Customer Churn Prediction

Bhavans Vivekananda College
GROUP - 10

Presented by :

Vaishali Vadde | Syed Danish | Shravya | Pravalika

Abstract

Customer churn prediction involves using data analysis and machine learning techniques to identify customers at risk of leaving a service or product. By analyzing historical data, such models help businesses implement proactive retention strategies. This enhances customer satisfaction and reduces revenue loss

The study explores machine learning algorithms like Logestic Regression, Decision Tree, Random Forest, K-Nearest Neighbours, Support Vector Machines, Bagging, and Boosting to predict customer churn based on the percentage of customers who stop using a company's product or service

Objective

To identify the most suitable machine learning model to customer churn, enabling businesses to implement targeted retention strategies and reduce revenue loss.

Content

Introduction

- Customer churn refers to the loss of customers over time, particularly critical for businesses relying on recurring revenue models. It impacts financial sustainability and customer retention strategies.

- Advances in data analytics and machine learning enable businesses to predict churn, understand customer behaviors, and proactively address risks, improving retention rates and satisfaction.

- Reducing churn is cost-effective compared to acquiring new customers, making it essential for maintaining profitability and enhancing business reputation.

- This project uses classification techniques to analyze historical data, identify churn drivers, and provide actionable insights to strengthen customer loyalty and retention strategies.

Literature Review

Literature Review-1

Paper : "A Comprehensive Review of Machine Learning for Churn Prediction" by Smith, J., & Lee, H.

Year & Journal: Published in 2018 in Journal of Data Science and Applications.

Summary: The authors provided a detailed review of various machine learning models used for customer churn prediction, such as logistic regression, decision trees, and support vector machines. They compared these models based on performance metrics and concluded that ensemble methods, such as random forests, often achieved better accuracy.

Literature Review-2

Paper : "Deep Learning Techniques for Predicting Customer Churn: A Review" by Patel, M., & Zhao, L.

Year & Journal: Published in 2020 in International Journal of Artificial Intelligence and Data Mining.

Summary: This review focused on deep learning models used in churn prediction, emphasizing the ability of neural networks to capture complex data patterns. The authors found that while deep learning models could outperform traditional methods, they required substantial data and computational resources. They also discussed the trade-off between model performance and interpretability.

Data preprocessing

Data

Dataset : Our Dataset consists of 14 variables and 3150 records

Source : https://archive.ics.uci.edu/dataset/320/student+performance

Variables :

Categorical	Continuous
Complains	Call Failures
Charge amount	Subscription Length
Tariff Plan	Frequency Of Use
Status	Frequency Of Sms
Churn	Customer Value

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Call	Failu	Complain	Subscripti	Charge	A	Seconds o	Frequency	Frequency	Distinct C	Age Group	Tariff Plan	Status	Age	Customer	Churn
2	8	0	38	0	4370	71	5	17	3	1	1	30	197.64	0
3	0	0	39	0	318	5	7	4	2	1	2	25	46.035	0
4	10	0	37	0	2453	60	359	24	3	1	1	30	1536.52	0
5	10	0	38	0	4198	66	1	35	1	1	1	15	240.02	0
6	3	0	38	0	2393	58	2	33	1	1	1	15	145.805	0
7	11	0	38	1	3775	82	32	28	3	1	1	30	282.28	0
8	4	0	38	0	2360	39	285	18	3	1	1	30	1235.96	0
9	13	0	37	2	9115	121	144	43	3	1	1	30	945.44	0
10	7	0	38	0	13773	169	0	44	3	1	1	30	557.68	0
11	7	0	38	1	4515	83	2	25	3	1	1	30	191.92	0
12	6	0	38	0	5918	95	7	12	3	1	1	30	268.52	0
13	9	0	38	0	2238	54	8	17	3	1	2	30	123.68	0
14	25	0	38	3	15140	225	54	32	3	1	1	30	830.6	0
15	4	0	38	1	3095	27	483	8	3	1	1	30	2056.88	0
16	9	0	37	0	15485	182	150	30	2	1	1	25	1380.015	0
17	3	0	37	1	6500	86	186	26	3	1	1	30	1007.44	0
18	0	0	37	0	875	14	0	11	2	1	2	25	40.005	1
19	2	0	38	0	710	14	13	8	3	1	2	30	80.96	0
20	0	0	37	0	0	0	0	0	2	1	2	25	0	1
21	3	0	37	0	7508	127	384	43	2	1	1	25	2071.575	0
22	7	0	37	1	11465	154	11	47	4	1	1	45	317.975	0
23	8	0	37	1	6718	75	108	37	2	1	1	25	791.685	0
24	23	1	33	0	955	47	16	17	2	1	2	25	117.09	1
25	21	1	36	8	10435	93	1	42	5	2	1	55	159.42	0
26	13	1	36	1	5818	98	26	24	2	1	1	25	383.22	1

Data Cleaning

Data Cleaning :

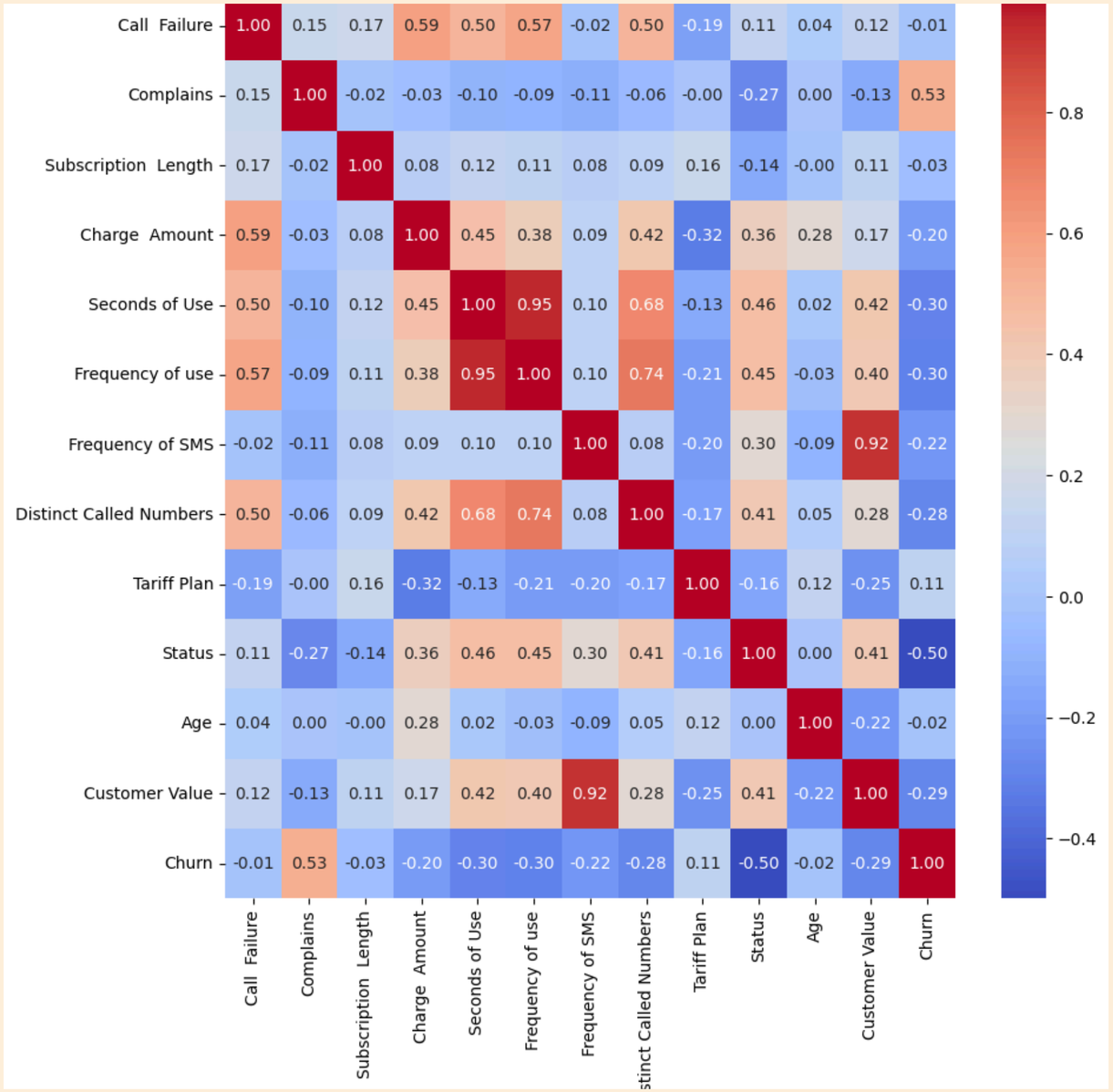
Data cleaning is the process of identifying and correcting or removing inaccurate, incomplete, or irrelevant data from a dataset .The process performed for data cleaning are as follows:

