Customer Churn

Prediction
Using Machine Learning

Group:09/BHAVANS VIVEKANADA DEGREE COLLEGE

T.Naga Sravanthi/Vaishnavi Shivalingala/S.Siddhartha

Abstract

The project aims to develop a robust model for predicting customer churn in various industries, enabling organizations to implement proactive retention strategies.

This study investigates a range of machine learning algorithms, including Logistic Regression, Decision Trees, Random Forests, K-Nearest Neighbors, Support Vector Machines, Bagging, and Boosting techniques, to effectively identify patterns and factors contributing to customer attrition. By leveraging these predictive models, businesses can enhance customer satisfaction and loyalty, leading to improved operational efficiency and long-term profitability.

Objective

To find the best machine learning model for predicting customer churn so companies can take steps to keep their customers happy and reduce the number of people who leave. This goal aims to help businesses understand why customers leave and take action to improve their experience and loyalty.

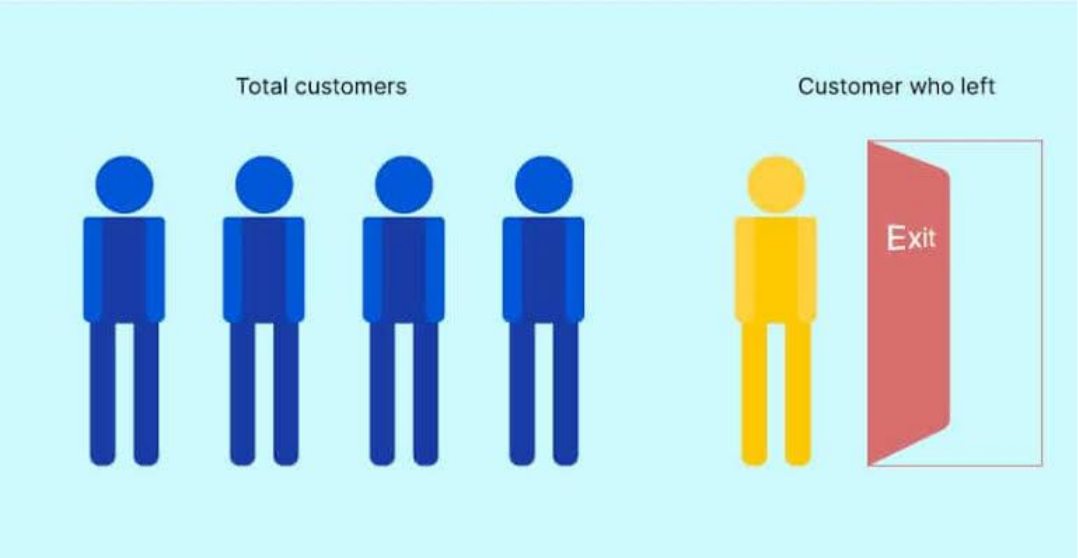
CONTENT

- Introduction
- Literature Review
- Data Pre-Processing
- Exploratory Data Analysis
- Data Modeling & Evaluation
- Summary
- Appendix

INTRODUCTION

- Customer churn prediction is about figuring out
- which customers are likely to stop using a company's products or services. By analysing data, businesses can identify patterns or signs that suggest a customer might leave

- New tools, like data analytics,
  help companies retain customers.
  Machine learning predicts churn,
  revealing why customers leave.
  This insight aids strategy building,
  boosts satisfaction, and builds bonds,
  ultimately reducing customer churn.



Data Pre-Processing

Data

Data Set: Our data set contains a total of 3150 rows of data, each row representing a customer, bear information for 13 columns.

Source: https://archive.ics.uci.edu/dataset/563/iranian+churn+dataset

Variables:

1	Continuous variables
2	Call Failutres
3	Complains
4	subscription Length
5	Charge Amount
6	Seconds of Use
7	Frequency of use
8	Frequency of SMS
9	Distinct Called Numbers
10	Age Group
11	Tariff Plan
12	Status
13	Age
14	Customer Value
15	Churn

	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

Data Cleaning

- Removing Columns: Since the dataset had multiple entries for the same dates with different times, we decided to remove the "date" column to simplify the data.

- Next, we looked for any missing values and checked how many unique values each column had.

- We also transformed the categorical variables by labeling them. If you have a "Shift Type" variable indicating "Day" or "Night," you can create two columns: "Day Shift" and "Night Shift." For day shifts, set "Day Shift" to 1 and "Night Shift" to 0; for night shifts, set "Night Shift" to 1 and "Day Shift" to 0.

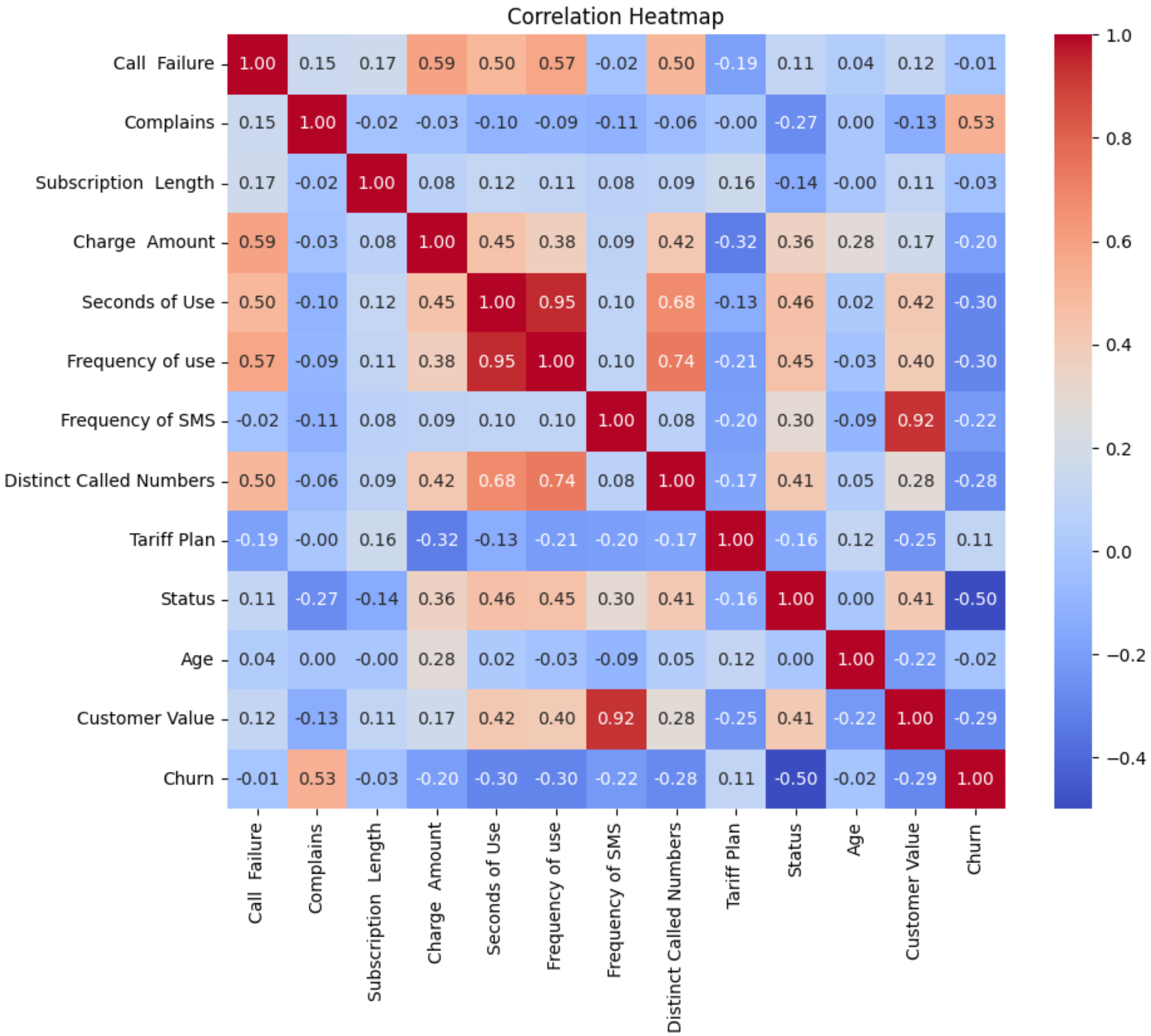
- To perform dummy variable encoding we divided the data into two sets
  - continuous data
  - categorical data

- But this data set contains only continuous variable so we can ignore the "Dummy Variable Encoding"

```
 #   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)
```

Exploratory Data Analysis

Correlation Matrix

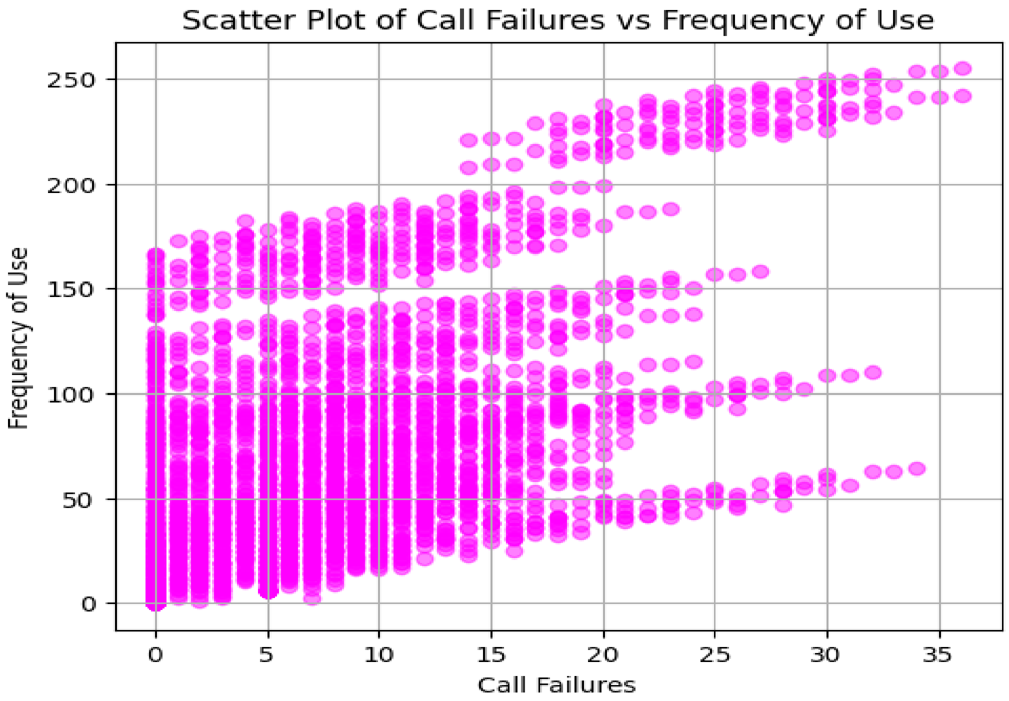


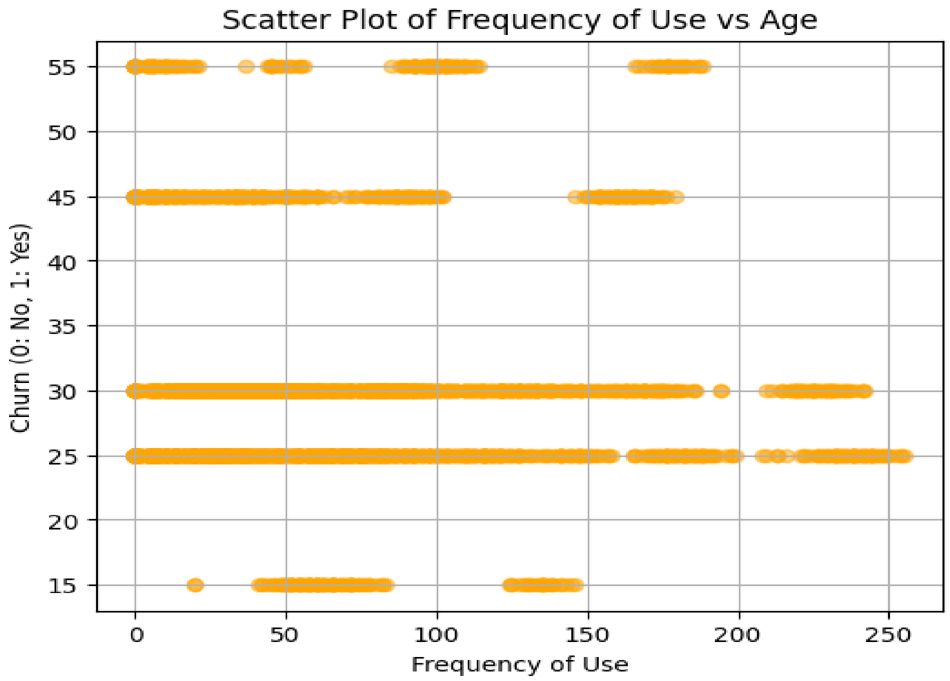
As we can see terms like

- **Call Failure**
- **Charge Amount**
- **Seconds of Use**
- **Frequency of Use**
- **Distinct Called Numbers**
- **Complaints**
- **Churn**
- **Customer Value**

