

INDEX

Sr.no		Topics
1		Introduction
	1.1	Problem Statements
	1.2	Understanding Data
2		Methodology Overview
	2.1	Data Preprocessing
	2.2	Data Modeling
	2.3	Evaluation of data
	2.4	Find Predictable output
3		Data Preprocessing and Transform
	3.1	Explore Data
	3.2	Missing value analysis
	3.3	Outlier Analysis
	3.4	Data Visualization
	3.5	Feature Engineering
4		Data Modeling
	4.1	Logistic Regression
	4.2	Decision Tree Classification
	4.3	Random Forest Classification
	4.4	Naïve Bayes Classification
	4.5	Matrices to find accuracy
	4.6	Create predicted variable on test data
5		Conclusion
6		References

INTRODUCTION

At Santander , mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals.

Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan?

1.1 Problem Statement

In this challenge, we need to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

1.2 Understanding Data

Python Libraray Use: Pandas

You are provided with an anonymized dataset containing numeric feature variables, the binary target column, and a string ID_code column. The task is to predict the value of target column in the test set.

First we load dataset and observed dataset

Size of trian.csv : - 200000 rows, 202 Columns (including dependent variable)

Missing Values: No

Size of test.csv : - 200000 rows, 201 Columns

Missing Values: No

Variable list in Train and Test Dataset

Variables	Data_Type	Train	Test
ID_Code	Object	Yes	Yes
Target [0, 1]	Float64	Yes	No
var_0, var_1,...,var_199	Float64	Yes	Yes

METHODOLOGY

2.1 Data Preprocessing

Here we required to built predictive model, therefore before moving towards modeling we have to manipulate data by applying multiple preprocessing techniques like follows:

- 1) **Exploratory analytics**
- 2) **Missing Value Analysis**
- 3) **Outlier Analysis**
- 4) **Feature Engineering**
- 5) **Data Visualization**

2.2 Data Modeling

After Pre-Processing steps , we will move towards data modeling steps. Data modeling is a set of tools and techniques used to understand and analyse how an organization should collect, update, and store data. It is a critical skill for the business analyst who is involved with discovering, analyzing, and specifying changes to how software systems create and maintain information.

As our target variable says about binary data that is about to categorical format we will go with **classification modeling** techniques like:

- 1) **Logistic Regression**
- 2) **Decision Tree Classification**
- 3) **Random Forest Classification**
- 4) **Naïve Bayes Classification**

2.3 Evaluation of Model

As our dataset tells about classification variable and follows classification modeling , so for evaluate best fit model we have to find accuracy of model and for that we choose to go with classification matrices like:

- 1) **Confusion Matrix**
- 2) **Area Under Curve**

2.4 Find Predictable output

After selection of best match model we will apply this model on test dataset and find predictable target variable.

DATA PREPROCESSING

Data preprocessing is a **data** mining technique that involves transforming raw **data** into an understandable format. Real world **data** is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. **Data preprocessing** is a proven method of resolving such issues.

3.1 Data Exploration

Python Library use: pandas, numpy

Exploratory Data Analysis (EDA) is the first step in your data analysis process. This process makes deeper analysis easier because it can help target future searches and begin the process of excluding irrelevant data points and search paths that may turn up no results.

What I have done :

I observed the type of each variable where I have seen that ID_Code is in object type, target variable in float64 format and remaining all 200 variables in int64 format.

As our '**target**' variable showing as binary/categorical values [0,1] we have to convert their type from '**float64**' to '**int64**'.

3.2 Missing Value Analysis

Python Library use: pandas, numpy

Missing data in the training data set can reduce the power / fit of a model or can lead to a biased model because we have not

analysed the behavior and relationship with other variables correctly. It can lead to wrong prediction or classification.

In our dataset we found **no any missing values** in train data and test data.

3.3 Outlier Analysis

Python Library use: pandas, numpy

Outlier is a commonly used terminology by analysts and data scientists as it needs close attention else it can result in wildly wrong estimations. Simply speaking, Outlier is an observation that appears far away and diverges from an overall pattern in a sample.

Here we are using **boxplot.stats in R and IQR in Python** technique for identifying outlier in datasets.

What **boxplot.stats** and **IQR** technique does?

Boxplot.stats internally follows IQR technique.

Then, let's understand **IQR (Interquartile Range)**

The **interquartile range** (IQR) is a measure of variability, based on dividing a data set into quartiles.

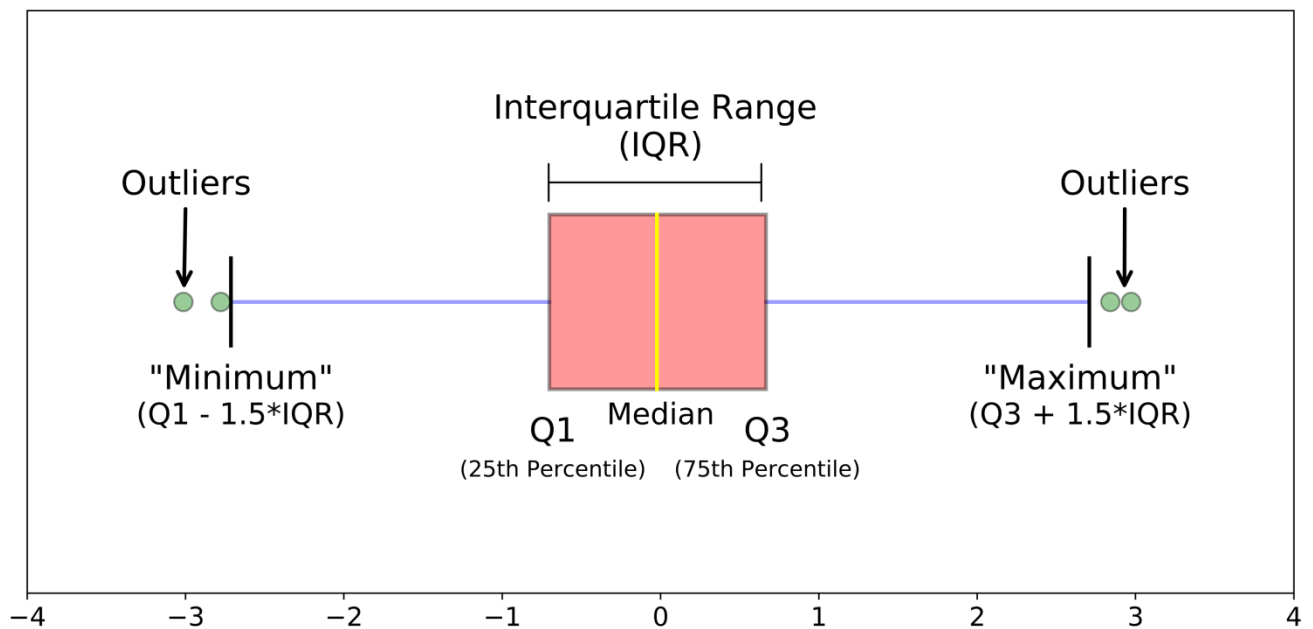
Quartiles divide a rank-ordered data set into four equal parts. The values that divide each part are called the first, second, and third quartiles; and they are denoted by Q1, Q2, and Q3, respectively.

- **Q1** is the "**middle**" value in the *first* half of the rank-ordered data set.
- **Q2** is the **median** value in the set.

- **Q3** is the "**middle**" value in the *second half* of the rank-ordered data set.

The interquartile range is equal to Q3 minus Q1.

For example, consider the following numbers: -2.5, -1, -0.5, 0, 0, 0.5, 1, 2.5. Q1 is the middle value in the first half of the data set. Since there are an even number of data points in the first half of the data set, the middle value is the average of the two middle values; that is, $Q1 = (-1 + -0.5)/2$ or $Q1 = -0.75$. Q3 is the middle value in the second half of the data set. Again, since the second half of the data set has an even number of observations, the middle value is the average of the two middle values; that is, $Q3 = (1 + 0.5)/2$ or $Q3 = 0.75$. The interquartile range is Q3 minus Q1, so $IQR = 0.75 - (-0.75) = 1.5$



BoxPlot Fig

As per seeing in BoxPlot fig we can conclude that above "Maximum" and Below "Minimum" we can consider outliers.

What I have done :

Step 1: First I detect outlier by using **boxplot** technique.

Step 2: Then by applying **IQR formula** I found maximum and minimum value of each variable.

Step 3: after finding minimum and maximum value, I have made 'NAN' to that all **observations** which are above the limit of maximum and below the limit minimum

Step 4: As our problem statement is saying about whether customer make transaction in future or not by considering target value **0 means 'No' and 1 means 'Yes'**. So, we **cant remove 'nan'** rows because it will delete some customer data. So here we try to **impute** this '**nan**' values with Missing value imputation methods.

Step 5: Missing value imputation method like: Mean, Meadian and KNN.

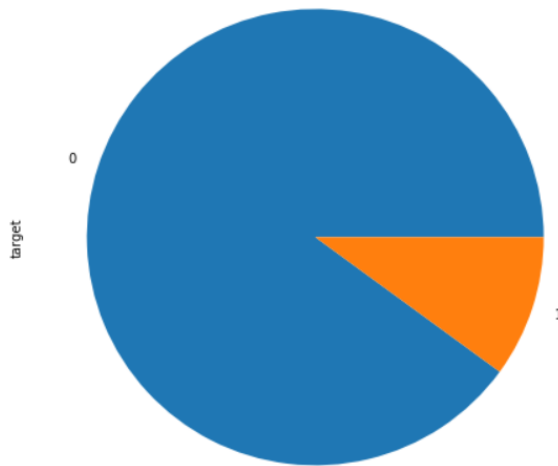
Step 6: But here, best match imputation we find is **Mean** method. So, we will impute 'nan' values with Mean method.

3.5 Data visualization

Python Libraray use: Matplotlib

R libarary use: ggplot

✚ Here first we visualize our **target variable** how its internally distributed.



