DATA 583 Project
Iranian Churn Data Analysis

Tim Pulfer, Jacob Rosen, Dhun Sheth

Abstract

This project investigates customer churn within a dataset, exploring variables predicting churn, thresholds for churn-inducing call failures, and the relationship between call failures and customer complaints. It examines whether thresholds differ for high and low-value customers and estimates potential churn reduction resulting from a proposed solution to decrease call failures by 50%. Statistical and machine learning techniques are employed, starting with exploratory data analysis, fitting distributions, and assessing feature importance. Supervised and unsupervised modeling approaches are utilized, with a thorough examination of model results and simulations to predict churn. The final resulting model used to simulate new sample data is a boosting tree classification model. Key findings include identifying that more than 9 call failures for a customer is likely to result in them churning, the number of call failures doesn't necessarily result in a customer complaint, and a 50% reduction in call failures would yield an approximate 6% reduction in churn whereas a 50% reduction in inactivity would lead to ~23% reduction in churn. The project provides actionable strategies for mitigating churn by improving service quality, offering valuable insights for informed decision-making in customer retention strategies.

Table of Contents

Introduction

The project will aim to analyze churn data from an Iranian telecommunications company. The goal is to understand the relationship between customer interactions to the rate at which they churn. We hypothesize that call failures and complaints will be key predictors of churn, but modeling the relationship with other predictors could give the company a clearer idea of the factors contributing to customer churn. The model can also be used for scenario forecasting, in which the company can predict how certain changes in customer interactions will affect churn rates. In addition, the company can implement a proactive approach to customer retention by identifying customers at risk of churning and implementing aggressive marketing and retention campaigns to target this specific customer group.

The following are the five scientific questions explored in this report that will help the company better understand the relationship between customer interactions and churn rates:

1. Which variables are most important for predicting customer churn?

We hypothesize that call failures and complaints will be key predictors of churn considering they are indications of negative user experience.

2. Does customer value affect the likelihood of a customer churning?

We hypothesize that customer value will not affect the likelihood of churn. This is an important question to explore so that the company can prioritize factors that affect churn in high-valued customers if we detect a difference.

3. What is the call failure rate at which we expect a customer to churn?

Assuming that call failure rate is a key predictor of churn, we believe that providing the company with an expected threshold will give greater control over customer retention. The idea is to provide some number of call failures that result in a higher likelihood of a customer churning to then allow the company to establish preventative measures to ensure it is not exceeded.

4. What is the number of call failures before we can expect a complaint?

Assuming that there is a relationship between call failures and complaints and that complaints predict churn, we want to investigate the threshold for when call failures result in a complaint. Similar to churn, by giving the company a tangible number it can then set preventative measures to ensure this threshold is not crossed, thereby possibly increasing customer satisfaction and retention.

5. What is the expected reduction in churn if a proposed business solution can reduce call failures by 50%?

Based on the hypothesis that churn is largely predicted by call failures, we explore how efforts in call failure reduction will help the retention of customers.

Modeling Concepts

Principal Component Analysis

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction, which simplifies the complexity of high-dimensional data while retaining most, if not all of the information. It identifies the directions (principal components) that maximize the variance in the data, transforming the original data points into a new coordinate system. This process is often used to reduce the number of variables in a data set, making data visualization

and further analysis more manageable. Additionally, PCA is often used to reduce or eliminate multicollinearity between predictors.

Supervised vs. Unsupervised

Supervised and unsupervised learning are two main categories of machine learning algorithms. Supervised learning involves training a model on a labeled dataset, which means that each training example is paired with an output label. The model learns to predict the output from the input data, making it suitable for tasks such as classification and regression. In contrast, unsupervised learning deals with datasets without predefined labels, and the goal is to discover inherent patterns, relationships, or structures within the data. Common unsupervised learning tasks include clustering, where the algorithm groups similar data points together, and dimensionality reduction, which simplifies data without losing essential information.

Boosting Tree Classification

Boosting Tree Classification is a machine learning ensemble technique that combines multiple weak learners, typically decision trees, to form a strong classifier. The core idea is to train decision trees sequentially, with each tree attempting to correct the errors made by the previous ones. Boosting involves adjusting the weights of incorrectly classified instances so that subsequent trees focus more on difficult cases. This process continues until a specified number of trees are created or no further improvement can be made. The final model makes predictions based on a weighted vote of all the trees' decisions.

Data Collection and Preparation

The dataset used for this project was collected by an Iranian telecommunications company (cite). It comprises a comprehensive overview of customer interactions and characteristics within a telecommunications context over 9 months, focusing on understanding customer churn. It consists of 13 variables, which are a mix of numerical, binary, and ordinal types. These variables capture a wide range of information, including customer usage patterns, service satisfaction, subscription details, demographic profiles, and overall customer value. The database has a total of 3150 rows, each representing a customer.

Data Transformations

The dataset contained two binary variables ('Status' and 'Tariff Plan') that were stored as categorical variables (using levels 1 and 2). These variables were converted to binary variables. The columns ‘Status’ and ‘Tariff Plan’ were renamed for clarity and all other columns were renamed to reduce the space between words from two to one, for consistency. The column ‘Call Failure Rate’ was created by taking the amount from ‘Call Failure’ divided by ‘Frequency of Use’. NaN values in the ‘Call Failure Rate’ column were replaced with the column’s mean, ensuring all data points were usable for analysis. Acknowledging the critical role of data normalization, we split the dataset into training and test sets, applying standard scaling to facilitate more effective model training.

Exploratory Data Analysis

Outliers

Cook’s distance was considered to detect outliers; however, this metric measures the effect the deletion of an observation has on a linear regression line. When considering the complex relationships and various modeling options, deleting

observations based on the effect it has on the linear relationship did not seem sensible. Also considering the imbalanced nature of the dataset, since the occurrence of churn is rare, it was hypothesized that some of the extreme values may hold some predictive information for customer churn. As such, for the outlier analysis, no outliers were removed. Values were analyzed to ensure they made sense and no illogical values were observed.

Balance of Response

Of the 3150 customers represented in the dataset only 500 churned, which makes up 15% of all observed values. The imbalance in our response variable can pose an issue in some instances of supervised learning since it could bias our model predictions. To prevent this, we set the 'class_weights' parameter in each model to 'balanced' which automatically adjusts the weights of classes in the training data inversely proportional to their frequencies. This helps against the bias in the dataset as it gives more importance to the minority class, improving the model's ability to generalize and make accurate predictions for both classes. We also use the F1 score as our main model evaluation metric instead of misclassification rate which would not be appropriate given the imbalance.

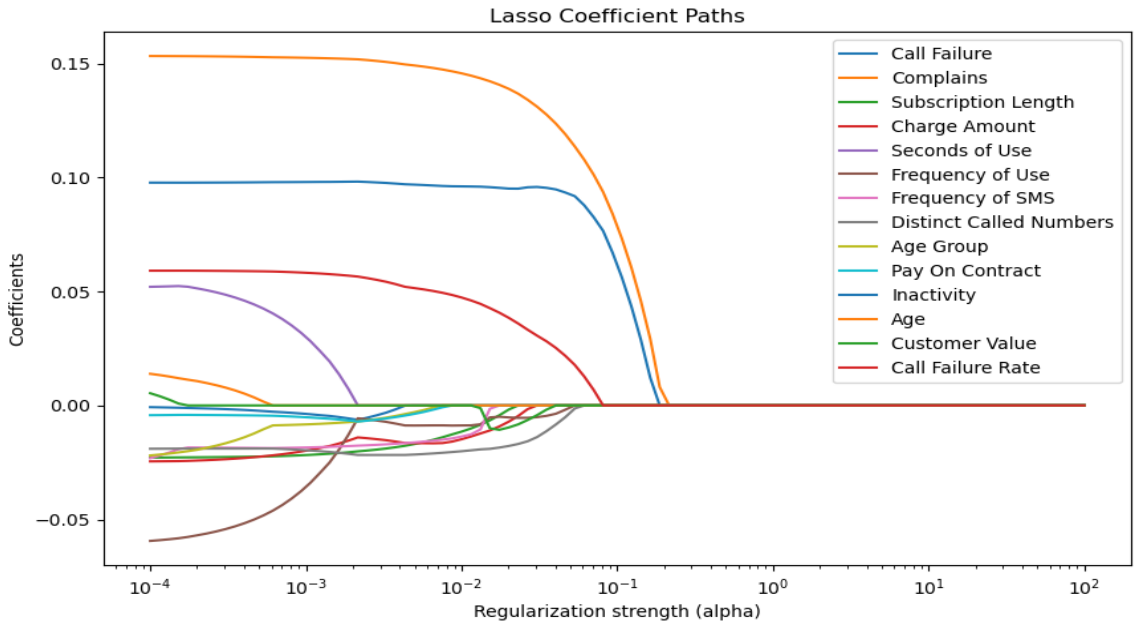
Fitting Distributions on Predictors

Distributions were fit on all predictors for exploration and resampling purposes. The binary variables were fit with a binomial distribution. A Poisson distribution was tried for all the count variables but in all cases was deemed an inappropriate fit. In the end, all numeric variables were fit using Kernel Density Estimation, except for Age/Age Group and Charge Amount. Age/Age Group was fit with a normal distribution and Charge Amount was fit with an exponential distribution.

Features Evaluation of Importance

LASSO

In the Lasso Coefficient Path plot, the most significant variables appear to be 'Complains' and 'Inactivity' as evidenced by their coefficient paths which remain substantially non-zero across a wide range of regularization strengths (alpha). This indicates that these predictors have a strong and stable influence on the response variable; they are robust against the shrinkage effect induced by increasing alpha values. Their coefficients are the last to converge towards zero, suggesting they are essential features for the model. On the other hand, variables like 'Seconds of Use' and 'Pay On Contract' seem to be much less significant. Their paths quickly drop to zero as alpha increases, showing that these variables are easily discarded by the Lasso model when regularization is applied. This behaviour indicates that they contribute less to the model's predictive power and are therefore considered less important in the presence of other variables.



