greatlearning

"Predicting Customer Profitability by Analysing Buying Behaviour of Customers for a Superstore Chain".

Submitted in Partial Fulfilment of requirements for the award of a certificate of

Post Graduate Program in Data Science

Capstone Project Report

Submitted to

greatlearning

Submitted by:

Jayshree Sahu

Pragya Apurva

Priyanka Koparkar

Swati Mishra

Under the guidance of

Deep Sachdev

Batch: **December - 2019**

Year of Completion: **2021**

CERTIFICATE OF COMPLETION SIGNED BY MENTOR

This is to certify that the participants **Jayshree Sahu, Pragya Apurva, Priyanka Koparkar, Swati Mishra**, who are the students of Great Learning, have completed their project on ***"Predicting Customer Profitability by Analysing Buying Behaviour of Customers for a Superstore Chain."***

This project is the record of authentic work carried out by them during the academic year **2021.**

Deep Sachdev

Mentor

Date: 24/03/2021

Place: Pune

ACKNOWLEDGEMENTS

First of all, we would like to express our thanks to each other as a great team where each member contributed so efficiently and made this project a success.

We are grateful to all the Great Learning faculties who have contributed hugely to our learning throughout the curriculum. We thank Great Learning as an institute to provide us this platform and helped us with all the resources and staff.

Also, our special thanks to Mr. Deep Sachdev, our mentor during the project whose support made it easier to achieve our tasks.

Last, but not least, we thank our Program Coordinator Mr. Shomopom Mukherjee, who has been actively involved and supportive at each step during this journey with Great Learning.

ABSTRACT

This project involves data analytics tools to help determine profitability for a retail store based on the type of customers. Customer type, here, means a group of customers showing similar buying behavior. Each customer type has a different impact on profitability which helps the marketing department decide specific schemes for a given customer type and hence, increase profitability.

Customer type or buying behavior can be determined by certain parameters (independent variables) on which profitability (target variable) is dependent.

The exercise is to help the marketing department of the Company in modifying its marketing strategies according to the customer type.

We would be using different Data Analytics techniques to achieve our results which is a powerful tool to learn and understand information hidden in any given data and use it for predicting the future trends for taking strategic business decisions.

***Keywords**: Predictive Analysis, Retail Sales, Profitability, customer behavior analysis*

***Techniques and Tools:** Python, Statistical tools and models, Excel*

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Table 1(a)

GLOSSARY OF TERMS / ABBREVIATIONS

Table 1(b)

SNo	Abbreviation	Explanation
1	AD-RSQ	Adjusted R-Squared
2	CMO	Chief Marketing Officer
3	LR	Linear Regression
4	MAE	Mean Absolute Error
5	ML	Machine Learning
6	MSE	Mean Square Error
7	RMSE	Root Mean Square Error
8	RSQ	R-Squared
9	Std dev	Standard deviation
10	VIF	Variation Inflation Factor
11	X / x	Predictors / Independent variables
12	Y / y	Target / Dependent variable
13	y-hat	y predicted

EXECUTIVE SUMMARY

Problem Statement

The dataset belongs to a Superstore chain that is growing at a rapid pace. The CMO of this company understands the rising competition which may impact the profitability of the store and thus wishes to use the existing data to understand the buying behavior of its customer. To understand the buying behavior of the store's regular customers, he wants to analyze the profitability of each customer who has ever visited the store. This study will help the company to modify the marketing strategies for the customers.

Data Description

In the data there are 20 variables out of which Profitability is the Dependent Variable, CustomerID variable is insignificance in data analysis, Rest all 18 variables are Independent Variables that we used in model analysis.

Main Results

On comparing the above metrics we have the following observations:

- When comparing linear models and tree-based models, *linear models have better performance* than tree-based models.
- Out of the above three, we find Lasso is the most robust model and easy to relate with our business problem.
- While in Random Forest, R-squared values for train and test data are 99% and 93%, for Lasso the model performance is only 88% and 87% for train and test data respectively *leading to over fitting in Random Forest*
- Adjusted R-squared values are also higher for Random Forest train and test data at 0.99 and 0.93 respectively and lesser for Lasso train and test at 0.88 and .87 respectively suggesting overfitting again.
- While comparing error terms like MAE, MSE, and RMSE values, we notice the difference between train and test errors in Random Forest is very large; however, the difference in test and train errors for Lasso is very minimal. This suggests the Lasso regression model is much more robust than the Random Forest model.

Table 2: Model comparison between Random Forest and Lasso Regression

Random Forest	Train	Test	Lasso Regression	Train	Test
MAE	14.67	44.89	**MAE**	61.13	61.99
MSE	482.09	3579.90	**MSE**	6509.64	6832.57
RMSE	21.96	59.83	**RMSE**	80.68	82.66
RSQ	0.99	0.93	**RSQ**	0.88	0.87
AD-RSQ	0.99	0.93	**AD-RSQ**	0.88	0.87

- As we understand the objective of making our model is to able to explain the features that are important to business and profitability, we need a model that is better at explaining the contribution of each feature and the way each feature affect the profitability and this can be better done for linear models. Hence, comparing the feature importance and evaluation metrics for each of the models, the best explainable and most robust model is Lasso Regression.

Recommendations & Conclusions

1. ***AvgNumberOfProduct*** comes out to be the most important feature which means an increase in the number of products sold enhances profitability and it is reasonable to understand that selling more products means adding more to margins and hence, profitability increase for the business.
2. ***RFM_label*** also shows a high positive relationship with profitability. RFM_label which is a derived variable denotes the customers in three slabs i.e., High, medium, and low denoting the customer's loyalty. The focus for the client should be to help customers climb the loyalty ladder, from low to medium and from medium to high at the same time retaining customers at high.

3. ***AvgSaving*** is another very strong driver of profitability according to the model. It makes sense as we understand that customers attract to deals where they can save money. Also, saving money in a way encourages customers to buy more too.
4. ***VisitInlastMonth*** is another significant feature that is contributing positively to an increase in profitability. The point is quite logical as the more the number of visiting customers, the more will be the sales and hence, profitability.
5. ***Tenure*** is another important feature in the model which affects profitability positively. It is very logical to understand that maintaining old customers is less costly than bringing in new customers, hence, the ability to retain customers adds to profitability.
6. ***SatisfactionScore:*** also comes out to be an important variable for improving profitability. It is a known rule that a satisfied customer means a potential repeating customer meaning more sales and profitability.
7. In continuation to above, we have observed there are few features like ***MembershipCard*** and ***ResonseToPromotion (derived from coupons, free delivery, footfall in discount period)*** exist in the dataset which should, intuitionally, have shown positive linear relation with profitability, but *neither shows any significant effect on profitability directly nor appears in the list of the important features*. Here, we may advise that the Marketing department should relook into the guiding parameters for these promotional strategies to make them effective.

Section1: Introduction

1.1 Need of the Study

As we all know profitability determines the growth and future for a business and every business goal towards increasing profitability. There are many driving factors for increasing profitability like increasing revenue, deploying marketing schemes, enhanced customer satisfaction, etc. In this study, we will study these same aspects of the given retail sales dataset and try to correlate it with their business challenges and help them increase their profitability.

The first step to healthy business practice is a satisfied customer. With rising competition in retail sales and the varied backgrounds of customers, it is difficult to keep up with all customers' expectations. However, if we can determine what attracts a particular customer to the store, we may be able to devise customized marketing plans according to the customers' preferences and choices.

This project helps to understand the buying behavior of the customers in the retail superstore and its impact on the profitability of the store.

Given certain parameters, predicting the profitability will help us determine which customer type is more profitable than others. Also, if we understand, what attracts the customers to buy, the retail chain can modify its marketing strategies according to the buying behavior of the customer.

1.2 Objectives

From the problem statement, the business objective of this exercise is to understand the impact of customers' buying behavior on profitability and in turn, use those features in predicting profitability in future

Hence, to determine profitability, things that need to be analyzed are:

1. Do customers respond differently to different sales benefits (i.e. marketing strategies)?
2. What are the different buying patterns/behaviors shown by the customers?
3. Whether customer attributes like Tenure, satisfaction score, complaints, membership type, etc. impact profitability?
4. Whether demographics age, gender, city tier, etc. affects the profitability?
5. How the different buying behavior affect profitability?
6. Build a model that predicts profitability based on different behaviors?

1.3 Data Sources

Source of data – Great Learning as part of Capstone project

1.4 Statistical tools & techniques used.

Our analysis is done majorly by using the following Python and some part using excel by using the following **tools**:

Table 3 : Tools Used

S No	Library / Module	Module / Function	Utility
1	Numpy		for numerical calc
2	Pandas		for data manipulation
3	matplotlib.pyplot		for data visualization
4	Seaborne		for data visualization
5	scipy.stats	zscore	for standardizing the data
6	sklearn.decomposition	PCA	for feature reduction
7	statsmodels.stats.outliers_influence	VIF	for feature selection
8	sklearn.feature_selection	SelectKBest	for feature selection
9	sklearn.ensemble	ExtraTreesRegressor	for feature selection
10	sklearn.metrics	mean_squared_error	for model evaluation
11	sklearn.metrics	mean_absolute_error	for model evaluation
12	sklearn.metrics	r2_score	for model evaluation

13	Math	sqrt	for square root function
14	sklearn.model_selection	train_test_split	for splitting data
15	statsmodels.api		for model execution
16	sklearn.ensemble	RandomForestRegressor	for model execution
17	sklearn.linear_model	LinearRegression	for model execution
18	sklearn.linear_model	Lasso	for model execution
19	sklearn.model_selection	cross_val_score	for cross validation
20	sklearn.model_selection	RepeatedKFold	for hyper tuning
21	sklearn.impute	KNNImputer	for model execution

We have also used a few **statistical techniques** at various stages of pre-processing and feature engineering as explained below:

Table 4: Techniques Used

S No	Technique	Utility
1	Binning	Pre-processing of tenure, last month coupon used, visit in last month
2	logarithmic transformation	Treated outliers in AvgNumberOfProduct, SavingPercent, Tenure
3	Capping	Treated outliers in AvgBillAmount, AvgSaving
4	RFM (Recency, Frequency, Monetary)	Created new variable from data using "AvgNumberOfProduct" for Recency, "VisitInLastMonth" for frequency, "AvgBillAmount" for Monetary
5	New feature by combining existing features	Created new variable ResponseToPromotion using TakenFreeDelivery, FootfallInDiscountPeriod & TakenProdFromPromo

Limitations

1. Timelines for variables are at a different length. Some variables correspond to monthly data e.g., complaints, the coupon used, etc., some to yearly like average saving, and some for one day like HigestSavings. Also, the overall period is not clearly defined to understand the real impact of the feature on target e.g., tenure is given for a customer some years, but it does not clarify if the customer is currently a buyer or left long before.
2. The model works well on the given set of train and test data. In the future, if the distribution of the data changes, the model may not be equally effective and may need retraining the model.

Section 2: Literature Review

1. In this paper, the author tries to use machine learning algorithms like XG boost, Light GBM, and Cat Boost to predict the profitability of each customer against of credit provided. They have used historical data of first-time purchases to understand profitability in the future that is 180 days of first-time purchase. Precision and recalls have been used for computing model performance. All the models performed very closer to each other but XGboost and Light GBM were not able to identify the unprofitable customers. They have also used baseline models that are Logistical regression and Random forest classifier without hyperparameter tuning for comparison to find the difference in prediction. Their data were biased towards profitable customers. Cat Boost and Random Forest Classifier were better than others, but other model's results were not too low. They also recommended that hyperparameter tuning will help in yielding better results.[1]

2. In this paper, the author tries to study consumer behaviors which is a major fact affecting the market nowadays. The behavior of the customer is impacting the industries and make them study their pre-buying, purchasing, and post-purchasing behavior. It helps in making marketing plans. [2]
3. In the paper, the author emphasized retail shops and also put the focus on retailer's ability to understand trading activities. They discussed the focus area which will help retailers in gaining major market share, which is the loyalty of customers, good products offered, customer connectivity, relationship with vendors, services offered, location, and uniqueness. The author also discussed the parameters like Response to Sales Promotion Method, period and regularity of purchase, and method of purchase which affects customers buying behavior.[3]
4. In the paper, the author focuses on the term CLV, which is Customer Lifetime Value, and states that it helps in increasing profitability. CLV helps in profitability when a customer buys different types of products, stays for a long term, and buys frequently. The author used time series analysis on retailer database and have computed loyalty based on duration, buying frequency, and RFM and tried finding the coefficient correlation between observed future profitability and loyalty. They have also used three approaches of model building, which are simple generalized gamma distribution model, continuous mixture model, and latent class mixture model. Their model showed a correlation of only 0.3 between future profitability, purchase frequency, and RFM even with duration it was around 0.37. The author suggested that traditional loyalty metrics are not useful in profitability and gross contribution may have good relationship with the frequency model.[4]
5. The author suggested that traditional retail shops need to be evolved and these are in the process to increase profitability. They stated that using footfall as a parameter will not show the result on profitability. Thus, a retailer can collect more and more information regarding customer and their buying behavior is important to diversify the portfolio in the right source. The cost of space required for retail shops and other operational costs have increased and there is also a need to invest in digital transformation and reducing the price due to competition, all leads to low profitability and more liabilities. Focus on customer, logistics, distribution and, get specialized in one or two areas. Bring automation, robotics, and artificial intelligence, it will reduce cost and enable investment in areas of business.[5]
6. Here, the author focuses on challenges related to Customer account profitability. It can be on an individual, group, and distribution channel level. They also suggest that checking customer profitability database helps in tracking the resources which can attract new customer and retain the existing one. It will also help in estimating operating income from new and existing customers. Suggested competitive and cost and better services may increase retention of customers.[6]
7. The author used two random forest techniques to predict the future profitability of a customer, their future purchases, partial defection, and their retention ratio. They have extended random forest for regression and with the classifier, they tried to get retention ratio. Here, the random forest technique outperformed simple linear and logistic regression. AUC, Sensitivity - True positives/ Total, Specificity - True negatives/ Total non-events, ROC curve and Mean absolute deviation are the criteria for evaluating the models.[7]
8. Here, they suggest that calculating profitability will be very useful for making strategies, but customer’s profitability can change over time like a low profitable customer may become highly profitable in the future and vice versa. The author used a linear regression model for prediction. Selected parameters for model building are a form of past marketing strategies for customers, customer purchase history, and customer heterogeneity whereas the target variable is the log of gross profit.[8]
9. The author used various feature selection models for regression problems and suggested Lasso as a better model than traditional forward, backward, or stepwise selection.[9]
10. Valeria Fonti claims to conclude the same that Lasso performs better for feature selection in linear regression problems and provides relevant features.[10]
11. The author suggests that in the case of multicollinearity Lasso performs better than simple ordinary least squares. Lasso, Ridge, and Elastic Nets are useful for a multicollinearity regression problem. Ridge does

not make the coefficient as 0 but the results are not good for interpretation. Whereas Elastic Nets cause double shrinkage. Lasso tends to make one out of all the highly correlated variables as 0 but in our case, it does not affect like that.[11]

Section 3: Data Pre-processing

Following checks made on the dataset and the observations at each step are noted below:

3.1 Data Information

Provided data has the following datatype:
float64 – 9 int64 – 8 object – 3

3.2 Missing Values

There are 9 variables with missing values that were treated:

Table 5: Missing values treatment summary

Variable Name	No. of missing values	Preferred Treatment	Comparison with other methods
Tenure	301	Knn	• Mean: It leads to change in distribution although the mean remains the same • Median: Changes mean and distribution both • KNN: Both mean and distribution are close to original data • LR: Considering the business perspective, predicting the 'Age' of the Customer based on other variables is not logical.
HighestSaving	241	Linear Regression	• Mean & Median: Distribution changed as per distplot from the original • Linear Regression: As per distplot there is not much difference from the original. Columns having high correlation, taken for computation as it leads to a good balance of error and adjusted R-squared
PreferredPaymentMode	6	Mode	• Mode: Categorical variable and having less than 2% of the total records as the missing value
Age	302	Knn	• Mean & Median: It leads to change in distribution although the mean remains the same • KNN: Distribution is close to original data and Mean is also not deviating much (decrease in the mean by 0.07) • LR: Considering the business perspective, predicting the 'Age' of the Customer based on other variables is not logical.
MembershipCard	5	Mode	• Mode: Categorical variable and having less than 2% of the total records as the missing value
AvgSaving	254	Imputed from 'SavingPercent'	• Using formula: AvgSaving = AvgBillAmount * SavingPercent
VisitInLastMonth	52	Knn	• Mean: It leads to change in distribution although the mean remains the same • Median: Mean and distribution both changed. • KNN: Distribution and Mean is close to original data
AvgNumberOfProduct	216	Median	• Mean: Not possible as there are too many (258) outliers in the data • Median: The distribution, skewness and, mean value remains very close to the original data • KNN: Distribution is similar, skewness is reduced but mean and median change more as compared to the median

AvgBillAmount	278	Imputed from 'SavingPercent'	• Using by formula: AvgBillAmount = AvgSaving * SavingPercent

3.3 Check for object variables and remove disparities

For three variables, it is confirmed that the same value had been entered with two names. The two different names are typo errors and have the same meaning, hence we replaced them with one single value., which is corrected as below:

1. values *'COD' & 'CashOnDelivery'* in 'PreferredPaymentMode' column mean the same, replaced as 'COD'
2. values *'M' & 'Male*' in the 'Gender' column have the same meaning replaced as 'Male'
3. values *'G' and 'Gold'* in the 'MembershipCard' column mean the same and replaced with 'Gold'

3.4 Duplicated field

There are no duplicated entries in the dataset

3.5 Adding New Variable

1. Response to Promotion

✓ We added a new variable 'Response to Promotion' using three binary variables: TakenFreeDelivery, FootfallInDiscountPeriod' and 'TakenProdFromPromo'

✓ The variable is divided into three categories as per response behavior:

a. **High**: where all three variables are '1' i.e., 'yes'
b. **Low:** where all three variables are '0' i.e., 'no'
c. **Medium:** rest all others i.e., where both '0's and '1' are present

2. RFM_label

✓ We added a new variable RFM_label using three variables – AvgNumberOfProduct as Recency, VisitInLastMonth as frequency and AvgBillAmount as monetary.

✓ We used the qcut function to divide the customers into 3 equal-sized bins and then ranked them accor dingly. This is done for each of the 3 individual scores.

✓ We then concatenated the individual scores to form RFM score for all the customers and categorized them into 3 groups:

a. **Most_Imp** (which is given the label as 1 in label encoding): where all the three R, F, and M are 1 or two of the three are 1.
b. **Med_Imp** (which is given the label as 2 in label encoding): the once not categorized as most_i mp or least_imp.
c. **Least_Imp** (which is given the label as 3 in label encoding): where all the three R, F, and M are 3 or at least two are 3.

3.6 Removing Unwanted Variable

We can see that **CustomerID** has no role to play in the analytical study as it is unique to each column. Hence, we may drop this at this stage.

3.7 Variables with Outliers

1. Outlier's treatment summary

There are 8 variables with outliers. Below table shows the methods used for outlier's treatment:

Table 6: Various treatments used for outlier removal

Column Name	Outliers in original data	Using Binning on Original Data	Using Capping on original data	Using Capping at Upper Bound on Scaled Data	Using Logarithmic transformation
AvgBillAmount	14	-	Capping at the upper bound value	3	NA
AvgNumberOfProduct	258	-	Capping at the upper bound value	81	0 at upper bound, 68 at lower bound
AvgSaving	67	-	Capping at the upper bound value	14	NA

HighestSaving	2	-	No changes as the outer bound are 885.61 and the max value is 909.8, so not much deviation	2	NA
LastMonthCouponUsed	581	Binned in 3 Categories	-	110	NA
SavingPercent	52	-	No changes as the outer bound are 4.825 and the max value is5, so not much deviation	52	0
Tenure	445	Binned in 4 Categories	-	117	101 at upper bound
VisitInLastMonth	609	Binned in 3 Categories	-	131	NA

Outlier treatment is a very important part of Pre-processing of Data, as outlier can impact models and provide wrong input. The number of outliers is large so deleting them will result in heavy data loss. Only a few models are robust from outliers, so to open for different models we treated outliers.