- Identify missing values and decide on strategies . After searching for missing values we got to know that there are no missing values in our data
- Replaced two columns in 0's and 1's

```
Tariff Plan
1    2905
0     245
Name: count, dtype: int64
Status
1    2368
0     782
Name: count, dtype: int64
```

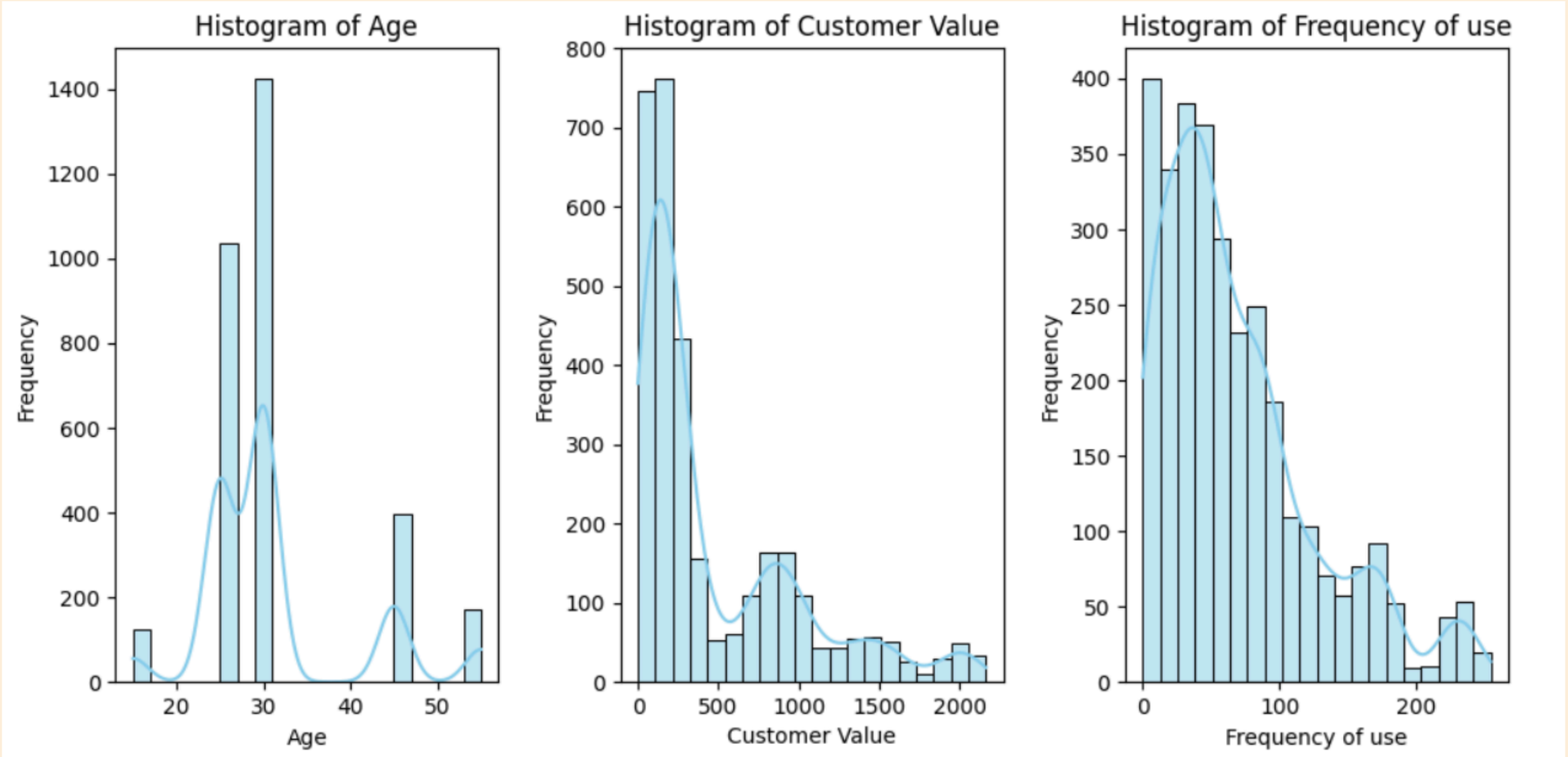
Exploratory Data Analysis

CORELATION HEATMAP



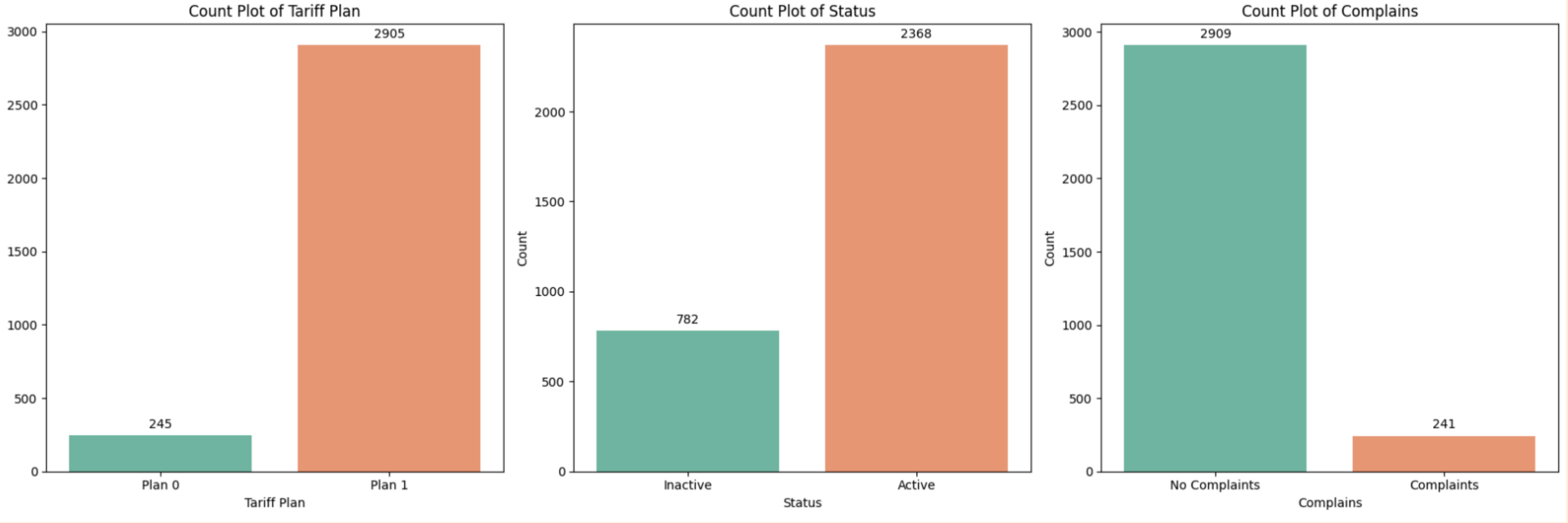
This visualizes the relationships between numerical features. It highlights correlations, helping identify features that might be redundant or strongly associated with churn