Call Failure	Complains	Subscription Length	Charge Amount	Seconds of Use	Frequency of Use	Frequency of SMS	Distinct Called Numbers	Age Group	Pay On Contract	Inactivity	Age	Customer Value	Call Failure Rate
-0.0	0.143488	-0.008482	-0.012295	-0.0	-0.008268	-0.010526	-0.019271	-0.0	-0.0	0.095938	-0.0	-0.001187	0.044549
-0.0	0.142172	-0.006522	-0.010987	-0.0	-0.005244	-0.001463	-0.019035	-0.0	-0.0	0.095723	-0.0	-0.010018	0.042670
-0.0	0.140632	-0.004321	-0.009214	-0.0	-0.004962	-0.000000	-0.018476	-0.0	-0.0	0.095432	-0.0	-0.010685	0.040686
-0.0	0.138857	-0.001798	-0.007111	-0.0	-0.005254	-0.000000	-0.017783	-0.0	-0.0	0.095102	-0.0	-0.009662	0.038428
-0.0	0.136714	-0.000000	-0.004618	-0.0	-0.005448	-0.000000	-0.016901	-0.0	-0.0	0.095068	-0.0	-0.008303	0.035924
-0.0	0.134057	-0.000000	-0.001615	-0.0	-0.005421	-0.000000	-0.015714	-0.0	-0.0	0.095675	-0.0	-0.006396	0.033219
-0.0	0.131027	-0.000000	-0.000000	-0.0	-0.005139	-0.000000	-0.013874	-0.0	-0.0	0.095822	-0.0	-0.004256	0.030614
-0.0	0.127569	-0.000000	-0.000000	-0.0	-0.004572	-0.000000	-0.011273	-0.0	-0.0	0.095433	-0.0	-0.001845	0.028130
-0.0	0.123571	-0.000000	-0.000000	-0.0	-0.003635	-0.000000	-0.008342	-0.0	-0.0	0.094738	-0.0	-0.000000	0.025219
-0.0	0.118926	-0.000000	-0.000000	-0.0	-0.001870	-0.000000	-0.005131	-0.0	-0.0	0.093370	-0.0	-0.000000	0.021750
-0.0	0.113583	-0.000000	-0.000000	-0.0	-0.000000	-0.000000	-0.001330	-0.0	-0.0	0.091775	-0.0	-0.000000	0.017752
-0.0	0.107770	-0.000000	-0.000000	-0.0	-0.000000	-0.000000	-0.000000	-0.0	-0.0	0.087786	-0.0	-0.000000	0.012705
-0.0	0.101192	-0.000000	-0.000000	-0.0	-0.000000	-0.000000	-0.000000	-0.0	-0.0	0.082578	-0.0	-0.000000	0.006797
-0.0	0.093628	-0.000000	-0.000000	-0.0	-0.000000	-0.000000	-0.000000	-0.0	-0.0	0.076590	-0.0	-0.000000	0.000004
-0.0	0.084067	-0.000000	-0.000000	-0.0	-0.000000	-0.000000	-0.000000	-0.0	-0.0	0.067029	-0.0	-0.000000	0.000000

VIF

Our team focused on optimizing a dataset for predictive modeling, with a particular emphasis on addressing multicollinearity and enhancing feature selection. Minimizing multicollinearity is important for ensuring the interpretability, precision, and stability of the estimated coefficients, for linear models. High multicollinearity among predictor variables can lead to inflated standard errors, making it difficult to discern the individual impacts of variables and potentially resulting in misleading conclusions. Although this is less of a concern for tree-based models which work well with complex data and can handle such relationships between features, we did not initially know if a linear or tree-

based model would be chosen. In addition, with the independent sampling from densities to be done later, reducing multicollinearity would only help better our results.

Our approach initiated with an assessment of the Variance Inflation Factor (VIF) to identify and subsequently exclude features that presented high correlation, potentially skewing model results. This process led us to decisively remove 'Age', 'Seconds of Use', and 'Frequency of SMS', recognizing their redundant correlations with other variables. Moreover, ‘Age’ was removed because in the data set, only 5 ages were present, 15, 25, 30, 45, and 55. As such, to avoid any potential decision boundaries that could lie on ages not captured in the original data, ‘Age Group’ was used instead, and ‘Age’ was removed for all models.

To further refine our dataset, we introduced the ‘Average Sec per Call’ feature, a strategic move that allowed us to encapsulate relevant information from two original features while adeptly managing missing values through mean imputation. Our team took great care in reordering the dataset, ensuring the ‘Churn’ variable was placed at the end for ease of access during model training.

Our iterative process of VIF analysis and feature exclusion was rigorous, and specifically aimed at eliminating any remaining features that could contribute to multicollinearity. The final subset of predictors can be seen in the below image with their corresponding VIF values - notice they are all between 1 and 5, indicating the initial goal of minimizing multicollinearity was achieved.

```
{'const': 37.878955065474166,
 'Call Failure': 3.8799041873195628,
 'Complains': 1.1634499860765335,
 'Subscription Length': 1.1935637966059744,
 'Charge Amount': 2.4756125328811613,
 'Frequency of Use': 3.6161356956007777,
 'Distinct Called Numbers': 2.4345454252933223,
 'Age Group': 1.2615035105810766,
 'Pay On Contract': 1.3602007816093478,
 'Inactivity': 1.8570667879548919,
 'Customer Value': 1.4810279752825513,
 'Call Failure Rate': 2.0628915783890345,
 'Average Sec per Call': 1.2110893650675132}
```

Significance Test

Columns that are likely to fail the significance test at predicting churn based on z-score:

1. Customer Value
2. Seconds of Use
3. Frequency of SMS
5. Frequency of Use
6. Distinct Called Numbers

```
{'Call Failure': -0.044713908423587405,
 'Complains': -1.4909644129411967,
 'Subscription Length': -0.004803669460067019,
 'Charge Amount': -0.05596636778159771,
 'Seconds of Use': 0.4134023111059113,
 'Frequency of Use': 0.3832342546417229,
 'Frequency of SMS': 0.3506855753343583,
 'Distinct Called Numbers': 0.30245884315673255,
 'Age Group': -0.2581308989667535,
 'Pay On Contract': -0.8518335480541744,
 'Inactivity': -1.8667652346109505,
 'Age': -0.01243243611162162,
 'Customer Value': 0.4293941629761614,
 'Call Failure Rate': -1.3498067472208841,
 'Average Sec per Call': -0.003159812627362492}
```

It should be noted that this test assumes a normal distribution, which may be true for a large enough sample size but our data set has less than 4000 rows and is imbalanced for the desired response. As such, these scores do not necessarily reflect an accurate measure of variable importance.

Model Fitting

The model fitting process was iterative, and several configurations and models were tested. As mentioned earlier, due to a small set of age values, the 'Age' column was replaced by 'Age Group' for all models in all instances.

The same 75% train and 25% test set was used for all models to allow for accurate performance comparison.

Four supervised learning techniques were used to fit predictive models with all predictor features and with the optimal subset of features to test which model fit would perform best on the test set. The modeling techniques used were logistic regression (LR), support vector classification (SVC), random forest (RF), and boosting tree classification (BTC). The models fitted with the full set of predictors included the additional engineered columns mentioned above. The models fitted with the optimal subset only included features that resulted in a VIF value of less than five. Cross-validation metrics were recorded for all 8 models in addition to the test set performance. The results are summarized below.

An idea to potentially reduce multicollinearity further was to run principal component analysis (PCA) on the initial data, keep all principal components, and then map the new simulated samples in the same PCA space to take advantage of the PCA assumption of no multicollinearity between principal components. As long as the covariance structure of the simulated data matched that of the original data, the idea seemed valid, and based on the initial testing results using the training and test set, the VIF of all predictors from the test set was very close to 1. However, when the same test was run on the original and simulated data, the VIF of several features was significantly above 10, so this idea was not pursued further.

Under unsupervised methods, a k-means and Gaussian mixture model were used to cluster the data into 2 groups.

Count of Non-Churn and Churn for Training Set Used in All Models

Class	Count
Non-churn: 0	2,010
Churn: 1	352

Count of Non-Churn and Churn for Test Set Used in All Models

Class	Count
Non-churn: 0	645
Churn: 1	143

Supervised Model Fitting

Model Fit on all Features plus Engineered Features

Cross-Validation Results

Model	F1		ROC AUC		Balanced Accuracy		Matthews Correlation Coefficient	
	Mean	Stdv	Mean	Stdv	Mean	Stdv	Mean	Stdv
LR	0.64424	0.03037	0.93962	0.0137	0.8654	0.02488	0.59526	0.03895
SVC	0.72162	0.0334	0.96394	0.00812	0.89792	0.02186	0.68147	0.04018
RF	0.84888	0.02789	0.98384	0.00699	0.89658	0.02577	0.82673	0.02949
BTC	0.88163	0.0237	0.98753	0.00459	0.93345	0.01715	0.8612	0.02799

Test Set

Model	F1	Error rate	AUC
LR	0.65229	0.35667	0.922
SVC	0.74928	0.20245	0.955
RF	0.84058	0.145	0.98196
BTC	0.85517	0.13913	0.98454

Model Fit on Engineered and Optimal VIF Features

Cross-Validation Results

Model	F1		ROC AUC		Balanced Accuracy		Matthews Correlation Coefficient	
	Mean	Stdv	Mean	Stdv	Mean	Stdv	Mean	Stdv
LR	0.63843	0.03162	0.939	0.01401	0.86066	0.02825	0.58771	0.04197
SVC	0.73445	0.03515	0.96453	0.00937	0.90072	0.02281	0.69501	0.04225
RF	0.84406	0.02554	0.98172	0.00906	0.89344	0.02516	0.82134	0.02625
BTC	0.88225	0.02971	0.98759	0.00512	0.93577	0.0206	0.86195	0.0352

Test Set

Model	**F1**	**Error rate**	**AUC**
LR	0.65583	0.3598	0.92104
SVC	0.76023	0.19822	0.95632
RF	0.84477	0.14082	0.98266
BTC	0.875	0.12558	0.98707

Unsupervised Model Fitting

K-Means

Confusion Matrix on Test Set

	Predicted 0	Predicted 1
Actual 0	288	357
Actual 1	6	137

Mixture Model

Confusion Matrix on Test Set

	Predicted 0	Predicted 1
Actual 0	494	151
Actual 1	18	125

Model Conclusion

Based on the above results, the unsupervised models were not considered further. The K-Means clustering was not optimal compared to the supervised models. In addition, the Mixture Model performed well on classifying the Churned group, but it misclassified many non-churn customers as churned - although this is not that big of a deal, the supervised models performed better in comparison.