3. Outliers' treatment methods explained.

a. Binning:

Binning is the process of categorizing continuous numerical data. Binning helps to create buckets due to which the values lying above or below the upper and lower bound respectively, will be categorized in one group and will prevent outlier effects on modeling. There are two ways to create buckets, qcut and cut. qcut categorizes based on quantile and tries to allot observations to each bin. Whereas cut categorizes based on the range assigned to it [13]. Here, we have used qcut and binned 3 variables:

- ✓ Tenure: In 4 categories
- ✓ LastMonthCouponUsed: 3
- ✓ categoriesVisitInLastMonth: 3 categories

b. Log Transformation:

- ✓ Applied on 3 variables, AverageNumberOfProduct, SavingPercent, Tenure.
- ✓ After transformation, log data of **AverageNumberOfProduct** has no upper outliers but it introduces 68 lower outliers instead. **Tenure** shows a reduction in outliers from 445 to 101 on log transformation**. Log of SavingsPercent** removes outliers completely.
- ✓ We have kept the logged and original columns in the data at present.

c. Capping on original data at a different level for 3 variables:

In this method, we are capping the variable at the upper bound to limit the extreme values. Here, we have used this method on 3 variables:

- ✓ **AvgNumberOfproduct**: As per the formula of calculating outliers using IQR, the upper bound is 38.45 and till the 99th percentile it is increasing gradually but after the 99th percentile there is a sudden increase in value by 35+. But due to skewness of variable, we capped at Upper Bound.
- ✓ **AvgBillAmount**: Capped at the upper bound. 99th percentile is 19220.74074 whereas the tolerable value for outlier as per IQR formula is 20515.81662.
- ✓ **AvgSaving**: Due to skewness of variable, capped at the upper bound. The upper bound is 386.164 and the 99th percentile is 394.02.

d. Capping on scaled data at the Upper Bound:

Here scaling is applied on all columns. As scaling brings data at one level and the mean becomes 0(here we have used z-score, i.e., (x-mean/std dev) method to converge the columns so that mean=0 and std dev=1, for treating the outliers, we have capped the outlier points at upper bound for all variables. Here we see that after applying 'scaling', the number of outliers decreases drastically, that is from 2028 to 501 (More than 1/4th), so we can say data is not of poor quality. [12]

Section 4: Exploratory Data Analysis (EDA)

4.1 Data Description of numerical features

Table 7	count	mean	Std	min	0.25	0.5	0.75	max
CustomerID	4530	9E+08	1307.84269	9E+08	9E+08	9E+08	9E+08	9E+08
Profitability	4530	425.6877	231.372916	48.3027	224.8461	437.454	621.487	911.003
Tenure	4530	2.131443	2.24981	0.500606	1	1.197094	2.44025	24.4329
HighestSaving	4530	318.1045	190.05106	24.1653	158.3536	301.2459	449.255	909.959
Age	4530	48.97925	17.66381	18	35	49	64	80
SavingPercent	4530	1.831651	1.033959	0.4	0.95	1.67	2.5	5
AvgSaving	4530	127.0778	97.323801	4.86	48.72	100.74	183.698	394.02
LastMonthCouponUsed	4530	1.824503	2.008014	0	1	1	2	16
VisitInLastMonth	4530	1.955188	2.680198	0	1	1	2	25
AvgNumberOfProduct	4530	14.55483	11.789478	0.5	6.2	11.4	19.1	57.8
AvgBillAmount	4530	7434.611	4613.78752	486.705	3703.136	6864.291	10428.2	20515.8
Tenure_S	4530	-0.03399	0.835769	-0.72496	-0.50296	-0.41535	0.13727	3
HighestSaving_S	4530	-5.1E-05	0.999956	-1.5468	-0.84066	-0.08872	0.69015	3
SavingPercent_S	4530	-0.00074	0.997858	-1.38478	-0.85279	-0.15636	0.64647	3
AvgSaving_S	4530	-0.00042	0.998778	-1.25037	-0.8024	-0.27109	0.5762	3
LastMonthCouponUsed_S	4530	-0.02724	0.882025	-0.90871	-0.41065	-0.41065	0.08741	3
VisitInLastMonth_S	4530	-0.04124	0.802722	-0.72957	-0.35643	-0.35643	0.01672	3
AvgNumberOfProduct_S	4530	-0.01588	0.936975	-1.15418	-0.68973	-0.26603	0.36138	3
AvgBillAmount_S	4530	-9E-05	0.999832	-1.50451	-0.80823	-0.12392	0.64759	3
Tenure_Bin	4530	2.168433	0.878145	1	2	2	3	4
LastMonthCouponUsed_Bin	4530	1.488521	0.612287	1	1	1	2	3
VisitInLastMonth_Bin	4530	1.447241	0.628303	1	1	1	2	3
AvgNumberOfProduct_log	4530	2.33781	0.898401	-0.69315	1.824549	2.433613	2.94969	4.55282
SavingPercent_log	4530	0.42626	0.630525	-0.91629	-0.05129	0.512824	0.91629	1.60944
Tenure_log	4530	0.437819	0.726258	-0.69194	0	0.179897	0.8921	3.19593

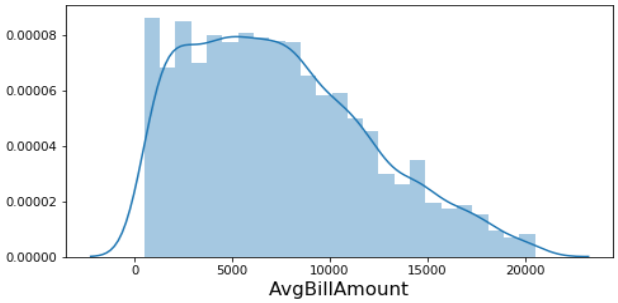
4.2 Data Description of categorical features

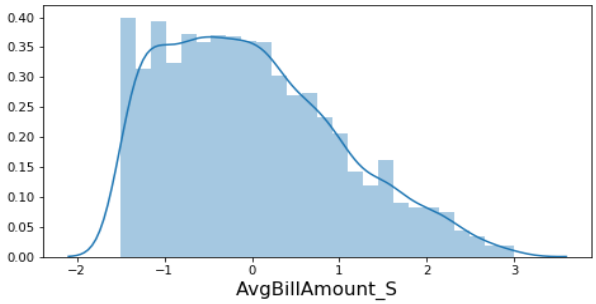
Table 8	Count	Unique	Top	Freq
PreferredPaymentMode	4530	5	Debit Card	1890
Gender	4530	2	Male	2702
MembershipCard	4530	3	Gold	2311
ResponseToPromotions	4530	3	Medium	3444
RFM_label	4530	3	Least_Imp	2195
SatisfactionScore	4530	5	3	1384
CityTier	4530	3	1	2970
TakenFreeDelivery	4530	2	1	3226
FootfallInDiscountPeriod	4530	2	1	3116
TakenProdFromPromo	4530	2	0	2862
Complain	4530	2	0	3242

4.3 Data Visualization for checking distribution, outliers, and impact of Continuous variables

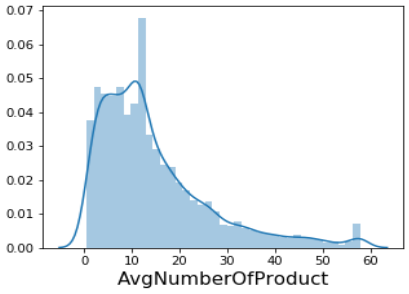
Checking distributions

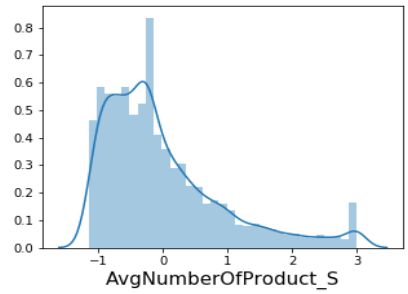
Distplot 1- The distribution of the **AvgBillAmount** variable is not changed much after the treatment.

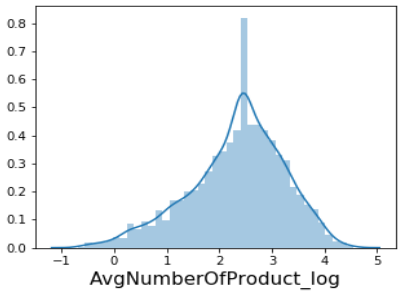




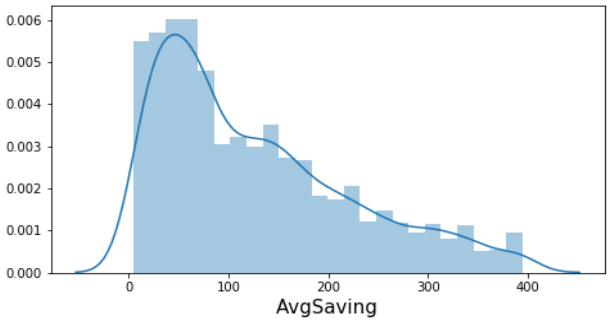
Distplot 2- The variable, **AvgNumberOfProduct** has shown a better distribution after logarithmic transformation, it has become more normal.

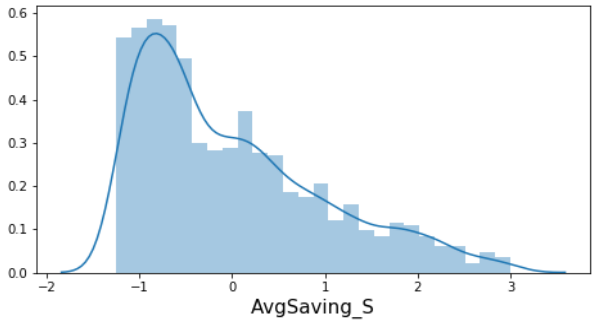






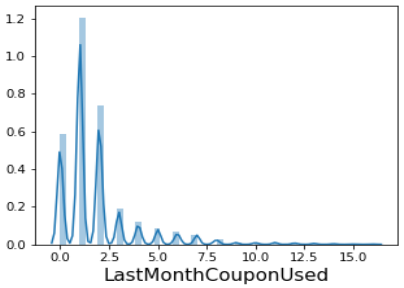
Distplot 3- The distribution for **AvgSaving** has not changed much after the treatment.

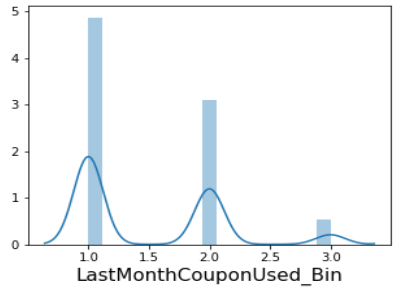


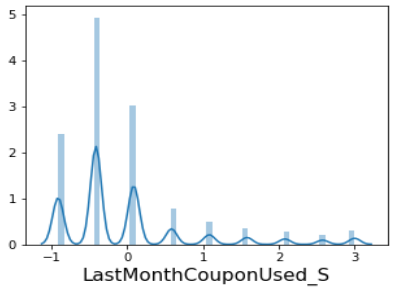


4.2 Data Distribution through distplot

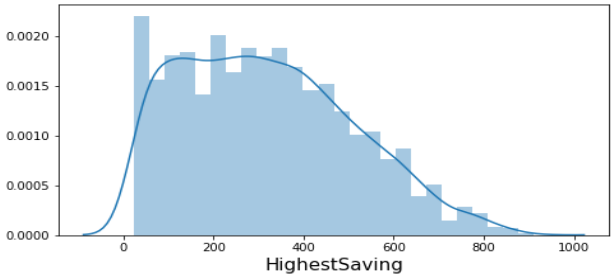
Distplot 4- For **LastMonthCouponUsed**, with binning the variables has become more interpretable since it has less no. of modes.

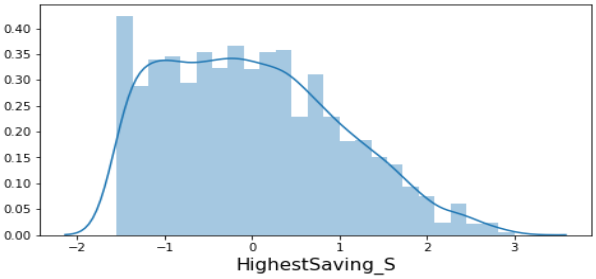




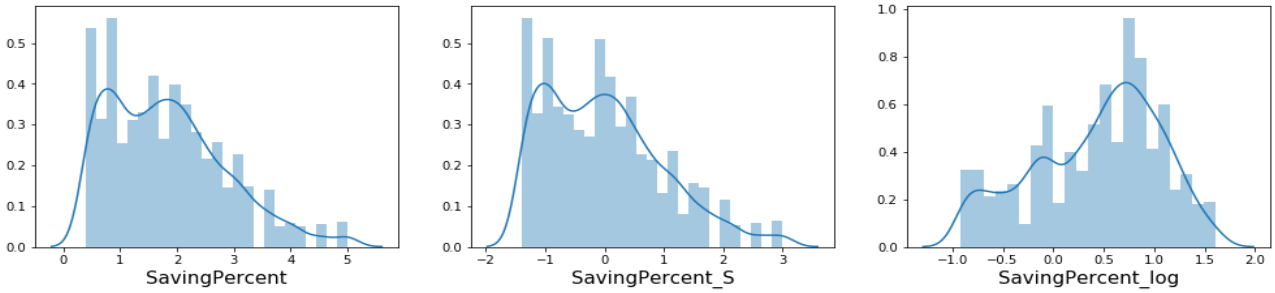


Distplot 5- The distribution of **HighestSaving** is not changed much after the treatment.

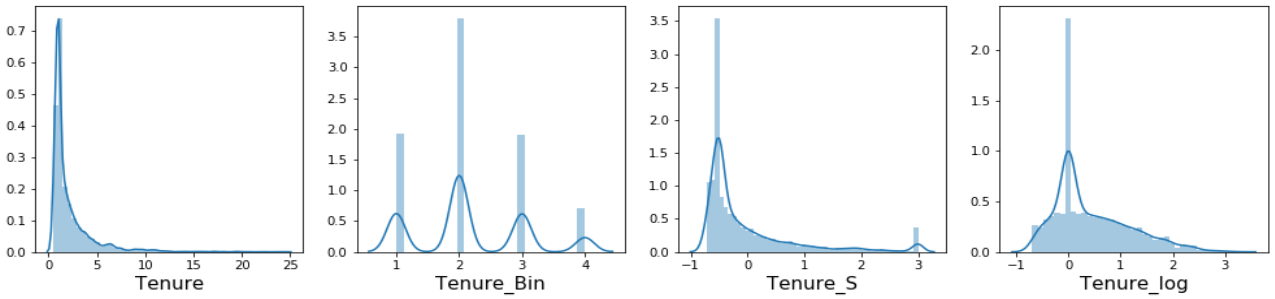




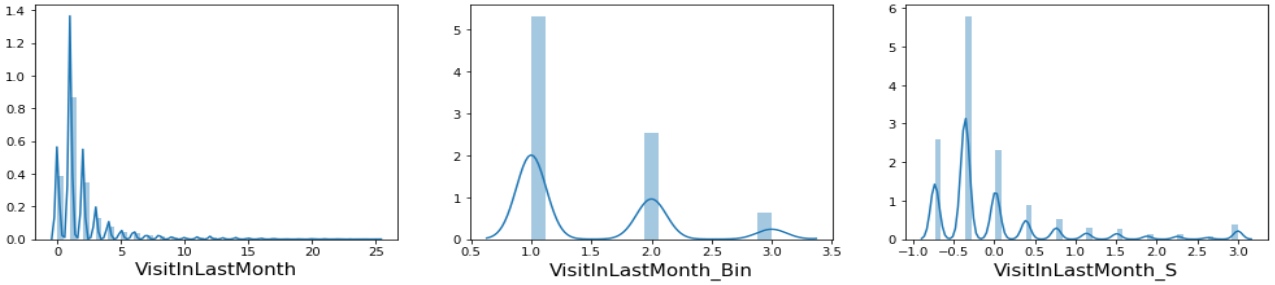
Distplot 6- For **SavingPercent**, data has become less skewed with logarithmic transformation.



Distplot 7- The distribution of the **Tenure** variable has become normal from skewed with logarithmic transformation.



Distplot 8- For **VisitInLastMonth**, it is more interpretable with binning as it has reduced the number of modes.



Observations:

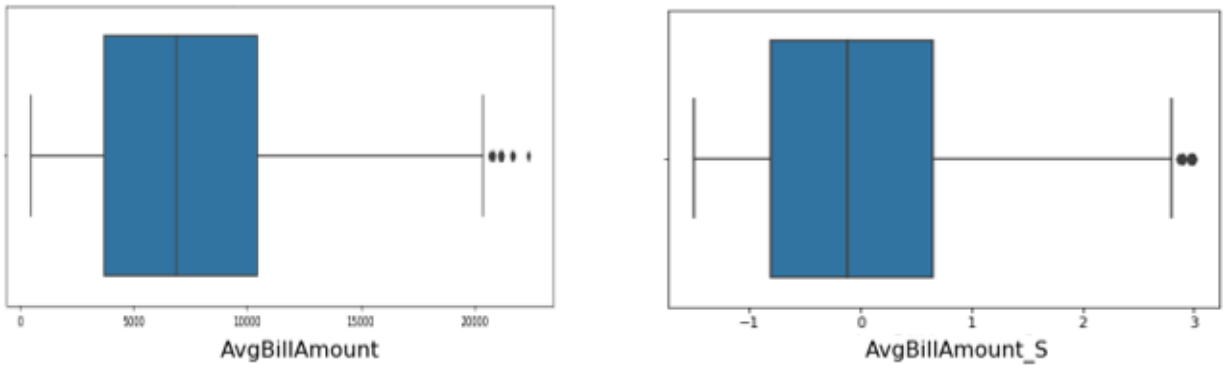
It may be observed that for most of the variables, with treatments the skewness of the data has reduced.

- ✓ The logarithmic transformation has helped us to observe normality in 3 of the variables.
- ✓ With Binning, we could achieve fewer categories of variables thus simplifying the observations.

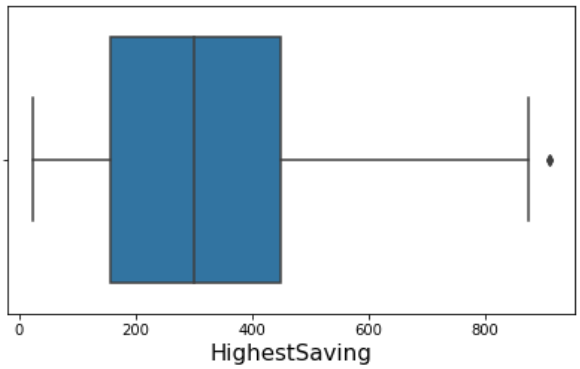
4.3 Outliers visualization through Box-plot

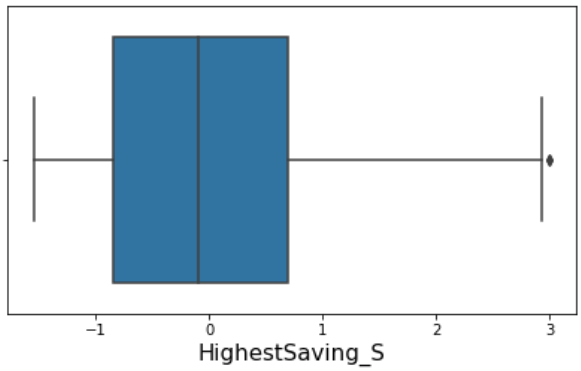
Compare Box-plots – for the new and original variable.

