Customer Churn prediction

Virginia Tech, Department of Computer Science

Author: Shekhar Kumar

Instructor: Dr. Reza Jafari

Dec 8, 2023

Machine Learning CS5805

Table of Contents

Table of figures and tables

1.Abstract

Businesses have to be extremely concerned about customer churn because it affects gains, expansion, and client happiness. Predictive simulation techniques are used to anticipate and minimize turnover in order to address this problem. The churn dataset is made up of training

and testing files that each contain customer records with different properties. The dataset's objective is to assess churn prediction models' efficacy on unknown data so that companies may maximize their customer retention tactics. Customer demographics, usage trends, interactions with support, and spending patterns are among the features. Businesses can improve overall service quality by proactively engaging with customers by having a greater knowledge of churn and its determinants.

2.Introduction

Reducing client attrition is critical in today's evolving customer-focused companies. Churn, which occurs when customers end their membership or a connection, can have a significant impact on companies. One effective technique for anticipating and addressing customer attrition before it happens is predictive modeling. An extensive churn dataset consisting of records is examined and used for model building. Customer variables like age, gender, tenure, support interactions, and financial activities are all included in the dataset. The goal is to evaluate the churn models' forecasting ability on data that has never been seen before, giving companies information to improve and optimize their customer retention tactics.

Phase I 3.Description of the dataset

The purpose of the churn dataset is to solve the issues associated with customer churn in a variety of businesses by compiling an extensive set of customer information. There are two primary parts to it: a testing file and a training file. Effective churn prediction models developed, and trained and their effectiveness can be assessed using customer data that hasn't been seen

The dataset has a binary classification target variable along with a combination of numerical and categorical variables:

Target:

The goal variable in the churn dataset is "Churn," a binary classification that indicates whether a customer's subscription with the company is expected to continue (True) or end (False). The "False" class indicates clients who have already ended their affiliation or are in danger of doing so, whereas the "True" class indicates clients who are expected to stay in their relationship and show loyalty.

Features:

The dataset contains a wide range of features, each of which offers a distinct perspective on the interactions, behavior, and attributes of customers:

1 - CustomerID: A special number that is given to every customer to enable customized tracking.

2 - Age: The customer's age, which enables age-based segmentation and analysis.

3 - Gender: The customer's gender adds a demographic component to churn analysis.means divorced or widowed)

4 - Tenure: The length of time, expressed in months, that a client has been using the goods or services of the business.

5 - Usage Frequency: A measure of activity levels based on how frequently a client has used the business's services over the course of the previous month.

6 - Support Calls: A measure of customer service interactions, the number of calls a client placed to the customer support team in the previous month.

7 - Payment Delay: A measure of the customer's payment tardiness over the previous month that provides information about their financial habits.

8 - Subscription Type: The kind of subscription that the client choose to differentiate between various service packages.

9 -Contract Length: The length of the agreement the client signs with the business, which indicates the degree of commitment.

10 - Total Spend: An important financial indicator that represents the total amount a client has paid for goods or services from the business.

11 -Last Interaction: The duration in days since the client's most recent communication with the business, signifying the recentness of the engagement.

Purpose and Significance::

The main objective of the churn dataset is to function as a fundamental resource for the creation and instruction of predictive algorithms designed to foresee client attrition. Businesses can use sophisticated machine learning algorithms to identify underlying trends, correlations, and predicted factors that contribute to customer attrition by utilizing a wide range of customer-centric characteristics. The developed models enable targeted retention tactics and promote a proactive approach to customer relationship management by helping firms to proactively identify customers who are at danger of leaving.

The dataset is valuable not only for predictive modeling but also for assessing how well churn prediction algorithms function in practice. The comprehensive attributes of the dataset provide significant understanding of the traits and actions of customers linked to attrition, enabling data-driven choices to improve customer contentment, customize advertising plans, and carry out customized retention campaigns. To put it simply, the churn dataset is a vital resource for companies looking to comprehend, anticipate, and manage consumer attrition, which eventually leads to better client retention and long-term company expansion.

4. Phase II Preprocessing

4.1 Null Check:

```
Number of observation in training dataset: 440833
Null value count
Age                   1
Gender                1
Tenure                1
Usage_Frequency       1
Support_Calls         1
Payment_Delay         1
Subscription_Type     1
Contract_Length       1
Total_Spend           1
Last_Interaction      1
Churn                 1
dtype: int64
Training: Total number of rows with null value = 11
```

The isna() function was used to do the null value check on the training and test datasets. One row in the training dataset was found to have all feature values set to null. In order to ensure the integrity of the data, the row containing null values was eliminated from the training dataset. By taking this preemptive step, possible problems that could come from missing or incomplete data during model training are avoided.

Conversely, there were no null values found in any of the characteristics in the test dataset. It is comforting to know that the test dataset has no null values, indicating that the model can be used on it without raising any issues with missing data. In order to guarantee that machine learning models are robust, handling null values is an essential step in the preprocessing of

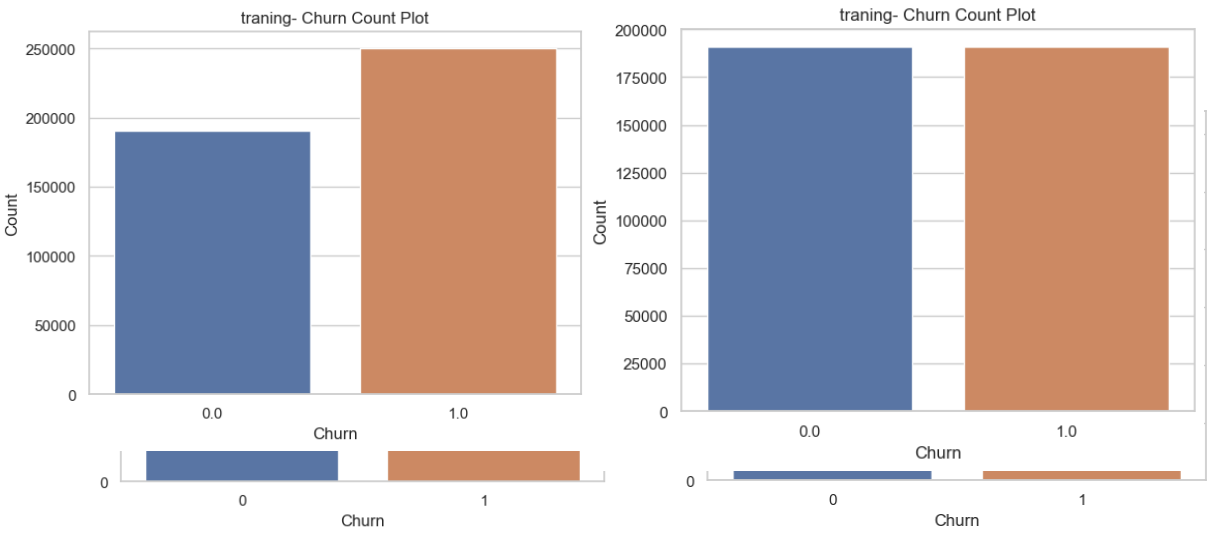
data. The null value check in both datasets enhances the overall quality and dependability of the data for ensuing analyses and model evaluations.

4.2 Data duplications and removal:

```
Train data duplicated= 0
Test data duplicated= 0
```

Using the duplicated() method, the existence of duplicate data was examined in the training and test datasets. According to the analysis, there were no duplicate rows in either the test or training sets. The lack of duplicate data guarantees the datasets' integrity and uniqueness, which is a desirable result. The identification and elimination of duplicate entries are crucial phases in the preparation of data since they might introduce bias and affect the prediction models' accuracy. The verification that there are no duplicates in both datasets improves the data's dependability and quality, providing a strong basis for the creation and assessment of machine learning models

4.3 Down sampling:

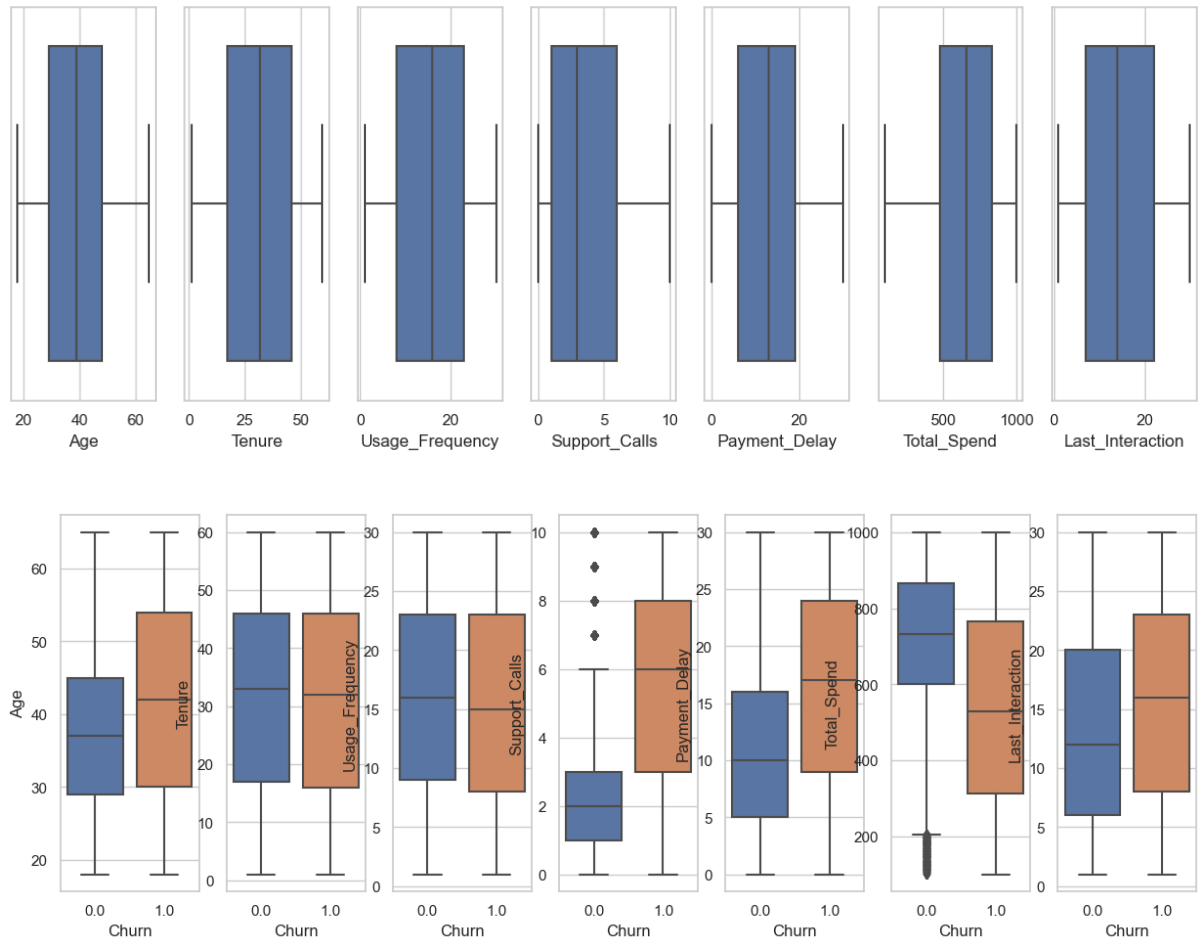


By observing the churn feature, it became clear that there was an imbalance in the churn feature distribution, with one class having noticeably more observations than the other. In the training set, there were 249,999 observations for churn=1 and 190,833 observations for

churn=0. In the test data, churn=0 had 33,881 observations, and churn=1 had 30,493 observations. Down sampling was done to rectify this disparity and provide a more equal representation of both classes.

In order to align the number of churn=0 and churn=1 cases, observations were methodically eliminated from the training and test datasets throughout the downsampling process. By taking this method, the model's bias toward the majority class is lessened and its capacity to identify patterns pertaining to the minority class is improved. Consequently, following downsampling, the number of observations for churn=0 and churn=1 is equal in both the training and test datasets. The robustness and fairness of the predictive models are enhanced by this balanced dataset, especially when handling imbalanced classification problems such as churn prediction.

4.4 Outlier Analysis:



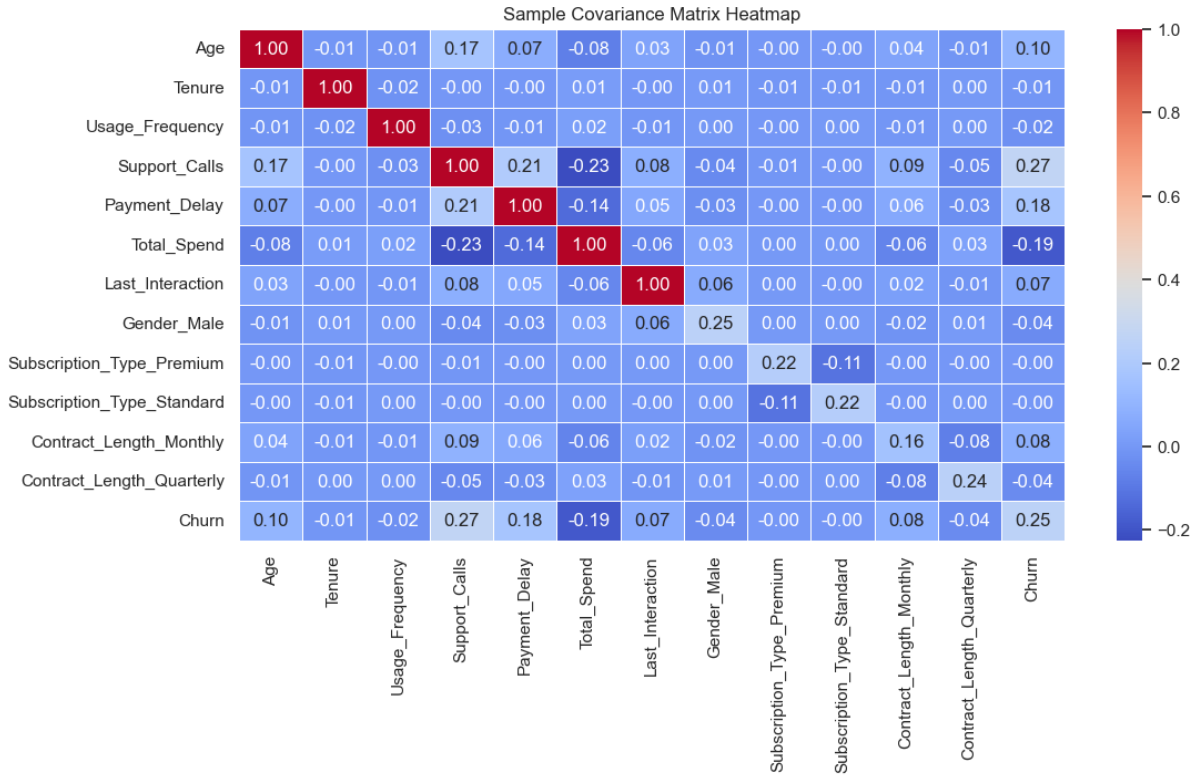
To find outliers in the dataset, I have used box plots. Individual features were shown in the first set of box plots, which showed that there were no obvious outliers for several features. Later on, I made further box plots to investigate the link between the features and the goal variable, comparing 'Churn' to 'Support Calls' and 'Total Spend' to 'Churn.'

An anomaly was noticed in the 'Churn vs. Support Calls' graphic. Since "Support Calls" is a numerical feature that naturally varies greatly among clients, I decided not removing this outlier. Certain clients might receive assistance calls more frequently than others. This fluctuation is to be expected and offers useful data for churn prediction.

Similarly, an outlier was identified in the 'Total Spend vs. Churn' plot. Once more, because 'Total Spend' depends on the actions of specific customers, I decided not to remove this

anomaly. Consumers may display a range of purchasing habits, thus a high or low total spend does not always mean that something is wrong. By keeping these anomalies in the model, one can make sure that it takes into account the entire range of consumer behavior and develop a more sophisticated understanding of how attributes relate to the goal variable.

4.5 Covariance Matrix

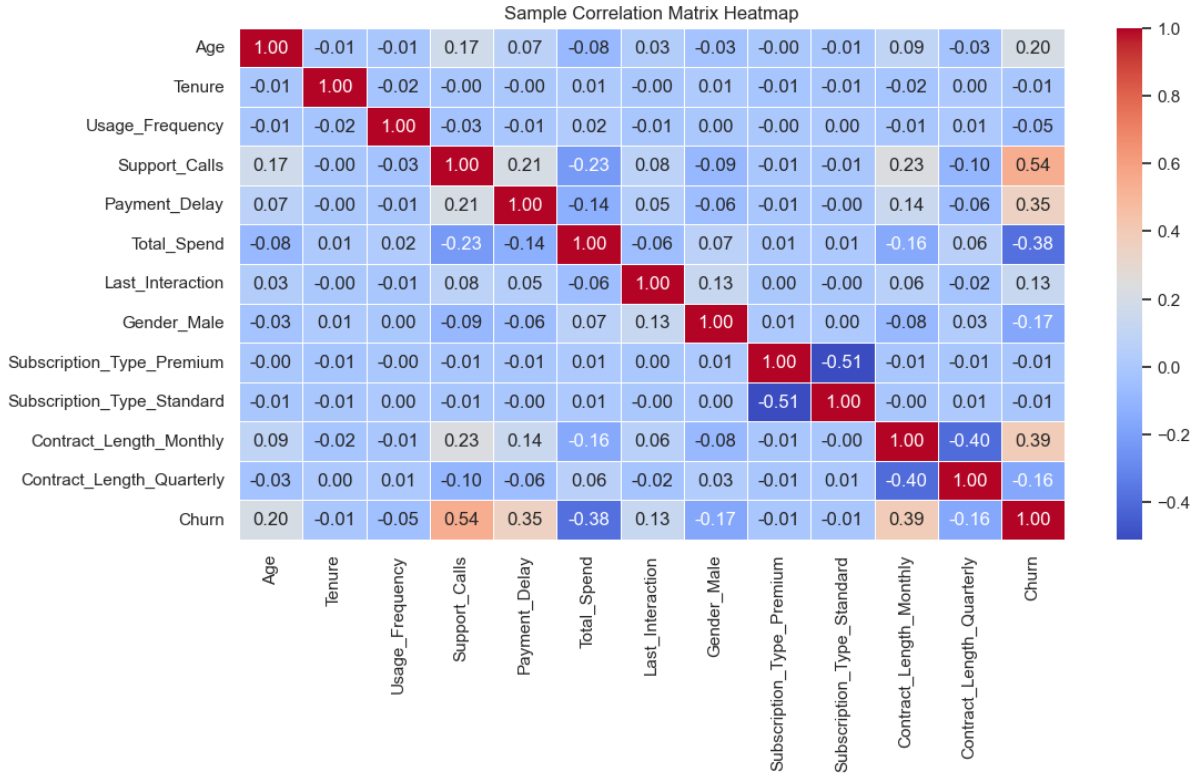


The Covariance Matrix for each of our dataset's features. The covariance between two distinct traits is represented by each element in the matrix. The covariance between a feature and itself is represented by the diagonal elements.

- The following features, in particular: 'Tenure,' 'Usage Frequency,' 'Support Calls,' 'Payment Delay,' 'Total Spend,' and 'Last Interaction Count' all exhibit a covariance of 1. This indicates that they tend to travel in the same direction and have a solid, positive association.

- The covariance value that I discovered when examining the association between "Churn" and "Support Calls" was 0.3. This shows a positive association, implying that a higher chance of churn is linked to an increase in the number of support calls.
- Notably, the correlation between "Total Spend" and "Support Calls" is negatively high at -0.22. This suggests a negative connection, meaning that the number of support calls tends to decline with a customer's total spending, and vice versa.

4.6 Correlation matrix



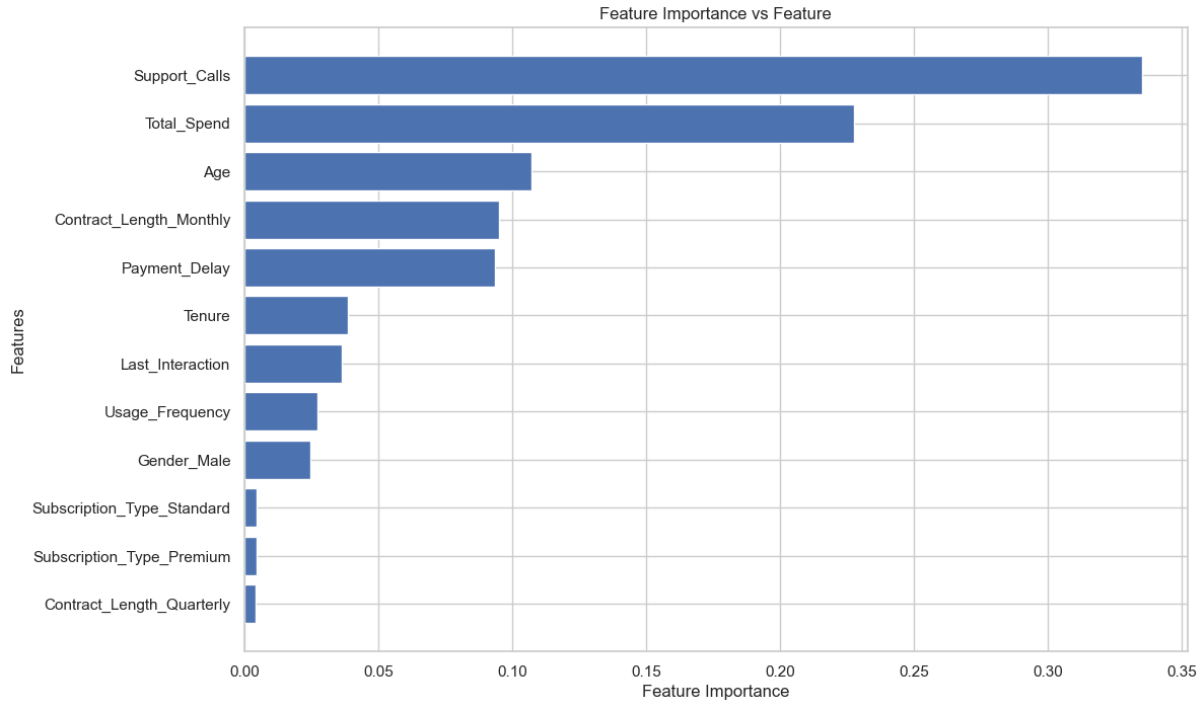
For the features in our dataset, I created a Correlation Matrix. The diagonal members of a correlation matrix always have a value of 1, signifying the correlation between each attribute and itself.

- There is a strong correlation—0.59—between "Support Calls" and "Churn." This implies a quite substantial positive association, suggesting that a higher chance of churn is linked to an increase in the quantity of support calls.

- The second-highest correlation (0.46) between "Churn" and "Contract Length Monthly" is observed. This shows a positive association, indicating that there may be a greater chance of customer churn for those with shorter monthly contract periods.
- owever, there is comparatively less of a correlation between "Total Spend" and "Churn." This implies that there may be less correlation between a customer's overall spending and their risk of leaving.
- 'Subscription Type Premium' and 'Subscription Type Standard' have a -0.51 correlation. These two subscription kinds appear to be somewhat negatively associated, as seen by the negative correlation; as one type increases, the other tends to decrease.

4.7 Dimensionality reduction/feature selection

4.7.1 Random Forest Analysis



We used Random Forest for feature reduction in an effort to refine our model and improve interpretability. The Random Forest model's computed feature importance shed light on the

relative importance of each feature in predicting our target variable. The feature importance are listed below:

Contract_Length_Quarterly= 0.0044

Subscription_Type_Premium= 0.0046

Subscription_Type_Standard= 0.0046

Gender_Male= 0.0246

Usage_Frequency= 0.0275

Last_Interaction= 0.0364

Tenure= 0.0389

Payment_Delay= 0.0935

Contract_Length_Monthly= 0.095

Age= 0.1074

Total_Spend= 0.2275

Support_Calls= 0.3354

Selected features= ['Gender_Male', 'Usage_Frequency', 'Last_Interaction', 'Tenure', 'Payment_Delay', 'Contract_Length_Monthly', 'Age', 'Total_Spend', 'Support_Calls']

Eliminated Features= ['Contract_Length_Quarterly', 'Subscription_Type_Premium', 'Subscription_Type_Standard']

We determined each feature's relevance using a 0.005 threshold. Features with comparatively lower importance were Contract_Length_Quarterly, Subscription_Type_Premium, and Subscription_Type_Standard. As a result, we have strategically chosen to eliminate certain attributes from our dataset.

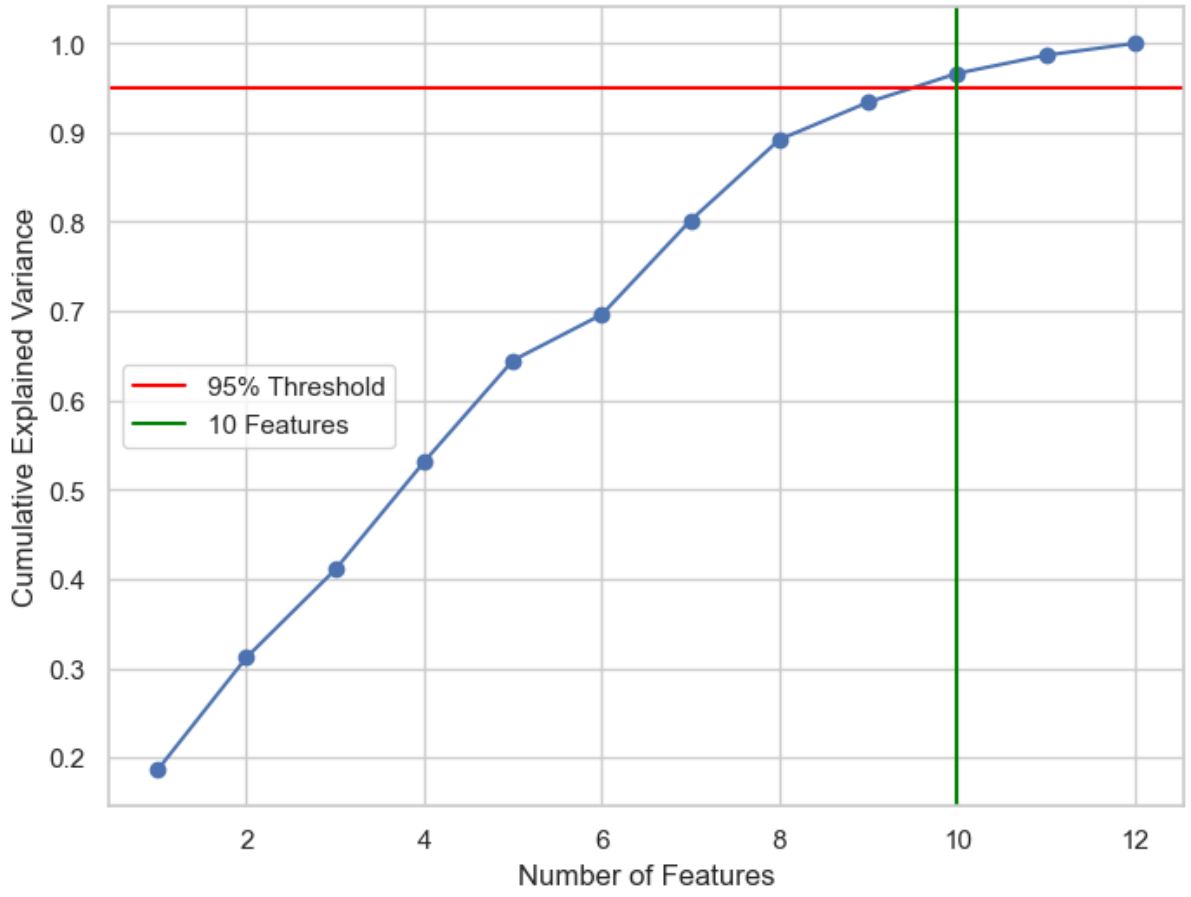
In addition to streamlining our dataset, this methodical feature reduction concentrates our model on the most important features, which may enhance interpretability and performance. The objective is to develop a more accurate and efficient predictive model for customer attrition by removing factors that have less of an impact.

