyRate", "JobLevel"

III. Information Gain

The code uses the Information Gain method for feature selection. Information Gain quantifies how much a feature contributes to predicting the target variable by measuring the reduction in uncertainty (entropy) of the target when the feature is known. In the code, the information.gain function evaluates the information gain of each feature with respect to predicting attrition in the dataset. The resulting selected_features object contains the information gain values for each feature, aiding in the identification of the most predictive features for employee attrition.

Features Selected:
"Age", "Department", "BusinessTravel", "DistanceFromHome", "EducationField", "EnvironmentSatisfaction", "Gender",

"HourlyRate", "JobInvolvement", "JobLevel", "JobRole", "JobSatisfaction", "MaritalStatus", "MonthlyIncome", "MonthlyRate", "NumCompaniesWorked", "OverTime", "StockOptionLevel", "TotalWorkingYears", "WorkLifeBalance", "YearsAtCompany", "YearsInCurrentRole", "YearsWithCurrManager"

IV. Lasso

The code utilizes LASSO (Least Absolute Shrinkage and Selection Operator) for feature selection. LASSO introduces a penalty term to encourage sparse coefficient vectors, effectively performing feature selection by shrinking less important features' coefficients towards zero. In this context, LASSO is applied to predict employee attrition based on various features. The cv.glmnet function performs cross-validated LASSO regression to select the optimal features, resulting in a list of the top selected features printed in the output.

Features Selected:
"Gender", "JobSatisfaction", "MonthlyRate", "NumCompaniesWorked", "RelationshipSatisfaction"

V. Genetic Algorithm

The code employs a Genetic Algorithm (GA) for feature selection. GA is a metaheuristic inspired by natural selection and genetics, used to find optimal solutions to optimization and search problems. In this context, GA iteratively evolves a population of potential feature subsets, evaluating their fitness using a user-defined fitness function. The fitness function in this code trains a logistic regression model on the selected features and returns the negative log-likelihood as the fitness value. GA then evolves the population through selection, crossover, and mutation operations over multiple generations to find the feature subset with the highest fitness. The resulting selected features are printed as the output, aiding in identifying the most relevant features for predicting employee attrition.

Features Selected:
"EmployeeNumber", "Gender", "MonthlyRate", "NumCompaniesWorked", "PerformanceRating", "TrainingTimesLastYear"

VI. Random Forest's Importance Score

Random Forest's importance score quantifies the significance of features in predicting the target variable. Random Forest model is trained using the randomForest function, and importance scores for each feature are computed using the importance function. The top 10 features with the highest importance scores are then selected and printed, revealing the most influential predictors of employee attrition within the dataset.

Features Selected:
"MonthlyIncome", "JobRole", "OverTime", "Age", "DailyRate", "HourlyRate", "MonthlyRate", "TotalWorkingYears", "YearsAtCompany", "DistanceFromHome"

VII. Recursive Feature Elimination (RFE)

RFE (Recursive Feature Elimination) is a feature selection technique that iteratively removes less important features from the dataset until the optimal subset of features is identified. RFE is implemented using the rfe function from the caret package. This function ranks features based on their importance using Random Forest as the underlying model. The selected features, representing the most predictive subset for the target variable ("Attrition"), are then printed as the output.

Features Selected:
"OverTime", "JobRole", "HourlyRate", "DailyRate", "MonthlyRate", "MonthlyIncome", "Age", "PercentSalaryHike", "JobSatisfaction", "DistanceFromHome", "StockOptionLevel", "EducationField", "NumCompaniesWorked", "EnvironmentSatisfaction", "RelationshipSatisfaction", "Education", "WorkLifeBalance", "TrainingTimesLastYear", "TotalWorkingYears", "YearsAtCompany", "YearsWithCurrManager", "MaritalStatus", "JobInvolvement", "YearsSinceLastPromotion", "BusinessTravel", "YearsInCurrentRole", "JobLevel", "Gender"

10. Algorithms Implemented:

I. k-Nearest Neighbors (kNN):

The k-Nearest Neighbors (kNN) algorithm is a simple yet effective method used for classification or regression tasks. It determines the class or value of a query point by examining the labels of its nearest neighbors. kNN is intuitive, easy to implement, and suitable for small to medium-sized datasets, but its performance depends on the choice of distance metric and the number of neighbors (k).

In algorithmic configurations, default parameters are preset values used when specific parameters aren't provided by the user. For instance, in default usage of the k-Nearest Neighbors (KNN) algorithm, the model is trained with a default value of k, often set to 5, without explicit tuning. This method provides a quick implementation for rapid prototyping or when detailed parameter optimization isn't necessary.

In contrast, hyperparameter tuning involves systematically exploring parameter spaces to find values that enhance model performance. In the case of KNN, a grid search methodology evaluates various values of k, typically ranging from 1 to 20, to identify the optimal k value. This optimized parameter selection process results in potentially improved model performance compared to default settings, catering to specific dataset and task requirements.

II. Decision Tree

Decision Trees are versatile machine learning models used for classification and regression tasks. They construct a tree-like structure by recursively partitioning the data based on feature attributes, making them interpretable and suitable for both numerical and categorical data. However, they can overfit and may require pruning or regularization techniques to improve generalization.

In the default parameter scenario, the decision tree model is trained using default settings. Specifically, the rpart function from the rpart package is utilized to build the decision tree. By default, this function employs a complexity parameter (cp) of 0.01, which controls the tree's complexity and potential overfitting. The default settings are applied without explicit specification, resulting in a decision tree model trained with default parameter values. This approach is straightforward and requires minimal user intervention but may not necessarily yield the best-performing model, as the default parameter values are generic and not tailored to the specific dataset.

Contrastingly, the hyperparameter tuning scenario involves systematically optimizing the decision tree model's parameters to improve its performance. Here, the complexity parameter (cp) is tuned using a grid search approach, varying from 0.01 to 0.5 in increments of 0.01. The train function from the caret package is employed, incorporating the rpart method for building decision trees along with the trainControl function to specify the cross-validation strategy. By searching for the optimal cp value, the hyperparameter tuning process aims to mitigate overfitting and enhance the model's generalization capability. This approach is critical as it helps identify the most suitable model configuration for the given dataset, potentially leading to improved predictive performance and robustness.

III. Logistic Regression

Logistic regression is a predictive modeling technique used for binary classification tasks. It estimates the probability of a binary outcome based on input features using the logistic function. It's valued for its simplicity, interpretability, and effectiveness in handling linear and nonlinear relationships between variables.

In the default parameter scenario, logistic regression models are trained using default settings. The glm function from the stats package is utilized to build the logistic regression model. By default, this function employs a family parameter set to "binomial" for binary classification tasks. The default settings are applied without explicit specification, resulting in logistic regression models trained with default parameter values. This approach is straightforward and requires minimal user intervention but may not necessarily yield the best-performing model, as the default parameter values are generic and not tailored to the specific dataset.

Contrastingly, the hyperparameter tuning scenario involves systematically optimizing logistic regression model parameters to improve performance. Here, the glmnet method is employed for logistic regression along with the train function from the caret package. The alpha and lambda hyperparameters are tuned using a grid search approach, varying across a specified range. By searching for optimal hyperparameter values, the hyperparameter tuning process aims to enhance model generalization and performance. This approach is crucial as it helps identify the most suitable model configuration for the given dataset, potentially leading to improved predictive performance and robustness.

IV. Naïve Bayes

Naive Bayes is a simple yet powerful probabilistic classifier based on Bayes' theorem with strong independence assumptions between features. It calculates the probability of a given class label based on the presence of various features. Despite its simplicity, Naive Bayes often performs well in practice, particularly with high-dimensional data and large datasets.

In the default parameter scenario, a Naive Bayes model is trained using default settings. The naive_bayes method from the

caret package is employed to build the Naive Bayes model. Naive Bayes is inherently simple and does not have any hyperparameters to tune. Therefore, there are no specific hyperparameters to configure for this model. By default, the model is trained using the data provided without any further customization. This approach is straightforward and requires minimal user intervention. Naive Bayes models are known for their simplicity and efficiency in handling large datasets with high-dimensional feature spaces. Despite their simplicity, they can perform well in many classification tasks, particularly when the independence assumption holds true among the features.

Unlike some other algorithms, Naive Bayes does not typically involve hyperparameter tuning. This is because Naive Bayes models make strong assumptions about the independence of features, and there are no parameters to adjust or optimize beyond the default settings. Therefore, in this scenario, the default model is used without any hyperparameter tuning. However, the model's performance can still be evaluated using standard metrics such as accuracy, sensitivity, precision, and specificity, which are calculated based on predictions made on the test set.

V. Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of individual trees. It builds each tree independently by randomly selecting features to split on and combining their predictions through averaging or voting, resulting in improved accuracy and robustness compared to a single decision tree.

The Random Forest model is trained using default parameter values, which are carefully chosen to provide a robust and reliable performance out of the box. These default settings have been fine-tuned and optimized by experts to deliver effective results across a wide range of datasets and scenarios without requiring manual adjustments. By leveraging these default parameters, users can effortlessly build accurate predictive models without the need for extensive parameter tuning or customization. This approach simplifies the model-building process and streamlines the workflow, making it accessible to users of varying levels of expertise. Additionally, default parameters often strike a balance between model complexity and computational efficiency, ensuring that the model achieves high performance while remaining computationally tractable. Overall, relying on default parameters enables users to quickly deploy Random Forest models with confidence, knowing that they are leveraging tried-and-tested settings that yield strong predictive performance.

VI. SVM

SVM, or Support Vector Machine, is a supervised learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that separates different classes in the feature space while maximizing the margin between them. SVM can handle both linear and nonlinear data through the use of kernel functions, allowing it to map the input data into higher-dimensional spaces where linear separation is possible. It is effective in scenarios with high-dimensional data and is robust against overfitting, especially when the number of features is greater than the number of samples

The default SVM model, encompassing polynomial, radial, and linear kernels, is instantiated using the svmPoly, svmRadial, and svmLinear methods from the caret package. In default mode, these SVM models are constructed without explicit tuning of hyperparameters, relying on standard settings provided by the algorithms. This entails employing default values for parameters such as the degree of the polynomial kernel, scaling factor, regularization parameter (C), and the gamma parameter for the radial kernel. The models are trained on the provided dataset without any specific adjustments or optimizations.

In contrast, the hyperparameter-tuned SVM models undergo an extensive search for optimal parameter configurations using a grid search technique. The hyperparameter grid, specified through the tuneGrid argument in the train function, explores various combinations of parameters such as the degree of the polynomial kernel, scaling factor, regularization parameter (C), and gamma parameter for the radial kernel. Hyperparameter tuning aims to systematically optimize the SVM models' performance by iteratively adjusting these parameters based on cross-validated performance metrics. This process involves evaluating multiple candidate models to identify the configuration that maximizes classification accuracy and generalization on the dataset. Hyperparameter tuning enhances the SVM models' adaptability to diverse datasets and improves their ability to capture complex patterns in the data.

11. Code*:*

Note*:*

i. code **a** to **e** are for class imbalance handling.
ii. code **f** to **l** are for feature selection techniques.
iii. code **m** is for data preprocessing to be appended before each algorithm given in **n** to **cc**.

iv. code **dd** is for predictions and evaluation to be appended after each code given in **n** to **cc**.
v. code **ee** is about the shiny app integrated with SVM algorithm with Polynomial kernal.

a. Random oversampling of original dataset and saving it.

```
library(randomForest)

d <- read.csv("Dataset.csv")
d$Attrition <- as.character(d$Attrition)

minority_attrition <- subset(d, Attrition == "Yes")
majority_attrition <- subset(d, Attrition == "No")

oversampled_minority_attrition <- minority_attrition[sample(nrow(minority_attrition), nrow(majority_attrition),
replace = TRUE), ]

balanced_data_attrition <- rbind(oversampled_minority_attrition, majority_attrition)

write.csv(balanced_data_attrition, file = "random_over_sampled.csv", row.names = FALSE)

model <- randomForest(Attrition ~ ., data = balanced_data_attrition)
```

b. Random undersampling and saving it.

```
d <- read.csv("Dataset.csv")
# Assuming 'd' is your dataset with a column 'Attrition' indicating the class
d$Attrition <- as.character(d$Attrition)  # Convert 'Attrition' to character if it's a factor

minority_attrition <- subset(d, Attrition == "Yes")
majority_attrition <- subset(d, Attrition == "No")

# Randomly sample from the majority attrition
sampled_majority_attrition <- majority_attrition[sample(nrow(majority_attrition), nrow(minority_attrition)), ]

# Combine minority and sampled majority
balanced_data_attrition <- rbind(minority_attrition, sampled_majority_attrition)

# Save balanced data to a new CSV file
write.csv(balanced_data_attrition, file = "random_under_sampled.csv", row.names = FALSE)
```

c. Both Over and Under sampling and saving it.

```
library(ROSE)
data <- read.csv("Dataset.csv")

# Check the structure of the initial dataset
str(data)

# Check the proportion of attrition cases
prop.table(table(data$Attrition))

# Perform oversampling using ROSE package
oversampled_data <- ovun.sample(Attrition ~ ., data = data, method = "both", p = 0.5, seed = 222, N = 1000)$data

# Check the balance after oversampling
table(oversampled_data$Attrition)

# Save the oversampled dataset as "bothsampled.csv"
write.csv(oversampled_data, file = "bothsampled.csv", row.names = FALSE)
```

```
# Confirm that the dataset has been saved
if (file.exists("bothsampled.csv")) {
  print("Oversampled dataset saved successfully as 'bothsampled.csv'.")
} else {
  print("Error: Oversampled dataset could not be saved.")
}
```

d. Random oversampling using ovum.sample from the ROSE library

```
library(ROSE)
d <- read.csv("Dataset.csv")
d$Attrition <- as.character(d$Attrition)  # Convert 'Attrition' to character if it's a factor

# Perform undersampling using ROSE
result <- ovun.sample(Attrition ~ ., data = d, method = "over", N = 2 * sum(d$Attrition == "No"))

# Extract balanced data from the result
result1 <- result$data

# Save balanced data with undersampling to a new CSV file
write.csv(result1, file = "rose_ovum_sample_over_sampled.csv", row.names = FALSE)
```

e. Random undersampling using ovum.sample from the ROSE library

```
library(ROSE)
d <- read.csv("Dataset.csv")
d$Attrition <- as.character(d$Attrition)  # Convert 'Attrition' to character if it's a factor

# Perform undersampling using ROSE
result <- ovun.sample(Attrition ~ ., data = d, method = "under", N = 2 * sum(d$Attrition == "Yes"))

# Extract balanced data from the result
result1 <- result$data

# Save balanced data with undersampling to a new CSV file
write.csv(result1, file = "rose_ovum_sample_under_sampled.csv", row.names = FALSE)
```

f. Correlation-based Feature Selection (CFS) technique

```
library(caTools)
library(FSelector)
data <- read.csv("random_over_sampled.csv")

# Define categorical columns
categorical_columns <- c("Department", "BusinessTravel", "EducationField",
                         "Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column
data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors
data[, c("Attrition", categorical_columns)] <- lapply(data[, c("Attrition", categorical_columns)], as.factor)

# Manual scaling normalization
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Apply normalization to numerical columns
```

```
numeric_columns <- sapply(data, is.numeric)
if (sum(numeric_columns) > 0) {
  data[, numeric_columns] <- lapply(data[, numeric_columns], normalize)

  cat("\nDataset normalized.\n")
} else {
  cat("\nNo numerical columns found in the dataset.\n")
}

# Apply CFS feature selection
selected_features <- cfs(Attrition ~ ., data = data)

# Print selected features
cat("Selected features based on CFS:\n", selected_features, "\n")
```