Boxplot 1 – For **AvgBillAmount**, outliers are reduced in scaling the data:

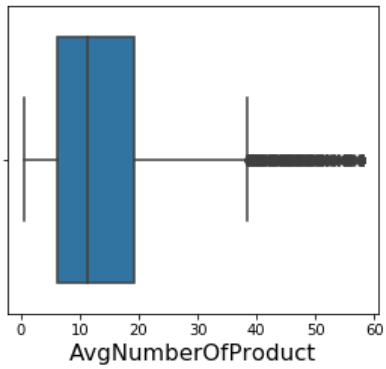


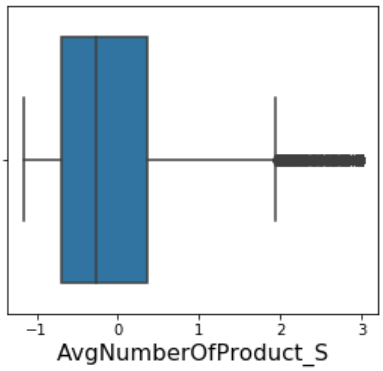
Boxplot 2 – For **HighestSaving**, outliers have moved closer to the whisker:

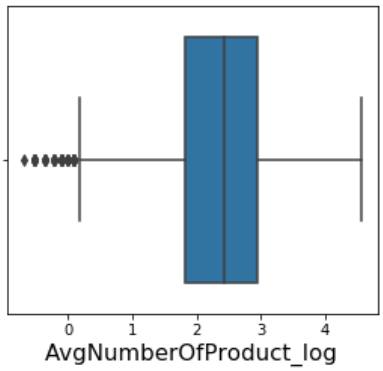




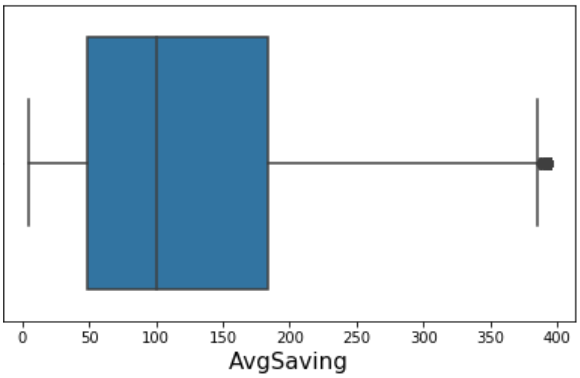
Boxplot 3 – For **AvgNumberOfProduct**, outliers are reduced by scaling and reversed by log transformation:

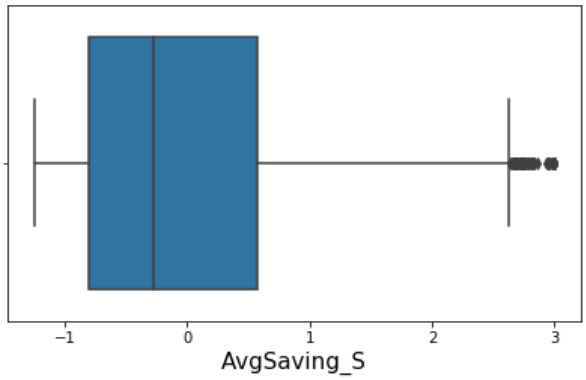




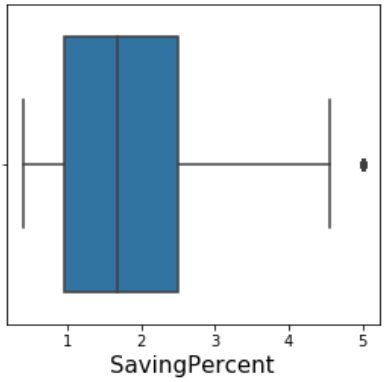


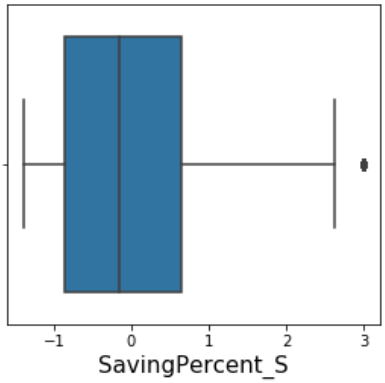
Boxplot 4 – For **AvgSaving**, outliers are reduced and come closer to the upper bound:

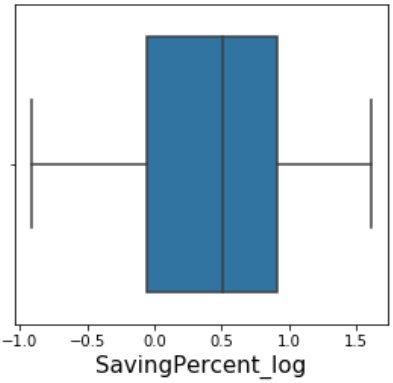




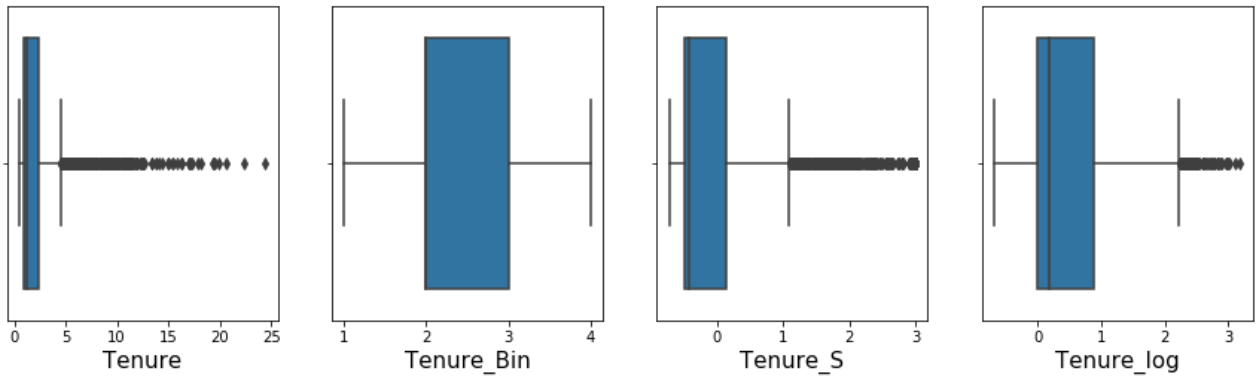
Boxplot 5 – For **SavingPercent**, outliers completely removed by logarithmic transformation:



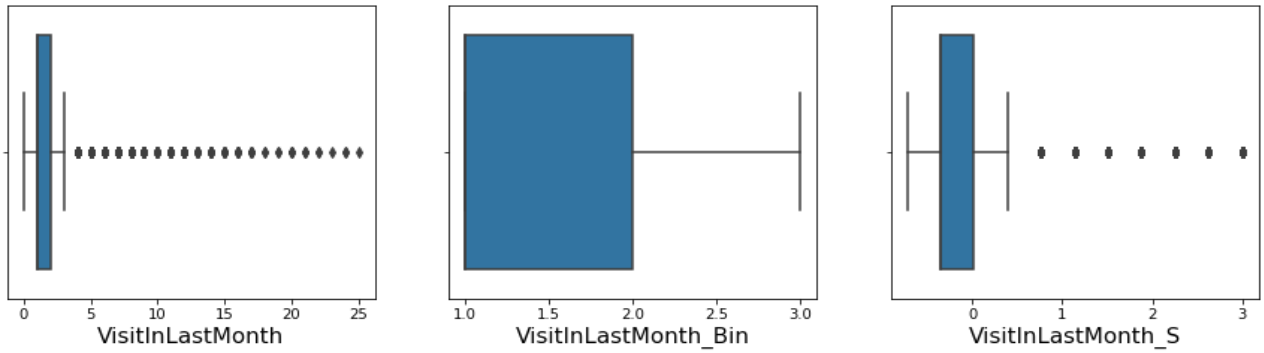




Boxplot 6 – For **Tenure**, outliers have reduced but not considered by either Scaling or log but completely removed by Binning:



Boxplot 7 – For **VisitInLastMonth**, outliers reduced by Standardization but completely removed by Binning:

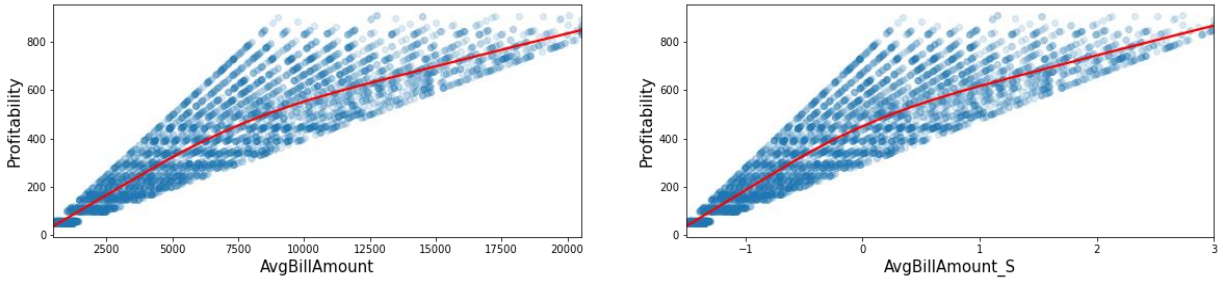


Observations:

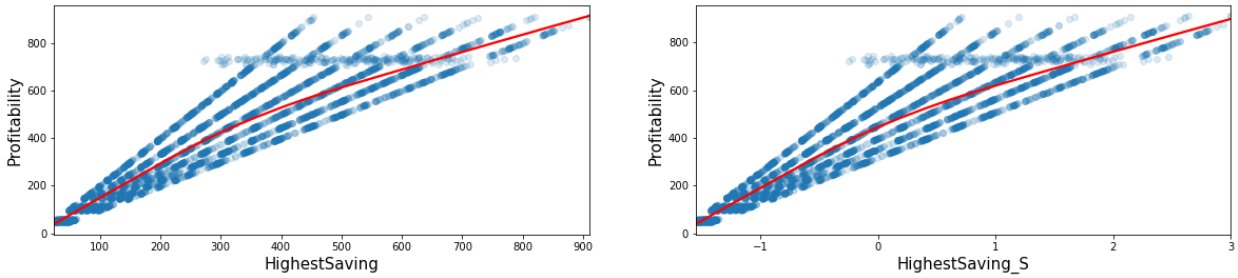
It can be noted that the number of outliers has been impacted greatly after treatment, but we would be keeping all the variables in the dataset and check their impact on the target.

4.4 Study the effect of continuous variables on Profitability

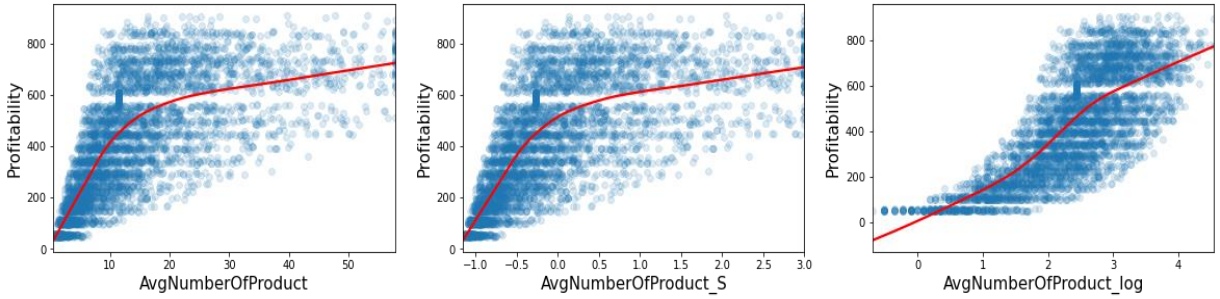
Reg-plot 1 – **AverageBillAmount** shows strong linear relation with Profitability both before and after treatment



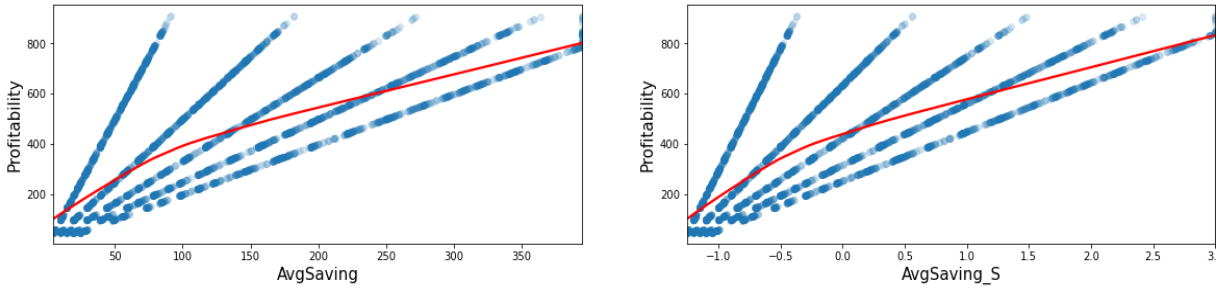
Reg-plot 2 – **HighestSaving** shows strong linear relation with Profitability both before and after treatment



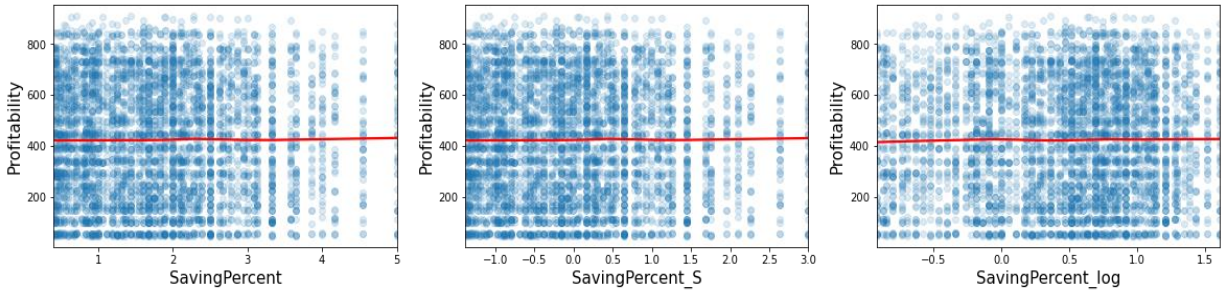
Reg-plot 3 – **AvgSaving** shows considerably good linear relation with Profitability both before and after treatment



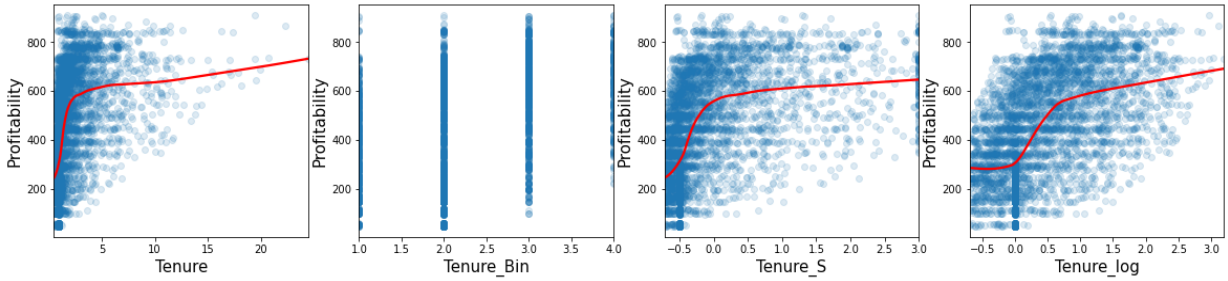
Reg-plot 4 – **AvgNumberOfProduct** has a stronger correlation with Profitability for a lesser number of products than the higher numbers. With log transformation, it has a consistent linear relation



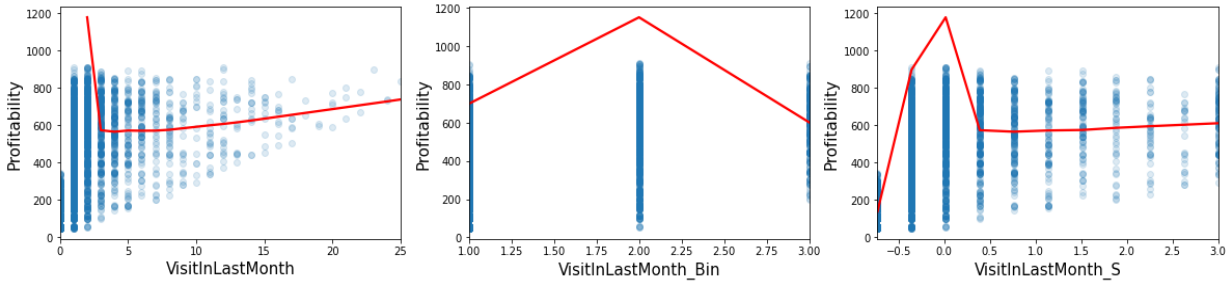
Reg-plot 5 – Between Profitability and **SavingPercent**, no trend is observed



Reg-plot 6 –Strong positive relation with profitability is observed for **Tenure_log** when compared with other variants of this variable.



Reg-plot 7 – **VisitInLastMonth** is giving varied results on different relations with different variants of these variables. No trend could be observed.

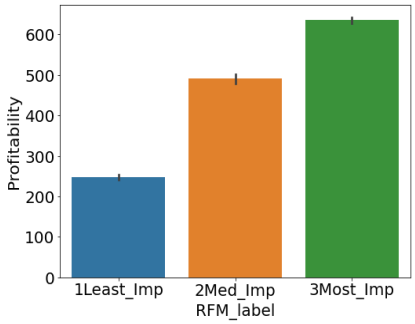


4.5 Study the impact of categorical features on Profitability

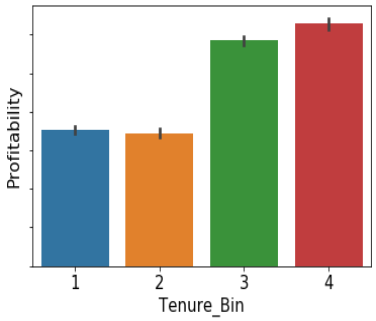
***Checking impact of categorical features on target*.**

Bar-plot 1-3: These variables show a positive trend with the average 'Profitability' in the data. Each level of increase in the order of their respective categories shows a higher impact on the target variable.

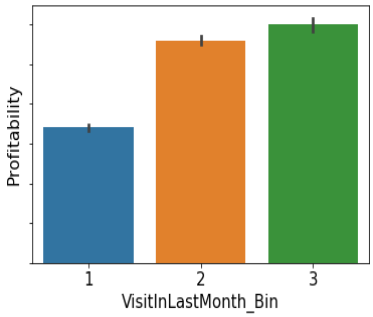
Barplot 1



Barplot 2

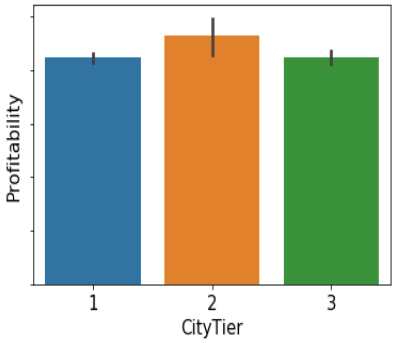


Barplot 3

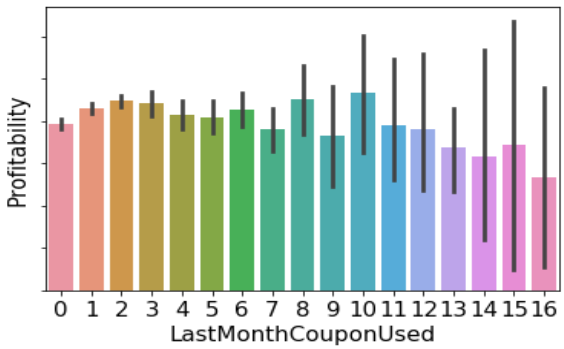


***Bar-plot 4-7*:** These are the ones that show some change in the average 'Profitability' with their categories, but no particular trend could be observed in relation to the target variable.

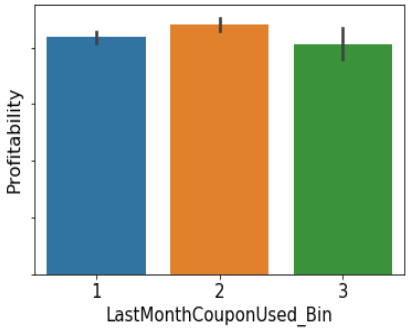
Barplot 4



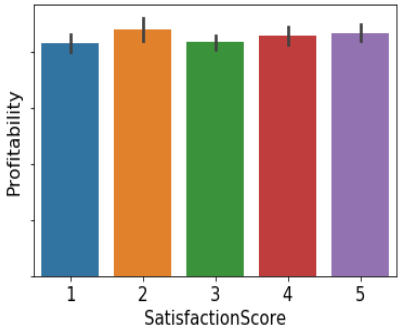
Barplot 5



Barplot 6

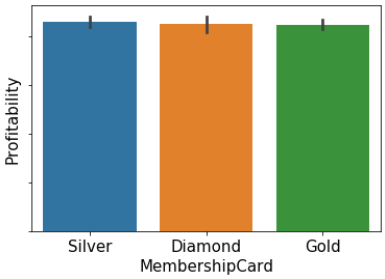


Barplot 7

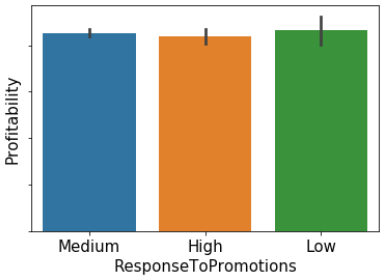


Bar-plot 8-15: For the rest of the above variables, all their categories seem to have a similar impact on the average 'Profitability'. No change of impact is observed on the target variable with the changed category.