The models that used the optimal subset of features, specifically the Boosting Tree Classifier, performed better overall on the test set and cross-validated metrics than the models trained with all features. However, looking at the cross-validated mean and standard deviation of scoring metrics, we see both the subset and original BTC are very close.

Initially, the BTC subset is the favourable choice because it performed slightly better, and more importantly, the VIF of the features chosen are between 1 and 5, indicating minimal multicollinearity. This is important because we will be simulating new data independently from the KDE's we fit earlier and using the model to predict on these simulated data. If we attempt to use the BTC (original) model with all columns, the independence assumption is violated, as evidenced in the VIF analysis, and our predictions on the simulated data are less likely to be accurate.

We also attempted to use PCA to remove multicollinearity by sampling the independently fitted KDE's and transforming those values into the PCA space - which would theoretically ensure independence among the features via PCA assumptions. However, it did not transfer to the original and simulated data, which will be shown later. As such, even though the models trained on the PCA transformed data were promising, due to the multicollinearity not generalizing on the sampled data, this idea was dropped.

VIF on Test Set


```
{'const': 1.0102717894086974,
 'Call Failure': 1.02064268731066,
 'Complains': 1.0559072698929224,
 'Subscription Length': 1.0658070078802182,
 'Charge Amount': 1.0140449469719175,
 'Seconds of Use': 1.0128563339072594,
 'Frequency of Use': 1.0486651311584543,
 'Frequency of SMS': 1.0597606822203012,
 'Distinct Called Numbers': 1.0707868323872947,
 'Age Group': 1.0278024494120153,
 'Pay On Contract': 1.0365640933874802,
 'Inactivity': 1.1067183414866855,
 'Customer Value': 1.1794284619563509,
 'Call Failure Rate': 1.0520129247338759,
 'Average Sec per Call': 1.0235346124125106}
```

Given the results explained above, we chose to move forward with the BTC model trained with the subset of features. Even though the BTC (original) showed more stability in the cross-validated metrics by having a lower standard deviation, the BTC (subset) model had a better performance on the test set and higher means for CV scoring metrics. Finally, because the optimal VIF predictor columns were used, multicollinearity was minimized unlike in the BTC (original) model, and as mentioned before, although this is not a major concern for tree-based models which can handle such relationships between features, this would help when sampling from independent densities.

Simulating New Data

Applying Kernel Density Estimation (KDE) separately to each feature inherently presumes that there is no correlation among the features. Contrary to this assumption, the Variance Inflation Factor (VIF) analysis reveals significant correlations among features. Nevertheless, Principal Component Analysis (PCA) operates under the premise that the principal components are orthogonal to each other, thus eliminating the correlation between these components. Therefore, by transforming the simulated data into the PCA space, and ensuring that the covariance structure of the sampled data aligns with that of the original training data, the features in the PCA-transformed space of the sampled data can be considered uncorrelated.

However, upon closer examination, it becomes evident that there are hidden variables and, quite possibly, complex non-linear relationships between some of the predictors. This complexity suggests that the interactions among variables are not entirely captured by our current simulation approach. Given that we opted to simulate data for each predictor independently, rather than employing a more integrated multivariate simulation method, some limitations have surfaced. Specifically, despite our efforts to isolate and simulate each variable individually, we still encounter instances of multicollinearity within the PCA mapping of our generated samples. This phenomenon is highlighted in the figure provided below, indicating that the interdependencies among predictors persist in our synthetic dataset, reflecting a nuanced challenge in our simulation process.

VIF Test on Generated Samples

{'const': 1.3666065703377472,
 'Call Failure': 1.8704713560278436,
 'Complains': 17.554608095042752,
 'Subscription Length': 40.75946103458566,
 'Charge Amount': 120.61178295120015,
 'Seconds of Use': 9.685384594908435,
 'Frequency of Use': 87.13641733824014,
 'Frequency of SMS': 125.7549720734526,
 'Distinct Called Numbers': 1.9651781400930308,
 'Age Group': 2.7194350100456535,
 'Pay On Contract': 3.510393411426643,
 'Inactivity': 47.64645110436364,
 'Customer Value': 21.62742612112231,
 'Call Failure Rate': 7.769975179925251,
 'Average Sec per Call': 1.088554847395684}

KDE on Individual Features

We generate a synthetic dataset tailored to mirror real-world data characteristics without using actual personal information. The process starts by assigning statistical models to various data features, such as call failures and subscription lengths, to ensure realism. We use techniques like Kernel Density Estimations for some features and simple probability distributions for others, like whether a customer complains. After generating the initial data, we refine it by categorizing continuous variables, enforcing logical rules (like the number of call failures cannot exceed total calls), and adjusting values to realistic thresholds. We also calculate additional metrics, such as call failure rates, to enhance the dataset's depth. The final step involves organizing and cleaning the dataset, ensuring it's consistent and filled where data might be missing. This meticulous approach results in a detailed yet entirely synthetic dataset, ready for analytical and modeling purposes, sidestepping privacy concerns associated with real data.

Scientific Questions

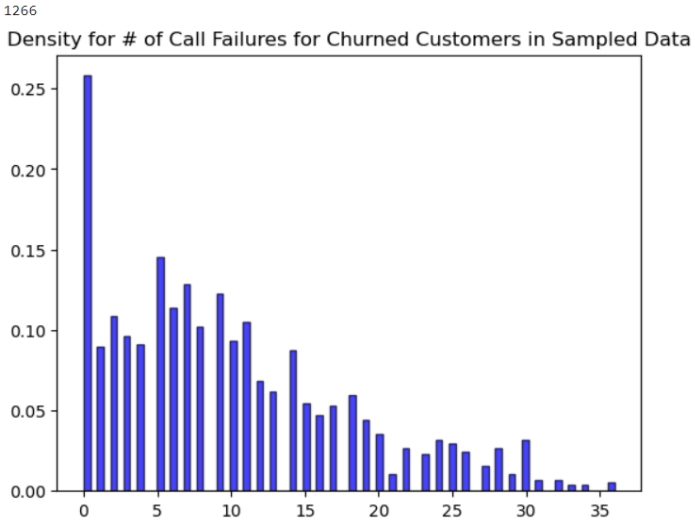
1. **Which variables are most important for predicting customer churn?**

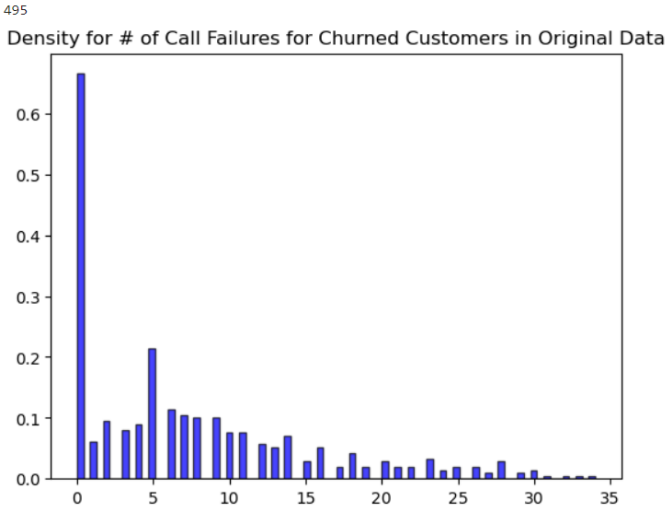
This was answered based on the output from LASSO. The 2 most important features were Complaints and Inactivity, followed by Call Failure Rate, Distinct Numbers Called, and Customer Value. The least important features included Pay On Contract and Seconds of Use. Call Failure was also very low, but we assumed this to be the case due to the presence of the engineered feature, Call Failure Rate, which combined number of Call Failures and Frequency of Use.

2. **Does customer value affect the likelihood of a customer churning?**

As customer value is a relatively important feature, identified by the LASSO regularization, it can be concluded that customer value does affect the likelihood of a customer churning.

3. **What is the call failure rate at which we expect a customer to churn?**





Because the distribution of the number of call failures is right-skewed, the straightforward confidence interval calculation using a t-distribution cannot be used as it assumes a symmetric distribution about the mean. Alternatively, bootstrapping was used to resample with replacement 10,000 times, the values were ordered, and the 2.5 / 97.5 percentile values were taken to determine the 95% confidence interval for number of call failures for churned customers.

95% CI for Number of Call Failure for Churned Customer: **(9.435, 10.322)**

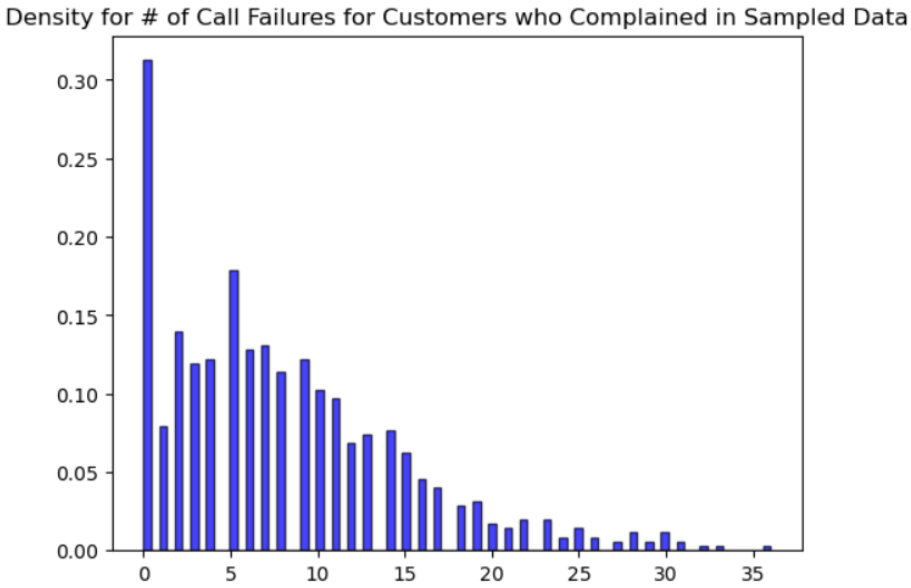
Similarly, the 95% confidence interval for the number of call failures was taken for customers that did not churn.
95% CI for Number of Call Failure for Non-churn Customer: **(7.568, 7.846)**

A 95% CI around the number of call failures for churned customers is approximately 9 to 11 call failures. Because the CI for the number of call failures for churned vs non-churned customers is different, this means a customer is more likely to churn if the number of call failures they experience is greater than 9.