are most positively Correlated

Scatter Plot

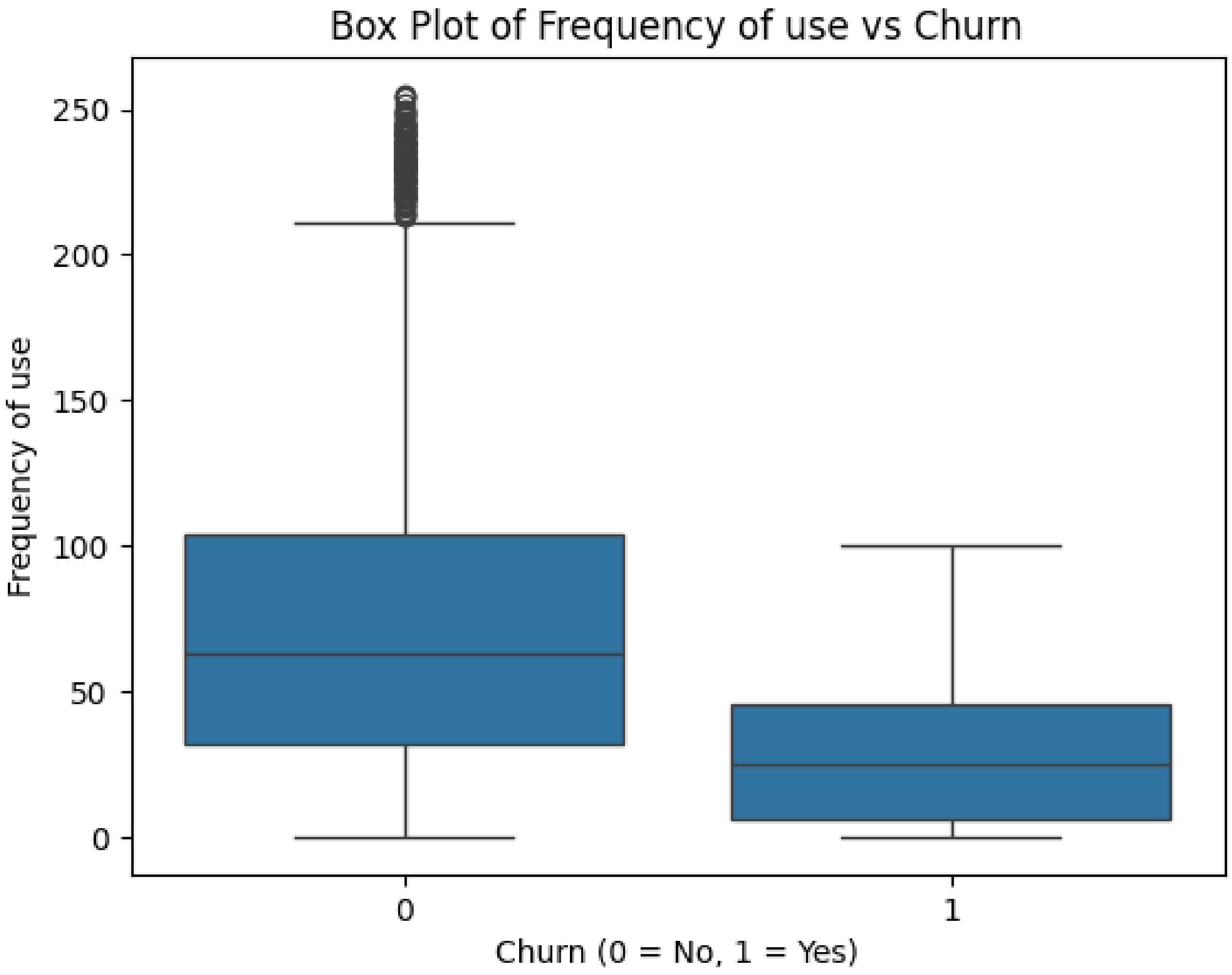




- The scatter plot illustrates the relationship between the frequency of use of a service and the number of call failures. The data suggests a positive correlation, indicating that as the frequency of use increases, the number of call failures also tends to rise.

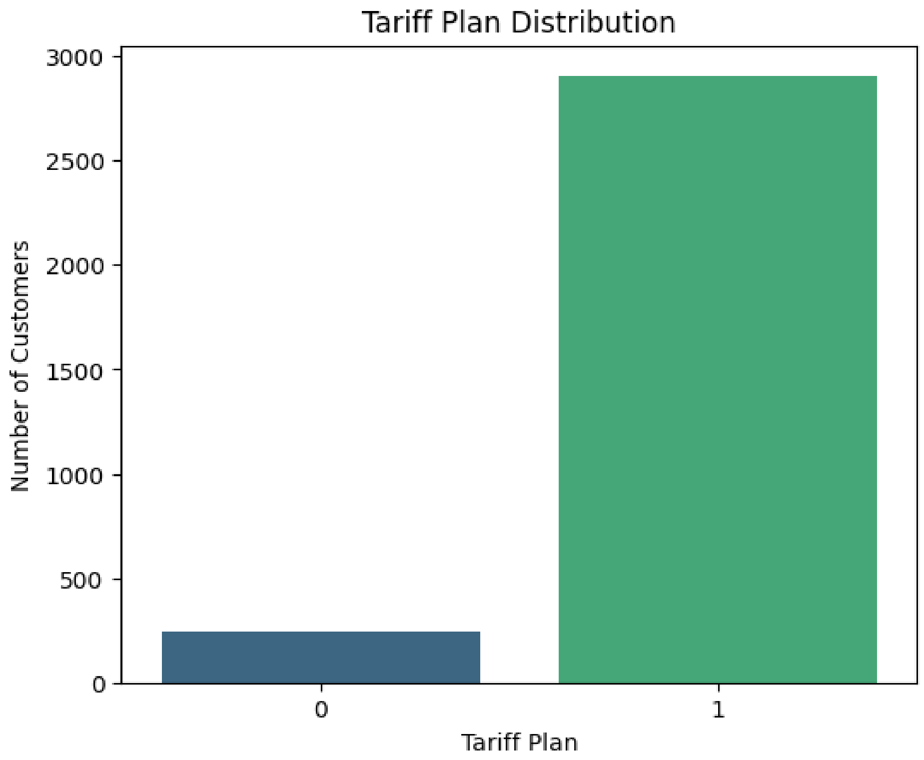
- This graph shows the relationship between how often customers use a service and their age.
- There doesn't seem to be a clear relationship between age and how often customers use the service.

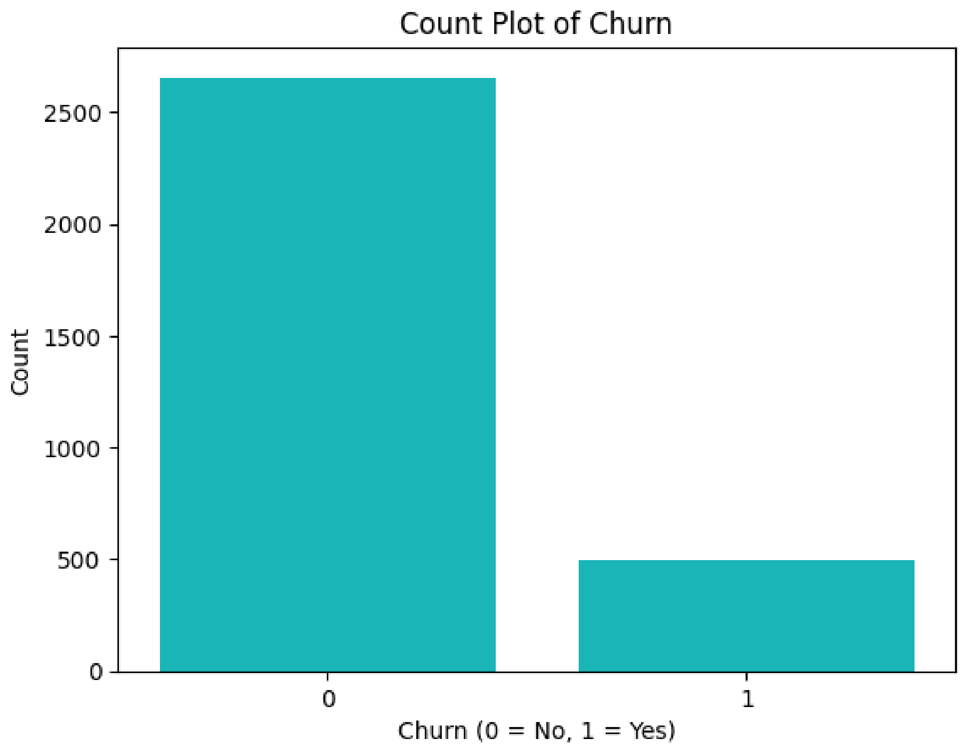
Box Plot



- This graph shows how often customers used a service before they stopped using it (churned).
- Customers who stopped using the service (churned) used it less often than those who kept using it.

Count Plot

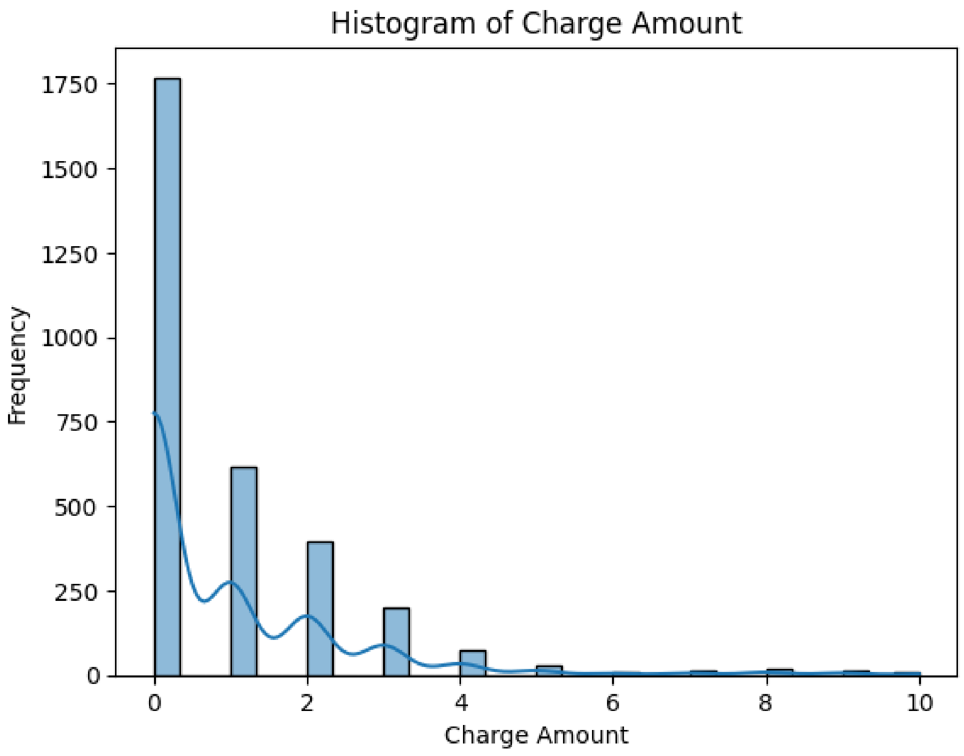


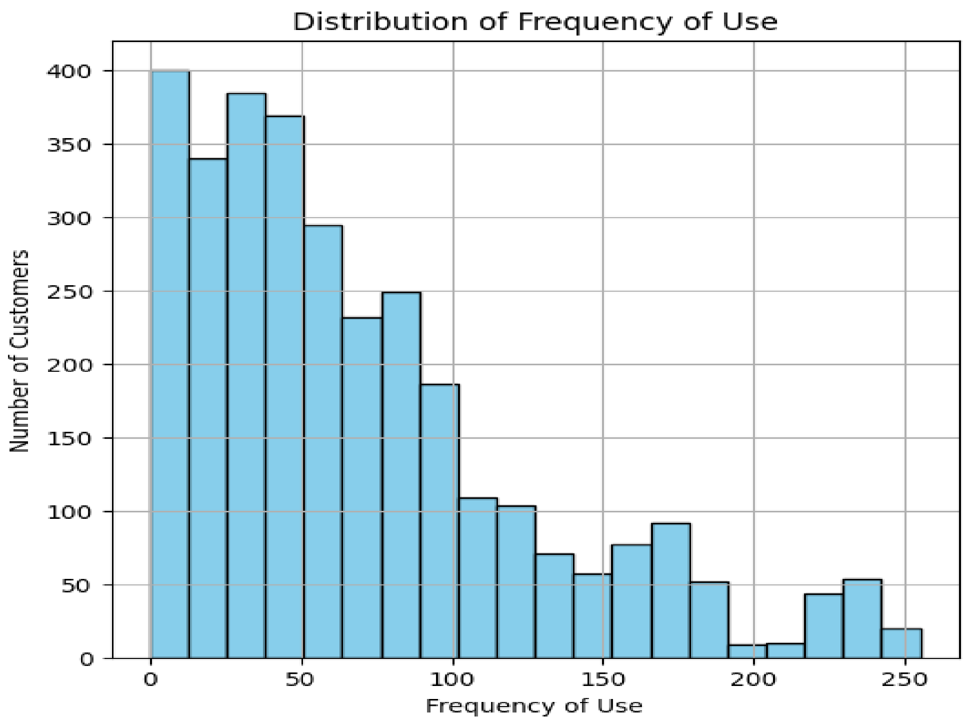


- This graph shows how many customers chose each of two different tariff plans.
- Most customers (around 3000) chose Tariff Plan 1, while only a few (around 300) chose Tariff Plan 0.