4.7.2 Principal Component Analysis

Observation	Component-1	Component-2	Component-3	Component-4	Component-5	Component-6	Component-7	Component-8	Component-9	Component-10
0	-0.168	0.284	-0.463	-1.382	-0.803	-0.010	-1.030	-0.010	-0.396	0.427
1	-0.347	-0.931	-1.247	-1.548	0.529	-0.145	0.173	0.719	0.466	0.570
2	-1.483	-0.963	-0.303	-0.204	0.241	-0.168	0.000	0.697	-0.335	-0.492
3	-2.268	1.489	-0.813	-0.016	-0.599	-0.333	0.916	0.715	0.321	0.617
4	1.402	0.093	-0.462	-2.227	0.909	-1.853	0.223	-0.020	-0.014	-0.566

A conditional number of 3.75 indicated a moderate degree of multicollinearity, according to the dataset's first assessment. Principal Component Analysis (PCA) was used as a response to lessen multicollinearity and dimensionality. The dataset showed a significant improvement after the PCA transformation, with the conditional number falling to 2.54. This reduction indicates that the intercorrelations between the initial features have been successfully mitigated. Ten main components, each of which represents a unique linear combination of the starting variables, were derived through the use of PCA. A predefined threshold of 95% variance retention served as the basis for the decision to keep these 10 components,

guaranteeing that a significant amount of the variability included in the original dataset was retained by the modified data.



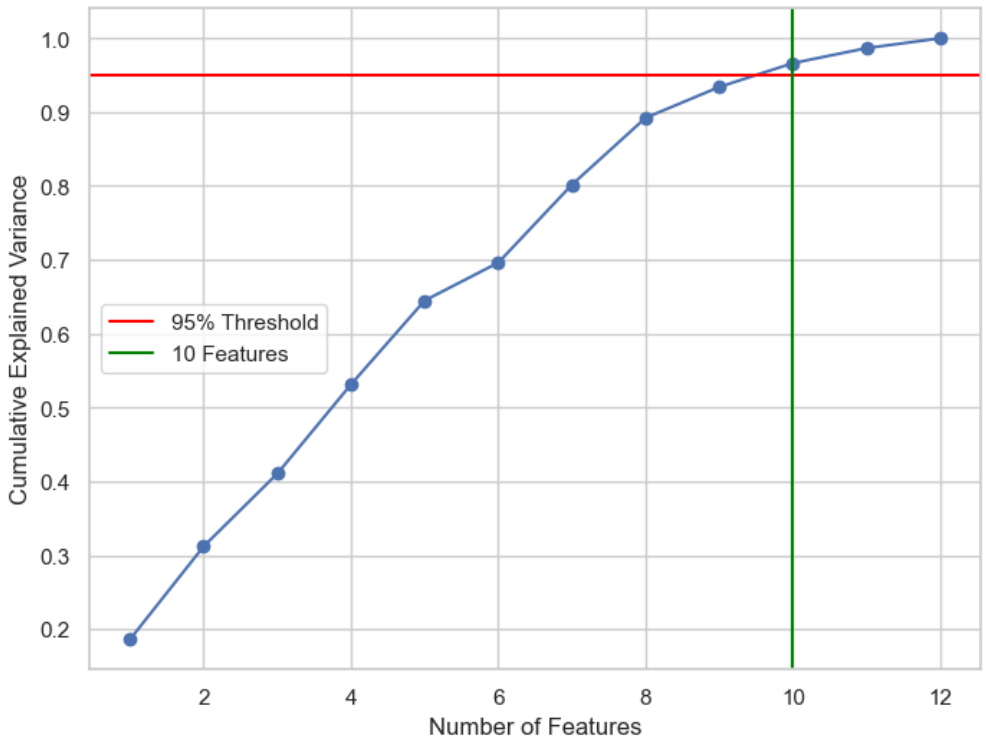
An understanding of the connections between the initial characteristics and the recently derived components can be gained from the loadings on each primary component. In addition to addressing multicollinearity issues, this condensed dataset makes it easier to illustrate the underlying patterns in the data in a clearer and more concise manner. This procedure helps to make the model more interpretable, which opens the door for more precise and successful analyses in the pipeline's next phases of data exploration and modeling.

4.7.3 Singular Value Decomposition Analysis

The dataset's singular value decomposition (SVD) produced the singular values [274.975, 225.973, 225.148, 221.028, 217.179, 211.472, 207.416, 191.7, 129.983, 114.453], each of

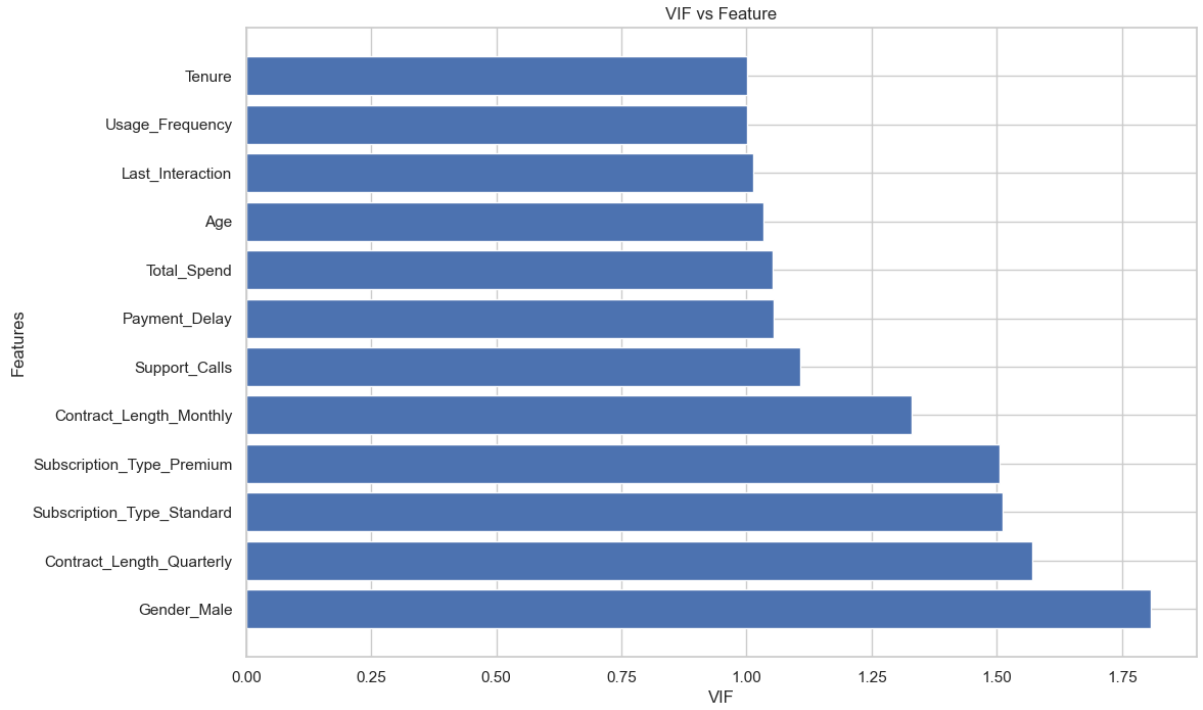
which represents the strength or relevance of its corresponding singular component. Ten components—a compressed representation of the original data—were produced by the SVD technique. The links between the initial features and the derived components are further demonstrated by the loadings of the first five observations on these ten major components.

Observation	Component-1	Component-2	Component-3	Component-4	Component-5	Component-6	Component-7	Component-8	Component-9	Component-10
0	-0.194	-0.331	-0.532	-1.302	-0.582	0.597	0.037	-1.031	-0.005	0.060
1	-0.406	0.876	-0.791	-1.327	1.014	1.249	-0.015	0.243	0.717	-0.631
2	-1.553	0.991	0.156	-0.139	0.479	0.862	-0.068	0.064	0.696	0.576
3	-2.323	-1.510	-0.256	0.113	-0.261	1.091	-0.229	0.969	0.714	-0.533
4	1.341	-0.087	-0.053	-2.110	1.243	1.006	-1.749	0.275	-0.022	0.273



This decomposition allows for a more streamlined and understandable representation of the dataset by highlighting the most important components and providing a foundation for dimensionality reduction. Using these principle components offers a useful way to record the important data while reducing the effect of less important dimensions in later analysis and modeling projects.

4.7.4 VIF



VIF Value for each features:

	Feature	VIF
0	Gender_Male	1.808764
1	Contract_Length_Quarterly	1.571163
2	Subscription_Type_Standard	1.511580
3	Subscription_Type_Premium	1.506384
4	Contract_Length_Monthly	1.329895
5	Support_Calls	1.106998
6	Payment_Delay	1.055270
7	Total_Spend	1.051531
8	Age	1.033161
9	Last_Interaction	1.012836
10	Usage_Frequency	1.001613
11	Tenure	1.001282

Regarding multicollinearity in the dataset, the interpretation of Variance Inflation Factor (VIF) values points to a good situation. Nearly all features have VIF values of 1, which indicates low multicollinearity. In general, VIF values of less than five are deemed appropriate, and in this instance, all features comfortably fall well below that cutoff. The lack of higher VIF scores suggests that there is little to no correlation between the traits.

The VIF study concludes by assuring that there are no notable issues with multicollinearity among the characteristics. Regression analysis benefits greatly from this result since it suggests that the features can be viewed as independent variables, allowing for a more precise evaluation of each feature's influence on the target variable.

4.8 Discretization & Binarization

Age	Tenure	Usage_Frequency	Support_Calls	Payment_Delay	Total_Spend	Last_Interaction	Gender_Male	Subscription_Type_Premium	Subscription
-0.351372	0.671318	1.078194	-0.860651	0.797234	-0.612633	-0.632480	False	False	False
0.528922	1.659117	0.380405	0.109257	-0.042606	0.414528	-1.328673	False	False	True
-0.271345	0.845636	-0.433683	-0.860651	-0.642492	0.909649	-0.632480	True	False	True
-1.391719	-1.129961	1.078194	-0.537348	-0.762470	1.455849	-1.328673	False	False	True
1.489243	1.310482	1.427089	1.079165	1.877029	0.897275	-0.284383	True	False	False
-1.151639	-1.711019	-0.084788	1.402467	-0.642492	0.012758	0.063713	False	False	True
1.489243	0.090260	1.078194	0.755862	-0.522515	-2.071878	1.688164	True	False	False
-1.231666	-1.536702	-0.549981	-1.183954	-0.762470	-0.275852	-0.632480	True	True	False
-0.191318	-1.304279	1.659685	-0.537348	1.277143	0.727015	-1.328673	False	False	False
-0.591452	0.961847	-0.433683	-0.214046	-1.242379	0.066742	-1.444705	True	False	True

In the binarization process, one-hot encoding has been employed with the utilization of Pandas' get_dummies function, configured with drop_first=True to handle categorical columns characterized by a limited number of unique values. Specifically, categorical features such as 'Gender' with 'Male' and 'Female,' 'Subscription type' encompassing 'Standard,' 'Basic,' and 'Premium,' and 'Contract length' involving 'Quarterly,' 'Monthly,' and 'Annual' have undergone transformation. This technique generates binary columns for each unique category within these features, effectively converting the categorical information into a binary matrix. The decision to drop the first column mitigates multicollinearity concerns and streamlines the representation of categorical data.

When working with categorical columns that have a limited number of unique values, one-hot encoding is especially useful. By converting categorical data into a format that is

compatible with machine learning techniques, it improves the data's interpretability and makes it easier to train models. Through the conversion of categorical variables into a binary matrix, this binarization technique optimizes the dataset for further analytical activities and machine learning model construction by ensuring that each category is sufficiently represented without adding superfluous information.

5.Phase II: Regression Analysis

5.1 T-test analysis

OLS Regression Results

Dep. Variable:	Support_Calls	R-squared:	0.300
Model:	OLS	Adj. R-squared:	0.300
Method:	Least Squares	F-statistic:	1789.
Date:	Fri, 08 Dec 2023	Prob (F-statistic):	0.00
Time:	11:12:30	Log-Likelihood:	-62012.
No. Observations:	50000	AIC:	1.240e+05
Df Residuals:	49987	BIC:	1.242e+05
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.5141	0.010	-49.568	0.000	-0.534	-0.494
Age	0.0604	0.004	15.793	0.000	0.053	0.068
Tenure	0.0022	0.004	0.587	0.557	-0.005	0.010
Usage_Frequency	-0.0008	0.004	-0.203	0.839	-0.008	0.007
Payment_Delay	0.0165	0.004	4.130	0.000	0.009	0.024
Last_Interaction	0.0056	0.004	1.470	0.142	-0.002	0.013
Total_Spend	-0.0201	0.004	-4.977	0.000	-0.028	-0.012
Gender_Male	0.0075	0.008	0.972	0.331	-0.008	0.023
Subscription_Type_Premium	-0.0132	0.009	-1.431	0.152	-0.031	0.005
Subscription_Type_Standard	-0.0039	0.009	-0.428	0.669	-0.022	0.014
Contract_Length_Monthly	0.0531	0.011	4.823	0.000	0.032	0.075
Contract_Length_Quarterly	-0.0093	0.008	-1.113	0.266	-0.026	0.007
Churn	1.0179	0.009	108.479	0.000	0.999	1.036

Omnibus:	288.830	Durbin-Watson:	1.997
Prob(Omnibus):	0.000	Jarque-Bera (JB):	293.784
Skew:	0.188	Prob(JB):	1.61e-64
Kurtosis:	2.991	Cond. No.	5.56

The t-test analysis conducted on the coefficients within the Ordinary Least Squares (OLS) regression model provides a detailed evaluation of the statistical significance of each predictor concerning support calls. Significance is determined by assessing the p-values associated with each coefficient, with values below the conventional threshold of 0.05 indicating statistical importance. Notably, Age, Payment_Delay, Total_Spend, Contract_Length_Monthly, and Churn emerge as highly significant predictors, unveiling distinctive relationships with support calls. Age and Payment_Delay exhibit positive associations, suggesting an increase in support calls with higher age and payment delays, while Total_Spend demonstrates a negative correlation, implying a decrease in support calls with higher total spending. Conversely, non-significant coefficients such as Tenure, Usage_Frequency, Last_Interaction, Gender_Male, Subscription_Type_Premium, and Contract_Length_Quarterly lack statistical significance, underscoring their limited impact on predicting support calls.

This comprehensive examination of individual coefficients guides the refinement of the regression model, emphasizing the importance of influential predictors while filtering out less impactful variables. By understanding the statistical significance of each feature, practitioners can enhance the accuracy and interpretability of the support call prediction model. This nuanced assessment enables data-driven decision-making, helping organizations focus on the most critical factors that contribute significantly to the variability in support call outcomes.

5.2 F-test analysis

The F-test analysis serves as a pivotal evaluation of the overall significance of the regression model in explaining the variance observed in the dependent variable, Support_Calls. The obtained F-statistic of 1789.3 is compared to the critical F-value of 1.75, setting a significance level of 0.05. This statistical comparison aids in determining whether the

collective impact of the independent variables significantly contributes to explaining the variability in support calls. The null hypothesis posits that all coefficients in the model are equal to zero, suggesting that none of the independent variables play a significant role in predicting the dependent variable.

R-squared:	0.300
Adj. R-squared:	0.300
F-statistic:	1789.
Prob (F-statistic):	0.00
Log-Likelihood:	-62012.
AIC:	1.240e+05
BIC:	1.242e+05

Examining the F-test findings closely reveals that the null hypothesis (1789.3 > Critical F) was rejected, highlighting the statistical importance of the regression model as a whole. This suggests that there is a substantial relationship between the number of support calls and at least one of the independent variables—Age, Payment_Delay, Total_Spend, Contract_Length_Monthly, and Churn. Even while this statistical significance provides a strong basis for additional investigation, care must be taken and each coefficient's practical importance must be explored. To guarantee the regression model's dependability and practicality, more diagnostics and validation procedures are advised. In essence, the F-test results provide confidence in the model's ability to elucidate and predict variations in support calls, paving the way for meaningful insights into the relationships between the selected features and support call outcomes.

5.3 Regression model

5.3.1 OLS- Stepwise regression

Features with high p-values were methodically removed from the model in each iteration. The p-value and the corrected R-squared were evaluated simultaneously as part of the

iterative process. The iteration procedure was stopped if a feature showed a high p-value and the modified R-squared value either fell or stayed the same. Finding a compromise between maintaining statistically significant predictors and guaranteeing the model's overall explanatory power was the aim. In the final iteration of OLS, features such as Usage_Frequency, Tenure, Gender_Male, Contract_Length_Quarterly, Subscription_Type_Premium, and Last_Interaction were identified and subsequently removed. The elimination of these features was guided by their high p-values, signifying their limited contribution to predicting the target variable. The OLS summary for the last iteration reflects that the p-values for all remaining features reached zero, reinforcing their statistical significance in predicting the outcome. This systematic and criteria-driven feature selection process ensures that the retained variables are both statistically significant and contribute meaningfully to the predictive capability of the model.

OLS model equation

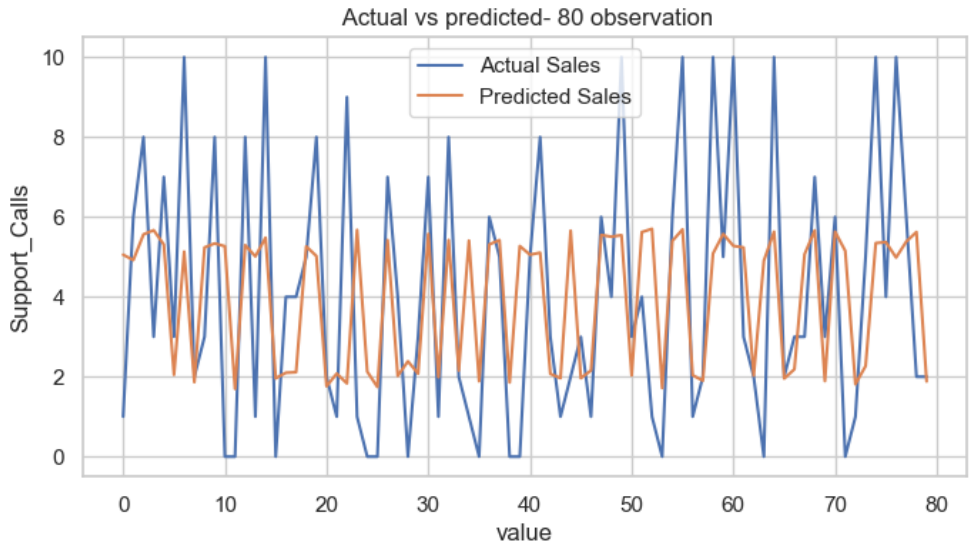
Support_Calls= -0.52 + 0.06 Age + 0.017 Payment_Delay - 0.02 Total_Spend + 0.058 Contract_Length_Monthly + 1.018 Churn

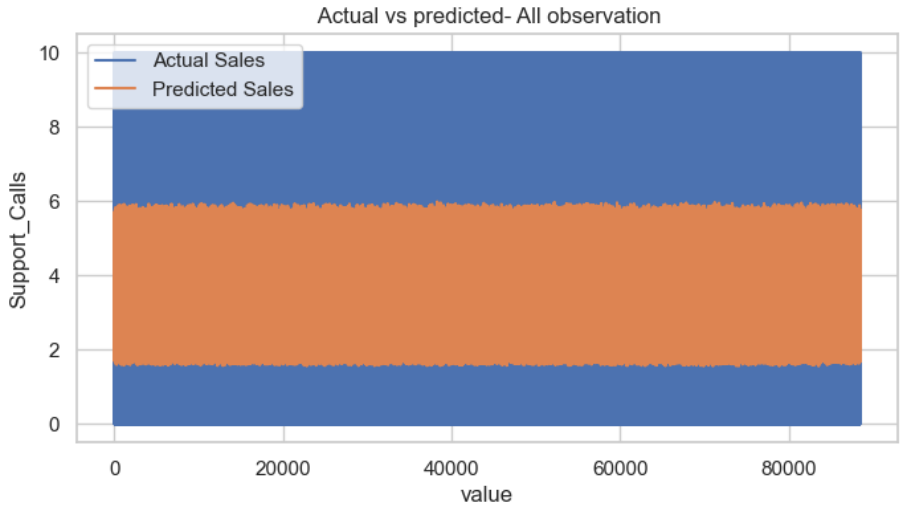
OLS Regression Results

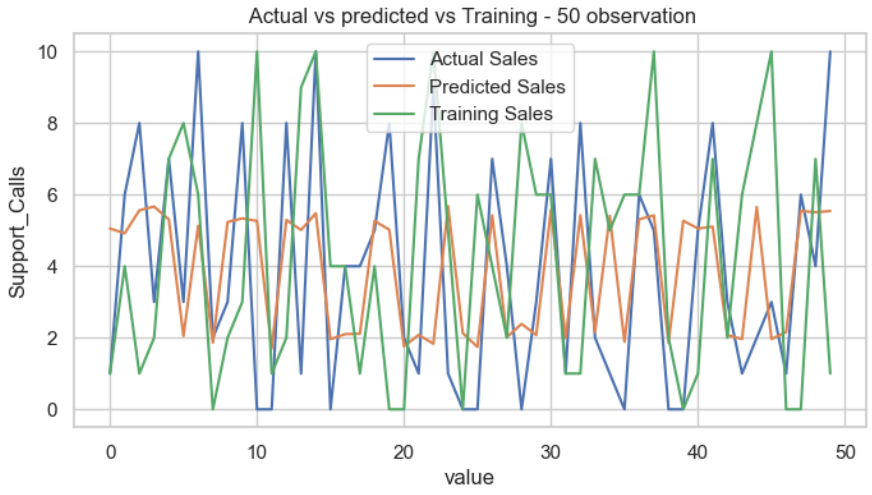
Dep. Variable:	Support_Calls	R-squared:	0.300
Model:	OLS	Adj. R-squared:	0.300
Method:	Least Squares	F-statistic:	4293.
Date:	Fri, 08 Dec 2023	Prob (F-statistic):	0.00
Time:	11:12:33	Log-Likelihood:	-62016.
No. Observations:	50000	AIC:	1.240e+05
Df Residuals:	49994	BIC:	1.241e+05
Df Model:	5		
Covariance Type:	nonrobust		

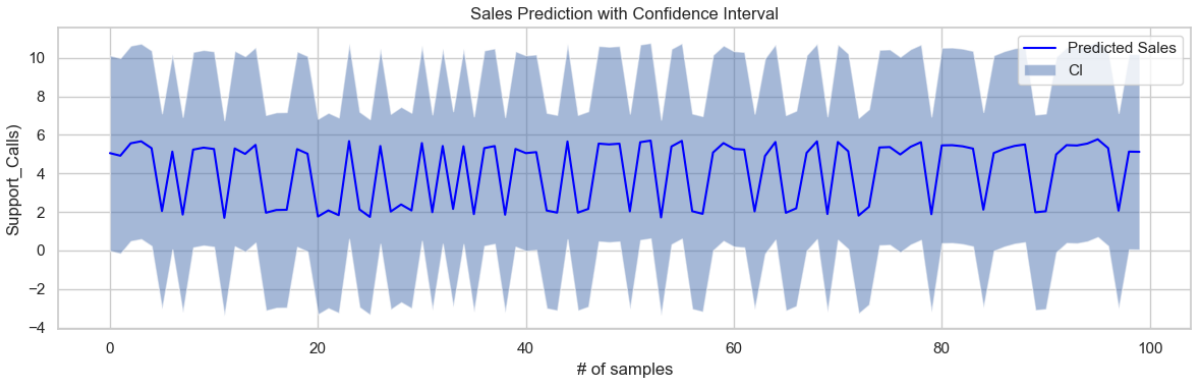
	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.5205	0.006	-90.382	0.000	-0.532	-0.509
Age	0.0604	0.004	15.795	0.000	0.053	0.068
Payment_Delay	0.0166	0.004	4.141	0.000	0.009	0.024
Total_Spend	-0.0202	0.004	-4.984	0.000	-0.028	-0.012
Contract_Length_Monthly	0.0576	0.010	5.641	0.000	0.038	0.078
Churn	1.0184	0.009	110.427	0.000	1.000	1.036

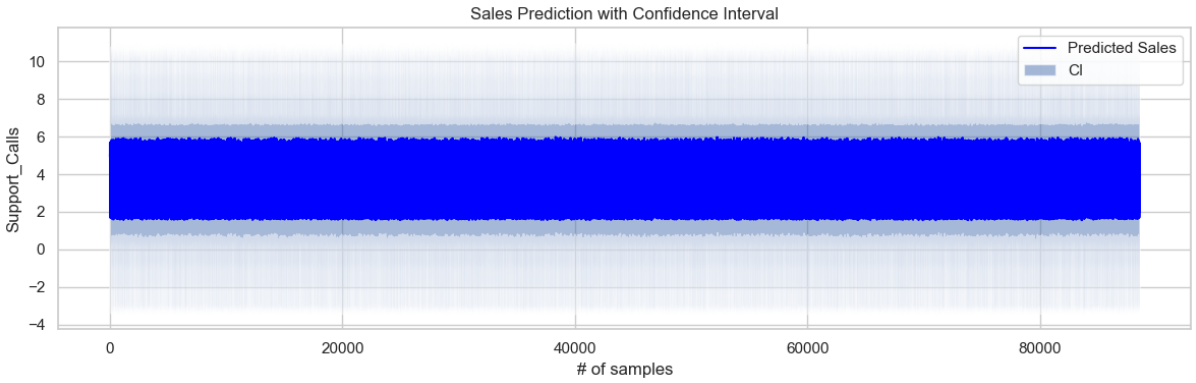
Omnibus:	287.877	Durbin-Watson:	1.998
Prob(Omnibus):	0.000	Jarque-Bera (JB):	292.798
Skew:	0.187	Prob(JB):	2.63e-64
Kurtosis:	2.991	Cond. No.	3.74