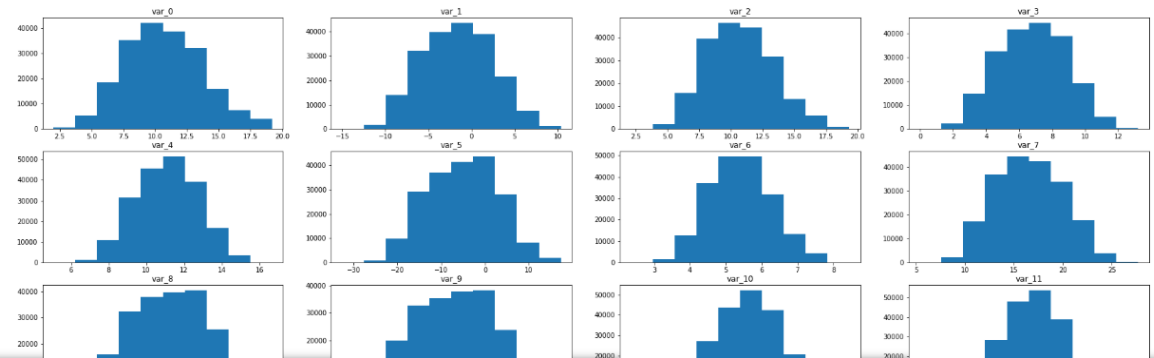
Observed:

Here we can observed in our target varibale more on observation caontains '0' ie 'No'. So we can conclude that around 90% data is '0' and remaining '1'.

✚ Visulizing each **numeric variables distribution over dataset**.
That is of 200 varibles from var_0 to var_199 we are observing how it's flows.

```
In [22]: #histograms are used to check distribution of data
#draw histograms of numeric data in training set
print("Distribution of columns")
plt.figure(figsize=(30,185))
for i,col in enumerate(numerical_features):
    plt.subplot(50,4,i+1)
    plt.hist(train_data[col])
    plt.title(col)
```

Distribution of Columns



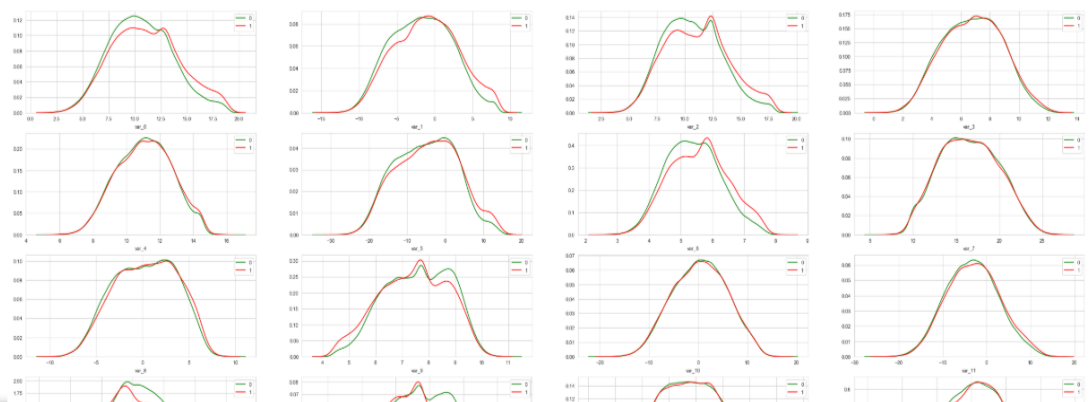
Observed:

Here we can observed all numeric feature variable are normally distributed format.

Here, we are visualizing each **variable distribution** **respective to its target** value i.e each variable distribution from var_0 to var_199 respective to target values 0 and 1.

```
In [23]: import seaborn as sns
print("Distribution of columns per target class")
sns.set_style('whitegrid')
plt.figure(figsize=(40,200))
for i,col in enumerate(numerical_features):
    plt.subplot(50,4,i+1)
    sns.distplot(train_data[train_data['target']==0][col],hist=False,label='0',color='green')
    sns.distplot(train_data[train_data['target']==1][col],hist=False,label='1',color='red')
```

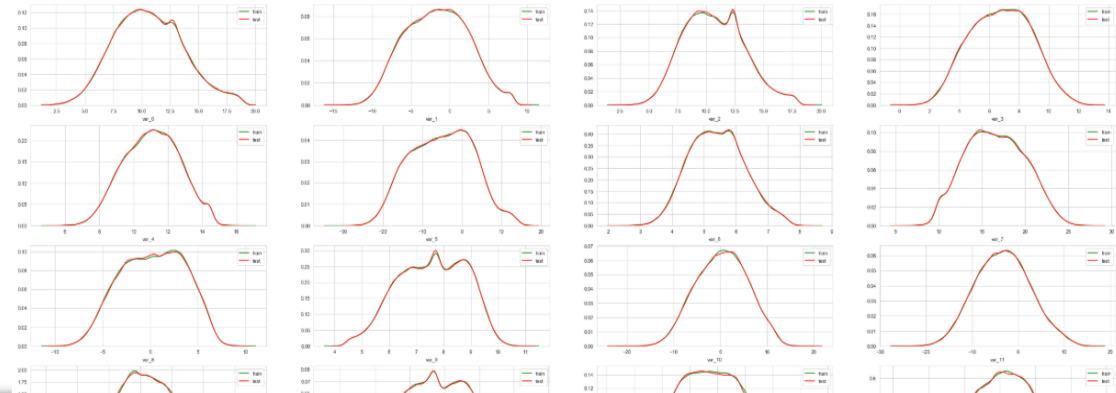
Distribution of columns per target class



Distribution of numeric variables in train and test data

```
In [24]: #Distribution of numeri variables in train and test data
print("Distribution of columns for test and train dataset")
sns.set_style('whitegrid')
plt.figure(figsize=(40,200))
for i,col in enumerate(numerical_features):
    plt.subplot(50,4,i+1)
    sns.distplot(train_data[col],hist=False,label='train',color='green')
    sns.distplot(test_data[col],hist=False,label='test',color='red')
```

Distribution of columns for test and train dataset

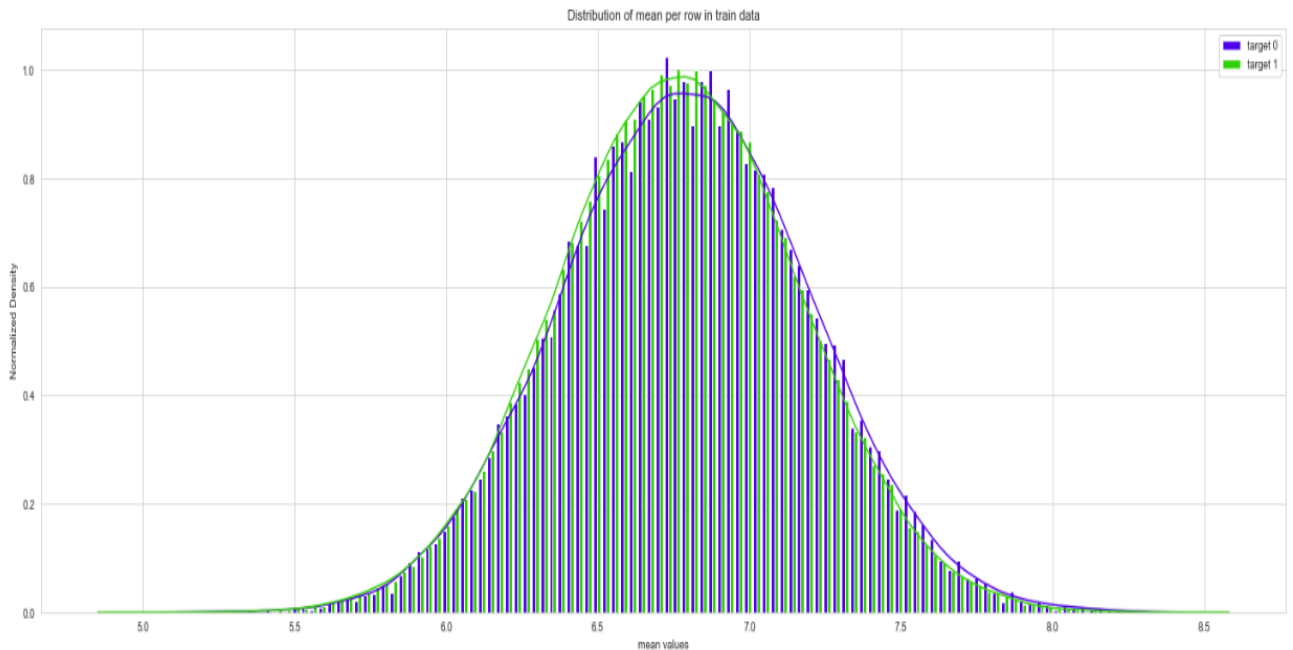


Observed:

Overall Test and Train datasets variables moving together.

✚ Distribution of **mean, median, standard deviation, kurtosis frequency, skewness frequency, min, max** of train dataset for both **targets (0 and 1)**

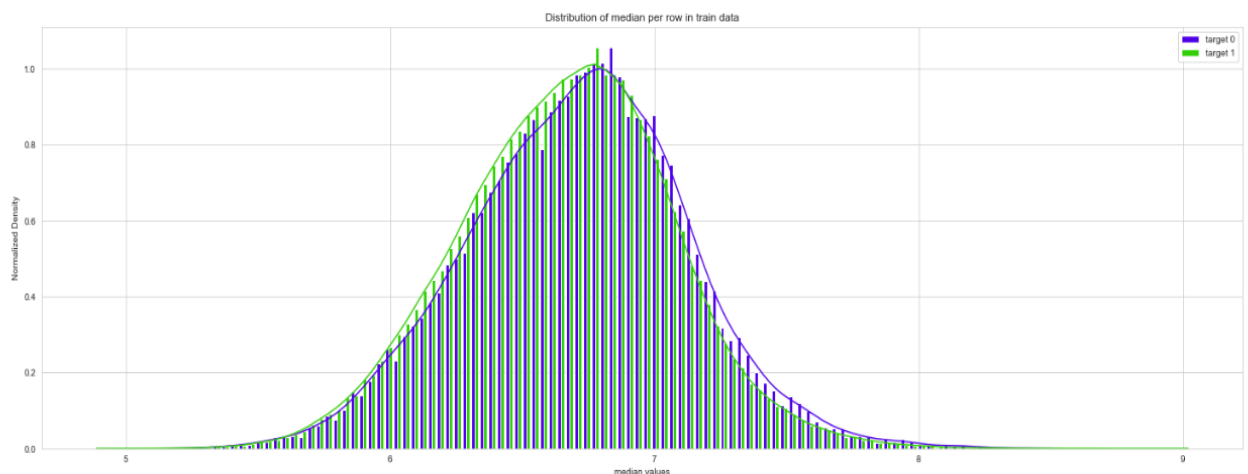
1. **Mean** per row with target 0 and 1



Observed:

Mean of each row respective to target 1 and target 0 are going in some how in simultaneously with range from around 45 to 90.