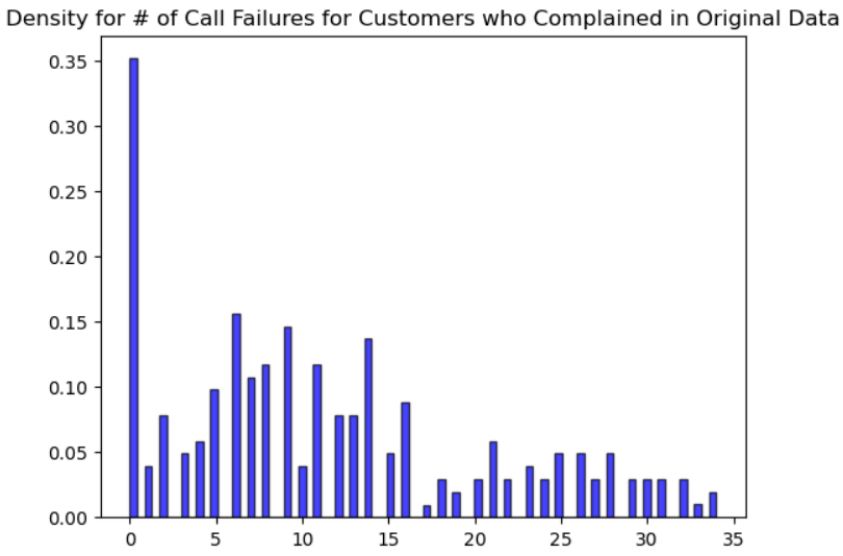
Looking at the histogram, we see a similar pattern between the sampled and original, however, looking at the density of 0 Call Failures in the original data is above 0.6 whereas in the sampled data it's slightly more than 0.25. Moreover, the 2nd peak is much lower in the original data compared to the sampled. The difference is most likely due to the independent sampling whereas in the original data, there is probably some correlation (if not linear then something non-linear) between the Number of Call Failures and Churn or some other feature. A potential solution to this problem is fitting a multivariate distribution and sampling from that instead.

4. What is the number of call failures before we can expect a complaint?

782



241



Similar to the previous result, the number of call failures is right-skewed, and so a similar bootstrapping method was used to determine the 95% confidence interval for customers who did and did not complain.

95% CI for Number of Call Failure for Customers who Complained: **(7.587, 8.543)**
95% CI for Number of Call Failure for Customers who Did Not Complain: **(7.836, 8.117)**

A 95% CI around the number of call failures for customers who complained is approx. 7 to 9 call failures. Because the CI for the number of call failures for complained vs no-complaint customers overlaps, this means a customer who complains is likely complaining NOT because of the number of call failures.

Looking at the histogram, we see a similar pattern between the sampled and original, where the 0 and 2nd peak densities match reasonably well. The discrepancy occurs to higher call failure numbers (greater than 20) where the density in the original data is higher than that observed in the sampled data. The difference might be due to the density fit where it drops to 0 much more quickly than in the original data. The error could again be because of the independent sampling and sampling from a multivariate distribution may improve this result as well.

5. **What is the expected reduction in churn if a proposed business solution can reduce call failures by 50%?**

Prediction of Churn on the Original Sampled Data

Class	Predicted Count
No-churn - 0	8,734
Churn - 1	1,266

Prediction of Churn on the 50% Reduced Call Failures Data

Class	Predicted Count
No-churn - 0	8,815
Churn - 1	1,185

A 50% reduction in number of call failures leads to a 6% reduction in churn.

The number of call failures in the sampled data was all halved, to represent a 50% reduction in call failures. The BTC (subset) model was again used to predict on this altered sampled data, and the number of churned customers was 1185, compared to 1266 in the original prediction of the sampled data, resulting in a 6% decrease.

Perhaps a larger decrease in churn could be attained if a more significant feature was reduced. From the Lasso graph, Inactivity was a very important feature. In fact, looking at the results below - p_inactive (probability of inactivity) is halved from 25% to 12.5% instead of number of call failures, and approx. a 23% reduction in churn is observed, dropping from 1266 to 972 predicted customers churned.

Prediction of Chun on 50% Reduced Inactivity Rate Data

Class	Predicted Count
No-churn - 0	9,028
Churn - 1	972

Conclusion

The study of churn data from an Iranian telecommunications company has provided insightful findings into the dynamics of customer retention. By employing LASSO regression, it was determined that complaints and inactivity are more critical than call failures in predicting churn, challenging our initial hypothesis. This analysis has also shown that customer value has a significant effect on churn likelihood, suggesting that different customer segments may require distinct retention strategies.

Further investigations revealed that customers are more likely to churn if they experience more than 9 call failures, but the direct relationship between call failures and complaints was not as clear as anticipated. Additionally, predictive modeling indicated that a 50% reduction in call failures would lead to a modest 6% decrease in churn, whereas addressing inactivity could potentially result in a more substantial reduction.

These findings emphasize the complexity of customer churn and the need for a nuanced approach to customer service and retention strategies. The project highlights the power of data analytics in uncovering the multifaceted nature of customer interactions and their impact on churn, providing valuable insights for the company to enhance customer satisfaction and loyalty.