g. Chi Square (χ^2) test

```
 # Function to calculate chi-square statistic
chi_square <- function(feature, target) {
  observed <- table(feature, target)
  expected <- outer(rowSums(observed), colSums(observed)) / sum(observed)
  chi_square <- sum((observed - expected)^2 / expected)
  return(chi_square)
}

# Load the dataset
data <- read.csv("random_over_sampled.csv")

# Define categorical columns
categorical_columns <- c("Department", "BusinessTravel", "EducationField",
                         "Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column
data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors
data[, c("Attrition", categorical_columns)] <- lapply(data[, c("Attrition", categorical_columns)], as.factor)

# Calculate chi-square statistic for each feature
chi_square_scores <- sapply(data[, -which(names(data) == "Attrition")],
                            function(x) chi_square(x, data$Attrition))

# Sort features based on chi-square scores
sorted_features <- names(sort(chi_square_scores, decreasing = TRUE))

# Select top features (you can change the number of features)
selected_features <- sorted_features[1:10]

# Print selected features based on chi-square scores
cat("Selected features based on Chi-square Test:\n", selected_features, "\n")
```

h. Information Gain

```
# Load the required libraries
library(caTools)
library(FSelector)

# Load the dataset
data <- read.csv("random_over_sampled.csv")

# Define categorical columns
categorical_columns <- c("Department", "BusinessTravel", "EducationField",
```

```
                "Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column
data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors
data[, c("Attrition", categorical_columns)] <- lapply(data[, c("Attrition", categorical_columns)], as.factor)

# Apply Information Gain feature selection
selected_features <- information.gain(Attrition ~ ., data = data)

# Print the object to check its structure and contents
print(selected_features)
```

i. Least Absolute Shrinkage and Selection Operator (LASSO)

```
# Load the required libraries
library(caTools)
library(glmnet)

# Load the dataset
data <- read.csv("random_over_sampled.csv")

# Define categorical columns
categorical_columns <- c("Department", "BusinessTravel", "EducationField",
                         "Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column
data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors
data[, c("Attrition", categorical_columns)] <- lapply(data[, c("Attrition", categorical_columns)], as.factor)

# Perform LASSO feature selection
x <- model.matrix(Attrition ~ ., data)[,-1]
y <- as.numeric(data$Attrition) - 1
lasso_model <- cv.glmnet(x, y, family = "binomial", alpha = 1)
selected_features <- rownames(coef(lasso_model, s = "lambda.min"))[-1]

# Print selected features
if (length(selected_features) >= 10) {
  cat("Top 10 selected features based on LASSO (L1 Regularization):\n", selected_features[1:10], "\n")
} else {
  cat("Less than 10 features selected.\n")
}
```

j. Genetic Algorithm

```
# Load the required libraries
library(caTools)
library(GA)

# Load the dataset
data <- read.csv("random_over_sampled.csv")

# Define categorical columns
categorical_columns <- c("Department", "BusinessTravel", "EducationField",
                         "Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column
data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)
```

```
# Convert specified categorical columns to factors
data[, c("Attrition", categorical_columns)] <- lapply(data[, c("Attrition", categorical_columns)], as.factor)

# Define fitness function
fitness <- function(features) {
  # Extract the indices of selected features
  selected_indices <- which(features == 1)

  # Check if any features are selected
  if (length(selected_indices) == 0) {
    return(NA) # Return NA if no features are selected
  }

  # Extract selected features including 'Attrition'
  selected_features <- c("Attrition", names(data)[selected_indices])

  # Train logistic regression model using selected features
  model <- glm(Attrition ~ ., data = data[selected_features], family = "binomial")

  # Return the negative log-likelihood as the fitness value
  return(-logLik(model))
}

# Perform genetic algorithm feature selection
ga_result <- ga(type = "binary", fitness = fitness, nBits = ncol(data) - 1, maxiter = 50, run = 100)

# Extract selected features
selected_features <- names(data)[which(as.logical(ga_result@solution))]

# Print selected features
if (length(selected_features) > 0) {
  cat("Selected features based on Genetic Algorithm (GA):\n", selected_features, "\n")
} else {
  cat("No features selected\n")
}
```

k. Importance Score based feature selection from Random Forest

```
library(caTools)
library(rpart)
library(randomForest)

# Load the dataset
data <- read.csv("random_over_sampled.csv")
# Define categorical columns
categorical_columns <- c("Department", "BusinessTravel", "EducationField",
                         "Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column
data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors
data[, c("Attrition", categorical_columns)] <- lapply(data[, c("Attrition", categorical_columns)], as.factor)

# Identify outliers using Z-score method
outliers <- which(abs(scale(data[, sapply(data, is.numeric)])) > 3, arr.ind = TRUE)

# Remove outliers if found
if (length(outliers) > 0) {
  cat("Outliers detected using Z-score method in the following rows:\n")
```

```
 #print(data[outliers, ])

 # Remove outliers
 data <- data[-outliers, ]

 cat("\nOutliers removed using Z-score method.\n")
} else {
 cat("No outliers detected using Z-score method.\n")
}

# Manual scaling normalization
normalize <- function(x) {
 return((x - min(x)) / (max(x) - min(x)))
}

# Apply normalization to numerical columns
numeric_columns <- sapply(data, is.numeric)
if (sum(numeric_columns) > 0) {
 data[, numeric_columns] <- lapply(data[, numeric_columns], normalize)

 cat("\nDataset normalized.\n")
} else {
 cat("\nNo numerical columns found in the dataset.\n")
}

# Set the seed for reproducibility
set.seed(123)

# Train a Random Forest model to determine feature importance
rf_model <- randomForest(Attrition ~ ., data = train_data, ntree = 500)

# Extract variable importance scores
importance_scores <- importance(rf_model)

# Select the top features based on importance scores (e.g., top 10 features)
top_features <- names(importance_scores[order(-importance_scores), ][1:10])
print(top_features)
```

l. Recursive Feature Elimination (RFE)

```
# Load the required libraries
library(caTools)
library(caret)

# Load the dataset
data <- read.csv("random_over_sampled.csv")

# Define categorical columns
categorical_columns <- c("Department", "BusinessTravel", "EducationField",
                         "Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column
data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors
data[, c("Attrition", categorical_columns)] <- lapply(data[, c("Attrition", categorical_columns)], as.factor)

# Apply RFE feature selection
ctrl <- rfeControl(functions = rfFuncs, method = "cv", number = 10)
selected_features <- rfe(data[, -which(names(data) == "Attrition")], data$Attrition, sizes = c(1:ncol(data)-1),
rfeControl = ctrl)
```

```
# Print selected features
cat("Selected features based on RFE:\n", selected_features$optVariables, "\n")
```

m. Data Preprocessing to be performed for all the algorithms:

```
#Load the necessary Libraries
# Load the dataset
data <- read.csv("bothsampled.csv")

# Define categorical columns
categorical_columns <- c("Department", "BusinessTravel", "EducationField",
                         "Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column
data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors
data[, c("Attrition", categorical_columns)] <- lapply(data[, c("Attrition", categorical_columns)], as.factor)

# Count rows before outlier handling
rows_before <- nrow(data)

# Identify outliers using Z-score method
outliers <- which(abs(scale(data[, sapply(data, is.numeric)])) > 3, arr.ind = TRUE)

# Remove outliers if found
if (length(outliers) > 0) {
  # Remove outliers
  data <- data[-outliers, ]}

# Count rows after outlier handling
rows_after <- nrow(data)

# Print count of rows before and after outlier handling
cat("Rows before Z-score method of outlier handling:", rows_before, "\n")
cat("Rows after Z-score method of outlier handling:", rows_after, "\n")

# Manual scaling normalization
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))}

# Apply normalization to numerical columns
numeric_columns <- sapply(data, is.numeric)
if (sum(numeric_columns) > 0) {
  data[, numeric_columns] <- lapply(data[, numeric_columns], normalize)

  cat("\nDataset normalized.\n")
} else {
  cat("\nNo numerical columns found in the dataset.\n")
}

# Set the seed for reproducibility
set.seed(123)
data <- data[sample(nrow(data)), ]

# Split the data into training (70%) and testing (30%) sets
split <- sample.split(data$Attrition, SplitRatio = 0.75)
train_data <- subset(data, split == TRUE)
test_data <- subset(data, split == FALSE)
```

```
# Create a training control specifying the 10-fold cross-validation
train_control <- trainControl(method = "cv", number = 10)
```

n. kNN algorithm with default parameters:

```
library(caTools)
library(caret)

# Create a k-Nearest Neighbors model using 10-fold cross-validation
model <- train(Attrition ~ ., data = train_data, method = "knn", trControl = train_control)
```

o. kNN algorithm with hyper parameter tuning:

```
# Define the hyperparameter grid
hyper_params <- expand.grid(k = seq(1, 20, by = 1))

# Create a k-Nearest Neighbors model with hyperparameter tuning
model <- train(Attrition ~ ., data = train_data, method = "knn", trControl = train_control, tuneGrid = hyper_params)
```

p. kNN algorithm with selected features:

```
# create a set of features and assign to a selected_features variable to perform training on selected features

selected_features <- c("Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel",
"MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager")

# Create a kNN model with selected features
model <- train(Attrition ~ ., data = train_data[, c(selected_features, "Attrition")], method = "knn", trControl =
train_control)
```

q. Logistic Regression algorithm with default parameters:

```
#library(caTools)
#library(caret)

# Create a logistic regression model using 10-fold cross-validation
model <- train(Attrition ~ ., data = train_data, method = "glm", family = "binomial", trControl = train_control)
```

r. Logistic Regression algorithm with hyper parameter tuning:

```
 # Define the hyperparameter grid
hyper_params <- expand.grid(alpha = seq(0, 1, by = 0.1), lambda = seq(0.0001, 0.1, by = 0.01))

# Create a logistic regression model with hyperparameter tuning
model <- train(Attrition ~ ., data = train_data, method = "glmnet", trControl = train_control, tuneGrid =
hyper_params)
```

s. Logistic Regression algorithm with selected features:

```
# Use the selected_features
selected_features <- c("Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel",
"MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager")

# Create a decision tree model with selected features
model <- train(Attrition ~ ., data = train_data[, c(selected_features, "Attrition")], method = "rpart", trControl =
train_control)
```

t. Naïve Bayes algorithm with default parameters:

```
#library(caTools)
#library(caret)

# Create a Naive Bayes model using 10-fold cross-validation
model <- train(Attrition ~ ., data = train_data, method = "naive_bayes", trControl = train_control)
```

u. Naïve Bayes algorithm with selected features:

```
# Use the selected_features
selected_features <- c("Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel",
"MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager")

# Create a Naive Bayes model with selected features
model <- train(Attrition ~ ., data = train_data[, c(selected_features, "Attrition")], method = "naive_bayes", trControl
= train_control)
```

v. Decision Tree Algorithm with default parameters:

```
#library(caTools)
#library(rpart)
#library(caret)

# Create a decision tree model using 10-fold cross-validation
model <- train(Attrition ~ ., data = train_data, method = "rpart", trControl = train_control)
```

w. Decision Tree Algorithm with hyper parameter tuning:

```
# Define the hyperparameter grid
hyper_params <- expand.grid(cp = seq(0.01, 0.5, by = 0.01))

# Create a decision tree model with hyperparameter tuning
model <- train(Attrition ~ ., data = train_data, method = "rpart", trControl = train_control, tuneGrid =
hyper_params)
```

x. Decision Tree Algorithm with selected features:

```
# Use the selected_features
selected_features <- c("Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel",
"MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager")

# Create a decision tree model with selected features
model <- train(Attrition ~ ., data = train_data[, c(selected_features, "Attrition")], method = "rpart", trControl =
train_control)
```

y. Random forest algorithm with default parameters:

```
#library(caTools)
#library(caret)

# Create a Random Forest model using 10-fold cross-validation
model <- train(Attrition ~ ., data = train_data, method = "rf", trControl = train_control)
```

z. Random forest algorithm with selected features:

```
# Use the selected_features
selected_features <- c("Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel",
"MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager")

# Create a Random Forest model with selected features
model <- train(Attrition ~ ., data = train_data[, c(selected_features, "Attrition")], method = "rf", trControl =
train_control)
```

aa. SVM algorithm with default parameters:

```
#library(caTools)
#library(rpart)
#library(caret)

# Create an SVM model using 10-fold cross-validation for Linear kernal
svm_model <- train(Attrition ~ ., data = train_data, method = "svmLinear", trControl = train_control)

# Create an SVM model using 10-fold cross-validation for Radial kernal
svm_model <- train(Attrition ~ ., data = train_data, method = "svmRadial", trControl = train_control)

# Create an SVM model using 10-fold cross-validation for Polynomial kernal
svm_model <- train(Attrition ~ ., data = train_data, method = "svmPoly", trControl = train_control)
```

bb. SVM algorithm with hyper parameter tuning:

```
#For Linear kernal
# Define the hyperparameter grid
tuning_grid <- expand.grid(
  .C = c(0.1, 1, 10, 100)  # Tuning parameter for regularization)
# Create an SVM model with hyperparameter tuning
svm_model <- train(  Attrition ~ ., data = train_data, method = "svmLinear", trControl = train_control, tuneGrid =
tuning_grid)


#For Radial kernal
# Define the hyperparameter grid
tuning_grid <- expand.grid(
  .sigma = c(0.1, 0.8, 1),
  .C = c(0.1, 1, 20))
# Create an SVM model with hyperparameter tuning
svm_model <- train(  Attrition ~ ., data = train_data, method = "svmRadial", trControl = train_control, tuneGrid =
tuning_grid)


#For Polynomial kernal
# Define the hyperparameter grid
tuning_grid <- expand.grid(
  .degree = 2:5,
  .scale = c(0.1, 1, 10),
  .C = c(0.1, 1, 10))
# Create an SVM model with hyperparameter tuning
svm_model <- train(  Attrition ~ ., data = train_data, method = "svmPoly", trControl = train_control, tuneGrid =
tuning_grid)
```

cc. SVM algorithm with selected features:

```
# Use the selected_features
selected_features <- c("Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel",
"MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager")

#For Linear kernal
# Create an SVM model with selected features
svm_model <- train(  Attrition ~ ., data = train_data[, c(selected_features, "Attrition")], method = "svmLinear",
trControl = train_control, tuneGrid = tuning_grid)

#For Radial kernal
# Create an SVM model with selected features
svm_model <- train(  Attrition ~ ., data = train_data[, c(selected_features, "Attrition")], method = "svmRadial",
trControl = train_control, tuneGrid = tuning_grid)
```

```
#For Polynomial kernal
# Create an SVM model with selected features
svm_model <- train(  Attrition ~ ., data = train_data[, c(selected_features, "Attrition")], method = "svmPoly",
trControl = train_control, tuneGrid = tuning_grid)
```

dd. Predictions and Evaluation Parameters for all the algorithms:

```
 # Make predictions on the test set
predictions <- predict(model, newdata = test_data)

# Confusion matrix to evaluate the model
conf_matrix <- table(predictions, test_data$Attrition)
print(conf_matrix)

# Calculate accuracy
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
print(paste("Accuracy:", accuracy))

# Calculate sensitivity (true positive rate)
sensitivity <- conf_matrix[2, 2] / sum(conf_matrix[2, ])
print(paste("Sensitivity:", sensitivity))

# Calculate precision (positive predictive value)
precision <- conf_matrix[2, 2] / sum(conf_matrix[, 2])
print(paste("Precision:", precision))

# Calculate specificity (true negative rate)
specificity <- conf_matrix[1, 1] / sum(conf_matrix[1, ])
print(paste("Specificity:", specificity))
```