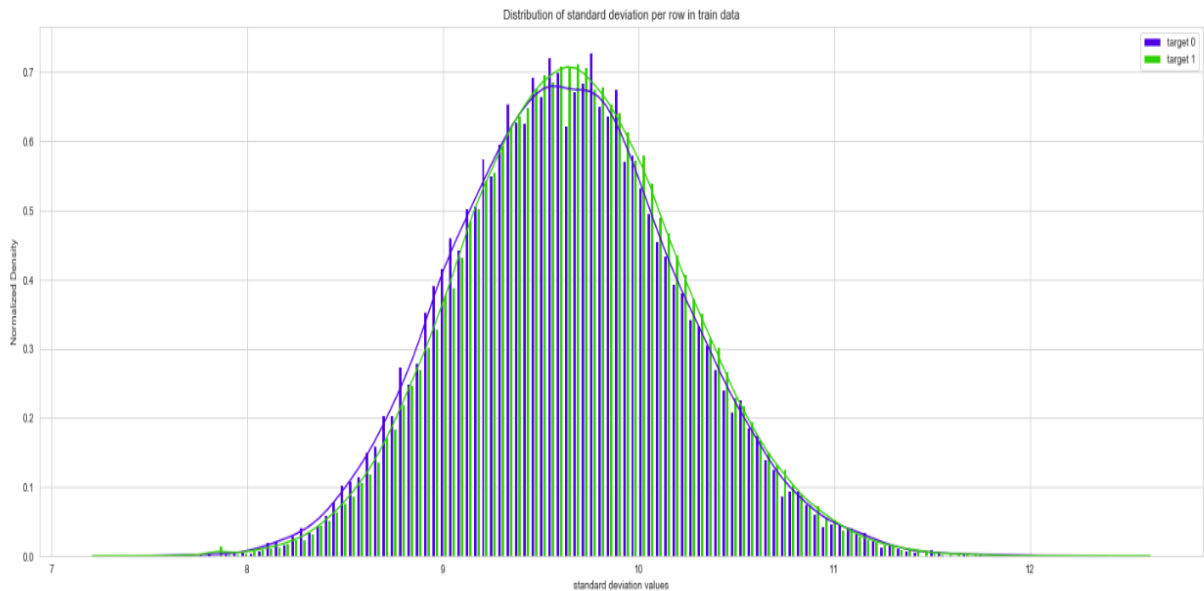
2. **Median** per row with target 0 and 1.



Observed:

Median of each row respective to target 0 range between 5.5- 8.3 and for target 1 range 4-8.2 .

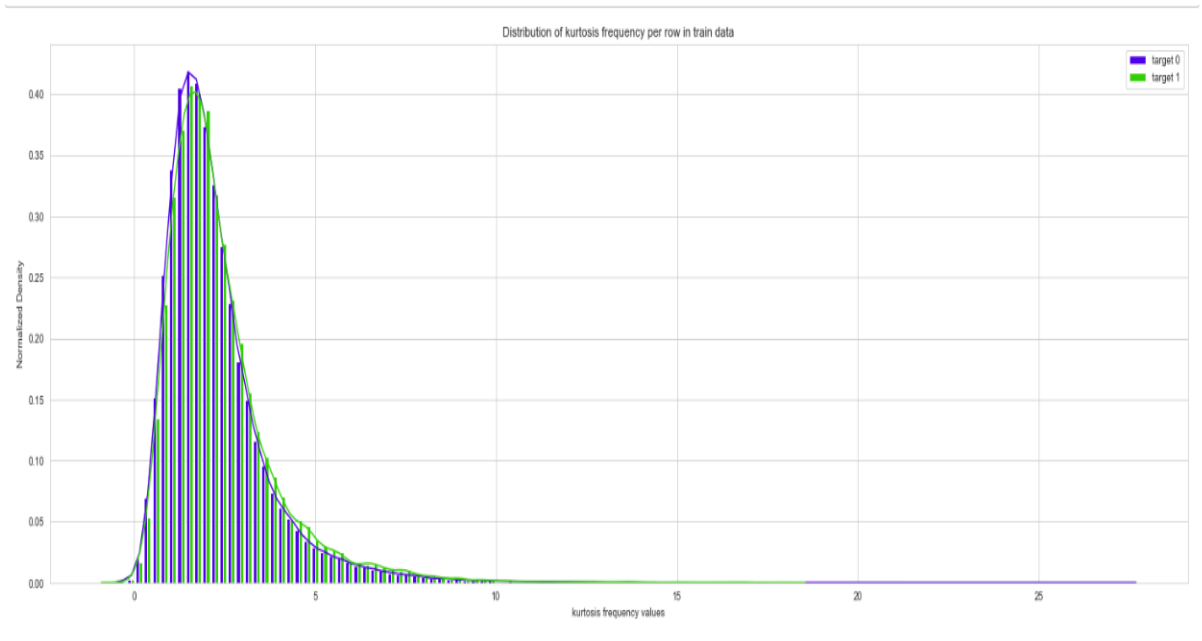
3. Standard Deviation per row with target 0 and 1



Observed:

SD of each row respective to target 0 range between 6.8- 11.8 and for target 1 range 7.3-12.9 .

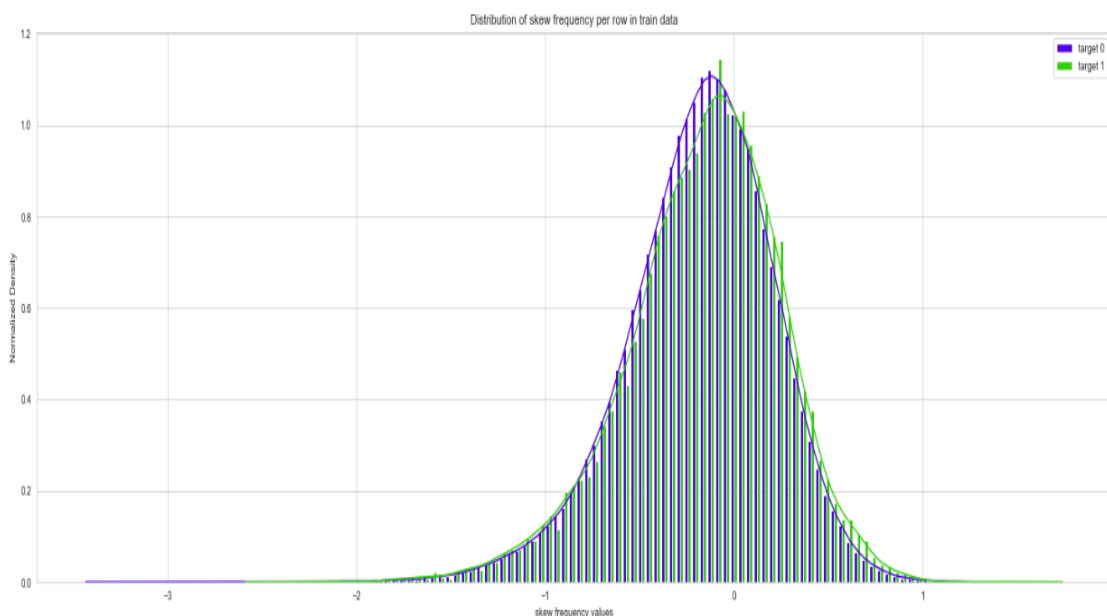
4. kurtosis frequency per row with target 0 and 1



Observed:

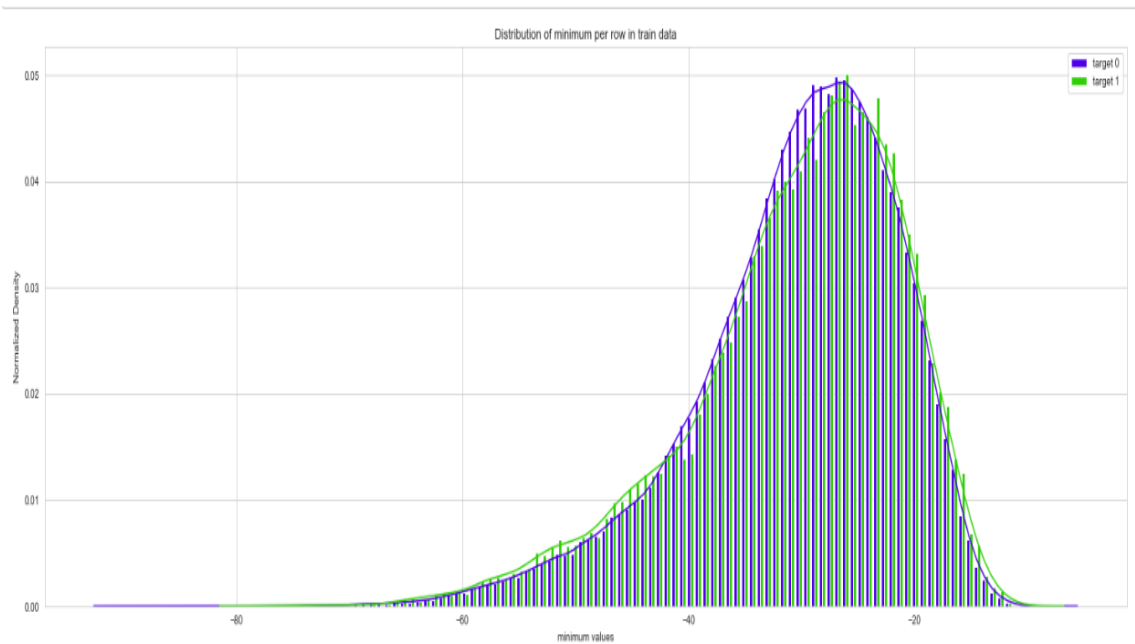
Kurtosis Frequency of each row respective to target 0 range between -0.5 - 10 and for target 1 range -0.3-15 and it right skewed observation.