Actual vs predicted- All observation
Actual Sales
Predicted Sales
Support_Calls
value
0
20000
40000
60000
80000
0
2
4
6
8
10


Actual vs predicted vs Training - 50 observation
Actual Sales
Predicted Sales
Training Sales
Support_Calls
value
0
10
20
30
40
50
0
2
4
6
8
10


Sales Prediction with Confidence Interval
Predicted Sales
CI
Support_Calls)
# of samples
0
20
40
60
80
100
−4
−2
0
2
4
6
8
10

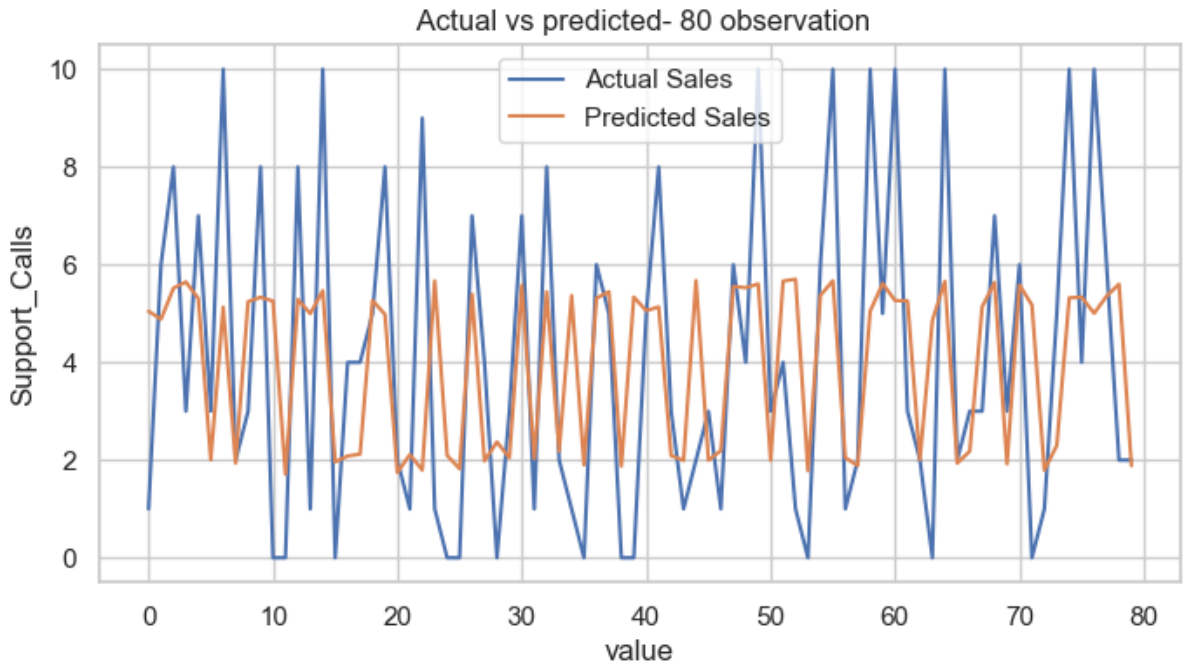


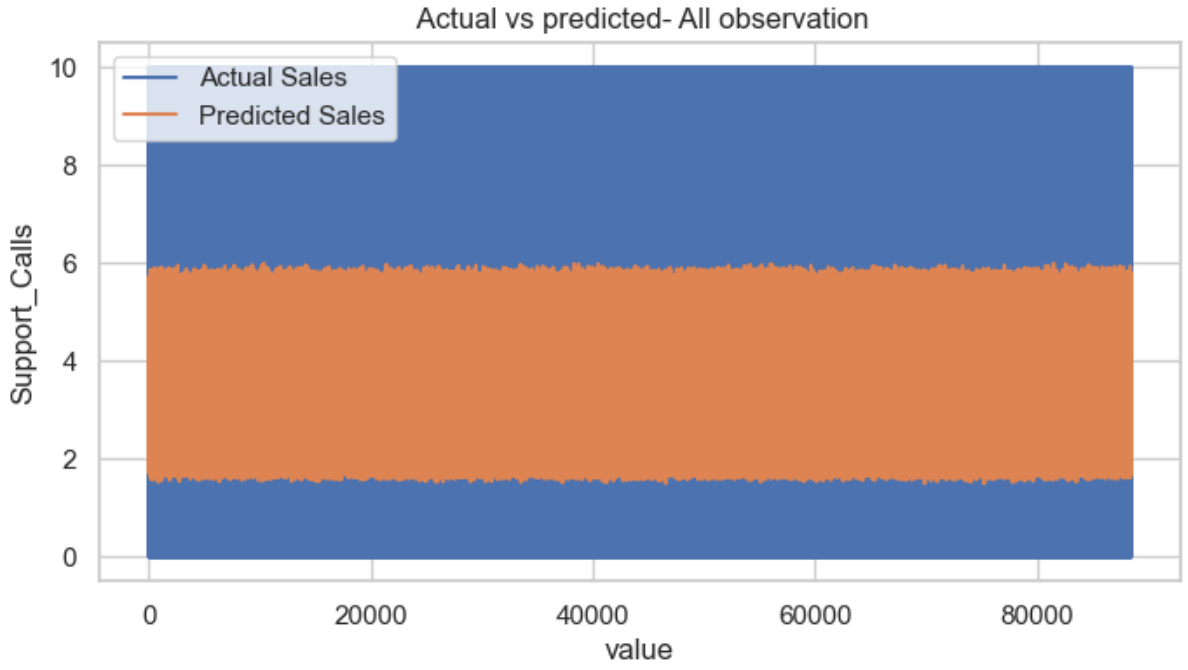
5.3.1 Linear Regression

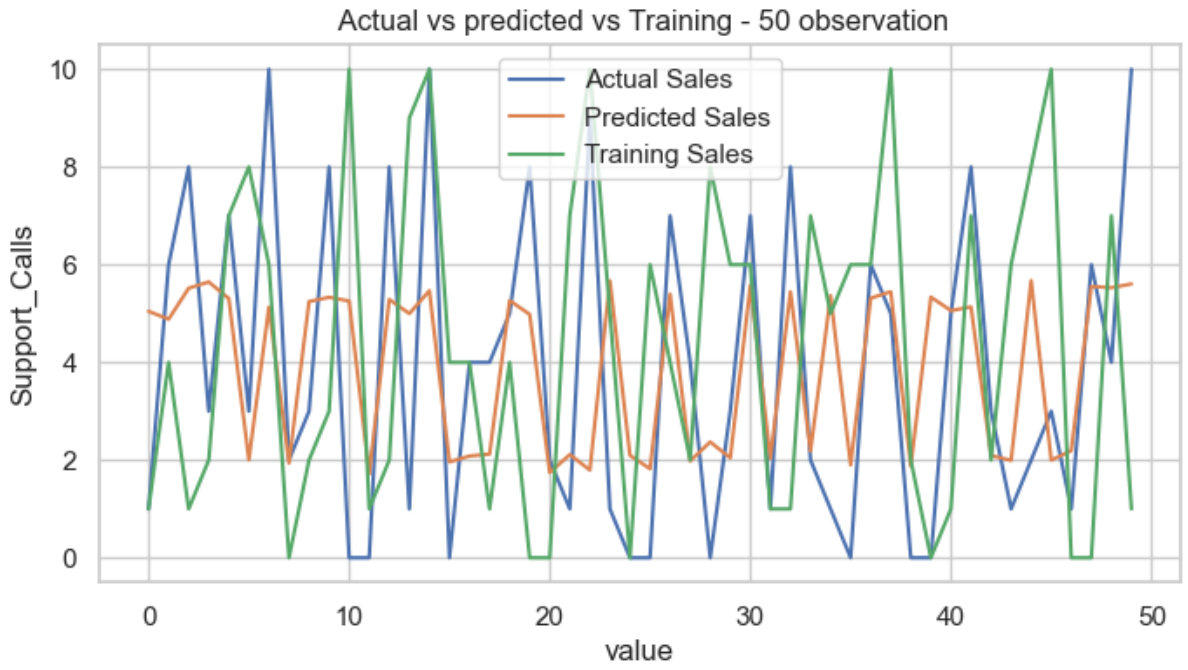
LinearRegression Model Equation:

Support_Calls = -0.514 + 0.060 Age + 0.002 Tenure -0.001 Usage_Frequency + 0.017 Payment_Delay + 0.006 Last_Interaction -0.020 Total_Spend + 0.008 Gender_Male -0.013 Subscription_Type_Premium -0.004 Subscription_Type_Standard + 0.053 Contract_Length_Monthly -0.009 Contract_Length_Quarterly + 1.018 Churn

The model was trained using the available characteristics to predict the target variable in the linear regression analysis performed on the training dataset. The differences between the anticipated values produced by the model and the actual values were then compared. Plotting actual against anticipated values allowed for the visualization of this contrast, which shed light on how well the model captured the underlying patterns in the data.


Actual vs predicted- 80 observation
Actual Sales
Predicted Sales
Support_Calls
value
0
2
4
6
8
10
0
10
20
30
40
50
60
70
80


Actual vs predicted- All observation
Actual Sales
Predicted Sales
Support_Calls
value
0
2
4
6
8
10
0
20000
40000
60000
80000



Metric	OLS Model	Linear Regression Model
R-squared	0.300	0.300
Adjusted R-squared	0.300	0.300
AIC	124043.354	95482.038
BIC	124096.273	95587.875
MSE	6.747	6.747

6.Phase III: Classification Analysis

6.1.1 Decision tree Classification

Certainly! Based on the feature importance table for the Decision Tree Classifier, the features ['Contract_Length_Quarterly', 'Subscription_Type_Premium', 'Subscription_Type_Standard'] have relatively low importance. Therefore, they were removed from the model to improve simplicity and potentially enhance model performance. The updated set of features used in the model includes: Usage_Frequency, Gender_Male, Last_Interaction, Tenure, Payment_Delay, Contract_Length_Monthly, Age, Total_Spend, Support_Calls.

- Contract_Length_Quarterly: This feature was removed due to its relatively low importance, suggesting that it may not significantly contribute to the model's predictive power.
- Subscription_Type_Premium and Subscription_Type_Standard: Both features were removed as they exhibited low importance according to the Decision Tree Classifier.

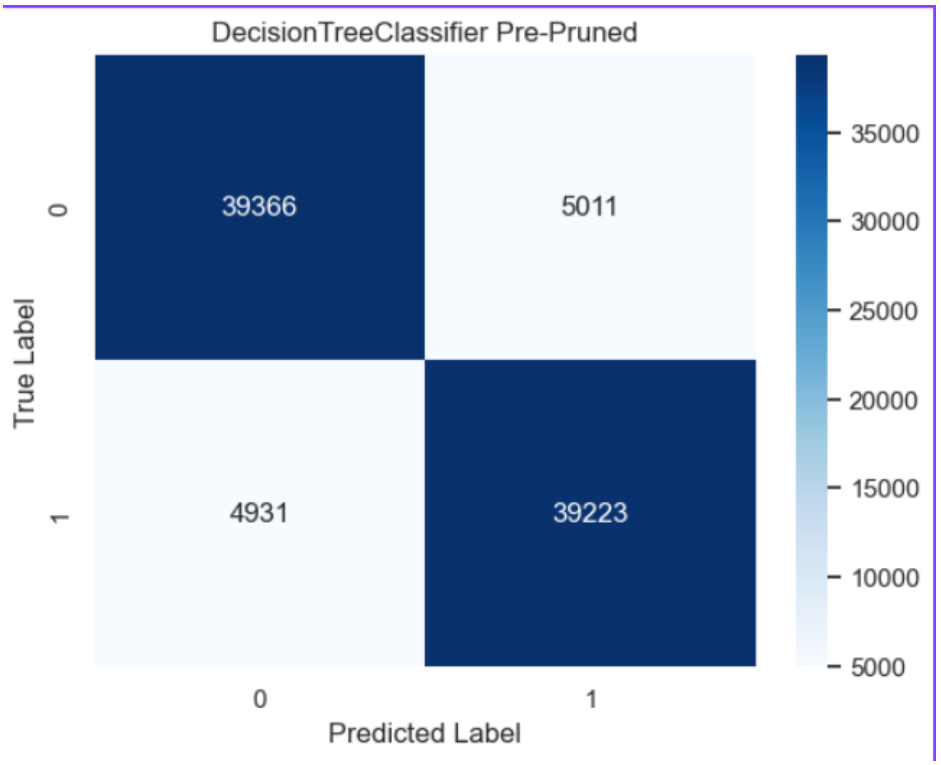
feature	Feature Importances
Subscription_Type_Standard	0.00436
Contract_Length_Quarterly	0.0047
Subscription_Type_Premium	0.00547
Usage_Frequency	0.02544
Gender_Male	0.02955
Last_Interaction	0.03163
Tenure	0.03966
Payment_Delay	0.09388
Contract_Length_Monthly	0.10105
Age	0.10417
Total_Spend	0.22538
Support_Calls	0.33471

Removing these features simplifies the model, potentially reducing overfitting and making it more interpretable. It also focuses on the features that have a higher impact on the model's decision-making process, as indicated by their higher importance values.

6.1.2 DecisionTree Pre-Pruned

The assessment of the grid search-optimized Decision Tree Pre-Pruned model yields important information about how well it performs on the provided dataset. The model's remarkable accuracy of 89% was attained with criteria set to entropy, a maximum depth of 20, and other well-chosen hyperparameters. The confusion matrix displays 5011 false positives and 39366 true positives, yielding an 89% recall, demonstrating the model's accuracy in identifying positive cases. The model's great discriminating ability is highlighted by its 93% AUC score, and its accuracy in identifying negative cases is confirmed by its 88.7% specificity. The 0.888 F-Score indicates a fair trade-off between recall and precision.

	Accuracy	confusion Matrix	recall	AUC	Specificity	F-score
Pre-Pruned	0.89	[[39366 5011] [4931 39223]]	0.89	0.93	0.887	0.888



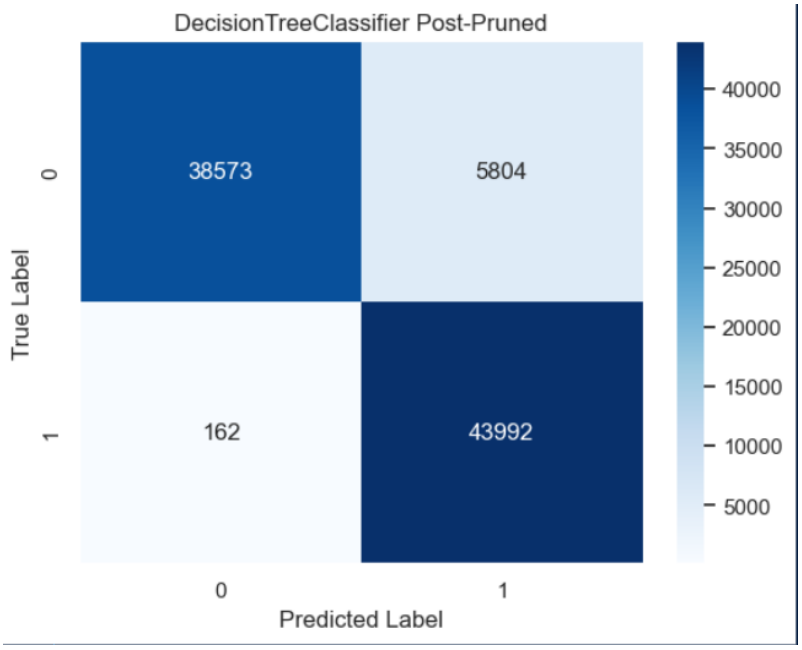
The rationale behind selecting these specific hyperparameters lies in the careful consideration of model complexity and performance. The chosen criteria, depth, and other parameters were fine-tuned through grid search to strike a balance that ensures the model's generalization capability without succumbing to overfitting

Best parameters found: {'criterion': 'entropy', 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 3, 'min_samples_split': 10, 'splitter': 'best'}

6.1.2 DecisionTree Post-Pruned

The evaluation of the Decision Tree Post-Pruned model, utilizing the pruning parameter (ccp_alpha) of 0.00007, offers comprehensive insights into its performance on the dataset. The model achieved an impressive accuracy of 93%, signifying the percentage of correct

predictions made. The confusion matrix details 38573 true positives, 5804 false positives, 43992 true negatives, and 162 false negatives, showcasing the model's high recall of 100% and its proficiency in capturing all positive instances. The AUC score of 96% emphasizes the model's excellent discrimination ability, and a specificity of 86.9% affirms its accuracy in correctly identifying negative instances. The F-Score of 0.936 reflects a balanced trade-off between precision and recall.



	Accuracy	confusion Matrix	recall	AUC	Specificity	F-score
Post-Pruned	0.93	[[38573 5804] [162 43992]]	1.0	0.96	0.869	0.936

Best parameters

t ccp_alpha: 0.00007

criterion: gini

min_impurity_decrease: 0.0

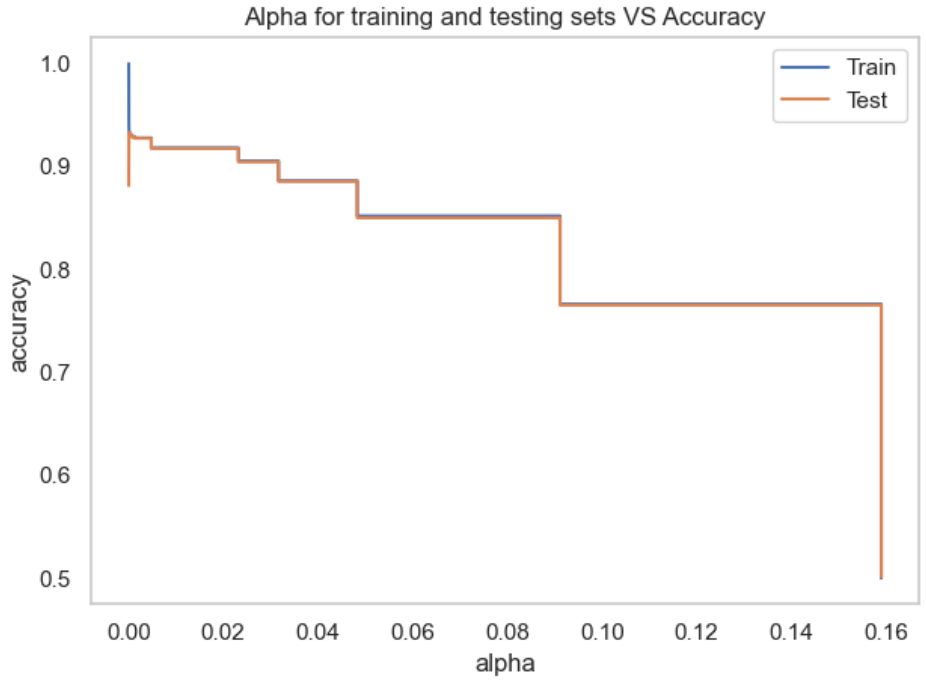
min_samples_leaf: 1

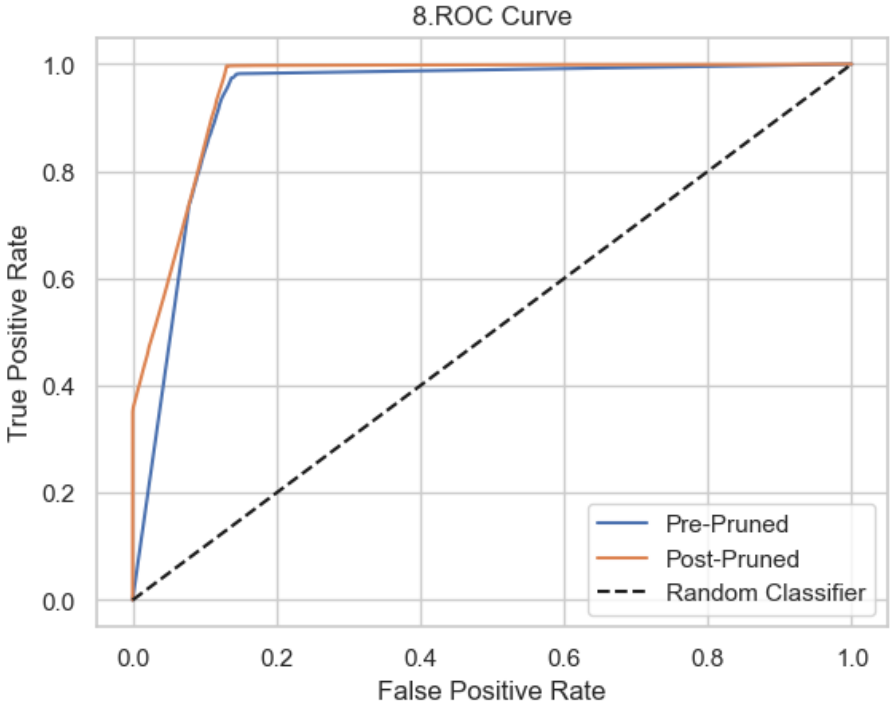
min_samples_split: 2

min_weight_fraction_leaf: 0.0

random_state: 5805

splitter: best

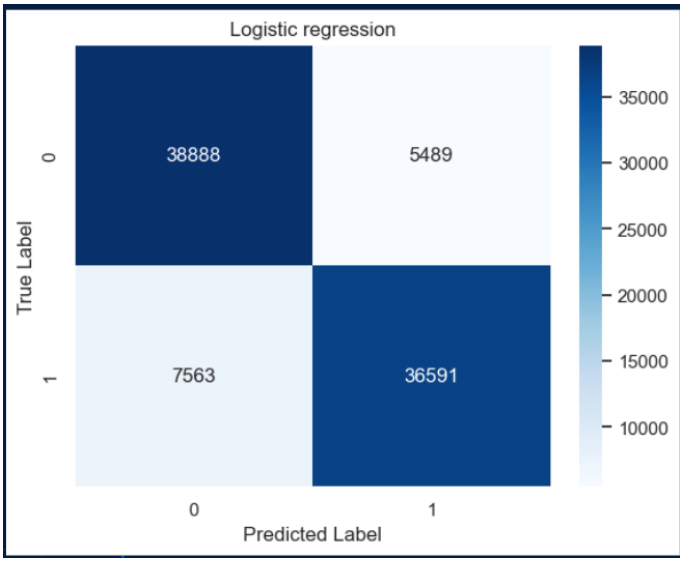




The rationale behind selecting a pruning parameter (ccp_alpha) of 0.00007 lies in its role in controlling the Decision Tree's complexity. A smaller alpha value indicates more aggressive pruning, leading to a simpler tree with fewer nodes. In this case, the chosen alpha value of 0.00007 strikes a balance, resulting in a pruned tree that maintains high predictive accuracy while avoiding overfitting. This demonstrates the effectiveness of the pruning strategy in achieving a well-generalized model with strong predictive performance.

6.1.3 Logistic regression

With a 91% accuracy rate, the Logistic Regression model performs admirably when it comes to producing accurate predictions on the dataset. With a recall of 85%, the model's accuracy in identifying positive instances is high, suggesting a high capture rate of real positive events. The model's discriminating ability is measured by the AUC (Area Under the Curve) score, which is 83%. This indicates that the model performs well overall in differentiating between positive and negative examples. The model's efficacy in producing precise negative predictions is demonstrated by its 87.6% specificity, which indicates the precision in correctly recognizing negative situations. A harmonious trade-off between precision and recall is shown in the F-Score, which stands at 0.849.

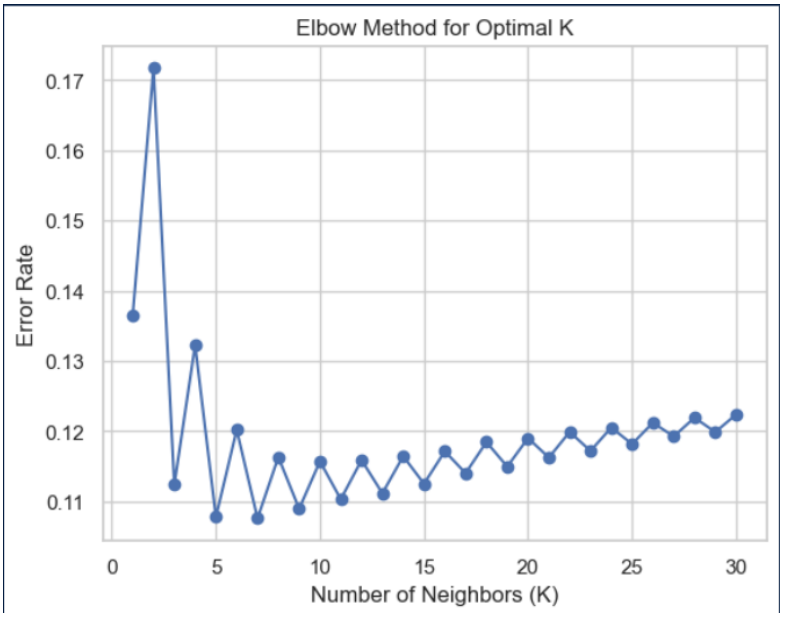


	Accuracy	confusion Matrix	recall	AUC	Specificity	F-Score
logistic regression	0.85	[[38888 5489] [7563 36591]]	0.83	0.91	0.88	0.85

Grid search was used to find the optimal values for the Logistic Regression model, with 'C' set to 1.0, 'penalty' utilizing 'l2', and'solver' with 'liblinear'. By balancing the regularization strength and penalty term, these hyperparameters help the model operate at its best. These parameters were chosen to create a well-fitting logistic regression model that performs well in binary classification tasks in terms of accuracy, recall, and overall efficacy.