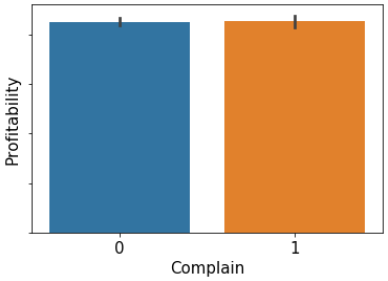
Barplot 8



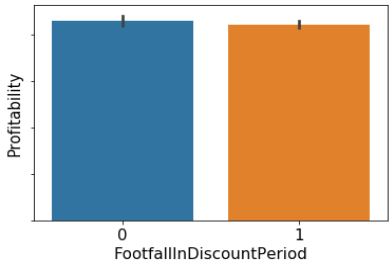
Barplot 9



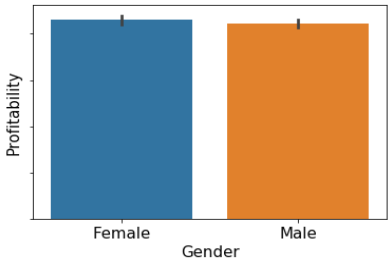
Barplot 10



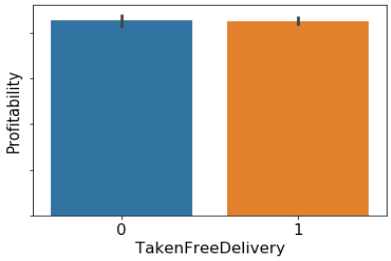
Barplot 11



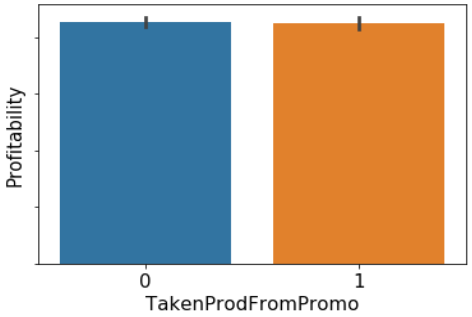
Barplot 12



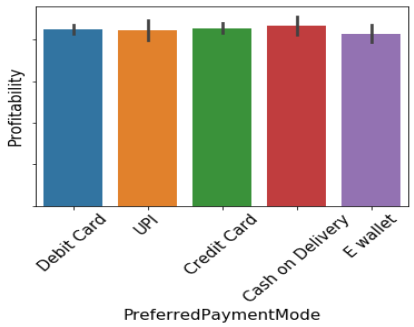
Barplot 13



Barplot 14

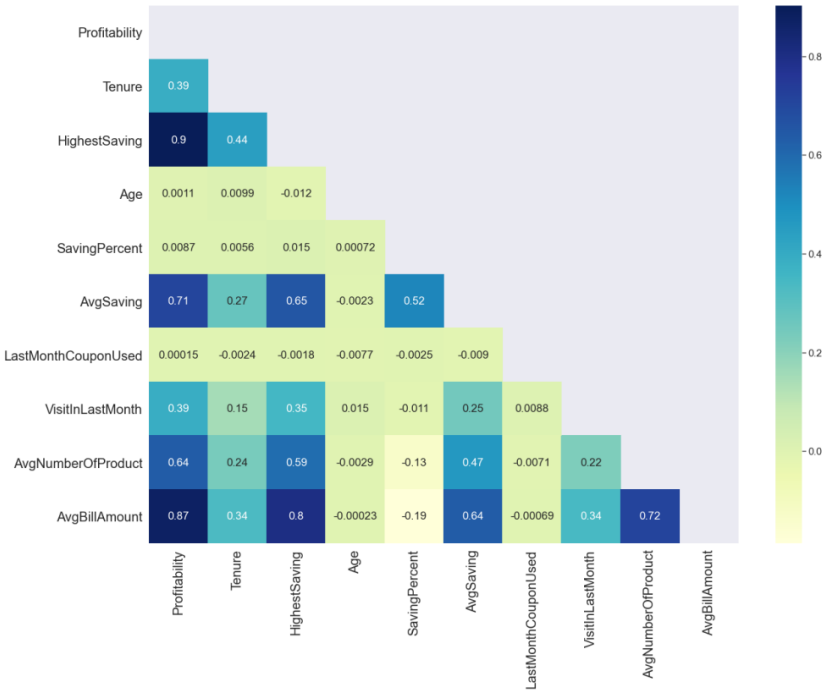


Barplot 15



4.6 Checking co-relation for numerical features.

Heatmap1



Observations:

From the above heatmap, we may notice:

- ✓ There is a very high correlation of 'AvgBillAmount', and 'HighestSaving' with the target variable 'Profitability'.
- ✓ There is a positive but moderately high correlation of 'AvgNumberOfProduct' and 'AvgSaving' with 'Profitability'.
- ✓ Besides, some independent variables like 'AvgBillAmount', 'HighestSaving' and 'AvgNumberOfProduct' also show a high correlation among them which proves the existence of multicollinearity in the data.

Section 5: Feature Selection

Now that we have a dataset with all original features and transformed features plus new features, we need to find out the best or most effective features. We calculated feature importance for all 36 variables with four different methods and made two sets of feature lists as below:
Set 1 Features - shortlisted from Variation Inflation Factor.
Set 2 Features - shortlisted from combining results of Select K Best, Extra Tree Regressor, Lasso Regression.

5.1 Variation Inflation Factor (VIF):

This technique is used to detect and remove multicollinearity. Multicollinearity occurs when there are two or more independent variables in a multiple regression model, which have a high correlation among themselves. When some features are highly correlated, we might have difficulty distinguishing between their individual effects on the dependent variable. VIF method picks each feature and regresses it against all of the other features. For each regression, the factor is calculated by the formula:
VIF = $1/(1-R^2)$
Here, R^2 is the coefficient of determination in linear regression and its value lies between 0 and 1.
We may observe, greater the value of R-squared, the greater is the VIF. As we know, a higher R-squared value denotes stronger collinearity, hence, greater VIF denotes greater correlation. Generally, a VIF above 5 indicates high multicollinearity.
Our results from VIF (*after eliminating VIF >5)* come down to 19 features and we call this Set1 Features:

Table 9: Features list from VIF technique – Set 1 Feature

S No	Set 1 Feature	VIF
1	CityTier	4.92
2	PreferredPaymentMode_Debit Card	3.70
3	AvgSaving_S	3.64
4	MembershipCard_Gold	3.42
5	TakenFreeDelivery	3.11
6	HighestSaving_S	2.92
7	PreferredPaymentMode_Credit Card	2.99
8	FootfallInDiscountPeriod	2.99
9	MembershipCard_Silver	2.55
10	Gender_Male	2.37
11	PreferredPaymentMode_E wallet	2.28
12	SavingPercent_S	2.11
13	AvgNumberOfProduct_S	1.79
14	TakenProdFromPromo	1.66
15	PreferredPaymentMode_UPI	1.46
16	Complain	1.38
17	Tenure_S	1.30
18	VisitInLastMonth_S	1.15
19	LastMonthCouponUsed_S	1.03

We applied the above Set 1 features to all the models and evaluated the performance.
Next, we used three more techniques for feature selection as below:

5.2 SelectKBest:

This is a class in sklearn.feature_selection module which can be used for feature selection/dimensionality reduction on datasets by providing the argument ‘k’ for the number of best features we want. This algorithm helps in reducing overfitting, improving accuracy, and hence, reducing training time. The list is summarised in Table 9.

5.3 Extra Tree Regressor:

The Extra Trees algorithm works by creating a large number of unpruned decision trees (extra trees) from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression. The list is summarised in Table 9.

5.4 Lasso Regression:

Using Linear Regression with L1 regularization is called Lasso Regularization. In the Lasso Feature selection method a threshold is decided for beta value and the features above the threshold values are considered as important features and below the feature value are assigned zero beta value. The list is summarised in Table 9 and the *14 shortlisted features are marked in **bold***.

Table 10: List of features importance wise from the above three methods

Index	Feature Name	Select_K	Extra_Tree	Lasso
1	**HighestSaving**	19956.04	0.252	91.112
2	**AvgBillAmount**	14814.64	0.152	73.766
3	**AvgNumberOfProduct_log**	6399.88	0.004	34.411
4	**AvgSaving_S**	4552.87	0.014	24.023
5	HighestSaving_S	19964.11	0.194	14.152
6	**RFM_label**	6520.30	0.178	11.636
7	**VisitInLastMonth_Bin**	1186.43	0.002	10.079
8	**SavingPercent**	0.34	0.003	9.149
9	**Tenure_log**	1958.48	0.002	8.107
10	**Complain**	0.05	0.002	2.385
11	**CityTier**	0.08	0.002	1.600
12	**SatisfactionScore**	2.18	0.002	1.204
13	**Age**	0.05	0.003	0.773
14	**LastMonthCouponUsed_Bin**	2.31	0.002	0.368
15	VisitInLastMonth_S	1066.54	0.002	0.159
16	AvgBillAmount_S	14778.45	0.131	0.000
17	AvgSaving	4561.30	0.014	0.000
18	Tenure_S	1274.21	0.002	0.000
19	Tenure_Bin	1037.97	0.002	0.000
20	VisitInLastMonth	830.78	0.002	0.000
21	FootfallInDiscountPeriod	1.26	0.002	0.000
22	ResponseToPromotions	0.74	0.001	0.000
23	PreferredPaymentMode_Ewallet	1.59	0.001	-0.716
24	**Gender_Male**	0.90	0.002	-1.069
25	Tenure	1015.91	0.002	-2.467
26	SavingPercent_log	0.43	0.003	-4.818
27	AvgNumberOfProduct_S	3391.04	0.003	-7.855
28	AvgNumberOfProduct	3274.60	0.004	-11.736

From the results of the above three techniques, we compared and brought down a common set of features calling them *Set 2 Features* which were applied to different models in our project. Again, we compared results and evaluated the performance of each model on the below set of features deduced from the above comparison.

Table 11: Set 2 Features from the above three techniques

S No	Set 2 Features
1	'HighestSaving'
2	'AvgBillAmount'

3	'AvgNumberOfProduct_log'
4	'AvgSaving_S'
5	'RFM_label'
6	'VisitInLastMonth_Bin'
7	'SavingPercent'
8	'Tenure_log'
9	'Complain'
10	'CityTier'
11	'SatisfactionScore'
12	'Age'
13	'LastMonthCouponUsed_Bin'
14	'Gender_Male'

Section 6: Model Selection

The next step after feature selection is to decide the model for our data and use our selected features and apply our model and evaluate. We tried three models in this exercise:

6.1 Linear Regression:

It is the most common and widely used algorithm Machine Learning algorithm for establishing a linear relationship between the target variable and the response variables. Based upon the following equation:

$$y = \theta 0 + \theta 1x1 + \theta 2x2 + \theta 3x3 + \ldots\ldots + \theta nxn$$

where y is the target variable
θ0 is the intercept
x1,x2,x3,….,xn are independent variables, and
θ1, θ2, θ3,……., θn are their respective coefficients.

This model aims to find the best fit line between the target variable and the independent variables by finding the most optimal values for all θ.

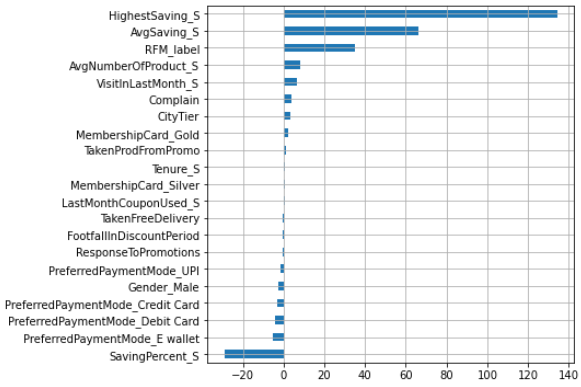
Since the Linear regression model is sensitive to certain assumptions including multicollinearity, we trained the model with the Set 1 features from the VIF technique only. The results are as below:

Table 12: Evaluation metrics for LR model

Model Name	Linear Regression -Train	Linear Regression-Test
MAE	62.09	44.89
MSE	6555.97	6824.92
RMSE	80.97	82.61
R-SQ	0.88	0.87
AD-RSQ	0.88	0.86

The model seems to have a good performance for both train and test. However, if we look into the individual parameters, we may find a few features are misguiding.

Figure1: Feature coefficients for LR model



While on one hand, the highest saving is a strong positive contributor to profitability, Saving_Percent_S is showing a negative impact on Profitability which does not make sense. Also, Gender and Payment modes showing negative relation with profitability is not logically acceptable in general. Hence, we should look into more techniques to develop our model.

6.2 Random Forest Regression:

The next model we tried is Random Forest which is based on the bagging algorithm and uses the Ensemble Learning technique. In this way, it reduces the overfitting problem in decision trees and also reduces the variance, and therefore improves the accuracy.

Also, the Random Forest algorithm is very stable. Even if a new data point is introduced in the dataset, the overall algorithm is not affected much since the new data may impact one tree, but it is very hard for it to impact all the trees. Since this model does not have any assumptions, we used the second set of selected features for this model.

The results are as below:

Table 13: Metrics with Set 1 features

Model Name	Random Forest - Train	Random Forest - Test
MAE	15.02	44.89
MSE	512.15	3669.78
RMSE	22.63	60.58
R-SQ	0.99	0.93
AD-RSQ	0.99	0.93

Table 14: Metrics with set 2 features

Model Name	Random Forest - Train	Random Forest - Test
MAE	14.68	44.89
MSE	482.09	3579.08
RMSE	21.96	59.82
RSQ	0.99	0.93
AD-RSQ	0.99	0.93

As we can see, for set 1 and set 2 features both, the results are almost the same and but over-fitting seems to be an issue in both cases. Hence, we may try the next model.

6.4 Lasso Regression:

It is a type of linear regression that uses shrinkage. In the *Shrinkage* method data values are shrunk towards a central point, like the mean. The lasso model helps to produce simple and sparse models (i.e. models with fewer features). Lasso is well-suited for models showing high multicollinearity. The advantage of Lasso is that automatically reduces the number of features or parameters in a model according to a threshold value which reduces noise and makes the model more effective.

Here we used set-2 of 14 selected features mentioned before.

Starting with the above 14 features, we tried several iterations of the model by dropping variables one by one for reasons explained below:

Table 15: Iterations of Lasso Regression starting from 14 features and dropping step by step

Iteration	Data	Features used/dropped with reasons	MAE	MSE	RMSE	R-SQ	ADJ R-SQ
1	Train	all 14 as from Lasso Set 2 features	55.35	5516.71	74.27	.90	.90
	Test		55.62	5709.21	75.56	.89	.89
2	Train	all above minus LastMonthCouponUsed as the beta value is zero	55.35	5516.71	74.27	.90	.90
	Test		55.62	5709.21	75.56	.89	.89
3	Train	Above minus Gender_male as Gender as a driver of profitability is not very reasonable and also beta is very small and negative	55.35	5517.60	74.28	.90	.90
	Test		55.62	5708.35	75.55	.89	.89
4	Train	above minus complain as complain having a positive impact on profitability does not seem logical	55.37	5525.71	74.34	.90	.90
	Test		55.59	5704.60	75.53	.89	.89
5	Train	all above minus average bill amount as it is collinear with average saving, hence, we can drop the one with a lower beta value	60.23	6286.96	79.29	.88	.88
	Test		60.70	6549.00	80.92	.87	.87
6	Train		61.13	6509.64	80.68	.88	.88

	Test	all above minus saving_percent as saving_percent showing negative effect on the profitability does not make sense	61.99	6832.57	82.66	.87	.87

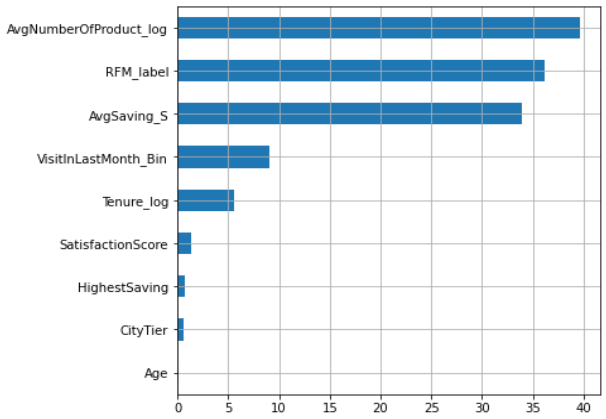
Section 7: Model Evaluation

On comparing the above metrics, we have the following observations:

- ✓ When comparing linear models and tree-based models, *linear models have better performance* than tree-based models.
- ✓ Out of the above three, we find Lasso is the most robust model and easy to relate with our business problem.
- ✓ While in Random Forest, R-squared values for train and test data are 99% and 93%, for Lasso the model performance is only 88% and 86.6% for train and test data respectively *leading to over fitting in Random Forest*
- ✓ Adjusted R-squared values for train and test data in Random Forest, the difference is much higher as compared to the difference of Adjusted R-squared in Lasso Regression which indicates the dependent variables selected for model training are not able to capture the same effect in test data as in the train data
- ✓ While comparing the other metrics, it is obvious to notice that the error terms like MAE, MSE, and RMSE values for both train and test data in Random Forest are lower than that in Lasso due to better performance, however, if we compare between train and test error values Random Forest is not reliable for prediction as the error values jumped up severely for test data.
- ✓ For our purposes we may compare MAE which gives a better picture of the actual difference, we may notice that the Mean Absolute Error for Random Forest train and test is approx. 14.98 and 41.20 respectively and for Lasso train and test data, it is 61.14 and 61.98 respectively. Here, we may notice that the absolute difference between y and y-hat is negligible in Lasso and quite substantial in random Forest.
- ✓ Again, if we compare the MAE value for train data in RF with train data in Lasso, RF seems to give excellent results (due to over fitting) and very low value as compare to Lasso but when we compare test MAE value for RF with test MAE value of Lasso, the difference becomes smaller suggesting Lasso as a more robust model.
- ✓ Finally, we may evaluate our model by its interpretability as that is one of the objectives of our project. In order to improve profitability, it is important to know which features impact profitability and to what extent. Linear models are easily explainable by a mathematical equation and make it more practical to apply the outcomes in business strategy. From the figure below, we see the important predictors in order of their coefficient value whose impact can be deduced with the linear equation in the recommendations section.

Figure 2: Important features from the final Lasso Regression model

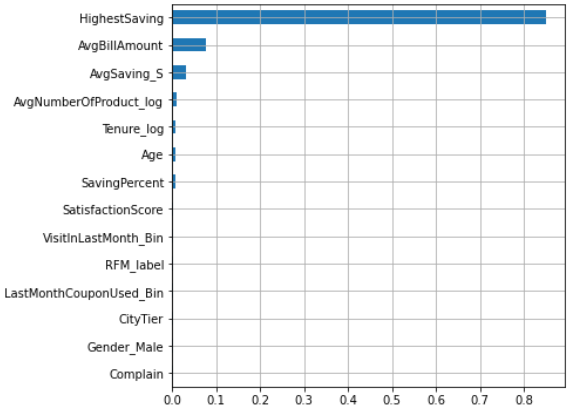
	features	Imp
1	AvgNumberOfProduct_log	39.645093
3	RFM_label	36.130139
2	AvgSaving_S	33.917030
4	VisitInLastMonth_Bin	9.000560
5	Tenure_log	5.561592
7	SatisfactionScore	1.285330
0	HighestSaving	0.707149
6	CityTier	0.557718
8	Age	0.160170



✓ In the case of tree-based models like Random Forest, the model only provides a value for each predictor in order of its significance in the model but does not provide any information on how the predictor will impact the target. Hence, if we want to make changes in the target through predictors, we cannot deduce any relationship between the two from the model. From the below figures, we may observe the feature importance provided by the Random Forest model:

Figure 3: Important features from the final Random Forest Model

	features	Imp
0	HighestSaving	0.850823
1	AvgBillAmount	0.076788
3	AvgSaving_S	0.029975
2	AvgNumberOfProduct_log	0.010362
7	Tenure_log	0.007880
11	Age	0.006820
6	SavingPercent	0.006559
10	SatisfactionScore	0.002447
5	VisitInLastMonth_Bin	0.002322
4	RFM_label	0.001528
12	LastMonthCouponUsed_Bin	0.001249
9	CityTier	0.001123
13	Gender_Male	0.001121
8	Complain	0.001002



✓ Based on the above observations, we choose Lasso Regression as our model of choice for this project.

Section 8: Recommendation and Conclusion

From the observations above we may conclude features are very important for the model and have a strong impact on profitability

Table 16: Final features and coefficient values

SN	Features	Coefficient
1	AvgNumberOfProduct_log	39.66
2	RFM_label	36.13
3	AvgSaving_S	33.92
4	VisitInLastMonth_Bin	9.00
5	Tenure_log	5.56
6	SatisfactionScore	1.29
7	HighestSaving	0.71
8	CityTier	0.56
9	Age	0.16

1. ***AvgNumberOfProduct*** comes out to be the most important feature which means an increase in the number of products sold enhances profitability and it is reasonable to understand that selling more products means adding more to margins and hence, profitability increases for the business.
2. ***RFM_label*** also shows a high positive relationship with profitability. RFM_label which is a derived variable denotes the customers in three slabs i.e., High, medium, and low. Businesses must advise policies to get customers to be high on all the three i.e., recency, frequency, and monetary (spending) which will, in turn, increase their RFM_label rating.
3. ***AvgSaving*** is another very strong driver of profitability according to the model. It makes sense as we understand that customers attract to deals where they can save money. Also, saving money in a way encourages customers to buy more too.
4. ***VisitInlastMonth*** is another significant feature that is contributing positively to an increase in profitability. The point is quite logical as the more the number of visiting customers, the more will be the sales and hence, profitability.

5. ***Tenure*** is another important feature in the model which affects profitability positively. It is very logical to understand that maintaining old customers is less costly than bringing in new customers, hence, the ability to retain customers adds to profitability.
6. ***SatisfactionScore:*** also comes out to be an important variable for improving profitability. It is a known rule that a satisfied customer means a potential repeating customer meaning more sales and profitability.
7. ***More features of Importance to observe:*** In continuation to above, we have observed there are few features like ***MembershipCard*** and ***ResonseToPromotion (derived from coupons, free delivery, footfall in discount period)*** that exist in the dataset which should intuitionally have shown positive linear relation with profitability, but *none of them shows any significant effect on profitability directly nor appears in the list of the important features*. Here, we may advise that the Marketing department should relook into the guiding parameters for these promotional strategies to make them effective.

Section 9: Model Deployment – Basic demonstration

Model Deployment is a key aspect of every ML project and is very important when you must communicate your work to the client who needs to see it in an easy-to-understand interface. Now that we have finalized our model and determined the important features, we deployed our model into a basic demonstration mode by using Flask and HTML.

Project Structure:

dep_model.ipynb — Contains code for the ML model to predict profitability and pickling.
lasso.pickle – Contains dep-model.ipynb code after pickling for serialization / deserialization.
app.py — Contains Flask APIs that receive inputs through GUI or API calls, computes the predicted value based on our model, and returns it.
index.htm — Contains the HTML template and CSS styling to allow the user to enter feature inputs.
result.html – Contains template for displaying the predicted output i.e., profitability.