5. skew frequency per row with target 0 and 1



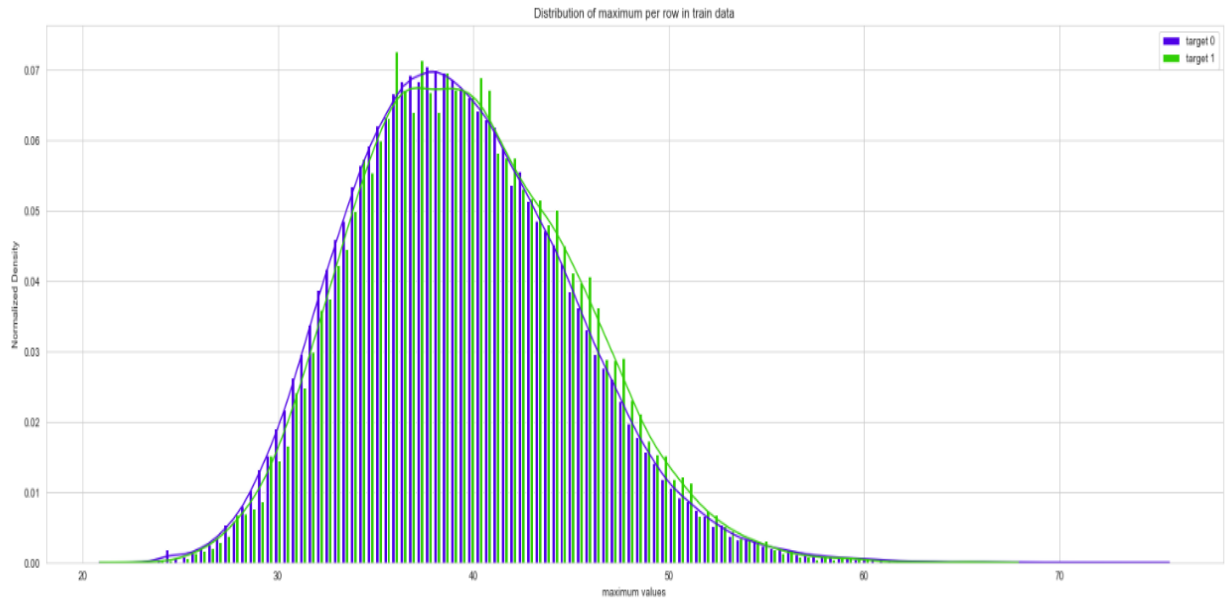
Skewed Frequency of each row respective to target 0 range between -1.8 - 1 and for target 1 range -2.6-1.8 and it left skewed observation

6. **minimum** per row with target 0 and 1



Minimum of each row respective to target 0 range between -0.90 - -10.90 and for target 1 range -80.5-0.90 and it **left** skewed.

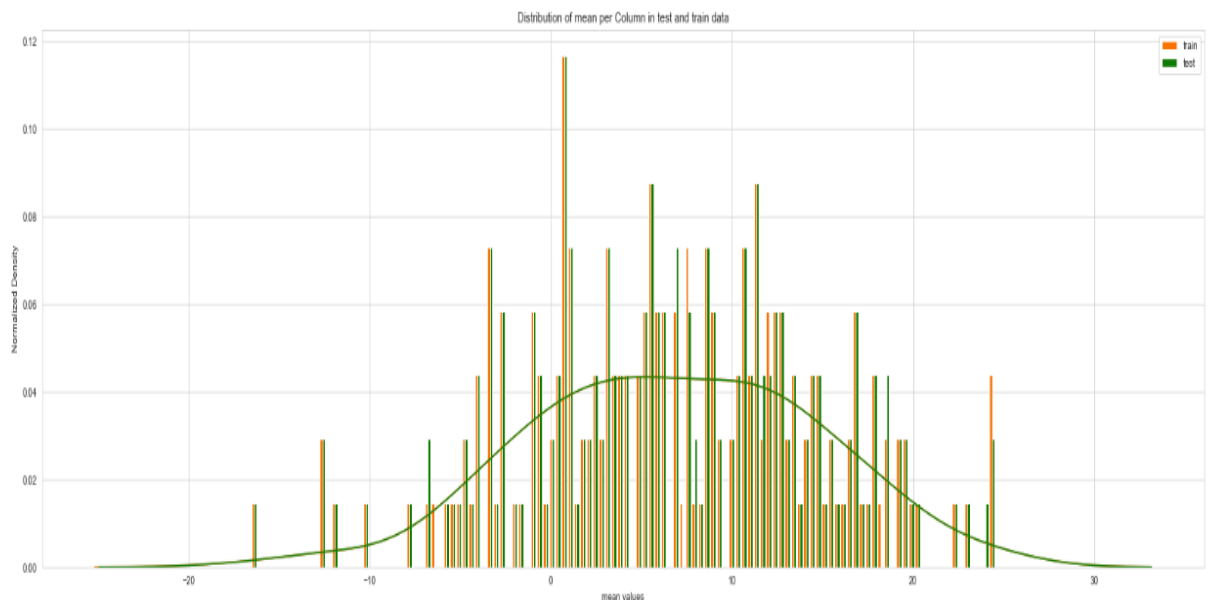
7. **Maximum** value per row with target 0 and 1



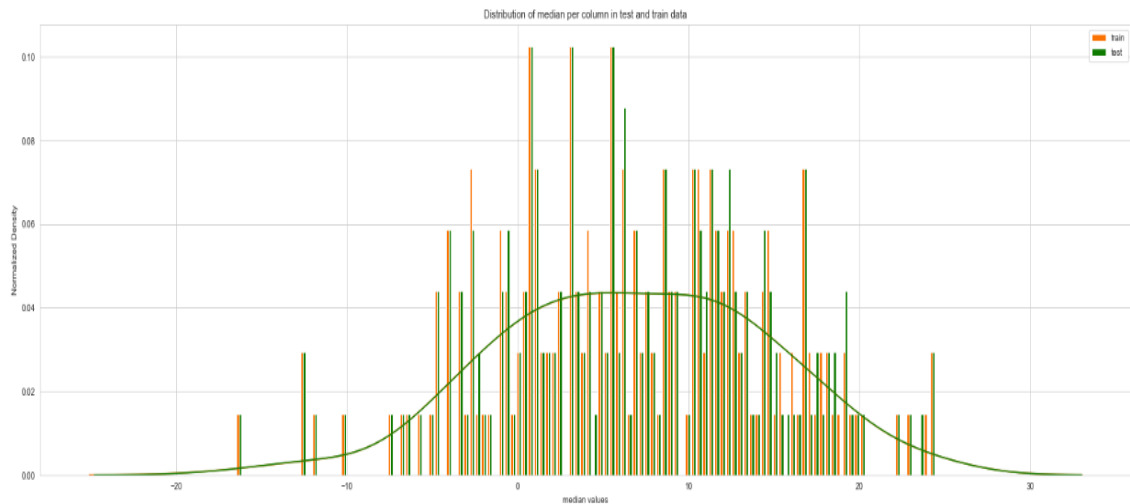
Maximum of each row respective to target 0 range between -0.5 - 10 and for target 1 range -0.3-15 and it **right** skewed.

✚ Distribution of **mean, median, standard deviation, kurtosis frequency, skewness frequency, min, max** of train and test dataset both **column wise**

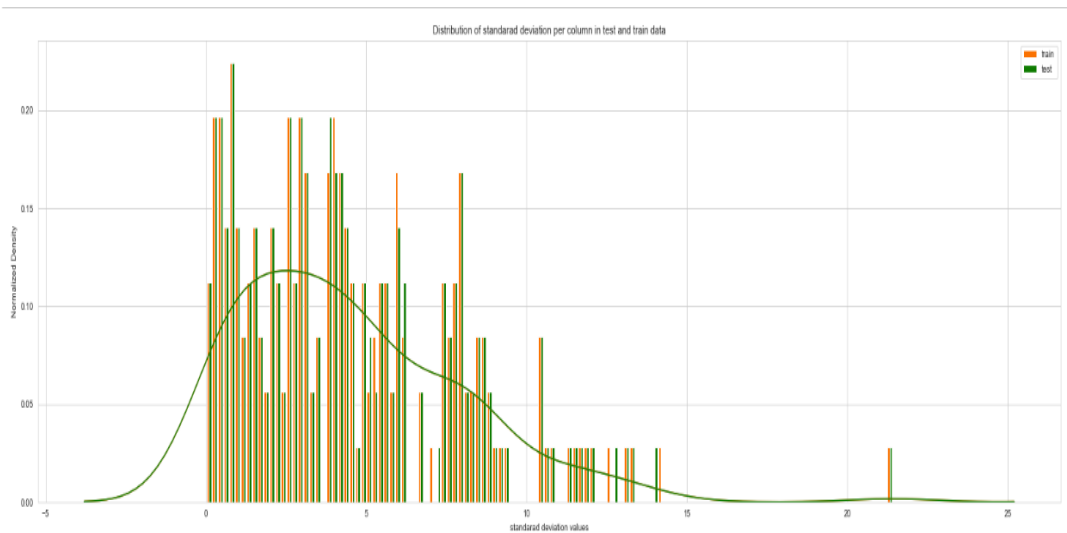
1. **Mean** for train and test dataset both column wise