ee. Shinny app code integrated with Random Forest algorithm with LASSO selected Features:

```
library(shiny)
library(caTools)
library(caret)

# Load the dataset
data <- read.csv("random_over_sampled.csv")

# Define categorical columns
categorical_columns <- c("Department", "BusinessTravel", "EducationField",
                         "Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column
data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors
data[, c("Attrition", categorical_columns)] <- lapply(data[, c("Attrition", categorical_columns)], as.factor)


# Set the seed for reproducibility
set.seed(123)
data <- data[sample(nrow(data)), ]

# Split the data into training (75%) and testing (25%) sets
split <- sample.split(data$Attrition, SplitRatio = 0.75)
train_data <- subset(data, split == TRUE)

selected_features <- c("Age", "BusinessTravel", "DailyRate", "Department", "DistanceFromHome", "Education",
"EducationField")
```

```
# Train the Random Forest model
model <- train(Attrition ~ ., data = train_data[, c(selected_features, "Attrition")], method = "rf")

# Define UI
ui <- fluidPage(
  titlePanel("Employee Attrition Prediction"),
  sidebarLayout(
    sidebarPanel(
      # Input fields for features
      numericInput("age", "Age:", min = 18, max = 65, value = 30),
      selectInput("business_travel", "Business Travel:",
                  choices = c("Non-Travel", "Travel_Rarely", "Travel_Frequently"),
                  selected = "Travel_Rarely"),
      numericInput("daily_rate", "Daily Rate:", min = 0, max = 2000, value = 1000),
      selectInput("department", "Department:",
                  choices = c("Sales", "Research & Development", "Human Resources"),
                  selected = "Sales"),
      numericInput("distance_from_home", "Distance From Home:", min = 1, max = 30, value = 10),
      numericInput("education", "Education:", min = 1, max = 5, value = 3),
      selectInput("education_field", "Education Field:",
                  choices = c("Life Sciences", "Medical", "Marketing", "Technical Degree", "Other"),
                  selected = "Life Sciences"),
      actionButton("submit", "Submit")  # Submit button
    ),
    mainPanel(
      textOutput("prediction")
    )
  )
)

# Define server logic
server <- function(input, output) {
  output$prediction <- renderText({
    # Check if the Submit button is clicked
    req(input$submit)

    # Create a new data frame with user input
    new_data <- data.frame(
      Age = input$age,
      BusinessTravel = input$business_travel,
      DailyRate = input$daily_rate,
      Department = input$department,
      DistanceFromHome = input$distance_from_home,
      Education = input$education,
      EducationField = input$education_field
    )

    # Make prediction using the trained model
    prediction <- predict(model, newdata = new_data)
    paste("Predicted Attrition:", prediction)
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

12. Shiny app:

The user interface (UI) of the Shiny app is designed with a structured layout comprising distinct sections, each focusing on different facets of employee information relevant to attrition prediction. These sections are visually organized within a form container, characterized by a clean appearance with rounded corners and a light gray background, contributing to an aesthetically pleasing interface. The form header stands out prominently with its larger font size, bold text, and centrally aligned position, serving as a clear indicator of the form's purpose. Within each section, users are presented with intuitive input fields, including numeric inputs for age, salary, etc., and select inputs for categorical variables like gender and marital status. Each input field is accompanied by a descriptive label, ensuring clarity and ease of use. The submit button, strategically positioned at the form's bottom, features a distinct green color, inviting user interaction, and providing feedback with a hover effect.

On the server side, the app employs logic to predict attrition based on user inputs. Upon clicking the "Predict Attrition" button, an observer function is triggered, capturing the user's input data and constructing a new entry/data frame. This data is then passed through the SVM model, previously trained on a dataset, to predict the likelihood of attrition. Subsequently, based on the prediction outcome (1 for attrition, 0 for no attrition), a corresponding text message is rendered to the user, offering real-time feedback. This prediction process facilitates informed decision-making by users, empowering them to assess potential attrition risks based on their input information.