HISTOGRAM



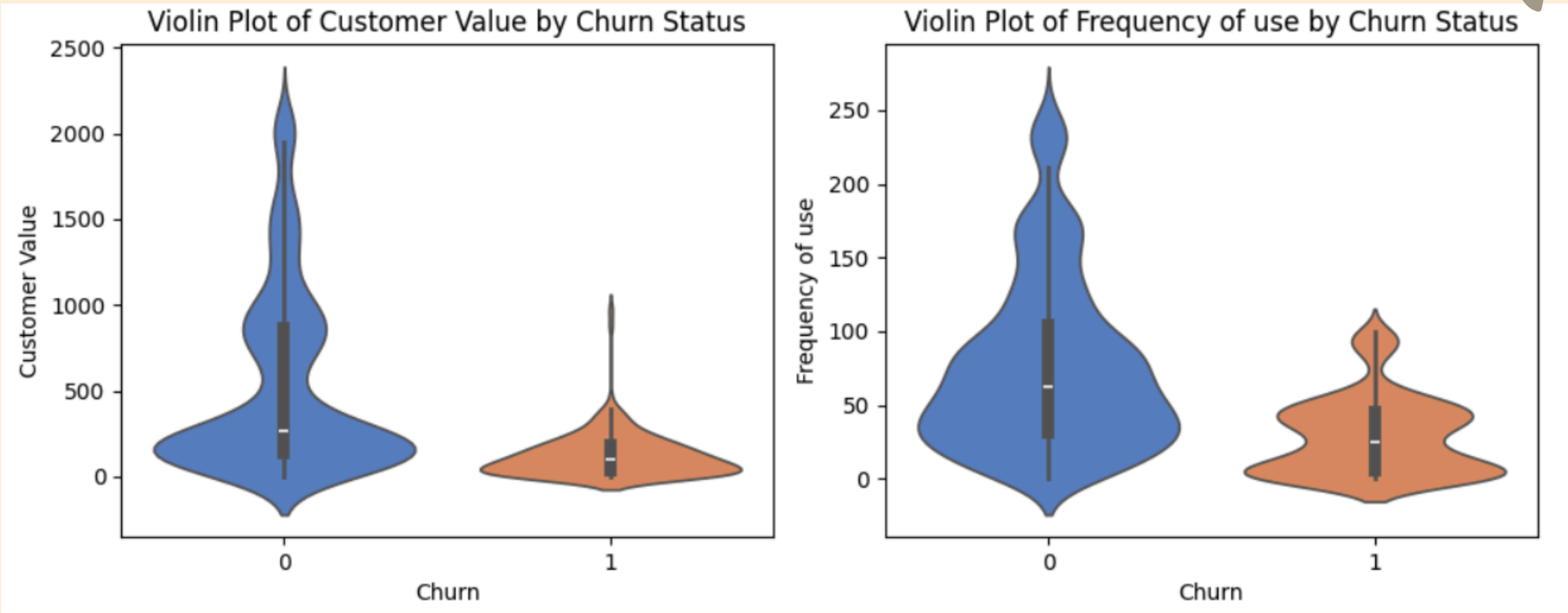
Histograms show the distribution of variables, helping us understand the frequency of different values for each feature.

COUNT PLOT



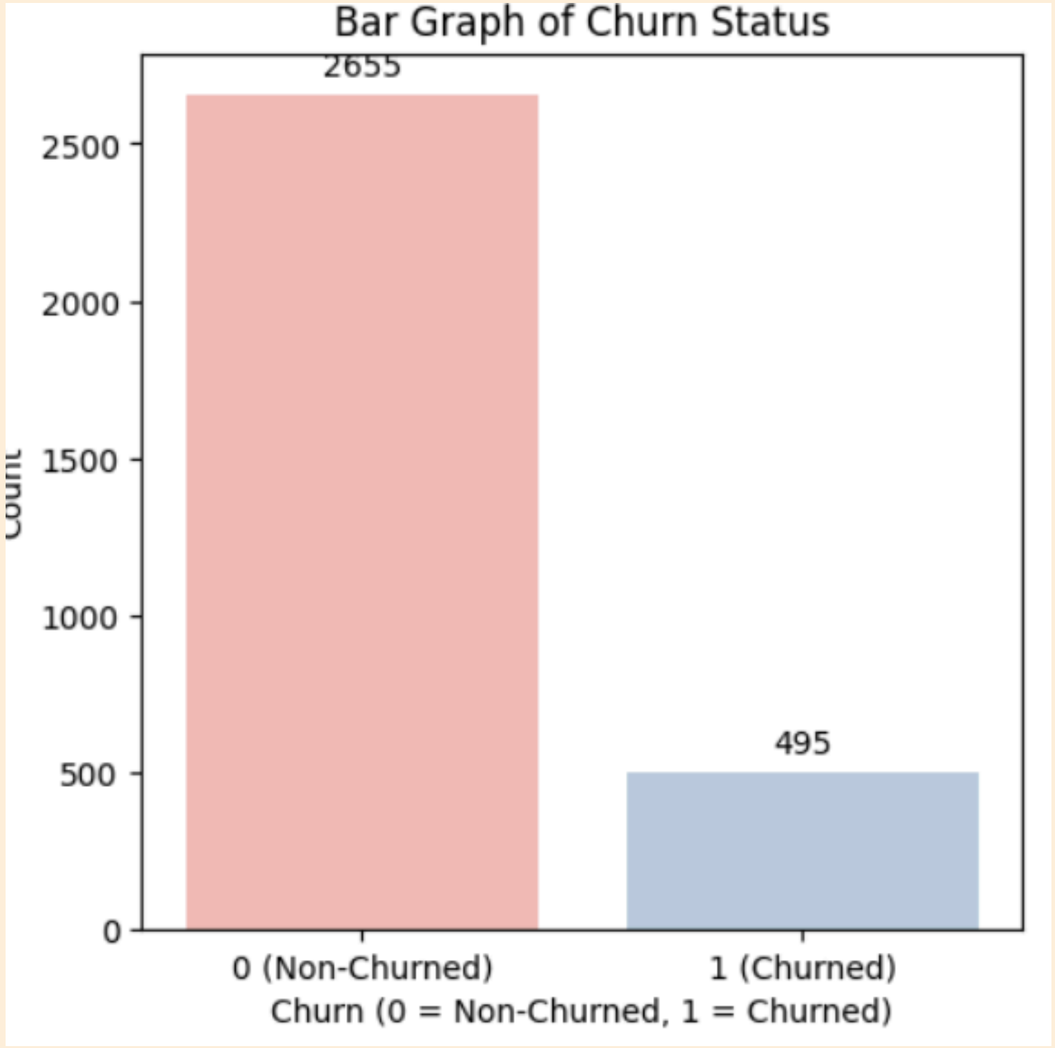
Count plots give an idea of the categorical variables and their distribution, showing, for example, the balance between churned and retained customers.

VIOLIN PLOT



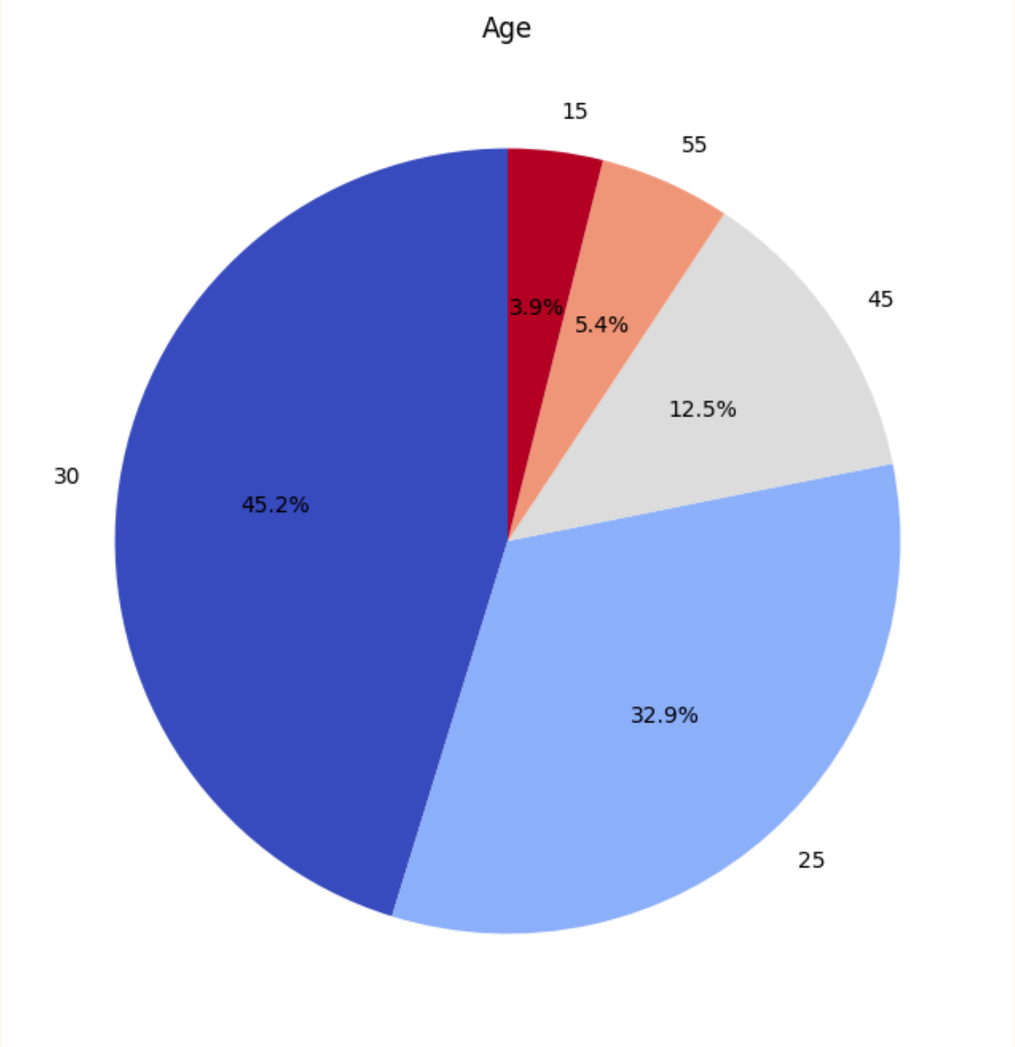
These plots show the spread and potential outliers for features. Violin plots combine box plot information with the distribution shape.

