INTRODUCTION

Given train dataset contains various features labelled as 'X_0' to 'X_19', with the last column named 'class', in total we have 21 variables. These columns represent different numerical and categorical features of the data. ‘X_0’, ‘X_5’ and ‘X_16’ are categorical variables in data, whereas rest others belong to numerical. ‘class’ is the target variable of this dataset which contains, 0: Negative and 1: Positive values. Total 8000 observations are present in this data. Given the binary nature of the 'class' column, it is suitable for binary classification algorithms such as logistic regression, decision trees, and ensemble methods such as Random Forest and eXtreme Gradient Boosting (XGBoost). Exploratory data analysis, visualization, and feature engineering are performed to better understand the relationships between features and the target variable.

EXPLORATORY DATA ANALYSIS

Initially, check for any missing values in data, after performing this I noticed there are no missing values in the data. Secondly, we can perform inferential statistics of variables in the dataset. Below table shows the Mean and Median of some variables.

Table 1: Mean & Median values of some variables

Variable	Mean	Median
X_1	20.540	18.659
X_2	5.250	5.277
X_7	79.726	79.763
X_10	0.0005	0.0004
X_12	1015.281	1015
X_18	3.541	3.311

Now we can visualize the density distribution of numerical variables in the data.

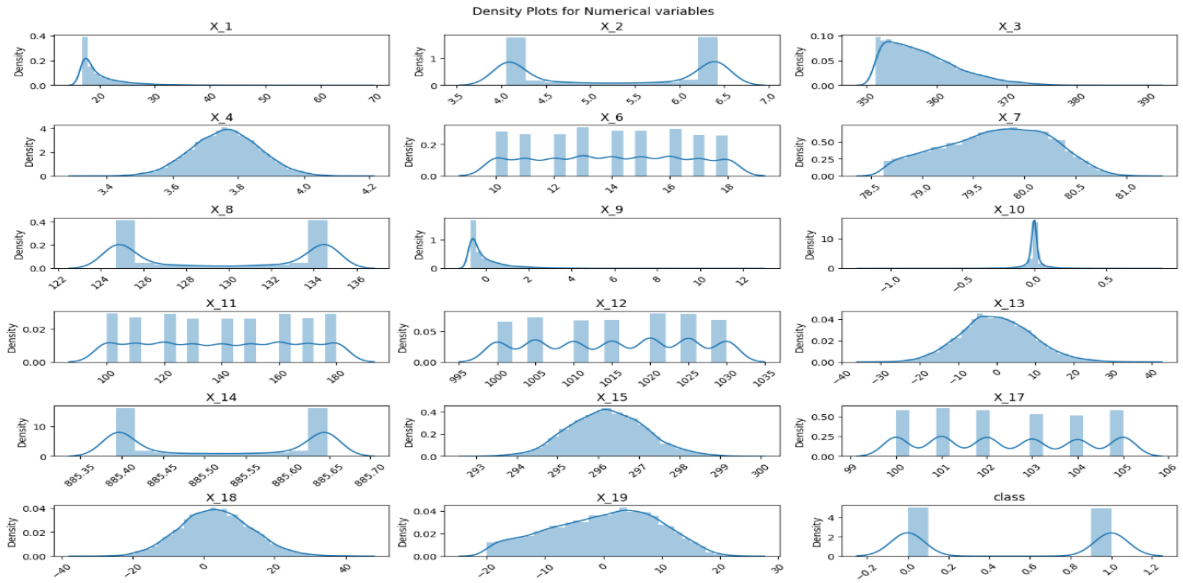

Figure 1: Density plot for numerical variables in data

From the above Figure 1, we can notice that distribution of 'X_1', 'X_3' and 'X_9' is skewed, whereas for other variables it seems normal. Now, we will visualize the count plots of categorical variables in dataset.