Model Interface for user:

Figure4: Input User Interface

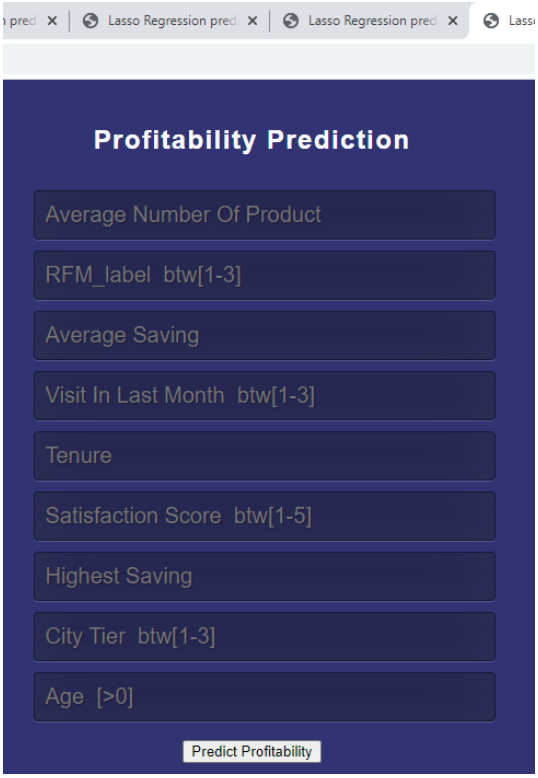
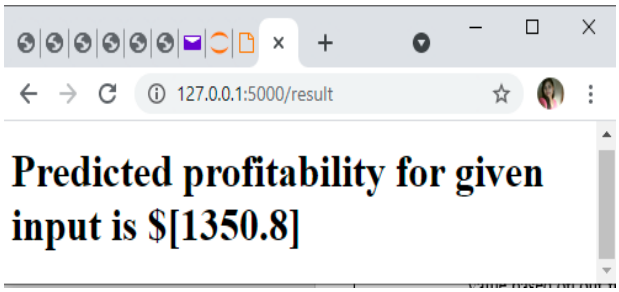


Figure5: Output User Interface



Bibliography

1. Mathias Kinnander (Spring term 2020). PREDICTING PROFITABILITY OF NEW CUSTOMERS USING GRADIENT BOOSTING TREE MODELS
2. K.C. Barmola & S.K. Srivastava (2010). THE ROLE OF CONSUMER BEHAVIOUR IN PRESENT MARKETING MANAGEMENT SCENARIO
3. Dr. Sunil Kakkar, Dr. Surbhi Mathur (April 2020). ORGANISED RETAIL IN INDIA AND CONSUMER BUYING BEHAVIOUR: A REVIEW OF LITERATURE
4. V. Kumar, Denish Shah, Rajkumar Venkatesan (December 2020). Managing retailer profitability—one customer at a time!
5. The Retail Profitability Challenge - A brave new world- September 2017-Deloitte (https://www2.deloitte.com/content/dam/Deloitte/be/Documents/Financial%20Advisory/The%20Retail%20Profitability%20Challenge.pdf)
6. Mahendra Gupta, George foster (January 1997). Customer Profitability Analysis: Challenges and New Directions-
7. Bart Larivie`re, Dirk Van den Poel (2005). Predicting customer retention and profitability by using random forests and regression forests techniques
8. Roland T. Rust V. Kumar, Rajkumar Venkatesan (2011). Will the frog change into a prince? Predicting future customer profitability
9. Author: Peter L. Flom, David L. Cassell (2008). Stopping stepwise: Why stepwise and similar selection methods are bad, and what you should use.
10. Valeria Fonti (March 2017). Feature Selection using LASSO
11. Deanna Schreiber-Gregory, Henry M Jackson Foundation (2018). Regulation Technique for Multicollinearity: Lasso, Ridge and Elastic Net
12. Data Vedas (February 2018). Outlier Treatment. (https://www.datavedas.com/outlier-treatment/)
13. Chris Moffitt (October 2019). Binning Data with Pandas qcut and cut (https://pbpython.com/pandas-qcut-cut.html)
14. Deepanshu Bhalla (2015). Detecting and solving the problem of Outlier. (https://www.listendata.com/2015/01/detecting-and-solving-problem-of-outlier.html)

Appendix

Data dictionary:

Data	Variable	Descerption
Retail	CustomerID	Unique customer ID
Retail	Profitiblity	Profitability from each customer
Retail	Tenure	Tenure of customer in organization (in Years)
Retail	TakenFreeDelivery	Opted for free home delivery (1 indicates Yes)
Retail	CityTier	Tier of city
Retail	HighestSaving	Highest saving made on a particular visit
Retail	PreferredPaymentMode	Preferred payment method of customer
Retail	Gender	Gender of customer
Retail	Age	Age of customer
Retail	FootfallInDiscountPeriod	Customer visit during discount period flag (1 indicates Yes)
Retail	TakenProdFromPromo	Product taken from promotional counter flag (1 indicates Yes)
Retail	SatisfactionScore	Satisfactory score of customer on service (5 being highest Rating)
Retail	MembershipCard	Existing membership type of customer
Retail	SavingPercent	Average saving percent from total average bill
Retail	Complain	Complaint registered in last one month
Retail	AvgSaving	Average saving in last one year
Retail	LastMonthCouponUsed	Coupon used by customer in last month
Retail	VisitInLastMonth	Total visit by customer in last month
Retail	AvgNumberOfProduct	Average number of product taken by a customer in one visit
Retail	AvgBillAmount	Average bill amount by a customer in one visit

Codes:

File1:Datapreprocessing.ipynb

```
# Problem Statement
The dataset belongs to a Supermarket chain which is growing at a rapid pace. The CMO of this company understands the rising competition which may impact the profitability of the store and thus wishes to use the existing data to understand the buying behavior of its customer. In order to understand the buying behavior of the store's regular customers, he wants to analyze profitability of each customer who has ever visited the store. This study will help the company to modify the marketing strategies for the customers.

**Importing important libraries**

from matplotlib import pyplot as plt
import matplotlib.cm as cm
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler
%matplotlib inline

**Loading the file**

df=pd.read_excel("Retail.xlsx",sheet_name='Retail')

df.head()
```

```
df.tail()

With the help of head and tail, we can see that data is loaded properly

print("Dataset has {} rows and {} columns".format(df.shape[0],
df.shape[1]))

df.info()

With the help of info function below information is derived:
1. All the columns are not having same number of non-null values, which
shows there are null values present in data.
2. 3 columns have data type as object, it means dataset has continuous
and categorical data.

Using describe function to review statistical summary like, Mean, Median,
Standard deviation and 1st & 3rd quartile and min-max of columns

df.describe().transpose()

Below are the observations:
    1. With the help of max value, we can see that values of different
columns are varying from 22389.97 for AvgBillAmount to 1 for Complain.
So, we need to standardise the columns.
    2. TakenFreeDelivery, FootfallInDiscountPeriod, TakenProdFromPromo
and Complain columns are having standard deviation ranging between 0.45
to 0.48.

start = "\033[1m"
end = "\033[0;0m"
print(start+ "ColumnWise Null values:"+ end)
print(df.isnull().sum())
print("\n")
print(start+"Total null Values in dataset ="+
end,df.isnull().sum().sum())

#printing names of all columns
print(start+ "Name of all the columns of the dataset are:\n"+
end,df.columns)

#Checking duplicate values in dataset
print(start+ "Total number of duplicate values in dataset ="+
end,df.duplicated().sum())

#Checking unique values in dataset
print(start+ "Unique values in each Columns:"+ end)
for col in (df.select_dtypes(['object','int64'])).columns:
```

```
    print('Column {} has {} unique values:\n{}'.format(col,len(df[col].unique()),df[col].value_counts(dropna=False)))

print('==========================================================\n')

# As confirmed the values COD & Cash On Delivery in 'PreferredPaymentMode' column, M & Male in 'Gender' column &
# G and Gold in'MembershipCard' column are typo errors repeated by mistake and have same meaning,so fixing them first.

df.replace(['COD','M','G'],['Cash on Delivery','Male',"Gold"],inplace=True)

# Checking values after replacement

l=['PreferredPaymentMode','Gender','MembershipCard']
for col in l:
    print('Column {} now has {} unique values: {} \n'.format(col, df[col].nunique(), df[col].unique()))

#Checking Histogram to show distribution of each column
df.hist(figsize=(20,15))

# Null Values Imputation

**Before moving forward, it is important to impute null values as in total 1655 values are missing in data**

#All columns having null values
df[df.columns[df.isnull().sum()!=0]].isnull().sum().sort_values(ascending=False)

# Null values imputation of 'PreferredPaymentMode'and 'MembershipCard' using Mode

For the categorical variables 'PreferredPaymentMode' and 'MembershipCard', the missing values are less than 2% of the total records, we can simply use mode of each column to impute the null values.

**Preferred Payment Mode**

#Checking unique Payment modes with their count and mode of the variable PreferredPaymentMode

print(start+ "Count of rows having unique Payment modes for each type:"+end)
print(df['PreferredPaymentMode'].value_counts())
print("\n")
```

```
print(start+"Mode of Variable PreferredPaymentMode ="+
end,df['PreferredPaymentMode'].mode())

#Filling null values with Mode that is Debit Card in PreferredPaymentMode
df['PreferredPaymentMode'].fillna(value = 'Debit Card', inplace = True)

#Checking if null values still there and change in number of counts after
imputation

print(start+ "Number of Null values in PrefferedPaymentMode:"+ end)
print(df['PreferredPaymentMode'].isnull().sum())
print("\n")
print(start+ "Count of rows having unique Payment modes for each type
after Null values Imputation:"+ end)
print(df['PreferredPaymentMode'].value_counts())

**Membership Card**

#Checking unique Membership Cards with their count and mode of the
variable MembershipCard

print(start+ "Count of rows having unique Membership Card for each
type:"+ end)
print(df['MembershipCard'].value_counts())
print("\n")
print(start+"Mode of Variable MembershipCard ="+
end,df['MembershipCard'].mode())

#Filling null values with Mode that is Gold in MembershipCard
df['MembershipCard'].fillna(value = 'Gold', inplace = True)

#Checking if null values still there and change in number of counts after
imputation

print(start+ "Number of Null values in MembershipCard:"+ end)
print(df['MembershipCard'].isnull().sum())
print("\n")
print(start+ "Count of rows having unique Membership Card for each type
after Null values Imputation:"+ end)
print(df['MembershipCard'].value_counts())

# Null values imputation of 'AvgNumberOfProduct' using Median

**AvgNumberOfProduct**

#Calculating Median of AvgNumberOfProduct
print(start+ "Median of AvgNumberOfProduct before Null values
Imputation is:"+ end, df['AvgNumberOfProduct'].median())
```

```
#Filling null values with Median that is 11.4 in AvgNumberOfProduct
df['AvgNumberOfProduct'].fillna(df['AvgNumberOfProduct'].median(),
inplace=True)

#Checking if null values still there and change in median after
imputation

print(start+ "Number of Null values in AvgNumberOfProduct:"+ end)
print(df['AvgNumberOfProduct'].isnull().sum())
print("\n")
print(start+ "Median  of AvgNumberOfProduct after Null values Imputation
is:"+ end, df['AvgNumberOfProduct'].median())

# Null values imputation of 'AvgSaving' and 'AvgBillAmount' using 'Saving
percent' Variable

**AvgSaving**

**(Formula For AvgSaving: AvgBillAmount * SavingPercent = AvgSaving)**

#Null Value imputation with the help of formula mentioned above which
is derived and verified through manual computation
df['AvgSaving'].fillna((df.AvgBillAmount   *   df.SavingPercent)   /100,
inplace=True)

#Checking Null Values if there are still present
df['AvgSaving'].isnull().sum()

**AvgBillAmount**

**(Formula   For   AvgBillAmount   :   AvgSaving   *   SavingPercent   =
AvgBillAmount)**

#Null Value imputation with the help of formula mentioned above which
is derived and verified through manual computation
df['AvgBillAmount'].fillna((df.AvgSaving   *   100)   /df.SavingPercent,
inplace=True)

#Checking Null Values if there are still present
df['AvgBillAmount'].isnull().sum()

# Null values imputation of 'HighestSaving' using Linear Regression

**HighestSaving**

#Checking Correlation of HighestSaving with other Variables
df[df.columns[1:]].corr()['HighestSaving']
```

```
#Creating New DataFrame, consisting of only such columns which are having
correlation above 0.3 and HighestSaving.
#Have tried deriving null values with different components, but this one
gives best result
df_LR=df[["HighestSaving","AvgBillAmount","AvgSaving","AvgNumberOfProd
uct","Tenure","VisitInLastMonth"]]

#Dividing new dataframe into test and train.
#train consisting of all such rows of HighestSaving where, there is no
Null Value
#Test dataset consists of such rows where the HighestSaving is Null

train_df_LR = df_LR[df_LR['HighestSaving'].notna()]
test_df_LR = df_LR[df_LR['HighestSaving'].isnull()]

#Checking if test and train data has correctly splitted

print(start+"5 rows of Test Data"+end)
print(test_df_LR.head())
print(start+"5 rows of Train Data"+end)
print(train_df_LR.head())

#Checking Null Values in train data for other columns
train_df_LR.isna().sum()

#Dropping all the null values fom train data to train in best way
train_df_LR = train_df_LR.dropna()

#Checking Null Values in test data for other columns
test_df_LR.isna().sum()

#Splitting train and test Data into X and Y

train_X = train_df_LR.drop('HighestSaving', axis=1)
train_y = train_df_LR.HighestSaving


test_X = test_df_LR.drop('HighestSaving', axis=1)
test_y = test_df_LR.HighestSaving

#Checking the shape of test_X dataset
test_X.shape

#importing important libraries for linear regression

import statsmodels.api as sm
from sklearn import linear_model
regr = linear_model.LinearRegression()
```

```
#Building Model
train_X_sm = sm.add_constant(train_X)
model = sm.OLS(train_y,train_X_sm).fit()
model.summary()

#Predicting y for train data to check Error rate
train_y_pred = model.predict(train_X_sm)
print(train_y_pred)

#Calculating Error rate
MSE =(np.sum(np.square(train_y - train_y_pred)))/train_df_LR.shape[0]
print ("MSE =", MSE)
print ("RMSE =", np.sqrt(MSE))

#Plotting Graph for error terms
res = (train_y - train_y_pred)
fig = plt.figure()
sns.distplot(res, bins = 15)
plt.title('Error Terms', fontsize = 15)
plt.xlabel('train_y - train_y_pred', fontsize = 15)
plt.show()

#Predicting Null Values for HighestSaving

test_X_sm = sm.add_constant(test_X)
test_y_pred = model.predict(test_X_sm)
print(test_y_pred)

#Creating a dataframe for filling the null value in HighestSaving
temp=model.predict(test_X_sm)[df['HighestSaving'].isnull()]

#Filling Null Values in original DataFrame with Predicted Values
df['HighestSaving'].fillna(temp,inplace=True)

#Checking Mean and Count of Null values

print(start+ "Number of Null values in HighestSaving:"+ end)
print(df['HighestSaving'].isnull().sum())
print("\n")
print(start+ "Mean  of HighestSaving after Null values Imputation is:"+
end, df['HighestSaving'].mean())

#Checking distribution of HighestSaving
fig = plt.figure()
sns.distplot(df['HighestSaving'], bins = 15)

# Null values imputation of 'Tenure', 'Age' and 'VisitInLastMonth' using
KNN
```

```
**Before applying KNN we will convert Categorical data into numerical with the help of One Hot Encoding**

# One Hot Encoding

#Creating new dataset, removing Customer_ID which is not important but will impact KNN result
Df1=df_1=df.drop('CustomerID',axis=1)
Df1.head()

#One Hot Encoding for converting categorical data to continuous
df_dum = pd.get_dummies(Df1, drop_first=True)
df_dum.head(2)

#printing names of all columns
print(start+ "Name of all the columns of the dataset are:\n"+ end,df_dum.columns)

#importing libraries for KNN
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5)

#Creating new dataset for knn
df_knn = df_dum.copy()

imputed_knn = imputer.fit_transform(df_knn)
imputed_knn

df_knn = pd.DataFrame(imputed_knn, columns = df_knn.columns)
df_knn

#Making new copy of the data for further work
Data=df.copy()

Data.head(2)

**Imputing null values with KNN values in column Age, VisitInLastMonth and tenure**

**Age**

print(start, "Unique Values in column Age before Imputation:", end,'\n',df['Age'].unique())
print(start, "Total Unique Values in column Age before Imputation:", end,df['Age'].nunique())

print(start, "Total Null Values in Age Column:", end,df['Age'].isnull().sum())
```

```
**From the below code we can see that for the columns 'Age' and 'VisitInLastMonths', the KNN imputer has introduced new float values where as the original values are integer, so it is important that when we copy the imputed values in the original df, we should round off the values, so that no new values are introduced**

print(start, "Unique Values in column Age after KNN:", end,'\n',df_knn['Age'].unique())
print(start, "Total Unique Values in column Age after KNN:", end,df_knn['Age'].nunique())

#Rounding off the values of Age that found through KNN
Data['Age']=np.round(df_knn['Age'].values)

print(start, "Total Unique Values in column Age after KNN and rounding it off:", end,Data['Age'].nunique())
print(start, "Total Null Values in Age Column after Imputation:", end,Data['Age'].isnull().sum())
print(start, "Mean of Age Column after Imputation:", end,Data['Age'].mean())

**VisitInLastMonth**

print(start, "Unique Values in column VisitInLastMonth before Imputation:", end,'\n',df['VisitInLastMonth'].unique())
print(start, "Total Unique Values in column VisitInLastMonth before Imputation:", end,df['VisitInLastMonth'].nunique())

print(start, "Total Null Values in VisitInLastMonth Column:", end,df['VisitInLastMonth'].isnull().sum())

print(start, "Unique Values in column VisitInLastMonth after KNN:", end,'\n',df_knn['VisitInLastMonth'].unique())
print(start, "Total Unique Values in column VisitInLastMonth after KNN:", end,df_knn['VisitInLastMonth'].nunique())

#Rounding off the values of VisitInLastMonth that found through KNN
Data['VisitInLastMonth']=np.round(df_knn['VisitInLastMonth'].values)

print(start, "Total Unique Values in column VisitInLastMonth after KNN and rounding it off:", end,Data['VisitInLastMonth'].nunique())
print(start, "Total Null Values in VisitInLastMonth Column after Imputation:", end,Data['VisitInLastMonth'].isnull().sum())
print(start, "Mean of VisitInLastMonth Column after Imputation:", end,Data['VisitInLastMonth'].mean())

**Tenure**
```

```
print(start, "Unique Values in column Tenure before Imputation:",
end,'\n',df['Tenure'].unique())
print(start, "Total Unique Values in column Tenure before Imputation:",
end,df['Tenure'].nunique())

print(start, "Total Null Values in Tenure Column:",
end,df['Tenure'].isnull().sum())

print(start, "Unique Values in column Tenure after KNN:",
end,'\n',df_knn['Tenure'].unique())
print(start, "Total Unique Values in column Tenure after KNN:",
end,df_knn['Tenure'].nunique())

#Imputing the values of Tenure that found through KNN
Data['Tenure']=df_knn['Tenure'].values

print(start, "Total Unique Values in column Tenure after KNN and rounding
it off:", end,Data['Tenure'].nunique())
print(start, "Total Null Values in Tenure Column after Imputation:",
end,Data['Tenure'].isnull().sum())
print(start, "Mean of Tenure Column after Imputation:",
end,Data['Tenure'].mean())

#Checking if there is still Null Values after Imputation :
print(start, "Total Null Values in dataset after Imputation:",
end,Data.isnull().sum().sum())

#Checking correlation of different columns through Heatmap
corr=df.corr()
plt.subplots(figsize=(20,15))
sns.heatmap(corr, annot=True)

#Barplot between Age and Profitability to check the distribution
plt.figure(figsize=(20,4))
sns.barplot(x='Age', y='Profitability', data=Data)
plt.xticks(rotation=90)

#Checking distribution of all the categorical columns with Profitability
for i in Data.drop('CustomerID',
axis=1).select_dtypes(['int64']).columns:
    plt.figure(figsize=(8,4))
    sns.barplot(x=i, y='Profitability', data=Data)
    plt.show()

print('================================================================
==================================')

# Checking the new dataset, adding new columns and Outlier Treatment
```

```
Data.head()

Data.info()

Observations:
1. No Null Values
2. 8 Columns having data type as int64, 3 as object and 9 as float64

# Adding new variables

**Variable 1 : 'ResponseToPromotions' as per the following conditions:**

condition1=(Data['TakenFreeDelivery']          ==          1)          &
(Data['FootfallInDiscountPeriod'] == 1) & (Data['TakenProdFromPromo'] ==
1)

condition2=(Data['TakenFreeDelivery']          ==          0)          &
(Data['FootfallInDiscountPeriod'] == 0) & (Data['TakenProdFromPromo'] ==
0)

condition3= (~condition1) & (~condition2)

Data.loc[condition1,'ResponseToPromotions'] = 'High'

Data.loc[condition2,'ResponseToPromotions'] = 'Low'

Data.loc[condition3,'ResponseToPromotions'] = 'Medium'

Data.head(10)

Data['ResponseToPromotions'].value_counts()

sns.barplot(x='ResponseToPromotions', y='Profitability', data=Data)

**Variable 2 : 'RFM_label' as per the following conditions:**

cols                                                                    =
['AvgBillAmount','VisitInLastMonth','AvgNumberOfProduct','CustomerID']

df2 = pd.DataFrame()
for col in cols:
    df2[col] = Data[col]

df2.head()

df2.info()

df2=df2.rename(columns={"AvgBillAmount":                    "monetary",
"VisitInLastMonth": "frequency","AvgNumberOfProduct": "recency"})
```

```
df2.head()

df2['r_quartile'] = pd.qcut(df2['recency'], 3, ['3','2','1'])
df2['f_quartile'] = pd.qcut(df2['frequency'], 3, ['3','2','1'])
df2['m_quartile'] = pd.qcut(df2['monetary'], 3, ['3','2','1'])

df2.head()

df2['RFM_Score']              =              df2.r_quartile.astype(str)+
df2.f_quartile.astype(str) + df2.m_quartile.astype(str)
df2.head()

df2.RFM_Score.value_counts().sort_index()

Process to label categories for RFM score:

Category A - all RFM with two or more 1s and all RFM with one 1 but rest
two are 2s
Category B - all RFM with two or more 3s and all RFM with one 3 but rest
two are 2s
Category C - all others

#     df["CITY"].replace(to_replace={"D.DO",     "DOLLARD",     "DDO"},
value="DOLLARD-DES-ORMEAUX", regex=True)
df2['RFM_label'] = df2.RFM_Score
df2['RFM_label'].replace(['111','112','113','121','122','131','211','2
12','221'], 'Most_Imp', inplace=True)
df2['RFM_label'].replace(['133','223','232','233','313','322','323','3
32','333'],'Least_Imp', inplace=True)
df2['RFM_label'].replace(['123','132','213','222','231','312'],
'Med_Imp', inplace=True)

df2.RFM_label.value_counts()

df2.head()

df = pd.concat([df,df2['RFM_label']],axis=1)
df.head()

df2.RFM_Score.value_counts().sort_index()

Process to label categories for RFM score:

Category A - all RFM with two or more 1s and all RFM with one 1 but rest
two are 2s
Category B - all RFM with two or more 3s and all RFM with one 3 but rest
two are 2s
Category C - all others
```

```
#     df["CITY"].replace(to_replace={"D.DO",     "DOLLARD",     "DDO"},
value="DOLLARD-DES-ORMEAUX", regex=True)
df2['RFM_label'] = df2.RFM_Score
df2['RFM_label'].replace(['111','112','113','121','122','131','211','2
12','221'], 'Most_Imp', inplace=True)
df2['RFM_label'].replace(['133','223','232','233','313','322','323','3
32','333'],'Least_Imp', inplace=True)
df2['RFM_label'].replace(['123','132','213','222','231','312'],
'Med_Imp', inplace=True)

df2.RFM_label.value_counts()

df2.head()

Data = pd.concat([Data,df2['RFM_label']],axis=1)
Data.head()



# Outlier Treatment

print(start,"Checking Outliers with the help of Boxplot",end)
fig, axes = plt.subplots(nrows=4,ncols=2)
fig.set_size_inches(18,16)

a = sns.boxplot(df['Tenure'] , ax=axes[0][0])
a.set_title("Tenure of Customers",fontsize=10)

a = sns.boxplot(df['HighestSaving'] , ax=axes[0][1])
a.set_title("HighestSaving Per Visit",fontsize=10)

a = sns.boxplot(df['Age']  , ax=axes[1][0])
a.set_title("Age of customer",fontsize=10)

a = sns.boxplot(df['AvgSaving'] , ax=axes[1][1])
a.set_title("Average saving in last one year",fontsize=10)

a = sns.boxplot(df['VisitInLastMonth'], ax=axes[2][0])
a.set_title("Total visit by customer in last month",fontsize=10)

a = sns.boxplot(df['AvgNumberOfProduct'], ax=axes[2][1])
a.set_title("Average number of product in one visit",fontsize=10)

a = sns.boxplot(df['AvgBillAmount'], ax=axes[3][0])
a.set_title("Average bill amount in one visit",fontsize=10)

a = sns.boxplot(df['SavingPercent'], ax=axes[3][1])
```

```
a.set_title("Average Saving Percent from total Average Bill
Amount",fontsize=10)

plt.tight_layout()
plt.show()

print(start,"Calculating Number of Outliers in each column with the help
of formula : (Q1-1.5*IQR) and (Q3+1.5*IQR) ",end)
Q1 = Data.quantile(0.25)
Q3 = Data.quantile(0.75)
IQR = Q3 - Q1
A=((Data < (Q1 - 1.5 * IQR)) | (Data > (Q3 + 1.5 * IQR))).sum()
B=A[A!=0].sort_values(ascending=False)
print(B)

#Checking names of all the columns:
Data.columns

#Scaling columns which have outliers
Data_scaled=Data.drop(['CustomerID','TakenFreeDelivery','CityTier','Pr
eferredPaymentMode','Gender','FootfallInDiscountPeriod','TakenProdFrom
Promo','MembershipCard','Complain','ResponseToPromotions','Profitabili
ty','Age','SatisfactionScore','RFM_label'],axis=1)

Data_scaled.head(10)

# Scaling Data and creating new dataframe for the Same

from scipy.stats import zscore
Data_scaled=Data_scaled.apply(zscore)
Data_scaled.head()

#Creating boxplot of scaled columns
Data_scaled.boxplot(figsize=(20,3))

# detecting no. of outlier based on 3*std deviation method

def detect_outlier(data_1):
    outliers=[]
    threshold=3.0
    mean_1 = np.mean(data_1)
    std_1 =np.std(data_1)


    for y in data_1:
        z_score= (y - mean_1)/std_1
        if np.abs(z_score) > threshold:
            outliers.append(y)
```

```
    return outliers

for i in Data_scaled:
    print (i,':', len(detect_outlier(Data_scaled[i])))

for i in Data_scaled:
    a=Data_scaled[i].dtype
    if a=='float64':
        Q1 = Data_scaled[i].quantile(0.25)
        Q3 = Data_scaled[i].quantile(0.75)
        IQR = Q3 - Q1
        A=Q3 + 1.5 * IQR
        print("Outlier for",i,"is",A)

for i in Data_scaled:
    print (i, 'max value:', Data_scaled[i].max())

Capping the outliers at upper bound

Data_scaled['SavingPercent']=np.where(Data_scaled['SavingPercent']>2.8
95,2.89,Data_scaled['SavingPercent'])

Data_scaled['HighestSaving']=np.where(Data_scaled['HighestSaving']>3,3
,Data_scaled['HighestSaving'])

Data_scaled['AvgBillAmount']=np.where(Data_scaled['AvgBillAmount']>2.8
313,2.8313,Data_scaled['AvgBillAmount'])

Data_scaled['AvgSaving']=np.where(Data_scaled['AvgSaving']>2.644,2.643
,Data_scaled['AvgSaving'])

Data_scaled['LastMonthCouponUsed']=np.where(Data_scaled['LastMonthCoup
onUsed']>0.8345,0.834,Data_scaled['LastMonthCouponUsed'])

Data_scaled['VisitInLastMonth']=np.where(Data_scaled['VisitInLastMonth
']>0.5764,0.576,Data_scaled['VisitInLastMonth'])

Data_scaled['AvgNumberOfProduct']=np.where(Data_scaled['AvgNumberOfPro
duct']>1.938,1.938,Data_scaled['AvgNumberOfProduct'])

Data_scaled['Tenure']=np.where(Data_scaled['Tenure']>1.0976,1.0976,Dat
a_scaled['Tenure'])

for i in Data_scaled:
    print (i, 'max value:', Data_scaled[i].max())

for i in Data_scaled:
    print (i, 'min value:', Data_scaled[i].min())
```

```
Data_scaled.rename(columns                                                =
{'Tenure':'Tenure_S','HighestSaving':'HighestSaving_S','SavingPercent'
:'SavingPercent_S','AvgSaving':'AvgSaving_S',

'LastMonthCouponUsed':'LastMonthCouponUsed_S','VisitInLastMonth':'Visi
tInLastMonth_S','AvgNumberOfProduct':'AvgNumberOfProduct_S','AvgBillAm
ount':'AvgBillAmount_S'}, inplace = True)

Data_scaled.columns

Data_scaled.boxplot(figsize=(20,3))

# Merging scaled columns to Data

Data['Tenure_S']= Data_scaled['Tenure_S']
Data['HighestSaving_S']= Data_scaled['HighestSaving_S']
Data['SavingPercent_S']= Data_scaled['SavingPercent_S']
Data['AvgSaving_S']= Data_scaled['AvgSaving_S']
Data['LastMonthCouponUsed_S']= Data_scaled['LastMonthCouponUsed_S']
Data['VisitInLastMonth_S']= Data_scaled['VisitInLastMonth_S']
Data['AvgNumberOfProduct_S']= Data_scaled['AvgNumberOfProduct_S']
Data['AvgBillAmount_S']= Data_scaled['AvgBillAmount_S']

Data.head(2)

Data.columns

# Binning on Original Data

**Tenure**

fig, axes = plt.subplots(ncols=2)
fig.set_size_inches(15,5)


a = sns.boxplot(Data['Tenure'] , ax=axes[0])
a.set_title("Outliers",fontsize=10)

a = sns.distplot(Data['Tenure'], ax=axes[1])
a.set_title("Distribution Of Tenure",fontsize=10)

for i in range(0,110,10):
    print(i,np.percentile(Data["Tenure"],i))

for i in range(89,101):
    print(i,np.percentile(Data["Tenure"],i))

bin_labels_5 = ['1', '2', '3', '4']
Data['Tenure_Bin'] = pd.qcut(Data['Tenure'],
```

```
                                q=[0,0.23,0.687, .9146, 1],
                                labels=bin_labels_5)
Data.head()

Data['Tenure_Bin'].value_counts()

sns.barplot(x='Tenure_Bin', y='Tenure', data=Data)

Data['Tenure_Bin']=Data['Tenure_Bin'].astype(int)

for i in np.arange(1,5):
    max = Data.Tenure[Data.Tenure_Bin==i].max()
    min = Data.Tenure[Data.Tenure_Bin==i].min()
    print("Max in bin",i, max)
    print("Min in bin",i, min)

sns.boxplot(x='Tenure_Bin',data=Data)

**LastMonthCouponUsed**

fig, axes = plt.subplots(ncols=2)
fig.set_size_inches(15,5)


a = sns.boxplot(Data['LastMonthCouponUsed'] , ax=axes[0])
a.set_title("Outliers",fontsize=10)

a = sns.distplot(Data['LastMonthCouponUsed'], ax=axes[1])
a.set_title("Distribution Of LastMonthCouponUsed",fontsize=10)

for i in range(0,110,10):
    print(i,np.percentile(Data["LastMonthCouponUsed"],i))

for i in range(89,101):
    print(i,np.percentile(Data["LastMonthCouponUsed"],i))

bin_labels_V = ['1', '2', '3']
Data['LastMonthCouponUsed_Bin'] = pd.qcut(Data['LastMonthCouponUsed'],
                                q=[0,.57,.93, 1],
                                labels=bin_labels_V)
Data.head()

Data['LastMonthCouponUsed_Bin'].value_counts()

Data['LastMonthCouponUsed_Bin']=Data['LastMonthCouponUsed_Bin'].astype
(int)

for i in np.arange(1,4):
```

```
    max1                                                                =
Data.LastMonthCouponUsed[Data.LastMonthCouponUsed_Bin==i].max()
    min1                                                                =
Data.LastMonthCouponUsed[Data.LastMonthCouponUsed_Bin==i].min()
    print("Max in bin",i, max1)
    print("Min in bin",i, min1)

sns.barplot(x='LastMonthCouponUsed_Bin',       y='LastMonthCouponUsed',
data=Data)

sns.boxplot(x='LastMonthCouponUsed_Bin',data=Data)

**VisitInLastMonth**

fig, axes = plt.subplots(ncols=2)
fig.set_size_inches(15,5)


a = sns.boxplot(Data['VisitInLastMonth'] , ax=axes[0])
a.set_title("Outliers",fontsize=10)

a = sns.distplot(Data['VisitInLastMonth'], ax=axes[1])
a.set_title("Distribution Of VisitInLastMonth",fontsize=10)

for i in range(0,110,10):
    print(i,np.percentile(Data["VisitInLastMonth"],i))

for i in range(89,101):
    print(i,np.percentile(Data["VisitInLastMonth"],i))

bin_labels_3 = ['1', '2', '3']
Data['VisitInLastMonth_Bin'] = pd.qcut(Data['VisitInLastMonth'],
                              q=[0,.62,.92, 1],
                              labels=bin_labels_3)
Data.head()

Data['VisitInLastMonth_Bin'].value_counts()

Data['VisitInLastMonth_Bin']=Data['VisitInLastMonth_Bin'].astype(int)

for i in np.arange(1,4):
    max1 = Data.VisitInLastMonth[Data.VisitInLastMonth_Bin==i].max()
    min1 = Data.VisitInLastMonth[Data.VisitInLastMonth_Bin==i].min()
    print("Max in bin",i, max1)
    print("Min in bin",i, min1)

sns.barplot(x='VisitInLastMonth_Bin', y='VisitInLastMonth', data=Data)

sns.boxplot(x='VisitInLastMonth_Bin',data=Data)
```

```
# Outlier Treatment with logarithmic transformation

**AvgNumberOfProduct**

#AvgNumberOfProduct variable transformation
Data['AvgNumberOfProduct_log']=Data["AvgNumberOfProduct"].map(lambda i:
np.log(i) if i > 0 else 0)
print(Data['AvgNumberOfProduct_log'].skew())
print(Data['AvgNumberOfProduct'].skew())

Q1, Q3 = np.percentile(Data['AvgNumberOfProduct_log'],[25,75])
IQR = Q3-Q1

# check out upper and lower outlier values through the standard formula
lower outlier value <Q1-1.5(IQR)
low_bound = Q1-1.5*IQR
# check upper outlier > Q3+1.5(IQR)
upp_bound = Q3+1.5*IQR


print('IQR = {},\nQ3 = {},\nQ1 = {}, \nlow_outlier = {}, \nupp_outlier
= {}'.format(IQR,Q3,Q1,low_bound,upp_bound))
print('\nAnything above',upp_bound, 'in the AvgNumberOfProduct_log is an
outlier, and we do have upper outliers in dataset')

print(start,"Number of Outliers on Lower
bound",end,Data[Data.AvgNumberOfProduct_log<0.1368]['AvgNumberOfProduc
t_log'].count())
print(start,"Number of Outliers on Upper
bound",end,Data[Data.AvgNumberOfProduct_log>4.6374]['AvgNumberOfProduc
t_log'].count())
print('\n')
print(start,"Boxplot for graphical representation :",end)
sns.boxplot(x='AvgNumberOfProduct_log',data=Data)

**SavingPercent**

#SavingPercent variable transformation
Data['SavingPercent_log']=Data["SavingPercent"].map(lambda i: np.log(i)
if i > 0 else 0)
print(Data['SavingPercent_log'].skew())
print(Data['SavingPercent'].skew())

Q1, Q3 = np.percentile(Data['SavingPercent_log'],[25,75])
IQR = Q3-Q1

# check out upper and lower outlier values through the standard formula
lower outlier value <Q1-1.5(IQR)
```

```
low_bound = Q1-1.5*IQR
# check upper outlier > Q3+1.5(IQR)
upp_bound = Q3+1.5*IQR


print('IQR = {},\nQ3 = {},\nQ1 = {}, \nlow_outlier = {}, \nupp_outlier
= {}'.format(IQR,Q3,Q1,low_bound,upp_bound))
print('\nAnything above',upp_bound, 'in the SavingPercent_log is an
outlier, and we do have upper outliers in dataset')

print(start,"Number of Outliers on Lower
bound",end,Data[Data.SavingPercent_log<-
1.5026]['SavingPercent_log'].count())
print(start,"Number of Outliers on Upper
bound",end,Data[Data.SavingPercent_log>2.3677]['SavingPercent_log'].co
unt())
print('\n')
print(start,"Boxplot for graphical representation :",end)
sns.boxplot(x='SavingPercent_log',data=Data)

**Tenure**

#Tenure variable transformation
Data['Tenure_log']=Data["Tenure"].map(lambda i: np.log(i) if i > 0 else
0)
print(Data['Tenure_log'].skew())
print(Data['Tenure'].skew())

Q1, Q3 = np.percentile(Data['Tenure_log'],[25,75])
IQR = Q3-Q1

# check out upper and lower outlier values through the standard formula
lower outlier value <Q1-1.5(IQR)
low_bound = Q1-1.5*IQR
# check upper outlier > Q3+1.5(IQR)
upp_bound = Q3+1.5*IQR


print('IQR = {},\nQ3 = {},\nQ1 = {}, \nlow_outlier = {}, \nupp_outlier
= {}'.format(IQR,Q3,Q1,low_bound,upp_bound))
print('\nAnything above',upp_bound, 'in the Tenure_log is an outlier,
and we do have upper outliers in dataset')

print(start,"Number of Outliers on Lower
bound",end,Data[Data.Tenure_log<-1.3382]['Tenure_log'].count())
print(start,"Number of Outliers on Upper
bound",end,Data[Data.Tenure_log>2.2303]['Tenure_log'].count())
print('\n')
print(start,"Boxplot for graphical representation :",end)
```

```
sns.boxplot(x='Tenure_log',data=Data)

# Capping on Original Data for columns: AvgNumberOfProduct,
AvgBillAmount, AvgSaving

#Doing transformation at the end as used the original columns for
different methods above

**Checking tolerable value for outlier**

for i in Data:
    a=Data[i].dtype
    if a=='float64':
        Q1 = Data[i].quantile(0.25)
        Q3 = Data[i].quantile(0.75)
        IQR = Q3 - Q1
        A=Q3 + 1.5 * IQR
        print("Outlier for",i,"is",A)

**AvgNumberOfProduct**

fig, axes = plt.subplots(ncols=2)
fig.set_size_inches(15,5)


a = sns.boxplot(Data['AvgNumberOfProduct'] , ax=axes[0])
a.set_title("Outliers",fontsize=10)

a = sns.distplot(Data['AvgNumberOfProduct'], ax=axes[1])
a.set_title("Distribution Of AvgNumberOfProduct",fontsize=10)

for i in range(0,110,10):
    print(i,np.percentile(Data["AvgNumberOfProduct"],i))

for i in range(89,101):
    print(i,np.percentile(Data["AvgNumberOfProduct"],i))

Outlier_N=Data[Data["AvgNumberOfProduct"]>38.45]
print(start,'Total Number of Outliers
:',end,Outlier_N["AvgNumberOfProduct"].count())

Outlier_N99=Data[Data["AvgNumberOfProduct"]>57.826]
print(start,'Total Number of Outliers at 99th percentile
:',end,Outlier_N99["AvgNumberOfProduct"].count())

Data['AvgNumberOfProduct'].skew()

Using Upper Bound of outlier for capping as AvgNumberOfProduct is highly
skewed. Outlier value is 38.45.
```

```
# Capping the outliers of AvgNumberOfProduct to upper bound
Data['AvgNumberOfProduct']=np.where(Data['AvgNumberOfProduct']>38.45,3
8.45,Data['AvgNumberOfProduct'])

sns.boxplot(x='AvgNumberOfProduct',data=Data)

**AvgBillAmount**

fig, axes = plt.subplots(ncols=2)
fig.set_size_inches(15,5)


a = sns.boxplot(Data['AvgBillAmount'] , ax=axes[0])
a.set_title("Outliers",fontsize=10)

a = sns.distplot(Data['AvgBillAmount'], ax=axes[1])
a.set_title("Distribution Of AvgBillAmount",fontsize=10)

for i in range(0,110,10):
    print(i,np.percentile(Data["AvgBillAmount"],i))

for i in range(89,101):
    print(i,np.percentile(Data["AvgBillAmount"],i))

Outlier_NAB=Data[Data["AvgBillAmount"]>20515.81662]
print(start,'Total            Number            of            Outliers
:',end,Outlier_NAB["AvgBillAmount"].count())

As   maximum   tolerable   value   while   calculating   outlier   is
20515.816620094934 and 99th percentile is 19220.74074. So will cap the
outlier with the tolerable values.

# Capping the outliers of AvgBillAmount to tolerable upper bound value
on extremities
Data['AvgBillAmount']=np.where(Data['AvgBillAmount']>20515.81,20515.81
,Data['AvgBillAmount'])

sns.boxplot(x='AvgBillAmount',data=Data)

**AvgSaving**

fig, axes = plt.subplots(ncols=2)
fig.set_size_inches(15,5)


a = sns.boxplot(Data['AvgSaving'] , ax=axes[0])
a.set_title("Outliers",fontsize=10)
```

```
a = sns.distplot(Data['AvgSaving'], ax=axes[1])
a.set_title("Distribution Of AvgSaving",fontsize=10)

for i in range(0,110,10):
    print(i,np.percentile(Data["AvgSaving"],i))

for i in range(89,101):
    print(i,np.percentile(Data["AvgSaving"],i))

Outlier_NAS=Data[Data["AvgSaving"]>386.16]
print(start,'Total Number of Outliers
:',end,Outlier_NAS["AvgSaving"].count())

Outlier_N99_AS=Data[Data["AvgSaving"]>394.02]
print(start,'Total Number of Outliers at 99th percentile
:',end,Outlier_N99_AS["AvgSaving"].count())

Data['AvgSaving'].skew()

Using Upper Bound of outlier for capping as AvgSaving is skewed. Outlier
value is 386.16

# Capping the outliers of AvgSaving to tolerable Upper Bound Value on
extremities
Data['AvgSaving']=np.where(Data['AvgSaving']>386.16,386.159,Data['AvgS
aving'])

sns.boxplot(x='AvgSaving',data=Data)

Data.head()

Data.to_csv('Retail_Final.csv',index=False)

Data Exported
```