Appendix

```
import pandas as pd
import altair as alt
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from scipy.stats import norm, binom, poisson, gaussian_kde, gamma, weibull_min,
lognorm
from sklearn.neighbors import KernelDensity
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from scipy.stats import norm
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import lightgbm as lgb
from sklearn.metrics import accuracy_score, f1_score, log_loss, roc_auc_score
from sklearn.metrics import confusion_matrix
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import statsmodels.gam.api as smg
from sklearn.decomposition import PCA
from scipy.stats import expon
from scipy import stats
import math
from IPython.display import display, HTML
from bootstrapped import bootstrap as bs
from bootstrapped import stats_functions as bs_stats
np.random.seed(123)
```

```
data = pd.read_csv('Customer Churn.csv')

# changing status column from 1: Active, 2: In-Active to 0 and 1
data['Status'] = data['Status'] - 1

# changing Tariff Plan from 1: Pay as You Go, 2: Contractual to 0 and 1
data['Tariff Plan'] = data['Tariff Plan'] - 1

# changing Status column name to Inactivity to represent 0: False (active) and 1:
True (inactive)
# changing Tariff Plan column name to Pay On Contract to represent 0: False (pay-
as-you-go), 1: True (Contractual)
# Renaming other columns to make them consistent - removing 2 space between
words, upper casing subsequent words
data.rename(columns={'Tariff Plan':'Pay On Contract','Status': 'Inactivity',
'Call  Failure':'Call Failure',
                    'Subscription  Length':'Subscription Length',
                    'Charge  Amount':'Charge Amount', 'Frequency of
use':'Frequency of Use'}, inplace=True)

# Caluclating the call failure rate and adding it as a column
data['Call Failure Rate'] = data['Call Failure'] / data['Frequency of Use']
data
```

```
print(data.info())

data['Call Failure Rate'].isnull().sum()
data[data.isnull().any(axis=1)]
data[(data['Call Failure'] == 0) & (data['Frequency of Use']==0)] # look at rows
for Call Failure is NaN
```

```
data[data['Call Failure Rate']==0] # Compare it to rows where Call Failure rate
is actually 0
# Don't want to remove the rows where Call Failure is NaN because there could be
some analysis done on this set it to the mean of the column

mean_cfr = data['Call Failure Rate'].mean()
data['Call Failure Rate'] = data['Call Failure Rate'].fillna(value=mean_cfr)
data.loc[data['Call Failure']>data['Frequency of Use'],'Call Failure'] =
data['Frequency of Use'] # can't have more failures then number of calls
data.info()
```

```
data = data[['Call Failure', 'Complains', 'Subscription Length', 'Charge Amount',
       'Seconds of Use', 'Frequency of Use', 'Frequency of SMS',
       'Distinct Called Numbers', 'Age Group', 'Pay On Contract', 'Inactivity',
       'Age', 'Customer Value', 'Call Failure Rate', 'Churn']]
```

```
graph = []
for i in data.columns:
    chart = alt.Chart(data).mark_boxplot().encode(
        x=alt.X(f'{i}'),
    )
    graph.append(chart)

alt.vconcat(*graph)
```

```
chart = alt.Chart(data).mark_bar().encode(
    x=alt.X('Churn'),
    y=alt.Y('count()'))

chart
```

```
plt.hist(data['Call Failure'], bins=80, density=True, alpha=0.7, color='blue',
edgecolor='black')

kde_call_failure = KernelDensity(bandwidth=0.4)
kde_call_failure.fit(data['Call Failure'].values.reshape(-1,1))
x = np.linspace(data['Call Failure'].min(), data['Call Failure'].max(),
1000).reshape(-1,1)


lambda_estimate = data['Call Failure'].mean()
max_call_failures = data['Call Failure'].max()
x_p = np.arange(0, max_call_failures+1)
poisson_dist = poisson.pmf(x_p, lambda_estimate)
```

```
plt.plot(x_p, poisson_dist, 'r', linewidth=2, label='Poisson w/ mean lambda')
plt.plot(x, np.exp(kde_call_failure.score_samples(x)), label='KDE',
color='black', linestyle='--')
plt.xlabel('Number of Call Failures')
plt.legend()
plt.show()
```

```
plt.hist(data['Subscription Length'], bins=40, density=True, alpha=0.7,
color='blue', edgecolor='black')

a, loc, scale = gamma.fit(data['Subscription Length'])

x_f = np.linspace(data['Subscription Length'].min(), data['Subscription
Length'].max(), 1000)
fitted_pdf = gamma.pdf(x, a, loc, scale)

kde_sub_length = KernelDensity(bandwidth=1.5)
kde_sub_length.fit(data['Subscription Length'].values.reshape(-1,1))
x = np.linspace(data['Subscription Length'].min(), data['Subscription
Length'].max(), 1000).reshape(-1,1)

plt.plot(x_f, fitted_pdf, 'r', label='Gamma fit')
plt.plot(x, np.exp(kde_sub_length.score_samples(x)), label='KDE', color='black',
linestyle='--')
plt.xlabel('Subscription Length')
plt.legend()
plt.show()
```

```
# Time of Use
plt.hist(data['Seconds of Use'], bins=80, density=True, alpha=0.7, color='blue',
edgecolor='black')

kde_sec_of_use = KernelDensity(bandwidth=350)
kde_sec_of_use.fit(data['Seconds of Use'].values.reshape(-1,1))
x = np.linspace(data['Seconds of Use'].min(), data['Seconds of Use'].max(),
1000).reshape(-1,1)


plt.plot(x, np.exp(kde_sec_of_use.score_samples(x)), label='KDE', color='black',
linestyle='--')
plt.xlabel('Seconds of Use')
plt.legend()
plt.show()
```

```
# Number of Calls - Poisson
```

```
plt.hist(data['Frequency of Use'], bins=100, density=True, alpha=0.7,
color='blue', edgecolor='black')

kde_freq_of_use = KernelDensity(bandwidth=1.5)
kde_freq_of_use.fit(data['Frequency of Use'].values.reshape(-1,1))
x = np.linspace(data['Frequency of Use'].min(), data['Frequency of Use'].max(),
1000).reshape(-1,1)

lambda_estimate = data['Frequency of Use'].mean()
max = data['Frequency of Use'].max()
x_p = np.arange(0, max+1)
poisson_dist = poisson.pmf(x_p, lambda_estimate)

plt.plot(x_p, poisson_dist, 'r', linewidth=2, label='Poisson w/ mean lambda')
plt.plot(x, np.exp(kde_freq_of_use.score_samples(x)), label='KDE', color='black',
linestyle='--')
plt.xlabel('Frequency of Use')
plt.legend()
plt.show()
```

```
# Number of SMS - Poisson
plt.hist(data['Frequency of SMS'], bins=100, alpha=0.7, density=True,
color='blue', edgecolor='black')

kde_freq_of_sms = KernelDensity(bandwidth=3)
kde_freq_of_sms.fit(data['Frequency of SMS'].values.reshape(-1,1))
x = np.linspace(data['Frequency of SMS'].min(), data['Frequency of SMS'].max(),
1000).reshape(-1,1)


plt.plot(x, np.exp(kde_freq_of_sms.score_samples(x)), label='KDE', color='black',
linestyle='--')
plt.xlabel('Frequency of SMS')
plt.legend()
plt.show()
```

```
plt.hist(data['Distinct Called Numbers'], bins=100, alpha=0.7, density=True,
color='blue', edgecolor='black')

kde_distinct_num_called = KernelDensity(bandwidth=0.7)
kde_distinct_num_called.fit(data['Distinct Called Numbers'].values.reshape(-1,1))
x = np.linspace(data['Distinct Called Numbers'].min(), data['Distinct Called
Numbers'].max(), 1000).reshape(-1,1)

lambda_estimate = data['Distinct Called Numbers'].mean()
```

```
max = data['Distinct Called Numbers'].max()
x_p = np.arange(0, max+1)
poisson_dist = poisson.pmf(x_p, lambda_estimate)

plt.plot(x_p, poisson_dist, 'r', linewidth=2, label='Poisson w/ mean lambda')

plt.plot(x, np.exp(kde_distinct_num_called.score_samples(x)), label='KDE',
color='black', linestyle='--')
plt.xlabel('Distinct Called Numbers')
plt.legend()
plt.show()
```

```
plt.hist(data['Customer Value'], bins=100, alpha=0.7, density=True, color='blue',
edgecolor='black')

kde_cust_value = KernelDensity(bandwidth=10)
kde_cust_value.fit(data['Customer Value'].values.reshape(-1,1))
x = np.linspace(data['Customer Value'].min(), data['Customer Value'].max(),
1000).reshape(-1,1)

plt.plot(x, np.exp(kde_cust_value.score_samples(x)), label='KDE', color='black',
linestyle='--')
plt.xlabel('Customer Value')
plt.legend()
plt.show()
```

```
plt.hist(data['Call Failure Rate'], bins=100, density=True, alpha=0.7,
         color='blue', edgecolor='black')

kde_call_fail_rate = KernelDensity(bandwidth=0.016)
kde_call_fail_rate.fit(data['Call Failure Rate'].values.reshape(-1,1))
x = np.linspace(data['Call Failure Rate'].min(), data['Call Failure Rate'].max(),
1000).reshape(-1,1)

plt.plot(x, np.exp(kde_call_fail_rate.score_samples(x)), label='KDE',
color='black', linestyle='--')
plt.legend()
plt.xlabel('Call Failure Rate')
plt.show()
```

```
norm_fit = norm.fit(data['Age'])
x = np.linspace(0, 70, 100)
norm_pdf = norm.pdf(x, *norm_fit)
```

```
plt.hist(data['Age'], bins=[0,10,20,30,40,50,60,70], density=True, alpha=0.7,
color='blue', edgecolor='black')
# Adding labels and title
plt.xlabel('Age')
plt.plot(x, norm_pdf, 'r', linewidth=2, label='Norm fit')

# Display the plot
plt.legend()
plt.show()
```

```
print(data[data['Age Group']==1]['Age'].value_counts())

print(data[data['Age Group']==2]['Age'].value_counts())

print(data[data['Age Group']==3]['Age'].value_counts())

print(data[data['Age Group']==4]['Age'].value_counts())

print(data[data['Age Group']==5]['Age'].value_counts())
```

```
plt.hist(data['Age Group'], bins=[1, 2, 3, 4, 5], density=True, alpha=0.7,
color='blue', edgecolor='black')

norm_age_group = norm.fit(data['Age Group'])
x = np.linspace(0, 6, 100)
norm_pdf = norm.pdf(x, *norm_age_group)

plt.plot(x, norm_pdf, 'r', linewidth=2, label='Norm fit')

plt.xlabel('Age Group')
# Display the plot
plt.legend()
plt.show()
```

```
plt.hist(data['Charge Amount'], bins=20, alpha=0.7, density=True, color='blue',
edgecolor='black')

loc_charge_amount, scale_charge_amount = expon.fit(data['Charge Amount'])
x = np.linspace(0, np.max(data['Charge Amount']), 1000)
y = expon.pdf(x, loc=loc_charge_amount, scale=scale_charge_amount)
plt.plot(x, y, 'r-', linewidth=2, label='Exponential Fit')

plt.legend()
plt.xlabel('Charge Amount')
plt.show()
```

```
p_no_complain = data['Complains'].value_counts().values[1]/data.shape[0]
print(p_no_complain)

p_no_pay_on_contract = data['Pay On
Contract'].value_counts().values[1]/data.shape[0]
print(p_no_pay_on_contract)

p_inactive = data['Inactivity'].value_counts().values[1]/data.shape[0]
print(p_inactive)
```

```
x_train, x_test, y_train, y_test = train_test_split(data.drop(columns='Churn'),
data['Churn'], test_size=0.2, random_state=42)

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

```
alphas = np.logspace(-4, 2, 100)
coefs = []

for alpha in alphas:
    lasso = Lasso(alpha=alpha, max_iter=10000)
    lasso.fit(x_train_scaled, y_train)
    coefs.append(lasso.coef_)

plt.figure(figsize=(10, 6))
plt.plot(alphas, coefs)
plt.xscale('log')
plt.xlabel('Regularization strength (alpha)')
plt.ylabel('Coefficients')
plt.title('Lasso Coefficient Paths')
plt.legend(labels=x_train.drop(columns=[]).columns)
plt.show()
a = pd.DataFrame(coefs[35:50], columns=x_train.drop(columns=[]).columns)
a
```

```
endog = y_train
exog = x_train
exog = sm.add_constant(exog)
vif_score = {}
for i in range(0,len(exog.columns.tolist())):
    vif_score[exog.columns.tolist()[i]] = variance_inflation_factor(exog, i)
vif_score # age and age group probably has correlation so pick age group to
ignore potential decision boundries in-between ages
```

```
endog = y_train
exog = x_train.drop(columns=['Age'])
exog = sm.add_constant(exog)
vif_score = {}
for i in range(0,len(exog.columns.tolist())):
    vif_score[exog.columns.tolist()[i]] = variance_inflation_factor(exog, i)
vif_score # frequency of use correlated with seconds of use && frequency of sms
correlated with customer value perhaps
```

```
data['Average Sec per Call'] = data['Seconds of Use']/data['Frequency of Use']
mean_avg_sec_per_call = data['Average Sec per Call'].mean()
data.fillna({'Average Sec per Call':mean_avg_sec_per_call}, inplace=True)

data = data[['Call Failure', 'Complains', 'Subscription Length', 'Charge Amount',
       'Seconds of Use', 'Frequency of Use', 'Frequency of SMS',
       'Distinct Called Numbers', 'Age Group', 'Pay On Contract', 'Inactivity',
       'Age', 'Customer Value', 'Call Failure Rate', 'Average Sec per Call',
       'Churn']] # re-ordering to have response column at the end

kde_average_sec_per_call = KernelDensity(bandwidth=12)
kde_average_sec_per_call.fit(data['Average Sec per Call'].values.reshape(-1,1))

x_train, x_test, y_train, y_test = train_test_split(data.drop(columns=['Churn',
'Age', 'Seconds of Use']), data['Churn'],
                                                    test_size=0.2,
random_state=42)

endog = y_train
exog = x_train
exog = sm.add_constant(exog)
vif_score = {}
for i in range(0,len(exog.columns.tolist())):
    vif_score[exog.columns.tolist()[i]] = variance_inflation_factor(exog, i)
vif_score
```