- This graph shows how many customers churned (stopped using the service) and how many did not.
- Most customers (around 2500) did not churn, while only a few (around 500) did.

Histogram

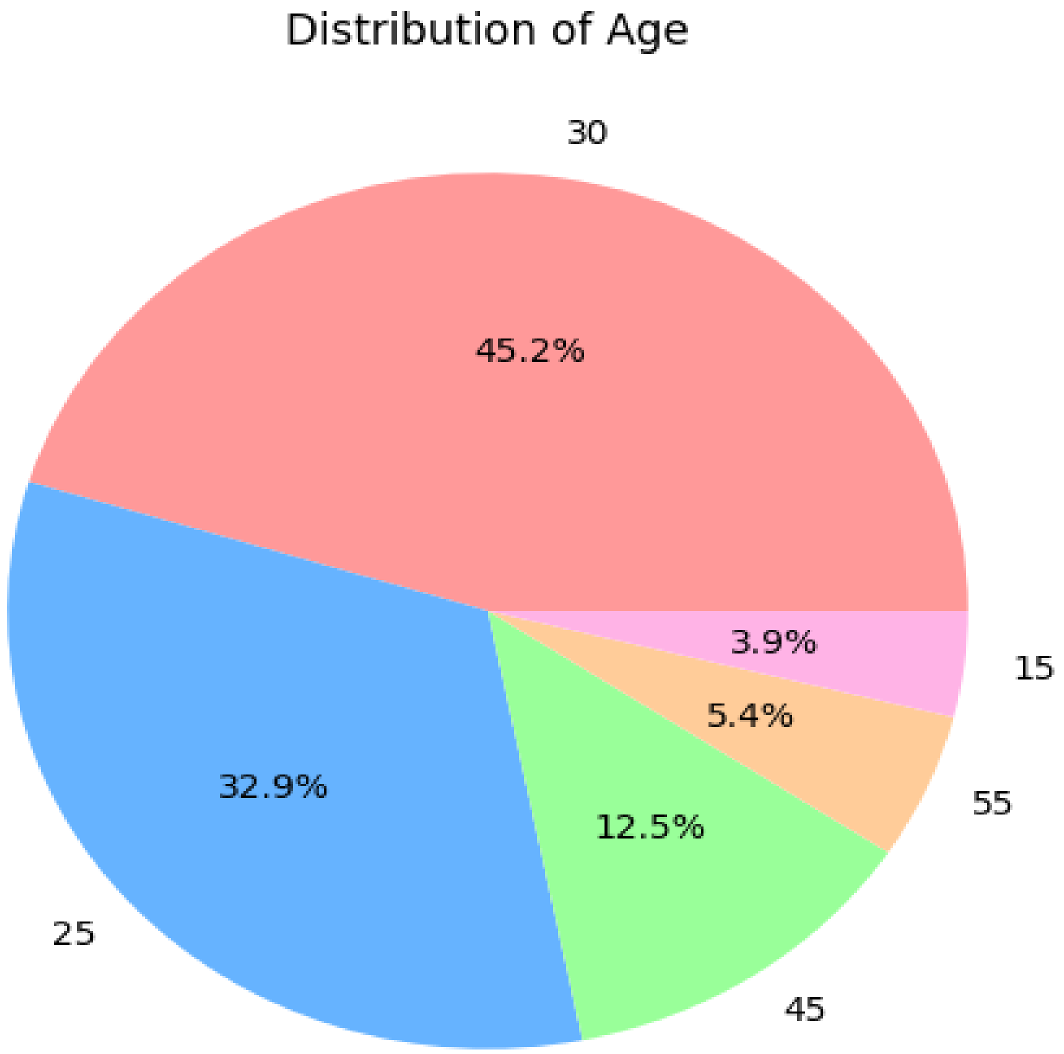




- This graph shows how often different charge amounts occur.
- Most charges are small (around 0), and the number of charges decreases as the amount gets larger.

- This graph shows how often customers use a service.
- Most customers use the service a small number of times (around 0-50), with the number of customers decreasing as the frequency of use increases.

Pie Chart



- This graph shows the distribution of ages.
- Most people are 30 years old, followed by 25, 45, 55, and 15.

Multicollinearity Check

Variables with the greatest variance inflation factor (VIF > 4) were removed

	variables	VIF
0	Call Failure	6.1
1	Complains	1.2
2	Subscription Length	15.1
3	Charge Amount	4.4
4	Seconds of Use	38.1
5	Frequency of use	44.2
6	Frequency of SMS	47.9
7	Distinct Called Numbers	7.0
8	Tariff Plan	15.5
9	Status	7.0
10	Age	16.6
11	Customer Value	77.0

Customer Value was removed

	variables	VIF
0	Call Failure	6.0
1	Complains	1.2
2	Subscription Length	13.3
3	Charge Amount	4.2
4	Seconds of Use	29.1
5	Frequency of use	44.2
6	Frequency of SMS	1.7
7	Distinct Called Numbers	7.0
8	Tariff Plan	15.5
9	Status	6.6
10	Age	12.4

Frequency of use was removed

	variables	VIF
0	Call Failure	4.4
1	Complains	1.2
2	Subscription Length	13.1
3	Charge Amount	3.0
4	Seconds of Use	4.7
5	Frequency of SMS	1.7
6	Distinct Called Numbers	6.0
7	Tariff Plan	14.1
8	Status	6.1
9	Age	12.2

Tariff Plan was removed

Multicollinearity Check

Variables with the greatest variance inflation factor (VIF > 4) were removed

	variables	VIF
0	Call Failure	4.4
1	Complains	1.2
2	Subscription Length	9.6
3	Charge Amount	2.5
4	Seconds of Use	4.7
5	Frequency of SMS	1.6
6	Distinct Called Numbers	6.0
7	Status	5.7
8	Age	9.1

Subscription Length was removed

	variables	VIF
0	Call Failure	4.1
1	Complains	1.2
2	Charge Amount	2.3
3	Seconds of Use	4.7
4	Frequency of SMS	1.6
5	Distinct Called Numbers	5.9
6	Status	5.7
7	Age	4.1

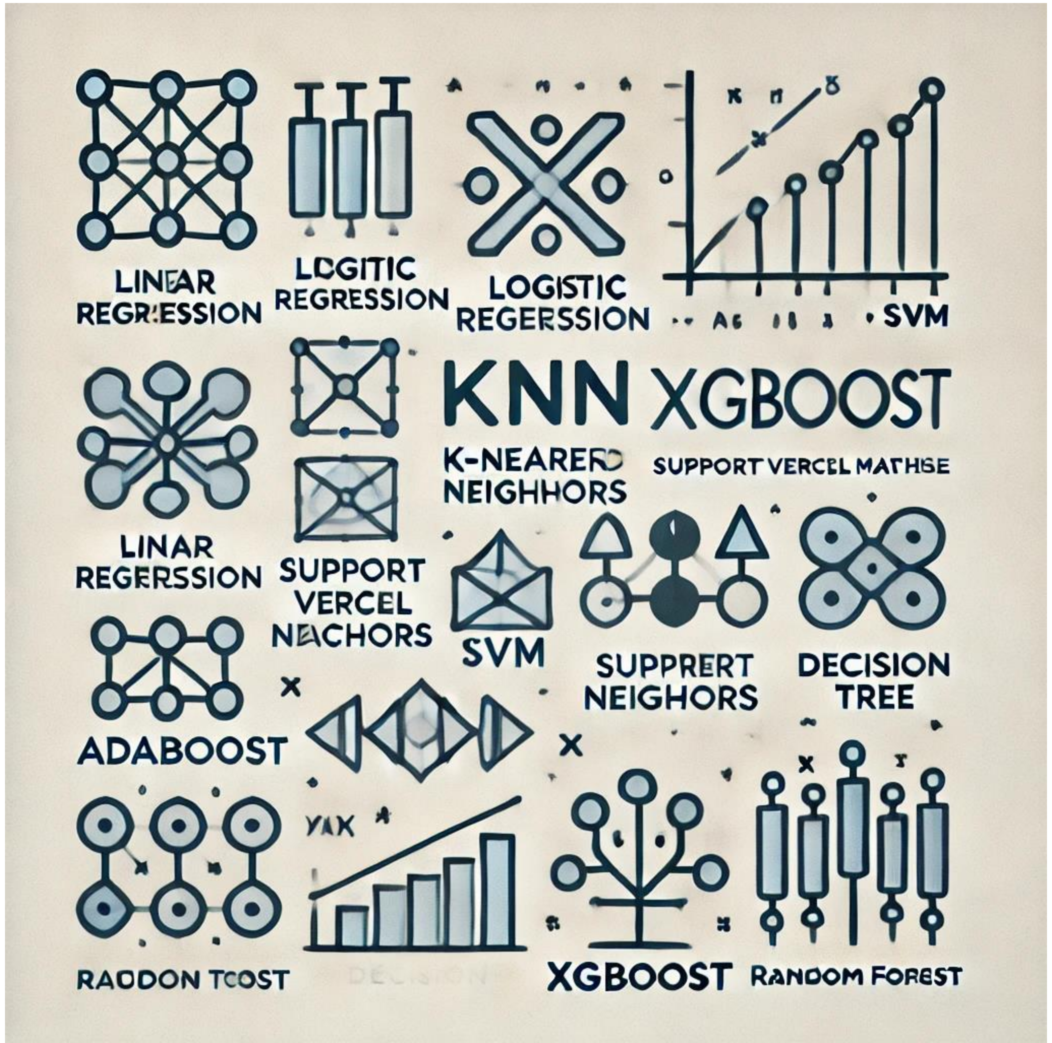
Distinct Called Numbers was removed

	variables	VIF
0	Call Failure	3.7
1	Complains	1.2
2	Charge Amount	2.3
3	Seconds of Use	3.7
4	Frequency of SMS	1.6
5	Status	5.3
6	Age	4.0

Status 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

Machine Learning Algorithms



ML Algorithms

- Logistic regression
- K-Nearest Neighbors(KNN)
- Decision Tree
- Random Forest
- AdaBoost
- XGBoost
- Support Vector Machine(SVM)
- ANN

Logistic Regression

Train test	Model – 1 Accuracy	Model – 2 Accuracy
60-40	0.842	0.892
65-35	0.842	0.877
70-30	0.833	0.891
75-25	0.823	0.885
80-20	0.831	0.880

KNN

Train test	Model – 1 Accuracy	Model – 2 Accuracy
60-40	0.849	0.850
65-35	0.848	0.850
70-30	0.838	0.842
75-25	0.831	0.833
80-20	0.839	0.841

Decision Tree

Train test	Model – 1 Accuracy	Model – 2 Accuracy
60-40	0.933	0.910
65-35	0.932	0.909
70-30	0.925	0.893
75-25	0.932	0.911
80-20	0.936	0.907

Random Forest

Train test	Model – 1 Accuracy	Model – 2 Accuracy
60-40	94	92
65-35	93	92
70-30	94	91
75-25	94	91
80-20	93	91

Adaboost

Train Test	**Model – 1 Accuracy**	**Model – 2 Accuracy**
60-40	93	95
65-35	92	90
70-30	92	89
75-25	91	89
80-20	92	89

Extreme Gradient Boosting

Train Test	**Model – 1 Accuracy**	**Model – 2 Accuracy**
60-40	95	92
65-35	95	92
70-30	95	92
75-25	95	92
80-20	94	92

SVM

Train test	Model – 1 Accuracy	Model – 2 Accuracy
60-40	0.892	0.885
65-35	0.892	0.884
70-30	0.888	0.882
75-25	0.888	0.880
80-20	0.871	0.861

ANN

Train test	Architecture	Epochs	Model – 1 Accuracy	Model-2 Accuracy
60-40	50-45-34-24-1	350	90.54	84.80
60-40	53-47-43-36-1	250	87.41	85.28
60-40	55-45-35-25-1	450	89.37	85.68
65-35	40-40-30-17-1	100	89.20	83.39
65-35	60-56-36-27-1	250	92.79	84.67
65-35	62-58-35-17-1	200	90.73	87.81
70-30	54-39-30-17-1	250	93.11	89.60
70-30	64-67-43-37-1	200	91.08	87.89
70-30	56-59-46-17-1	250	93.08	88.77
75-25	64-54-50-37-1	200	93.30	88.20
75-25	69-58-34-27-1	250	92.63	90.18
75-25	56-58-30-27-1	200	92.99	87.99
80-20	57-53-45-36-1	250	90.11	87.55
80-20	57-52-43-38-1	250	90.14	89.03
80-20	59-56-43-36-1	250	88.73	87.03

Algorithms Comparision

Model-1

Algorithms	Accuracy
Logistic Regression	84.2
K-Nearest Neighbors(KNN)	84.9
Decision Tree	93.6
Random Forest	94
Ada Boost	93
XG Boost	95
Support Vector Machine(SVM)	89.2
ANN	93.30

Algorithms Comparision

Model-2

Algorithms	**Accuracy**
Logistic Regression	89.20
K-Nearest Neighbors(KNN)	85
Decision Tree	91.10
Random Forest	92
Ada Boost	95
XG Boost	92
Support Vector Machine(SVM)	86.10
ANN	90.18

SUMMARY

In this project, a 60-40 train-test split yielded the best model performance for AdaBoost and XGBoost, both achieving an accuracy of 95% in Model-1 and Model-2, respectively. Random Forest and ANN also demonstrated strong performance with accuracies above 90% across the splits. Logistic Regression, KNN, and SVM showed moderate accuracy, indicating these algorithms might not be the best fit for the dataset.