BAR PLOT



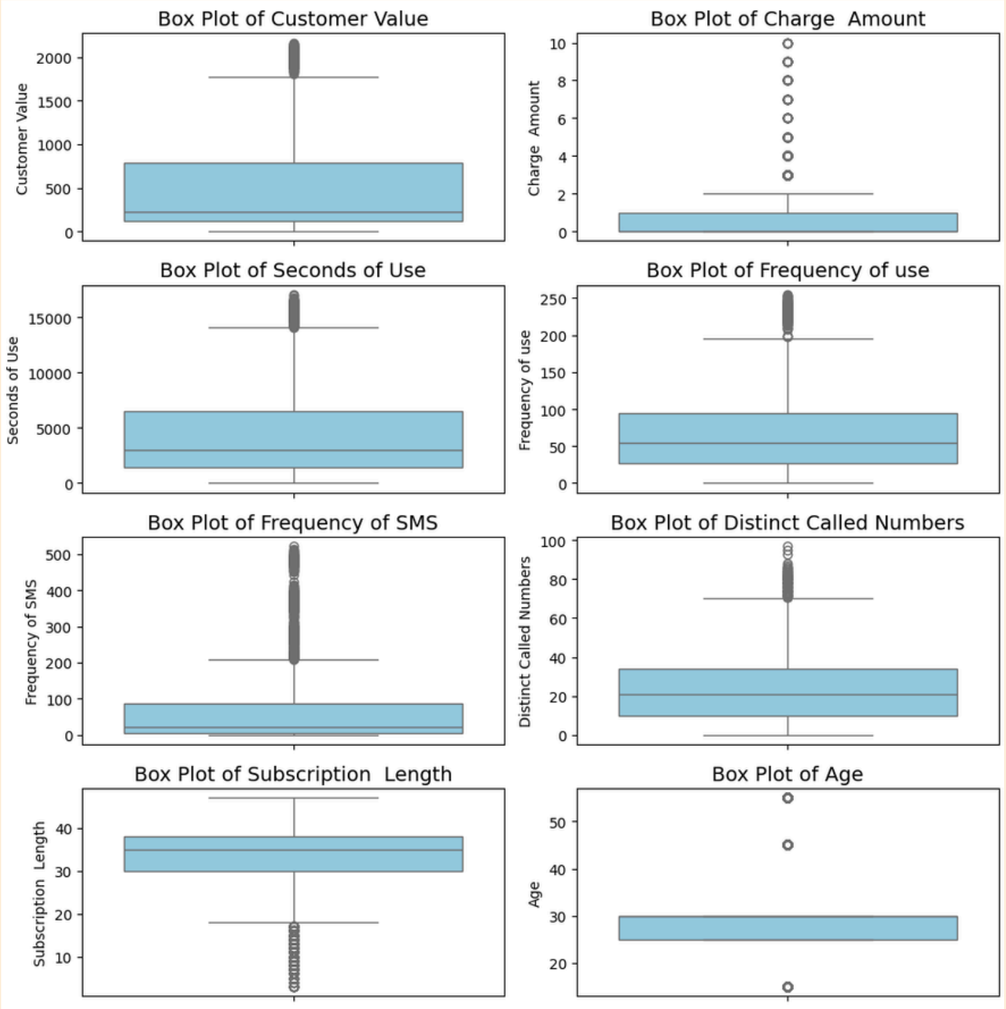
Bar plots summarize categorical data, showing the breakdown of groups like customer demographics.

PIE PLOT



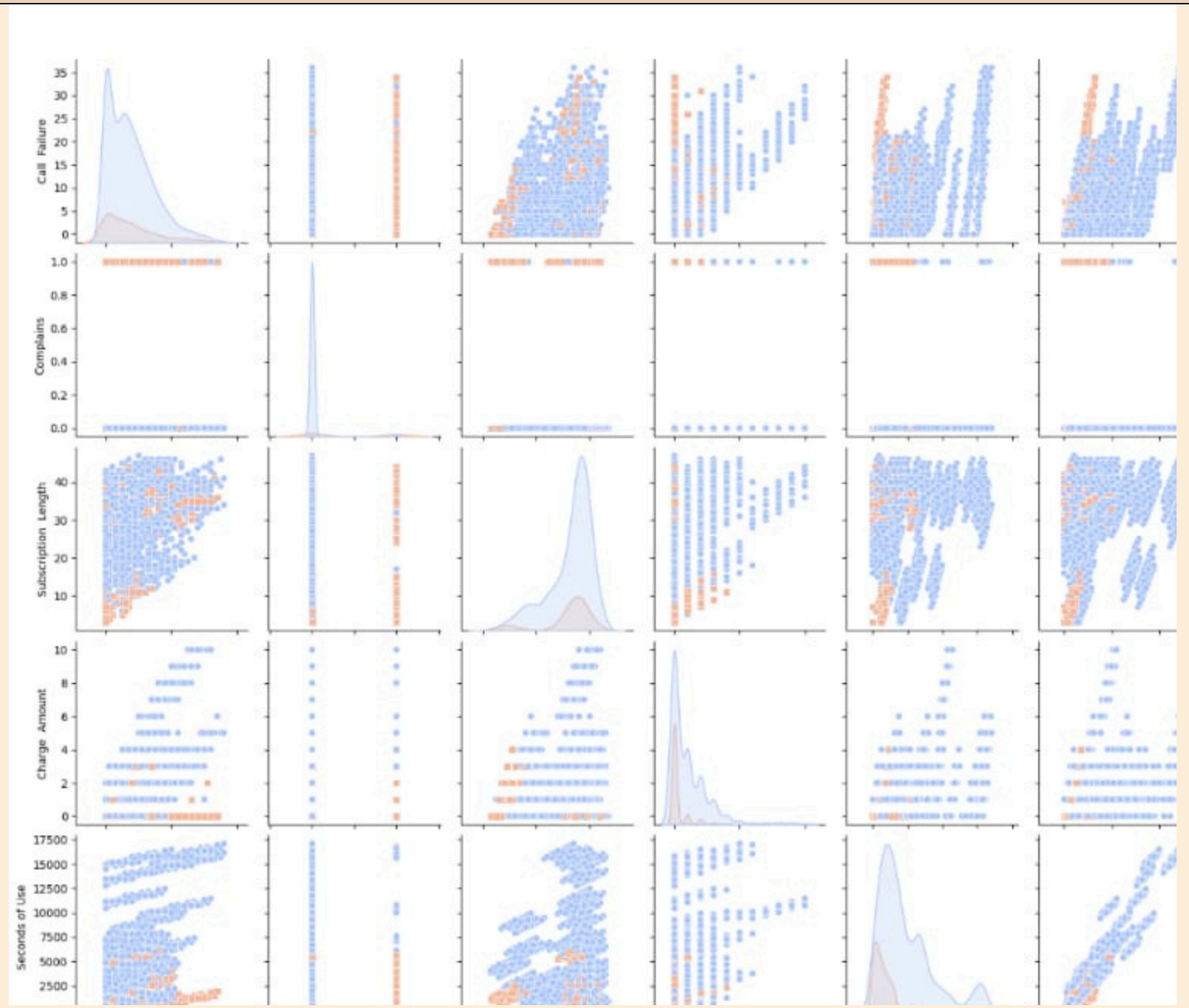
Pie plots visually break down proportions of a variable.

BOX PLOT



Box plots summarize the distribution of data, highlighting the median, variability, and outliers for easy comparison.

PAIR PLOT



A pair plot visualizes relationships between multiple numerical variables in a dataset through scatter plots and histograms.