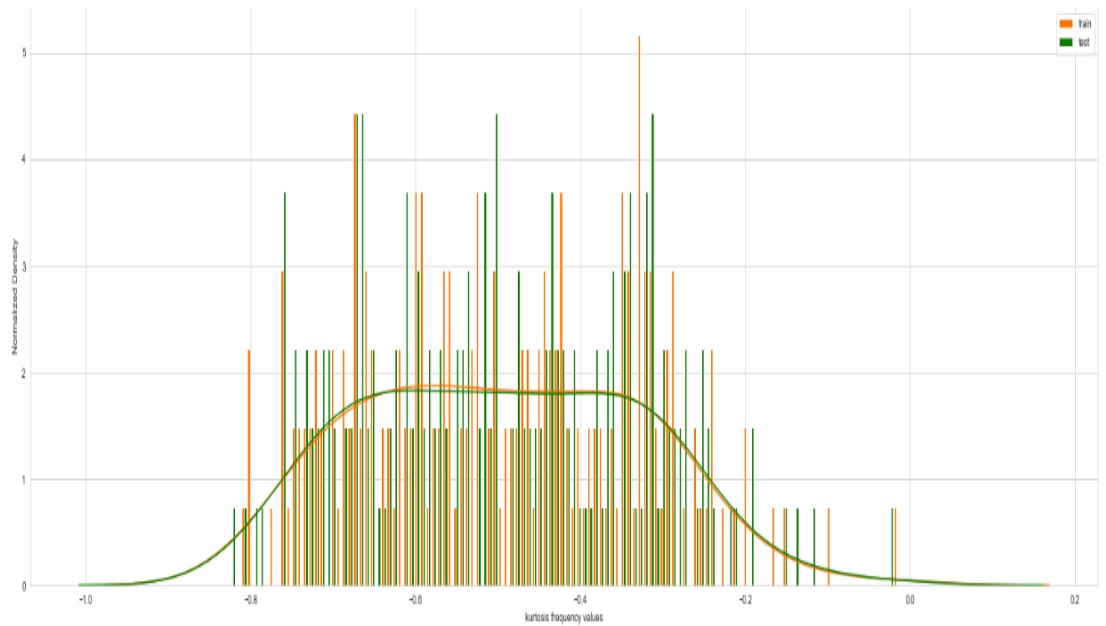
2. **Median** for train and test dataset both column wise



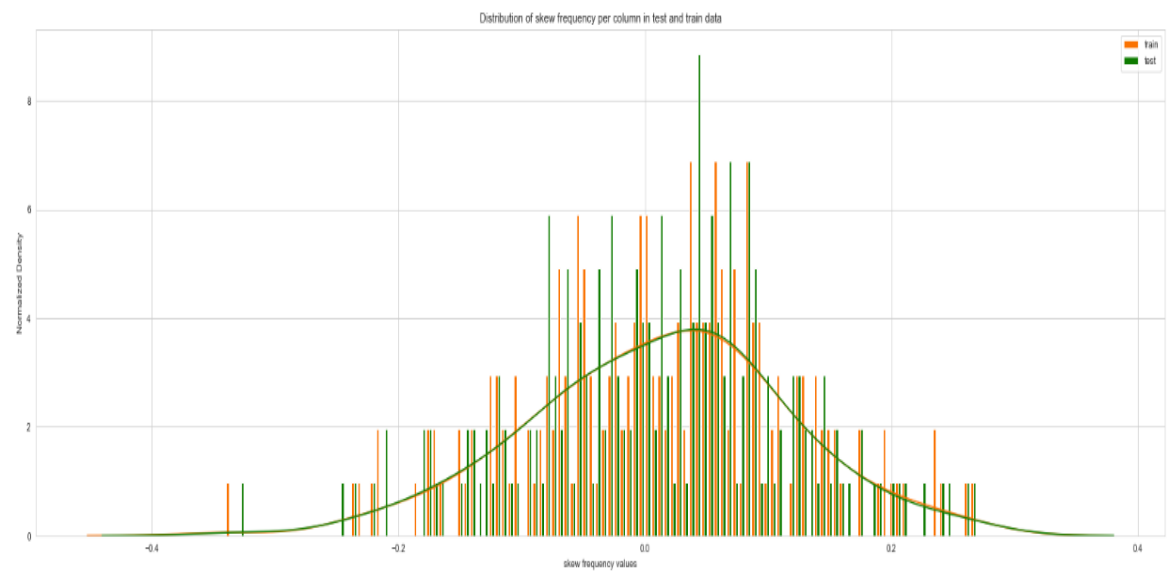
3. **Standard deviation** for train and test dataset both column wise



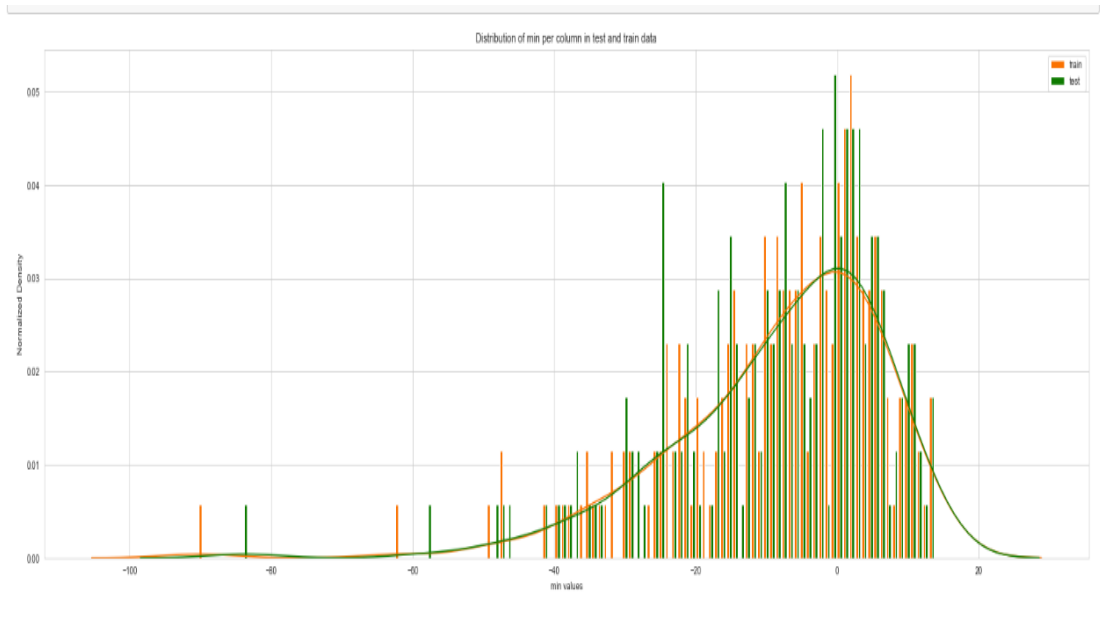
4. **kurtosis frequency** for train and test dataset both column wise



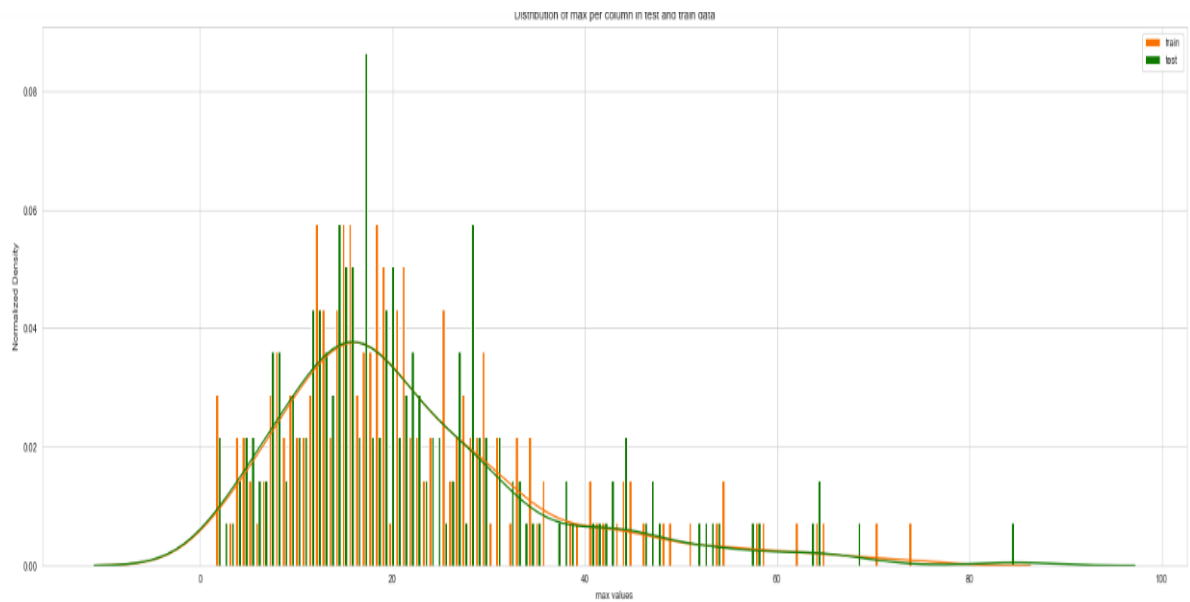
5. skew frequency for train and test dataset both column wise



6. **Minimum** for train and test dataset both column wise



7. **Maximum** for train and test dataset both column wise



3.6 Feature Engineering

Python Library use: numpy, pandas, seaborn, sklearn.preprocessing.MinMaxScaler, sklearn.decomposition i.PCA

R libraray Use: corrgram, dplyr, e1071

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data. Feture Engineering is part of ‘**Transform data**’ in machine learning steps.

1. Feture Selection

In machine learning and statistics, **feature selection**, also known as **variable selection**, **attribute selection** or **variable subset selection**, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for several reasons:

- simplification of models to make them easier to interpret by researchers/users,
- shorter training times,
- to avoid the curse of dimensionality,
- enhanced generalization by reducing overfitting (formally, reduction of variance)

As our datasets contains undefined 200 numerical variables , one Id_code which is object form and target variable is now in int format. For our model development process we no need of

'ID_code' variable so first we have to save this in another variable and then manually we can remove this variable.