These findings suggest that ensemble methods like XGBoost and AdaBoost are highly effective for this task, leveraging their ability to handle complex relationships and variations in data.

Future Scope

Add relevant features (e.g., external influences or behavioral data) to enhance prediction accuracy.

Reduce multicollinearity using PCA and identify/remove outliers to improve model stability.

Optimize parameters like tree count and max depth for Random Forest, boosting rounds for XGBoost, and learning rates for AdaBoost.

Utilize advanced neural network architectures to capture complex relationships, especially with expanded datasets.

Incorporate diverse and larger datasets to improve model generalizability and robustness.

Work Distribution

NAME	WORK DONE
VAISHNAVI SHIVALINGALA	Collecting Data and Performing Data pre-processing
SIDHHARTHA.S	Exploratory Data Analysis
NAGA SRAVANTHI.T	ML Algorithms

Google colab

THANK YOU

Done By:

Naga Sravanthi T

Vaishnavi Shivalingala

S.Siddhartha

APPENDIX

Loading the Dataset

```python
data= pd.read_csv('iranian+churn+dataset (1).zip')
data
```

	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

Null Values

```
data.isna().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

```
data.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
```

Describing the data

	Call Failure	Complains	Subscription Length	Charge Amount	Seconds of Use	Frequency of use	Frequency of SMS	Distinct Called Numbers	Tariff Plan	Status	Age	Customer Value	Churn
count	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000
mean	7.627937	0.076508	32.541905	0.942857	4472.459683	69.460635	73.174921	23.509841	0.922222	0.751746	30.998413	470.972916	0.157143
std	7.263886	0.265851	8.573482	1.521072	4197.908687	57.413308	112.237560	17.217337	0.267864	0.432069	8.831095	517.015433	0.363993
min	0.000000	0.000000	3.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	15.000000	0.000000	0.000000
25%	1.000000	0.000000	30.000000	0.000000	1391.250000	27.000000	6.000000	10.000000	1.000000	1.000000	25.000000	113.801250	0.000000
50%	6.000000	0.000000	35.000000	0.000000	2990.000000	54.000000	21.000000	21.000000	1.000000	1.000000	30.000000	228.480000	0.000000
75%	12.000000	0.000000	38.000000	1.000000	6478.250000	95.000000	87.000000	34.000000	1.000000	1.000000	30.000000	788.388750	0.000000
max	36.000000	1.000000	47.000000	10.000000	17090.000000	255.000000	522.000000	97.000000	1.000000	1.000000	55.000000	2165.280000	1.000000

Dividing the data

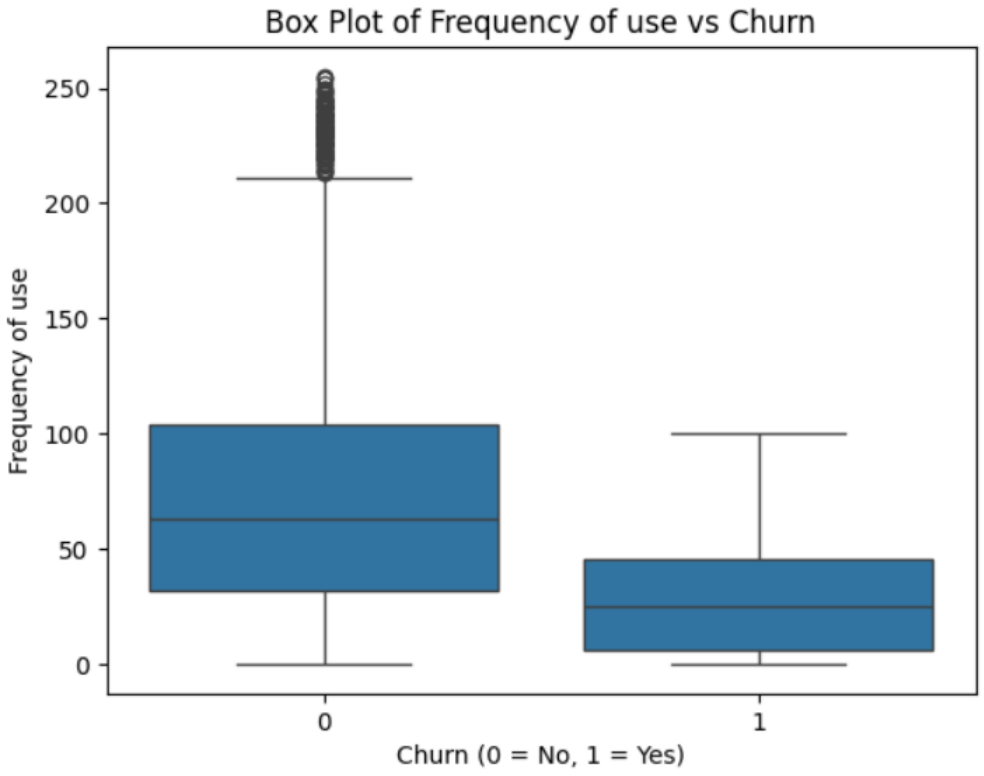
```python
X=data.drop(['Churn'],axis=1)
print(X)
y = data['Churn']
print(y)
```

```
      Call  Failure  Complains  Subscription  Length  Charge  Amount  \
0                8          0                     38               0
1                0          0                     39               0
2               10          0                     37               0
3               10          0                     38               0
4                3          0                     38               0
...            ...        ...                    ...             ...
3145            21          0                     19               2
3146            17          0                     17               1
3147            13          0                     18               4
3148             7          0                     11               2
3149             8          1                     11               2

      Seconds of Use  Frequency of use  Frequency of SMS  \
0               4370                71                 5
1                318                 5                 7
2               2453                60               359
3               4198                66                 1
4               2393                58                 2
...              ...               ...               ...
3145            6697               147                92
3146            9237               177                80
3147            3157                51                38
3148            4695                46               222
```

```
      Distinct Called Numbers  Tariff Plan  Status  Age  Customer Value
0                          17            1       1   30         197.640
1                           4            1       0   25          46.035
2                          24            1       1   30        1536.520
3                          35            1       1   15         240.020
4                          33            1       1   15         145.805
...                       ...          ...     ...  ...             ...
3145                       44            0       1   25         721.980
3146                       42            1       1   55         261.210
3147                       21            1       1   30         280.320
3148                       12            1       1   30        1077.640
3149                        9            1       1   30         100.680

[3150 rows x 12 columns]
0       0
1       0
2       0
3       0
4       0
       ..
3145    0
3146    0
3147    0
3148    0
3149    1
Name: Churn, Length: 3150, dtype: int64
```

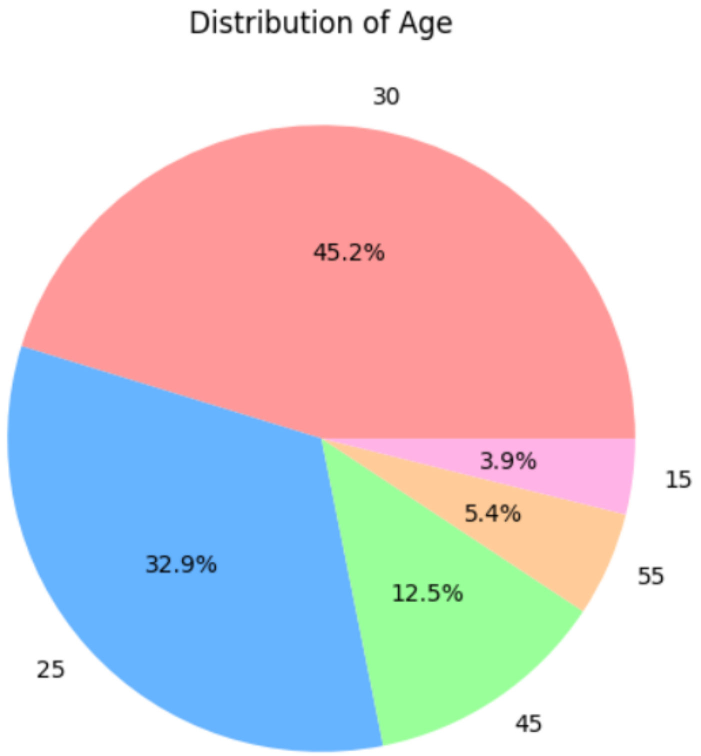
Box Plot

```
sns.boxplot(x='Churn', y='Frequency of use', data=data)
plt.xlabel('Churn (0 = No, 1 = Yes)')
plt.ylabel('Frequency of use')
plt.title('Box Plot of Frequency of use vs Churn')
plt.show()
```



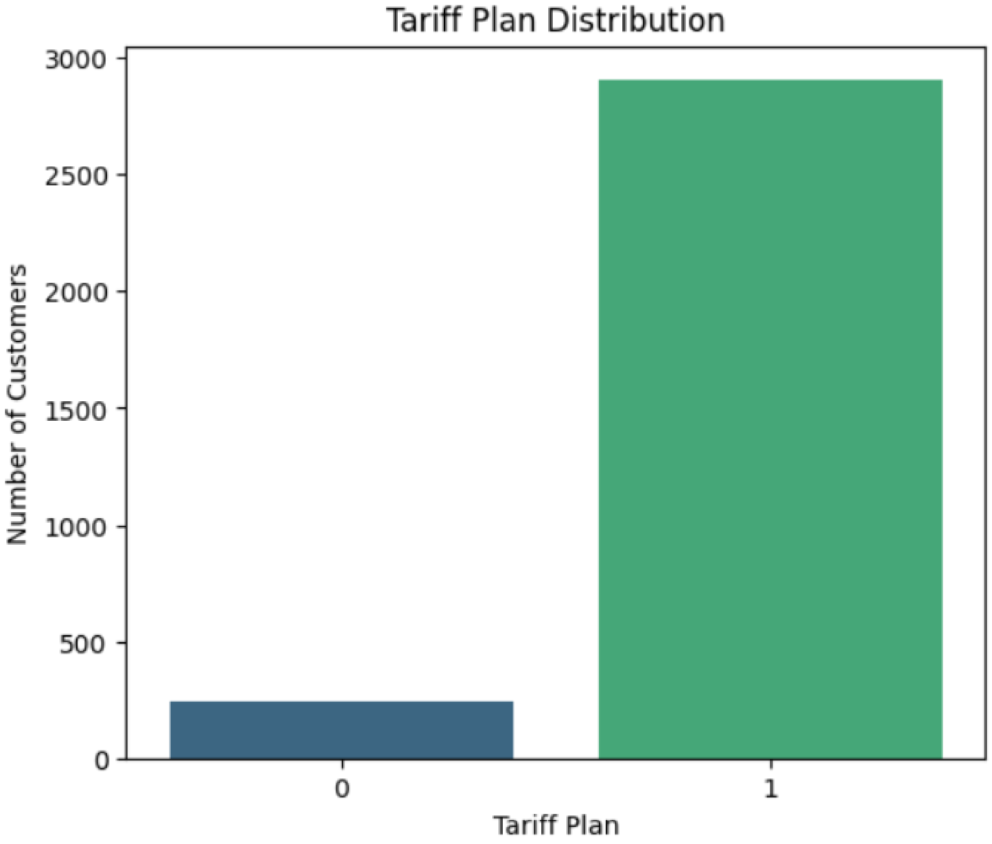
Pie Chart