Multicollinearity check

Multicollinearity occurs when independent variables in a model are too closely related or highly correlated , making it difficult to determine their individual effects on the dependent variable.

	variables	VIF
0	Call Failure	6.1
1	Complains	1.2
2	Subscription Length	15.4
3	Charge Amount	4.4
4	Seconds of Use	38.4
5	Frequency of use	44.3
6	Frequency of SMS	46.2
7	Distinct Called Numbers	6.9
8	Tariff Plan	15.8
9	Status	6.9
10	Age	16.8
11	Customer Value	74.2

Customer Value has higher vif

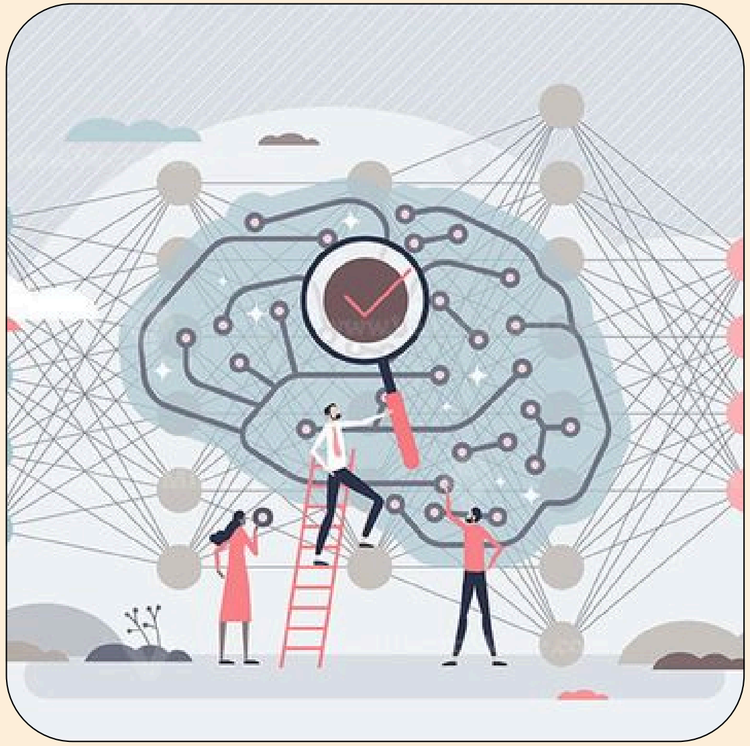
	variables	VIF
0	Call Failure	6.1
1	Complains	1.2
2	Subscription Length	13.3
3	Charge Amount	4.2
4	Seconds of Use	29.3
5	Frequency of use	44.3
6	Frequency of SMS	1.7
7	Distinct Called Numbers	6.9
8	Tariff Plan	15.8
9	Status	6.5
10	Age	12.5

Customer Value was removed

	variables	VIF
0	Call Failure	3.7
1	Complains	1.2
2	Charge Amount	2.3
3	Seconds of Use	3.0
4	Frequency of SMS	1.4
5	Age	2.6

The final variables with vif > 4

Machine Learning Algorithms



Machine Learning Algorithms

Logistic Regression:
A classification algorithm that models the probability of a binary outcome using a logistic function. It predicts probabilities and applies a threshold to classify data points.

K-Nearest Neighbors (KNN):
A non-parametric algorithm that classifies data based on the majority class of the nearest neighbors. It calculates the distance between data points to make predictions.

Support Vector Machine (SVM):
A classification algorithm that finds the optimal hyperplane to separate classes in a high-dimensional space. It uses support vectors to maximize the margin between classes.

Decision Tree:
A tree-like model that splits the data into branches based on feature values to predict outcomes. Each internal node represents a decision, and each leaf node represents a final class or value.

Machine Learning Algorithms

Random Forest:
An ensemble learning method that creates multiple decision trees and aggregates their results to improve accuracy. It reduces overfitting by averaging the predictions of individual trees.

AdaBoost:
An ensemble technique that combines weak classifiers to form a strong classifier by assigning higher weights to misclassified instances. It iteratively adjusts the weights to improve prediction accuracy.

XGBoost:
A gradient boosting algorithm that improves model performance by combining multiple weak learners. It is optimized for speed and efficiency, often used for structured/tabular data.

80-20 SPLIT

ALGORITHMS	MODEL -1 ACCURACY	MODEL - 2 ACCURACY
LOGISTIC REGRESSION	0.841	0.897
K- NEAREST NEIGHBOR	0.843	0.895
SUPPORT VECTOR MACHINE	0.827	0.827
DECISION TREE	0.921	0.905
RANDOM FOREST	0.948	0.921
ADABOOST	0.902	0.895
XGBOOST	0.949	0.924

Model 1 - Before VIF | Model 2 - After VIF

75-25 SPLIT

ALGORITHMS	MODEL -1 ACCURACY	MODEL - 2 ACCURACY
LOGISTIC REGRESSION	0.838	0.896
K- NEAREST NEIGHBOR	0.822	0.895
SUPPORT VECTOR MACHINE	0.822	0.822
DECISION TREE	0.928	0.918
RANDOM FOREST	0.940	0.925
ADABOOST	0.909	0.895
XGBOOST	0.949	0.920

Model 1 - Before VIF | Model 2 - After VIF

70-30 SPLIT

ALGORITHMS	MODEL -1 ACCURACY	MODEL - 2 ACCURACY
LOGISTIC REGRESSION	0.837	0.890
K- NEAREST NEIGHBOR	0.82	0.8
SUPPORT VECTOR MACHINE	0.821	0.821
DECISION TREE	0.921	0.913
RANDOM FOREST	0.945	0.917
ADABOOST	0.901	0.8
XGBOOST	0.942	0.922

Model 1 - Before VIF | Model 2 - After VIF

60-40 SPLIT

ALGORITHMS	MODEL -1 ACCURACY	MODEL - 2 ACCURACY
LOGISTIC REGRESSION	0.844	0.899
K- NEAREST NEIGHBOR	0.830	0.897
SUPPORT VECTOR MACHINE	0.832	0.832
DECISION TREE	0.927	0.912
RANDOM FOREST	0.946	0.921
ADABOOST	0.902	0.897
XGBOOST	0.944	0.919

Model 1 - Before VIF | Model 2 - After VIF

COMPARISION

75-25 Split Before Applying VIF

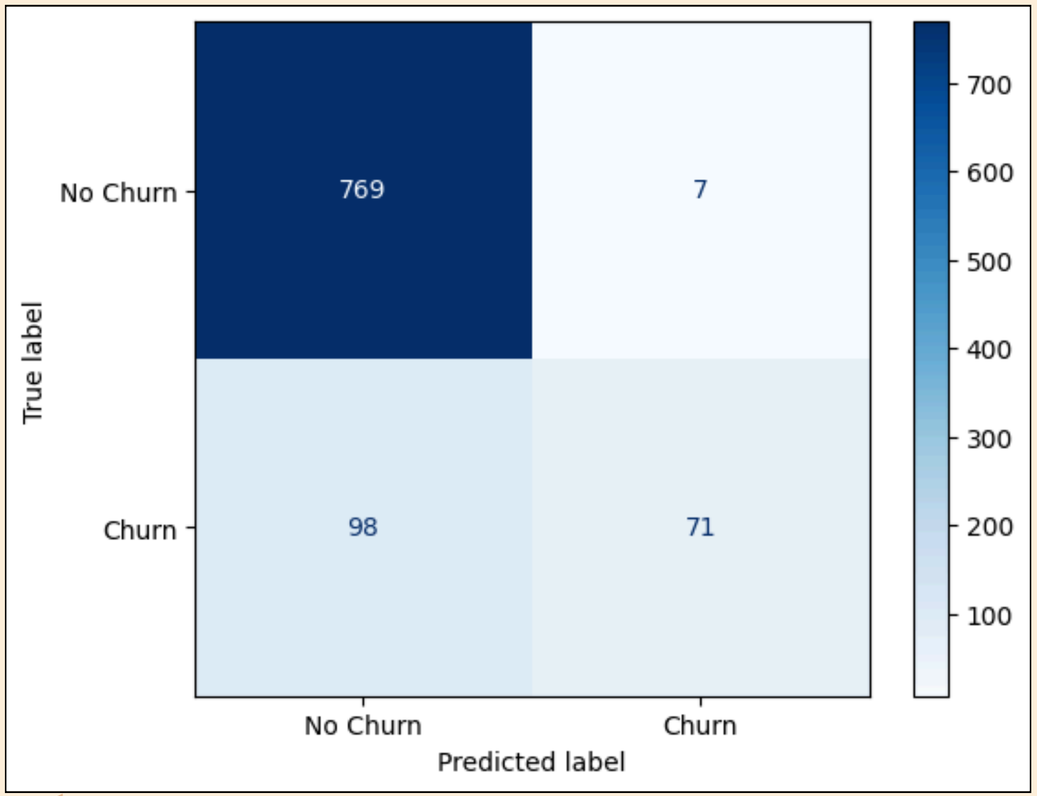
ALGORITHMS	ACCURACY
LOGISTIC REGRESSION	0.838
K- NEAREST NEIGHBOR	0.822
SUPPORT VECTOR MACHINE	0.822
DECISION TREE	0.928
RANDOM FOREST	0.940
ADABOOST	0.909
XGBOOST	0.949