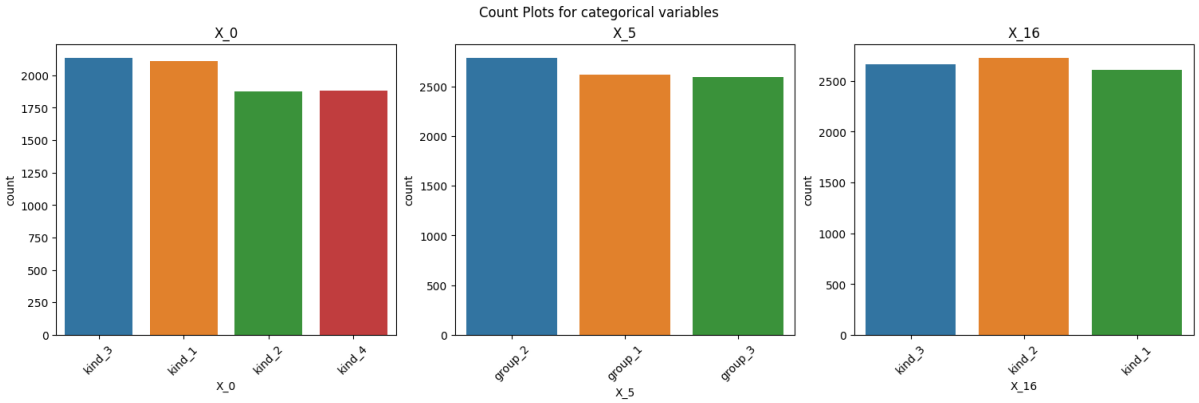

Figure 2: Count plots for Categorical Variables in data

From Figure 2, we can notice that there are 4 kinds in 'X_0' variable, 3 groups in 'X_5' variable and 3 kinds in 'X_16' variable.

As we notice that 'X_1', 'X_3' and 'X_9' are skewed, this shows there are outliers present in these variables. We remove those outliers, by just having data lies is in the upper and lower boundaries.

$$\text{Upper Boundary} = Q3 + IQR * 1.5$$

$$\text{Lower Boundary} = Q1 - IQR * 1.5$$

Where Q1 is first quantile, Q3 is third quantile and IQR is Inter Quantile Region.

CORRELATION:

Before performing any Machine learning it is important to check if there is any correlation between variable. From below Figure 3, we can notice that 'X_11' and 'X_17' are highly correlated with values -0.97, 'X_11' and 'X_12' also have correlation value of 0.80. 'X_12' is having -0.74 correlation value with both 'X_17' and 'X_6'.