```
Age_counts = data['Age'].value_counts()
plt.figure(figsize=(6, 6))
plt.pie(Age_counts, labels=Age_counts.index, autopct='%1.1f%%',colors = ['#ff9999','#66b3ff','#99ff99','#ffcc99','#ffb3e6'])
plt.title('Distribution of Age')
plt.show()
```

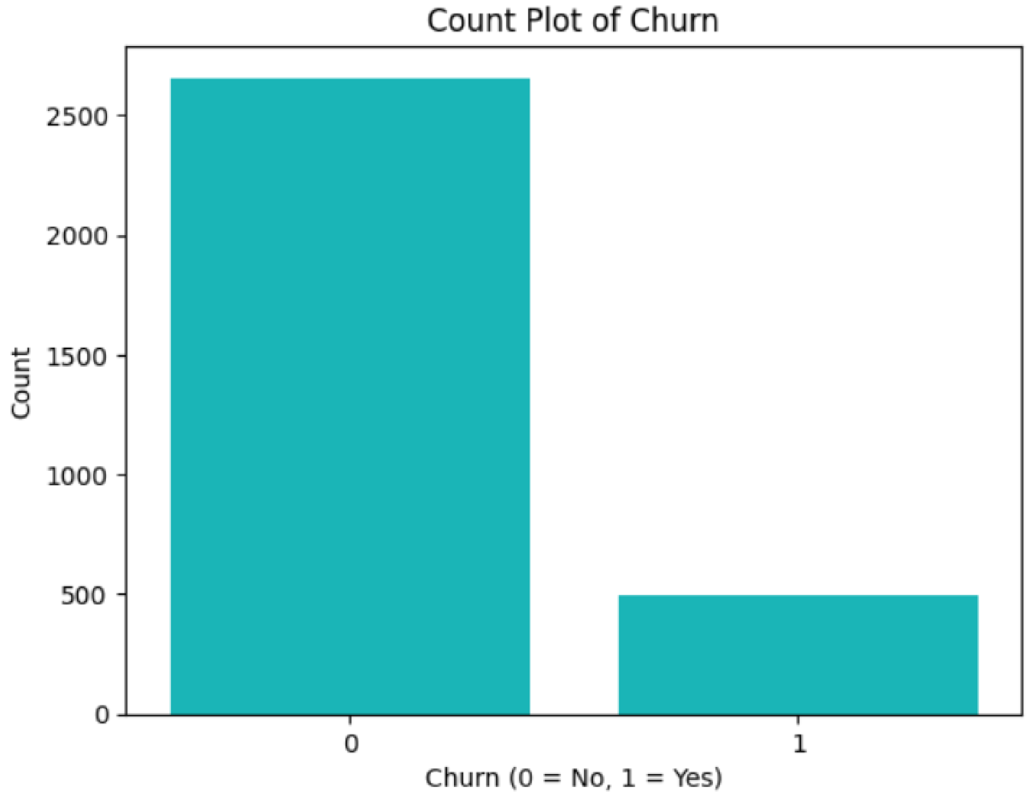


Count Plot

```
plt.figure(figsize=(6,5))
sns.countplot(x='Tariff Plan', data=data, palette='viridis')
plt.title('Tariff Plan Distribution')
plt.xlabel('Tariff Plan')
plt.ylabel('Number of Customers')
plt.show()
```



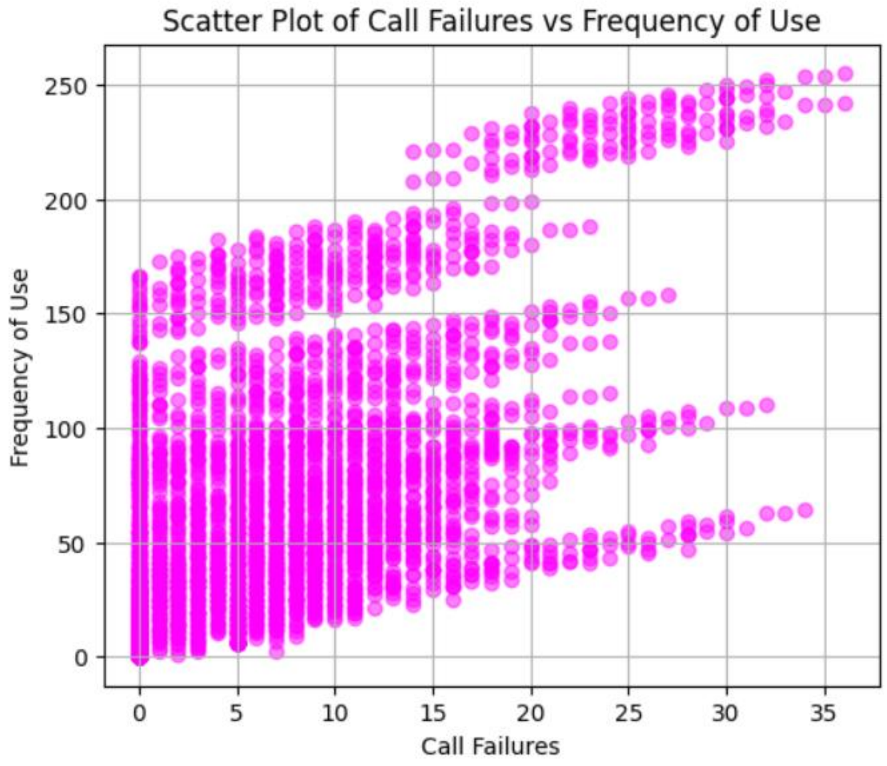

```
sns.countplot(x='Churn', data=data,color='darkturquoise')
plt.xlabel('Churn (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.title('Count Plot of Churn')
plt.show()
```

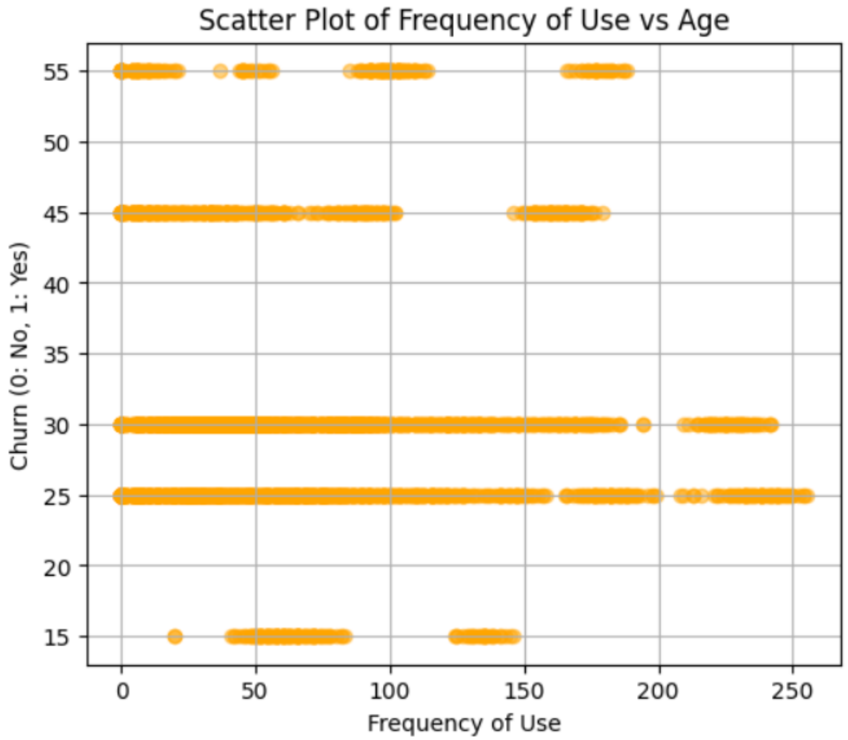


Scatter Plot

```
plt.figure(figsize=(6, 5))
plt.scatter(data['Call  Failure'], data['Frequency of use'], alpha=0.5, color='magenta')
plt.title('Scatter Plot of Call Failures vs Frequency of Use')
plt.xlabel('Call Failures')
plt.ylabel('Frequency of Use')
plt.grid(True)
plt.show()
```

```
plt.figure(figsize=(6,5))
plt.scatter(data['Frequency of use'], data['Age'], alpha=0.5, color='orange')
plt.title('Scatter Plot of Frequency of Use vs Age')
plt.xlabel('Frequency of Use')
plt.ylabel('Churn (0: No, 1: Yes)')
plt.grid(True)
plt.show()
```

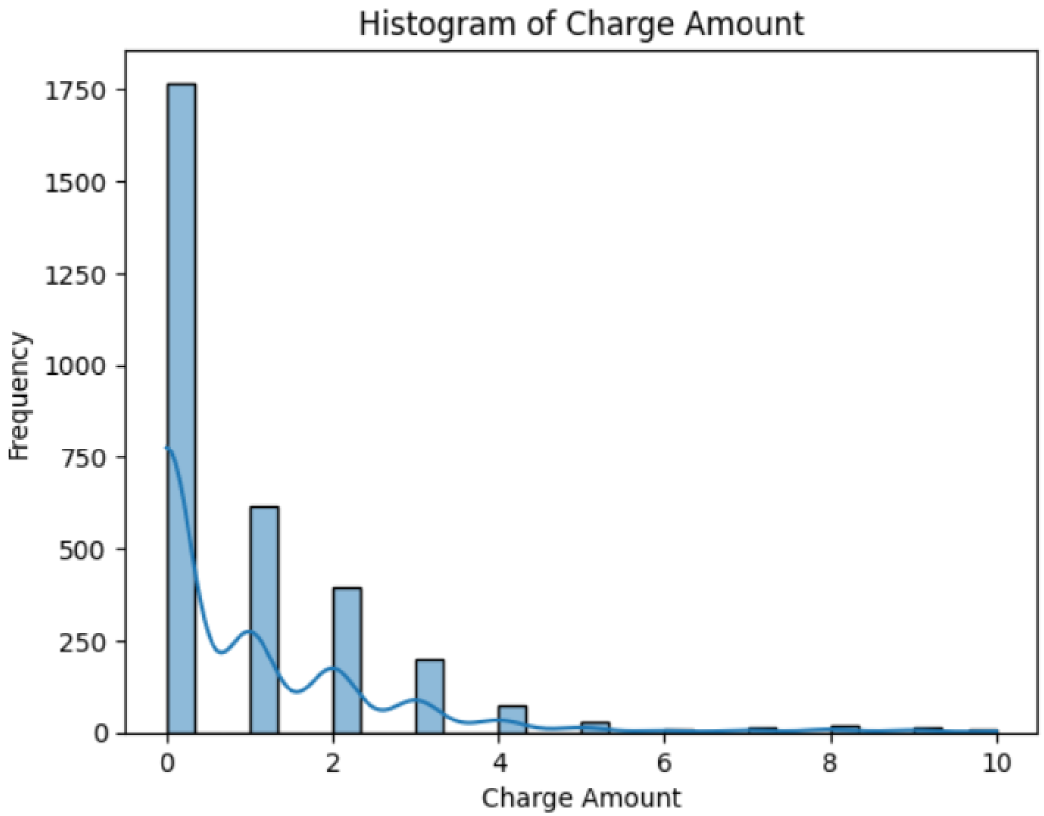


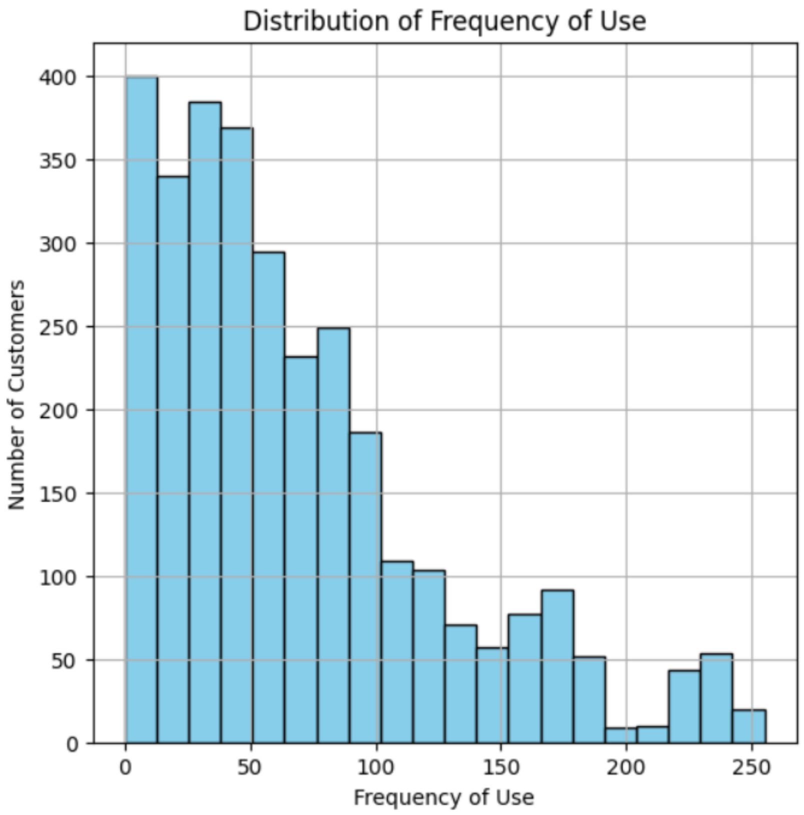


Histogram