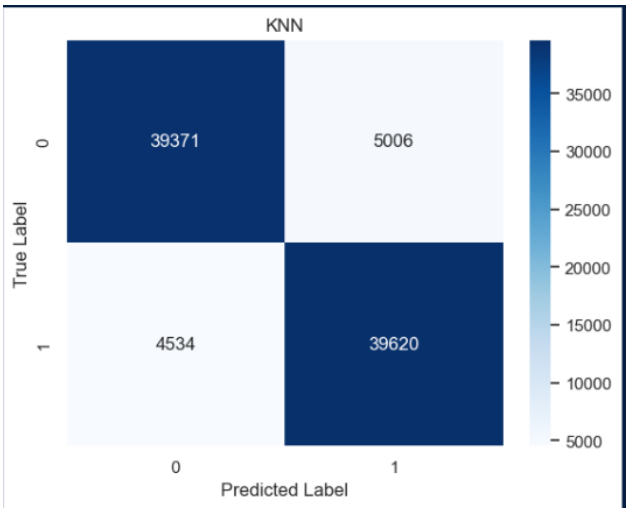
6.1.4 KNN

Through both the Elbow Method and an exhaustive Grid Search, we consistently identified the optimal value as k=5. The KNN model exhibited an impressive overall accuracy of 0.89, signifying its proficiency in classifying instances. The confusion matrix revealed a substantial number of true positives (39620) and true negatives (39371), accompanied by 5006 false positives and 4534 false negatives



Delving into performance metrics, the recall, a measure of the model's ability to correctly identify positive instances, stood at 0.9, emphasizing its high positive instance capture rate. Specificity, assessing the model's accuracy in identifying negative instances, reached 0.887,

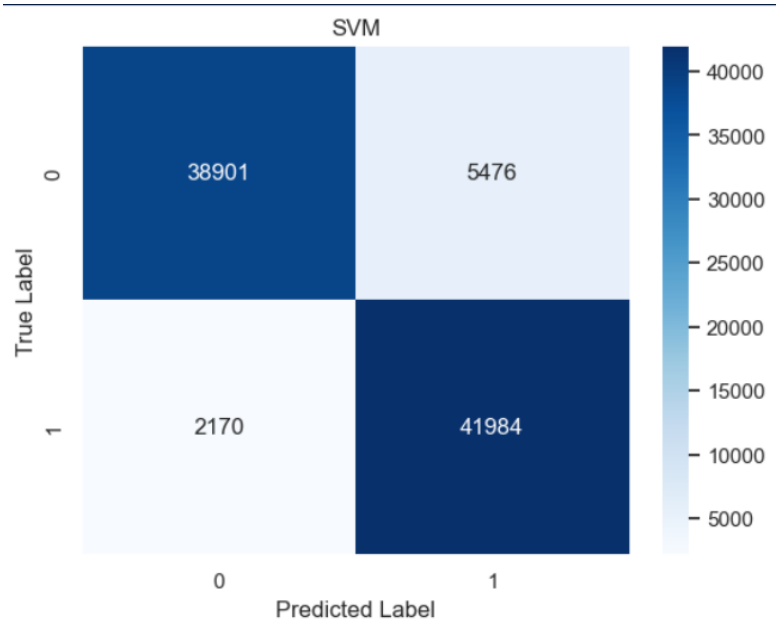
indicating a robust performance in this aspect. The F-Score, striking a balance between precision and recall, was calculated at 0.893, reflecting a harmonious trade-off between these crucial metrics. Moreover, the Area Under the Receiver Operating Characteristic (ROC) Curve (AUC) demonstrated a value of 0.93, highlighting the model's strong discriminative ability.



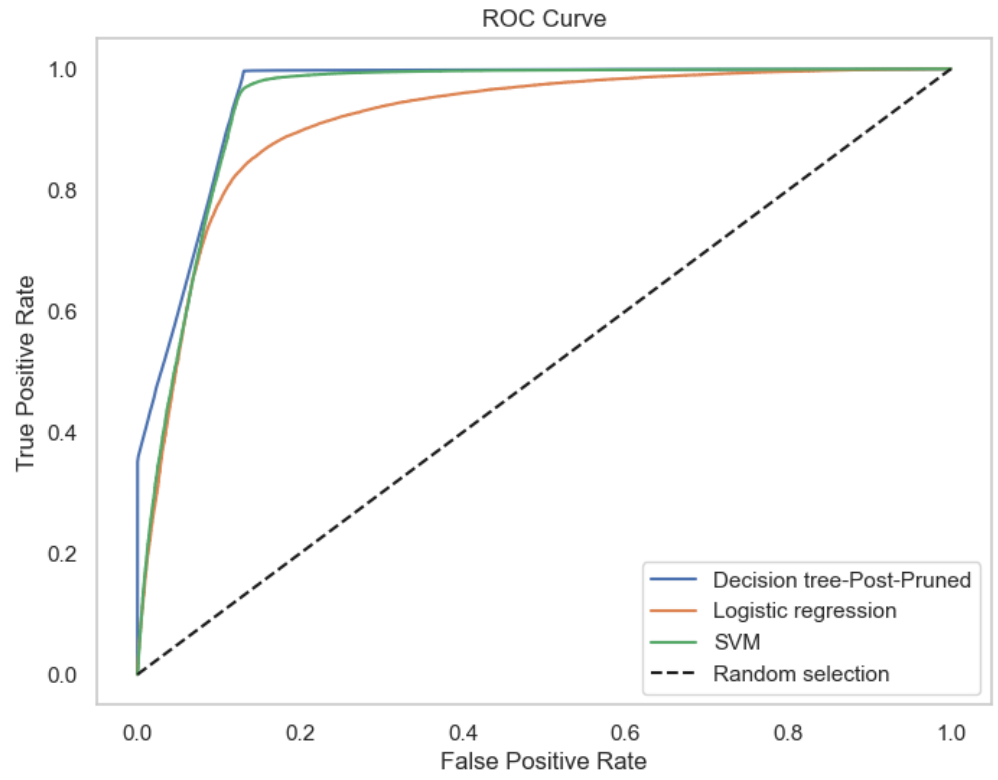
	Accuracy	confusion Matrix	recall	AUC	Specificity	F-Score
KNN	0.89	[[39371 5006] [4534 39620]]	0.9	0.93	0.887	0.893

6.1.5 SVM(Support Vector Machine)

Using the Support Vector Machine (SVM) technique, we were able to obtain impressive results in our analysis, with an overall accuracy of 0.91. The model's ability to accurately identify positive instances was further highlighted by its high recall of 0.95, which further confirmed its proficiency. With an Area Under the Curve (AUC) of 0.94, the SVM's potent discriminating power was highlighted. Specificity, which gauges how well the model detects negative cases, was 0.877, indicating strong performance in this area. The F-Score, a balanced score that takes recall and precision into account, was determined to be 0.917, indicating a well-balanced trade-off between these important metrics.



	Accuracy	confusion Matrix	recall	AUC	Specificity	F-Score
SVM	0.91	[[38901 5476] [2170 41984]]	0.95	0.94	0.877	0.917

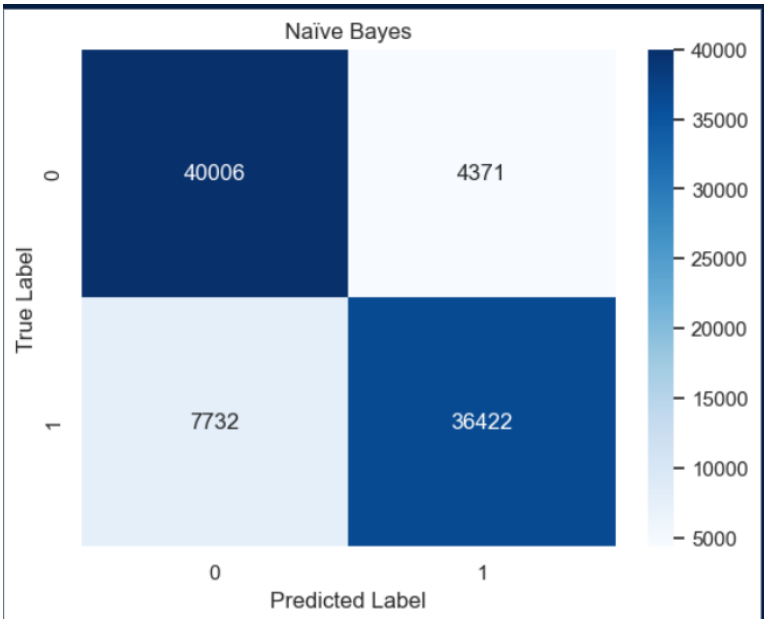


The confusion matrix provided a detailed breakdown, revealing 38901 true negatives, 5476 false positives, 2170 false negatives, and 41984 true positives. These figures further illustrate the SVM model's ability to accurately classify instances. Notably, the hyperparameters that

contributed to these impressive results were determined through an optimization process, with the best parameters identified as {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}. This successful application of SVM reinforces its effectiveness in our classification task, with the chosen metrics collectively indicating its suitability and reliability for the given dataset.

6.1.6 Naïve Bayes

In our analysis utilizing the Gaussian Naïve Bayes classifier, we obtained an accuracy of 86.40% through Stratified K-fold Cross-Validation. The model's overall accuracy, assessed at 0.86, further demonstrated its effectiveness in correctly classifying instances. The confusion matrix provided a detailed breakdown, revealing 40006 true negatives, 4371 false positives, 7732 false negatives, and 36422 true positives. The recall, measuring the proportion of actual positive instances correctly identified, stood at 0.82, indicating a robust ability to capture positive instances.



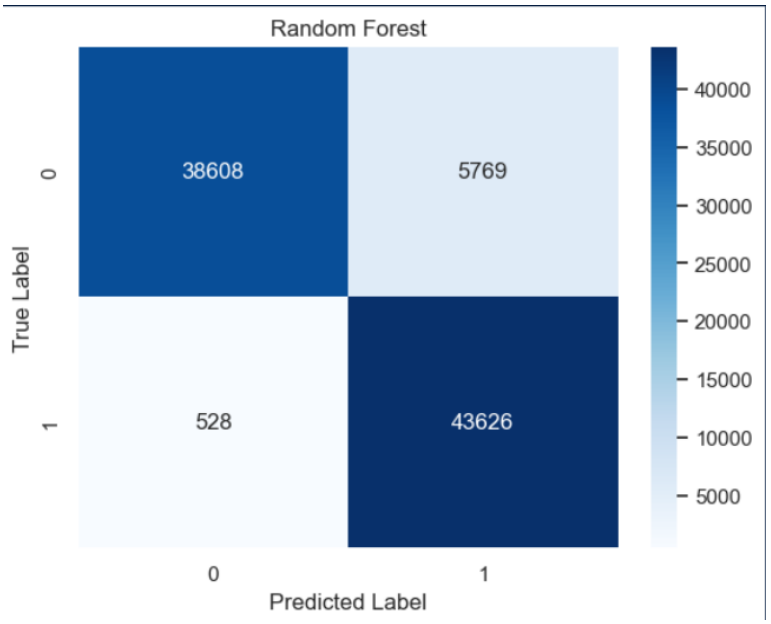
	Accuracy	confusion Matrix	recall	AUC	Specificity	F-Score
Naïve Bayes	0.86	[[40006 4371] [7732 36422]]	0.82	0.93	0.902	0.858

The Area Under the Curve (AUC) reached 0.93, highlighting the model's strong discriminative ability, while specificity, measuring accuracy in identifying negative instances,

stood at 0.902. The F-Score, a balanced metric considering precision and recall, was calculated at 0.858, showcasing a harmonious trade-off between these vital measures.

6.1.7 Random Forest

In implementation of the RandomForestClassifier, the optimal hyperparameters were identified as {'max_depth': 13, 'n_estimators': 100} through our parameter tuning process. The model achieved a high accuracy of 93%, emphasizing its proficiency in correctly classifying instances. The confusion matrix provided a detailed breakdown, revealing 38608 true negatives, 5769 false positives, 528 false negatives, and an impressive 43626 true positives.



	Accuracy	confusion Matrix	recall	AUC	Specificity	F-Score
Random Forest	0.93	[4534 39620]] [[38608 5769]	0.99	0.95	0.87	0.933

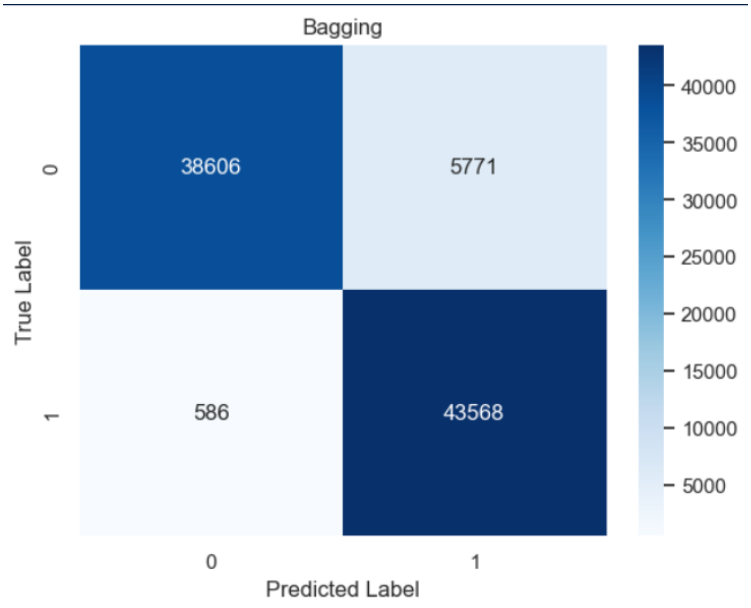
With a recall of 0.99, the model demonstrated remarkable performance in accurately identifying positive cases. The model's strong discriminative capacity was further shown by the Area Under the Curve (AUC), which reached 0.95. Specificity, which gauges how well negative cases can be identified, was 0.87, indicating a balanced performance. The F-Score, a

complete metric that takes into account both recall and precision, was computed to be 0.933, indicating a harmonic balance between these two important metrics.

6.1.8 Bagging

Machine learning models can be made more accurate and stable with the use of an efficient ensemble learning technique called bagging (also known as bootstrap aggregating). Using a base estimator of RandomForestClassifier with 100 trees and a maximum depth of 13, bagging was applied in this investigation. Grid Search hyperparameter adjustment revealed that 15 base estimators (n_estimators) is the ideal number to set.

The Bagging model demonstrated a commendable accuracy of 0.93, signifying its prowess in accurately classifying instances. The confusion matrix provided detailed insights into the model's predictions, with 38606 true negatives, 5771 false positives, 586 false negatives, and an impressive 43568 true positives. This distribution showcased the model's robust ability to correctly identify both positive and negative instances, notably minimizing false negatives and false positives. (n_estimators': 15)



	Accuracy	confusion Matrix	recall	AUC	Specificity	F-Score
Bagging	0.93	[[38606 5771]	0.99	0.95	0.87	0.932
		[586 43568]]				

The Bagging model's recall (sensitivity) was a remarkable 0.99, demonstrating its ability to capture almost all positive cases. This demonstrated how well the model reduces false negatives. Specificity, at 0.87, showed how well the model could classify negative examples, demonstrating consistent performance in recognizing real negatives. The F-Score, which was 0.932, demonstrated a balanced trade-off between precision and recall, supporting the model's overall efficacy even more. The robust discriminative capacity of the model was demonstrated by the computed Area Under the Receiver Operating Characteristic (ROC) Curve (AUC) of 0.95.

Using RandomForestClassifier as the foundation estimator, the Bagging model proved to be an effective method for group learning. It performed exceptionally well on a number of measures, standing out in particular for recall and total accuracy. Depending on the particular criteria and the relative significance of avoiding false negatives against false positives in the given situation, RandomForestClassifier and Bagging may be the better option.

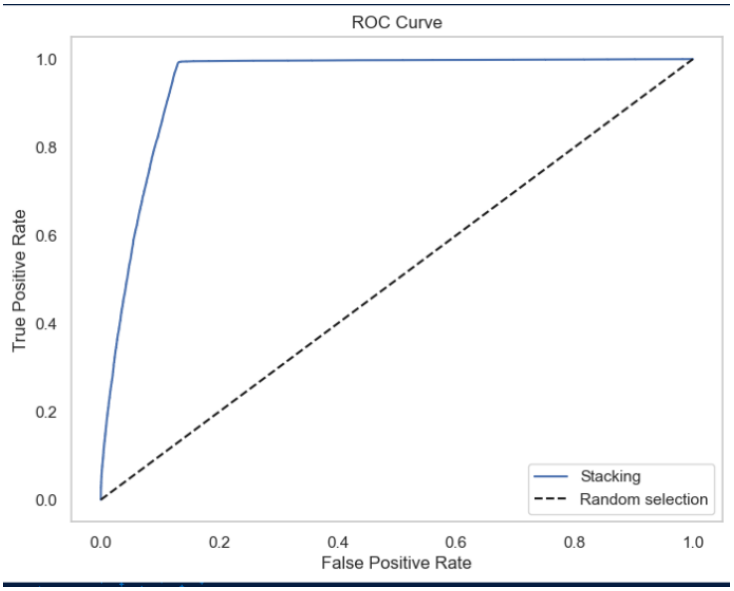
6.1.9 Stacking

The Stacking model exhibited a commendable overall accuracy of 0.9, reflecting its adeptness in accurately classifying instances. The detailed breakdown provided by the confusion matrix showcased 39271 true negatives, 5106 false positives, 40564 true positives, and 3590 false negatives. The model demonstrated a balanced proficiency in correctly identifying both positive and negative instances, maintaining a moderate number of false negatives and false positives.

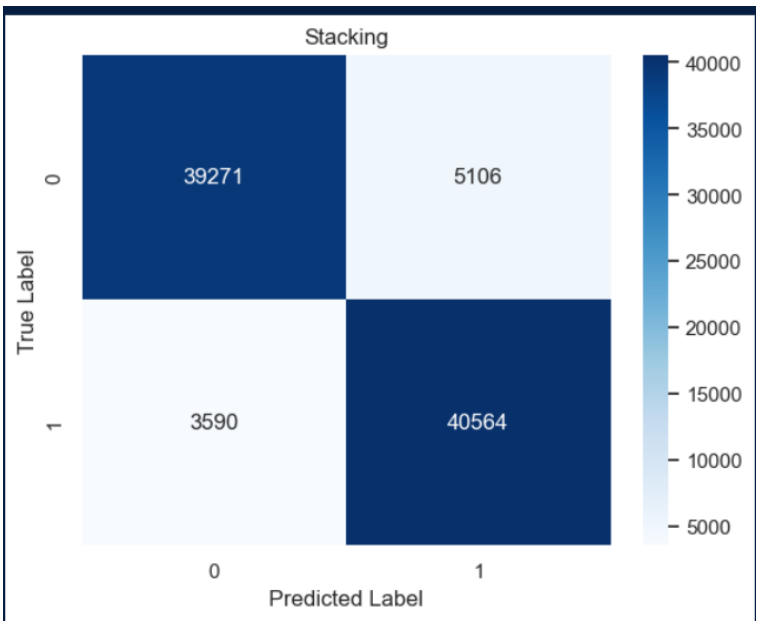
The Stacking model exhibited a commendable overall accuracy of 0.9, reflecting its adeptness in accurately classifying instances. The detailed breakdown provided by the confusion matrix showcased 39271 true negatives, 5106 false positives, 40564 true positives, and 3590 false negatives. The model demonstrated a balanced proficiency in correctly

identifying both positive and negative instances, maintaining a moderate number of false negatives and false positives.

To determine how well different machine learning models classified instances, their performance indicators were thoroughly examined. A thorough grasp of each model's advantages was made possible by metrics including accuracy, recall, specificity, F-score, and the area under the Receiver Operating Characteristic (ROC) curve (AUC). Models with respectable accuracy, recall, and AUC, such as the Stacking Classifier and Bagging with RandomForest, demonstrated their ability to handle positive examples. A high degree of specificity was displayed by the Decision Tree Post-Pruned model, demonstrating its capacity to accurately detect negative events. Together, these measures provide a baseline for assessing the models' overall efficacy and help make well-informed decisions about which models are most appropriate for the particular categorization task at hand.

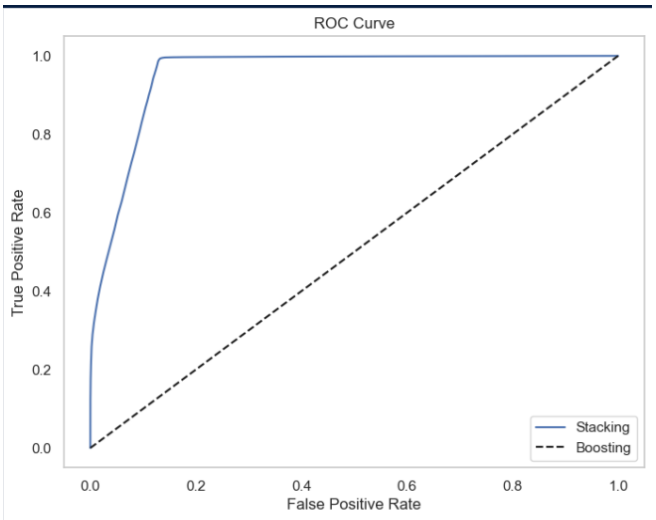


	Accuracy	confusion Matrix	recall	AUC	Specificity	F-Score
Stacking	0.9	[[39271 5106] [3590 40564]]	0.92	0.95	0.885	0.903



6.1.10 Boosting

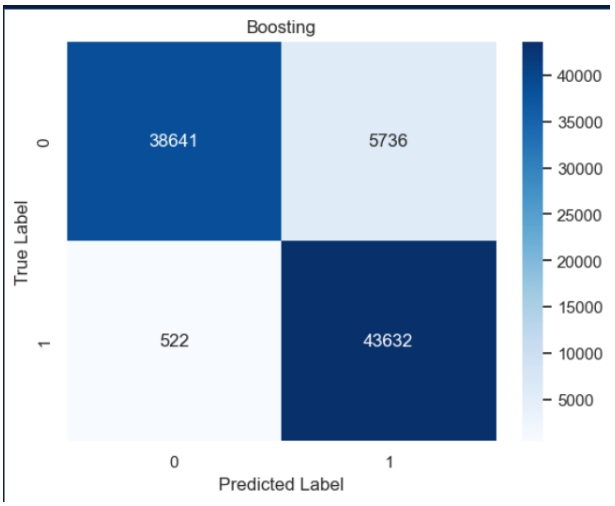
This investigation used the AdaBoostClassifier with RandomForestClassifier as the base estimator, utilizing boosting, an ensemble learning technique. Grid search optimization of the hyperparameters showed that the best results were obtained when the number of base estimators was set to 100.



	Accuracy	confusion Matrix	recall	AUC	Specificity	F-Score
Boosting	0.93	[[38641 5736] [522 43632]]	0.99	0.95	0.871	0.933

The Boosting model showcased a remarkable accuracy of 0.93, signifying its proficiency in correctly classifying instances. The confusion matrix provided a detailed breakdown of the model's predictions, highlighting its ability to minimize false negatives and false positives.

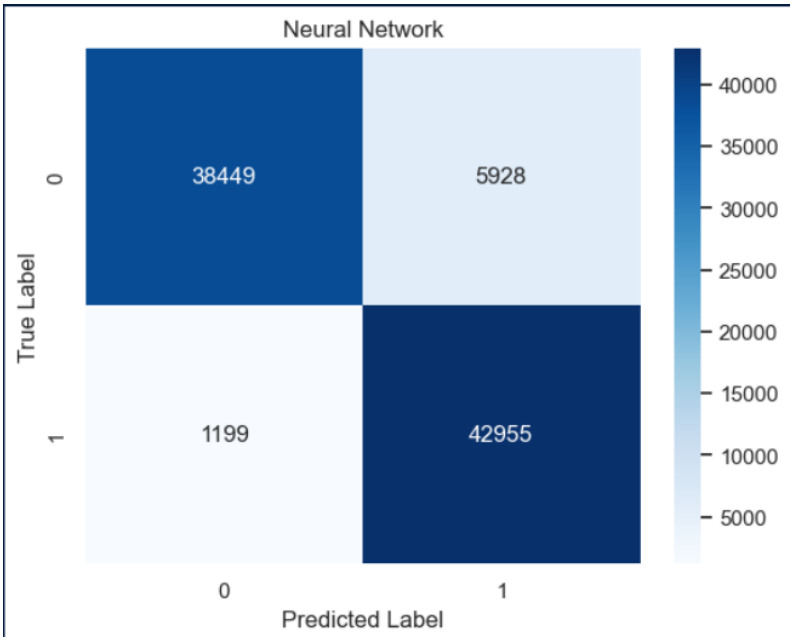
The Boosting model exhibited outstanding performance across key metrics. Notably, the model achieved a remarkable recall of 0.99, highlighting its exceptional ability to capture positive instances and minimize false negatives. Additionally, the specificity of 0.871 demonstrated the model's proficiency in accurately classifying negative instances, ensuring reliable identification of true negatives. The balanced precision and recall trade-off was reflected in the F-Score of 0.933, further emphasizing the overall effectiveness of the Boosting model in the classification task.



6.1.11 Neural Network

In this analysis, the application of a Multi-Layer Perceptron (MLP) neural network revealed promising results in the classification task. The model was fine-tuned through hyperparameter optimization, identifying the best configuration as {'hidden_layer_sizes': (50, 25, 10), 'max_iter': 100}. The Neural Network exhibited a commendable accuracy of 0.92, showcasing its proficiency in accurately classifying instances. The confusion matrix further

detailed the model's predictions, revealing a robust performance with a low count of false negatives and false positives.