13. Evaluation Parameters :

In our employee retention prediction project, we employ a variety of evaluation parameters to comprehensively assess the performance of our machine learning model. These metrics play a crucial role in providing insights into the effectiveness and reliability of our model's predictions, particularly in the context of predicting attrition.

Accuracy serves as a foundational metric, allowing us to gauge the overall correctness of our model's predictions. It quantifies the ratio of correctly predicted instances of attrition to the total number of instances in our dataset, providing a broad understanding of our model's predictive capability.

Furthermore, **sensitivity**, also known as recall or the true positive rate, holds significant importance in employee retention prediction. Sensitivity measures the proportion of actual instances of attrition that our model correctly identifies among all instances of attrition in the dataset. This metric is particularly valuable as it helps us understand how effectively our model detects employees who are likely to leave the company.

Precision, or the positive predictive value, complements sensitivity by evaluating the proportion of true positive predictions among all instances predicted as attrition by our model. Precision is essential in scenarios where the cost associated with incorrectly predicting attrition is high. It provides insights into the model's ability to minimize false positive predictions, thereby assisting in making more accurate and actionable decisions related to employee retention strategies.

Lastly, **specificity**, or the true negative rate, assesses the model's ability to correctly identify instances of non-attrition, which is equally important in employee retention prediction. It quantifies the proportion of actual instances of non-attrition that our model correctly identifies among all instances of non-attrition in the dataset. Specificity complements sensitivity and provides insights into the model's capability to accurately identify employees who are likely to stay with the company.

By examining these evaluation parameters comprehensively, we gain valuable insights into the strengths and weaknesses of our model in predicting employee attrition. This enables us to make informed decisions and implement targeted retention strategies to retain valuable talent within the organization.

14. Results and Discussions:

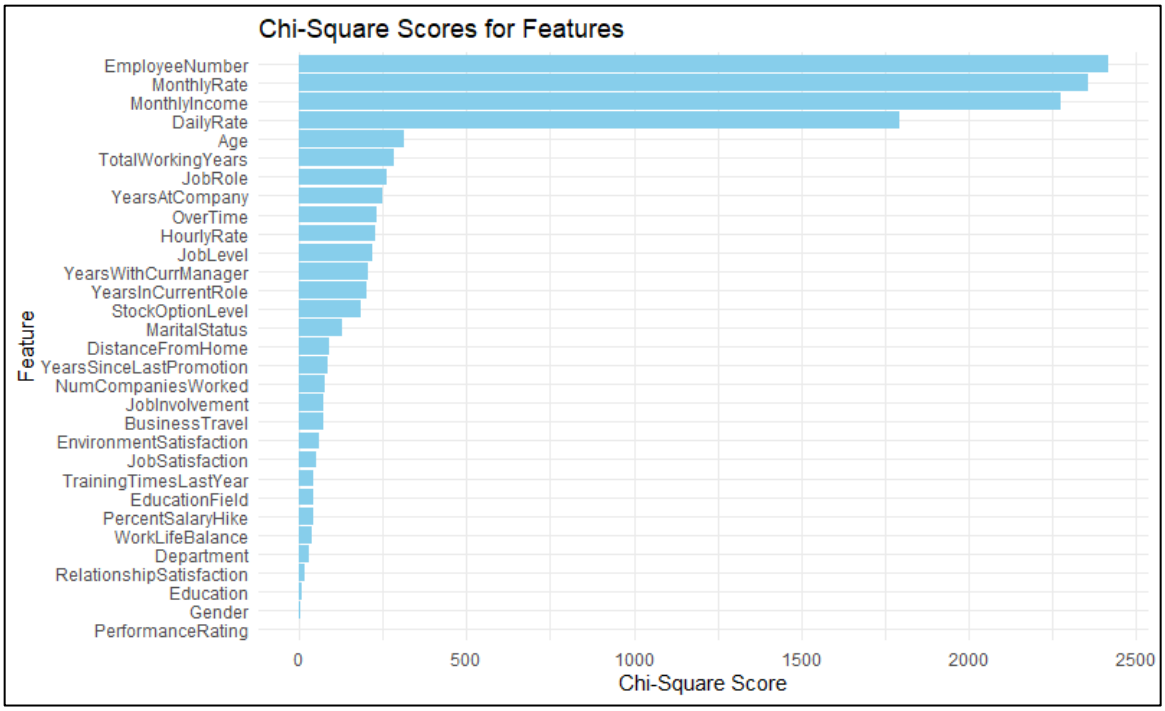
A. Feature selection

i. CFS

Feature selected are:
Age, BusinessTravel, EnvironmentSatisfaction, JobInvolvement, JobLevel, MonthlyIncome, OverTime, StockOptionLevel, YearsAtCompany, YearsWithCurrManager

ii. Chi Square

Feature	ChiSquareScore
Age	315.3092264
BusinessTravel	70.61059632
DailyRate	1794.973933
Department	31.38839839
DistanceFromHome	90.56197745
Education	9.862164485
EducationField	43.45045674
EmployeeNumber	2418
EnvironmentSatisfaction	59.33730712
Gender	3.921161237
HourlyRate	226.6285417
JobInvolvement	72.47673072
JobLevel	219.4300478
JobRole	260.111413
JobSatisfaction	52.22474472
MaritalStatus	129.4487773
MonthlyIncome	2276.331619
MonthlyRate	2358.625953
NumCompaniesWorked	77.81181937
OverTime	232.6078718
PercentSalaryHike	41.33961184
PerformanceRating	0.037104199
RelationshipSatisfaction	14.76499721
StockOptionLevel	186.4794092
TotalWorkingYears	282.0465359
TrainingTimesLastYear	43.90091664
WorkLifeBalance	38.87191714
YearsAtCompany	249.8458832
YearsInCurrentRole	200.7466377
YearsSinceLastPromotion	83.90282488
YearsWithCurrManager	205.3773984

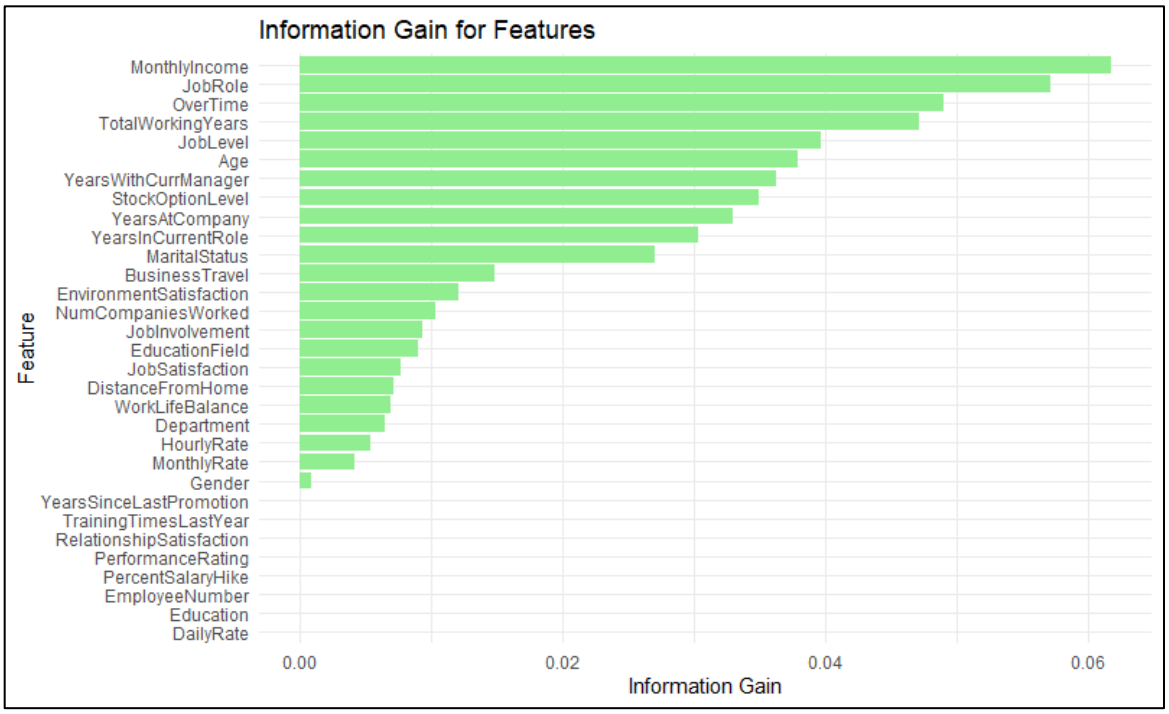


Features selected are:
MonthlyRate, MonthlyIncome, DailyRate, Age, TotalWorkingYears, JobRole, YearsAtCompany, OverTime, HourlyRate, JobLevel

iii. Information Gain

Feature	Gain
Age	0.037953248
BusinessTravel	0.014845339
DailyRate	0
Department	0.006503633
DistanceFromHome	0.00712368
Education	0
EducationField	0.009034191
EmployeeNumber	0
EnvironmentSatisfaction	0.012123576
Gender	0.000811295
HourlyRate	0.005322012
JobInvolvement	0.009374112
JobLevel	0.039648317
JobRole	0.057173861
JobSatisfaction	0.007715517
MaritalStatus	0.027048342
MonthlyIncome	0.061796829
MonthlyRate	0.004196587
NumCompaniesWorked	0.010338056
OverTime	0.049000636
PercentSalaryHike	0

PerformanceRating	0
RelationshipSatisfaction	0
StockOptionLevel	0.03500689
TotalWorkingYears	0.047126485
TrainingTimesLastYear	0
WorkLifeBalance	0.006870389
YearsAtCompany	0.032974001
YearsInCurrentRole	0.030363229
YearsSinceLastPromotion	0
YearsWithCurrManager	0.036244573



Features Selected are:
Age, Department, BusinessTravel, DistanceFromHome, EducationField, EnvironmentSatisfaction, Gender, HourlyRate, JobInvolvement, JobLevel, JobRole, JobSatisfaction, MaritalStatus, MonthlyIncome, MonthlyRate, NumCompaniesWorked, OverTime, StockOptionLevel, TotalWorkingYears, WorkLifeBalance, YearsAtCompany, YearsInCurrentRole, YearsWithCurrManager

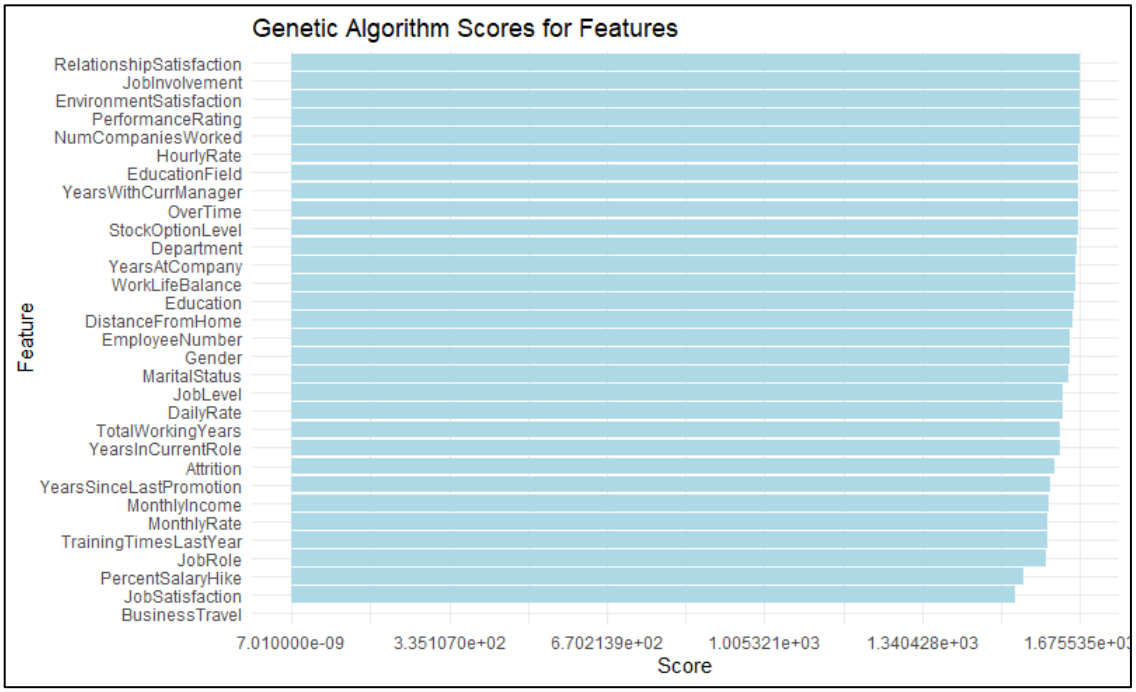
iv. Lasso

Features Selected are:
Age, BusinessTravel, DailyRate, Department, DistanceFromHome, Education, EducationField

v. Genetic Algorithm

	x
Attrition	1621.949978
BusinessTravel	7.01E-09
DailyRate	1639.657395
Department	1668.365709

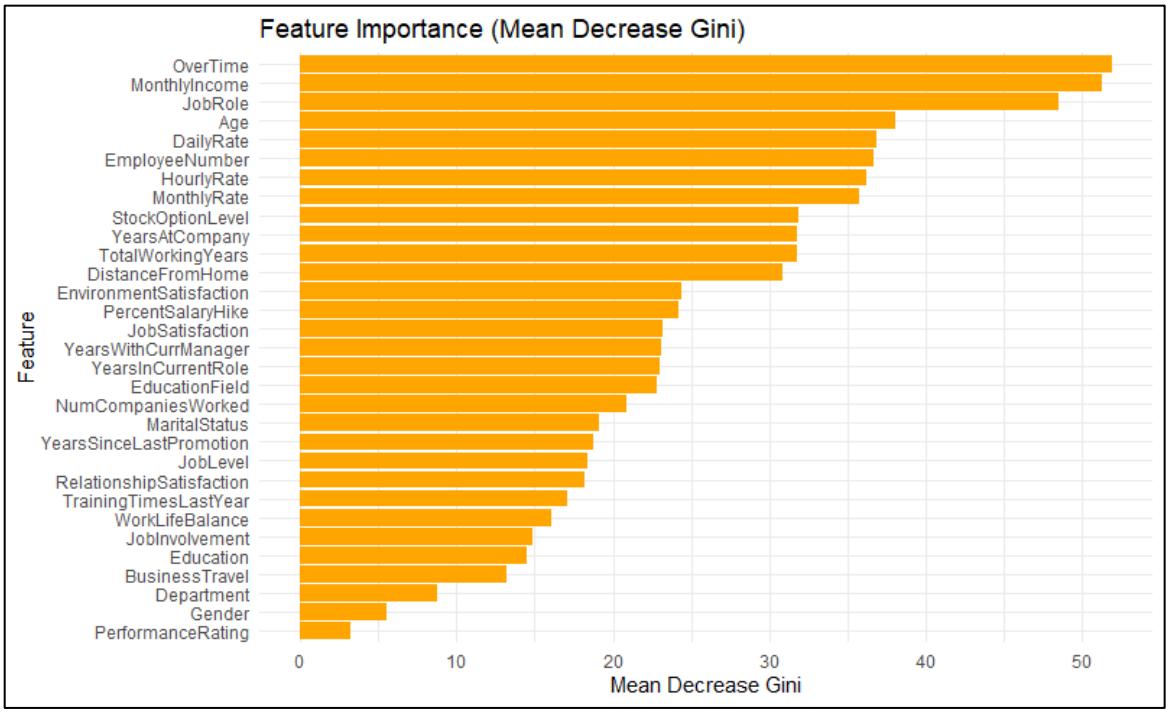
DistanceFromHome	1659.82764
Education	1662.467622
EducationField	1673.325929
EmployeeNumber	1653.708751
EnvironmentSatisfaction	1675.296153
Gender	1653.106874
HourlyRate	1673.591713
JobInvolvement	1675.447656
JobLevel	1640.445247
JobRole	1603.543978
JobSatisfaction	1537.307029
MaritalStatus	1652.041111
MonthlyIncome	1610.150534
MonthlyRate	1606.679417
NumCompaniesWorked	1675.043724
OverTime	1671.561043
PercentSalaryHike	1557.069888
PerformanceRating	1675.15645
RelationshipSatisfaction	1675.534874
StockOptionLevel	1671.001937
TotalWorkingYears	1633.560298
TrainingTimesLastYear	1605.253588
WorkLifeBalance	1667.471743
YearsAtCompany	1667.480879
YearsInCurrentRole	1632.955759
YearsSinceLastPromotion	1612.080462
YearsWithCurrManager	1673.067296



Features Selected are:
Gender, JobSatisfaction, MonthlyRate, NumCompaniesWorked, RelationshipSatisfaction

vi. Importance Score

	MeanDecreaseGini
Age	38.03947013
BusinessTravel	13.23978667
DailyRate	36.88881916
Department	8.803784686
DistanceFromHome	30.84804054
Education	14.52894715
EducationField	22.81950383
EmployeeNumber	36.69694915
EnvironmentSatisfaction	24.41017306
Gender	5.527355943
HourlyRate	36.181871
JobInvolvement	14.85362606
JobLevel	18.35388856
JobRole	48.47040138
JobSatisfaction	23.20097567
MaritalStatus	19.13341276
MonthlyIncome	51.27210846
MonthlyRate	35.79469902
NumCompaniesWorked	20.86985237
OverTime	51.93527292
PercentSalaryHike	24.21111822
PerformanceRating	3.194797356
RelationshipSatisfaction	18.15947837
StockOptionLevel	31.89167153
TotalWorkingYears	31.77883072
TrainingTimesLastYear	17.08399917
WorkLifeBalance	16.05136748
YearsAtCompany	31.81745439
YearsInCurrentRole	22.96137446
YearsSinceLastPromotion	18.75122047
YearsWithCurrManager	23.06935874



Features Selected are:
MonthlyIncome, JobRole, OverTime, Age, DailyRate, HourlyRate, MonthlyRate, TotalWorkingYears, YearsAtCompany, DistanceFromHome

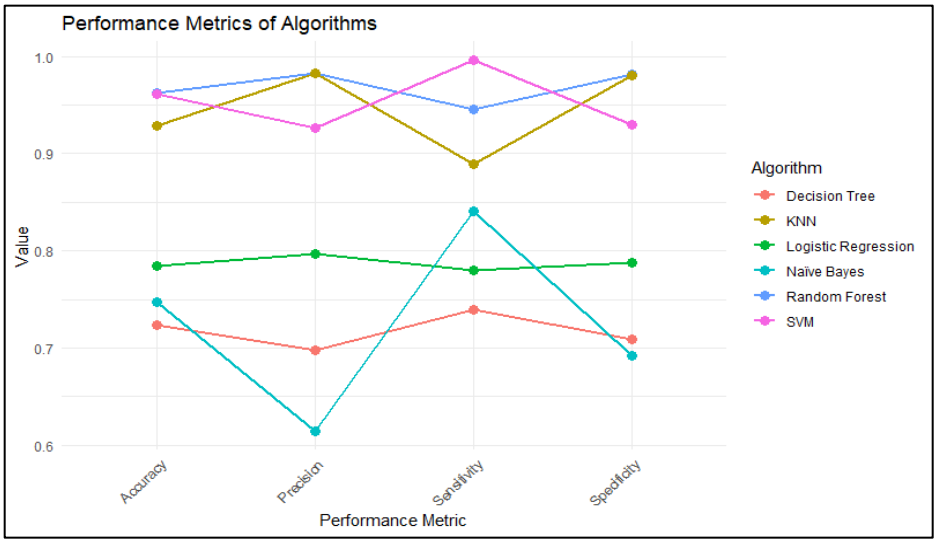
vii. RFE

Features Selected are:
OverTime, JobRole, HourlyRate, DailyRate, MonthlyRate, MonthlyIncome, Age, PercentSalaryHike, JobSatisfaction, DistanceFromHome, StockOptionLevel, EducationField, NumCompaniesWorked, EnvironmentSatisfaction, RelationshipSatisfaction, Education, WorkLifeBalance, TrainingTimesLastYear, TotalWorkingYears, YearsAtCompany, YearsWithCurrManager, MaritalStatus, JobInvolvement, YearsSinceLastPromotion, BusinessTravel, YearsInCurrentRole, JobLevel, Gender

B. Algorithms Implemented

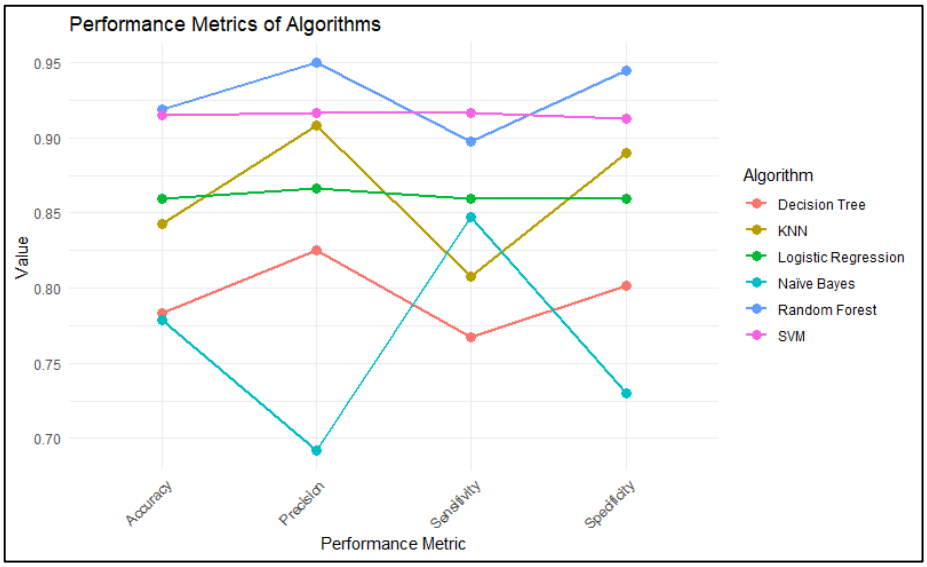
i. With all features for random over sampled dataset

	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.723893805	0.739776952	0.698245614	0.709459459
Random Forest	0.962831858	0.945945946	0.98245614	0.981412639
KNN	0.92920354	0.888888889	0.98245614	0.98
Logistic Regression	0.784070796	0.780068729	0.796491228	0.788321168
Naïve Bayes	0.746902655	0.841346154	0.614035088	0.691876751
SVM	0.961061947	0.996226415	0.926315789	0.93



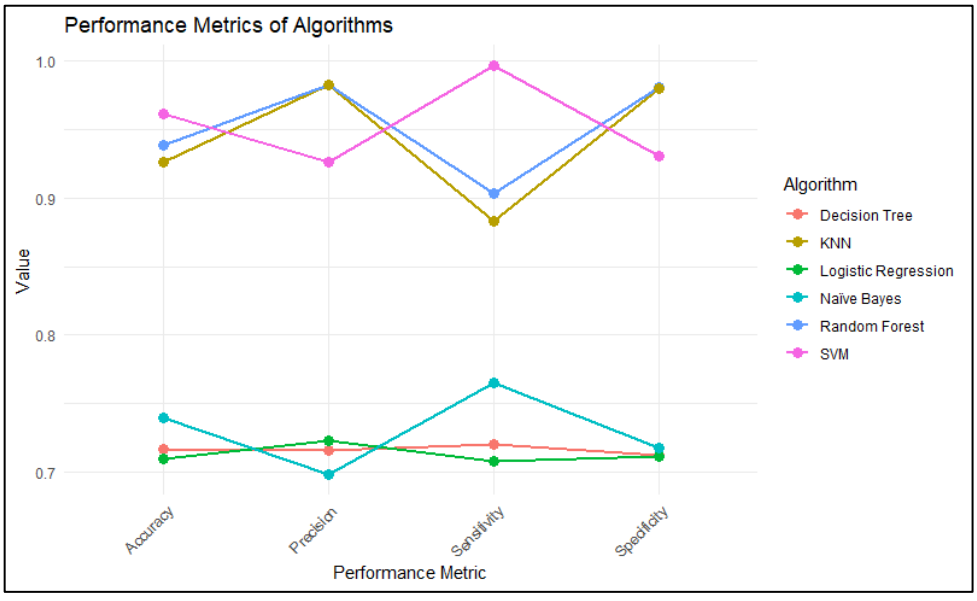
ii. With all features for Both over and under sampled dataset

	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.782978723	0.76744186	0.825	0.801886792
Random Forest	0.919148936	0.897637795	0.95	0.944444444
KNN	0.842553191	0.807407407	0.908333333	0.89
Logistic Regression	0.859574468	0.859504132	0.866666667	0.859649123
Naïve Bayes	0.778723404	0.846938776	0.691666667	0.729927007
SVM	0.914893617	0.916666667	0.916666667	0.913043478



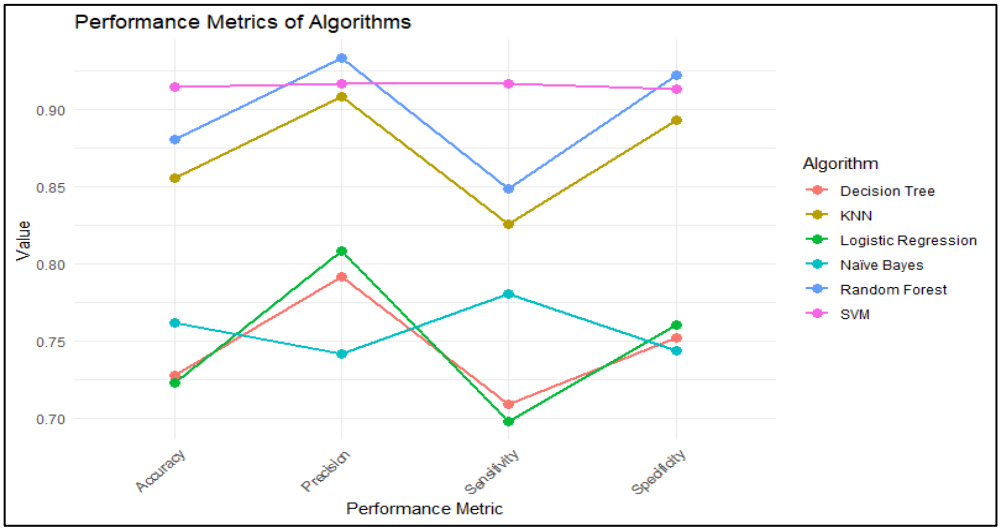
iii. With CFS selected features for random over sampled dataset

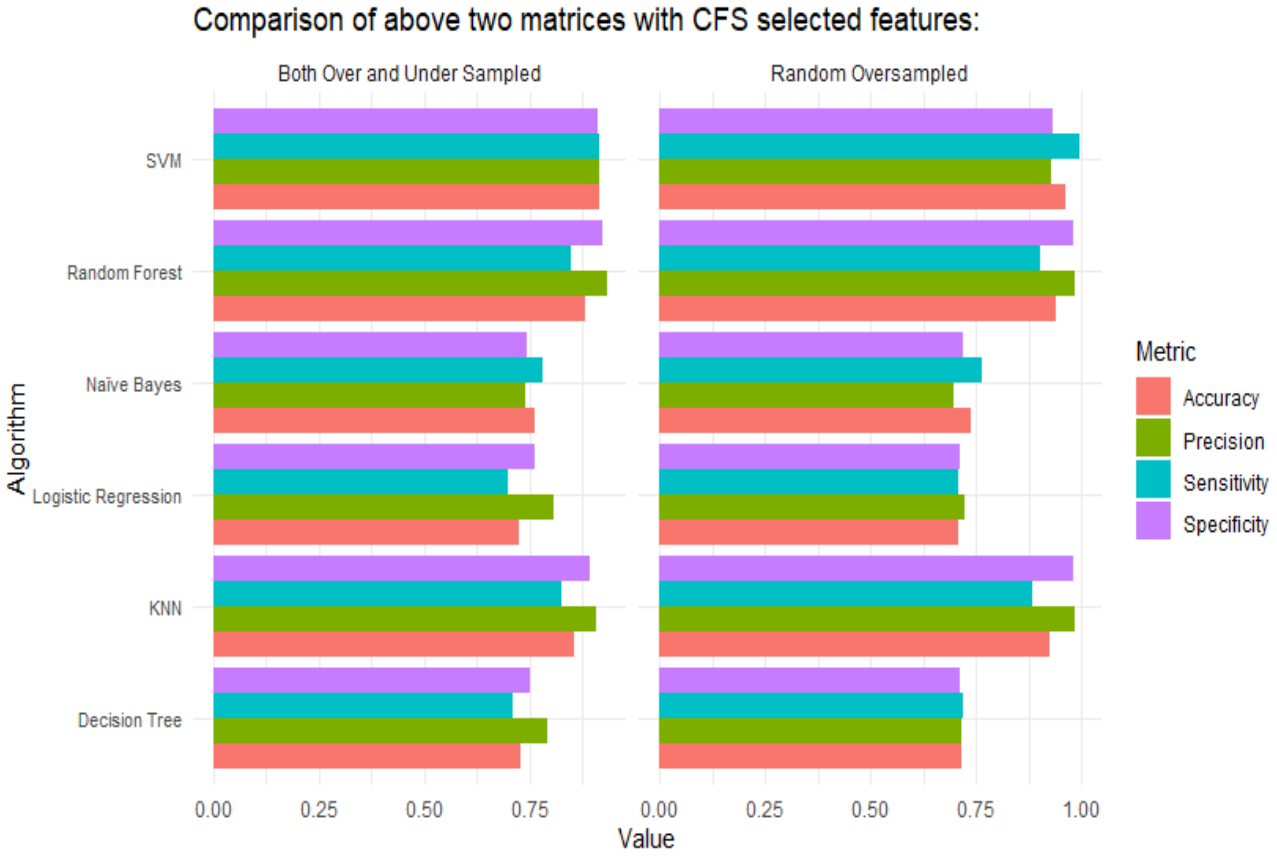
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.716814159	0.720848057	0.715789474	0.712765957
Random Forest	0.938053097	0.903225806	0.98245614	0.980392157
KNN	0.925663717	0.883280757	0.98245614	0.97983871
Logistic Regression	0.709734513	0.70790378	0.722807018	0.711678832
Naïve Bayes	0.739823009	0.765384615	0.698245614	0.718032787
SVM	0.961061947	0.996226415	0.926315789	0.93



iv. With CFS selected features for Both over and under sampled dataset