```
sns.histplot(data['Charge  Amount'], bins=30, kde=True)
plt.xlabel('Charge Amount')
plt.ylabel('Frequency')
plt.title('Histogram of Charge Amount')
plt.show()
```

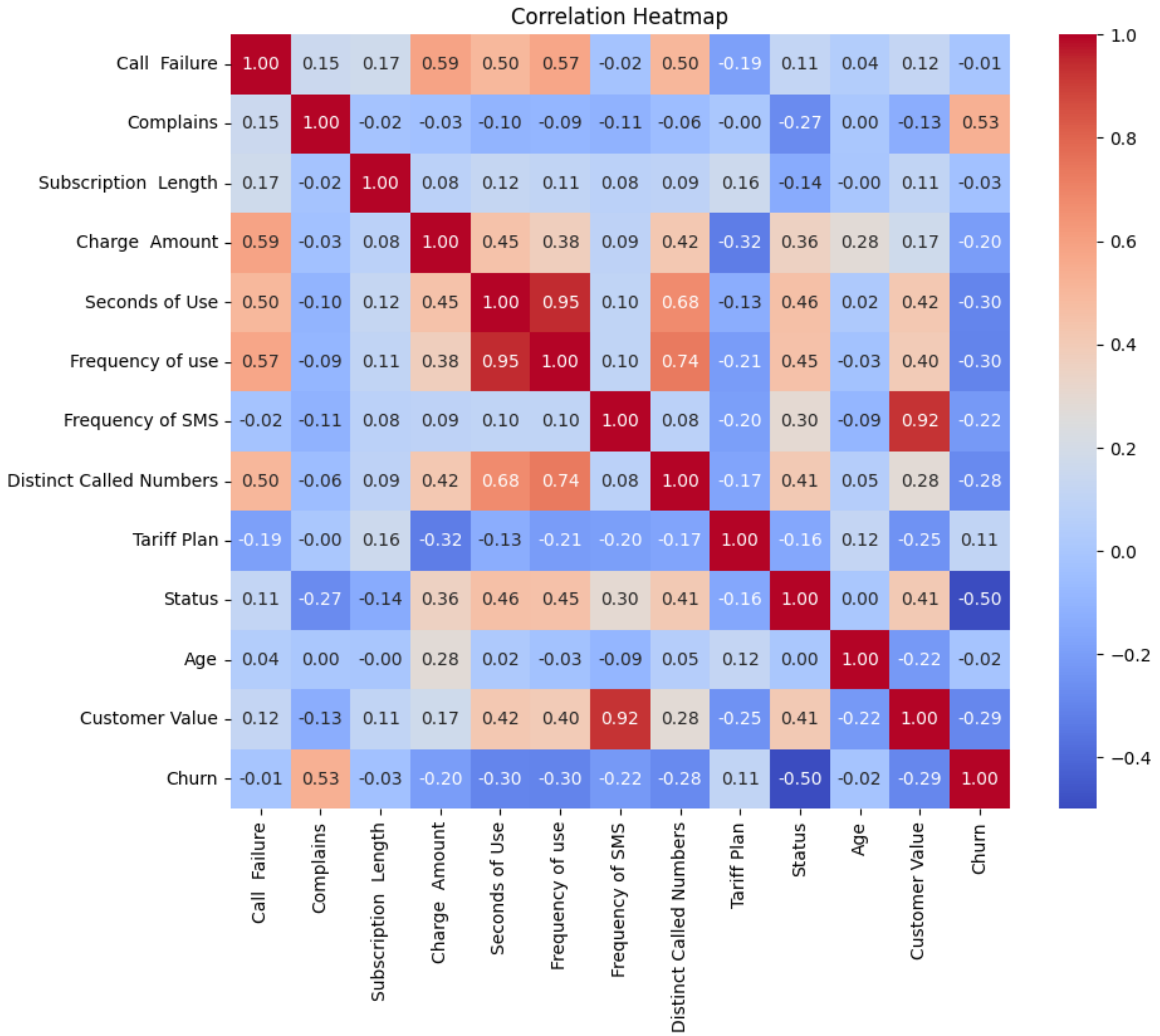
```
plt.figure(figsize=(6,6))
plt.hist(data['Frequency of use'], bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Frequency of Use')
plt.xlabel('Frequency of Use')
plt.ylabel('Number of Customers')
plt.grid(True)
plt.show()
```





Correlation Matrix

```python
corr_matrix = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



Before Multicollinearity

Applying different training and testing data splits:

- 60-40
- 65-35
- 70-30
- 75-25
- 80-20

```
from sklearn.model_selection import train_test_split
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.40, random_state=42)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.35, random_state=42)
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.30, random_state=42)
X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.25, random_state=42)
X_train5, X_test5, y_train5, y_test5 = train_test_split(X, y, test_size=0.20, random_state=42)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(C=1e9)

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

```
[0 0 0 ... 0 0 0]
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
from sklearn.metrics import confusion_matrix
```

```
z=confusion_matrix(y_test1, predictions1)
z
```

```
array([[1026,   29],
       [ 169,   36]])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test1,predictions1)
```

```
0.8428571428571429
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test1,predictions1))
```

```
              precision    recall  f1-score   support

           0       0.86      0.97      0.91      1055
           1       0.55      0.18      0.27       205

    accuracy                           0.84      1260
   macro avg       0.71      0.57      0.59      1260
weighted avg       0.81      0.84      0.81      1260
```

K-nearest neighbors(KNN)

```
from sklearn.neighbors import KNeighborsClassifier
```

```
model=KNeighborsClassifier(n_neighbors=55)
```

```
model.fit(X_train1, y_train1)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=55)
```

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

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
knn = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
knn
```

	Predicted	Actual
2965	0	0
969	0	0
1385	0	0
1233	0	0
2996	0	0
...	...	...
1406	0	0
269	0	0
629	0	0
1033	0	0
286	0	0

1260 rows × 2 columns

Evaluation Metric

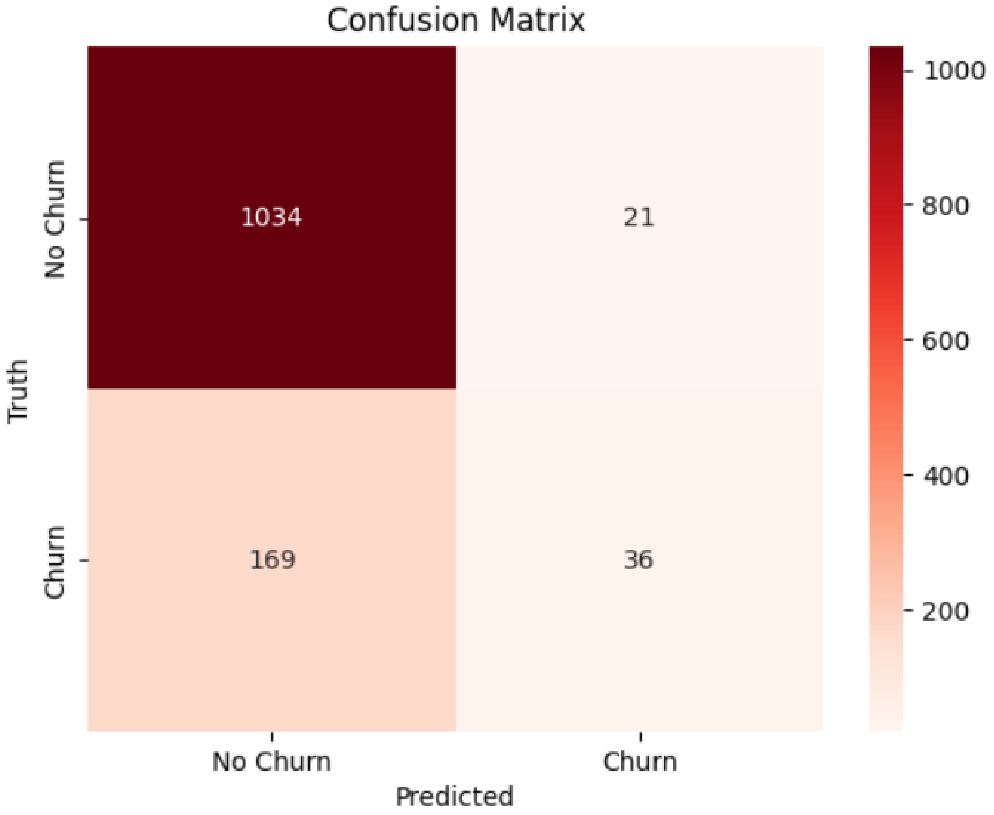
```
[ ] from sklearn.metrics import accuracy_score
    accuracy_score(y_test1,y_pred1)
```

```
0.8492063492063492
```

```
[ ] from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test1,y_pred1)
    cm
```

```
array([[1034,   21],
       [ 169,   36]])
```

```python
sns.heatmap(cm, annot=True, fmt="d", cmap="Reds", xticklabels=["No Churn", "Churn"], yticklabels=["No Churn", "Churn"])
plt.title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```




```python
from sklearn.metrics import classification_report
classification_rep = classification_report(y_test1, y_pred1)
print(classification_rep)
```

```
              precision    recall  f1-score   support

           0       0.86      0.98      0.92      1055
           1       0.63      0.18      0.27       205

    accuracy                           0.85      1260
   macro avg       0.75      0.58      0.60      1260
weighted avg       0.82      0.85      0.81      1260
```

Support Vector Machines(SVM)

```
from sklearn.svm import SVC
```

```
model1 = SVC(kernel='linear')
```

```
model1.fit(X_train1, y_train1)
```

```
SVC
SVC(kernel='linear')
```

```
y_pred1 = model1.predict(X_test1)
```

```
y_pred1
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

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

	Predicted	Actual
2965	0	0
969	0	0
1385	0	0
1233	0	0
2996	0	0
...	...	...
1406	0	0
269	0	0
629	0	0
1033	0	0
286	0	0

1260 rows × 2 columns

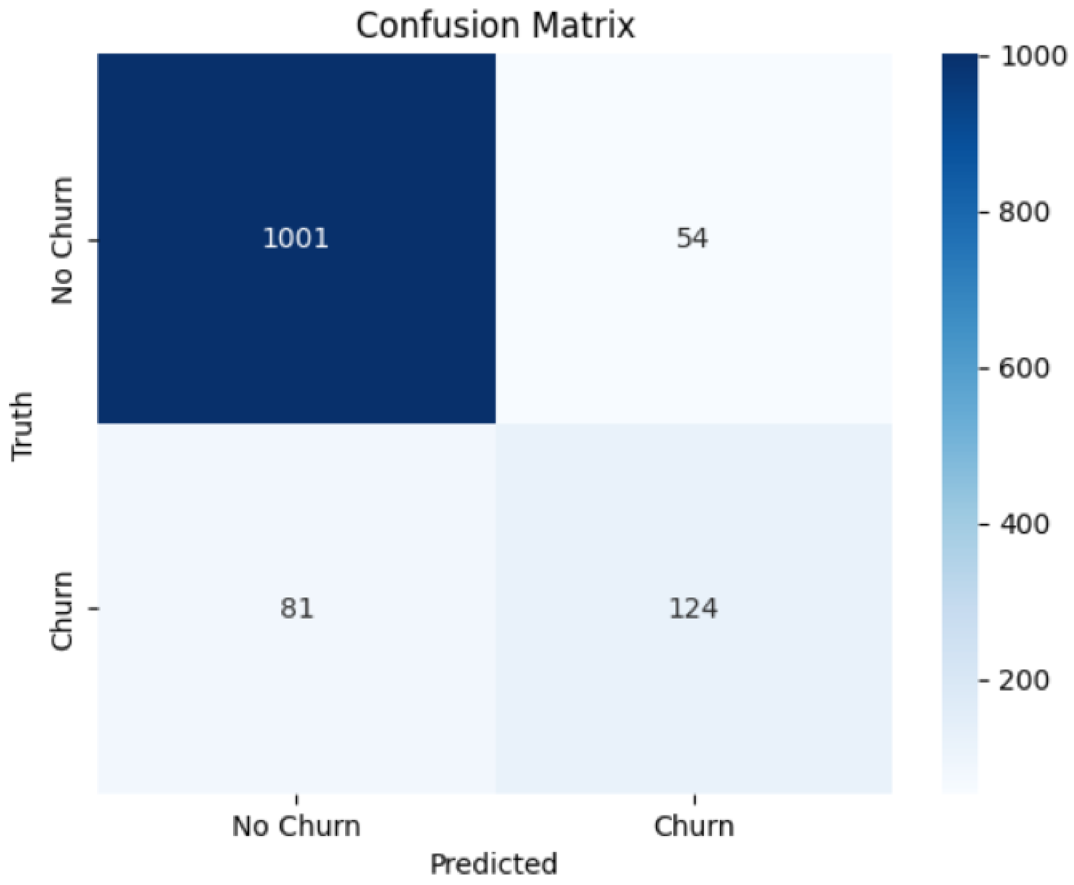
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test1,y_pred1)
```

```
0.8928571428571429
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test1,y_pred1)
cm
```