File2: EDA.ipynb

```
## Imports dataset and libraries

# Import all libraries

from matplotlib import pyplot as plt
import matplotlib.cm as cm
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler
%matplotlib inline
```

```
# Import dataset and read file

Data = pd.read_csv('Retail_Final.csv')

## Study the dataset in detail

### Overview the dataframe
1. Check the head and tail of dataframe, if it is loaded correctly
2. Look at the columns and the type of values mentioned
3. Check the dimensions of the dataframe i.e. no of rows and columns

# display the head and check dataset

Data.head()

Data.tail()

# Check the dimensions of Dataset

print('Dataset has {} rows and {}
columns'.format(Data.shape[0],Data.shape[1]))

### Check for duplication

# Check if there are any duplicated enetries in the dataset

print('The dataset has {} duplicate
entries'.format(Data.duplicated().sum()))

### Observe the columns and datatypes in detail

# list out all the column names

Data.columns.sort_values()

# Check the information of each column/variable like no of not-null
values and datatype of each variable

print('There are {} null values in the dataset as all the columns have
{} non-null count.\
\n\nThere are {} columns with following datatypes :\n{}\n\n'.

format(Data.isnull().sum().sum(),Data.shape[0],len(Data.columns),Data.
dtypes.value_counts()))

Data.info()

# Check the object columns and relate for the types of values and counts
```

```
for col in (Data.select_dtypes(['object'])).columns:
    print('Column          {}          has          {}          unique
values:\n{}'.format(col,len(Data[col].unique()),Data[col].value_counts
(dropna=False)))

print('============================================================\n')

### Five Point Summary

Data.describe().transpose()

Data.describe(include='object').transpose()

## Data Visualization

### Check distribution of each column

#### Using histograms

# use matplotlib for compiled histogram

hist = Data.hist(figsize=(20,20))

### Compare distribution of treated columns before and after treatment

# create a list for each variable on the basis of treatment during data
pre-processing

col_no_change  =  ['Age',  'TakenFreeDelivery',  'TakenProdFromPromo',
'ResponseToPromotions',
                   'CityTier',                'PreferredPaymentMode',
'Gender','SatisfactionScore',              'FootfallInDiscountPeriod',
'MembershipCard', 'Complain', 'Profitability','RFM_label']

all_HighestSaving = ['HighestSaving', 'HighestSaving_S']

all_Tenure = ['Tenure', 'Tenure_Bin', 'Tenure_S', 'Tenure_log']

all_SavingPercent      =      ['SavingPercent',      'SavingPercent_S',
'SavingPercent_log']

all_AvgSaving = ['AvgSaving', 'AvgSaving_S']

all_LastMonthCouponUsed           =           ['LastMonthCouponUsed',
'LastMonthCouponUsed_Bin', 'LastMonthCouponUsed_S']

all_AvgNumberOfProduct = ['AvgNumberOfProduct', 'AvgNumberOfProduct_S',
'AvgNumberOfProduct_log']
```

```
all_AvgBillAmount = ['AvgBillAmount', 'AvgBillAmount_S']

all_VisitInLastMonth                =                ['VisitInLastMonth',
'VisitInLastMonth_Bin','VisitInLastMonth_S']

# Compartive distribution of all variables before and after treatment
using distplot
list_var       =       [all_AvgBillAmount,       all_AvgNumberOfProduct,
all_AvgSaving,all_LastMonthCouponUsed,
            all_HighestSaving,      all_SavingPercent,      all_Tenure,
all_VisitInLastMonth]

for var in list_var:
    fig,axes = plt.subplots(1,len(var),figsize=(15,4))
    for item in var:
        hist = sns.distplot(Data[item], ax=axes[var.index(item)])

#### Using Box-plots

# Compartive distribution of all variables brfore and after treatment
using box-plot
list_var        =        [all_AvgBillAmount,        all_HighestSaving,
all_AvgNumberOfProduct,
            all_AvgSaving,        all_SavingPercent,        all_Tenure,
all_VisitInLastMonth]

for var in list_var:
    fig,axes = plt.subplots(1,len(var),figsize=(15,4))
    for item in var:
        hist = sns.boxplot(Data[item], ax=axes[var.index(item)])

### Relation of IV with Profitability

#### Using Bar-plots

#Checking distribution of all the categorical columns with Profitability

for              i              in              Data.drop('CustomerID',
axis=1).select_dtypes(exclude='float').columns.sort_values():
#     plt.figure(figsize=(10,4))
    sns.barplot(x=i, y='Profitability', data=Data, )
#     plt.bar(i,Data['Profitability'],width=0.3)
    plt.show()

print('================================================================
====================================')

cat=Data.drop('CustomerID', axis=1).select_dtypes(exclude='float')
```

```
# cat=cat.to_list()
cat['VisitInLastMonth']=Data['VisitInLastMonth']

cat3=cat.columns
type(cat3)
cat3=cat3.to_list()

cat3

Data.loc[:,'FootfallInDiscountPeriod']

#Checking distribution of all the categorical columns with Profitability
i=0
while i < len(cat3):
    fig, axes = plt.subplots(ncols=4,figsize=(30,5))
    a=sns.barplot(x=Data.loc[:,cat3[i]],     y=Data["Profitability"],ax
=axes[0])
    a=sns.barplot(x=Data.loc[:,cat3[i+1]],   y=Data["Profitability"],ax
=axes[1])
    a=sns.barplot(x=Data.loc[:,cat3[i+2]],   y=Data["Profitability"],ax
=axes[2])
    a=sns.barplot(x=Data.loc[:,cat3[i+3]],   y=Data["Profitability"],ax
=axes[3])
    i=i+4

#### Using scatterplot

list_var        =        [all_AvgBillAmount,        all_HighestSaving,
all_AvgNumberOfProduct,
            all_AvgSaving,        all_SavingPercent,        all_Tenure,
all_VisitInLastMonth]

for var in list_var:
    fig,axes = plt.subplots(1,len(var),figsize=(15,4))
    for item in var:
        regp   =   sns.regplot(x=item,   y='Profitability',   data=Data,
lowess=True ,
                           ax=axes[var.index(item)],
scatter_kws={'alpha':0.15}, line_kws={'color': 'red'})

 for       i       in       Data.drop(['CustomerID','Profitability'],
axis=1).select_dtypes(include='float').columns.sort_values():
    plt.figure(figsize=(10,4))
    sns.scatterplot(x=i, y='Profitability', data=Data)
    plt.show()

print('================================================================
====================================')
```

```
## Correlation between variables

#### Using Correaltion metrics

Data.corr()

#### using Heat map

corr=Data.corr()
plt.subplots(figsize=(20,15))
sns.heatmap(corr, annot=True)

## Impact of different parameters on linear relation with probablility

### Through scatter-plot

list_var        =        [all_AvgBillAmount,        all_HighestSaving,
all_AvgNumberOfProduct,all_AvgSaving,
            all_SavingPercent, all_Tenure, all_VisitInLastMonth]

for var in list_var:
    dicreet_par                                                    =
Data.drop('CustomerID',axis=1).select_dtypes(exclude='float').columns
    for par in dicreet_par:
        fig,axes = plt.subplots(1,len(var),figsize=(15,4))
        for item in var:
            scatp    =    sns.scatterplot(x=item,    y='Profitability',
data=Data, ax=axes[var.index(item)], alpha=0.3, hue=par)

for i in (Data.select_dtypes(['float64'])).columns:
    ax = sns.regplot(x=i,y='Profitability', data=Data )
    plt.show()
    print(ax)

### Through Barplots

plt.figure(figsize=(15,5))
sns.barplot(data=Data,x='Tenure_Bin',y='Profitability',hue='VisitInLas
tMonth_Bin')
plt.show()
```

File3: Feature Selection and Mode Evaluation.ipynb

```
# Import libraries , import file and understand data

# Import libraries
import pandas as pd
import numpy as np
```

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# read data
df = pd.read_csv('Retail_Final.csv')
df.head()

# check info
df.info()

# CustomerId as it has no value to the model building
df.drop('CustomerID', axis=1, inplace=True)

# Visualize data

# check for outliers
plt.figure(figsize=(18,5))
plot = df.boxplot()
plt.xticks(rotation=90)

# There are 4 object variables which we need to convert to int type
# lets ananlyse and treat them one by one

df.select_dtypes("object")

# check the first object variable
print('Category\t   count')
print(df.PreferredPaymentMode.value_counts())     # Check the different
categories in variable

sns.barplot(x='PreferredPaymentMode', y='Profitability', data=df)     #
plot the profitability against each category

As we may notice average profitability does not vary too much with any
of the mode of payment.

# check the second object variable
print('Category  count')
print(df.Gender.value_counts())     # Check the different categories in
variable

sns.barplot(x='Gender',  y='Profitability',  data=df)       #  plot  the
profitability against each category

As we may notice average profitability does not vary too much with Gender
type

# check the third object variable
```

```
print('Category  count')
print(df.MembershipCard.value_counts())         #  Check  the  different
categories in variable

sns.barplot(x='MembershipCard', y='Profitability', data=df)     # plot
the profitability against each category

As we may notice average profitability does not vary too much with any
of the type of membership.

# check the fourth object variable
print('Category  count')
print(df.ResponseToPromotions.value_counts())     # Check the different
categories in variable

sns.barplot(x='ResponseToPromotions', y='Profitability', data=df)     #
plot the profitability against each category

As we may notice average profitability does not vary too much with any
of the type of response to promotion

# check the fifth object variable
print('Category     count')
print(df.RFM_label.value_counts())     # Check the different categories
in variable

sns.barplot(x='RFM_label', y='Profitability', data=df)     # plot the
profitability against each category

# check the fifth object variable
print('Cat count')
print(df.SatisfactionScore.value_counts())       # Check the different
categories in variable

sns.barplot(x='SatisfactionScore', y='Profitability', data=df)    # plot
the profitability against each category

As we may notice average profitability shows a positive relationship
with RFM_label. Average profitability increases with Most_imp label and
viceversa.

# Pre-process dataset

df.columns

df['ResponseToPromotions'].unique()

df['RFM_label']=df["RFM_label"].replace(['Most_Imp','Med_Imp','Least_I
mp'],[3,2,1])
```

```
df['ResponseToPromotions']=df['ResponseToPromotions'].replace(['High',
'Medium','Low'],[3,2,1])

# Convert the object variables to numeric type with one-hot encoding

df = pd.get_dummies(df , drop_first=True)

df.info()

df.describe().transpose()

# plot boxplot again to check outliers in new data

plt.figure(figsize=(18,5))
plot = df.boxplot()
plt.xticks(rotation=90)

# check for null values

df.isnull().sum().sum()

# plt heatmap to check collinearity

plt.figure(figsize=(18,18)
          )
map = sns.heatmap(df.corr(), annot=True)

As collinearity is highly evident between several variables, PCA can be
a good option to reduce multicollinearity and number of variables both

df.info()

X = df.drop('Profitability', axis=1)
y = df.Profitability

# As the data is at different scale , we need to scale the data
from scipy.stats import zscore
X_scaled = X.apply(zscore)
X_scaled.head()

# VIF for feature selection

from            statsmodels.stats.outliers_influence            import
variance_inflation_factor

df[['RFM_label','ResponseToPromotions']].tail()

# start with all features
```

```
X = df.drop('Profitability',axis=1)
y = df['Profitability']

# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 41:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

vif_df.sort_values(by='feature')


X = df.drop(['Profitability','HighestSaving'],axis=1)
y = df['Profitability']

# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X = df.drop(['Profitability','HighestSaving','AvgBillAmount'],axis=1)
y = df['Profitability']

# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X                                                                    =
df.drop(['Profitability','HighestSaving','AvgBillAmount','AvgSaving'],
axis=1)
y = df['Profitability']
```

```
# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X                                                                    =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent'],axis=1)
y = df['Profitability']

# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X                                                                      =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct'],axis=1)
y = df['Profitability']

# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X                                                                    =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',
            'ResponseToPromotions'],axis=1)
y = df['Profitability']
```

```
# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X                                                                    =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',
            'ResponseToPromotions','Tenure_Bin'],axis=1)
y = df['Profitability']


# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X                                                                     =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'ResponseToPromotions','Tenure_Bin','AvgNumberOfProduct_log'],axis=1)
y = df['Profitability']


# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))
```

```
X                                                                    =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'ResponseToPromotions','Tenure_Bin','AvgNumberOfProduct_log','LastMont
hCouponUsed'],axis=1)
y = df['Profitability']


# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X                                                                    =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'ResponseToPromotions','Tenure_Bin','AvgNumberOfProduct_log','LastMont
hCouponUsed',
             'VisitInLastMonth_Bin'],axis=1)
y = df['Profitability']


# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X                                                                    =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'ResponseToPromotions','Tenure_Bin','AvgNumberOfProduct_log','LastMont
hCouponUsed','VisitInLastMonth_Bin',
            'LastMonthCouponUsed_Bin'],axis=1)
y = df['Profitability']
```

```
# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X                                                                    =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'ResponseToPromotions','Tenure_Bin','AvgNumberOfProduct_log','LastMont
hCouponUsed','VisitInLastMonth_Bin',
            'LastMonthCouponUsed_Bin','Tenure'],axis=1)
y = df['Profitability']


# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X                                                                     =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'ResponseToPromotions','Tenure_Bin','AvgNumberOfProduct_log','LastMont
hCouponUsed','VisitInLastMonth_Bin',

'LastMonthCouponUsed_Bin','Tenure','VisitInLastMonth'],axis=1)
y = df['Profitability']


# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
```

```
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X                                                                    =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'ResponseToPromotions','Tenure_Bin','AvgNumberOfProduct_log','LastMont
hCouponUsed','VisitInLastMonth_Bin',

'LastMonthCouponUsed_Bin','Tenure','VisitInLastMonth','RFM_label'],axi
s=1)
y = df['Profitability']


# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

X                                                                    =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'ResponseToPromotions','Tenure_Bin','AvgNumberOfProduct_log','LastMont
hCouponUsed','VisitInLastMonth_Bin',

'LastMonthCouponUsed_Bin','Tenure','VisitInLastMonth','RFM_label','Sav
ingPercent_log'],axis=1)
y = df['Profitability']


# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
```

```
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

# trial - since avg bill amount has no more representation in the list
, lets try to drop the second one response medium
X                                                                    =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'ResponseToPromotions','Tenure_Bin','AvgNumberOfProduct_log','LastMont
hCouponUsed','VisitInLastMonth_Bin',

'LastMonthCouponUsed_Bin','Tenure','VisitInLastMonth','RFM_label','Sav
ingPercent_log','Tenure_log'],axis=1)
y = df['Profitability']


# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

# we see not much effect on avg bill amount by dropping response variable,
means there is no correlation between the two

# trial - lets drop avg bill amount and see the effect

X                                                                    =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'ResponseToPromotions','Tenure_Bin','AvgNumberOfProduct_log','LastMont
hCouponUsed','VisitInLastMonth_Bin',

'LastMonthCouponUsed_Bin','Tenure','VisitInLastMonth','RFM_label','Sav
ingPercent_log','Tenure_log',
            'Age'],axis=1)
y = df['Profitability']

# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
```

```
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

# response is not affected but avg_saving has,means the two are
correlated


X                                                                =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'ResponseToPromotions','Tenure_Bin','AvgNumberOfProduct_log','LastMont
hCouponUsed','VisitInLastMonth_Bin',

'LastMonthCouponUsed_Bin','Tenure','VisitInLastMonth','RFM_label','Sav
ingPercent_log','Tenure_log',
            'Age','AvgBillAmount_S'],axis=1)
y = df['Profitability']


# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

# as we see response_medium is again not affected but avg_bill amount
has,means the avg bill and avg saving are correlated.

# we will procees as per vif score i.e. dropping avg bill amount and
Response to promotion medium instead of avg saving


X                                                                =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'ResponseToPromotions','Tenure_Bin','AvgNumberOfProduct_log','LastMont
hCouponUsed','VisitInLastMonth_Bin',
```

```
'LastMonthCouponUsed_Bin','Tenure','VisitInLastMonth','RFM_label','Sav
ingPercent_log','Tenure_log',
            'Age','AvgBillAmount_S','SatisfactionScore'],axis=1)
y = df['Profitability']


# calculate vif

vif    =    [variance_inflation_factor(X.values,    ix)    for    ix    in
range(X.shape[1])]
i=0
for column in X.columns:
    if i < 24:
        i = i+1
vif_df = pd.DataFrame.from_dict({'feature':X.columns,'vif':vif})
print (vif_df.sort_values(by='vif',ascending=False))

# here we may see both RFM_label and Satisfaction level are slightly
above 5, but from EDA we have a direct dependence
# of profitability on RFM_label, hence, we may try to drop satisfaction
score first and see if it effects RFM_level

vif_df.nlargest(20,'vif')

from sklearn.decomposition import PCA
# load data

# feature extraction
pca = PCA(n_components=17)
fit = pca.fit(X)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)

# SelectKBest for feature importance

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression

X = df.drop('Profitability',axis=1)
y = df['Profitability']

best_features = SelectKBest(score_func=f_regression,k=20)
fit = best_features.fit(X,y)

best_features                                                    =
pd.DataFrame.from_dict({'features':X.columns,"Score":fit.scores_})
best_features.sort_values(by='features')
```

```
# ExtraTreesRegressor for feature importance

from sklearn.ensemble import ExtraTreesRegressor
extra_tree = ExtraTreesRegressor()

extra_tree.fit(X,y)

extra_tree.feature_importances_

#     feature_imp     =     pd.Series(extra_tree.feature_importances_,
index=X.columns)
# feature_imp.nlargest(40)#.plot(kind='barh')

feature_imp                                                          =
pd.DataFrame.from_dict({'features':X.columns,"Imp":extra_tree.feature_
importances_})
feature_imp.sort_values(by='features')

# check dataset
df.head(2)

# Trying following models with selected features :

    XGBoost
    Random Forest Regressor
    Linear Regression
    Lassor Regressor

    All four models will be tested on train and test with 70:30 split
on:
    1.All variables - 40 no.s

    2.Selected variables as per final VIF list - 20 no

    3.Modified list with 19 final variable list and additional important
variables like Satisfaction score, last month coupon used etc

# import metrics for comparing various regression model

from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import r2_score
from math import sqrt

## Divide dataframe into X and y

# divide X and y with all features

X_all = df.drop('Profitability', axis=1)
```

```
y = df['Profitability']

# define X with selected features - 20 nos

X_selected                                                              =
df.drop(['Profitability','HighestSaving','AvgSaving','AvgBillAmount','
SavingPercent','AvgNumberOfProduct',

'Tenure_Bin','AvgNumberOfProduct_log','LastMonthCouponUsed','LastMonth
CouponUsed_Bin',

'VisitInLastMonth_Bin','Tenure','VisitInLastMonth','SavingPercent_log'
,'Tenure_log','Age',
            'AvgBillAmount_S','SatisfactionScore'],axis=1)
y = df['Profitability']

X_selected.head(2)

## Scale the X_all and X_selected, and define y

from scipy.stats import zscore
X_all = X_all.apply(zscore)              # scale X with all variables
before training
X_selected = X_selected.apply(zscore)  # scale X with selected features
before training
y = df['Profitability']

## Use  on X-G Boost model for training and testing the two datasets

# import xgboost

import xgboost as xgb

### Training X_all and y with XGBoost and calculate various evaluation
metrics

# convert the dataset into an optimized data structure called Dmatrix
that XGBoost supports for acclaimed performance
# and efficiency gains.

df_dmatrix = xgb.DMatrix(data=X_all,label=y)

# divide data with all features into test and train
from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X_all,   y,
test_size=0.3, random_state=123)

# cretaing instance for xgboost
```

```
xg_reg = xgb.XGBRegressor(objective ='reg:linear', n_estimators = 10,
seed = 123)

# fit on train data
xg_reg.fit(X_train,y_train)

# calculate error for train X and train y

pred_ytrain = xg_reg.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square         =         {},\n\tAdj         R-square         =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

# calculate error for test X and test y

pred_ytest = xg_reg.predict(X_test)    # predict from X test
mae_ts = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts        =        1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

### Training X_selected and y with XGBoost and calculate various
evaluation metrics

# convert the dataset into an optimized data structure called Dmatrix
that XGBoost supports for acclaimed performance
# and efficiency gains.

df_dmatrix = xgb.DMatrix(data=X_selected,label=y)
```

```
# divide data into test and train

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_selected, y,
test_size=0.3, random_state=123)

# cretaing instance for xgboost

xg_reg = xgb.XGBRegressor(objective ='reg:linear', n_estimators = 10,
seed = 123)

# fit on train data

xg_reg.fit(X_train,y_train)