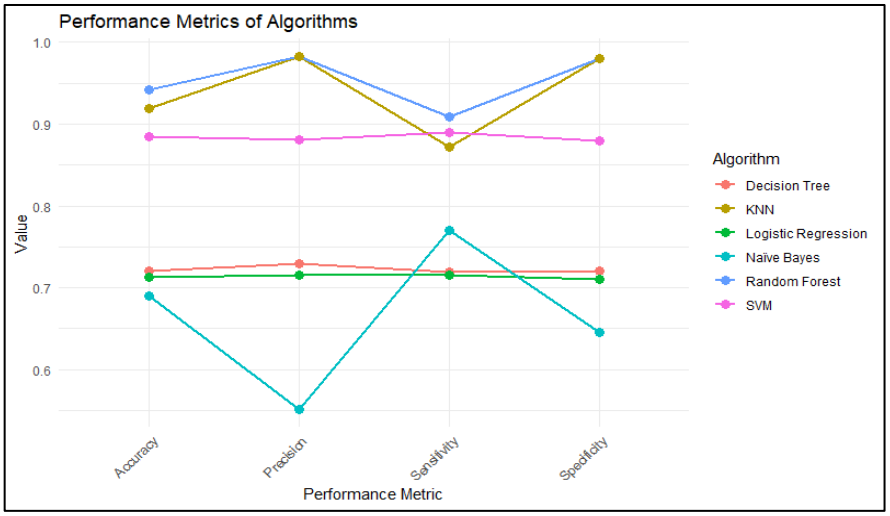
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.727659574	0.708955224	0.791666667	0.752475248
Random Forest	0.880851064	0.848484848	0.933333333	0.922330097
KNN	0.855319149	0.825757576	0.908333333	0.893203883
Logistic Regression	0.723404255	0.697841727	0.808333333	0.760416667
Naïve Bayes	0.761702128	0.780701754	0.741666667	0.743801653
SVM	0.914893617	0.916666667	0.916666667	0.913043478





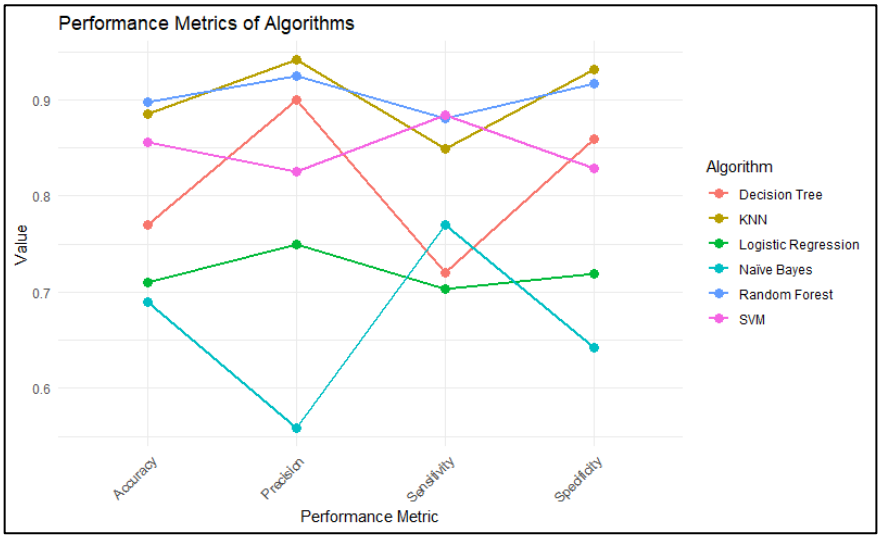
v. With Chi Square selected features for random over sampled dataset

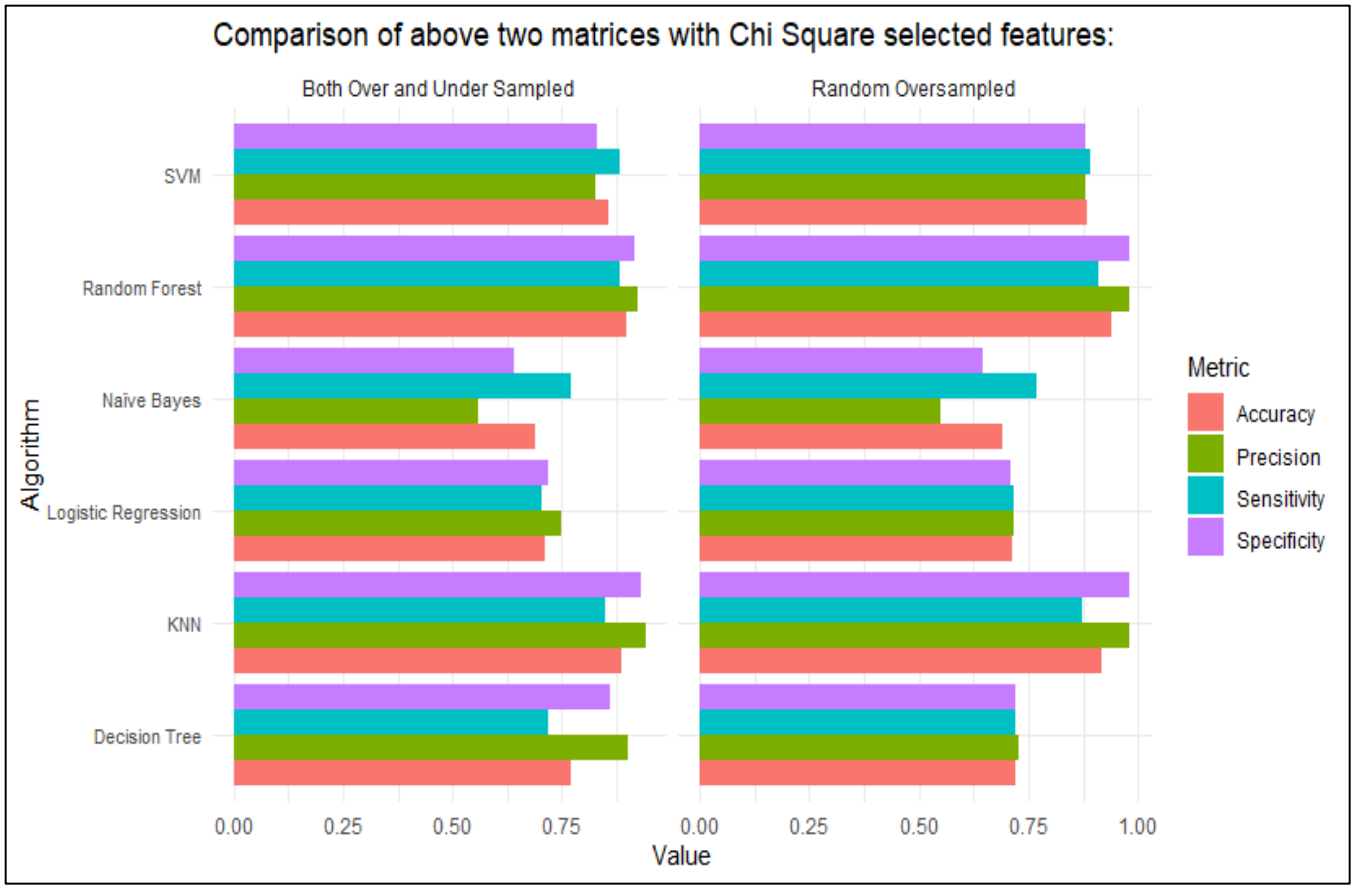
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.720353982	0.719723183	0.729824561	0.721014493
Random Forest	0.94159292	0.909090909	0.98245614	0.980544747
KNN	0.918584071	0.872274143	0.98245614	0.979508197
Logistic Regression	0.713274336	0.715789474	0.715789474	0.710714286
Naïve Bayes	0.690265487	0.769607843	0.550877193	0.645429363
SVM	0.884955752	0.890070922	0.880701754	0.879858657



vi. With Chi Square selected features for Both over and under sampled dataset

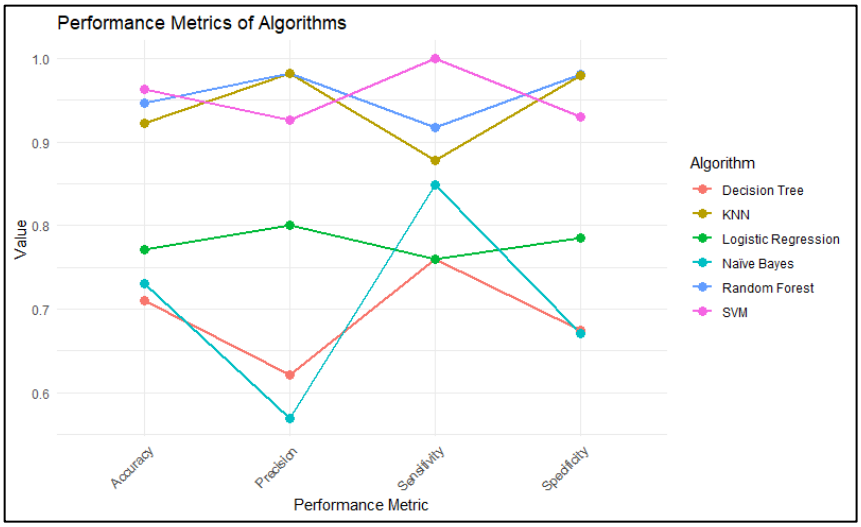
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.770212766	0.72	0.9	0.858823529
Random Forest	0.89787234	0.880952381	0.925	0.917431193
KNN	0.885106383	0.84962406	0.941666667	0.931372549
Logistic Regression	0.710638298	0.703125	0.75	0.719626168
Naïve Bayes	0.689361702	0.770114943	0.558333333	0.641891892
SVM	0.855319149	0.883928571	0.825	0.829268293





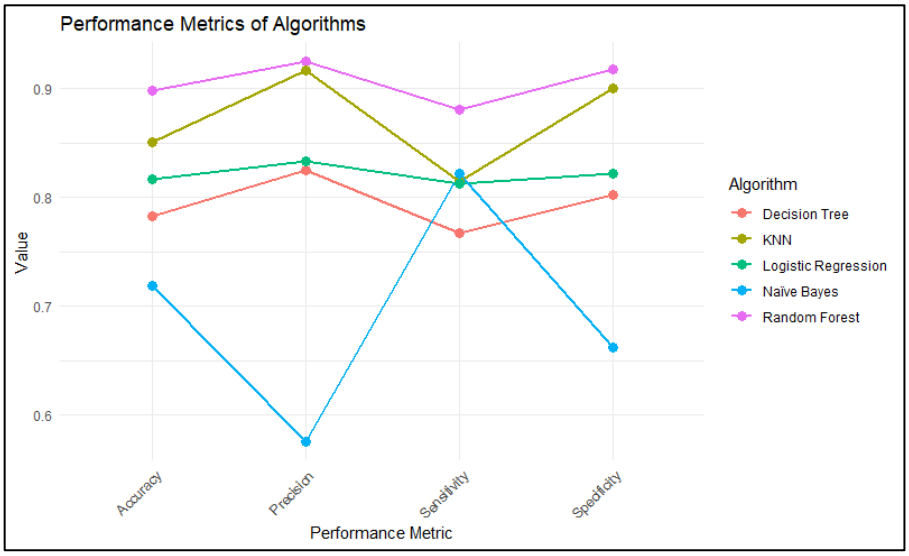
vii. With Information Gain selected features for random over sampled dataset

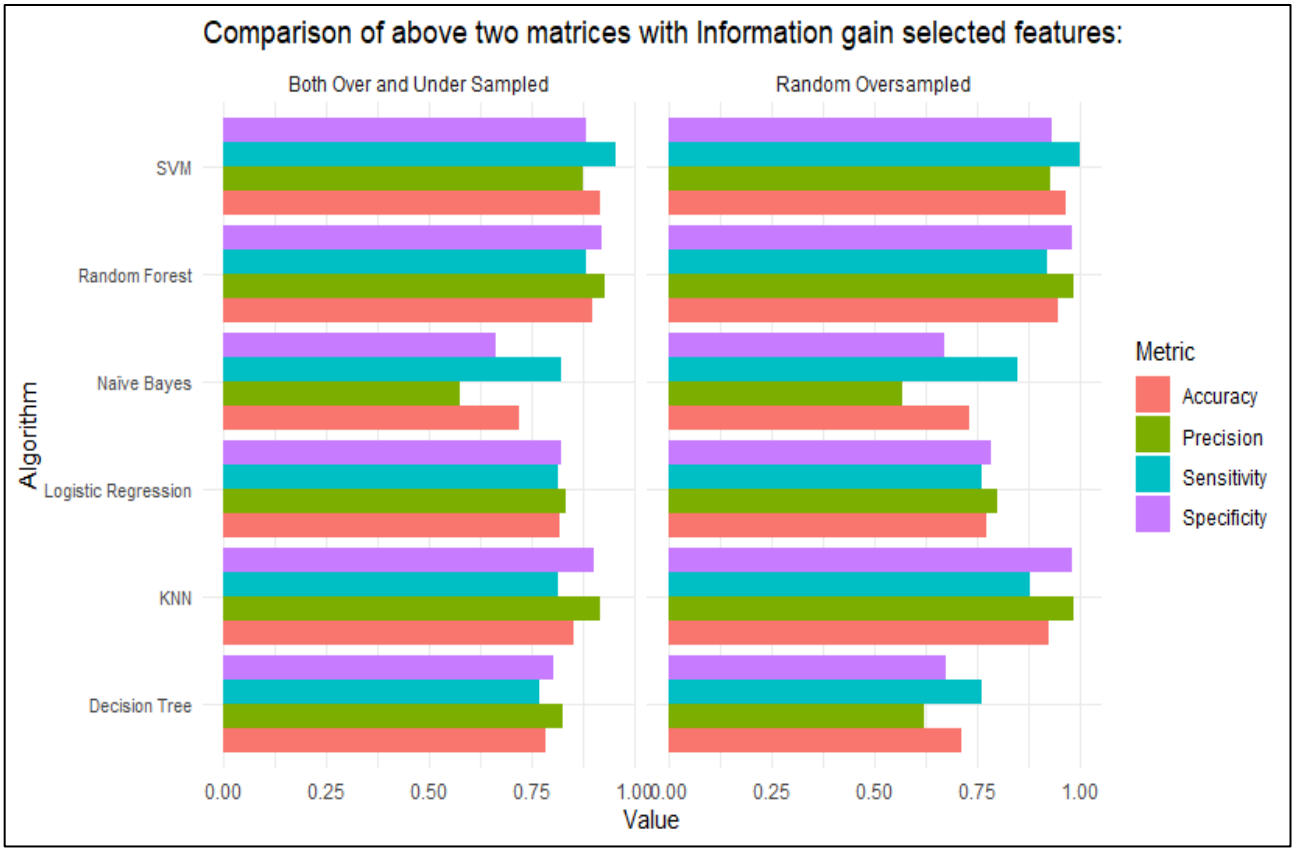
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.709734513	0.759656652	0.621052632	0.674698795
Random Forest	0.946902655	0.918032787	0.98245614	0.980769231
KNN	0.922123894	0.877742947	0.98245614	0.979674797
Logistic Regression	0.771681416	0.76	0.8	0.78490566
Naïve Bayes	0.730973451	0.848167539	0.568421053	0.671122995
SVM	0.962831858	1	0.926315789	0.930232558



viii. With Information Gain selected features for Both over and under sampled dataset

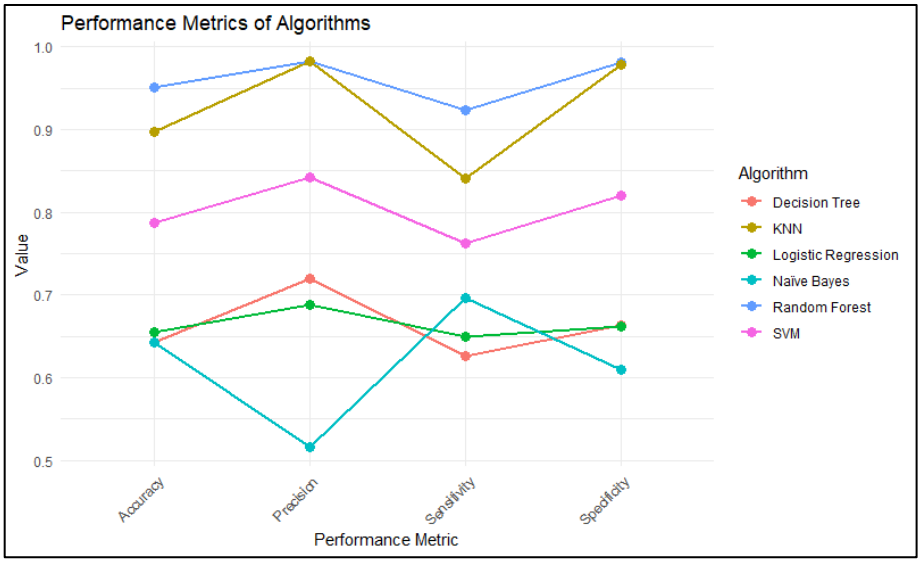
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.782978723	0.76744186	0.825	0.801886792
Random Forest	0.89787234	0.880952381	0.925	0.917431193
KNN	0.85106383	0.814814815	0.916666667	0.9
Logistic Regression	0.817021277	0.81300813	0.833333333	0.821428571
Naïve Bayes	0.719148936	0.821428571	0.575	0.662251656
SVM	0.914893617	0.954545455	0.875	0.88





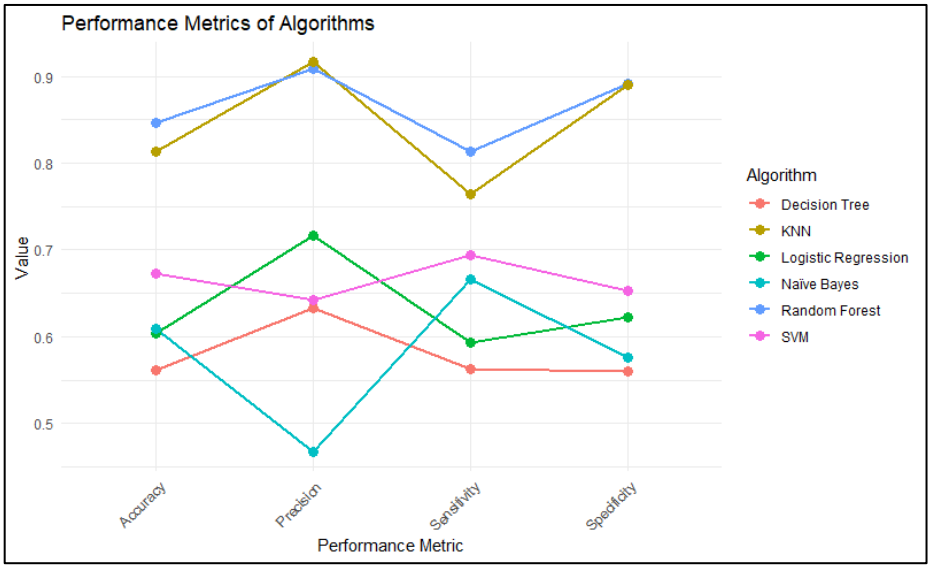
ix. With LASSO selected features for random over sampled dataset

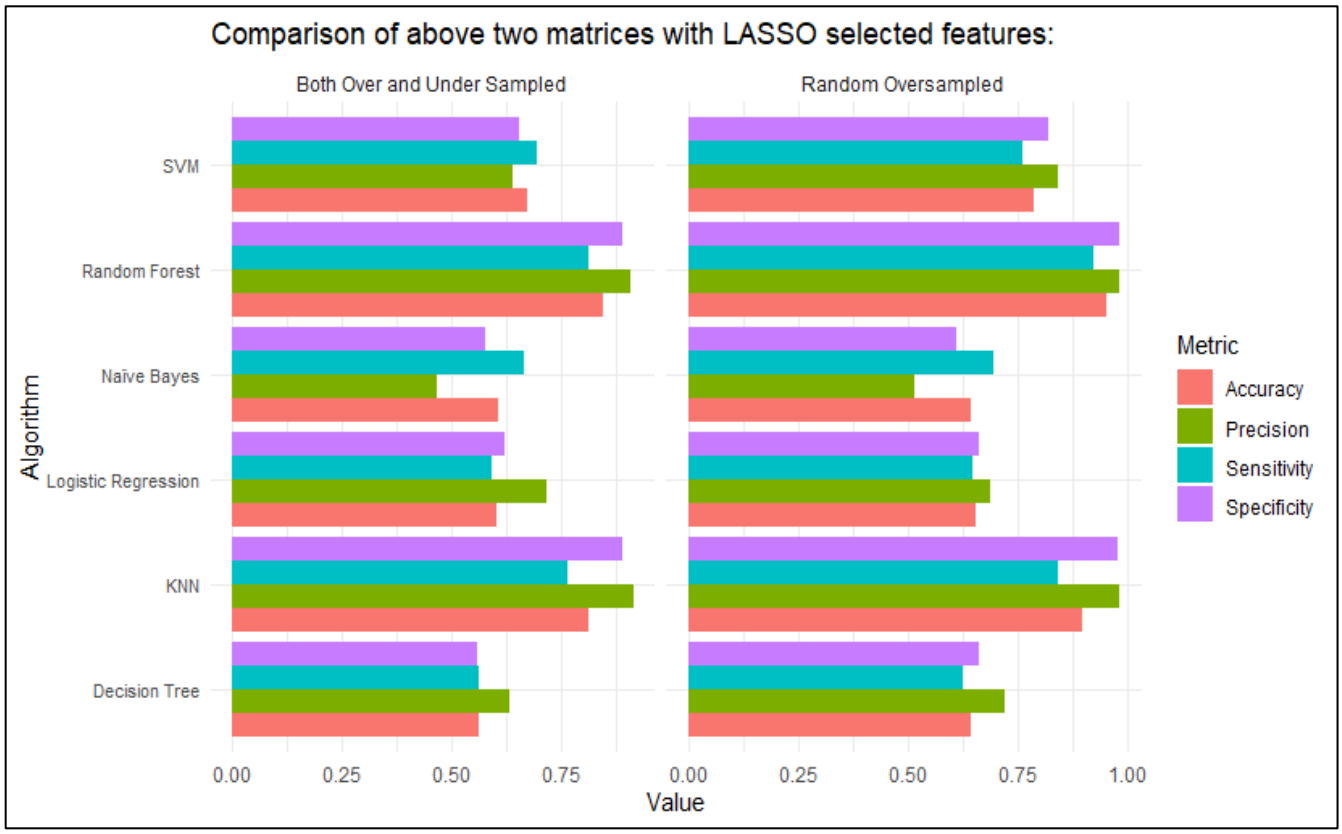
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.642477876	0.626911315	0.719298246	0.663865546
Random Forest	0.950442478	0.924092409	0.98245614	0.980916031
KNN	0.897345133	0.840840841	0.98245614	0.978448276
Logistic Regression	0.654867257	0.649006623	0.687719298	0.661596958
Naïve Bayes	0.642477876	0.696682464	0.515789474	0.610169492
SVM	0.787610619	0.761904762	0.842105263	0.82



x. With LASSO selected features for Both over and under sampled dataset

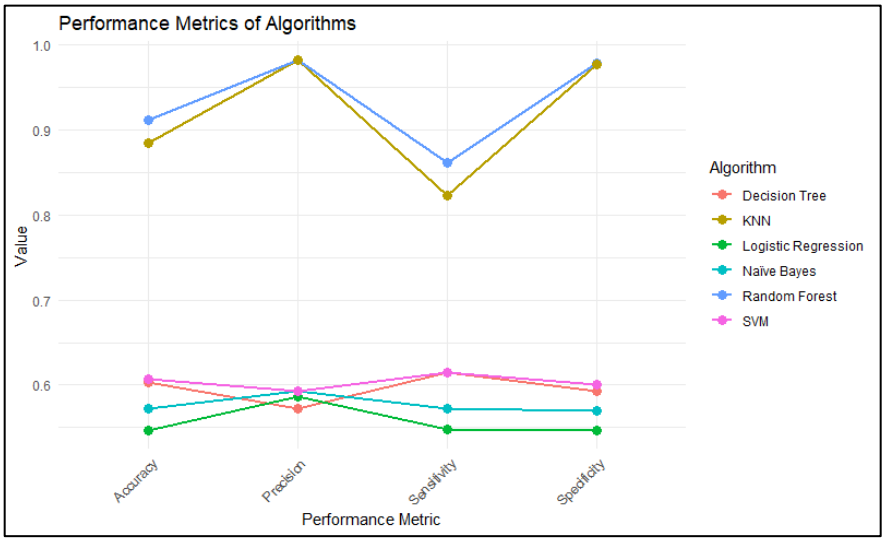
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.561702128	0.562962963	0.633333333	0.56
Random Forest	0.846808511	0.813432836	0.908333333	0.891089109
KNN	0.812765957	0.763888889	0.916666667	0.89010989
Logistic Regression	0.604255319	0.593103448	0.716666667	0.622222222
Naïve Bayes	0.608510638	0.666666667	0.466666667	0.57615894
SVM	0.672340426	0.693693694	0.641666667	0.653225806





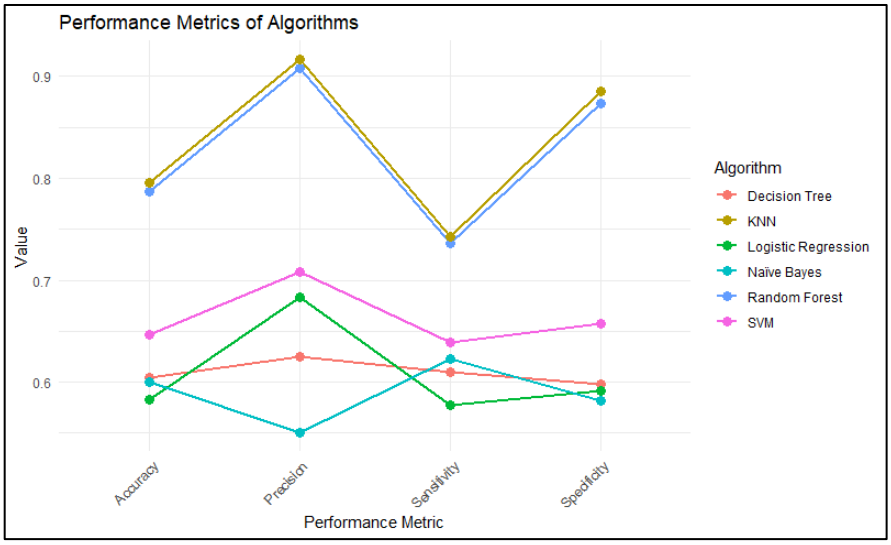
xi. With Genetic Algorithm selected features for random over sampled dataset

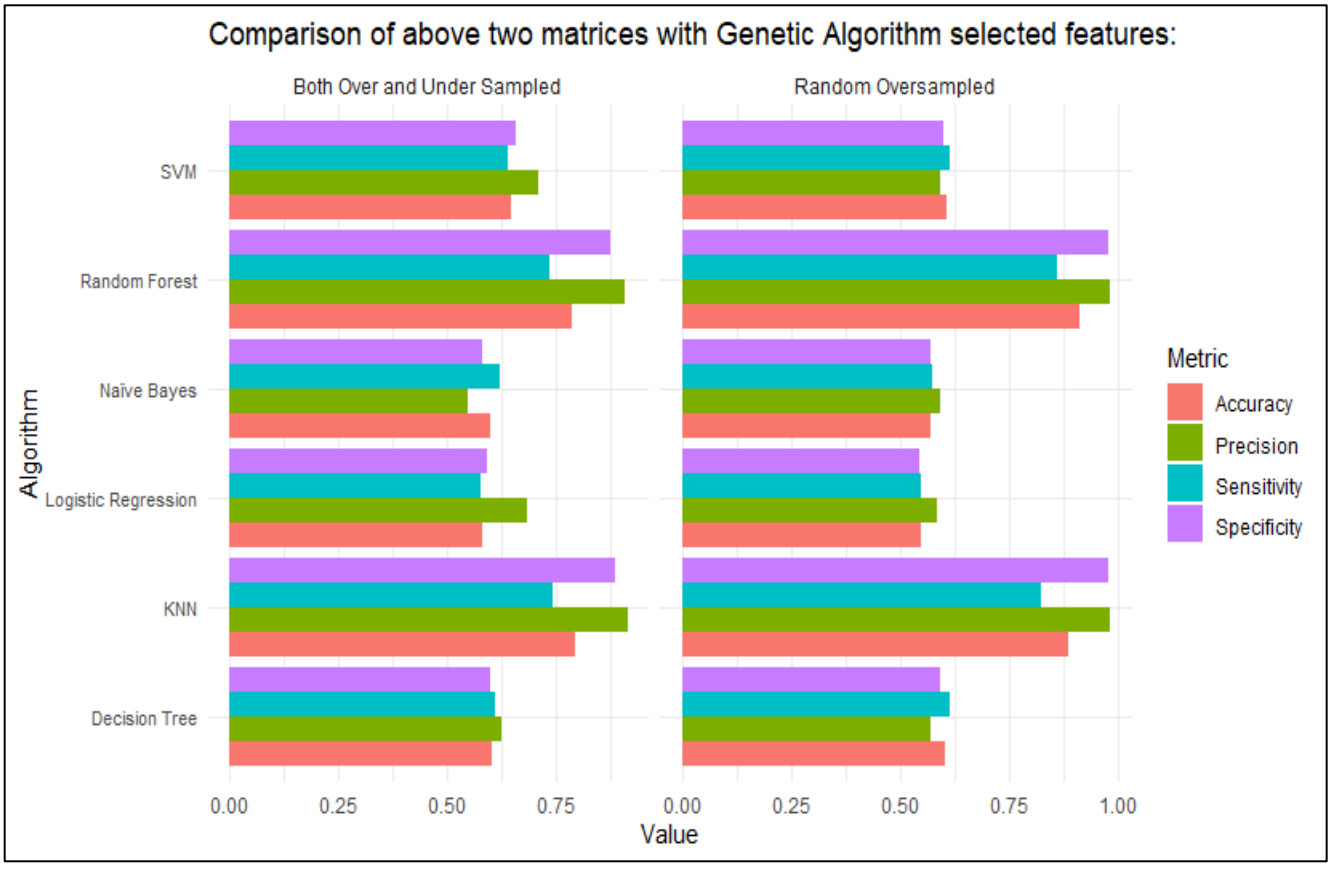
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.603539823	0.61509434	0.571929825	0.593333333
Random Forest	0.911504425	0.861538462	0.98245614	0.979166667
KNN	0.884955752	0.823529412	0.98245614	0.977777778
Logistic Regression	0.546902655	0.547540984	0.585964912	0.546153846
Naïve Bayes	0.571681416	0.572881356	0.592982456	0.57037037
SVM	0.607079646	0.614545455	0.592982456	0.6



xii. With Genetic Algorithm selected features for Both over and under sampled dataset

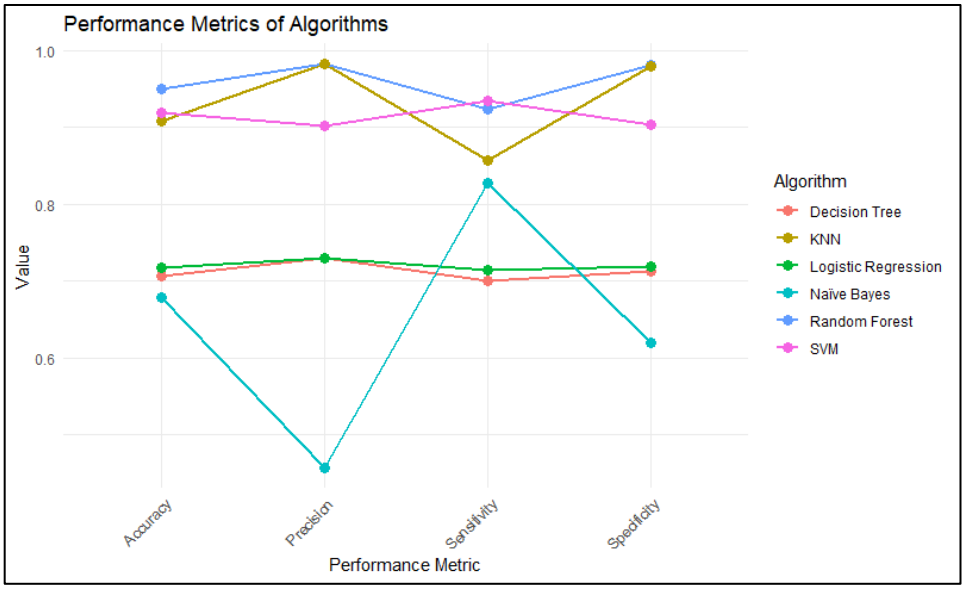
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.604255319	0.609756098	0.625	0.598214286
Random Forest	0.787234043	0.736486486	0.908333333	0.873563218
KNN	0.795744681	0.743243243	0.916666667	0.885057471
Logistic Regression	0.582978723	0.577464789	0.683333333	0.591397849
Naïve Bayes	0.6	0.622641509	0.55	0.581395349
SVM	0.646808511	0.639097744	0.708333333	0.656862745





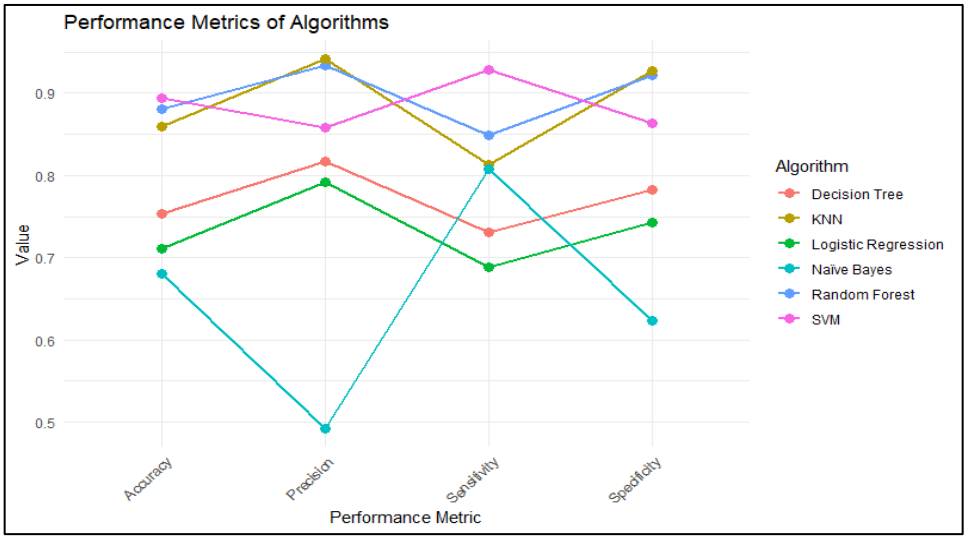