```
array([[1001,   54],
       [  81,  124]])
```

```
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Churn", "Churn"], yticklabels=["No Churn", "Churn"])
plt.title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```




```
from sklearn.metrics import classification_report
classification_rep = classification_report(y_test1, y_pred1)
print(classification_rep)
```

```
              precision    recall  f1-score   support

           0       0.93      0.95      0.94      1055
           1       0.70      0.60      0.65       205

    accuracy                           0.89      1260
   macro avg       0.81      0.78      0.79      1260
weighted avg       0.89      0.89      0.89      1260
```

Decision Trees

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import metrics
```

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

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

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

```
Accuracy: 0.9341269841269841
```

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train1,y_train1)

#Predict the response for test dataset
y_pred1 = clf.predict(X_test1)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test1, y_pred1))
```

```
Accuracy: 0.8952380952380953
```

```
clf = DecisionTreeClassifier(criterion="gini", max_depth=2)

# Train Decision Tree Classifer
clf = clf.fit(X_train1,y_train1)

#Predict the response for test dataset
y_pred1 = clf.predict(X_test1)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test1, y_pred1))
```

```
Accuracy: 0.8952380952380953
```

```
clf = DecisionTreeClassifier(criterion="gini", max_depth=6)

# Train Decision Tree Classifer
clf = clf.fit(X_train1,y_train1)

#Predict the response for test dataset
y_pred1 = clf.predict(X_test1)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test1, y_pred1))
```

```
Accuracy: 0.9166666666666666
```

```
#Predict the response for train dataset
y_pred_train1 = clf.predict(X_train1)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_train1, y_pred_train1))
```

```
Accuracy: 0.9497354497354498
```

Random Forest

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import plot_tree
from matplotlib import pyplot as plt
```

```
rf = RandomForestClassifier()
```

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

```
▾ RandomForestClassifier
RandomForestClassifier()
```

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

```
print(classification_report(y_test1, y_pred1))
print(confusion_matrix(y_test1, y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.97      1055
           1       0.90      0.76      0.82       205

    accuracy                           0.95      1260
   macro avg       0.92      0.87      0.89      1260
weighted avg       0.94      0.95      0.94      1260

[[1037   18]
 [  50  155]]
```

Adaboost

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report

model = AdaBoostClassifier(n_estimators=50, random_state=42)

model.fit(X_train1, y_train1)

y_pred1 = model.predict(X_test1)

accuracy = accuracy_score(y_test1, y_pred1)
print(f"Accuracy: {accuracy:.2f}")

print(classification_report(y_test1, y_pred1))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6.
  warnings.warn(
Accuracy: 0.93
              precision    recall  f1-score   support

           0       0.95      0.96      0.96      1055
           1       0.80      0.75      0.77       205

    accuracy                           0.93      1260
   macro avg       0.88      0.86      0.87      1260
weighted avg       0.93      0.93      0.93      1260
```

XGBoost

```python
import xgboost as xgb

model = xgb.XGBClassifier()
model = model.fit(X_train1, y_train1)

y_pred1 = model.predict(X_test1)
y_pred1

xg = pd.DataFrame({'Predicted': y_pred1, 'Actual': y_test1})
xg
accuracy1 = accuracy_score(y_test1, y_pred1)
print(f"Set 5 Accuracy: {accuracy1:.2f}")
print(classification_report(y_test1, y_pred1))
```

```
Set 5 Accuracy: 0.95
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1055
           1       0.90      0.78      0.84       205

    accuracy                           0.95      1260
   macro avg       0.93      0.88      0.90      1260
weighted avg       0.95      0.95      0.95      1260
```

ANN

```
import tensorflow as tf
```

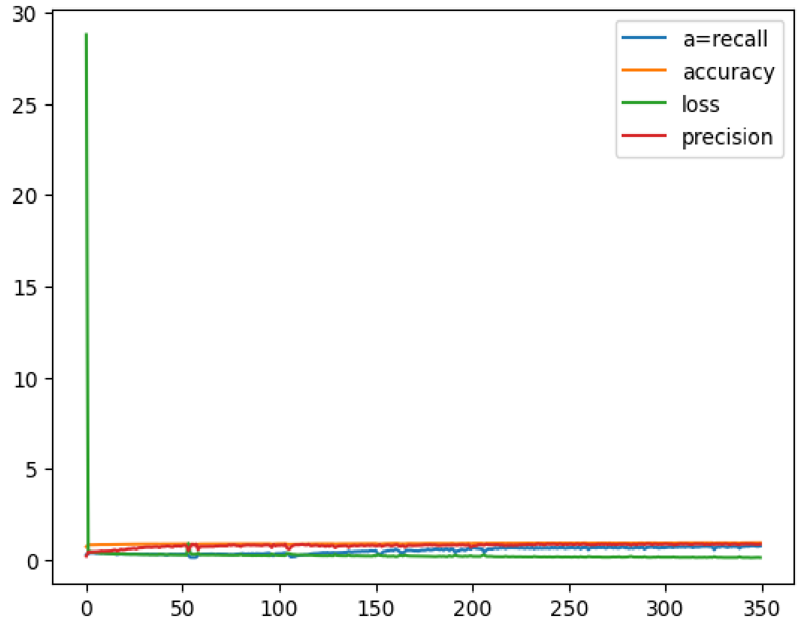
```
tf.random.set_seed(42)

# STEP1: Creating the model

model= tf.keras.Sequential([tf.keras.layers.Dense(50, activation='relu'),
                            tf.keras.layers.Dense(45, activation='relu'),
                            tf.keras.layers.Dense(34, activation='relu'),
                            tf.keras.layers.Dense(24, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP2: Compiling the model

model.compile(loss= tf.keras.losses.binary_crossentropy,
              optimizer= tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                        tf.keras.metrics.Precision(name='precision'),
                        tf.keras.metrics.Recall(name='a=recall')
              ]
              )

# STEP1: Fit the model

history= model.fit(X_train1, y_train1, epochs= 350, verbose=0)
```

```
model.evaluate(X_test1, y_test1)
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - a=recall: 0.6102 - accuracy: 0.9054 - loss: 0.3690 - precision: 0.7867
[0.38202184438705444,
 0.9142857193946838,
 0.7975460290908813,
 0.6341463327407837]
```

```
pd.DataFrame(history.history).plot()
```

```
<Axes: >
```



Multicollinearity Check

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    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)

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

```
calc_vif(X.drop('Customer Value', axis=1))
```

	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

```
calc_vif(X.drop(['Customer Value','Frequency of use'], axis=1))
```

	variables	VIF
0	Call Failure	4.4
1	Complains	1.2
2	Subscription Length	13.1
3	Charge Amount	3.0
4	Seconds of Use	4.7
5	Frequency of SMS	1.7
6	Distinct Called Numbers	5.9
7	Tariff Plan	14.3
8	Status	6.0
9	Age	12.2

```python
calc_vif(X.drop(['Customer Value','Frequency of use','Tariff Plan'], axis=1))
```

	variables	VIF
0	Call Failure	4.4
1	Complains	1.2
2	Subscription Length	9.4
3	Charge Amount	2.5
4	Seconds of Use	4.7
5	Frequency of SMS	1.6
6	Distinct Called Numbers	5.9
7	Status	5.7
8	Age	9.1

```python
calc_vif(X.drop(['Customer Value','Frequency of use','Tariff Plan','Subscription  Length','Distinct Called Numbers'], axis=1))
```

	variables	VIF
0	Call Failure	3.8
1	Complains	1.2
2	Charge Amount	2.3
3	Seconds of Use	3.7
4	Frequency of SMS	1.6
5	Status	5.3
6	Age	4.0

```python
calc_vif(X.drop(['Customer Value','Frequency of use','Tariff Plan','Subscription  Length'], axis=1))
```

	variables	VIF
0	Call Failure	4.1
1	Complains	1.2
2	Charge Amount	2.3
3	Seconds of Use	4.7
4	Frequency of SMS	1.6
5	Distinct Called Numbers	5.9
6	Status	5.6
7	Age	4.1

```python
calc_vif(X.drop(['Customer Value','Frequency of use','Tariff Plan','Subscription  Length','Distinct Called Numbers','Status'], axis=1))
```

	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

After Multicollinearity

Applying different training and testing data splits:

- 60-40
- 65-35
- 70-30
- 75-25
- 80-20

```
X_nomulti = X.drop(['Customer Value','Frequency of use','Tariff Plan','Subscription  Length','Distinct Called Numbers','Status'], axis=1)
```

```
X_train1_nomulti, X_test1_nomulti, y_train1, y_test1 = train_test_split(X_nomulti, y, test_size=0.40, random_state=42)
X_train2_nomulti, X_test2_nomulti, y_train2, y_test2 = train_test_split(X_nomulti, y, test_size=0.35, random_state=42)
X_train3_nomulti, X_test3_nomulti, y_train3, y_test3 = train_test_split(X_nomulti, y, test_size=0.30, random_state=42)
X_train4_nomulti, X_test4_nomulti, y_train4, y_test4 = train_test_split(X_nomulti, y, test_size=0.25, random_state=42)
X_train5_nomulti, X_test5_nomulti, y_train5, y_test5 = train_test_split(X_nomulti, y, test_size=0.20, random_state=42)
```

Logistic Regression:

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(C=1e9)

logreg.fit(X_train_nomulti1, y_train_nomulti1)
predictions1 = logreg.predict(X_test_nomulti1)
print(predictions1)
```

```
[0 0 0 ... 0 0 0]
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: Conve
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
from sklearn.metrics import confusion_matrix
```

```
z=confusion_matrix(y_test_nomulti1, predictions1)
z
```

```
array([[1045,   10],
       [ 125,   80]])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test_nomulti1,predictions1)
```

```
0.8928571428571429
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test_nomulti1,predictions1))
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      1055
           1       0.89      0.39      0.54       205

    accuracy                           0.89      1260
   macro avg       0.89      0.69      0.74      1260
weighted avg       0.89      0.89      0.87      1260
```

K-nearest neighbors(KNN)

```
from sklearn.neighbors import KNeighborsClassifier
```

```
model=KNeighborsClassifier(n_neighbors=55)
```

```
model.fit(X_train_nomulti1, y_train_nomulti1)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=55)
```

```
y_pred_nomulti1 = model.predict(X_test_nomulti1)
y_pred_nomulti1
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
knn = pd.DataFrame({'Predicted':y_pred_nomulti1,'Actual':y_test_nomulti1})
knn
```

	Predicted	Actual
2965	0	0
969	0	0
1385	0	0
1233	0	0
2996	0	0
...	...	...
1406	0	0
269	0	0
629	0	0
1033	0	0
286	0	0

1260 rows × 2 columns

Evaluation Metric

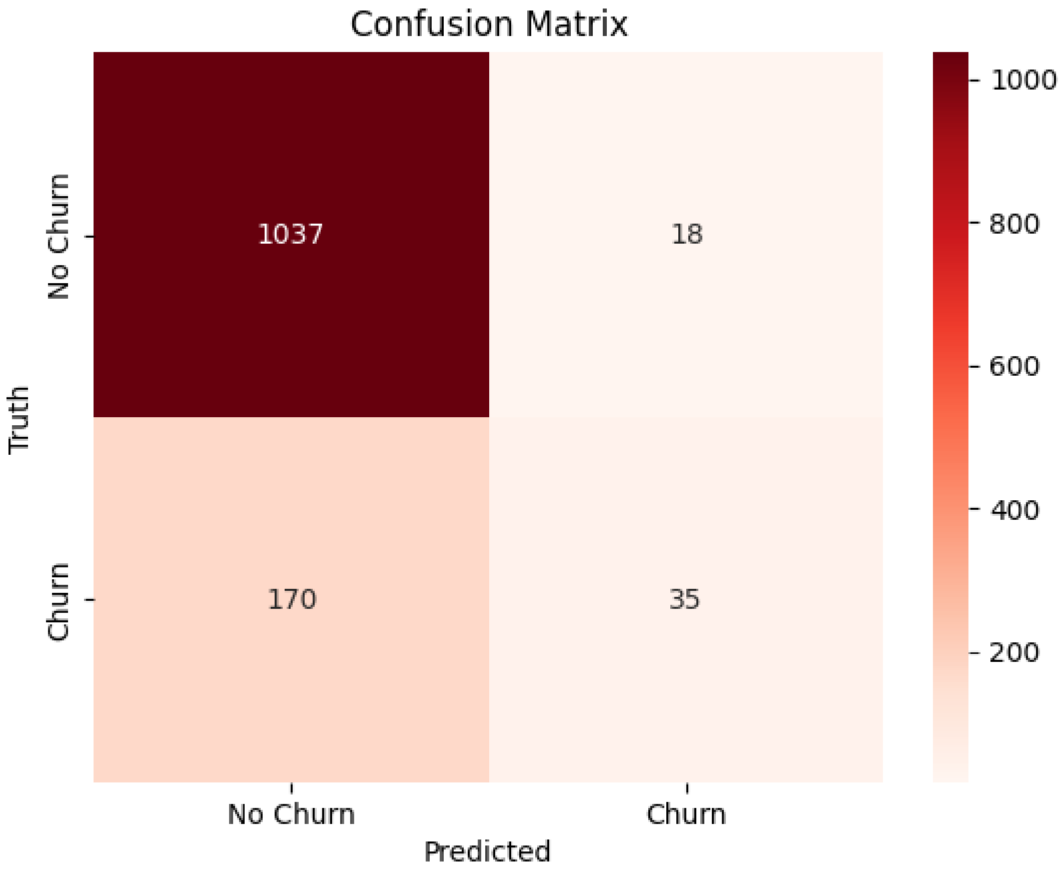
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test_nomulti1,y_pred_nomulti1)
```

```
0.8507936507936508
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test_nomulti1,y_pred_nomulti1)
cm
```

```
array([[1037,   18],
       [ 170,   35]])
```

```
sns.heatmap(cm, annot=True, fmt="d", cmap="Reds", xticklabels=["No Churn", "Churn"], yticklabels=["No Churn", "Churn"])
plt.title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```