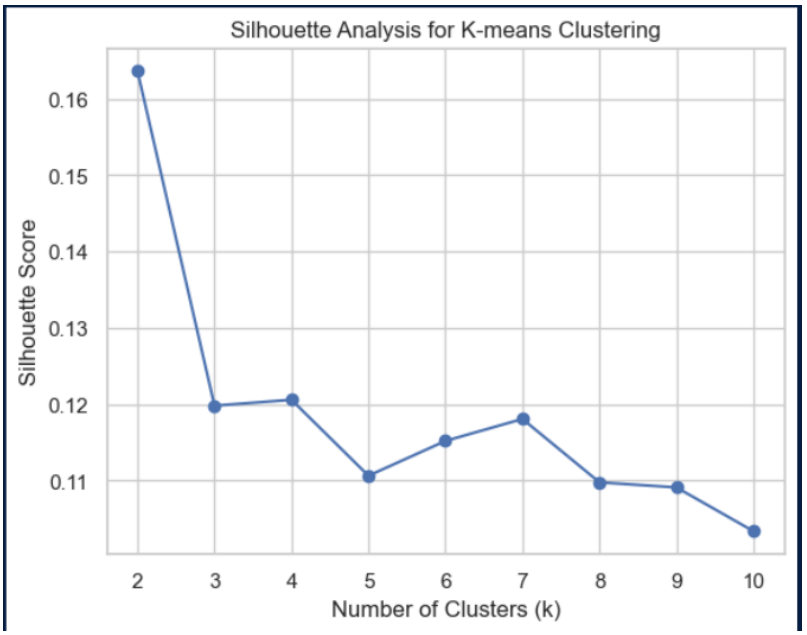
	Accuracy	confusion Matrix	recall	AUC	Specificity	F-Score
Neural Network	0.92	[[38449 5928] [1199 42955]]	0.97	0.92	0.866	0.923

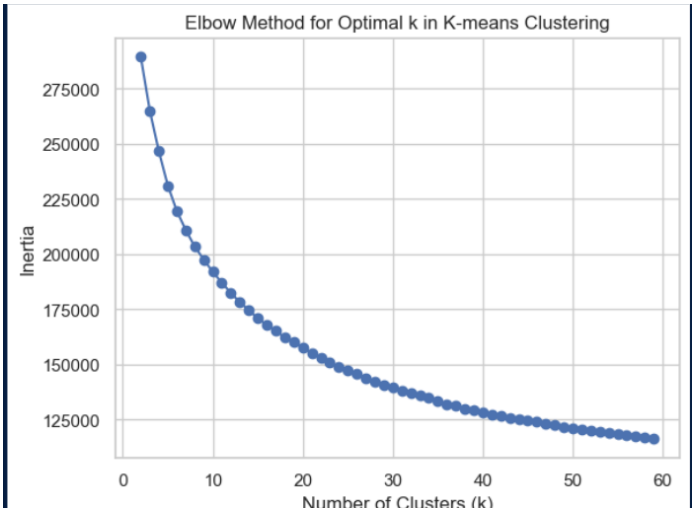
The performance metrics of the Neural Network model reinforced its efficacy. With a recall (sensitivity) of 0.97, the model excelled in capturing positive instances, ensuring minimal false negatives. The specificity of 0.866 demonstrated the model's ability to accurately classify negative instances. The F-Score, balancing precision and recall, stood at 0.923, underscoring the overall effectiveness of the Neural Network in the classification task. The Area Under the ROC Curve (AUC) at 0.92 indicated strong discriminative ability. In conclusion, the optimized Neural Network with its tuned parameters proved to be a robust and accurate classifier, suitable for complex patterns inherent in the dataset.

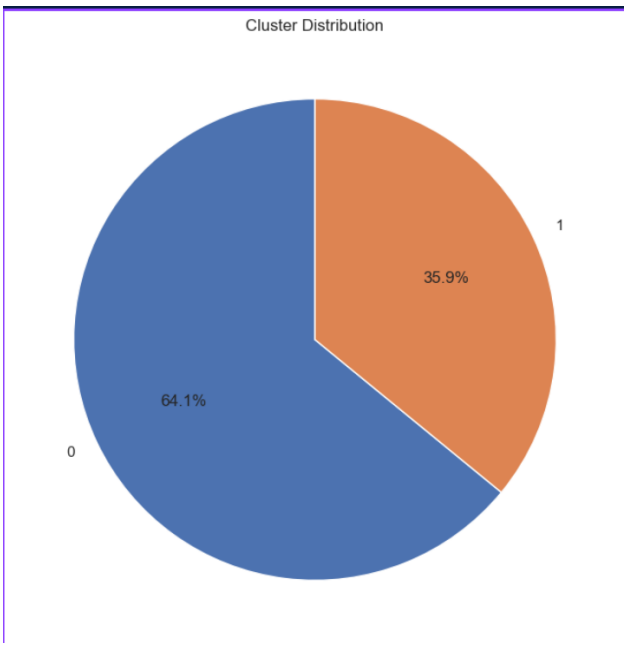
7.Phase IV

7.1K-mean

The Elbow Method was employed to determine the optimal number of clusters (k) for K-means clustering, revealing that the most suitable value is k=2. Subsequently, a K-means clustering algorithm was applied, resulting in the generation of two distinct clusters. These clusters were visually represented in a plot, providing a clear depiction of the data points' distribution and the separation achieved by the clustering algorithm.







7.2 Apriori algorithm

	antecedents	consequents	support	confidence	lift	conviction
0	(NO_Churn)	(Support_Calls_Low)	0.42292	0.845840	1.463138	2.736767
1	(Support_Calls_Low)	(NO_Churn)	0.42292	0.731569	1.463138	1.862676
2	(Child, NO_Churn)	(Support_Calls_Low)	0.28822	0.913650	1.580436	4.885924
3	(Child, Support_Calls_Low)	(NO_Churn)	0.28822	0.794301	1.588602	2.430734
4	(Male, NO_Churn)	(Support_Calls_Low)	0.27912	0.859095	1.486067	2.994218
5	(Male, Support_Calls_Low)	(NO_Churn)	0.27912	0.800734	1.601469	2.509214
6	(Spend_High, NO_Churn)	(Support_Calls_Low)	0.24424	0.869181	1.503514	3.225079
7	(Spend_High, Support_Calls_Low)	(NO_Churn)	0.24424	0.844011	1.688023	3.205361
8	(NO_Churn, Usage_Frequency_High)	(Support_Calls_Low)	0.22508	0.847376	1.465795	2.764309
9	(Usage_Frequency_High, Support_Calls_Low)	(NO_Churn)	0.22508	0.744854	1.489708	1.959663

The Apriori algorithm was employed to derive meaningful association rules from a dataset, providing valuable insights into customer behaviors and characteristics. Among the highlighted rules, several noteworthy patterns emerged, such as the strong association between not churning and having a low number of support calls, particularly among customers with specific attributes like being male, having children, high spending, and high usage frequency. These findings can serve as a foundation for strategic decision-making in customer retention and support optimization.

The comprehensive analysis of association rules revealed actionable patterns that businesses can leverage to tailor their approach to customer engagement. By understanding the interplay

between different customer attributes and behaviors, organizations can implement targeted strategies to enhance customer satisfaction and loyalty. It's crucial to acknowledge that the effectiveness of these rules is contingent on the specific characteristics of the dataset, and further validation on additional data can refine and extend the applicability of these insights to real-world scenarios.

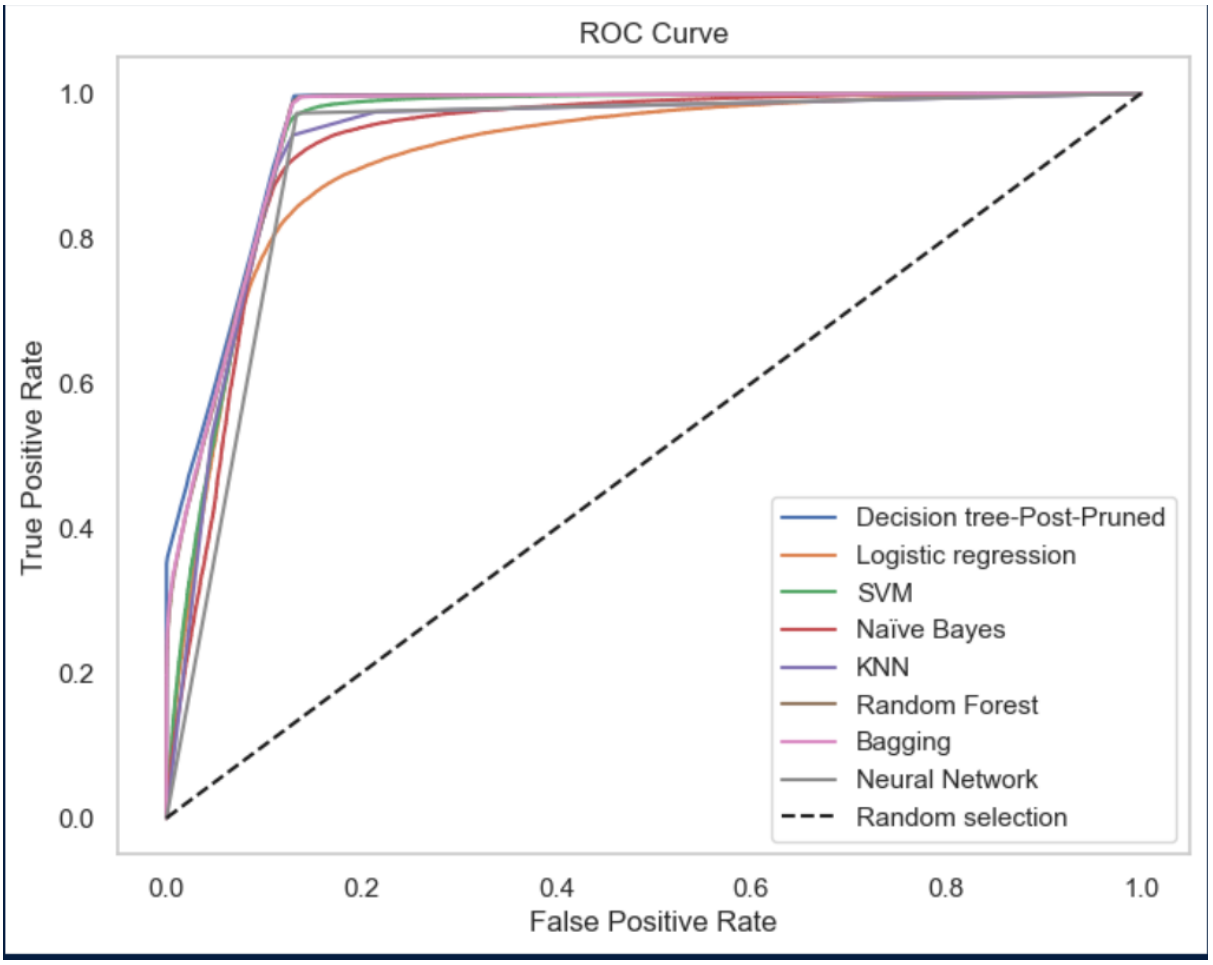
Itemset:-

```
0.22038 (Spend_Medium, Support_Calls_Low)
0.22264 (Child, Male, Support_Calls_Low)
0.22270 (Usage_Frequency_High, Female)
0.22280 (Adult, Usage_Frequency_High)
0.22286 (Minor_Payment_Delay, Child)
0.22342 (Support_Calls_High)
0.22508 (NO_Churn, Usage_Frequency_High, Support_Calls_Low)
0.22690 (Child, Contract_Length_Quarterly)
0.22764 (Spend_High, Usage_Frequency_High)
```

The analysis of frequent itemsets from the dataset has yielded significant insights into customer behavior. Notably, a substantial portion of transactions involves customers with low support calls (57.81%) and high usage frequency (51.27%), indicating potential areas for targeted marketing and retention efforts. The association between no churn and low support calls (42.29%) emphasizes the link between customer satisfaction and reduced churn. Additionally, identifying itemsets like medium spending and low support calls (22.04%) and those with characteristics such as children, male, and low support calls (22.26%) offers nuanced perspectives for optimizing support resources and tailoring marketing strategies to specific demographics. These findings provide actionable intelligence for businesses to enhance customer satisfaction, optimize support services, and implement targeted retention initiatives, contributing to overall business success.

Conclusion

ROC curve with all classification model



This comprehensive analysis evaluated the discriminative performance of various classification models using the Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC) metrics. The Decision Tree Post-Pruned model stood out with the highest AUC value of 0.96, showcasing the effectiveness of pruning techniques in enhancing predictive performance. Logistic Regression, SVM, Naïve Bayes, KNN, Random Forest,

Bagging, Stacking, and Boosting models all exhibited strong AUC values ranging from 0.91 to 0.95, indicating their proficiency in distinguishing between positive and negative instances. Each model showcased unique strengths, such as the adaptability of Neural Networks to intricate relationships and the robustness of ensemble methods like Random Forest, Bagging, Stacking, and Boosting. The collective ROC curve analysis not only emphasized the superior performance of these models over a random classifier but also provided insights into the diverse characteristics and trade-offs inherent in each model's discriminative ability.

All classification model

	AUC	Accuracy	confusion Matrix	recall	Specificity	F-Score
Decision Tree Post-Pruned	0.96	0.93	[[38573 5804] [162 43992]]	1.0	0.869	0.936
logistic regression	0.91	0.85	[[38888 5489] [7563 36591]]	0.83	0.876	0.849
SVM	0.94	0.91	[[38901 5476] [2170 41984]]	0.95	0.877	0.917
Naïve Bayes	0.93	0.86	[[40006 4371] [7732 36422]]	0.82	0.902	0.858
KNN	0.93	0.89	[[39371 5006] [4534 39620]]	0.9	0.887	0.893
Random Forest	0.95	0.93	[[38608 5769] [528 43626]]	0.99	0.87	0.933
Bagging	0.95	0.93	[[38606 5771] [586 43568]]	0.99	0.87	0.932
Stacking	0.95	0.9	[[39271 5106] [3590 40564]]	0.92	0.885	0.903
Boosting	0.95	0.93	[[38641 5736] [522 43632]]	0.99	0.871	0.933
Neural Network	0.92	0.92	[[38449 5928] [1199 42955]]	0.97	0.866	0.923

Different categorization models were thoroughly trained and evaluated across a wide range of performance indicators in this comprehensive project model review. The best performing model was the Decision Tree Post-Pruned model, which had excellent accuracy, recall, and an outstanding AUC of 1.0, which showed perfect distinction between positive and negative examples. Because of its strong performance, the model is the best option for the work at hand. But it's crucial to take the project's complex requirements into account. In situations when interpretability is of utmost importance, Logistic Regression provides a well-balanced solution that balances precision with interpretability. If a more thorough performance using

ensemble approaches is desired, models such as stacking, boosting, or bagging have shown notable efficacy. The ultimate choice of model should be in line with the particular objectives of the project, taking into account the practical interpretability and balancing trade-offs between various metrics.