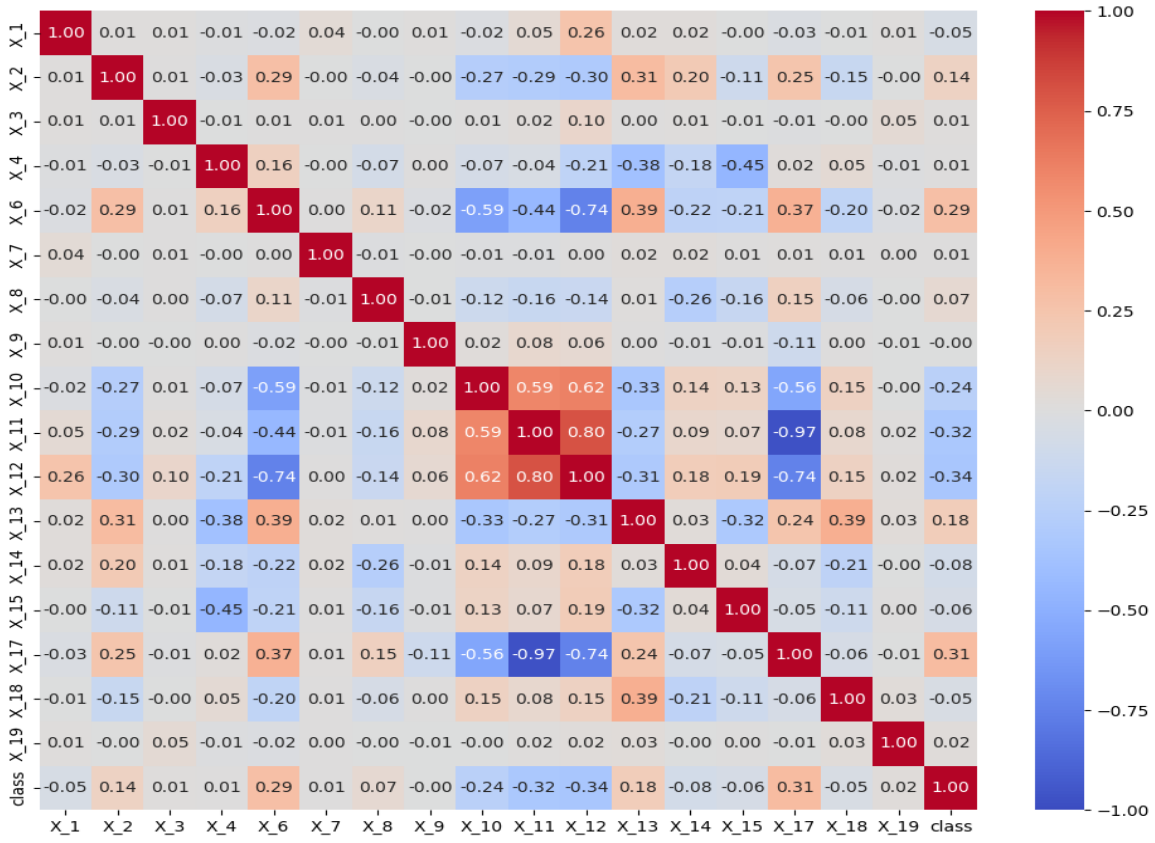

Figure 3: Correlation matrix of data

FEATURE SELECTION & ENGINEERING FOR MODELLING

Feature selection refers to selecting appropriate features for the data modelling, Feature Engineering refers to transforming the data into similar scales. These two operations are crucial and often leads to most effective model performance.

From fig 3, we have got correlation values of each variable w.r.t other variables, having highly correlated variables in the model might yield to poor performance. Considering this, we have to either drop any one of the correlated variables or perform L1 regularization operation to remove correlation. In this analysis, removing correlated variable is chosen, to reduce the computational complexity. Thus, I chose to drop 'X_6' and 'X_17', from the independent variables, as they are highly correlated with 'X_12' & 'X_11'. As usual 'class' is the target variable for the model.

Each variable of the data is on different scales, building model with different scales in data might not perform well. So, we use StandardScaler(), to standardise numerical data and LabelEncoder(), to encode categorical data. LabelEncoder assigns unique value for each category in variable.

StandardScaler is given by $Z = \frac{X_i - \mu}{\sigma}$, Where X_i- value of observation, μ - Mean of the variable and σ – standard deviation of the variable.

LOGISTIC REGRESSION

Logistic Regression is a statistical model commonly used for binary classification problems, where the outcome variable is categorical and has two classes. It models the probability that an instance belongs to a particular class. The logistic regression model uses the logistic function, also known as the sigmoid function, to model the relationship between the input features and the probability of the positive class (class 1).

The sigmoid function is defined as: $\sigma(z) = \frac{1}{1+e^{-z}}$, where z is a linear combination of the input features. The linear combination, denoted as z, is calculated as the weighted sum of the input features plus a bias term:$z = b0 + b1 \cdot x1 + b2 \cdot x2 + \cdots + bn \cdot xn$. Here $x1, x2,\ldots,xn$ are the input features, and $b0,b1,\ldots,bn$ are the coefficients (weights) associated with each feature.

The model is trained by minimizing a cost function, often the logistic loss or cross-entropy loss, which penalizes deviations between the predicted probabilities and the actual class labels. Training typically involves using optimization algorithms such as gradient descent.

Now we train the Logistic Regression model, as it is considered as one of the best base models for the binary classification. So, we fit this model for train data, there after we predict the values to check the model performance on both train data and validation data. Below table shows the accuracy and F1-score of model on both training and validation data.

Table 2: Accuracy & F1-Score of Train & Validation data for Logistic Regression Model

	Accuracy	F1-Score
Train data	0.6494	0.6496
Validation data	0.6395	0.6379

From Table2, we can notice that accuracy & F1-score is around 0.65 for train data and around 0.64 for validation data. With these values, we can conclude that model is neither overfitting nor underfitting. However, accuracy and f1-score are not great as they capture only 65% of the data correctly, so now we try to build other base model such as decision tree to notice any improvements in model performance.