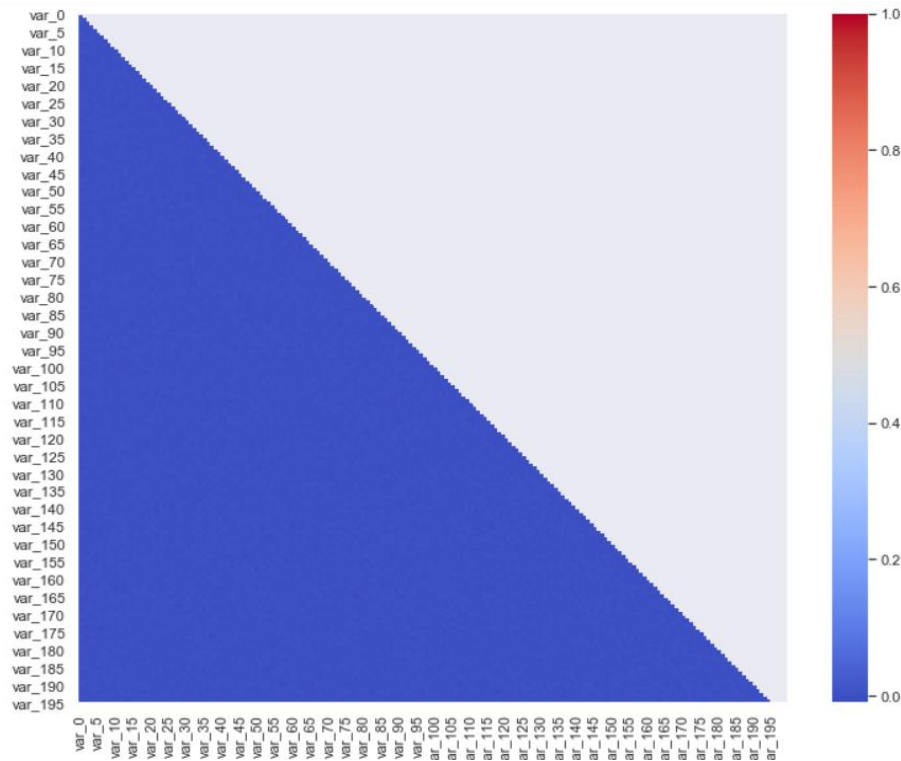
And for numerical features variables we can apply:

1] Correlation

Correlation analysis is a statistical method used to evaluate the strength of relationship between two quantitative variables. A high correlation means that two or more variables have a strong relationship with each other, while a weak correlation means that the variables are hardly related. It ranges from -1.0 to +1.0. The closer r is to +1 or -1, the more closely the two variables are related.

$$\text{Formula for Correlation} = r = \frac{\text{Cov}(X,Y)}{\sigma_x \sigma_y}$$

So in our dataset we draw a Correlation between numeric features variables.



We can say that:

1. We have 200 features that are mostly uncorrelated between them.
2. 200 numerical features that their histograms have a shape like the one of a normal distribution.¶

2] Principal Component Analysis

Principal component analysis is a fast and flexible unsupervised method for dimensionality reduction in data. Its behavior is easiest to visualize by looking at a two-dimensional dataset.

PCA as dimensionality reduction

Using PCA for dimensionality reduction involves zeroing out one or more of the smallest principal components, resulting in a lower-

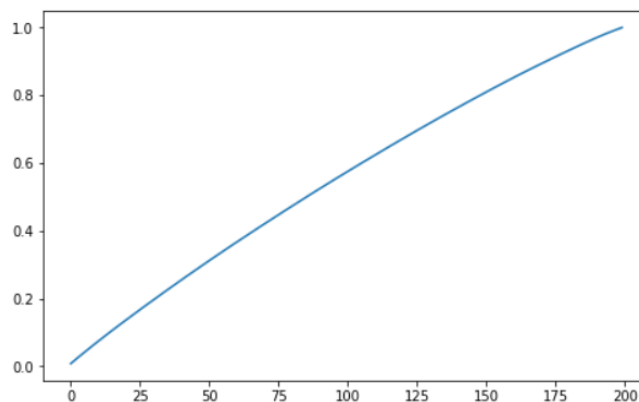
dimensional projection of the data that preserves the maximal data variance.

Here in our dataset we seen that after considering **PCA()** and **transformation** our dataset shape not reduce it **remains same with 200** numerical variables.

Choosing the number of components

A vital part of using PCA in practice is the ability to estimate how many components are needed to describe the data. This can be determined by looking at the cumulative *explained variance ratio* as a function of the number of components.

After `pca.explained_variance_ratio_` we plot data to see which variables are finally get selected here we seen that all variable are lineary plot to graph. Means all considere.



Here we can say that,

1. The line of cumulative sums of explained variance ratio when you PCA the data set, it is indicative of a dataset that has already undergone PCA (get straight line ie $y=x$).
2. We have to go with all 200 variables.

2.Adding Features for Feature Engineering

As or dataset consisting of **anonymous 200 numerical variables (var_0 to var_199)** . so, here we can add features new feature like **mean, median, maximum , minimum , sum, standard deviation, kurtosis, skewness** row wise to both train and test dataset.

Define each term:

Mean: Sum of all obseration/ number of observation

Median : Middle or 50th value of Observation

Maximum: Highest value in range

Minimum : Lowest value in range

Sum: Sumation of all observation

Standard deviation : The standard deviation is a statistic that measures the dispersion of a dataset relative to its mean and is calculated as the square root of the variance.

$$\text{Standard Deviation} = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}}$$

where:

x_i = Value of the i^{th} point in the data set

\bar{x} = The mean value of the data set

n = The number of data points in the data set

Skewness: Skewness is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point.

$$S_k = \frac{\sum_{i=1}^T (x_i - \bar{x})^3}{\sigma^3}$$

Kurtosis: Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution. That is, data sets with high kurtosis tend to have heavy tails, or outliers. Data sets with low kurtosis tend to have light tails, or lack of outliers. A uniform distribution would be the extreme case.

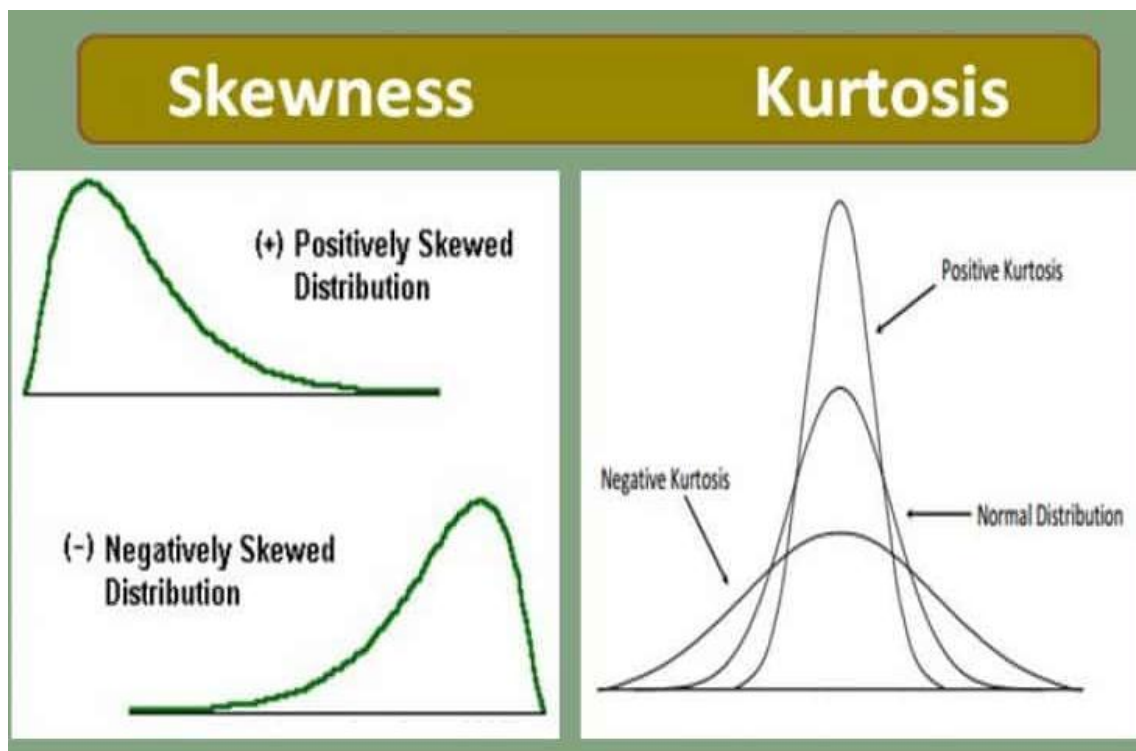
$$\text{Kurtosis} = \frac{\sum_{i=1}^N \frac{(X_i - \bar{X})^4}{N}}{s^4}$$

where,

\bar{X} is the mean,

s is the standard deviation

and N is the sample size



And now our dataset shape will be:

Train dataset: 200000 rows 210 columns.

Test dataset: 200000 rows 209 columns.

MODEL DEVELOPMENT

Data modeling is a process used to define and analyze data requirements needed to support the business processes within the scope of corresponding information systems in organizations. Data models provide a framework for data to be used within information systems by providing specific definition and format. If a data model is used consistently across systems then compatibility of data can be achieved. If the same data structures are used to store and access data then different applications can share data seamlessly.

In **machine learning**, there are **three types**:

1. Supervised Learning

This algorithm consist of a target / outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables). Using these set of variables, we generate a function that map inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Examples of Supervised Learning: **Regression, Decision Tree, Random Forest, KNN, Logistic Regression etc.**

2. Unsupervised Learning

In this algorithm, we do not have any target or outcome variable to predict / estimate. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Examples of Unsupervised Learning: **Apriori algorithm, K-means.**

3. Reinforcement Learning:

Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it trains itself continually using trial and error. This machine learns from past experience and tries to capture the best possible knowledge to make accurate business decisions.

As observing dataset, we find output on based on pattern defined in training dataset, so here we go with **Supervised Learning**.

In **supervised learning**, we have two type:

1] Regression Models

In machine learning, regression algorithms attempt to estimate the mapping function (f) from the input variables (x) to numerical or continuous output variables (y). In this case, y is a real value, which can be an integer or a floating point value. Therefore, regression prediction problems are usually quantities or sizes.

For example, when provided with a dataset about houses, and you are asked to predict their prices, that is a regression task because price will be a continuous output.

As per observing on dataset we can conclude that our dataset is in **descrete format**. Our **target variable has a categoriacal** value which is refering to a classification or descrete class variable.

2] Clasification Models

On the other hand, classification algorithms attempt to estimate the mapping function (f) from the input variables (x) to discrete or categorical output variables (y). In this case, y is a category that the mapping function predicts. If provided with a single or several input variables, a classification model will attempt to predict the value of a single or several conclusions.

For example, when provided with a dataset about houses, a classification algorithm can try to predict whether the prices for the houses “sell more or less than the recommended retail price.”