#### Metrics for Train data

# calculate error for train X and train y

pred_ytrain = xg_reg.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square          =          {},\n\tAdj          R-square          =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

#### Metrics for Test data

# calculate error for test X and test y

pred_ytest = xg_reg.predict(X_test)    # predict from X test
mae_tr = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts       =       1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))
```

```
print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

#### Feature importance in the model

# import statsmodels.api as sm
# importance_split = sm.OLS(y, X_selected).fit()
# print(importance_split.params.sort_values(ascending=False))
xg_reg.feature_importances_

# print(xg_reg.coef_)
# print(xg_reg.intercept_)

# ananlyse importance features for XGBoost

feature_imp_xgb        =        pd.Series(xg_reg.feature_importances_,
index=X_selected.columns)
plt.figure(figsize=(6,6))
feature_imp_xgb.nlargest(21).plot(kind='barh')
plt.grid()

feature_imp_xgb                                                    =
pd.DataFrame.from_dict({'features':X_selected.columns,"Imp":xg_reg.fea
ture_importances_})
feature_imp_xgb.sort_values(by='Imp', ascending=False)

### Training X_mod and y with Random Forest Regressor and calculate
various evaluation metrics

X_mod                                                              =
df[['HighestSaving','AvgBillAmount','AvgNumberOfProduct_log','AvgSavin
g_S','RFM_label',

'VisitInLastMonth_Bin','SavingPercent','Tenure_log','Complain','CityTi
er','SatisfactionScore','Age',
'LastMonthCouponUsed_Bin','Gender_Male']]

# convert the dataset into an optimized data structure called Dmatrix
that XGBoost supports for acclaimed performance
# and efficiency gains.

df_dmatrix = xgb.DMatrix(data=X_mod,label=y)

# divide data into test and train

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_mod, y,
test_size=0.3, random_state=123)

# cretaing instance for xgboost

xg_reg = xgb.XGBRegressor(objective ='reg:linear', n_estimators = 10,
seed = 123)

# fit on train data

xg_reg.fit(X_train,y_train)

#### Metrics for Train data

# calculate error for train X and train y

pred_ytrain = xg_reg.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square         =         {},\n\tAdj         R-square         =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

#### Metrics for Test data

# calculate error for test X and test y

pred_ytest = xg_reg.predict(X_test)    # predict from X test
mae_tr = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts       =       1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))
```

```
#### Feature importance in the model

# import statsmodels.api as sm
# importance_split = sm.OLS(y, X_selected).fit()
# print(importance_split.params.sort_values(ascending=False))
xg_reg.feature_importances_

# print(xg_reg.coef_)
# print(xg_reg.intercept_)

# ananlyse importance features for XGBoost

feature_imp_xgb        =        pd.Series(xg_reg.feature_importances_,
index=X_mod.columns)
plt.figure(figsize=(6,6))
feature_imp_xgb.nlargest(21).plot(kind='barh')
plt.grid()

feature_imp_xgb                                                    =
pd.DataFrame.from_dict({'features':X_mod.columns,"Imp":xg_reg.feature_
importances_})
feature_imp_xgb.sort_values(by='Imp', ascending=False)

## Use Random Forest Regressor model for training and testing the two
datasets

# import Rf regressor

from sklearn.ensemble import RandomForestRegressor

### Training X_all and y with Random Forest Regressor and calculate
various evaluation metrics

# divide data with all features into test and train
from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X_all,   y,
test_size=0.3, random_state=123)

# creating instance for Rf regressor
RF_reg = RandomForestRegressor()

# fit on train data
RF_reg.fit(X_train, y_train)

# calculate error for train X and train y

pred_ytrain = RF_reg.predict(X_train)    # predict from X train
```

```
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square         =         {},\n\tAdj          R-square         =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

# calculate error for test X and test y

pred_ytest = RF_reg.predict(X_test)    # predict from X test
mae_tr = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts        =        1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

### Training X_selected and y with Random Forest Regressor and calculate
various evaluation metrics

# divide data with selected features into test and train
from sklearn.model_selection import train_test_split
X_train,  X_test,  y_train,  y_test  =  train_test_split(X_selected,  y,
test_size=0.3, random_state=123)

# cretaing instance for Rf regressor
RF_reg = RandomForestRegressor()

# fit on train data
RF_reg.fit(X_train, y_train)

#### Metrics for Train data

# calculate error for train X and train y
```

```
pred_ytrain = RF_reg.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square          =          {},\n\tAdj          R-square          =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

#### Metrics for Test data

# calculate error for test X and test y

pred_ytest = RF_reg.predict(X_test)    # predict from X test
mae_tr = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts       =       1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

#### Feature importance in model

RF_reg.feature_importances_
feature_imp_rf        =        pd.Series(RF_reg.feature_importances_,
index=X_selected.columns)
plt.figure(figsize=(6,6))
feature_imp_rf.nlargest(21).plot(kind='barh')
plt.grid()

feature_imp_rf                                                    =
pd.DataFrame.from_dict({'features':X_selected.columns,"Imp":RF_reg.fea
ture_importances_})
feature_imp_rf.sort_values(by='Imp', ascending=False)
```

```
### Training X_mod and y with RandomForestRegressor and calculate various
evaluation metrics

X_mod                                                                    =
df[['HighestSaving','AvgBillAmount','AvgNumberOfProduct_log','AvgSavin
g_S','RFM_label',

'VisitInLastMonth_Bin','SavingPercent','Tenure_log','Complain','CityTi
er','SatisfactionScore','Age',
'LastMonthCouponUsed_Bin','Gender_Male']]

# divide data with selected features into test and train
from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X_mod,   y,
test_size=0.3, random_state=123)

# cretaing instance for Rf regressor
RF_reg = RandomForestRegressor()

# fit on train data
RF_reg.fit(X_train, y_train)

#### Metrics for Train data

# calculate error for train X and train y

pred_ytrain = RF_reg.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square          =          {},\n\tAdj          R-square          =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

#### Metrics for Test data

# calculate error for test X and test y

pred_ytest = RF_reg.predict(X_test)    # predict from X test
mae_tr = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
```

```
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts        =        1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

#### Feature importance in model

RF_reg.feature_importances_
feature_imp_rf         =         pd.Series(RF_reg.feature_importances_,
index=X_mod.columns)
plt.figure(figsize=(6,6))
feature_imp_rf.nsmallest(21).plot(kind='barh')
plt.grid()

feature_imp_rf                                                          =
pd.DataFrame.from_dict({'features':X_mod.columns,"Imp":RF_reg.feature_
importances_})
feature_imp_rf.sort_values(by='Imp', ascending=False)

## Use Linear Regression Model for training and testing the two datasets

importance_split = sm.OLS(y, X).fit()
print(importance_split.params)
reg.coef_
reg.intercept_
reg.predict(np.array([[3, 5]]))

# import Linear Regression

from sklearn.linear_model import LinearRegression

### Training X_all and y with Linear Regression and calculate various
evaluation metrics

# divide data with all features into test and train
from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X_all,   y,
test_size=0.3, random_state=123)

# cretaing instance for Rf regressor
LR = LinearRegression()
```

```
# fit on train data
LR.fit(X_train, y_train)

# calculate error for train X and train y

pred_ytrain = LR.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square         =         {},\n\tAdj         R-square         =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

# calculate error for test X and test y

pred_ytest = LR.predict(X_test)    # predict from X test
mae_tr = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts        =        1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

feature_imp_lr                                                    =
pd.DataFrame.from_dict({'features':X_all.columns,"Imp":abs(LR.coef_)})
feature_imp_lr.sort_values(by='features')

### Training X_selected and y with Linear Regression and calculate
various evaluation metrics

# divide data with all features into test and train for selected features
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y,
test_size=0.3, random_state=123)
```

```
# cretaing instance for Rf regressor
LR = LinearRegression()

# fit on train data
LR.fit(X_train, y_train)

#### Metrics for Train data

# calculate error for train X and train y

pred_ytrain = LR.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square          =          {},\n\tAdj          R-square          =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

#### Metrics for Test data

# calculate error for test X and test y

pred_ytest = LR.predict(X_test)    # predict from X test
mae_tr = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts        =        1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

#### Feature importance in model

# analyse important features

LR.coef_
```

```
feature_imp_lr = pd.Series(LR.coef_, index=X_selected.columns)
plt.figure(figsize=(6,6))
feature_imp_lr.nsmallest(21).plot(kind='barh')
plt.grid()

feature_imp_lr                                                    =
pd.DataFrame.from_dict({'features':X_selected.columns,"Imp":abs(LR.coe
f_)})
feature_imp_lr.sort_values(by='Imp', ascending=False)

## Use Lasso Regression for training and testing the two datasets

from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold

model = Lasso(alpha=1.0)

cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

from sklearn.model_selection import train_test_split

X_train,   X_test,   y_train,   y_test   =   train_test_split(X_all,   y,
test_size=0.3, random_state=123)

scores      =      cross_val_score(model,      X_train,      y_train,
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)

from numpy import absolute
from numpy import mean
from numpy import std
scores = absolute(scores)
print('Mean MAE: %.3f (%.3f)' % (mean(scores), std(scores)))

model.fit(X_train,y_train)

y_pred1=model.predict(X_test)

from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test,y_pred1)

print(model.score(X_test , y_test))

print(model.score(X_train , y_train))

from math import sqrt
rmse = sqrt(mean_squared_error(y_test, y_pred1))

print(rmse)
```

```
### Training X_all and y with Lasso Regression and calculate various
evaluation metrics

# divide data with selected features into test and train for all features
from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X_all,   y,
test_size=0.3, random_state=123)


# creating instance for Lasso
model_lasso = Lasso(alpha=1.0)
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
scores     =     cross_val_score(model_lasso,     X_train,     y_train,
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)

# fit on train data
model_lasso.fit(X_train,y_train)

# calculate error for train X and train y

pred_ytrain = model_lasso.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square         =         {},\n\tAdj         R-square         =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

# calculate error for test X and test y

pred_ytest = model_lasso.predict(X_test)    # predict from X test
mae_tr = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts        =        1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))
```

```
print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_tr,
mse_ts,rmse_ts,R2_ts,adR2_ts))

feature_imp_las                                                    =
pd.DataFrame.from_dict({'features':X_all.columns,"Imp":model_lasso.coe
f_})
feature_imp_las.sort_values(by='features')

### Training X_selected and y with Lasso Regression and calculate various
evaluation metrics

# divide data with selected features into test and train for selected
features
from sklearn.model_selection import train_test_split
X_train,  X_test,  y_train,  y_test  =  train_test_split(X_selected,  y,
test_size=0.3, random_state=123)

# cretaing instance for Lasso
model_lasso = Lasso(alpha=1.0)
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
scores     =     cross_val_score(model_lasso,     X_train,     y_train,
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)

# fit on train data
model_lasso.fit(X_train,y_train)

#### Metrics for Train data

# calculate error for train X and train y

pred_ytrain = model_lasso.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square          =          {},\n\tAdj          R-square          =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

#### Metrics for Test data

# calculate error for test X and test y
```

```
pred_ytest = model_lasso.predict(X_test)    # predict from X test
mae_tr = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts        =        1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

#### Feature importance in model

# analyse important features
feature_imp_las             =             pd.Series(model_lasso.coef_,
index=X_selected.columns)
plt.figure(figsize=(6,6))
feature_imp_las.nlargest(21).plot(kind='barh')
plt.grid()

import statsmodels.api as sm
importance_split = sm.OLS(y, X_selected).fit()
print(importance_split.params.sort_values(ascending=False))

feature_imp_las                                                    =
pd.DataFrame.from_dict({'features':X_selected.columns,"Imp":model_lass
o.coef_})
feature_imp_las.sort_values(by='Imp', ascending=False)

### Training X_mod and y with Lasso Regression and calculate various
evaluation metrics

X_mod                                                              =
df[['HighestSaving','AvgBillAmount','AvgNumberOfProduct_log','AvgSavin
g_S','RFM_label',

'VisitInLastMonth_Bin','SavingPercent','Tenure_log','Complain','CityTi
er','SatisfactionScore','Age',
'LastMonthCouponUsed_Bin','Gender_Male']]

# divide data with selected features into test and train for modified X
depending on feature selection comparison

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_mod, y,
test_size=0.3, random_state=123)


# creating instance for Lasso
model_lasso = Lasso(alpha=1.0)
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(model_lasso, X_train, y_train,
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)

# fit on train data
model_lasso.fit(X_train,y_train)

# calculate error for train X and train y

pred_ytrain = model_lasso.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr = 1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square = {},\n\tAdj R-square =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

# calculate error for test X and test y

pred_ytest = model_lasso.predict(X_test)    # predict from X test
mae_ts = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts = 1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square = {},\n\tAdj R-square = {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))
```

```
feature_imp_las                                                    =
pd.DataFrame.from_dict({'features':X_mod.columns,"Imp":model_lasso.coe
f_})
feature_imp_las.sort_values(by='Imp', ascending=False)

# remove lastMonthCouponUsed from X_mod as it shows 0 coeff and check
results

X_mod                                                              =
df[['HighestSaving','AvgBillAmount','AvgNumberOfProduct_log','AvgSavin
g_S','RFM_label',

'VisitInLastMonth_Bin','SavingPercent','Tenure_log','Complain','CityTi
er','SatisfactionScore','Age',
'Gender_Male']]

from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X_mod,   y,
test_size=0.3, random_state=123)


# creating instance for Lasso
model_lasso = Lasso(alpha=1.0)
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
scores     =     cross_val_score(model_lasso,     X_train,     y_train,
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)

# fit on train data
model_lasso.fit(X_train,y_train)

# calculate error for train X and train y

pred_ytrain = model_lasso.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square          =          {},\n\tAdj          R-square          =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

# calculate error for test X and test y
```

```
pred_ytest = model_lasso.predict(X_test)    # predict from X test
mae_ts = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts        =        1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

feature_imp_las                                                    =
pd.DataFrame.from_dict({'features':X_mod.columns,"Imp":model_lasso.coe
f_})
feature_imp_las.sort_values(by='Imp', ascending=False)

# Lets Gender_Male as it does logically makes sense to have profitability
be dependent on Gender

X_mod                                                              =
df[['HighestSaving','AvgBillAmount','AvgNumberOfProduct_log','AvgSavin
g_S','RFM_label',

'VisitInLastMonth_Bin','SavingPercent','Tenure_log','Complain','CityTi
er','SatisfactionScore','Age']]

from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X_mod,   y,
test_size=0.3, random_state=123)


# creating instance for Lasso
model_lasso = Lasso(alpha=1.0)
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
scores     =     cross_val_score(model_lasso,     X_train,     y_train,
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)

# fit on train data
model_lasso.fit(X_train,y_train)

# calculate error for train X and train y

pred_ytrain = model_lasso.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
```

```
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square         =         {},\n\tAdj         R-square         =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

# calculate error for test X and test y

pred_ytest = model_lasso.predict(X_test)    # predict from X test
mae_ts = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts       =       1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

feature_imp_las                                                    =
pd.DataFrame.from_dict({'features':X_mod.columns,"Imp":model_lasso.coe
f_})
feature_imp_las.sort_values(by='Imp', ascending=False)

# drop complain as itis not logical that complaints have positive affect
on profitability

X_mod                                                              =
df[['HighestSaving','AvgBillAmount','AvgNumberOfProduct_log','AvgSavin
g_S','RFM_label',

'VisitInLastMonth_Bin','SavingPercent','Tenure_log','CityTier','Satisf
actionScore','Age']]

from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X_mod,   y,
test_size=0.3, random_state=123)
```

```
# creating instance for Lasso
model_lasso = Lasso(alpha=1.0)
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
scores     =     cross_val_score(model_lasso,     X_train,     y_train,
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)

# fit on train data
model_lasso.fit(X_train,y_train)

# calculate error for train X and train y

pred_ytrain = model_lasso.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square          =          {},\n\tAdj          R-square          =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

# calculate error for test X and test y

pred_ytest = model_lasso.predict(X_test)    # predict from X test
mae_ts = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts       =       1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

feature_imp_las                                                    =
pd.DataFrame.from_dict({'features':X_mod.columns,"Imp":model_lasso.coe
f_})
feature_imp_las.sort_values(by='Imp', ascending=False)
```

```
# We may also remove AvgBillAmount as it collinear to AvgSaving and also
has a very small coeff value

X_mod                                                                =
df[['HighestSaving','AvgNumberOfProduct_log','AvgSaving_S','RFM_label'
,

'VisitInLastMonth_Bin','SavingPercent','Tenure_log','CityTier','Satisf
actionScore','Age']]

from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X_mod,   y,
test_size=0.3, random_state=123)


# creating instance for Lasso
model_lasso = Lasso(alpha=1.0)
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
scores     =     cross_val_score(model_lasso,     X_train,     y_train,
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)

# fit on train data
model_lasso.fit(X_train,y_train)

# calculate error for train X and train y

pred_ytrain = model_lasso.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square         =         {},\n\tAdj         R-square         =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

# calculate error for test X and test y

pred_ytest = model_lasso.predict(X_test)    # predict from X test
mae_ts = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
```

```
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts        =        1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

feature_imp_las                                                    =
pd.DataFrame.from_dict({'features':X_mod.columns,"Imp":model_lasso.coe
f_})
feature_imp_las.sort_values(by='Imp', ascending=False)

# We may also remove SavingPercent as it is inexplicable being negative
to profitability

X_mod                                                              =
df[['HighestSaving','AvgNumberOfProduct_log','AvgSaving_S','RFM_label'
,

'VisitInLastMonth_Bin','Tenure_log','CityTier','SatisfactionScore','Ag
e']]

from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X_mod,   y,
test_size=0.3, random_state=123)


# creating instance for Lasso
model_lasso = Lasso(alpha=1.0)
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
scores     =     cross_val_score(model_lasso,     X_train,     y_train,
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)

# fit on train data
model_lasso.fit(X_train,y_train)

# calculate error for train X and train y

pred_ytrain = model_lasso.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
```

```
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square          =          {},\n\tAdj          R-square          =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

# calculate error for test X and test y

pred_ytest = model_lasso.predict(X_test)    # predict from X test
mae_ts = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts        =        1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

feature_imp_las                                                    =
pd.DataFrame.from_dict({'features':X_mod.columns,"Imp":model_lasso.coe
f_})
feature_imp_las.sort_values(by='Imp', ascending=False)

# analyse important features
feature_imp_las = pd.Series(model_lasso.coef_, index=X_mod.columns)
plt.figure(figsize=(6,6),)
feature_imp_las.nsmallest(21).plot(kind='barh')
plt.grid()



# We may also remove SavingPercent as it is inexplicable being negative
to profitability

X_mod                                                              =
df[['HighestSaving','AvgNumberOfProduct_log','AvgBillAmount','AvgSavin
g_S','RFM_label',

'VisitInLastMonth_Bin','Tenure_log','CityTier','SatisfactionScore','Ag
e']]

from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X_mod,   y,
test_size=0.3, random_state=123)
```

```
# creating instance for Lasso
model_lasso = Lasso(alpha=1.0)
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
scores     =     cross_val_score(model_lasso,     X_train,     y_train,
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)

# fit on train data
model_lasso.fit(X_train,y_train)

# calculate error for train X and train y

pred_ytrain = model_lasso.predict(X_train)    # predict from X train
mae_tr = mean_absolute_error(y_train, pred_ytrain)  # calculate MAE for
Y train and y predicted
mse_tr = mean_squared_error(y_train, pred_ytrain) # calculate MSE for Y
train and y predicted
rmse_tr = sqrt(mean_squared_error(y_train,pred_ytrain))
R2_tr = r2_score(y_train,pred_ytrain)
adR2_tr      =      1-(1-r2_score(y_train,pred_ytrain))*(len(y_train)-
1)/(len(y_train)-X_train.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtrain and ytrain :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square         =         {},\n\tAdj         R-square         =
{}'.format(mae_tr,mse_tr,rmse_tr,R2_tr,adR2_tr))

# calculate error for test X and test y

pred_ytest = model_lasso.predict(X_test)    # predict from X test
mae_ts = mean_absolute_error(y_test, pred_ytest)  # calculate MAE for Y
test and y predicted
mse_ts = mean_squared_error(y_test, pred_ytest) # calculate MSE for Y
test and y predicted
rmse_ts = sqrt(mean_squared_error(y_test,pred_ytest))
R2_ts = r2_score(y_test,pred_ytest)
adR2_ts       =       1-(1-r2_score(y_test,pred_ytest))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1)
# print("RMSE: %f" % (rmse_ytr))

print('For Xtest and ytest :\n\tMAE = {},\n\tMSE = {},\n\tRMSE = {},\
\n\tR-square    =    {},\n\tAdj    R-square    =    {}'.format(mae_ts,
mse_ts,rmse_ts,R2_ts,adR2_ts))

feature_imp_las                                                    =
pd.DataFrame.from_dict({'features':X_mod.columns,"Imp":model_lasso.coe
f_})
feature_imp_las.sort_values(by='Imp', ascending=False)
```

```
# analyse important features
feature_imp_las = pd.Series(model_lasso.coef_, index=X_mod.columns)
plt.figure(figsize=(6,6))
feature_imp_las.nlargest(21).plot(kind='barh')
plt.grid()

df.corr()

df.skew()

df.kurtosis()
```