DECISION TREES

A Decision Tree is a supervised machine learning algorithm used for both classification and regression tasks. The tree structure consists of nodes and leaves. Nodes represent decision points based on feature conditions, and leaves represent the final predicted class or regression value. A decision tree is constructed recursively by selecting the best split at each node until a stopping condition is met. The decision tree creates explicit rules that can be easily interpreted and understood. Each path from the root to a leaf corresponds to a decision rule.

At each node, the decision tree algorithm selects the feature that best splits the data into subsets, aiming to maximize the homogeneity (purity) of each subset. For classification, common splitting criteria include Gini impurity and entropy. Decision trees can capture non-linear relationships between features and the target variable, making them versatile for a wide range of datasets. Decision trees can also provide feature importance and handle missing values, due

to these reasons, we can rely on decision trees at times. Now we build Decision Tree model with hyperparameters such as:

1. **criterion:** This hyperparameter specifies the criterion used for splitting nodes. In the context of a Decision Tree, the two common criteria are:
    - **'gini'**: Gini impurity (a measure of how often a randomly chosen element would be incorrectly classified).
    - **'entropy'**: Information gain using entropy (a measure of disorder or impurity in a set).
2. **max_depth:** Maximum depth of the tree. It controls the maximum depth from the root node to the furthest leaf node. Setting it to **None** allows nodes to expand until they contain fewer samples than **min_samples_split** or no splits result in a node with fewer than **min_samples_split** samples.
    - Values: **None**, 5, 10, 15.
3. **min_samples_split:** The minimum number of samples required to split an internal node. If a node has fewer samples than **min_samples_split**, it won't be split.
    - Values: 2, 5, 10.
4. **min_samples_leaf:** The minimum number of samples required to be at a leaf node. If a leaf node has fewer samples than **min_samples_leaf**, it may be pruned and joined with a neighboring leaf node.
    - Values: 1, 2, 4.

Using GridSearchCV, we get the best hyperparameter values that gives optimal performance of the model. Best Hyperparameters are: **'criterion'**: 'entropy', **'max_depth'**: 5, **'min_samples_leaf'**: 1, **'min_samples_split'**: 5.

Table 3: Accuracy & F1-score of Train & Validation data for Decision Tree model

	Accuracy	F1-Score
Train data	0.6987	0.5866
Validation data	0.7000	0.5902

From Table 3, we can notice accuracy and f1-score of decision tree model with best hyperparameters. This model is producing better accuracy that logistic regression model, however it is not producing great f1-score. For this reason, we now build ensemble methods such as Random Forest and eXtreme Gradient Boosting (XGBoost).

RANDOM FORESTS

Random Forest is an ensemble learning technique that builds multiple decision trees during training and merges their predictions for more robust and accurate results. It is widely used for both classification and regression tasks. The key idea behind Random Forests is to introduce randomness in the training process to create diverse trees, which, when combined, lead to a more powerful and stable model. During the training process, each tree in the Random Forest is built on a random subset of the training data. This is achieved through bootstrap sampling, where each tree is trained on a randomly selected subset of the original data with replacement. The combination of bootstrapping and feature randomness helps reduce overfitting, making Random Forests less sensitive to noise and outliers in the data. Since each tree is trained independently, Random Forests can be parallelized, making them efficient for training on large datasets.

High accuracy, robustness, versatility, handling missing values are some of the major reasons to opt random forest models. We build random forest model with hyperparameters

1. **n_estimators:** This hyperparameter represents the number of decision trees in the Random Forest. Increasing the number of trees generally improves the model's performance up to a certain point, but it also increases computational complexity.
    - Values: 50, 100, 200.
2. **max_depth:** Maximum depth of the individual decision trees in the Random Forest. It controls the maximum depth from the root node to the furthest leaf node in each tree. Setting it to **None** allows the trees to expand until they contain fewer samples than **min_samples_split** or no splits result in a node with fewer than **min_samples_split** samples.
    - Values: **None**, 10, 20.
3. **min_samples_split:** The minimum number of samples required to split an internal node in an individual decision tree. If a node has fewer samples than **min_samples_split**, it won't be split.
    - Values: 2, 5, 10.
4. **min_samples_leaf:** The minimum number of samples required to be at a leaf node in an individual decision tree. If a leaf node has fewer samples than **min_samples_leaf**, it may be pruned and joined with a neighboring leaf node.
    - Values: 1, 2, 4.