So for our dataset we are choosing **classification models** like follows:

- 1] **Logistic Regression Model**
- 2] **Decision Tree Classification Model**
- 3] **Random Forest Classification Model**
- 4] **Naïve bayes Classification Model**

First you have split dataset in training and testing dataset where I am considering 80% of data in training.

Then apply model one by one.

4.1 Logistic Regression

Python Library use: `sklearn.linear_model.LogisticRegression`

R library use: `DMwR, caret, DataCombine, inTrees`

Logistic Regression was used in the **biological** sciences in early twentieth century. It was then used in many social science applications. Logistic Regression is used when the **dependent variable(target) is categorical**.

For example,

- To predict whether an email is spam (1) or (0)
- Whether the tumor is malignant (1) or not (0)

odds= $p / (1-p)$ = probability of event occurrence / probability of not event occurrence

$\ln(\text{odds}) = \ln(p/(1-p))$

$$\text{logit}(p) = \ln(p/(1-p)) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \dots + b_kX_k$$

Types of Logistic Regression

1. Binary Logistic Regression

The categorical response has only two possible outcomes.

Example: Spam or Not

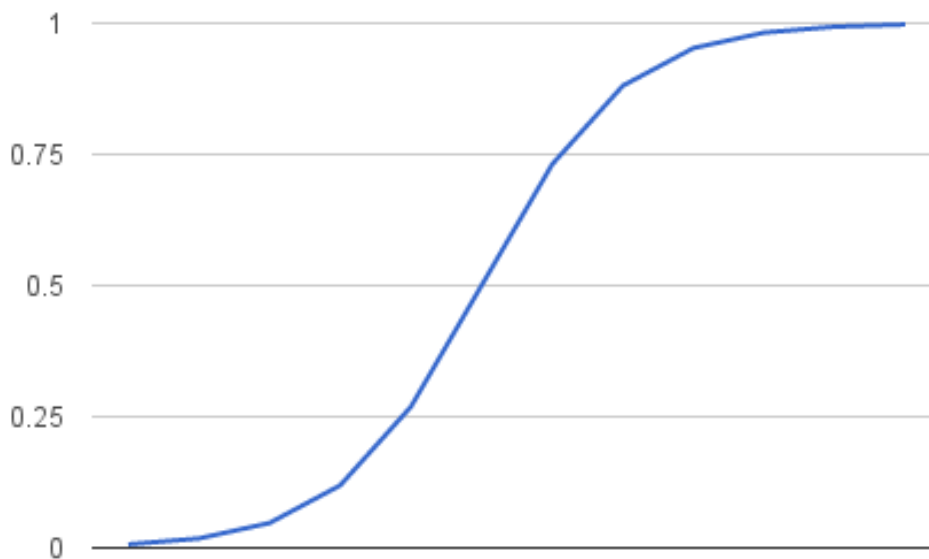
2. Multinomial Logistic Regression

Three or more categories without ordering. Example:

Predicting which food is preferred more (Veg, Non-Veg, Vegan)

3. Ordinal Logistic Regression

Three or more categories with ordering. Example: Movie rating from 1 to 5



We applied:

```
logreg = LogisticRegression().fit(X_train, y_train)
logit_pred = logreg.predict(X_test)
```

We fit Independent variable of training data with target value of same to logistic regression model.

And then predict target value for testing data.

Now we can compare actual target value of testing data with predicted target value and find accuracy.

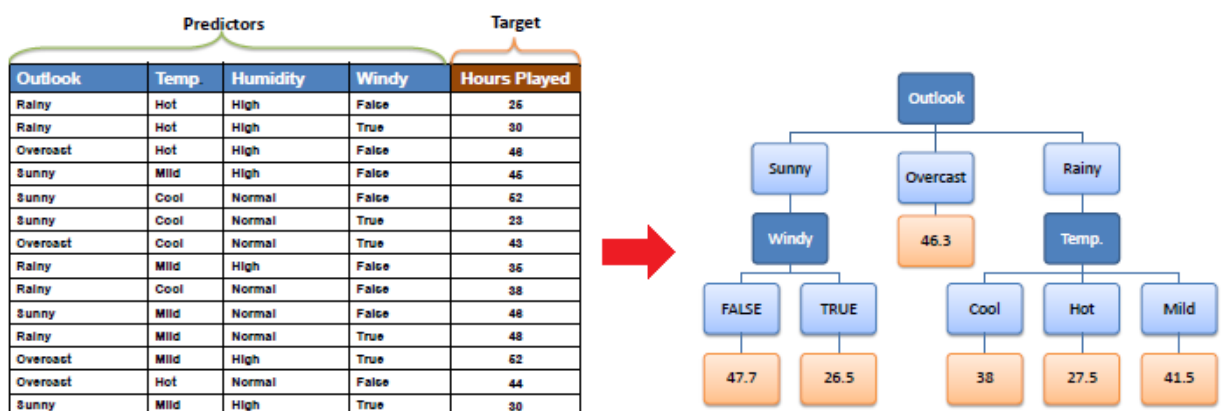
Accuracy: 91.40 % (approx)

4.2 Decision Tree Classification Model

Python Library Use: sklearn.tree

R Libraray Use: rpart, class

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.



Here we apply decision tree to numeric feature variable and identifying target value as last node which finally gives us output of 0 or 1.

4.3 Random Forest Classifier

Python library use: sklearn.ensemble. RandomForestClassifier

R library use: randomForest

It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Random forest is like bootstrapping algorithm with Decision tree (CART) model. Say, we have 1000 observations in the complete population with 10 variables. Random forest tries to build multiple CART models with different samples and different initial variables. For instance, it will take a random sample of 100 observations and 5 randomly chosen initial variables to build a CART model. It will repeat the process (say) 10 times and then make a final prediction on each observation. Final prediction is a function of each prediction. This final prediction can simply be the mean of each prediction.

Mexico has a population of 118 MM. Say, the algorithm Random forest picks up 10k observations with only one variable (for simplicity) to build each CART model. In total, we are looking at 5 CART models being built with different variables. In a real life problem, you will have more number of population sample and different combinations of input variables.

Salary bands :

Band 1 : Below \$40,000

Band 2: \$40,000 – 150,000

Band 3: More than \$150,000

Following are the outputs of the 5 different CART model.

CART 1 : Variable Age

	Salary Band	1	2	3
Age	Below 18	90%	10%	0%
	19-27	85%	14%	1%
	28-40	70%	23%	7%
	40-55	60%	35%	5%
	More than 55	70%	25%	5%

CART 2 : Variable Gender

	Salary Band	1	2	3
Gender	Male	70%	27%	3%
	Female	75%	24%	1%

CART 3 : Variable Education

	Salary Band	1	2	3
Education	<=High School	85%	10%	5%
	Diploma	80%	14%	6%
	Bachelors	77%	23%	0%
	Master	62%	35%	3%

CART 4 : Variable Residence

	Salary Band	1	2	3
Residence	Metro	70%	20%	10%
	Non-Metro	65%	20%	15%

CART 5 : Variable Industry

	Salary Band	1	2	3
Industry	Finance	65%	30%	5%
	Manufacturing	60%	35%	5%
	Others	75%	20%	5%

Using these 5 CART models, we need to come up with single set of probability to belong to each of the salary classes. For simplicity, we will just take a mean of probabilities in this case study. Other than simple mean, we also consider vote method to come up with the final prediction. To come up with the final prediction let's locate the following profile in each CART model :

1. Age : 35 years , 2. Gender : Male , 3. Highest Educational Qualification : Diploma holder, 4. Industry : Manufacturing, 5. Residence : Metro

For each of these CART model, following is the distribution across salary bands :

CART	Band	1	2	3
Age	28-40	70%	23%	7%
Gender	Male	70%	27%	3%
Education	Diploma	80%	14%	6%
Industry	Manufacturing	60%	35%	5%
Residence	Metro	70%	20%	10%
Final probability		70%	24%	6%

The final probability is simply the average of the probability in the same salary bands in different CART models. As you can see from this analysis, that there is 70% chance of this individual falling in class 1 (less than \$40,000) and around 24% chance of the individual falling in class 2.

4.4 Naive bayes Classification Model

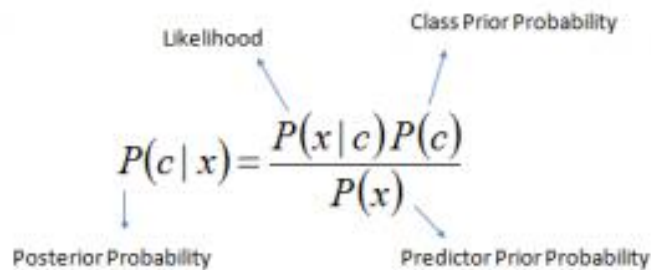
Python librarr use: sklearn.naive_bayes.GaussianNB

R librarr use: e1071

It is a classification technique based on Bayes' theorem with an assumption of independence between predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier would consider all of these properties to independently contribute to the probability that this fruit is an apple.

Naive Bayesian model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability $P(c|x)$ from $P(c)$, $P(x)$ and $P(x|c)$. Look at the equation below:



The diagram shows the equation $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$ with arrows pointing from labels to the corresponding parts of the equation. 'Likelihood' points to $P(x|c)$, 'Class Prior Probability' points to $P(c)$, 'Posterior Probability' points to $P(c|x)$, and 'Predictor Prior Probability' points to $P(x)$.

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

Here,

- $P(c/x)$ is the posterior probability of *class (target)* given *predictor (attribute)*.
- $P(c)$ is the prior probability of *class*.
- $P(x/c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

Example: Let's understand it using an example. Below I have a training data set of weather and corresponding target variable 'Play'. Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

Step 1: Convert the data set to frequency table

Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
All	5	9
	=5/14	=9/14
	0.36	0.64

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Problem: Players will pay if weather is sunny, is this statement is correct?

We can solve it using above discussed method, so $P(\text{Yes} | \text{Sunny}) = P(\text{Sunny} | \text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$

Here we have $P(\text{Sunny} | \text{Yes}) = 3/9 = 0.33$, $P(\text{Sunny}) = 5/14 = 0.36$, $P(\text{Yes}) = 9/14 = 0.64$

Now, $P(\text{Yes} | \text{Sunny}) = 0.33 * 0.64 / 0.36 = 0.60$, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

4.5 Matrices to find accuracy

Python libraries use: metrics

R Libraries Use: DMwR, pROC

As our dataset saying about classification modeling so we are applying classification metrics to find accuracy or how well performed our model to find best match model to get predictable output on test data.