xiii. With Importance Score selected features for random over sampled dataset

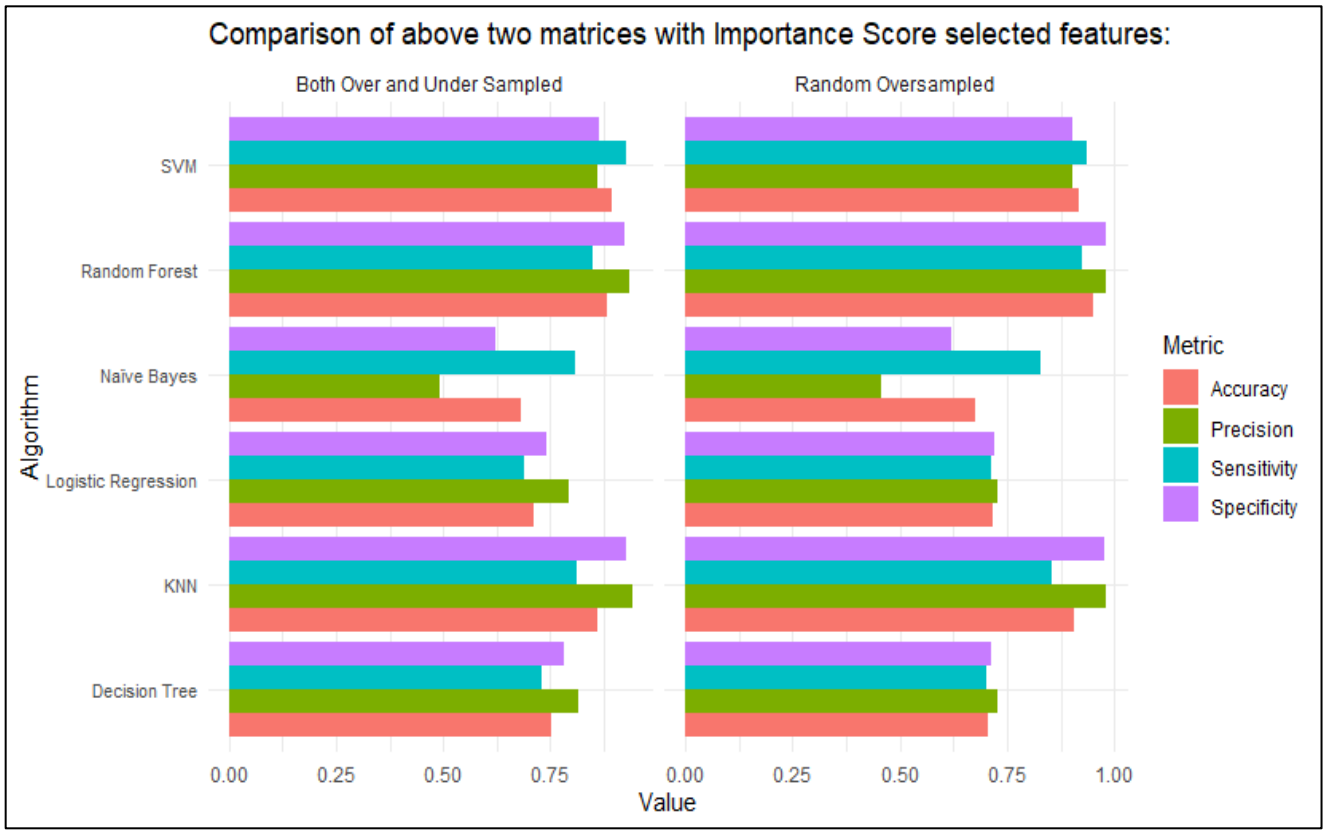
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.70619469	0.7003367	0.729824561	0.712686567
Random Forest	0.950442478	0.924092409	0.98245614	0.980916031
KNN	0.907964602	0.856269113	0.98245614	0.978991597
Logistic Regression	0.716814159	0.714776632	0.729824561	0.718978102
Naïve Bayes	0.677876106	0.828025478	0.456140351	0.620098039
SVM	0.918584071	0.934545455	0.901754386	0.903448276



xiv. With Importance Score selected features for Both over and under sampled dataset

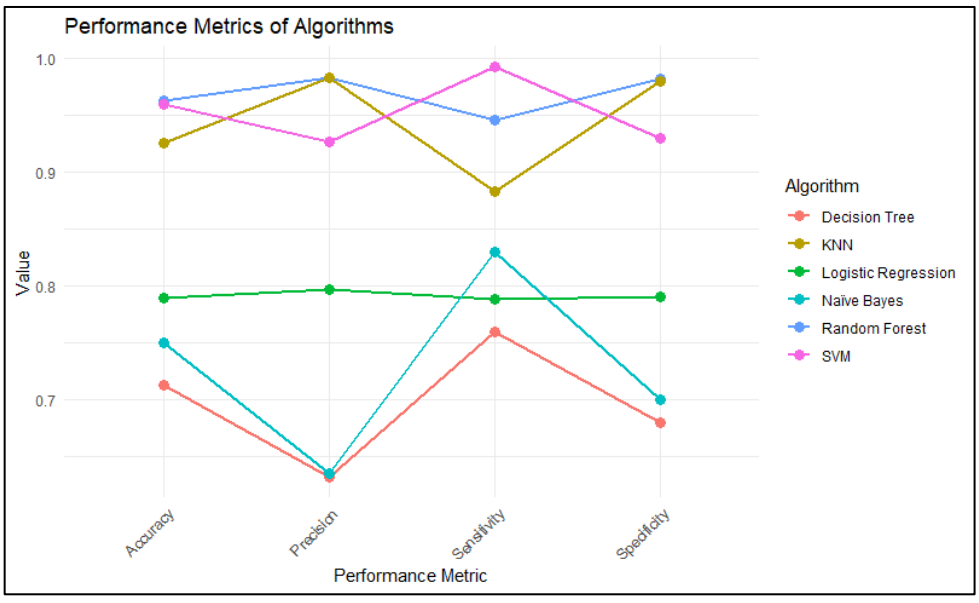
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.753191489	0.731343284	0.816666667	0.782178218
Random Forest	0.880851064	0.848484848	0.933333333	0.922330097
KNN	0.859574468	0.81294964	0.941666667	0.927083333
Logistic Regression	0.710638298	0.688405797	0.791666667	0.742268041
Naïve Bayes	0.680851064	0.808219178	0.491666667	0.62345679
SVM	0.893617021	0.927927928	0.858333333	0.862903226





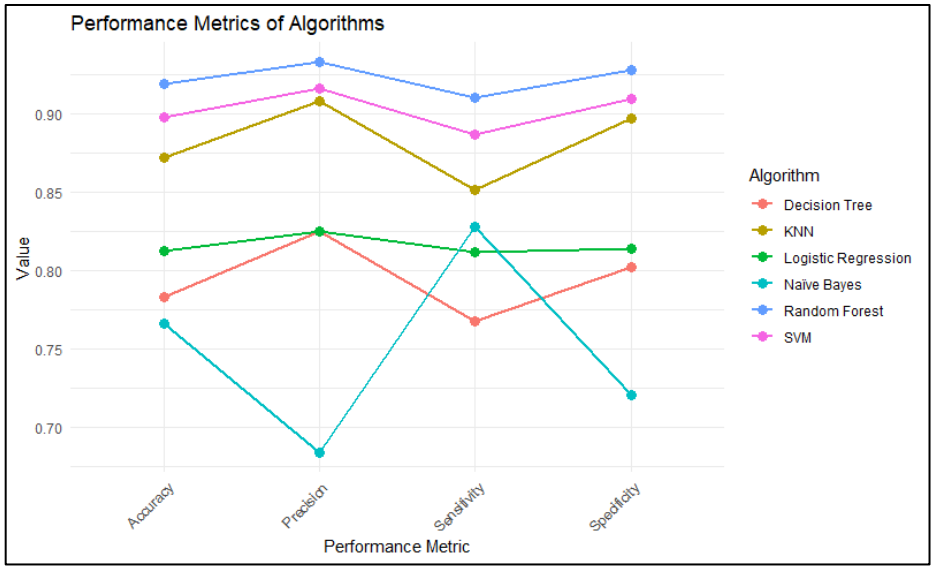
xv. With RFE selected features for random over sampled dataset

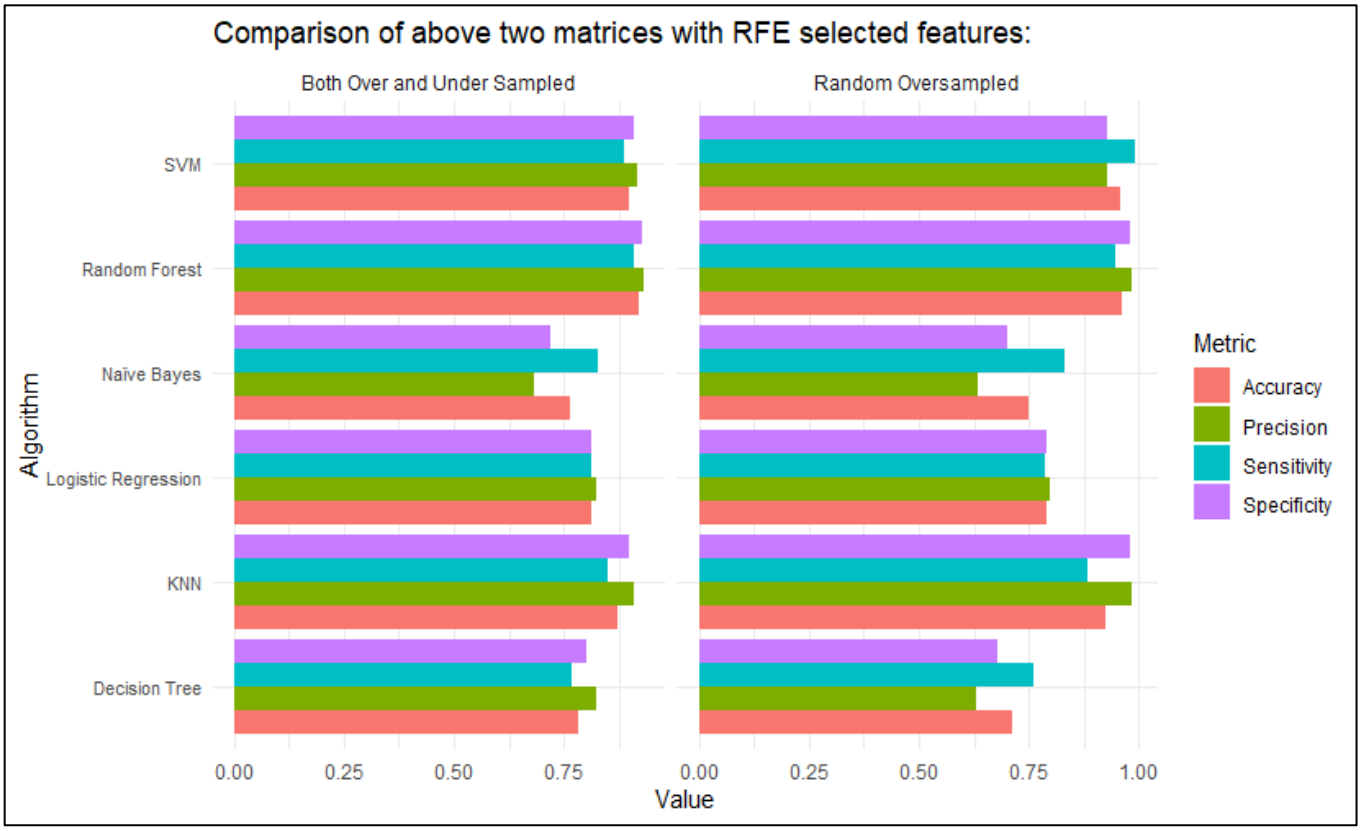
	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.713274336	0.759493671	0.631578947	0.679878049
Random Forest	0.962831858	0.945945946	0.98245614	0.981412639
KNN	0.925663717	0.883280757	0.98245614	0.97983871
Logistic Regression	0.789380531	0.788194444	0.796491228	0.790613718
Naïve Bayes	0.750442478	0.830275229	0.635087719	0.700288184
SVM	0.959292035	0.992481203	0.926315789	0.929765886



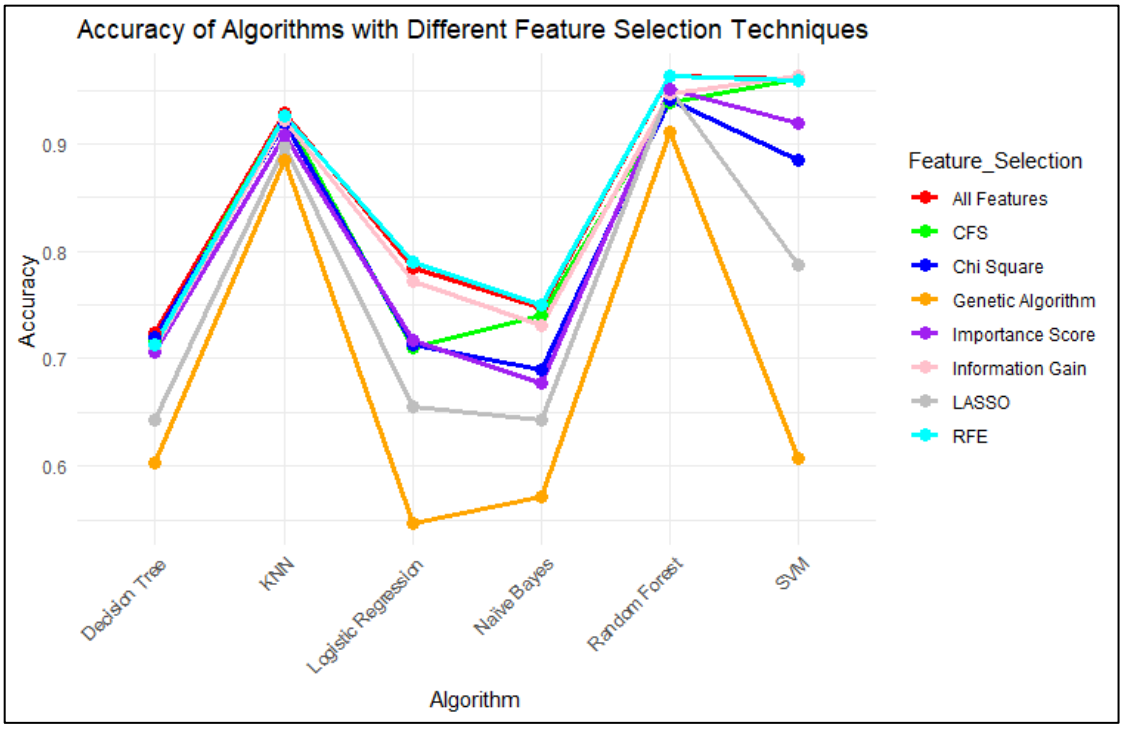
xvi. With RFE selected features for Both over and under sampled dataset

	Accuracy	Sensitivity	Precision	Specificity
Decision Tree	0.782978723	0.76744186	0.825	0.801886792
Random Forest	0.919148936	0.910569106	0.933333333	0.928571429
KNN	0.872340426	0.8515625	0.908333333	0.897196262
Logistic Regression	0.812765957	0.81147541	0.825	0.814159292
Naïve Bayes	0.765957447	0.828282828	0.683333333	0.720588235
SVM	0.89787234	0.887096774	0.916666667	0.90990991

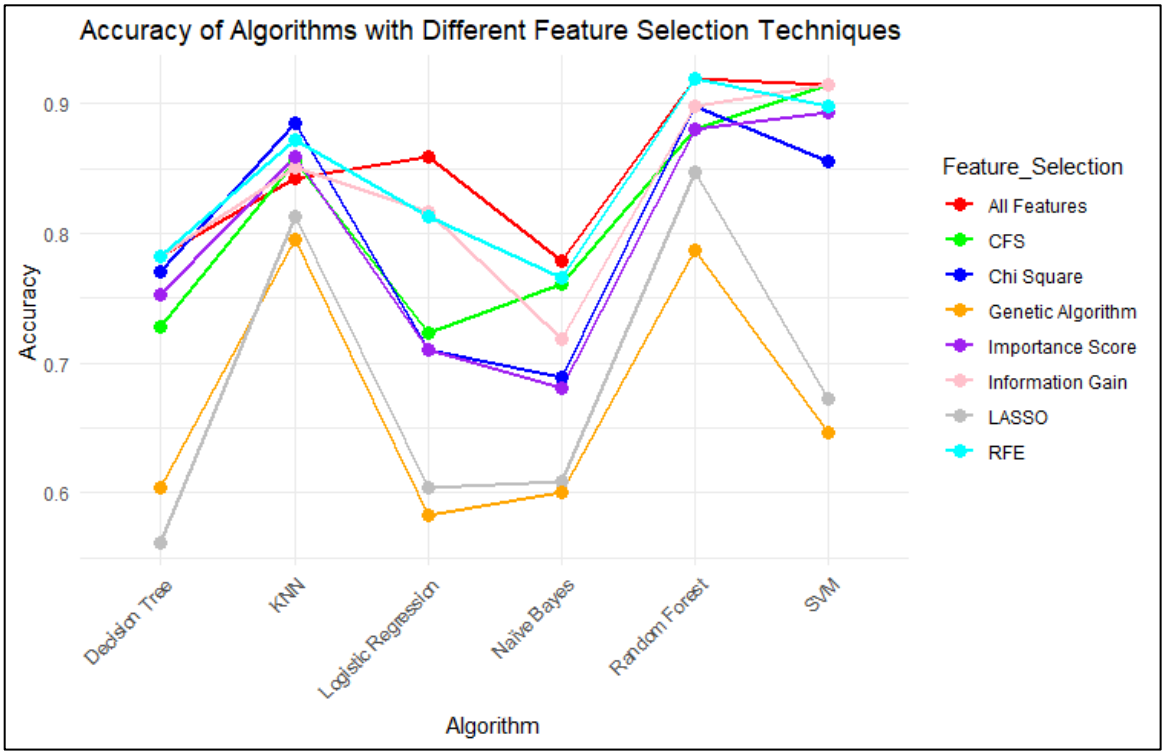




xvii. Visualization of random over sampled dataset



xviii. Visualization of both over and under sampled dataset



15. Conclusions:

This study delves into the complexities of employee attrition within organizational contexts and highlights the potential of machine learning techniques, including Random Forest, Support Vector Machine (SVM), and k-Nearest Neighbors (kNN), in predictive analytics. Through an extensive analysis of diverse employee characteristics and historical attrition data, our objective was to extract actionable insights to inform strategic decision-making and retention initiatives.

Our analysis underscores the impressive predictive capabilities of machine learning models such as Random Forest, SVM, and kNN. These models demonstrated commendable accuracy, sensitivity, and specificity in forecasting employee attrition, revealing their ability to discern underlying patterns and accurately predict turnover risk. Leveraging these models enables organizations to proactively identify employees at risk of attrition and implement targeted interventions, thereby fostering a more stable and engaged workforce.

Furthermore, our investigation identified key predictors influencing attrition likelihood, encompassing departmental affiliation, business travel frequency, job role, and overtime work. Recognizing the significance of these predictors empowers organizations to tailor retention strategies effectively, addressing underlying issues and bolstering employee satisfaction and retention rates.

Hyperparameter tuning emerged as a crucial aspect in optimizing model performance across all algorithms. Fine-tuning parameters such as tree depth, regularization parameter, and number of neighbors (k) allowed us to enhance predictive accuracy and develop more robust predictive models. This optimization process underscores the critical role of parameter selection in maximizing model effectiveness and generalizability.

After careful consideration, it is decided to implement the Random Forest algorithm with LASSO selected features. This decision is based on a strategic balance between model performance, interpretability, and computational efficiency. By leveraging the strengths of both methodologies, we aim to develop a robust and interpretable model that offers actionable insights for strategic decision-making and retention initiatives.