Using GridSearchCV, we get the best hyperparameter values that gives optimal performance of the model. Best Hyperparameters are: **max_depth**=10, **min_samples_leaf**=2, **min_samples_split**=5, **n_estimators**=50

Table 4: Accuracy & F1-Score for Train and Validation data for Random Forest Model

	Accuracy	F1-Score
Train data	0.7031	0.6129
Validation data	0.7045	0.6120

From Table 4, we can notice that accuracy is around 0.703 for both train and validation datasets and f1-score around 0.612 using Random Forest model. We can notice that this model is performing better than decision tree in both metrics, when it comes to logistic regression model f1-score is less for random forest model comparatively.

eXtreme Gradient Boosting (XGBoost)

XGBoost, which stands for eXtreme Gradient Boosting, is a powerful and efficient machine learning algorithm that belongs to the family of gradient boosting methods. It has gained popularity and widespread use in various data science competitions and real-world applications. XGBoost is an implementation of the gradient boosting framework, which builds an ensemble of weak learners (typically decision trees) sequentially, where each new tree corrects the errors of the previous ones. XGBoost incorporates L1 (LASSO) and L2 (Ridge) regularization terms in its objective function. This helps prevent overfitting and contributes to better generalization to unseen data. It employs tree pruning techniques to control the growth of individual trees, preventing them from becoming too deep and overfitting the training data.

High predictive performance, capturing non-linear relationships, reduce overfitting, robustness are some of the major reasons to choose XGBoost model. We now build XGBoost model with hyperparameters:

1. **n_estimators:** The number of boosting rounds or trees to be built. A higher number generally improves the model's performance, but it also increases computation time. Values include 50, 100, 200.
2. **max_depth:** Maximum depth of the decision trees. It controls the depth to which each tree is allowed to grow. Deeper trees can capture more complex relationships but may lead to overfitting. Values include 3, 5, 7.
3. **learning_rate:** Step size shrinkage used to prevent overfitting. It scales the contribution of each tree. Lower values require more trees but can improve generalization. Values include 0.01, 0.1, 0.2.
4. **subsample:** The fraction of training data to be randomly sampled for building each tree. It helps prevent overfitting by introducing randomness. Values include 0.8, 1.0.
5. **colsample_bytree:** The fraction of features (columns) to be randomly sampled for building each tree. It introduces additional randomness and helps prevent overfitting. Values include 0.8, 1.0.
6. **gamma:** Minimum loss reduction required to make a further partition on a leaf node. It adds regularization by penalizing the creation of new nodes in the tree. A higher value leads to fewer splits. Values include 0, 1.

Using GridSearchCV, we get the best hyperparameter values that gives optimal performance of the model. Best Hyperparameters are: **n_estimators:** 100, **max_depth**=7, **learning_rate:** 0.01, **subsample:** 0.8, **colsample_bytree:** 0.8, **gamma**: 0.

Table 5: Accuracy & F1-Score of train and validation data for XGBoost model

	Accuracy	F1-Score
Train data	0.7019	0.6081
Validation data	0.7005	0.6030

From Table 5, we can notice accuracy and f1-score values on train and validation data for XGBoost model. However, this model is not performing as great as Random Forest model as Accuracy and F1-Score is slightly greater for Random Forest.

ROC Plots for models

ROC curve illustrates the trade-off between sensitivity (true positive rate) and specificity (true negative rate) across different probability thresholds.