```
x_train, x_test, y_train, y_test = train_test_split(data.drop(columns=['Churn',
'Age', 'Seconds of Use', 'Frequency of SMS']),
                                                    data['Churn'], test_size=0.2,
random_state=42)

endog = y_train
exog = x_train
exog = sm.add_constant(exog)
vif_score = {}
```

```
for i in range(0,len(exog.columns.tolist())):
    vif_score[exog.columns.tolist()[i]] = variance_inflation_factor(exog, i)
vif_score
```

```
plt.hist(data['Average Sec per Call'], bins=20, alpha=0.7, density=True,
color='blue', edgecolor='black')

x = np.linspace(data['Average Sec per Call'].min(), data['Average Sec per
Call'].max(), 1000).reshape(-1,1)

plt.plot(x, np.exp(kde_average_sec_per_call.score_samples(x)), label='KDE',
color='black', linestyle='--')
plt.xlabel('Average Sec per Call')
plt.legend()
plt.show()
```

```
N = len(data.columns)-1
z_score = {}
for i in range(N):
    mu_0, sigma_0 = norm.fit(data[data['Churn'] == 0].iloc[:,[i,15]])
    mu_1, sigma_1 = norm.fit(data[data['Churn'] == 1].iloc[:,[i,15]])
    z = (mu_0-mu_1) / np.sqrt(sigma_0**2 + sigma_1**2)
    z_score[data.columns[i]] = z
z_score
```

```
x_train_ori, x_test_ori, y_train_ori, y_test_ori =
train_test_split(data.drop(columns=['Churn', 'Age']),
                                                    data['Churn'],
test_size=0.25, random_state=42)

print(y_train_ori.value_counts())
print(y_test_ori.value_counts())

scaler_ori = StandardScaler()
x_train_ori_scaled = scaler_ori.fit_transform(x_train_ori)
x_test_ori_scaled = scaler_ori.transform(x_test_ori)
```

```
logreg = LogisticRegression(max_iter=1000, class_weight='balanced',
random_state=42)
svc = SVC(probability=True, class_weight='balanced', random_state=42)
rf = RandomForestClassifier(class_weight='balanced', random_state=42)
btc = lgb.LGBMClassifier(verbosity=-1, class_weight='balanced', random_state=42)
models = {'log_ori': logreg, 'svc_ori': svc, 'rf_ori': rf, 'btc_ori': btc}

scoring=['f1', 'roc_auc', 'balanced_accuracy', 'matthews_corrcoef']
```

```
cv_scores_mean_ori = {}
cv_scores_stdv_ori = {}
for name, model in models.items():
    # Compute cross-validation scores
    scores = cross_validate(model, x_train_ori_scaled, y_train_ori, cv=10,
scoring=scoring)
    cv_scores_mean_ori[name] = {metric: scores[f'test_{metric}'].mean() for
metric in scoring}
    cv_scores_stdv_ori[name] = {metric: scores[f'test_{metric}'].std() for metric
in scoring}

cv_scores_df_mean_ori = pd.DataFrame(cv_scores_mean_ori).T
print(cv_scores_df_mean_ori)
cv_scores_df_stdv_ori = pd.DataFrame(cv_scores_stdv_ori).T
print(cv_scores_df_stdv_ori)
```

```
logreg_ori = LogisticRegression(max_iter=1000, class_weight='balanced',
random_state=42)
logreg_ori.fit(x_train_ori_scaled, y_train_ori)

svc_ori = SVC(probability=True, class_weight='balanced', random_state=42)
svc_ori.fit(x_train_ori_scaled, y_train_ori)

rf_ori = RandomForestClassifier(class_weight='balanced', random_state=42)
rf_ori.fit(x_train_ori_scaled, y_train_ori)

btc_ori = lgb.LGBMClassifier(verbosity=-1, class_weight='balanced',
random_state=42)
btc_ori.fit(x_train_ori_scaled, y_train_ori)

models_ori = {'log_ori': logreg_ori, 'svc_ori': svc_ori, 'rf_ori': rf_ori,
'btc_ori': btc_ori}
model_scores_ori = {}
for model in models_ori:
    y_pred_ori = models_ori[model].predict(x_test_ori_scaled)
    y_pred_prob_ori = models_ori[model].predict_proba(x_test_ori_scaled)[:, 1]
    f1 = f1_score(y_test_ori, y_pred_ori)
    ll = log_loss(y_test_ori, y_pred_prob_ori)
    roc_auc = roc_auc_score(y_test_ori, y_pred_prob_ori)
    cm = confusion_matrix(y_test_ori, y_pred_ori)
    model_scores_ori[model] = [f1, ll, roc_auc, cm]
model_scores_ori
```

```
x_train_sub, x_test_sub, y_train_sub, y_test_sub =
train_test_split(data.drop(columns=['Churn', 'Age', 'Seconds of Use', 'Frequency
of SMS']),
                                                    data['Churn'],
test_size=0.25, random_state=42)

print(y_train_sub.value_counts())
print(y_test_sub.value_counts())

scaler_sub = StandardScaler()
x_train_sub_scaled = scaler_sub.fit_transform(x_train_sub)
x_test_sub_scaled = scaler_sub.transform(x_test_sub)
```

```
logreg = LogisticRegression(max_iter=1000, class_weight='balanced',
random_state=42)
svc = SVC(probability=True, class_weight='balanced', random_state=42)
rf = RandomForestClassifier(class_weight='balanced', random_state=42)
btc = lgb.LGBMClassifier(verbosity=-1, class_weight='balanced', random_state=42)
models = {'log_sub': logreg, 'svc_sub': svc, 'rf_sub': rf, 'btc_sub': btc}
scoring=['f1', 'roc_auc', 'balanced_accuracy', 'matthews_corrcoef']

cv_scores_mean_sub = {}
cv_scores_stdv_sub = {}
for name, model in models.items():
    # Compute cross-validation scores
    scores = cross_validate(model, x_train_sub_scaled, y_train_sub, cv=10,
scoring=scoring)
    cv_scores_mean_sub[name] = {metric: scores[f'test_{metric}'].mean() for
metric in scoring}
    cv_scores_stdv_sub[name] = {metric: scores[f'test_{metric}'].std() for metric
in scoring}

cv_scores_df_mean_sub = pd.DataFrame(cv_scores_mean_sub).T
print(cv_scores_df_mean_sub)
cv_scores_df_stdv_sub = pd.DataFrame(cv_scores_stdv_sub).T
print(cv_scores_df_stdv_sub)
```

```
logreg_sub = LogisticRegression(max_iter=1000, class_weight='balanced',
random_state=42)
logreg_sub.fit(x_train_sub_scaled, y_train_sub)

svc_sub = SVC(probability=True, class_weight='balanced', random_state=42)
svc_sub.fit(x_train_sub_scaled, y_train_sub)

rf_sub = RandomForestClassifier(class_weight='balanced', random_state=42)
```

```
rf_sub.fit(x_train_sub_scaled, y_train_sub)

btc_sub = lgb.LGBMClassifier(verbosity=-1, class_weight='balanced',
random_state=42)
btc_sub.fit(x_train_sub_scaled, y_train_sub)

models_sub = {'log_sub': logreg_sub, 'svc_sub': svc_sub, 'rf_sub': rf_sub,
'btc_sub': btc_sub}
model_scores_sub = {}
for name, model in models_sub.items():
    y_pred_sub = model.predict(x_test_sub_scaled)
    y_pred_prob_sub = model.predict_proba(x_test_sub_scaled)[:, 1]
    f1 = f1_score(y_test_sub, y_pred_sub)
    ll = log_loss(y_test_sub, y_pred_prob_sub)
    roc_auc = roc_auc_score(y_test_sub, y_pred_prob_sub)
    cm = confusion_matrix(y_test_sub, y_pred_sub)
    model_scores_sub[name] = [f1, ll, roc_auc, cm]
model_scores_sub
```

```
print(y_train_ori.value_counts())
print(y_test_ori.value_counts())

pca = PCA(random_state=42)
x_train_ori_pca = pca.fit_transform(x_train_ori_scaled)
x_test_ori_pca = pca.transform(x_test_ori_scaled)
```

```
exog = pd.DataFrame(x_test_ori_pca, columns=x_train_ori.columns)
exog = sm.add_constant(exog)
vif_score = {}
for i in range(0,len(exog.columns.tolist())):
    vif_score[exog.columns.tolist()[i]] = variance_inflation_factor(exog, i)
vif_score
```

```
logreg = LogisticRegression(max_iter=1000, class_weight='balanced',
random_state=42)
svc = SVC(probability=True, class_weight='balanced', random_state=42)
rf = RandomForestClassifier(class_weight='balanced', random_state=42)
btc = lgb.LGBMClassifier(verbosity=-1, class_weight='balanced', random_state=42)
models = {'log_pca_ori': logreg, 'svc_pca_ori': svc, 'rf_pca_ori': rf,
'btc_pca_ori': btc}

scoring=['f1', 'roc_auc', 'balanced_accuracy', 'matthews_corrcoef']
cv_scores_mean_pca_ori = {}
cv_scores_stdv_pca_ori = {}
for name, model in models.items():
```