Looking forward, our study suggests several avenues for future research. Longitudinal analyses tracking changes in employee turnover patterns over time can yield insights into temporal trends and dynamics, facilitating proactive intervention strategies. Additionally, integrating external data sources such as economic indicators and industry benchmarks into predictive models can further enhance accuracy and contextualize attrition predictions within broader organizational and market dynamics.

In conclusion, the application of machine learning techniques, including Random Forest, SVM, and kNN, holds significant promise for addressing the challenge of employee attrition. By leveraging predictive models and actionable insights derived from data-driven analyses, organizations can navigate workforce dynamics effectively, fostering a culture of engagement, stability, and growth.

References:

i. *Praveen Ranjan Srivastava, Prajwal Eachempati, Intelligent Employee Retention System for Attrition Rate Analysis and Churn Prediction, JGIM, 2021*
ii. *Shobhanam Krishna and Sumati Sidharth, HR Analytics: Employee Attrition Analysis using Random Forest, IJPE, 2022*
iii. *Francesca Fallucchi , Marco Coladangelo , Romeo Giuliano and Ernesto William De Luca, Predicting Employee Attrition Using Machine Learning Techniques, IEEE access, 2021*
iv. *N. B. Yahia, J. Hlel and R. Colomo-Palacios, "From Big Data to Deep Data to Support People Analytics for Employee Attrition Prediction," in IEEE Access, vol. 9, pp. 60447-60458, 2021, doi: 10.1109/ACCESS.2021.3074559.*
v. *Raza A, Munir K, Almutairi M, Younas F, Fareed MMS. Predicting Employee Attrition Using Machine Learning Approaches. Applied Sciences. 2022; 12(13):6424. https://doi.org/10.3390/app12136424*
vi. *Xu, Z., Song, B. (2006). A Machine Learning Application for Human Resource Data Mining Problem. In: Ng, WK., Kitsuregawa, M., Li, J., Chang, K. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2006. Lecture Notes in Computer Science(), vol 3918. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11731139_99*
vii. *Wardhani, Fitri & Lhaksmana, Kemas. (2022). Predicting Employee Attrition Using Logistic Regression With Feature Selection. Sinkron. 7. 2214-2222. 10.33395/sinkron.v7i4.11783.*
viii. *K. Mridha, R. N. Shaw, and A. Ghosh, "Study and Prediction Analysis of the Employee Turnover using Machine Learning Approaches," in IEEE Xplore, 2021.*

ix. *R. Boutaba et al., "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," Journal of Internet Services and Applications, 2018.*

x. *N. P. S., P. S. Aithal, G. B. J., S. Soans, and H. Jayaraj, "A Study on Employee Retention as a Tool for Improving Organizational Effectiveness," International Journal of Management, Technology, and Social Sciences (IJMTS), 2021.*

xi. *K. Alshehhi, S. Bin Zawbaa, A. A. Abonamah, and M. U. Tariq, "Employee retention prediction in corporate organizations using machine learning methods," Academy of Entrepreneurship Journal, 2021.*

xii. *V. Kakulapati and S. Subhani, "Predictive Analytics of Employee Attrition using K-Fold Methodologies," International Journal of Management Science and Computing (IJMSC), 2023. https://doi.org/10.5815/ijmsc.2023.01.03*

xiii. *A. K. Dubey, I. Maheshwari, and A. Mishra, "Predict Employee Retention Using Data Science," International Journal of Electrical, Electronics and Computer Systems Engineering (IJEECSE), 2018.*

xiv. *P. R. Srivastava and P. Eachempati, "Intelligent Employee Retention System for Attrition Rate Analysis and Churn Prediction," Journal of Global Information Management, 2021.*

xv. *M. Lazzari, J. M. Alvarez, and S. Ruggieri, "Predicting and explaining employee turnover intention," International Journal of Data Science and Analytics, 2022.*

xvi. *Younis, S., Ahsan, A. & Chatteur, F.M. An employee retention model using organizational network analysis for voluntary turnover. Soc. Netw. Anal. Min. 13, 28 (2023). https://doi.org/10.1007/s13278-023-01031-w*

xvii. *M. Karthika, Employee Performance Prediction Using EPP Framework, International Journal of Scientific Research in Computer Science Engineering and Information Technology, December 2020, DOI:10.32628/CSEIT206654*

xviii. *P. S., Nethravathi and Aithal, P. S. and J., Gayathri Babu and Soans, Sonia and Jayaraj, Honey, A Study on Employee Retention as a Tool for Improving Organizational Effectiveness (September 30, 2021). International Journal of Management, Technology, and Social Sciences (IJMTS), 6(2), 121-132. ISSN: 2581-6012.*

xix. *Halim, Z., Maria, Waqas, M. et al. Identifying factors for employee retention using computational techniques: an approach to assist the decision-making process. SN Appl. Sci. 2, 1612 (2020). https://doi.org/10.1007/s42452-020-03415-5*