75-25 Split After Applying VIF

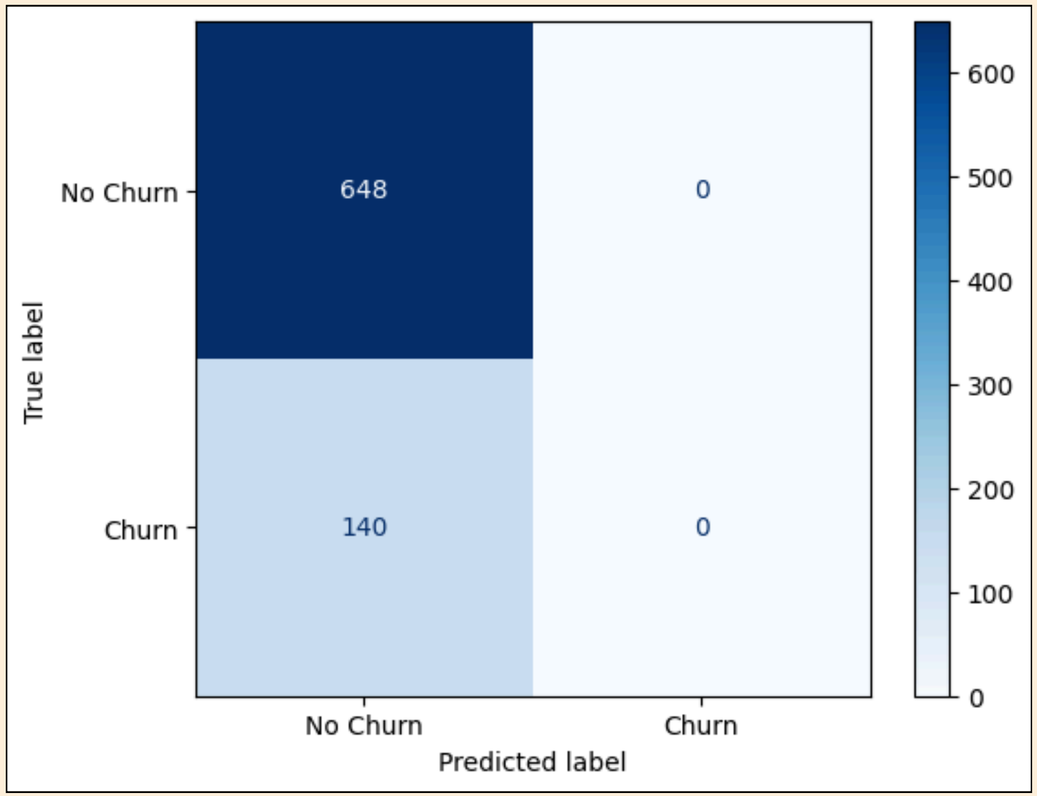
ALGORITHMS	ACCURACY
LOGISTIC REGRESSION	0.896
K- NEAREST NEIGHBOR	0.895
SUPPORT VECTOR MACHINE	0.822
DECISION TREE	0.918
RANDOM FOREST	0.925
ADABOOST	0.895
XGBOOST	0.920

CONFUSION MATRICS

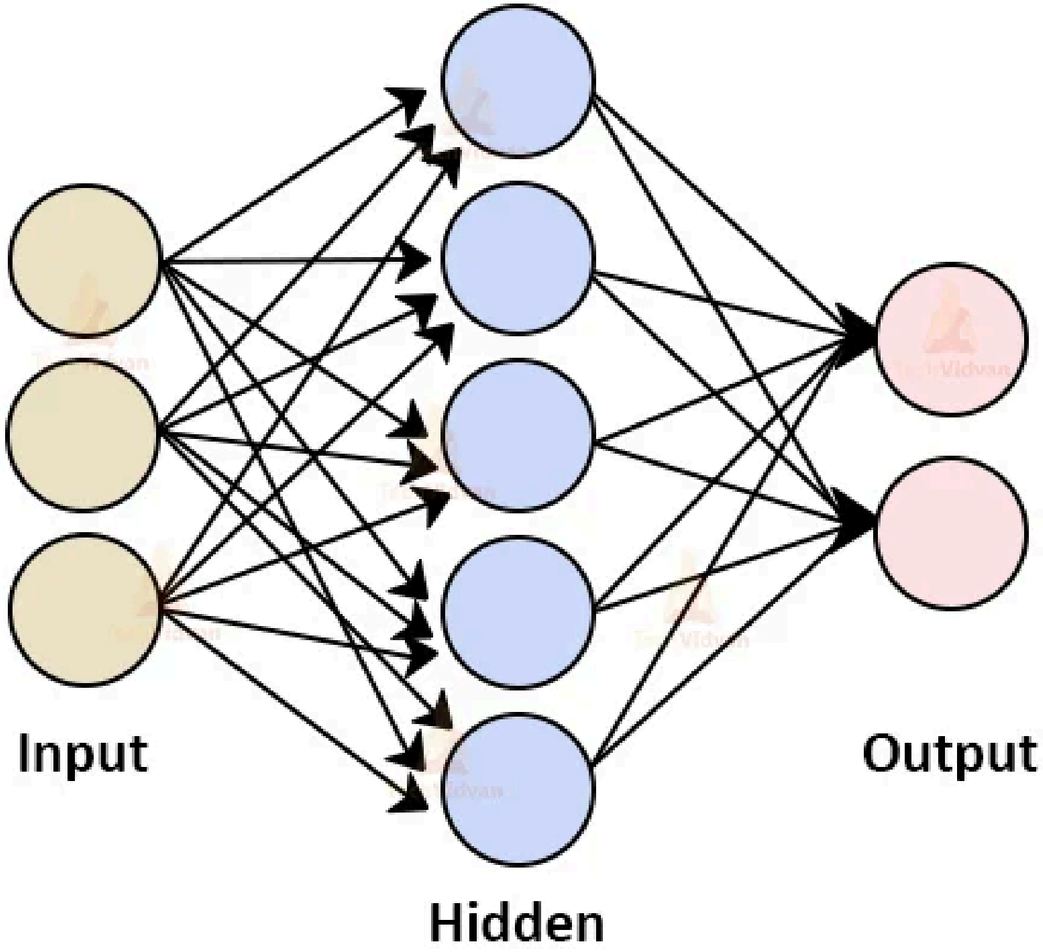
XG BOOST
Before Applying VIF for 75-25 split