```
    # Compute cross-validation scores
    scores = cross_validate(model, x_train_ori_pca, y_train_ori, cv=10,
scoring=scoring)
    cv_scores_mean_pca_ori[name] = {metric: scores[f'test_{metric}'].mean() for
metric in scoring}
    cv_scores_stdv_pca_ori[name] = {metric: scores[f'test_{metric}'].std() for
metric in scoring}

cv_scores_df_mean_pca_ori = pd.DataFrame(cv_scores_mean_pca_ori).T
print(cv_scores_df_mean_pca_ori)
cv_scores_df_stdv_pca_ori = pd.DataFrame(cv_scores_stdv_pca_ori).T
print(cv_scores_df_stdv_pca_ori)
```

```
logreg_pca = LogisticRegression(max_iter=1000, class_weight='balanced',
random_state=42)
logreg_pca.fit(x_train_ori_pca, y_train_ori)

svc_pca = SVC(probability=True, class_weight='balanced', random_state=42)
svc_pca.fit(x_train_ori_pca, y_train_ori)

rf_pca = RandomForestClassifier(class_weight='balanced', random_state=42)
rf_pca.fit(x_train_ori_pca, y_train_ori)

btc_pca = lgb.LGBMClassifier(verbosity=-1, class_weight='balanced',
random_state=42)
btc_pca.fit(x_train_ori_pca, y_train_ori)

models_pca = {'log_pca': logreg_pca, 'svc_pca': svc_pca, 'rf_pca': rf_pca,
'btc_pca': btc_pca}
model_scores_pca = {}
for name, model in models_pca.items():
    y_pred_pca = model.predict(x_test_ori_pca)
    y_pred_prob_pca = model.predict_proba(x_test_ori_pca)[:, 1]
    f1 = f1_score(y_test_ori, y_pred_pca)
    ll = log_loss(y_test_ori, y_pred_prob_pca)
    roc_auc = roc_auc_score(y_test_ori, y_pred_prob_pca)
    cm = confusion_matrix(y_test_ori, y_pred_pca)
    model_scores_pca[name] = [f1, ll, roc_auc, cm]
model_scores_pca
```

```
print(y_train_sub.value_counts())
print(y_test_sub.value_counts())
```

```
kmeans = KMeans(n_clusters=2, random_state=42, n_init=25)
kmeans.fit(x_train_sub_scaled)
```

```
y_pred_kmean = kmeans.predict(x_test_sub_scaled)
```

```
cm_kmean = confusion_matrix(y_test_sub, y_pred_kmean)
cm_kmean
```

```
print(y_train_ori.value_counts())
print(y_test_ori.value_counts())
```

```
gmm = GaussianMixture(n_components=2, random_state=42)
gmm.fit(x_train_ori_scaled)

y_pred_gmm = gmm.predict(x_test_ori_scaled)
cm_gmm = confusion_matrix(y_test_ori, y_pred_gmm)
cm_gmm
```

```
def display_side_by_side(*args):
    html_str = ''
    for df in args:
        html_str += df.to_html()
    display(HTML(html_str.replace('table','table style="display:inline;margin-
right:20px;"')))

print('Cross-Validation Results')
display_side_by_side(cv_scores_df_mean_ori, cv_scores_df_stdv_ori)
display_side_by_side(cv_scores_df_mean_sub, cv_scores_df_stdv_sub)
display_side_by_side(cv_scores_df_mean_pca_ori, cv_scores_df_stdv_pca_ori)

print('')
print('Test Set Results')

df_ori = pd.DataFrame(model_scores_ori).transpose()
df_ori.columns = ['Accuracy', 'Error Rate', 'AUC', 'Confusion Matrix']
df_sub = pd.DataFrame(model_scores_sub).transpose()
df_sub.columns = ['Accuracy', 'Error Rate', 'AUC', 'Confusion Matrix']
df_pca = pd.DataFrame(model_scores_pca).transpose()
df_pca.columns = ['Accuracy', 'Error Rate', 'AUC', 'Confusion Matrix']

# Convert confusion matrix arrays to string for better display
df_ori['Confusion Matrix'] = df_ori['Confusion Matrix'].apply(lambda x: str(x))
df_sub['Confusion Matrix'] = df_sub['Confusion Matrix'].apply(lambda x: str(x))
df_pca['Confusion Matrix'] = df_pca['Confusion Matrix'].apply(lambda x: str(x))

display_side_by_side(df_ori, df_sub, df_pca)

display(cm_kmean)
```

```
display(cm_gmm)
```

```
kde = {
    'Call Failure': kde_call_failure,
    'Complains': p_no_complain,
    'Subscription Length': kde_sub_length ,
    'Charge Amount': [loc_charge_amount, scale_charge_amount],
    'Seconds of Use': kde_sec_of_use,
    'Frequency of Use': kde_freq_of_use,
    'Frequency of SMS': kde_freq_of_sms,
    'Distinct Called Numbers': kde_distinct_num_called,
    'Pay On Contract': p_no_pay_on_contract,
    'Inactivity': p_inactive,
    'Age Group': norm_age_group,
    'Customer Value': kde_cust_value
}

def sample_kde_with_threshold(density, lower_bound, num_samples):
    samples = []
    while len(samples) < num_samples:
        sample = density.sample(1)[0][0]
        if sample >= lower_bound:
            samples.append(sample)
    return np.array(samples)

lower_bound = 0
num_samples = 10000

samples = pd.DataFrame()
for feature, density in kde.items():
    if isinstance(density, KernelDensity):
        feature_samples = sample_kde_with_threshold(density, lower_bound,
num_samples)
    elif feature == 'Age Group':
        feature_samples = np.random.normal(density[0], density[1], num_samples)
    elif feature in ['Complains', 'Pay On Contract', 'Inactivity']:
        feature_samples = np.random.binomial(1, p=density, size=num_samples)
    elif feature == 'Charge Amount':
        feature_samples = np.random.exponential(scale=density[1],
size=num_samples) + density[0]
    else:
        feature_samples = np.full(num_samples, density)
    samples[feature] = feature_samples
samples.head()
samples.describe()
```

```
samples.loc[samples['Age Group'] <= 1.9, 'Age Group'] = 1
samples.loc[(samples['Age Group'] > 1.9) & (samples['Age Group'] <= 2.9), 'Age
Group'] = 2
samples.loc[(samples['Age Group'] > 2.9) & (samples['Age Group'] <= 3.9), 'Age
Group'] = 3
samples.loc[(samples['Age Group'] > 3.9) & (samples['Age Group'] <= 4.9), 'Age
Group'] = 4
samples.loc[samples['Age Group'] > 4.9, 'Age Group'] = 5
samples['Age Group'] = samples['Age Group'].astype(int)

samples.loc[(samples['Charge Amount'] < 0.8), 'Charge Amount'] = 0
samples['Charge Amount'] = samples['Charge Amount'].round().astype(int)

samples.loc[(samples['Call Failure'] < 0.95), 'Call Failure'] = 0
samples['Call Failure'] = samples['Call Failure'].round().astype(int)

samples['Subscription Length'] = samples['Subscription
Length'].round().astype(int)
samples['Seconds of Use'] = samples['Seconds of Use'].round().astype(int)
samples['Frequency of Use'] = samples['Frequency of Use'].round().astype(int)
samples['Frequency of SMS'] = samples['Frequency of SMS'].round().astype(int)
samples['Distinct Called Numbers'] = samples['Distinct Called
Numbers'].round().astype(int)

samples.loc[samples['Call Failure'] > samples['Frequency of Use'], 'Call
Failure'] = samples['Frequency of Use'] # can't have more failures than calls
samples.loc[samples['Frequency of Use']==0, 'Seconds of Use'] = 0 # if frequency
of use is 0 then seconds of use also has to be 0

samples['Call Failure Rate'] = samples['Call Failure'] / samples['Frequency of
Use']
mean_cfr = samples['Call Failure Rate'].mean()
samples['Call Failure Rate'] = samples['Call Failure
Rate'].fillna(value=mean_cfr)

samples['Average Sec per Call'] = samples['Seconds of Use']/samples['Frequency of
Use']
mean_avg_sec_per_call = samples['Average Sec per Call'].mean()
samples.fillna({'Average Sec per Call':mean_avg_sec_per_call}, inplace=True)

samples = samples[['Call Failure', 'Complains', 'Subscription Length', 'Charge
Amount',
       'Seconds of Use', 'Frequency of Use', 'Frequency of SMS',
```

```
       'Distinct Called Numbers', 'Age Group', 'Pay On Contract', 'Inactivity',
       'Customer Value', 'Call Failure Rate', 'Average Sec per Call']]
```

```
plt.hist(samples['Call Failure'], bins=40, alpha=0.7, density=True, color='blue',
edgecolor='black')

plt.show()
```

```
x_sample_ori_scaled = scaler_ori.transform(samples)
x_sample_pca = pca.transform(x_sample_ori_scaled)

exog = pd.DataFrame(x_sample_pca, columns=samples.columns)
exog = sm.add_constant(exog)
vif_score = {}
for i in range(0,len(exog.columns.tolist())):
    vif_score[exog.columns.tolist()[i]] = variance_inflation_factor(exog, i)
vif_score
```

```
x_sample_sub_scaled = scaler_sub.transform(samples.drop(columns=['Seconds of
Use', 'Frequency of SMS']))

exog = pd.DataFrame(x_sample_sub_scaled, columns=samples.drop(columns=['Seconds
of Use', 'Frequency of SMS']).columns)
exog = sm.add_constant(exog)
vif_score = {}
for i in range(0,len(exog.columns.tolist())):
    vif_score[exog.columns.tolist()[i]] = variance_inflation_factor(exog, i)
vif_score
```

```
y_pred_sample = btc_sub.predict(x_sample_sub_scaled)
unique_values, counts = np.unique(y_pred_sample, return_counts=True)
print(unique_values)
print(counts)
```

```
new_sample_data = samples.copy()
new_sample_data['Predict Churn'] = y_pred_sample
```

```
sample_call_failure = new_sample_data.loc[new_sample_data['Predict
Churn']==1,:]['Call Failure']
sample_call_failure_no_churn = new_sample_data.loc[new_sample_data['Predict
Churn']==0,:]['Call Failure']
```

```
print(new_sample_data.loc[new_sample_data['Predict Churn']==1,:]['Call
Failure'].shape[0])
```