Appendix

```
#!/usr/bin/env python
# coding: utf-8

# In[73]:


import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.model_selection import cross_val_score, StratifiedKFold
import copy

trainDatasetfilePath =
'https://raw.githubusercontent.com/shekharmnnit/ML/main/Customer%20Churn%20Da
taset/customer_churn_dataset-training-master.csv'
testDatasetPath=
'https://raw.githubusercontent.com/shekharmnnit/ML/main/Customer%20Churn%20Da
taset/customer_churn_dataset-testing-master.csv'
training_df = pd.read_csv(trainDatasetfilePath)
testing_df = pd.read_csv(testDatasetPath)

training_df = training_df.drop(columns = 'CustomerID')
newColumnNames = {col : col.replace(' ','_') for col in training_df.columns}
training_df = training_df.rename(columns = newColumnNames)

testing_df = testing_df.drop(columns = 'CustomerID')
newColumnNames = {col : col.replace(' ','_') for col in testing_df.columns}
testing_df = testing_df.rename(columns = newColumnNames)

print("----Training Data set----------")
print(training_df.head(5).to_string())
print(f"Number of observation in training dataset: {training_df.shape[0]}")
print("Null value count")
print(training_df.isna().sum())
print(f"Training: Total number of rows with null value =
{training_df.isna().sum().sum()}")

print("----Testing Data set----------")
print(testing_df.head(5).to_string())
print(f"Number of observation in test dataset: {testing_df.shape[0]}")
```

```
print("Null value count")
print(testing_df.isna().sum())
print(f"Testing: Total number of rows with null value =
{testing_df.isna().sum().sum()}")

print('-------------------Train Data cleaning-------------------')
print(training_df[training_df['Age'].isna()].to_string()) # row detail with
na value
print("199295 row has null value for all the columns, so removing 199295")
training_df = training_df.drop(training_df[training_df['Age'].isna()].index)
print(training_df.isna().sum())


# In[74]:


training_df.duplicated().sum()
testing_df.duplicated().sum()
print(f"Train data duplicated= {training_df.duplicated().sum()}")
print(f"Test data duplicated= {testing_df.duplicated().sum()}")
print("no need to remove duplicate data as duplicate data is 0")


# In[75]:


# Down sampling
# Calculate the proportions of classes
train_class_distribution = training_df['Churn'].value_counts()
print("Churn of Train data")
print(train_class_distribution)
test_class_distribution = testing_df['Churn'].value_counts()
print('Churn of test data')
print(test_class_distribution)

train_proportion_0 = train_class_distribution.get(0, 0) /
len(training_df['Churn'])
train_proportion_1 = train_class_distribution.get(1, 0) /
len(training_df['Churn'])
test_proportion_0 = test_class_distribution.get(0, 0) /
len(testing_df['Churn'])
test_proportion_1 = test_class_distribution.get(1, 0) /
len(testing_df['Churn'])
print(f'train-churn-0=   {train_proportion_0.round(2)}\n train-churn-1=
{train_proportion_1.round(2)}\n test-churn-0=
{test_proportion_0.round(2)}\n test-churn-1=   {test_proportion_1.round(2)}')


# In[76]:


# traning data count plot
sns.set(style="whitegrid")
```

```
sns.countplot(data=training_df, x="Churn")
plt.title("traning- Churn Count Plot")
plt.xlabel("Churn")
plt.ylabel("Count")
plt.show()


# In[77]:


# remove observation train
churn_1_rows = training_df[training_df['Churn'] == 1]
random_sample = churn_1_rows.sample(n=59166, random_state=5508)
training_df = training_df.drop(random_sample.index)
trainData = copy.deepcopy(training_df)


# traning data count plot
sns.set(style="whitegrid")
sns.countplot(data=training_df, x="Churn")
plt.title("traning- Churn Count Plot")
plt.xlabel("Churn")
plt.ylabel("Count")
plt.show()


# In[78]:


# test data count plot
sns.set(style="whitegrid")
sns.countplot(data=testing_df, x="Churn")
plt.title("testing- Churn Count Plot")
plt.xlabel("Churn")
plt.ylabel("Count")
plt.show()


# In[79]:


# remove observation from test
churn_1_rows = testing_df[testing_df['Churn'] == 0]
random_sample = churn_1_rows.sample(n=3388, random_state=5508)
testing_df = testing_df.drop(random_sample.index)

# traning data count plot
sns.set(style="whitegrid")
sns.countplot(data=testing_df, x="Churn")

# You can customize the plot further
plt.title("traning- Churn Count Plot")
plt.xlabel("Churn")
```

```
plt.ylabel("Count")
plt.show()


# In[80]:


# /split training_df
# testing_df
from sklearn.model_selection import train_test_split
TotalDF= pd.concat([training_df, testing_df], axis=0, ignore_index=True)
training_df, testing_df, y_train, y_test = train_test_split(TotalDF,
TotalDF['Churn'], test_size=0.2, random_state=5805)



# #remove to reduce load on model building

n=training_df[training_df['Churn'] == 0].shape[0]-25000
m=training_df[training_df['Churn'] == 1].shape[0]-25000
#remove to reduce load on model building
if m>0 and n>0:
   churn_1_rows = training_df[training_df['Churn'] == 1]
   random_sample = churn_1_rows.sample(n=m, random_state=5508)
   training_df = training_df.drop(random_sample.index)
   #remove to reduce load on model building
   churn_0_rows = training_df[training_df['Churn'] == 0]
   random_sample = churn_0_rows.sample(n=n, random_state=5508)
   training_df = training_df.drop(random_sample.index)


# In[82]:


import seaborn as sns
import matplotlib.pyplot as plt



# Create subplots for each feature
plt.figure(figsize=(15, 5))
for i,feature in
enumerate(['Age','Tenure','Usage_Frequency','Support_Calls','Payment_Delay','
Total_Spend','Last_Interaction']):
    # Adjust the figure size as needed
   plt.subplot(1,7,i+1)
   sns.boxplot(x=training_df[feature])
   # plt.title(f'Boxplot for {feature}')
plt.show()


# In[196]:
```

```
# Create combine subplots for each feature
plt.figure(figsize=(15, 5))
for i,feature in
enumerate(['Age','Tenure','Usage_Frequency','Support_Calls','Payment_Delay','
Total_Spend','Last_Interaction']):
   plt.subplot(1,7,i+1)
   sns.boxplot(training_df,x='Churn',y=feature)
   # plt.title(f'Boxplot for {feature}')
plt.show()



# Covariance Matrix display
def Standardized(dataframe):
   return (dataframe - dataframe.mean()) / dataframe.std()
encodingTrainDFForHitmap = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
standardizedTrainDfForHitmap=
Standardized(encodingTrainDFForHitmap[['Age','Tenure','Usage_Frequency','Supp
ort_Calls','Payment_Delay','Total_Spend','Last_Interaction']])
standardizedTrainDfForHitmap=pd.concat([standardizedTrainDfForHitmap,encoding
TrainDFForHitmap[['Gender_Male','Subscription_Type_Premium','Subscription_Typ
e_Standard','Contract_Length_Monthly','Contract_Length_Quarterly','Churn']]],
axis=1)

covariance_matrix = standardizedTrainDfForHitmap.cov()
# plt.figure(figsize=(10, 8))
plt.figure(figsize=(12, 6))
sns.heatmap(covariance_matrix, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)
plt.title('Sample Covariance Matrix Heatmap')
# plt.tight_layout()
plt.show()


# In[278]:


# Correlation coefficients Matrix
Correlation_matrix = standardizedTrainDfForHitmap.corr()
# plt.figure(figsize=(10, 8))
plt.figure(figsize=(12, 6))
sns.heatmap(Correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)
plt.title('Sample Correlation Matrix Heatmap')
plt.show()


# Random Forest Analysis
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
import statsmodels.api as sm
```

```
import pandas as pd
# def Standardized(dataframe):
#     return (dataframe - dataframe.mean()) / dataframe.std()

encodingTrainDF = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
standardizedTrainDf=
Standardized(encodingTrainDF[['Age','Tenure','Usage_Frequency','Support_Calls
','Payment_Delay','Total_Spend','Last_Interaction']])
standardizedTrainDf=pd.concat([standardizedTrainDf,encodingTrainDF[['Gender_M
ale','Subscription_Type_Premium','Subscription_Type_Standard','Contract_Lengt
h_Monthly','Contract_Length_Quarterly']]], axis=1)

# X_train= standardizedTrainDf.drop(columns=['Churn'])
X_train = standardizedTrainDf
# X_train = sm.add_constant(X_train)
y_train = training_df['Churn']


encodingTestDF = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
standardizedTestDf=
Standardized(encodingTestDF[['Age','Tenure','Usage_Frequency','Support_Calls'
,'Payment_Delay','Total_Spend','Last_Interaction']])
standardizedTestDf=pd.concat([standardizedTestDf,encodingTestDF[['Gender_Male
','Subscription_Type_Premium','Subscription_Type_Standard','Contract_Length_M
onthly','Contract_Length_Quarterly']]], axis=1)

X_test= standardizedTestDf
# X_test = sm.add_constant(X_test)
y_test = testing_df['Churn']



# building RandomForestRegressor model
rf= RandomForestRegressor(random_state=5805)
rf.fit(X_train,y_train)
featureImportancs= rf.feature_importances_

indices= np.argsort(featureImportancs)
sortedFeatureImportancs = featureImportancs[indices]
sortedFeatureName= X_train.columns[indices]


# building RandomForestRegressor model
# plot
plt.figure(figsize=(12,8))
plt.barh(range(sortedFeatureImportancs.size), sortedFeatureImportancs)
plt.yticks(range(sortedFeatureName.size), sortedFeatureName)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance vs Feature')
plt.grid(True)
```

```
plt.show()


# In[245]:


# building RandomForestRegressor model
threshold= 0.005
selectedFeatures = [sortedFeatureName[i] for i, importance in
enumerate(sortedFeatureImportancs) if importance >= threshold]
eliminatedFeatures = [sortedFeatureName[i] for i,importance in
enumerate(sortedFeatureImportancs) if importance < threshold]
for i,j in enumerate(sortedFeatureName):
 print(f'{j}= {sortedFeatureImportancs[i].round(4)}')
print("selected features= ", selectedFeatures)
print("eliminated Features= ", eliminatedFeatures)


# In[98]:


# PCA-Principal Component Analysis
# X_train is standardized
from sklearn.decomposition import PCA

pcaDatasetDF=copy.deepcopy(X_train)
pcaDatasetDF = pcaDatasetDF.astype(np.float64)
print(f'original data conditional number={np.linalg.cond(pcaDatasetDF)}')
pca=PCA()
pca.fit(pcaDatasetDF)
pcaCoordinate=pca.transform(pcaDatasetDF)
EVR= pca.explained_variance_ratio_
CVR = np.cumsum(EVR) # cumulative explained variance
numComponents = np.argmax(CVR >= 0.95) +1
print("number of Principal Component = "+ str(numComponents))



# PCA-Principal Component Analysis
pca=PCA(numComponents)
pca.fit(pcaDatasetDF)
pcaCoordinate=pca.transform(pcaDatasetDF)
# print(pcaCoordinate)


# PCA-Principal Component Analysis
newPCAdf = pd.DataFrame(pcaCoordinate)
# print(newPCAdf.shape[1])
newColumnNames = {col : 'Component-'+ str(col+1) for col in newPCAdf.columns}
newPCAdf = newPCAdf.rename(columns = newColumnNames)
print(f'transform data conditional number={np.linalg.cond(newPCAdf)}')
print(newPCAdf.head(5).to_string())
```

```
plt.figure(figsize=(8, 6))
plt.plot(np.arange(1, pcaDatasetDF.shape[1]+ 1), CVR, marker='o')
plt.xlabel('Number of Features')
plt.ylabel('Cumulative Explained Variance')
plt.axhline(y=0.95, color='red', label='95% Threshold')
plt.axvline(x=numComponents, color='green', label=f'{numComponents}
Features')
plt.legend()
plt.grid(True)
plt.show()
```

```
# SVD-Singular Value Decomposition Analysis
# X_train is standardized
from sklearn.decomposition import TruncatedSVD
svdDatasetDF=copy.deepcopy(X_train)
svd=TruncatedSVD(svdDatasetDF.shape[1])
svd.fit(svdDatasetDF)
svdCoordinate=svd.transform(svdDatasetDF)

EVR= svd.explained_variance_ratio_
CVR = np.cumsum(EVR) # cumulative explained variance
numComponents = np.argmax(CVR >= 0.95)+1
print("number of Component = "+ str(numComponents))
```

```
# In[103]:
```

```
# SVD-Singular Value Decomposition Analysis
svd=TruncatedSVD(numComponents)
svd.fit(svdDatasetDF)
svdCoordinate=svd.transform(svdDatasetDF)
```

```
newSVDdf = pd.DataFrame(svdCoordinate)
newColumnNames = {col: 'Component-' + str(col + 1) for col in
newSVDdf.columns}
newSVDdf = newSVDdf.rename(columns=newColumnNames)
print(newSVDdf.head().to_string())
singularValue=svd.singular_values_
print(f"\nSingular Values: {singularValue.round(3)}")

plt.figure(figsize=(8, 6))
plt.plot(np.arange(1, svdDatasetDF.shape[1]+ 1), CVR, marker='o')
plt.xlabel('Number of Features')
```

```
plt.ylabel('Cumulative Explained Variance')
plt.axhline(y=0.95, color='red', label='95% Threshold')
plt.axvline(x=numComponents, color='green', label=f'{numComponents}
Features')
plt.legend()
plt.grid(True)
plt.show()


# In[106]:


from statsmodels.stats.outliers_influence import variance_inflation_factor




# Calculate VIF for each feature
p_train = X_train.astype(int)

vif_data = pd.DataFrame()
vif_data["Feature"] = p_train.columns
vif_data["VIF"] = [variance_inflation_factor(p_train.values, i) for i in
range(p_train.shape[1])]

# Sort the features by VIF in descending order
vif_data = vif_data.sort_values(by="VIF",
ascending=False).reset_index(drop=True)

# Print the VIF values
print(vif_data)


# VIF
# plot
plt.figure(figsize=(12,8))
plt.barh(range(vif_data['VIF'].size), vif_data['VIF'])
plt.yticks(range(vif_data['Feature'].size), vif_data['Feature'])
plt.xlabel('VIF')
plt.ylabel('Features')
plt.title('VIF vs Feature')
plt.grid(True)
plt.show()


# In[157]:


# T-test analysis - LinearR
LinearRencodingTrainDF = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
```

```
LinearRstandardizedTrainDf=
Standardized(LinearRencodingTrainDF[['Age','Tenure','Usage_Frequency','Paymen
t_Delay','Last_Interaction','Total_Spend']])
LinearRstandardizedTrainDf=pd.concat([LinearRstandardizedTrainDf,LinearRencod
ingTrainDF[['Gender_Male','Subscription_Type_Premium','Subscription_Type_Stan
dard','Contract_Length_Monthly','Contract_Length_Quarterly','Churn']]],
axis=1)

# X_train= standardizedTrainDf.drop(columns=['Churn'])
X_train_LinearR = LinearRstandardizedTrainDf
# X_train = sm.add_constant(X_train)
y_train_LinearR = Standardized(training_df['Support_Calls'])


LinearRencodingTestDF = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
LinearRstandardizedTestDf=
Standardized(LinearRencodingTestDF[['Age','Tenure','Usage_Frequency','Payment
_Delay','Last_Interaction','Total_Spend']])
LinearRstandardizedTestDf=pd.concat([LinearRstandardizedTestDf,LinearRencodin
gTestDF[['Gender_Male','Subscription_Type_Premium','Subscription_Type_Standar
d','Contract_Length_Monthly','Contract_Length_Quarterly','Churn']]], axis=1)

X_test_LinearR= LinearRstandardizedTestDf
y_test_LinearR = Standardized(testing_df['Support_Calls'])

# add const
X_train_LinearR = sm.add_constant(X_train_LinearR)
X_test_LinearR = sm.add_constant(X_test_LinearR)


# In[158]:


# T-test analysis - LinearR
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
linreg = LinearRegression()
linreg.fit(X_train_LinearR, y_train_LinearR)
y_pred_train_LinearR = linreg.predict(X_train_LinearR)
y_pred_LinearR = linreg.predict(X_test_LinearR)

# Calculate R-squared
originalTestTaarget= testing_df['Support_Calls']
# deStandardizedTargetTest = y_test_ols * originalTestTaarget.std() +
originalTestTaarget.mean()
mse_LinearR = mean_squared_error(y_test_LinearR * originalTestTaarget.std() +
originalTestTaarget.mean(), y_pred_LinearR *originalTestTaarget.std() +
originalTestTaarget.mean())


r_squared_LinearR = linreg.score(X_train_LinearR, y_train_LinearR)
n = len(y_train_LinearR)
```

```
p = X_train_LinearR.shape[1] - 1  # Subtracting 1 for the intercept term
adjusted_r_squared_LinearR = 1 - (1 - r_squared_LinearR) * (n - 1) / (n - p -
1)
aic_LinearR = n*np.log(mse_LinearR) + 2*p
bic_LinearR = n*np.log(mse_LinearR) + p*np.log(n)

print(f"R-squared:{r_squared_LinearR}, adjusted R-square:
{adjusted_r_squared_LinearR}, AIC:{aic_LinearR}, BIC:{bic_LinearR} , Mean
Squared Error:{mse_LinearR} ")


# In[159]:


# T-test analysis - LinearR
coefficients = linreg.coef_
intercept = linreg.intercept_
coefficients_formatted = [f'{coef:.3f}' for coef in coefficients]
intercept_formatted = f'{intercept:.3f}'

coefficients = [(float(coef)) for coef in coefficients_formatted][1:]
columns = X_train_LinearR.columns[1:]

equation = f"Support_Calls = {float(intercept_formatted):.3f}"

for coef, column in zip(coefficients[0:], columns):
   if(coef>0):
       equation += f" + {coef:.3f} {column}"
   elif coef <0:
       equation += f" {coef:.3f} {column}"

print("LinearRegression Model Equation:\n", equation)




# T-test analysis - LinearR
from sklearn.metrics import make_scorer, mean_squared_error
originalTestTaarget= testing_df['Support_Calls'] # non Standardized Target
test
originalTrainTaarget= training_df['Support_Calls']

deStandardizedTargetTest = y_test_LinearR * originalTestTaarget.std() +
originalTestTaarget.mean()
deStandardizedTargetPred= y_pred_LinearR * originalTestTaarget.std() +
originalTestTaarget.mean()
deStandardizedTargetTrain= y_train_LinearR * originalTrainTaarget.std() +
originalTrainTaarget.mean()

# deStandardizedTargetPred_df= pd.DataFrame(deStandardizedTargetPred,
columns=['Churn'])
```

```
deStandardizedTargetTest=
deStandardizedTargetTest.reset_index().drop(['index'],axis=1)
# deStandardizedTargetPred=
deStandardizedTargetPred_df.reset_index().drop(['index'],axis=1)
deStandardizedTargetTrain=
deStandardizedTargetTrain.reset_index().drop(['index'],axis=1)



# plot
plt.figure(figsize=(8, 4))
plt.plot(deStandardizedTargetTest[:80], label='Actual Sales')
plt.plot(deStandardizedTargetPred[:80], label='Predicted Sales')
plt.xlabel("value")
plt.ylabel("Support_Calls")
plt.title(' Actual vs predicted- 80 observation')
plt.legend()
plt.grid(True)
plt.show()
# plot
plt.figure(figsize=(8, 4))
plt.plot(deStandardizedTargetTest, label='Actual Sales')
plt.plot(deStandardizedTargetPred, label='Predicted Sales')
plt.xlabel("value")
plt.ylabel("Support_Calls")
plt.title(' Actual vs predicted- All observation')
plt.legend()
plt.grid(True)
plt.show()

# plot
plt.figure(figsize=(8, 4))
plt.plot(deStandardizedTargetTest[:50], label='Actual Sales')
plt.plot(deStandardizedTargetPred[:50], label='Predicted Sales')
plt.plot(deStandardizedTargetTrain[:50], label='Training Sales')
plt.xlabel("value")
plt.ylabel("Support_Calls")
plt.title(' Actual vs predicted vs Training - 50 observation')
plt.legend()
plt.grid(True)
plt.show()



# T-test analysis - OLS
OLSencodingTrainDF = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
OLSstandardizedTrainDf=
Standardized(OLSencodingTrainDF[['Age','Tenure','Usage_Frequency','Payment_De
lay','Last_Interaction','Total_Spend']])
OLSstandardizedTrainDf=pd.concat([OLSstandardizedTrainDf,OLSencodingTrainDF[[
'Gender_Male','Subscription_Type_Premium','Subscription_Type_Standard','Contr
act_Length_Monthly','Contract_Length_Quarterly','Churn']]], axis=1)
```

```
# X_train= standardizedTrainDf.drop(columns=['Churn'])
X_train_ols = OLSstandardizedTrainDf
# X_train = sm.add_constant(X_train)
y_train_ols = Standardized(training_df['Support_Calls'])


OLSencodingTestDF = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
OLSstandardizedTestDf=
Standardized(OLSencodingTestDF[['Age','Tenure','Usage_Frequency','Payment_Del
ay','Last_Interaction','Total_Spend']])
OLSstandardizedTestDf=pd.concat([OLSstandardizedTestDf,OLSencodingTestDF[['Ge
nder_Male','Subscription_Type_Premium','Subscription_Type_Standard','Contract
_Length_Monthly','Contract_Length_Quarterly','Churn']]], axis=1)

X_test_ols= OLSstandardizedTestDf
y_test_ols = Standardized(testing_df['Support_Calls'])

# add const
X_train_ols = sm.add_constant(X_train_ols)
X_test_ols = sm.add_constant(X_test_ols)

# print(X_train_ols.head(5))


# In[162]:


# convert bool data to 0/1
for i,j in X_train_ols.dtypes.items():
   if j=='bool':
       X_train_ols[i]= X_train_ols[i].astype(int)
for i,j in X_test_ols.dtypes.items():
   if j=='bool':
       X_test_ols[i]= X_test_ols[i].astype(int)




# T-test analysis - OLS
results = pd.DataFrame(columns=['Features', 'AIC', 'BIC', 'Adj_R2',
'P-value'])
results._clear_item_cache()
import statsmodels.api as sm
OLSmodel = sm.OLS(y_train_ols,X_train_ols).fit()
results = results._append({'Features': OLSmodel.pvalues.idxmax(),
     'AIC': OLSmodel.aic,
     'BIC': OLSmodel.bic,
     'Adj_R2': OLSmodel.rsquared_adj,
     'P-value': OLSmodel.pvalues.max()}, ignore_index=True)
```

```
# In[164]:


# T-test analysis - OLS
print(OLSmodel.summary())
print(f'feature to remove: {OLSmodel.pvalues.idxmax()}')



# F-Test
from scipy.stats import f
F_statistic = OLSmodel.fvalue
df_model = OLSmodel.df_model
df_resid = OLSmodel.df_resid
alpha = 0.05
critical_F = f.ppf(1 - alpha, df_model, df_resid)

if F_statistic > critical_F:
   print("Reject null hypothesis. Model is statistically significant.")
else:
   print("Null hypothesis fail to reject. Model is not statistically
significant.")


# In[276]:


print(F_statistic, critical_F)


# In[166]:


# OLSmodel.pvalues[OLSmodel.pvalues.idxmax()]
while OLSmodel.pvalues[OLSmodel.pvalues.idxmax()].round(3) != 0:
   X_train_ols= X_train_ols.drop(columns=[OLSmodel.pvalues.idxmax()])

   OLSmodel = sm.OLS(y_train_ols,X_train_ols).fit()
   results = results._append({'Features': OLSmodel.pvalues.idxmax(),
     'AIC': OLSmodel.aic,
     'BIC': OLSmodel.bic,
     'Adj_R2': OLSmodel.rsquared_adj,
     'P-value': OLSmodel.pvalues.max()}, ignore_index=True)
   print(OLSmodel.summary())
print('dropped feature')
# Drop the last row
results = results.drop(results.index[-1])
print(results.round(6))
```

```
equation= 'Support_Calls= '
for i,feature in enumerate(OLSmodel.params.items()):
   if feature[0] != 'const':
       if (feature[1] > 0):
           equation = equation + ' +'
       else:
           equation = equation + ' -'
       equation = equation + ' ' + str(abs(feature[1].__round__(3))) + ' ' +
feature[0]
   else:
       equation= equation + str(feature[1].__round__(3))
print("OLS model equation\n",equation)
# remove Standardized dataframe * dataframe.std() +dataframe.mean()



X_test_ols= X_test_ols.drop(columns=results.Features)


# In[250]:


# T-test analysis - OLS
from prettytable import PrettyTable
from sklearn.metrics import make_scorer, mean_squared_error
OLStargetPred= OLSmodel.predict(X_test_ols)
originalTestTaarget= testing_df['Support_Calls'] # non Standardized Target
test
originalTrainTaarget= training_df['Support_Calls']

deStandardizedTargetTest = y_test_ols * originalTestTaarget.std() +
originalTestTaarget.mean()
deStandardizedTargetPred= OLStargetPred * originalTestTaarget.std() +
originalTestTaarget.mean()
deStandardizedTargetTrain= y_train_ols * originalTrainTaarget.std() +
originalTrainTaarget.mean()

deStandardizedTargetTest=
deStandardizedTargetTest.reset_index().drop(['index'],axis=1)
deStandardizedTargetPred=
deStandardizedTargetPred.reset_index().drop(['index'],axis=1)
deStandardizedTargetTrain=
deStandardizedTargetTrain.reset_index().drop(['index'],axis=1)



# plot
plt.figure(figsize=(8, 4))
plt.plot(deStandardizedTargetTest[:80], label='Actual Sales')
plt.plot(deStandardizedTargetPred[:80], label='Predicted Sales')
plt.xlabel("value")
plt.ylabel("Support_Calls")
```

```
plt.title(' Actual vs predicted- 80 observation')
plt.legend()
plt.grid(True)
plt.show()
# plot
plt.figure(figsize=(8, 4))
plt.plot(deStandardizedTargetTest, label='Actual Sales')
plt.plot(deStandardizedTargetPred, label='Predicted Sales')
plt.xlabel("value")
plt.ylabel("Support_Calls")
plt.title(' Actual vs predicted- All observation')
plt.legend()
plt.grid(True)
plt.show()

# plot
plt.figure(figsize=(8, 4))
plt.plot(deStandardizedTargetTest[:50], label='Actual Sales')
plt.plot(deStandardizedTargetPred[:50], label='Predicted Sales')
plt.plot(deStandardizedTargetTrain[:50], label='Training Sales')
plt.xlabel("value")
plt.ylabel("Support_Calls")
plt.title(' Actual vs predicted vs Training - 50 observation')
plt.legend()
plt.grid(True)
plt.show()


# accuracy
mse_OLS = mean_squared_error(deStandardizedTargetTest,
deStandardizedTargetPred)
print("Mean Squared Error:", mse_OLS.__round__(3))


#

# In[170]:


# T-test analysis - OLS
r_squared = linreg.score(X_test_LinearR, y_test_LinearR)
originalTrainTaarget= training_df['Support_Calls']
# deStandardizedTargetTest = y_test_ols * originalTestTaarget.std() +
originalTestTaarget.mean()
# mse_LinearR = mean_squared_error(y_train_LinearR *
originalTrainTaarget.std() + originalTrainTaarget.mean(),
y_pred_train_LinearR)

r_squared_OLS = OLSmodel.rsquared
adjusted_r_squared_OLS = OLSmodel.rsquared_adj
aic_OLS = OLSmodel.aic
bic_OLS = OLSmodel.bic
# compare table
```

```
table = PrettyTable()
table.field_names = ["Metric", "OLS Model", "Linear Regression Model"]
table.float_format = ".3"
table.add_row(["R-squared", r_squared_OLS, r_squared_LinearR])
table.add_row(["Adjusted R-squared", adjusted_r_squared_OLS,
adjusted_r_squared_LinearR])
table.add_row(["AIC", aic_OLS, aic_LinearR])
table.add_row(["BIC", bic_OLS, bic_LinearR])
table.add_row(["MSE", mse_OLS, mse_LinearR])
print(table)


# In[252]:


# T-test analysis - OLS
model1featureTest= X_test_ols
model1OriginalSalesDF= testing_df['Support_Calls']
prediction_intervals = OLSmodel.get_prediction(model1featureTest)
lowerBounds = prediction_intervals.summary_frame()["obs_ci_lower"]
upperBounds = prediction_intervals.summary_frame()["obs_ci_upper"]
lowerBounds = lowerBounds * model1OriginalSalesDF.std()
+model1OriginalSalesDF.mean()
upperBounds = upperBounds * model1OriginalSalesDF.std() +
model1OriginalSalesDF.mean()

# plot
No_of_ob_to_show= 100
plt.figure(figsize=(12,4))
plt.plot(deStandardizedTargetPred[:No_of_ob_to_show], label="Predicted
Sales",color= "blue")
plt.fill_between(range(len(deStandardizedTargetTest[:No_of_ob_to_show])),lowe
rBounds[:No_of_ob_to_show], upperBounds[:No_of_ob_to_show], alpha=0.5,
label="CI")
plt.xlabel("# of samples")
plt.ylabel("Support_Calls)")
plt.legend()
plt.grid(True)
plt.title("Sales Prediction with Confidence Interval")
plt.tight_layout()
plt.show()
# plot
plt.figure(figsize=(12,4))
plt.plot(deStandardizedTargetPred, label="Predicted Sales",color= "blue")
plt.fill_between(range(len(deStandardizedTargetTest)),lowerBounds,
upperBounds, alpha=0.5, label="CI")
plt.xlabel("# of samples")
plt.ylabel("Support_Calls")
plt.legend()
plt.grid(True)
plt.title("Sales Prediction with Confidence Interval")
plt.tight_layout()
plt.show()
```

```
# DecisionTreeClassifier
import pandas as pd
import seaborn as sbn
from sklearn.tree import DecisionTreeClassifier
encodingTrainDFForDT = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
# encodingTrainDFForDT =
encodingTrainDFForDT.drop(columns=['Contract_Length_Quarterly',
'Usage_Frequency', 'Subscription_Type_Premium',
'Subscription_Type_Standard'])