So here we are applying confusion matrix and AUC score to find accuracy of models.

Confusion Matrix:

It is a performance measurement for machine learning classification problem where output can be two or more classes. It

is a table with 4 different combinations of predicted and actual values.

Example confusion matrix for a binary classifier (though it can easily be extended to the case of more than two classes):

Let's now define the most basic terms

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

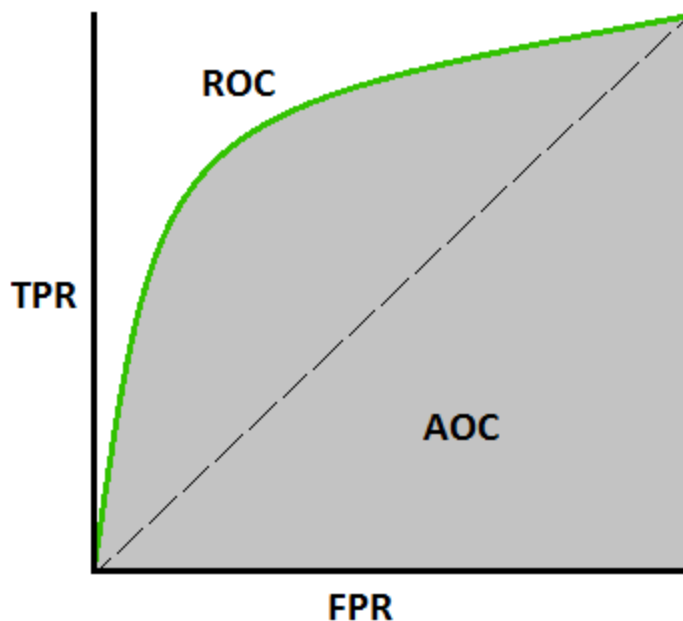
I've added these terms to the confusion matrix, and also added the row and column totals:

n=165		Predicted: NO	Predicted: YES	
Actual: NO		TN = 50	FP = 10	60
Actual: YES		FN = 5	TP = 100	105
		55	110	

This is a list of rates that are often computed from a confusion matrix for a binary classifier:

- **Accuracy:** Overall, how often is the classifier correct?
 $(TP+TN)/total = (100+50)/165 = 0.91$
- **Misclassification Rate:** Overall, how often is it wrong?
 - $(FP+FN)/total = (10+5)/165 = 0.09$
 - equivalent to 1 minus Accuracy
 - also known as "**Error Rate**"
- **True Positive Rate:** When it's actually yes, how often does it predict yes?
 - $TP/actual\ yes = 100/105 = 0.95$ also known as "**Sensitivity**" or "**Recall**"
- **False Positive Rate (Type-1 error) :** When it's actually no, how often does it predict yes?
 - $FP/actual\ no = 10/60 = 0.17$
- **False Negative Rate (Type-2 error) :** When it's actually Yes, and predicted No
 - $FN/ (actual\ yes) = 5/105 =$
- **True Negative Rate:** When it's actually no, how often does it predict no?
 - $TN/actual\ no = 50/60 = 0.83$
 - equivalent to 1 minus False Positive Rate
 - also known as "**Specificity**"
- **Precision:** When it predicts yes, how often is it correct?
 - $TP/predicted\ yes = 100/110 = 0.91$
- **Prevalence:** How often does the yes condition actually occur in our sample?
 - $actual\ yes/total = 105/165 = 0.64$

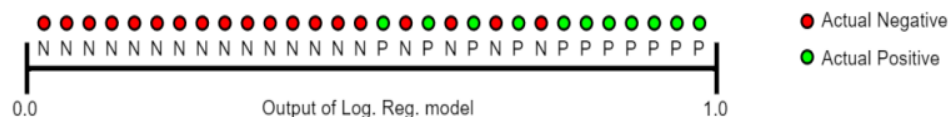
ROC Curve: This is a commonly used graph that summarizes the performance of a classifier over all possible thresholds. It is generated by plotting the True Positive Rate (y-axis) against the False Positive Rate (x-axis) as you vary the threshold for assigning observations to a given class.



AUC: Area Under the ROC Curve

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).

For example, given the following examples, which are arranged from left to right in ascending order of logistic regression predictions:



AUC represents the probability that a random positive (green) example is positioned to the right of a random negative (red) example.

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

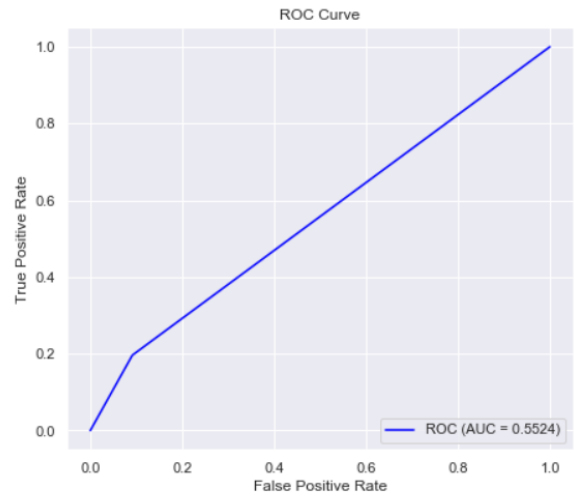
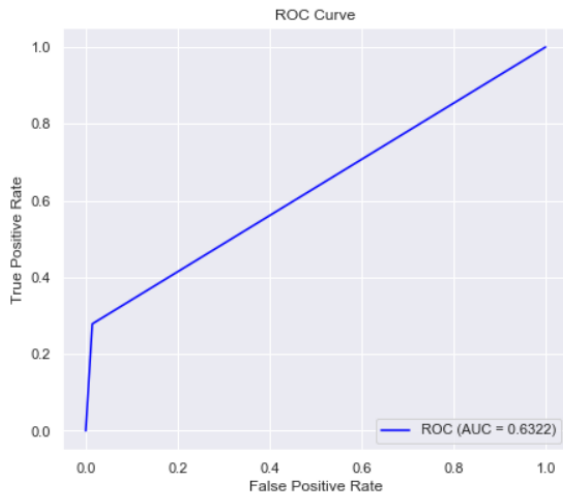
AUC is desirable for the following two reasons:

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

However, both these reasons come with caveats, which may limit the usefulness of AUC in certain use cases:

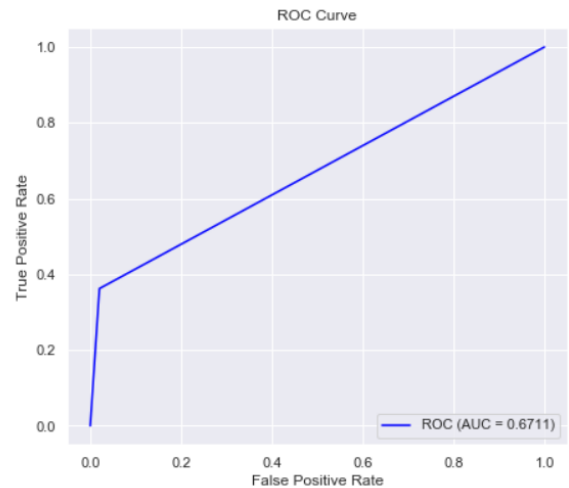
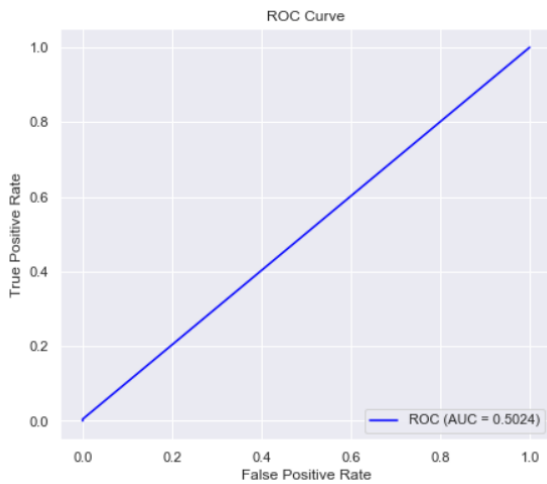
- **Scale invariance is not always desirable.** For example, sometimes we really do need well calibrated probability outputs, and AUC won't tell us about that.
- **Classification-threshold invariance is not always desirable.** In cases where there are wide disparities in the cost of false negatives vs. false positives, it may be critical to minimize one type of classification error. For example, when doing email spam detection, you likely want to prioritize minimizing false positives (even if that results in a significant increase of false negatives). AUC isn't a useful metric for this type of optimization.

For our train dataset ROC curv look like



Logistic Regression

Decision Tree Classification



Random Forest Classification

Naïve Bayes Classification

Matrix (aprox values considering R and Python Modeling)

	Accuracy	False Positive rate (Type I Error)	False Negative Rate (Type II Error)	True Negative Rate (Specificity)	True Positive Rate (Sensitivity /Recall)	AUC Score
Logistic Regression	91.50	1.40	72.14	98.5	27.86	0.63
Decisison Tree	85	9.14	80.3	90.8	19.62	0.56
Randon Forest	90	1.3	99.5	99.9	49	0.51
Naïve Bayes	92	2.0	63.77	97.99	36.2	0.68

4.6 Create predicted variable on test data

Here by observing Classification matrices accuracy we can say that ‘**Naïve Bayes**’ algorithm give us best match modeling to our dataset as it accuracy is **around 92%** which is good compare to other models and **AUC** is around **0.68** which is near to slightly go to 1.

So for Predicting target value for our dataset we will apply ‘Naïve bayes’ modeling.

CONCLUSION

While doing prediction for 'target' value for test dataset we observed that '0' mean 'No' and '1' means Yes. Therefore, after predicting Santander Customer transaction report for future we can say that more (around 95%) of customers can reject or never do a specific transaction in future.

REFERENCES

<https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html>

<https://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm>

<https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest-simplified/>

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

<https://www.kaggle.com/antmarakis/calculating-and-plotting-auc-score>