```
plt.hist(new_sample_data[new_sample_data['Predict Churn']==1]['Call Failure'],
bins=80, density=True, alpha=0.7, color='blue', edgecolor='black')
plt.title('Density for # of Call Failures for Churned Customers in Sampled Data')
plt.show()
```

```
print(data[data['Churn']==1]['Call Failure'].shape[0])
plt.hist(data[data['Churn']==1]['Call Failure'], bins=80, density=True,
alpha=0.7, color='blue', edgecolor='black')
plt.title('Density for # of Call Failures for Churned Customers in Original
Data')
plt.show()
```

```
bootstrap_means = []
for i in range(10000):
    bootstrap_sample = np.random.choice(sample_call_failure.values,
size=len(sample_call_failure.values), replace=True)
    sample_mean = np.mean(bootstrap_sample)
    bootstrap_means.append(sample_mean)

lower_bound = np.percentile(bootstrap_means, 2.5)
upper_bound = np.percentile(bootstrap_means, 97.5)

print(f"95% CI for Number of Call Failure for Churned Customer: ({lower_bound},
{upper_bound})")
```

```
m = sample_call_failure.mean()
std = sample_call_failure.std()
sem = std / np.sqrt(len(sample_call_failure))
print(m)
print(sem)
confidence_level = 0.99
ci = stats.t.interval(confidence_level, len(sample_call_failure)-1, loc=m,
scale=sem)
print(f"Mean: {m}")
print(f"99% CI: {ci}")
```

```
bootstrap_means = []
for i in range(10000):
    bootstrap_sample = np.random.choice(sample_call_failure_no_churn.values,
size=len(sample_call_failure_no_churn.values), replace=True)
    sample_mean = np.mean(bootstrap_sample)
    bootstrap_means.append(sample_mean)

lower_bound_no_churn = np.percentile(bootstrap_means, 2.5)
upper_bound_no_churn = np.percentile(bootstrap_means, 97.5)
```

```
print(f"95% CI for Number of Call Failure for Non-churn Customer:
({lower_bound_no_churn}, {upper_bound_no_churn})")
```

```
sample_call_failure_complain =
new_sample_data.loc[new_sample_data['Complains']==1,:]['Call Failure']
sample_call_failure_complain_no_complain =
new_sample_data.loc[new_sample_data['Complains']==0,:]['Call Failure']
```

```
print(new_sample_data.loc[new_sample_data['Complains']==1,:]['Call
Failure'].shape[0])
plt.hist(new_sample_data[new_sample_data['Complains']==1]['Call Failure'],
bins=80, density=True, alpha=0.7, color='blue', edgecolor='black')
plt.title('Density for # of Call Failures for Customers who Complained in Sampled
Data')
plt.show()
```

```
print(data.loc[data['Complains']==1,:]['Call Failure'].shape[0])
plt.hist(data[data['Complains']==1]['Call Failure'], bins=80, density=True,
alpha=0.7, color='blue', edgecolor='black')
plt.title('Density for # of Call Failures for Customers who Complained in
Original Data')
plt.show()
```

```
bootstrap_means_complain = []
for i in range(10000):
    bootstrap_sample = np.random.choice(sample_call_failure_complain.values,
size=len(sample_call_failure_complain.values), replace=True)
    sample_mean = np.mean(bootstrap_sample)
    bootstrap_means_complain.append(sample_mean)

lower_bound_complain = np.percentile(bootstrap_means_complain, 2.5)
upper_bound_complain = np.percentile(bootstrap_means_complain, 97.5)

print(f"95% CI for Number of Call Failure for Customer who Complained:
({lower_bound_complain}, {upper_bound_complain})")
```

```
m_complain = sample_call_failure_complain.mean()
std_complain = sample_call_failure_complain.std()
sem_complain = std_complain / np.sqrt(len(sample_call_failure_complain))
print(m_complain)
print(sem_complain)
confidence_level = 0.99
ci_complain = stats.t.interval(confidence_level,
len(sample_call_failure_complain)-1, loc=m_complain, scale=sem_complain)
```

```
print(f"Mean: {m_complain}")
print(f"99% CI: {ci_complain}")
```

```
bootstrap_means_complain = []
for i in range(10000):
    bootstrap_sample =
np.random.choice(sample_call_failure_complain_no_complain.values,
                                        size=len(sample_call_failure_complain_no_
complain.values), replace=True)
    sample_mean = np.mean(bootstrap_sample)
    bootstrap_means_complain.append(sample_mean)

lower_bound_complain_no_complain = np.percentile(bootstrap_means_complain, 2.5)
upper_bound_complain_no_complain = np.percentile(bootstrap_means_complain, 97.5)

print(f"95% CI for Number of Call Failure for Customer who Did Not Complain:
({lower_bound_complain_no_complain}, {upper_bound_complain_no_complain})")
```

```
new_samples = samples.copy()
new_samples['Call Failure'] = new_samples['Call Failure'] / 2
new_samples['Call Failure'] = new_samples['Call Failure'].astype(int)
```

```
x_new_sample_sub_scaled = scaler_sub.transform(new_samples.drop(columns=['Seconds
of Use', 'Frequency of SMS']))

y_pred_new_sample = btc_sub.predict(x_new_sample_sub_scaled)
unique_values_reduced, counts_reduced = np.unique(y_pred_new_sample,
return_counts=True)
print(unique_values_reduced)
print(counts_reduced)
```

```
reduction =  counts[1] - counts_reduced[1]
reduction_rate = (reduction/counts[1])*100
print(reduction)
print(reduction_rate)
```

```
np.random.seed(123)

kde = {
    'Call Failure': kde_call_failure,
    'Complains': p_no_complain,
    'Subscription Length': kde_sub_length ,
    'Charge Amount': [loc_charge_amount, scale_charge_amount],
    'Seconds of Use': kde_sec_of_use,
    'Frequency of Use': kde_freq_of_use,
```

```
    'Frequency of SMS': kde_freq_of_sms,
    'Distinct Called Numbers': kde_distinct_num_called,
    'Pay On Contract': p_no_pay_on_contract,
    'Inactivity': 0.125,
    'Age Group': norm_age_group,
    'Customer Value': kde_cust_value
}

def sample_kde_with_threshold(density, lower_bound, num_samples):
    samples = []
    while len(samples) < num_samples:
        sample = density.sample(1)[0][0]
        if sample >= lower_bound:
            samples.append(sample)
    return np.array(samples)

lower_bound = 0
num_samples = 10000

samples = pd.DataFrame()
for feature, density in kde.items():
    if isinstance(density, KernelDensity):
        feature_samples = sample_kde_with_threshold(density, lower_bound,
num_samples)
    elif feature == 'Age Group':
        feature_samples = np.random.normal(density[0], density[1], num_samples)
    elif feature in ['Complains', 'Pay On Contract', 'Inactivity']:
        feature_samples = np.random.binomial(1, p=density, size=num_samples)
    elif feature == 'Charge Amount':
        feature_samples = np.random.exponential(scale=density[1],
size=num_samples) + density[0]
    else:
        feature_samples = np.full(num_samples, density)
    samples[feature] = feature_samples
samples.head()
samples.describe()


samples.loc[samples['Age Group'] <= 1.9, 'Age Group'] = 1
samples.loc[(samples['Age Group'] > 1.9) & (samples['Age Group'] <= 2.9), 'Age
Group'] = 2
samples.loc[(samples['Age Group'] > 2.9) & (samples['Age Group'] <= 3.9), 'Age
Group'] = 3
samples.loc[(samples['Age Group'] > 3.9) & (samples['Age Group'] <= 4.9), 'Age
Group'] = 4
```

```
samples.loc[samples['Age Group'] > 4.9, 'Age Group'] = 5
samples['Age Group'] = samples['Age Group'].astype(int)

samples.loc[(samples['Charge Amount'] < 0.8), 'Charge Amount'] = 0
samples['Charge Amount'] = samples['Charge Amount'].round().astype(int)

samples.loc[(samples['Call Failure'] < 0.95), 'Call Failure'] = 0
samples['Call Failure'] = samples['Call Failure'].round().astype(int)

samples['Subscription Length'] = samples['Subscription
Length'].round().astype(int)
samples['Seconds of Use'] = samples['Seconds of Use'].round().astype(int)
samples['Frequency of Use'] = samples['Frequency of Use'].round().astype(int)
samples['Frequency of SMS'] = samples['Frequency of SMS'].round().astype(int)
samples['Distinct Called Numbers'] = samples['Distinct Called
Numbers'].round().astype(int)

samples.loc[samples['Call Failure'] > samples['Frequency of Use'], 'Call
Failure'] = samples['Frequency of Use'] # can't have more failures than calls
samples.loc[samples['Frequency of Use']==0, 'Seconds of Use'] = 0 # if frequency
of use is 0 then seconds of use also has to be 0

samples['Call Failure Rate'] = samples['Call Failure'] / samples['Frequency of
Use']
mean_cfr = samples['Call Failure Rate'].mean()
samples['Call Failure Rate'] = samples['Call Failure
Rate'].fillna(value=mean_cfr)

samples['Average Sec per Call'] = samples['Seconds of Use']/samples['Frequency of
Use']
mean_avg_sec_per_call = samples['Average Sec per Call'].mean()
samples.fillna({'Average Sec per Call':mean_avg_sec_per_call}, inplace=True)

samples = samples[['Call Failure', 'Complains', 'Subscription Length', 'Charge
Amount',
       'Seconds of Use', 'Frequency of Use', 'Frequency of SMS',
       'Distinct Called Numbers', 'Age Group', 'Pay On Contract', 'Inactivity',
       'Customer Value', 'Call Failure Rate', 'Average Sec per Call']]
x_sample_sub_scaled = scaler_sub.transform(samples.drop(columns=['Seconds of
Use', 'Frequency of SMS']))
y_pred_sample = btc_sub.predict(x_sample_sub_scaled)
unique_values, counts = np.unique(y_pred_sample, return_counts=True)
print(unique_values)
print(counts)
print((1266-counts[1])/1266*100)
```