RANDOM FOREST
After Applying VIF for 75-25 split

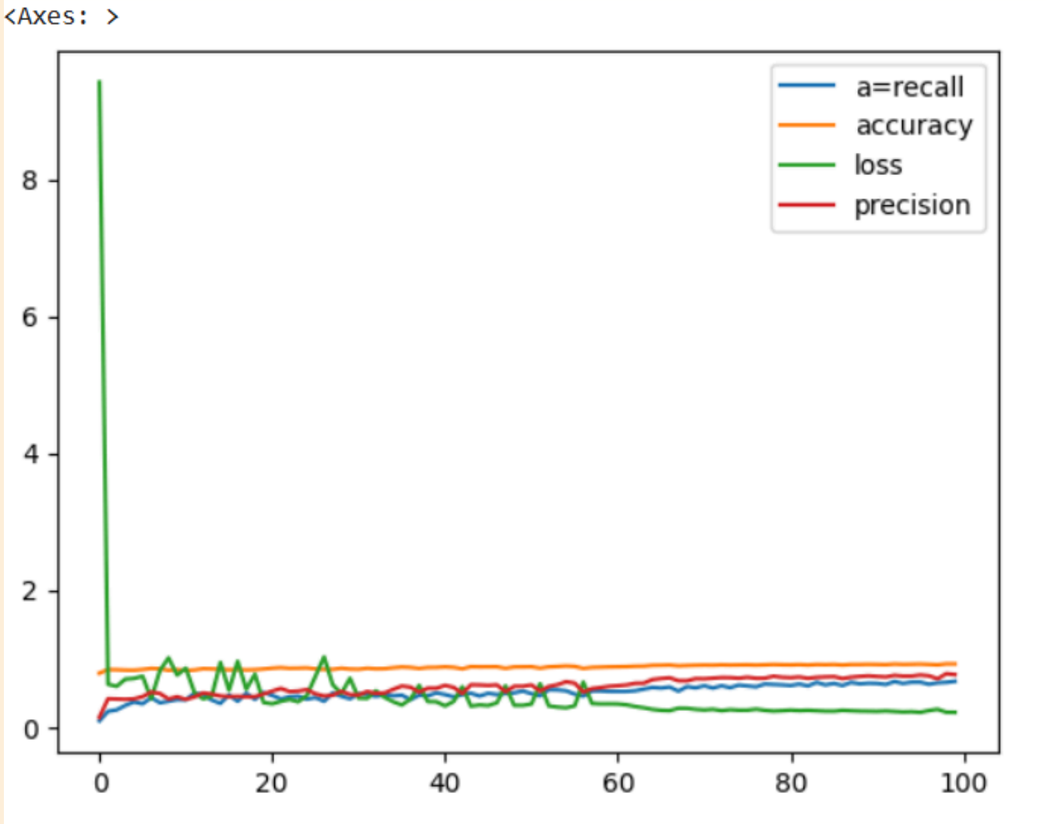


Architecture of Artificial Neural Network



Train test	Architecture	Optimizer	Epochs	Accuracy
80-20	20-17-14-11-8-1	Adam	120	0.8661
80-20	10-7-5-1	Adam	100	0.900
75-25	20-18-16-14-1	Adam	100	0.889
75-25	10-7-5-1	Adam	100	0.887
70-30	20-16-12-8-1	Adam	100	0.913
70-30	20-15-5-1	Adam	100	0.858
60-40	20-13-9-5-1	Adam	100	0.851
60-40	20-10-5-1	Adam	100	0.884

Neural Network Plot



Train Test Split	**70-30**
Architecture	**20-16-12-8-1**
Optimizer	**Adam**
Epochs	**100**

SUMMARY

- The goal of the project was to predict customer churn (i.e., the likelihood of a customer leaving a service or subscription) using machine learning techniques. By leveraging historical customer data, the project aimed to classify customers as "churned" or "retained."
- A 75-25 train-test split was identified as optimal, offering a perfect balance between sufficient training data for model learning and reliable evaluation on unseen data.
- For the given dataset, the XGBoost Algorithm demonstrated the highest accuracy of 94.9%, outperforming other models.
- However, after removing high multicollinearity variables, the Random Forest Algorithm delivered the best results with an accuracy of 92.5%, highlighting its ability to adapt effectively to reduced feature sets.

FUTURESCOPE

1. Multi-Channel Churn Prevention: Expand the model to include data from multiple customer interaction channels such as social media, chat support, and emails. This will provide a holistic view of customer sentiment and enable interventions across platforms for a seamless customer experience.
2. Gamification for Retention: Introduce gamification as a retention strategy based on model insights. For instance, reward loyal customers with points, badges, or incentives to prevent churn in high-risk segments, creating a more engaging customer experience.
3. Integration with Marketing Automation: Link the churn prediction system with marketing automation tools to trigger automated retention campaigns. For example, high-churn-risk customers can receive personalized marketing emails or notifications in real-time.

INSIGHTS

- High-Risk Customer Segments: The model highlights specific customer demographics or behavioral traits most prone to churn, such as those with shorter tenure, irregular usage patterns, or higher complaint frequency.
- Feature Importance and Root Causes: This insight provides actionable areas for improvement, such as revising pricing models, enhancing customer support, or improving billing transparency.
- Early Warning System for Business Decision-Making: Customers with declining engagement over time can be flagged, and proactive measures like special offers or service upgrades can be implemented to re-engage them.

THANK YOU

Colab Notebook

Done by:

Vaishali Vadde | Syed Danish | Shravya | Pravalika

APPENDIX

Loading the dataset

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

	Call Failure	Complains	Subscription Length	Charge Amount	Seconds of Use	Frequency of use	Frequency of SMS	Distinct Called Numbers	Age Group	Tariff Plan	Status	Age	Customer Value	Churn
0	8	0	38	0	4370	71	5	17	3	1	1	30	197.640	0
1	0	0	39	0	318	5	7	4	2	1	2	25	46.035	0
2	10	0	37	0	2453	60	359	24	3	1	1	30	1536.520	0
3	10	0	38	0	4198	66	1	35	1	1	1	15	240.020	0
4	3	0	38	0	2393	58	2	33	1	1	1	15	145.805	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3145	21	0	19	2	6697	147	92	44	2	2	1	25	721.980	0
3146	17	0	17	1	9237	177	80	42	5	1	1	55	261.210	0
3147	13	0	18	4	3157	51	38	21	3	1	1	30	280.320	0
3148	7	0	11	2	4695	46	222	12	3	1	1	30	1077.640	0
3149	8	1	11	2	1792	25	7	9	3	1	1	30	100.680	1

3150 rows × 14 columns

Checking Null Values

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

	0
Call Failure	0
Complains	0
Subscription Length	0
Charge Amount	0
Seconds of Use	0
Frequency of use	0
Frequency of SMS	0
Distinct Called Numbers	0
Age Group	0
Tariff Plan	0
Status	0
Age	0
Customer Value	0
Churn	0

dtype: int64

Checking for the data type

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3150 entries, 0 to 3149
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Call  Failure           3150 non-null   int64
 1   Complains               3150 non-null   int64
 2   Subscription  Length    3150 non-null   int64
 3   Charge  Amount          3150 non-null   int64
 4   Seconds of Use          3150 non-null   int64
 5   Frequency of use        3150 non-null   int64
 6   Frequency of SMS        3150 non-null   int64
 7   Distinct Called Numbers 3150 non-null   int64
 8   Age Group               3150 non-null   int64
 9   Tariff Plan             3150 non-null   int64
 10  Status                  3150 non-null   int64
 11  Age                     3150 non-null   int64
 12  Customer Value          3150 non-null   float64
 13  Churn                   3150 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 344.7 KB
```

Replacing the variables

```
df['Tariff Plan'] = df['Tariff Plan'].replace(2, 0)
df['Status'] = df['Status'].replace(2, 0)
```

Dropping the age column

```
df=df.drop('Age Group',axis=1)
```

Feature and Target Variable Division

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

Multicollinearity Check

MULTICOLLINEARITY - VIF

```
[ ] # Import library for VIF
    from statsmodels.stats.outliers_influence import variance_inflation_factor
    import pandas as pd
    from sklearn.model_selection import train_test_split

    def calc_vif(X):

        # Calculating VIF
        vif = pd.DataFrame()
        vif["variables"] = X.columns
        vif["VIF"] = [variance_inflation_factor(X.values, i).round(1) for i in range(X.shape[1])]

        return(vif)

    # Now you can call calc_vif with X_train
    calc_vif(X)
```

	variables	VIF
0	Call Failure	6.1
1	Complains	1.2
2	Subscription Length	15.4
3	Charge Amount	4.4
4	Seconds of Use	38.4
5	Frequency of use	44.3
6	Frequency of SMS	46.2
7	Distinct Called Numbers	6.9
8	Tariff Plan	15.8
9	Status	6.9
10	Age	16.8
11	Customer Value	74.2

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

80-20 SPLIT

```
logreg = LogisticRegression(C=1e9)

logreg.fit(X_train1, y_train1)
predictions1 = logreg.predict(X_test1)
print(predictions1)
```

```
[1 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0
```

Before applying VIF

```
logreg = LogisticRegression(C=1e9)
logreg.fit(x_nomulti_train1, y_nomulti_train1)
predictions1 = logreg.predict(x_nomulti_test1)
print(predictions1)
```

```
[1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 1
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0
 0]
```

After applying VIF

KNN

```
[ ] model = KNeighborsRegressor(n_neighbors=21)
    model.fit(x_nomulti_train1, y_nomulti_train1)
```

```
KNeighborsRegressor
KNeighborsRegressor(n_neighbors=21)
```

```
y_pred1 = model.predict(x_nomulti_test1)
knn = pd.DataFrame({'Predicted':y_pred1,'Actual':y_nomulti_test1})
knn
```

	Predicted	Actual
422	0.238095	1
1581	0.333333	1
1185	0.190476	0
683	0.000000	0
305	0.142857	0
...	...	...
1276	0.190476	1
3099	0.285714	1

Before applying VIF

```
model=KNeighborsClassifier(n_neighbors=25)
model.fit(X_train1, y_train1)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=25)
```

```
[ ] y_pred1 = model.predict(X_test1)
    knn = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
    knn
```

After applying VIF

SVM

```
model = SVC(kernel='rbf')
model.fit(X_train1, y_train1)
```

```
SVC()
```

```
y_pred1 = model.predict(X_test1)
svm = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
print(svm)
```