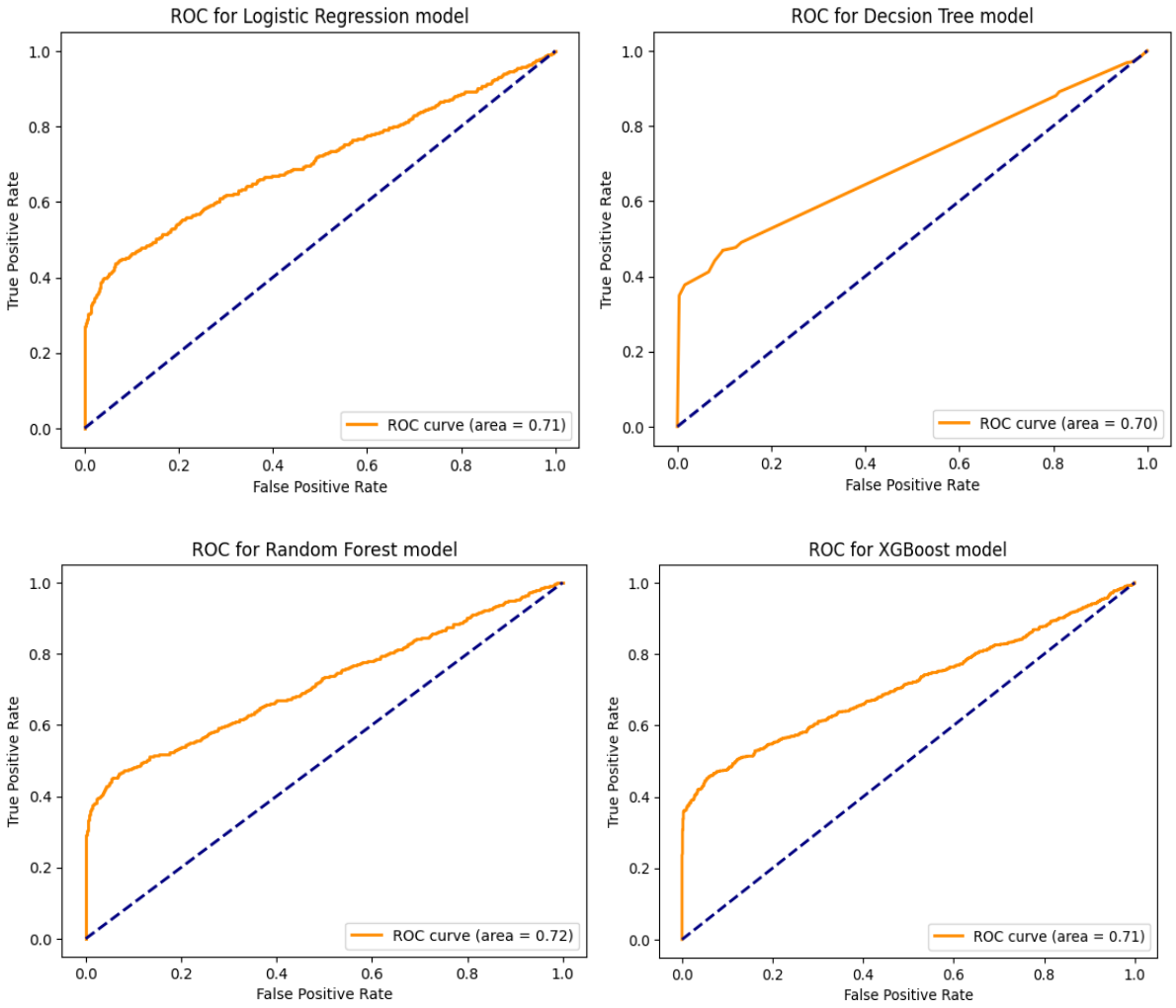

Figure 4: Receiver Operating Characteristic (ROC) Curve for models on validation data

From Figure 4, we have ROC curve area for all the models, Random Forest model gives highest area of 0.72.

CONCLUSION

In this analysis, we initially performed exploratory data analysis on data including handling missing values and removing outliers. There after we built base models such as Logistic Regression and Decision Tree models, with evaluating metrics accuracy and f1-score. As the results were not so great, we built ensemble methods such as Random Forests and XGBoost models. Thereafter, we visualized ROC curve area, after evaluating all these metrics, Random Forest model turns out to be best model for the data.

ASSUMPTIONS

In this analysis, we converted categorical variables using Label Encoder, this should be used when there is hierarchy or ranking, I assumed there is ranking in ‘X_0’, ‘X_5’ and ‘X_16’ and converted them using Label Encoder.