XTrain_DT = encodingTrainDFForDT[['Age', 'Tenure', 'Usage_Frequency',
'Support_Calls', 'Payment_Delay',
      'Total_Spend', 'Last_Interaction', 'Gender_Male',
      'Subscription_Type_Premium', 'Subscription_Type_Standard',
      'Contract_Length_Monthly', 'Contract_Length_Quarterly']]
yTrain_DT = encodingTrainDFForDT['Churn']

encodingTestDFForDT = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
XTest_DT = encodingTestDFForDT[['Age', 'Tenure', 'Usage_Frequency',
'Support_Calls', 'Payment_Delay',
      'Total_Spend', 'Last_Interaction', 'Gender_Male',
      'Subscription_Type_Premium', 'Subscription_Type_Standard',
      'Contract_Length_Monthly', 'Contract_Length_Quarterly']]
yTest_DT = encodingTestDFForDT['Churn']


# In[175]:


# DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, recall_score,
roc_auc_score, roc_curve,f1_score
DTclf = DecisionTreeClassifier(random_state=5805)
DTclf.fit(XTrain_DT,yTrain_DT)
yTrainPredicted_DT = DTclf.predict(XTrain_DT)
yTestPredicted_DT = DTclf.predict(XTest_DT)
print(f'DecisionTree Test accuracy
{accuracy_score(yTest_DT,yTestPredicted_DT).__round__(5)}')
# DecisionTreeClassifier
# feature importance
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["feature", "Feature Importances"]

# featureImportances={}
featureColumns= XTrain_DT.columns
for i,j in enumerate(featureColumns):
```

```
 table.add_row([j, DTclf.feature_importances_[i].round(5)])
table.sortby = "Feature Importances"
print(table)



# DecisionTreeClassifier
newXTrain_DT= XTrain_DT.drop(['Contract_Length_Quarterly'],axis=1)
newXTest_DT=XTest_DT.drop(['Contract_Length_Quarterly'],axis=1)


# In[177]:


# DecisionTreeClassifier
DTclf = DecisionTreeClassifier(random_state=5805)
DTclf.fit(newXTrain_DT,yTrain_DT)
yTrainPredicted_DT = DTclf.predict(newXTrain_DT)
yTestPredicted_DT = DTclf.predict(newXTest_DT)
print(f'DecisionTree Test accuracy
{accuracy_score(yTest_DT,yTestPredicted_DT).__round__(5)}')
# DecisionTreeClassifier
# feature importance
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["feature", "Feature Importances"]

# featureImportances={}
featureColumns= newXTrain_DT.columns
for i,j in enumerate(featureColumns):
 table.add_row([j, DTclf.feature_importances_[i].round(5)])
table.sortby = "Feature Importances"
print(table)



# DecisionTreeClassifier
newXTrain_DT=
XTrain_DT.drop(['Contract_Length_Quarterly','Subscription_Type_Premium'],axis
=1)
newXTest_DT=XTest_DT.drop(['Contract_Length_Quarterly','Subscription_Type_Pre
mium'],axis=1)
DTclf = DecisionTreeClassifier(random_state=5805)
DTclf.fit(newXTrain_DT,yTrain_DT)
yTrainPredicted_DT = DTclf.predict(newXTrain_DT)
yTestPredicted_DT = DTclf.predict(newXTest_DT)
print(f'DecisionTree Test accuracy
{accuracy_score(yTest_DT,yTestPredicted_DT).__round__(5)}')
# DecisionTreeClassifier
# feature importance
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["feature", "Feature Importances"]
```

```
# featureImportances={}
featureColumns= newXTrain_DT.columns
for i,j in enumerate(featureColumns):
 table.add_row([j, DTclf.feature_importances_[i].round(5)])
table.sortby = "Feature Importances"
print(table)


# In[179]:


# DecisionTreeClassifier
removeFeature=
['Contract_Length_Quarterly','Subscription_Type_Premium','Subscription_Type_S
tandard']
newXTrain_DT= XTrain_DT.drop(removeFeature,axis=1)
newXTest_DT=XTest_DT.drop(['Contract_Length_Quarterly','Subscription_Type_Pre
mium','Subscription_Type_Standard'],axis=1)
DTclf = DecisionTreeClassifier(random_state=5805)
DTclf.fit(newXTrain_DT,yTrain_DT)
yTrainPredicted_DT = DTclf.predict(newXTrain_DT)
yTestPredicted_DT = DTclf.predict(newXTest_DT)
print(f'DecisionTree Test accuracy
{accuracy_score(yTest_DT,yTestPredicted_DT).__round__(5)}')
# DecisionTreeClassifier
# feature importance
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["feature", "Feature Importances"]

# featureImportances={}
featureColumns= newXTrain_DT.columns
for i,j in enumerate(featureColumns):
 table.add_row([j, DTclf.feature_importances_[i].round(5)])
table.sortby = "Feature Importances"
print(table)


# In[180]:


# In[182]:


print(f'Hence feature to removed from Decision tree classifier are::
{removeFeature}')
removeFeature= ['Contract_Length_Quarterly','Subscription_Type_Premium']
XTrain_DT= XTrain_DT.drop(removeFeature,axis=1)
XTest_DT=XTest_DT.drop(removeFeature,axis=1)


# In[183]:
```

```
# DecisionTreeClassifier Pre-Pruned
from sklearn.model_selection import GridSearchCV
from sklearn import tree
import warnings
warnings.filterwarnings("ignore")
DTclf = DecisionTreeClassifier(random_state=5805)
tuned_parameters = {
'max_depth': [20], #[None, 5, 10, 20] [20]
'min_samples_split': [10], #[2, 5, 10][2]
'min_samples_leaf': [3], #[1, 2,3,4][3]
'max_features': ['sqrt'], #['auto', 'sqrt', 'log']['sqrt']
'splitter': ['best'], #['best', 'random'] ['best']
'criterion': ['entropy'] #['gini', 'entropy','log_loss'] ['entropy']


}
gridSearch = GridSearchCV(estimator=DTclf, param_grid=tuned_parameters, cv=5,
scoring='accuracy')
gridSearch.fit(XTrain_DT, yTrain_DT)
print("Best parameters found: ", gridSearch.best_params_)


# In[184]:


# DecisionTreeClassifier Pre-Pruned
gridSearch.best_estimator_.fit(XTrain_DT, yTrain_DT)
yTestProbPrePruned= gridSearch.best_estimator_.predict_proba(XTest_DT)[::,
-1]
yTestPredPrePruned = gridSearch.best_estimator_.predict(XTest_DT)
yTrainPredPrePruned= gridSearch.best_estimator_.predict(XTrain_DT)
prePrunedAccuracy= accuracy_score(yTest_DT, yTestPredPrePruned)
print(f'Pre-Pruned Test accuracy {prePrunedAccuracy.__round__(5)}')
# print(f'Pre-Pruned train accuracy {accuracy_score(yTrain_DT,
yTrainPredPrePruned).__round__(5)}')


# In[185]:


# DecisionTreeClassifier Pre-Pruned
confusionMatrixPrePruned = confusion_matrix(yTest_DT, yTestPredPrePruned)
recallPrePruned = recall_score(yTest_DT, yTestPredPrePruned)
rocAucPrePruned = roc_auc_score(yTest_DT, yTestProbPrePruned)
specificityPrePruned = confusionMatrixPrePruned[0, 0] /
(confusionMatrixPrePruned[0, 0] + confusionMatrixPrePruned[0, 1])
f_score_PrePruned = f1_score(yTest_DT, yTestPredPrePruned)



# DecisionTreeClassifier Post-Pruned
```

```
DTclf = DecisionTreeClassifier(random_state=5805)
DTclf.fit(XTrain_DT,yTrain_DT)

cpppath = DTclf.cost_complexity_pruning_path(XTrain_DT,yTrain_DT)
alphas = cpppath['ccp_alphas']
# Grid search for best alpha
postPrunAccuracyTrain, postPrunAccuracyTest = [],[]
for i in alphas:
   DTclf = DecisionTreeClassifier(random_state=5805,ccp_alpha=i)
   DTclf.fit(XTrain_DT,yTrain_DT)
   yTrainPredPostPruned = DTclf.predict(XTrain_DT)
   postPrunAccuracyTrain.append(accuracy_score(yTrain_DT,
yTrainPredPostPruned))
   yTestPredPostPruned = DTclf.predict(XTest_DT)
   postPrunAccuracyTest.append(accuracy_score(yTest_DT, yTestPredPostPruned))
print(f'alpha={alphas[postPrunAccuracyTest.index(max(postPrunAccuracyTest))].
round(5)}')


# In[188]:


# DecisionTreeClassifier Post-Pruned
fig, ax = plt.subplots()
ax.set_xlabel('alpha')
ax.set_ylabel('accuracy')
ax.set_title("Alpha for training and testing sets VS Accuracy")
ax.plot(alphas, postPrunAccuracyTrain, label="Train", drawstyle="steps-post")
ax.plot(alphas, postPrunAccuracyTest, label="Test", drawstyle="steps-post")
ax.legend()
plt.grid()
plt.tight_layout()
plt.show()


# In[189]:


# DecisionTreeClassifier Post-Pruned
DTclf = DecisionTreeClassifier(random_state=5805,
ccp_alpha=alphas[postPrunAccuracyTest.index(max(postPrunAccuracyTest))].round
(5))
DTclf.fit(XTrain_DT, yTrain_DT)
yTrainPredPostPruned = DTclf.predict(XTrain_DT)
yTestPredPostPruned = DTclf.predict(XTest_DT)
yTestProbPostPruned= DTclf.predict_proba(XTest_DT)[::, -1]
postrePrunedAccuracy= accuracy_score(yTest_DT, yTestPredPostPruned)
postrePrunedAccuracy_train= accuracy_score(yTrain_DT, yTrainPredPostPruned)
print(f'Post-Pruned Test accuracy {accuracy_score(yTest_DT,
yTestPredPostPruned).__round__(5)}')
print(f'Post-Pruned Train accuracy {accuracy_score(yTrain_DT,
yTrainPredPostPruned).__round__(5)}')
```

```
# In[190]:


# DecisionTreeClassifier Post-Pruned
for key, value in DTclf.get_params().items():
 print(f"{key}: {value}")

confusionMatrixPostPruned = confusion_matrix(yTest_DT, yTestPredPostPruned)
recallPostPruned = recall_score(yTest_DT, yTestPredPostPruned)
rocAucPostPruned = roc_auc_score(yTest_DT, yTestProbPostPruned)
specificityPostPruned = confusionMatrixPostPruned[0, 0] /
(confusionMatrixPostPruned[0, 0] + confusionMatrixPostPruned[0, 1])
f_score_PostPruned = f1_score(yTest_DT, yTestPredPostPruned)


table1 = PrettyTable()
table1.field_names = ["","Accuracy", "confusion Matrix", "recall",
'AUC','Specificity','F-score']
table1.add_row(["Pre-Pruned",prePrunedAccuracy.round(2),confusionMatrixPrePru
ned,recallPrePruned.round(2),rocAucPrePruned.round(2),specificityPrePruned.ro
und(3),f_score_PrePruned.round(3)])
table1.add_row(["Post-Pruned",postrePrunedAccuracy.round(2),confusionMatrixPo
stPruned,recallPostPruned.round(2),rocAucPostPruned.round(2),specificityPostP
runed.round(3),f_score_PostPruned.round(3)])
print(table1)


# In[191]:


# DecisionTreeClassifier Post-Pruned
sns.heatmap(confusionMatrixPostPruned, annot=True, fmt='d', cmap='Blues')
plt.title("DecisionTreeClassifier Post-Pruned")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
# DecisionTreeClassifier Pre-Pruned
sns.heatmap(confusionMatrixPrePruned, annot=True, fmt='d', cmap='Blues')
plt.title("DecisionTreeClassifier Pre-Pruned")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# In[192]:


# DecisionTreeClassifier Post-Pruned
# K-fold
X = pd.concat([XTrain_DT, XTest_DT], axis=0, ignore_index=True)
Y= pd.concat([yTrain_DT, yTest_DT], axis=0, ignore_index=True)
```

```
from sklearn.model_selection import StratifiedKFold



k_fold = StratifiedKFold(n_splits=5, shuffle=True, random_state=5805)
DTclf = DecisionTreeClassifier(random_state=5805,
ccp_alpha=alphas[postPrunAccuracyTest.index(max(postPrunAccuracyTest))].round
(5))
cv_accuracy = cross_val_score(DTclf, X, Y, cv=k_fold, scoring='accuracy',
error_score='raise',n_jobs=-1)
print(f"\nDecisionTreeClassifier Post-Pruned Stratified K-fold
Cross-Validation Accuracy: {np.mean(cv_accuracy):.2%}\n")


# In[193]:


# Post-Pruned
fprPrePrunedtree, tprPrePrunedtree, _ = roc_curve(yTest_DT,
yTestProbPrePruned)
fprPostPrunedtree, tprPostPrunedtree, _ =
roc_curve(yTest_DT,yTestProbPostPruned)
plt.plot(fprPrePrunedtree, tprPrePrunedtree, label='Pre-Pruned')
plt.plot(fprPostPrunedtree, tprPostPrunedtree, label='Post-Pruned')
plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('8.ROC Curve')
plt.legend()
plt.grid(True)
plt.show()


# In[197]:


# Logistic regression
#remove
'Contract_Length_Quarterly','Usage_Frequency','Tenure','Subscription_Type_P
remium'
removefeature= []
finalFeature= [item for item in ['Age', 'Tenure', 'Usage_Frequency',
'Support_Calls', 'Payment_Delay',
      'Total_Spend', 'Last_Interaction', 'Gender_Male',
      'Subscription_Type_Premium', 'Subscription_Type_Standard',
      'Contract_Length_Monthly', 'Contract_Length_Quarterly'] if item not in
removefeature]

from sklearn.linear_model import LogisticRegression
encodingTrainDFForLR = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
# encodingTrainDFForDT =
encodingTrainDFForDT.drop(columns=['Contract_Length_Quarterly',
```

```
'Usage_Frequency', 'Subscription_Type_Premium',
'Subscription_Type_Standard'])
XTrain_LR = encodingTrainDFForLR[finalFeature]
yTrain_LR = encodingTrainDFForLR['Churn']

encodingTestDFForLR = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
XTest_LR = encodingTestDFForLR[finalFeature]
yTest_LR = encodingTestDFForLR['Churn']

removeFeature= ['Contract_Length_Quarterly','Usage_Frequency']
XTrain_LR= XTrain_LR.drop(removeFeature,axis=1)
XTest_LR=XTest_LR.drop(removeFeature,axis=1)


# In[198]:


# Logistic regression
# gridSearch taking time and not executing
logregclf = LogisticRegression()

param_grid = {
    'penalty': ['l1', 'l2'],
    'C': np.logspace(-3, 3, 7,8),
    'solver': ['liblinear', 'saga']
}

grid_search = GridSearchCV(estimator=logregclf, param_grid=param_grid, cv=5,
scoring='accuracy')
grid_search.fit(XTrain_LR, yTrain_LR)

print("Best parameters found:", grid_search.best_params_)

logregclf = grid_search.best_estimator_



logregYTestPred = logregclf.predict(XTest_LR)
logregAccuracy = accuracy_score(yTest_LR, logregYTestPred)
yTestProbLogreg= logregclf.predict_proba(XTest_LR)[::, -1]
print(f'Logistic regression Accuracy = {logregAccuracy.__round__(5)}')



# In[200]:


# Logistic regression
confusionMatrixlogreg = confusion_matrix(yTest_LR, logregYTestPred)
recallLogreg = recall_score(yTest_LR, logregYTestPred)
rocAucLogreg = roc_auc_score(yTest_LR, yTestProbLogreg)
```

```
specificityLogreg = confusionMatrixlogreg[0, 0] / (confusionMatrixlogreg[0,
0] + confusionMatrixlogreg[0, 1])
f_score_Logreg = f1_score(yTest_LR, logregYTestPred)

table2 = PrettyTable()
table2.field_names = ["","Accuracy", "confusion Matrix", "recall",
'AUC','Specificity','F-Score']
table2.add_row(["Decision Tree Post-Pruned",postrePrunedAccuracy.round(2),
confusionMatrixPostPruned,recallPostPruned.round(2),rocAucPostPruned.round(2)
, specificityPostPruned.round(2),f_score_PostPruned.round(2)])
table2.add_row(["logistic
regression",logregAccuracy.round(2),confusionMatrixlogreg.round(2),recallLogr
eg.round(2),rocAucLogreg.round(2),specificityLogreg.round(2),f_score_Logreg.r
ound(2)])
print(table2)

fprlogregclf, tprlogregclf, _ = roc_curve(yTest_LR, yTestProbLogreg)
# ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fprPostPrunedtree, tprPostPrunedtree,  label='Decision
tree-Post-Pruned')
plt.plot(fprlogregclf, tprlogregclf, label='Logistic regression')
plt.plot([0, 1], [0, 1], 'k--', label='Random selection')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()


sns.heatmap(confusionMatrixlogreg, annot=True, fmt='d', cmap='Blues')
plt.title("Logistic regression")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# In[201]:


# svm
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

encodingTrainDFForSVM = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
standardizedTrainDfSVM=
Standardized(encodingTrainDFForSVM[['Age','Tenure','Usage_Frequency','Support
_Calls','Payment_Delay','Total_Spend','Last_Interaction']])
```

```
standardizedTrainDfSVM=pd.concat([standardizedTrainDfSVM,encodingTrainDFForSV
M[['Gender_Male','Subscription_Type_Premium','Subscription_Type_Standard','Co
ntract_Length_Monthly','Contract_Length_Quarterly']]], axis=1)
# encodingTrainDFForDT =
encodingTrainDFForDT.drop(columns=['Contract_Length_Quarterly',
'Usage_Frequency', 'Subscription_Type_Premium',
'Subscription_Type_Standard'])
XTrain_SVM = standardizedTrainDfSVM
yTrain_SVM = encodingTrainDFForSVM['Churn']

encodingTestDFForSVM = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
standardizedTestDfSVM=
Standardized(encodingTestDFForSVM[['Age','Tenure','Usage_Frequency','Support_
Calls','Payment_Delay','Total_Spend','Last_Interaction']])
standardizedTestDfSVM=pd.concat([standardizedTestDfSVM,encodingTestDFForSVM[[
'Gender_Male','Subscription_Type_Premium','Subscription_Type_Standard','Contr
act_Length_Monthly','Contract_Length_Quarterly']]], axis=1)

XTest_SVM = standardizedTestDfSVM
yTest_SVM = encodingTestDFForSVM['Churn']

removeFeature= ['Contract_Length_Quarterly','Usage_Frequency']
XTrain_SVM= XTrain_SVM.drop(removeFeature,axis=1)
XTest_SVM=XTest_SVM.drop(removeFeature,axis=1)



# under_sample = RandomUnderSampler(sampling_strategy='auto',
random_state=5805)
# Downsampling the data as it is taking long time.
# XTrain_SVM_GD, yTrain_SVM_GD = under_sample.fit_resample(XTrain_SVM,
yTrain_SVM)
XTrain_SVM_GD = XTrain_SVM[::4]
yTrain_SVM_GD = yTrain_SVM[::4]
print(len(XTrain_SVM_GD))
print(len(yTrain_SVM_GD))


# In[204]:


# #SVM
from sklearn.svm import SVC
svm_model = SVC()
param_grid = {'C': [0.1, 1, 10, 100], 'kernel': ['linear', 'rbf', 'poly'],
'gamma': ['scale', 'auto']}

grid_search = GridSearchCV(svm_model, param_grid, scoring='accuracy', cv=5)
grid_search.fit(XTrain_SVM_GD, yTrain_SVM_GD)

best_params = grid_search.best_params_
print(f'Best parameters found:{best_params}')
```

```
# In[205]:


# svm
from sklearn.svm import SVC
svmclf = SVC(kernel='rbf', C=10,probability=True,gamma= 'auto')
svmclf.fit(XTrain_SVM, yTrain_SVM)
svmYTestPred = svmclf.predict(XTest_SVM)
svmAccuracy = accuracy_score(yTest_SVM, svmYTestPred)

print(f'SVM Accuracy = {svmAccuracy.__round__(5)}')



# In[206]:


# from sklearn.metrics import accuracy_score
# svmAccuracy=accuracy_score(yTest_SVM, svmYTestPred)
# print(f'SVM Accuracy = {svmAccuracy.__round__(5)}')


# In[207]:


yTestProbsvm= svmclf.predict_proba(XTest_SVM)[::, -1]
confusionMatrixsvm = confusion_matrix(yTest_SVM, svmYTestPred)
recallsvm = recall_score(yTest_SVM, svmYTestPred)
rocAucsvm = roc_auc_score(yTest_SVM, yTestProbsvm)
specificitysvm = confusionMatrixsvm[0, 0] / (confusionMatrixsvm[0, 0] +
confusionMatrixsvm[0, 1])
f_score_svm = f1_score(yTest_SVM, svmYTestPred)

table2 = PrettyTable()
table2.field_names = ["","Accuracy", "confusion Matrix", "recall",
'AUC','Specificity','F-Score']
table2.add_row(["Decision Tree Post-Pruned",postrePrunedAccuracy.round(2),
confusionMatrixPostPruned,recallPostPruned.round(2),rocAucPostPruned.round(2)
, specificityPostPruned.round(3),f_score_PostPruned.round(3)])
table2.add_row(["logistic
regression",logregAccuracy.round(2),confusionMatrixlogreg.round(2),recallLogr
eg.round(2),rocAucLogreg.round(2),specificityLogreg.round(3),f_score_Logreg.r
ound(3)])
table2.add_row(["SVM",svmAccuracy.round(2),confusionMatrixsvm.round(2),recall
svm.round(2),rocAucsvm.round(2),
specificitysvm.round(3),f_score_svm.round(3)])
print(table2)


# In[208]:
```

```
# svm
fprsvm, tprsvm, _ = roc_curve(yTest_SVM, yTestProbsvm)
# ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fprPostPrunedtree, tprPostPrunedtree,  label='Decision
tree-Post-Pruned')
plt.plot(fprlogregclf, tprlogregclf, label='Logistic regression')
plt.plot(fprsvm, tprsvm, label='SVM')
plt.plot([0, 1], [0, 1], 'k--', label='Random selection')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()

#confusion Matrix
sns.heatmap(confusionMatrixsvm, annot=True, fmt='d', cmap='Blues')
plt.title("SVM")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# In[208]:




# In[209]:


# Naïve Bayes
#remove
'Contract_Length_Quarterly','Usage_Frequency','Tenure','Subscription_Type_P
remium'
removefeature= []
finalFeature= [item for item in ['Age', 'Tenure', 'Usage_Frequency',
'Support_Calls', 'Payment_Delay',
      'Total_Spend', 'Last_Interaction', 'Gender_Male',
      'Subscription_Type_Premium', 'Subscription_Type_Standard',
      'Contract_Length_Monthly', 'Contract_Length_Quarterly'] if item not in
removefeature]


encodingTrainDFForNB = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
# encodingTrainDFForDT =
encodingTrainDFForDT.drop(columns=['Contract_Length_Quarterly',
'Usage_Frequency', 'Subscription_Type_Premium',
'Subscription_Type_Standard'])
```

```
XTrain_NB = encodingTrainDFForNB[finalFeature]
yTrain_NB = encodingTrainDFForNB['Churn']

encodingTestDFForNB = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
XTest_NB = encodingTestDFForNB[finalFeature]
yTest_NB = encodingTestDFForNB['Churn']

removeFeature= ['Contract_Length_Quarterly','Usage_Frequency']
XTrain_NB= XTrain_NB.drop(removeFeature,axis=1)
XTest_NB=XTest_NB.drop(removeFeature,axis=1)




# Naïve Bayes
from sklearn.naive_bayes import GaussianNB
nbclf = GaussianNB()
nbclf.fit(XTrain_NB, yTrain_NB)
nbYTestPred = nbclf.predict(XTest_NB)
nbAccuracy = accuracy_score(yTest_NB, nbYTestPred)
yTestProbnb= nbclf.predict_proba(XTest_NB)[::, -1]
print(f'Naïve Bayes = {nbAccuracy.__round__(5)}')


# In[212]:


# Naïve Bayes k-fold

X = pd.concat([XTrain_NB, XTest_NB], axis=0, ignore_index=True)
Y= pd.concat([yTrain_NB, yTest_NB], axis=0, ignore_index=True)
k_fold = StratifiedKFold(n_splits=5, shuffle=True, random_state=5805)
nb_clf = GaussianNB()
cv_accuracy = cross_val_score(nb_clf, X, Y, cv=k_fold, scoring='accuracy',
error_score='raise',n_jobs=-1)
print(f"\nStratified K-fold Cross-Validation Accuracy:
{np.mean(cv_accuracy):.2%}\n")


# In[213]:


# Naïve Bayes
confusionMatrixnb = confusion_matrix(yTest_NB, nbYTestPred)
recallnb = recall_score(yTest_NB, nbYTestPred)
rocAucnb = roc_auc_score(yTest_NB, yTestProbnb)
specificitynb = confusionMatrixnb[0, 0] / (confusionMatrixnb[0, 0] +
confusionMatrixnb[0, 1])
f_score_nb = f1_score(yTest_NB, nbYTestPred)

table2 = PrettyTable()
```

```
table2.field_names = ["","Accuracy", "confusion Matrix", "recall",
'AUC','Specificity','F-Score']
table2.add_row(["Decision Tree Post-Pruned",postrePrunedAccuracy.round(2),
confusionMatrixPostPruned,recallPostPruned.round(2),rocAucPostPruned.round(2)
, specificityPostPruned.round(3),f_score_PostPruned.round(3)])
table2.add_row(["logistic
regression",logregAccuracy.round(2),confusionMatrixlogreg.round(2),recallLogr
eg.round(2),rocAucLogreg.round(2),specificityLogreg.round(3),f_score_Logreg.r
ound(3)])
table2.add_row(["SVM",svmAccuracy.round(2),confusionMatrixsvm.round(2),recall
svm.round(2),rocAucsvm.round(2),
specificitysvm.round(3),f_score_svm.round(3)])
table2.add_row(["Naïve
Bayes",nbAccuracy.round(2),confusionMatrixnb.round(2),recallnb.round(2),rocAu
cnb.round(2),specificitynb.round(3),f_score_nb.round(3)])
print(table2)


# In[214]:


# Naïve Bayes
fprnb, tprnb, _ = roc_curve(yTest_NB, yTestProbnb)
# ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fprPostPrunedtree, tprPostPrunedtree,  label='Decision
tree-Post-Pruned')
plt.plot(fprlogregclf, tprlogregclf, label='Logistic regression')
plt.plot(fprsvm, tprsvm, label='SVM')
plt.plot(fprnb, tprnb, label='Naïve Bayes')
plt.plot([0, 1], [0, 1], 'k--', label='Random selection')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()

#confusion Matrix
sns.heatmap(confusionMatrixnb, annot=True, fmt='d', cmap='Blues')
plt.title("Naïve Bayes")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# In[214]:




# In[215]:
```

```
# KNN
removefeature=['Contract_Length_Quarterly','Usage_Frequency']
encodingTrainDFForKNN = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
standardizedTrainDfKNN=
Standardized(encodingTrainDFForKNN[['Age','Tenure','Usage_Frequency','Support
_Calls','Payment_Delay','Total_Spend','Last_Interaction']])
standardizedTrainDfKNN=pd.concat([standardizedTrainDfKNN,encodingTrainDFForKN
N[['Gender_Male','Subscription_Type_Premium','Subscription_Type_Standard','Co
ntract_Length_Monthly','Contract_Length_Quarterly']]], axis=1)
# encodingTrainDFForDT =
encodingTrainDFForDT.drop(columns=['Contract_Length_Quarterly',
'Usage_Frequency', 'Subscription_Type_Premium',
'Subscription_Type_Standard'])
XTrain_KNN = standardizedTrainDfKNN.drop(columns=removefeature)
yTrain_KNN = encodingTrainDFForKNN['Churn']

encodingTestDFForKNN = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
standardizedTestDfKNN=
Standardized(encodingTestDFForKNN[['Age','Tenure','Usage_Frequency','Support_
Calls','Payment_Delay','Total_Spend','Last_Interaction']])
standardizedTestDfKNN =
pd.concat([standardizedTestDfKNN,encodingTestDFForKNN[['Gender_Male','Subscri
ption_Type_Premium','Subscription_Type_Standard','Contract_Length_Monthly','C
ontract_Length_Quarterly']]], axis=1)

XTest_KNN = standardizedTestDfKNN.drop(columns=removefeature)
yTest_KNN = encodingTestDFForKNN['Churn']

# removeFeature= ['Contract_Length_Quarterly','Usage_Frequency']
# XTrain_KNN= XTrain_KNN.drop(removeFeature,axis=1)
# XTest_KNN=XTest_KNN.drop(removeFeature,axis=1)




# KNN
from sklearn.neighbors import KNeighborsClassifier
error_rates = []
k_values = range(1,31)

# for k in k_values:
#     knnclf = KNeighborsClassifier(n_neighbors=k)
#     knnclf.fit(XTrain_KNN, yTrain_KNN)
#     knnYTestPred = knnclf.predict(XTest_KNN)
#     error_rates.append(np.mean(knnYTestPred != yTest_KNN))

for k in range(1, 31):
   knnclf = KNeighborsClassifier(n_neighbors=k)
```

```
    knnclf.fit(XTrain_KNN, yTrain_KNN)
    knnYTestPred = knnclf.predict(XTest_KNN)
    error = 1 - knnclf.score(XTest_KNN, yTest_KNN)
    error_rates.append(error)
    # k_values.append(k)


plt.plot(k_values, error_rates, marker='o')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Error Rate')
plt.show()



# knn
knn_model = KNeighborsClassifier()
param_grid = {
    'n_neighbors': range(1, 31)
}
grid_search = GridSearchCV(knn_model, param_grid, cv=2,
scoring='accuracy',return_train_score=True, verbose=1)
grid_search.fit(XTrain_KNN, yTrain_KNN)
print(f'Best parameters found {grid_search.best_params_}')


# In[219]:


# KNN
knnclf = grid_search.best_estimator_
knnYTestPred = knnclf.predict(XTest_KNN)
knnAccuracy= accuracy_score(yTest_KNN, knnYTestPred)
print(knnAccuracy)


# In[220]:


# KNN
# knnclf = KNeighborsClassifier(n_neighbors=25)
# knnclf.fit(XTrain_KNN, yTrain_KNN)
# knnYTestPred = knnclf.predict(XTest_KNN)
# knnAccuracy= accuracy_score(yTest_KNN, knnYTestPred)
# print(knnAccuracy)


# In[221]:


# KNN
```

```
yTestProbknn= knnclf.predict_proba(XTest_KNN)[::, -1]
confusionMatrixknn = confusion_matrix(yTest_KNN, knnYTestPred)
recallknn = recall_score(yTest_KNN, knnYTestPred)
rocAucknn = roc_auc_score(yTest_KNN, yTestProbknn)
specificityknn = confusionMatrixknn[0, 0] / (confusionMatrixknn[0, 0] +
confusionMatrixknn[0, 1])
f_score_knn = f1_score(yTest_KNN, knnYTestPred)

table2 = PrettyTable()
table2.field_names = ["","Accuracy", "confusion Matrix", "recall",
'AUC','Specificity','F-Score']
table2.add_row(["Decision Tree Post-Pruned",postrePrunedAccuracy.round(2),
confusionMatrixPostPruned,recallPostPruned.round(2),rocAucPostPruned.round(2)
, specificityPostPruned.round(3),f_score_PostPruned.round(3)])
table2.add_row(["logistic
regression",logregAccuracy.round(2),confusionMatrixlogreg.round(2),recallLogr
eg.round(2),rocAucLogreg.round(2),specificityLogreg.round(3),f_score_Logreg.r
ound(3)])
table2.add_row(["SVM",svmAccuracy.round(2),confusionMatrixsvm.round(2),recall
svm.round(2),rocAucsvm.round(2),
specificitysvm.round(3),f_score_svm.round(3)])
table2.add_row(["Naïve
Bayes",nbAccuracy.round(2),confusionMatrixnb.round(2),recallnb.round(2),rocAu
cnb.round(2),specificitynb.round(3),f_score_nb.round(3)])
table2.add_row(["KNN",knnAccuracy.round(2),confusionMatrixknn.round(2),recall
knn.round(2),rocAucknn.round(2),specificityknn.round(3),f_score_knn.round(3)]
)
print(table2)


# In[222]:


# KNN
fprknn, tprknn, _ = roc_curve(yTest_KNN, yTestProbknn)
# ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fprPostPrunedtree, tprPostPrunedtree,  label='Decision
tree-Post-Pruned')
plt.plot(fprlogregclf, tprlogregclf, label='Logistic regression')
plt.plot(fprsvm, tprsvm, label='SVM')
plt.plot(fprnb, tprnb, label='Naïve Bayes')
plt.plot(fprknn, tprknn, label='KNN')
plt.plot([0, 1], [0, 1], 'k--', label='Random selection')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()

#confusion Matrix
sns.heatmap(confusionMatrixknn, annot=True, fmt='d', cmap='Blues')
```

```
plt.title("KNN")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# In[223]:


# Random Forest
from sklearn.ensemble import RandomForestClassifier
removefeature=['Contract_Length_Quarterly','Usage_Frequency']
finalFeature= [item for item in ['Age', 'Tenure', 'Usage_Frequency',
'Support_Calls', 'Payment_Delay',
      'Total_Spend', 'Last_Interaction', 'Gender_Male',
      'Subscription_Type_Premium', 'Subscription_Type_Standard',
      'Contract_Length_Monthly', 'Contract_Length_Quarterly'] if item not in
removefeature]


encodingTrainDFForRF = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
# encodingTrainDFForDT =
encodingTrainDFForDT.drop(columns=['Contract_Length_Quarterly',
'Usage_Frequency', 'Subscription_Type_Premium',
'Subscription_Type_Standard'])
XTrain_RF = encodingTrainDFForRF[finalFeature]
yTrain_RF = encodingTrainDFForRF['Churn']

encodingTestDFForRF = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
XTest_RF = encodingTestDFForRF[finalFeature]
yTest_RF = encodingTestDFForRF['Churn']

# removeFeature= ['Contract_Length_Quarterly','Usage_Frequency']
# XTrain_RF= XTrain_RF.drop(removeFeature,axis=1)
# XTest_RF=XTest_SVM.drop(removeFeature,axis=1)


# In[224]:


#Random Forest grid search
XTrain_RF_GD = XTrain_RF[::4]
yTrain_RF_GD = yTrain_RF[::4]

rf_clf = RandomForestClassifier()

# Define the parameter grid for grid search
param_grid = {
   'n_estimators': [50, 100],
   'max_depth': [ 7, 9,13,11]
}
```

```
grid_search = GridSearchCV(estimator=rf_clf, param_grid=param_grid, cv=3,
scoring='accuracy')
grid_search.fit(XTrain_RF_GD, yTrain_RF_GD)
rfclf = grid_search.best_estimator_
randomForestBestParams= grid_search.best_params_
randomForestBestDepth= grid_search.best_params_['max_depth']
randomForestBestN_estimators= grid_search.best_params_['n_estimators']
print(f'Best parameters found {grid_search.best_params_}')


# In[224]:




# In[225]:


# Random Forest

# rfclf = RandomForestClassifier(n_estimators=100, random_state=5805)
# rfclf.fit(XTrain_RF, yTrain_RF)
rfYTestPred = rfclf.predict(XTest_RF)
rfAccuracy = accuracy_score(yTest_RF, rfYTestPred)
yTestProbrf= rfclf.predict_proba(XTest_RF)[::, -1]
print(f'Random Forest Test accuracy = {rfAccuracy.__round__(5)}')
print(f'Random Forest Train accuracy= {accuracy_score(yTrain_RF_GD,
rfclf.predict(XTrain_RF_GD)).__round__(5)}')


# In[226]:


# Random Forest
yTestProbrf= rfclf.predict_proba(XTest_RF)[::, -1]
confusionMatrixrf = confusion_matrix(yTest_RF, rfYTestPred)
recallrf = recall_score(yTest_RF, rfYTestPred)
rocAucrf = roc_auc_score(yTest_RF, yTestProbrf)
specificityrf = confusionMatrixrf[0, 0] / (confusionMatrixrf[0, 0] +
confusionMatrixrf[0, 1])
f_score_rf = f1_score(yTest_RF, rfYTestPred)

table2 = PrettyTable()
table2.field_names = ["","Accuracy", "confusion Matrix", "recall",
'AUC','Specificity','F-Score']
table2.add_row(["Decision Tree Post-Pruned",postrePrunedAccuracy.round(2),
confusionMatrixPostPruned,recallPostPruned.round(2),rocAucPostPruned.round(2)
, specificityPostPruned.round(3),f_score_PostPruned.round(3)])
table2.add_row(["logistic
regression",logregAccuracy.round(2),confusionMatrixlogreg.round(2),recallLogr
```

```
eg.round(2),rocAucLogreg.round(2),specificityLogreg.round(3),f_score_Logreg.r
ound(3)])
table2.add_row(["SVM",svmAccuracy.round(2),confusionMatrixsvm.round(2),recall
svm.round(2),rocAucsvm.round(2),
specificitysvm.round(3),f_score_svm.round(3)])
table2.add_row(["Naïve
Bayes",nbAccuracy.round(2),confusionMatrixnb.round(2),recallnb.round(2),rocAu
cnb.round(2),specificitynb.round(3),f_score_nb.round(3)])
table2.add_row(["KNN",knnAccuracy.round(2),confusionMatrixknn.round(2),recall
knn.round(2),rocAucknn.round(2),specificityknn.round(3),f_score_knn.round(3)]
)
table2.add_row(["Random
Forest",rfAccuracy.round(2),confusionMatrixrf.round(2),recallrf.round(2),rocA
ucrf.round(2),specificityrf.round(3),f_score_rf.round(3)])
print(table2)


# In[227]:


# Random Forest
fprrf, tprrf, _ = roc_curve(yTest_RF, yTestProbrf)
# ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fprPostPrunedtree, tprPostPrunedtree,  label='Decision
tree-Post-Pruned')
plt.plot(fprlogregclf, tprlogregclf, label='Logistic regression')
plt.plot(fprsvm, tprsvm, label='SVM')
plt.plot(fprnb, tprnb, label='Naïve Bayes')
plt.plot(fprknn, tprknn, label='KNN')
plt.plot(fprrf, tprrf, label='Random Forest')
plt.plot([0, 1], [0, 1], 'k--', label='Random selection')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()

#confusion Matrix
sns.heatmap(confusionMatrixrf, annot=True, fmt='d', cmap='Blues')
plt.title("Random Forest")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# In[228]:


# Bagging
from sklearn.ensemble import RandomForestClassifier
removefeature=['Contract_Length_Quarterly','Usage_Frequency']
```

```
finalFeature= [item for item in ['Age', 'Tenure', 'Usage_Frequency',
'Support_Calls', 'Payment_Delay',
      'Total_Spend', 'Last_Interaction', 'Gender_Male',
      'Subscription_Type_Premium', 'Subscription_Type_Standard',
      'Contract_Length_Monthly', 'Contract_Length_Quarterly'] if item not in
removefeature]


encodingTrainDFForBA = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
# encodingTrainDFForDT =
encodingTrainDFForDT.drop(columns=['Contract_Length_Quarterly',
'Usage_Frequency', 'Subscription_Type_Premium',
'Subscription_Type_Standard'])
XTrain_BA = encodingTrainDFForBA[finalFeature]
yTrain_BA = encodingTrainDFForBA['Churn']

encodingTestDFForBA = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
XTest_BA = encodingTestDFForBA[finalFeature]
yTest_BA = encodingTestDFForBA['Churn']


# In[229]:


# Bagging
from sklearn.ensemble import BaggingClassifier
XTrain_BA_GD = XTrain_BA[::4]
yTrain_BA_GD = yTrain_BA[::4]


ba_clf =
BaggingClassifier(base_estimator=RandomForestClassifier(n_estimators=randomFo
restBestN_estimators, max_depth= randomForestBestDepth), random_state=5805)
param_grid = {
   'n_estimators': [5,15,20],
}
grid_search = GridSearchCV(ba_clf, param_grid, cv=3, scoring='accuracy')
grid_search.fit(XTrain_BA_GD, yTrain_BA_GD)
baclf = grid_search.best_estimator_


# baclf =
BaggingClassifier(base_estimator=RandomForestClassifier(n_estimators=random
ForestBestN_estimators, max_depth= randomForestBestDepth),n_estimators=10,
random_state=5805)
# baclf.fit(XTrain_BA, yTrain_BA)
baYTestPred = baclf.predict(XTest_BA)
baAccuracy = accuracy_score(yTest_BA, baYTestPred)
yTestProbba= baclf.predict_proba(XTest_BA)[:, -1]
print(f'Bagging clf = {baAccuracy.__round__(5)}')
print(f'Best parameters found:{grid_search.best_params_}')
```

```
# In[230]:


# Bagging
yTestProbba= baclf.predict_proba(XTest_BA)[::, -1]
confusionMatrixba = confusion_matrix(yTest_BA, baYTestPred)
recallba = recall_score(yTest_BA, baYTestPred)
rocAucba = roc_auc_score(yTest_BA, yTestProbba)
specificityba = confusionMatrixba[0, 0] / (confusionMatrixba[0, 0] +
confusionMatrixba[0, 1])
f_score_ba = f1_score(yTest_BA, baYTestPred)

table2.add_row(["Bagging",baAccuracy.round(2),confusionMatrixba.round(2),reca
llba.round(2),rocAucba.round(2),specificityba.round(3),f_score_ba.round(3)])
print(table2)


# In[231]:


# Bagging
fprba, tprba, _ = roc_curve(yTest_BA, yTestProbba)
# ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fprPostPrunedtree, tprPostPrunedtree,  label='Decision
tree-Post-Pruned')
plt.plot(fprlogregclf, tprlogregclf, label='Logistic regression')
plt.plot(fprsvm, tprsvm, label='SVM')
plt.plot(fprnb, tprnb, label='Naïve Bayes')
plt.plot(fprknn, tprknn, label='KNN')
plt.plot(fprrf, tprrf, label='Random Forest')
plt.plot(fprba, tprba, label='Bagging')
plt.plot([0, 1], [0, 1], 'k--', label='Random selection')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()

#confusion Matrix
sns.heatmap(confusionMatrixba, annot=True, fmt='d', cmap='Blues')
plt.title("Bagging")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# In[232]:
```

```
# Stacking
from sklearn.ensemble import RandomForestClassifier
removefeature=['Contract_Length_Quarterly','Usage_Frequency']
finalFeature= [item for item in ['Age', 'Tenure', 'Usage_Frequency',
'Support_Calls', 'Payment_Delay',
      'Total_Spend', 'Last_Interaction', 'Gender_Male',
      'Subscription_Type_Premium', 'Subscription_Type_Standard',
      'Contract_Length_Monthly', 'Contract_Length_Quarterly'] if item not in
removefeature]


encodingTrainDFForST = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
# encodingTrainDFForDT =
encodingTrainDFForDT.drop(columns=['Contract_Length_Quarterly',
'Usage_Frequency', 'Subscription_Type_Premium',
'Subscription_Type_Standard'])
XTrain_ST = encodingTrainDFForST[finalFeature]
yTrain_ST = encodingTrainDFForST['Churn']

encodingTestDFForST = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
XTest_ST = encodingTestDFForST[finalFeature]
yTest_ST = encodingTestDFForST['Churn']


# In[233]:


# Stacking
from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier
#, StackingClassifier
from mlxtend.classifier import StackingClassifier
from sklearn.linear_model import LogisticRegression
base_estimators = [('rf',
RandomForestClassifier(n_estimators=randomForestBestN_estimators, max_depth=
randomForestBestDepth)),('gb', GradientBoostingClassifier(n_estimators=100,
random_state=5805))]
final_estimator = LogisticRegression()
# stclf = StackingClassifier(estimators=base_estimators,
final_estimator=final_estimator)

stclf = StackingClassifier(classifiers=[nbclf,rfclf],
meta_classifier=final_estimator,use_probas=True,
use_features_in_secondary=True)

stclf.fit(XTrain_ST, yTrain_ST)

stYTestPred = stclf.predict(XTest_ST)
stAccuracy = accuracy_score(yTest_ST, stYTestPred)
print(f'Stacking = {stAccuracy.__round__(5)}')
```

```
# In[234]:


# Stacking
yTestProbst= stclf.predict_proba(XTest_ST)[::, -1]
confusionMatrixst = confusion_matrix(yTest_ST, stYTestPred)
recallst = recall_score(yTest_ST, stYTestPred)
rocAucst = roc_auc_score(yTest_ST, yTestProbst)
specificityst = confusionMatrixst[0, 0] / (confusionMatrixst[0, 0] +
confusionMatrixst[0, 1])
f_score_st = f1_score(yTest_ST, stYTestPred)

table2.add_row(["Stacking",stAccuracy.round(2),confusionMatrixst.round(2),rec
allst.round(2),rocAucst.round(2),specificityst.round(3),f_score_st.round(3)])
print(table2)
fprst, tprst, _ = roc_curve(yTest_ST, yTestProbst)



#confusion Matrix
sns.heatmap(confusionMatrixst, annot=True, fmt='d', cmap='Blues')
plt.title("Stacking")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# In[253]:


# ROC curves stacking
plt.figure(figsize=(8, 6))

plt.plot(fprst, tprst, label='Stacking')
plt.plot([0, 1], [0, 1], 'k--', label='Random selection')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()


# In[235]:


#Boosting
removefeature=['Contract_Length_Quarterly','Usage_Frequency']
finalFeature= [item for item in ['Age', 'Tenure', 'Usage_Frequency',
'Support_Calls', 'Payment_Delay',
      'Total_Spend', 'Last_Interaction', 'Gender_Male',
      'Subscription_Type_Premium', 'Subscription_Type_Standard',
```

```
      'Contract_Length_Monthly', 'Contract_Length_Quarterly'] if item not in
removefeature]


encodingTrainDFForBOO = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
# encodingTrainDFForDT =
encodingTrainDFForDT.drop(columns=['Contract_Length_Quarterly',
'Usage_Frequency', 'Subscription_Type_Premium',
'Subscription_Type_Standard'])
XTrain_BOO = encodingTrainDFForBOO[finalFeature]
yTrain_BOO = encodingTrainDFForBOO['Churn']

encodingTestDFForBOO = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
XTest_BOO = encodingTestDFForBOO[finalFeature]
yTest_BOO = encodingTestDFForBOO['Churn']


# In[235]:




# In[236]:


#Boosting

XTrain_BOO_GD = XTrain_BOO[::4]
yTrain_BOO_GD = yTrain_BOO[::4]


from sklearn.ensemble import GradientBoostingClassifier

base_classifier = RandomForestClassifier( random_state=5805)
boo_clf = AdaBoostClassifier(base_classifier, random_state=5805)
param_grid = {
   'base_estimator__n_estimators': [10, 20, 50, 100],
}
grid_search = GridSearchCV(boo_clf, param_grid, cv=3, scoring='accuracy')
grid_search.fit(XTrain_BOO_GD, yTrain_BOO_GD)
booclf = grid_search.best_estimator_

# base_classifier = RandomForestClassifier( random_state=5805)
# booclf = AdaBoostClassifier(base_classifier, n_estimators=100,
random_state=5805)
# booclf.fit(XTrain_BOO, yTrain_BOO)
booYTestPred = booclf.predict(XTest_BOO)
booAccuracy = accuracy_score(yTest_BOO, booYTestPred)
print(f'Boosting = {booAccuracy.__round__(5)}')
print(f'Best parameters found : {grid_search.best_params_}')
```

```
# In[238]:


# Boosting
yTestProbboo= booclf.predict_proba(XTest_BOO)[::, -1]
confusionMatrixboo = confusion_matrix(yTest_BOO, booYTestPred)
recallboo = recall_score(yTest_BOO, booYTestPred)
rocAucboo = roc_auc_score(yTest_BOO, yTestProbboo)
specificityboo = confusionMatrixboo[0, 0] / (confusionMatrixboo[0, 0] +
confusionMatrixboo[0, 1])
f_score_boo = f1_score(yTest_BOO, booYTestPred)

table2.add_row(['Boosting',booAccuracy.round(2),confusionMatrixboo.round(2),r
ecallboo.round(2),rocAucboo.round(2),specificityboo.round(3),f_score_boo.roun
d(3)])
print(table2)


fprboo, tprboo, _ = roc_curve(yTest_BOO, yTestProbboo)


#confusion Matrix
sns.heatmap(confusionMatrixboo, annot=True, fmt='d', cmap='Blues')
plt.title("Boosting")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# In[255]:


# ROC curves Boosting
plt.figure(figsize=(8, 6))

plt.plot(fprboo, tprboo, label='Stacking')
plt.plot([0, 1], [0, 1], 'k--', label='Boosting')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()


# In[239]:
```

```
# Neural Network
removefeature=['Contract_Length_Quarterly','Usage_Frequency']
encodingTrainDFForNN = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
standardizedTrainDfNN=
Standardized(encodingTrainDFForNN[['Age','Tenure','Usage_Frequency','Support_
Calls','Payment_Delay','Total_Spend','Last_Interaction']])
standardizedTrainDfNN=pd.concat([standardizedTrainDfNN,encodingTrainDFForNN[[
'Gender_Male','Subscription_Type_Premium','Subscription_Type_Standard','Contr
act_Length_Monthly','Contract_Length_Quarterly']]], axis=1)
# encodingTrainDFForDT =
encodingTrainDFForDT.drop(columns=['Contract_Length_Quarterly',
'Usage_Frequency', 'Subscription_Type_Premium',
'Subscription_Type_Standard'])
XTrain_NN = standardizedTrainDfNN.drop(columns=removefeature)
yTrain_NN = encodingTrainDFForNN['Churn']


encodingTestDFForNN = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
standardizedTestDfNN=
Standardized(encodingTestDFForNN[['Age','Tenure','Usage_Frequency','Support_C
alls','Payment_Delay','Total_Spend','Last_Interaction']])
standardizedTestDfNN =
pd.concat([standardizedTestDfNN,encodingTestDFForNN[['Gender_Male','Subscript
ion_Type_Premium','Subscription_Type_Standard','Contract_Length_Monthly','Con
tract_Length_Quarterly']]], axis=1)

XTest_NN = standardizedTestDfNN.drop(columns=removefeature)
yTest_NN = encodingTestDFForNN['Churn']


# In[241]:


# Neural Network
from sklearn.model_selection import StratifiedKFold, cross_val_predict
from sklearn.neural_network import MLPClassifier

XTrain_NN_GD = XTrain_NN[::4]
yTrain_NN_GD = yTrain_NN[::4]
nn_clf = MLPClassifier(random_state=5805)
param_grid = {
   'hidden_layer_sizes': [(100,), (100, 50), (50, 25, 10)],
   'max_iter': [100, 500]
}
grid_search = GridSearchCV(estimator=nn_clf, param_grid=param_grid, cv=3,
scoring='accuracy')
grid_search.fit(XTrain_NN_GD, yTrain_NN_GD)
nnclf = grid_search.best_estimator_
print(f'Best parameters found: {grid_search.best_params_}')
```

```
# cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=5805)
# nnYTestPred = cross_val_predict(nnclf, XTest_NN, yTest_NN,
cv=cv,method='predict_proba')
# nnAccuracy = accuracy_score(yTest_NN, nnYTestPred.argmax(axis=1))

# nnclf = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=500,
random_state=5805)
# nnclf.fit(XTrain_NN, yTrain_NN)
nnYTestPred=nnclf.predict(XTest_NN)
nnAccuracy = accuracy_score(yTest_NN, nnYTestPred)
print(f'Neural Network = {nnAccuracy.__round__(5)}')


# In[242]:


# Neural Network
yTestProbnn= nnclf.predict_proba(XTest_NN)[::, -1]


confusionMatrixnn = confusion_matrix(yTest_NN, nnYTestPred)
recallnn = recall_score(yTest_NN, nnYTestPred)
rocAucnn = roc_auc_score(yTest_NN, nnYTestPred)
specificitynn = confusionMatrixnn[0, 0] / (confusionMatrixnn[0, 0] +
confusionMatrixnn[0, 1])
f_score_nn = f1_score(yTest_NN, nnYTestPred)



table2.add_row(["Neural
Network",nnAccuracy.round(2),confusionMatrixnn.round(2),recallnn.round(2),roc
Aucnn.round(2),specificitynn.round(3),f_score_nn.round(3)])
print(table2)

fprnn, tprnn, _ = roc_curve(yTest_NN, nnYTestPred)


# In[256]:


# ROC curves Boosting
plt.figure(figsize=(8, 6))

plt.plot(fprnn, tprnn, label='Stacking')
plt.plot([0, 1], [0, 1], 'k--', label='Neural Network')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()
```

```
# In[243]:


# Neural Network
# ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fprPostPrunedtree, tprPostPrunedtree,  label='Decision
tree-Post-Pruned')
plt.plot(fprlogregclf, tprlogregclf, label='Logistic regression')
plt.plot(fprsvm, tprsvm, label='SVM')
plt.plot(fprnb, tprnb, label='Naïve Bayes')
plt.plot(fprknn, tprknn, label='KNN')
plt.plot(fprrf, tprrf, label='Random Forest')
plt.plot(fprba, tprba, label='Bagging')
plt.plot(fprnn, tprnn, label='Neural Network')
plt.plot([0, 1], [0, 1], 'k--', label='Random selection')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()


#confusion Matrix
sns.heatmap(confusionMatrixnn, annot=True, fmt='d', cmap='Blues')
plt.title("Neural Network")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# In[261]:




# In[ ]:


# Phase IV: Clustering and Association


# In[257]:


# k-mean
encodingTrainDFForKM = pd.get_dummies(training_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
standardizedTrainDfKM=
Standardized(encodingTrainDFForKM[['Age','Tenure','Usage_Frequency','Support_
Calls','Payment_Delay','Total_Spend','Last_Interaction']])
```

```
standardizedTrainDfKM=pd.concat([standardizedTrainDfKM,encodingTrainDFForKM[[
'Gender_Male','Subscription_Type_Premium','Subscription_Type_Standard','Contr
act_Length_Monthly','Contract_Length_Quarterly']]], axis=1)
# encodingTrainDFForDT =
encodingTrainDFForDT.drop(columns=['Contract_Length_Quarterly',
'Usage_Frequency', 'Subscription_Type_Premium',
'Subscription_Type_Standard'])
XTrain_KM = standardizedTrainDfKM
yTrain_KM = encodingTrainDFForKM['Churn']

encodingTestDFForKM = pd.get_dummies(testing_df, columns=['Gender',
'Subscription_Type', 'Contract_Length'],drop_first=True)
standardizedTestDfKM=
Standardized(encodingTestDFForKM[['Age','Tenure','Usage_Frequency','Support_C
alls','Payment_Delay','Total_Spend','Last_Interaction']])
standardizedTestDfKM =
pd.concat([standardizedTestDfKM,encodingTestDFForKM[['Gender_Male','Subscript
ion_Type_Premium','Subscription_Type_Standard','Contract_Length_Monthly','Con
tract_Length_Quarterly']]], axis=1)

XTest_KM = standardizedTestDfKM
yTest_KM = encodingTestDFForKM['Churn']

removeFeature= ['Contract_Length_Quarterly','Usage_Frequency']
XTrain_KM= XTrain_KM.drop(removeFeature,axis=1)
XTest_KM=XTest_KM.drop(removeFeature,axis=1)


# In[258]:


# K-mean
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
k_values = range(2, 11)
silhouette_scores = []

for k in k_values:
   kmeans = KMeans(n_clusters=k, random_state=5805)
   cluster_labels = kmeans.fit_predict(XTrain_KM)

   silhouette_avg = silhouette_score(XTrain_KM, cluster_labels)
   silhouette_scores.append(silhouette_avg)

# Plot silhouette scores for different k values
plt.plot(k_values, silhouette_scores, marker='o')
plt.title('Silhouette Analysis for K-means Clustering')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.show()
```

```
# K-mean
inertia_values = []
k_values = range(2, 60)
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=5805)
    kmeans.fit(XTrain_KM)
    inertia_values.append(kmeans.inertia_)

plt.plot(k_values, inertia_values, marker='o')
plt.title('Elbow Method for Optimal k in K-means Clustering')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.show()


# In[272]:


# Apriori algorithm
# training_df['Tenure'].unique()
kmean_clusterDF=copy.deepcopy(training_df)
# divide
kmean_clusterDF['Tenure'] = pd.cut(kmean_clusterDF['Tenure'], bins=3,
labels=['Tenure_Short', 'Tenure_Medium', 'Tenure_Long'])
kmean_clusterDF['Usage_Frequency'] =
pd.cut(kmean_clusterDF['Usage_Frequency'], bins=2,
labels=['Usage_Frequency_Less', 'Usage_Frequency_High'])
kmean_clusterDF['Support_Calls'] = pd.cut(kmean_clusterDF['Support_Calls'],
bins=3, labels=['Support_Calls_Low',
'Support_Calls_Medium','Support_Calls_High'])
kmean_clusterDF['Payment_Delay'] = pd.cut(kmean_clusterDF['Payment_Delay'],
bins=3, labels=['No_Payment_Delay',
'Minor_Payment_Delay','High_Payment_Delay'])
kmean_clusterDF['Total_Spend'] = pd.cut(kmean_clusterDF['Total_Spend'],
bins=3, labels=['Spend_Low','Spend_Medium','Spend_High'])
kmean_clusterDF['Last_Interaction'] =
pd.cut(kmean_clusterDF['Last_Interaction'], bins=3,
labels=['Interaction_Low','Interaction_Average','Interaction_High'])
kmean_clusterDF['Age'] = pd.cut(kmean_clusterDF['Age'], bins=2,
labels=['Child','Adult'])
kmean_clusterDF['Churn'] = pd.cut(kmean_clusterDF['Churn'], bins=2,
labels=['NO_Churn','Churn'])

kmean_clusterDF['Subscription_Type'] =
kmean_clusterDF['Subscription_Type'].replace({
    'Premium': 'Subscription_Type_Premium',
    'Standard': 'Subscription_Type_Standard',
    'Basic': 'Subscription_Type_Basic'
})

kmean_clusterDF['Contract_Length'] =
kmean_clusterDF['Contract_Length'].replace({
```

```
    'Annual': 'Contract_Length_Annual',
    'Monthly': 'Contract_Length_Monthly',
    'Quarterly': 'Contract_Length_Quarterly'
})
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
subsetData=kmean_clusterDF
transactions = subsetData.values.tolist()
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)
# print(df_encoded)
for i,j in df_encoded.dtypes.items():
    if j=='bool':
        df_encoded[i]= df_encoded[i].astype(int)

frequent_itemsets = apriori(df_encoded, min_support=0.22, use_colnames=True)
rules = association_rules(frequent_itemsets, metric='confidence',
min_threshold=0.7)
# Display results
rules


# In[274]:


rules.columns


# In[90]:


# Apriori algorithm
print("Association Rules:")
print(rules)


# In[275]:


# Apriori algorithm
frequent_itemsets
frequent_itemsets_sorted = frequent_itemsets.sort_values(by='support',
ascending=False)
frequent_itemsets_sorted


# In[ ]:


print(frequent_itemsets_sorted)
```

```
# k-mean
n_clusters = 2

# Create KMeans instance
kmeans = KMeans(n_clusters=n_clusters, random_state=5805)

features_for_clustering= XTrain_KM.columns

# Fit and predict cluster labels
XTrain_KM['cluster_label'] = kmeans.fit_predict(XTrain_KM)



# In[65]:


#K-mean
removefeature=[]
finalFeature= [item for item in ['Age', 'Tenure', 'Usage_Frequency',
'Support_Calls', 'Payment_Delay',
      'Total_Spend', 'Last_Interaction', 'Gender_Male',
      'Subscription_Type_Premium', 'Subscription_Type_Standard',
      'Contract_Length_Monthly', 'Contract_Length_Quarterly'] if item not in
removefeature]



XCat_KM=  training_df[['Gender', 'Subscription_Type', 'Contract_Length']]
XNum_KM=
training_df[['Age','Tenure','Usage_Frequency','Support_Calls','Payment_Delay'
,'Total_Spend','Last_Interaction']]


print(training_df.head(5))


# In[66]:


#K-mean
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from kmodes.kmodes import KModes
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# numerical_features = XNum_KM
# categorical_features = XCat_KM

all_features = pd.concat([XNum_KM, XCat_KM], axis=1)
```

```
scaler = StandardScaler()
# numerical_features_scaled = scaler.fit_transform(numerical_features)

preprocessor = ColumnTransformer(
   transformers=[
       ('num', scaler, XNum_KM.columns),
       ('cat', 'passthrough', XCat_KM.columns)
   ])

pipeline = Pipeline([
   ('preprocessor', preprocessor),
   ('kmeans', KModes(n_clusters=2, init='Huang', n_init=5, verbose=1))
])

pipeline.fit(all_features)


#K-mean
cluster = pd.Series(pipeline.named_steps['kmeans'].labels_)

# Visualize the clusters
cluster_size = cluster.value_counts()
plt.bar(cluster_size.index, cluster_size.values)
plt.xlabel('Cluster')
plt.ylabel('Number of Samples')
plt.title('Cluster Sizes')
plt.show()



#K-mean
cluster_sizes = [50, 30]

# Assuming 'cluster_series' is the Pandas Series containing cluster labels
cluster_counts = cluster.value_counts()

# Plotting a pie chart
plt.figure(figsize=(8, 8))
plt.pie(cluster_counts, labels=cluster_counts.index, autopct='%1.1f%%',
startangle=90)
plt.title('Cluster Distribution')
plt.show()


# In[ ]:


#K-mean
print('The sizes of the clusters are imbalanced, with the 0th cluster having
the largest number of observations (175,000), followed by the 1st cluster
(108,000) and the 2nd cluster (90,000) \nThe 0th cluster is significantly
larger than the other clusters, suggesting that it might represent a more
```

```
dominant or prevalent group in the dataset.\nThe large size of the dominant
cluster could pose challenges in terms of interpretability. It might be more
challenging to distinguish unique patterns within this cluster due to its
size\nThe presence of multiple clusters indicates that there are distinct
subgroups in the data. Each cluster may represent a different pattern or
behavior among the observations\nThe results of association rule mining would
depend on the specific features used and the relationships explored. The
imbalanced cluster sizes may influence the rules generated, and its essential
to consider the context of the analysis\n ')
```

References

Plotly. (n.d.). "Plotly Express Documentation." https://www.plotly.com/python/plotly-express/

Hastie, T., Tibshirani, R., & Friedman, J. (2009). "The Elements of Statistical Learning: Data Mining, Inference, and Prediction." Springer

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 12(Oct), 2825-2830.