```
      Predicted  Actual
422           0       1
1581          0       1
1185          0       0
683           0       0
305           0       0
...         ...     ...
1276          0       1
3099          0       1
841           0       0
44            0       0
615           0       0

[630 rows x 2 columns]
```

Before applying VIF

```
# Replace the regressor with a classifier
from sklearn.svm import SVC
model = SVC(kernel='rbf')

# Fit the model
model.fit(x_nomulti_train1, y_nomulti_train1)

# Make predictions using the classifier model
y_pred1 = model.predict(x_nomulti_test1)

# Evaluate the model's performance using classification metrics
print("Accuracy:", accuracy_score(y_nomulti_test1, y_pred1))
print(classification_report(y_nomulti_test1, y_pred1))
cm = confusion_matrix(y_nomulti_test1, y_pred1)
```

```
Accuracy: 0.8269841269841269
              precision    recall  f1-score   support

           0       0.83      1.00      0.91       521
           1       0.00      0.00      0.00       109

    accuracy                           0.83       630
   macro avg       0.41      0.50      0.45       630
weighted avg       0.68      0.83      0.75       630
```

After applying VIF

Decision Tree

```
dt=DecisionTreeClassifier()
dt.fit(X_train1,y_train1)
```

```
DecisionTreeClassifier()
DecisionTreeClassifier()
```

+ Code + Text

```
y_pred1=dt.predict(X_test1)
```

+ Code + Text

```
print("Accuracy:",accuracy_score(y_test1,y_pred1))
```

```
Accuracy: 0.919047619047619
```

```
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)
clf.fit(X_train1,y_train1)
y_pred1 = clf.predict(X_test1)
print("Accuracy:",metrics.accuracy_score(y_test1, y_pred1))
classification_rep = classification_report(y_test1, y_pred1)
print(classification_rep)
```

```
Accuracy: 0.8888888888888888
              precision    recall  f1-score   support

           0       0.89      0.99      0.94       521
           1       0.91      0.39      0.55       109

    accuracy                           0.89       630
```

Before applying VIF

```
dt = DecisionTreeClassifier()
dt.fit(x_nomulti_train1,y_nomulti_train1)
```

```
DecisionTreeClassifier()
DecisionTreeClassifier()
```

```
y_pred1=dt.predict(x_nomulti_test1)
print("Accuracy:",accuracy_score(y_nomulti_test1,y_pred1))
print(classification_report(y_nomulti_test1, y_pred1))
```

```
Accuracy: 0.9047619047619048
              precision    recall  f1-score   support

           0       0.94      0.95      0.94       521
           1       0.74      0.69      0.71       109

    accuracy                           0.90       630
   macro avg       0.84      0.82      0.83       630
weighted avg       0.90      0.90      0.90       630
```

After applying VIF

Random Forest

```
rf=RandomForestClassifier()
rf.fit(X_train1,y_train1)
```

```
RandomForestClassifier()
```

```
y_pred1=rf.predict(X_test1)
print("Accuracy:",accuracy_score(y_test1,y_pred1))
print(classification_report(y_test1, y_pred1))
print(confusion_matrix(y_test1, y_pred1))
```

```
Accuracy: 0.9428571428571428
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       521
           1       0.89      0.76      0.82       109

    accuracy                           0.94       630
   macro avg       0.92      0.87      0.89       630
```

Before applying VIF

+ Code + Text

```
rf = RandomForestClassifier()
rf.fit(x_nomulti_train1,y_nomulti_train1)
```

```
RandomForestClassifier()
```

```
y_pred1=rf.predict(x_nomulti_test1)
print("Accuracy:",accuracy_score(y_nomulti_test1,y_pred1))
print(classification_report(y_nomulti_test1, y_pred1))
print(confusion_matrix(y_nomulti_test1, y_pred1))
```

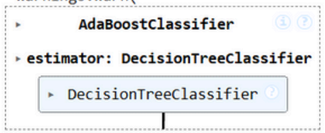
```
Accuracy: 0.9206349206349206
              precision    recall  f1-score   support

           0       0.94      0.97      0.95       521
           1       0.82      0.69      0.75       109

    accuracy                           0.92       630
   macro avg       0.88      0.83      0.85       630
weighted avg       0.92      0.92      0.92       630

[[505  16]
 [ 34  75]]
```

After applying VIF

AdaBoost

```
adaboost = AdaBoostClassifier(estimator=base_estimator, # Changed argument name here
                              n_estimators=3,random_state=0)
adaboost.fit(X_train1, y_train1)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecat
  warnings.warn(
```



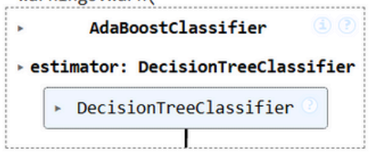

```
y_pred1 = adaboost.predict(X_test1)
print("Accuracy:",accuracy_score(y_test1,y_pred1))
print(classification_report(y_test1, y_pred1))
print(confusion_matrix(y_test1, y_pred1))
```

```
Accuracy: 0.9015873015873016
              precision    recall  f1-score   support

           0       0.91      0.97      0.94       521
           1       0.81      0.56      0.66       109

    accuracy                           0.90       630
```

Before applying VIF

```
adaboost = AdaBoostClassifier(estimator=base_estimator, # Changed argument name here
                              n_estimators=3,random_state=0)
adaboost.fit(x_nomulti_train2, y_nomulti_train2)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecate
  warnings.warn(
```




```
y_pred2 = adaboost.predict(x_nomulti_test2)
print("Accuracy:",accuracy_score(y_nomulti_test2,y_pred2))
print(classification_report(y_nomulti_test2, y_pred2))
print(confusion_matrix(y_nomulti_test2, y_pred2))
```

```
Accuracy: 0.8946700507614214
              precision    recall  f1-score   support

           0       0.89      0.99      0.94       648
           1       0.90      0.46      0.61       140

    accuracy                           0.89       788
   macro avg       0.90      0.72      0.77       788
```

After applying VIF

XGBoost

```
model1 = xgb.XGBClassifier()
model2 = xgb.XGBClassifier(n_estimators=100, max_depth=8, learning_rate=0.1, subsample=0.5)
train_model1 = model1.fit(X_train1, y_train1)
train_model2 = model2.fit(X_train1, y_train1)
pred1 = train_model1.predict(X_test1)
pred2 = train_model2.predict(X_test1)
print('Model 1 XGboost Report %r' % (classification_report(y_test1, pred1)))
print('Model 2 XGboost Report %r' % (classification_report(y_test1, pred2)))
print("Accuracy for model 1: %.2f" % (accuracy_score(y_test1, pred1) * 100))
print("Accuracy for model 2: %.2f" % (accuracy_score(y_test1, pred2) * 100))
print(confusion_matrix(y_test1, pred1))
```

```
Model 1 XGboost Report '            precision    recall  f1-score   support\n\n          0       0.96      0.98      0.97       521\n
Model 2 XGboost Report '            precision    recall  f1-score   support\n\n          0       0.96      0.98      0.97       521\n
Accuracy for model 1: 95.08
Accuracy for model 2: 94.92
[[510  11]
 [ 20  89]]
```

Before applying VIF

```
model1 = xgb.XGBClassifier()
model2 = xgb.XGBClassifier(n_estimators=100, max_depth=8, learning_rate=0.1, subsample=0.5)
train_model1 = model1.fit(x_nomulti_train1, y_nomulti_train1)
train_model2 = model2.fit(x_nomulti_train1, y_nomulti_train1)
pred1 = train_model1.predict(x_nomulti_test1)
pred2 = train_model2.predict(x_nomulti_test1)
pred1 = train_model1.predict(x_nomulti_test1)
pred2 = train_model2.predict(x_nomulti_test1)

print('Model 1 XGboost Report %r' %  (classification_report(y_nomulti_test1, pred1)))
print('Model 2 XGboost Report %r' % (classification_report(y_nomulti_test1, pred2)))
print("Accuracy for model 1: %.2f" % (accuracy_score(y_nomulti_test1, pred1) * 100))
print("Accuracy for model 2: %.2f" % (accuracy_score(y_nomulti_test1, pred2) * 100))
```

```
Model 1 XGboost Report '            precision    recall  f1-score   support\n\n          0       0.93      0.97      0.95       521\n
Model 2 XGboost Report '            precision    recall  f1-score   support\n\n          0       0.94      0.97      0.95       521\n
Accuracy for model 1: 91.11
Accuracy for model 2: 92.38
```

After applying VIF