```
from sklearn.metrics import classification_report
classification_rep = classification_report(y_test_nomulti1, y_pred_nomulti1)
print(classification_rep)
```

```
              precision    recall  f1-score   support

           0       0.86      0.98      0.92      1055
           1       0.66      0.17      0.27       205

    accuracy                           0.85      1260
   macro avg       0.76      0.58      0.59      1260
weighted avg       0.83      0.85      0.81      1260
```

Support Vector Machines(SVM)

```
[ ] from sklearn.svm import SVC
```

```
model1 = SVC(kernel='linear')
```

```
[ ] model1.fit(X_train_nomulti1, y_train_nomulti1)
```

```
SVC
SVC(kernel='linear')
```

```
[ ] y_pred_nomulti1 = model1.predict(X_test_nomulti1)
```

```
[ ] y_pred_nomulti1
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
svm = pd.DataFrame({'Predicted':y_pred_nomulti1,'Actual':y_test_nomulti1})
svm
```

	Predicted	Actual
2965	0	0
969	0	0
1385	0	0
1233	0	0
2996	0	0
...	...	...
1406	0	0
269	0	0
629	0	0
1033	0	0
286	0	0

1260 rows × 2 columns

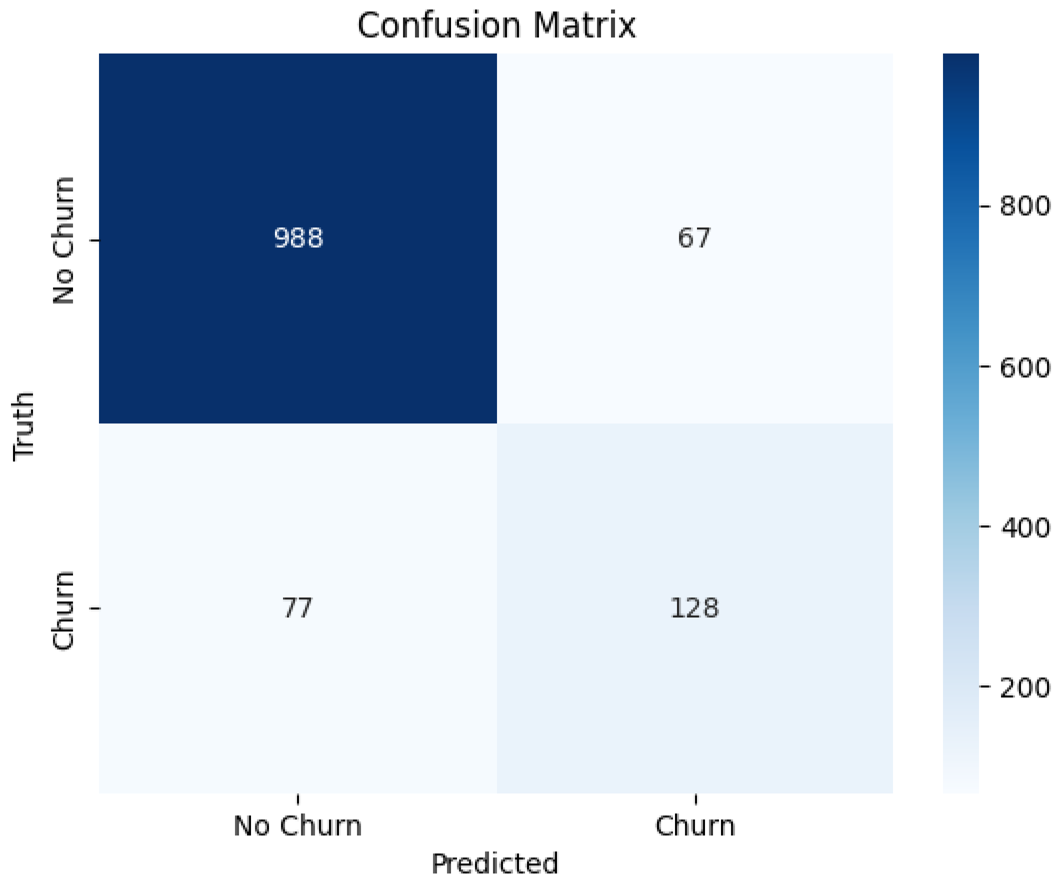
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test_nomulti1,y_pred_nomulti1)
```

```
0.8857142857142857
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test_nomulti1,y_pred_nomulti1)
cm
```

```
array([[988,  67],
       [ 77, 128]])
```

```python
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Churn", "Churn"], yticklabels=["No Churn", "Churn"])
plt.title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```




```python
from sklearn.metrics import classification_report
classification_rep = classification_report(y_test_nomulti1, y_pred_nomulti1)
print(classification_rep)
```

```
              precision    recall  f1-score   support

           0       0.93      0.94      0.93      1055
           1       0.66      0.62      0.64       205

    accuracy                           0.89      1260
   macro avg       0.79      0.78      0.79      1260
weighted avg       0.88      0.89      0.88      1260
```

Decision Trees

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import metrics
```

```
clf = DecisionTreeClassifier()
clf = clf.fit(X_train_nomulti1,y_train_nomulti1)
```

```
y_pred_nomulti1 = clf.predict(X_test_nomulti1)
```

```
print("Accuracy:",metrics.accuracy_score(y_test_nomulti1, y_pred_nomulti1))
```

```
Accuracy: 0.9103174603174603
```

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train_nomulti1,y_train_nomulti1)

#Predict the response for test dataset
y_pred_nomulti1 = clf.predict(X_test_nomulti1)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test_nomulti1, y_pred_nomulti1))
```

```
Accuracy: 0.8984126984126984
```

```
clf = DecisionTreeClassifier(criterion="gini", max_depth=2)

# Train Decision Tree Classifer
clf = clf.fit(X_train_nomulti1,y_train_nomulti1)

#Predict the response for test dataset
y_pred_nomulti1 = clf.predict(X_test_nomulti1)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test_nomulti1, y_pred_nomulti1))
```

```
Accuracy: 0.8952380952380953
```

```
clf = DecisionTreeClassifier(criterion="gini", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train_nomulti1,y_train_nomulti1)

#Predict the response for test dataset
y_pred_nomulti1 = clf.predict(X_test_nomulti1)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test_nomulti1, y_pred_nomulti1))
```

```
Accuracy: 0.8992063492063492
```

```
#Predict the response for train dataset
y_pred_train1 = clf.predict(X_train_nomulti1)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_train_nomulti1, y_pred_train1))
```

```
Accuracy: 0.9216931216931217
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```
rf = RandomForestClassifier()
```

```
rf.fit(X_train_nomulti1, y_train_nomulti1)
```

```
RandomForestClassifier
RandomForestClassifier()
```

```
y_pred_nomulti1 = rf.predict(X_test_nomulti1)
```

```
print(classification_report(y_test_nomulti1, y_pred_nomulti1))
print(confusion_matrix(y_test_nomulti1, y_pred_nomulti1))
```

```
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      1055
           1       0.85      0.60      0.70       205

    accuracy                           0.92      1260
   macro avg       0.89      0.79      0.83      1260
weighted avg       0.91      0.92      0.91      1260

[[1033   22]
 [  82  123]]
```

Adaboost

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
model = AdaBoostClassifier(n_estimators=50, random_state=42)
```

```
model.fit(X_train_nomulti1, y_train_nomulti1)
y_pred_nomulti1 = model.predict(X_test_nomulti1)
accuracy1 = accuracy_score(y_test_nomulti1, y_pred_nomulti1)
print(f"Set 1 Accuracy: {accuracy1:.2f}")
print(classification_report(y_test1, y_pred1))
```

```
Set 1 Accuracy: 0.89
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1055
           1       0.90      0.78      0.84       205

    accuracy                           0.95      1260
   macro avg       0.93      0.88      0.90      1260
weighted avg       0.95      0.95      0.95      1260

/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algo
  warnings.warn(
```

XGBoost

```python
import xgboost as xgb

model = xgb.XGBClassifier()
model = model.fit(X_train_nomulti1, y_train_nomulti1)

y_pred_nomulti1 = model.predict(X_test_nomulti1)
y_pred_nomulti1

xg = pd.DataFrame({'Predicted': y_pred_nomulti1, 'Actual': y_test_nomulti1})
xg
accuracy1 = accuracy_score(y_test_nomulti1, y_pred_nomulti1)
print(f"Set 5 Accuracy: {accuracy1:.2f}")
print(classification_report(y_test_nomulti1, y_pred_nomulti1))
```

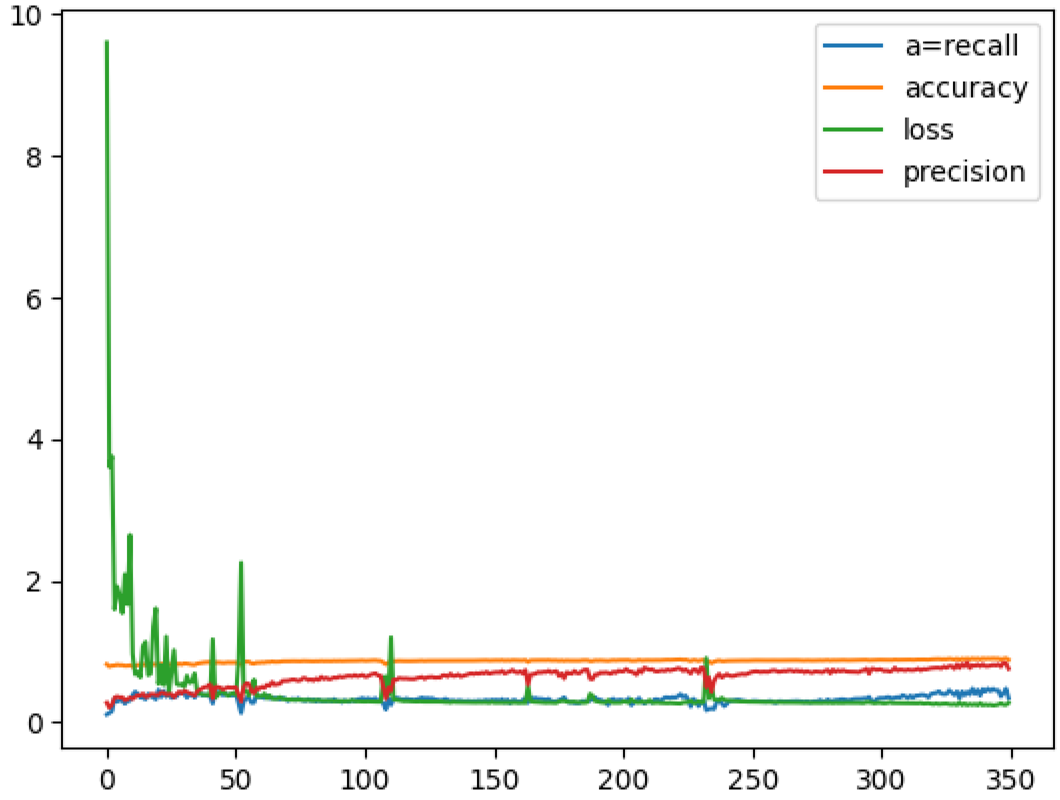
```
Set 5 Accuracy: 0.92
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      1055
           1       0.85      0.62      0.72       205

    accuracy                           0.92      1260
   macro avg       0.89      0.80      0.84      1260
weighted avg       0.92      0.92      0.92      1260
```

ANN

```
tf.random.set_seed(42)

# STEP1: Creating the model

model= tf.keras.Sequential([tf.keras.layers.Dense(50, activation='relu'),
                            tf.keras.layers.Dense(45, activation='relu'),
                            tf.keras.layers.Dense(34, activation='relu'),
                            tf.keras.layers.Dense(24, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP2: Compiling the model

model.compile(loss= tf.keras.losses.binary_crossentropy,
              optimizer= tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                        tf.keras.metrics.Precision(name='precision'),
                        tf.keras.metrics.Recall(name='a=recall')
              ]
              )

# STEP1: Fit the model

history= model.fit(X_train_nomulti1, y_train_nomulti1, epochs= 350, verbose=0)
```

```
pd.DataFrame(history.history).plot()
```

<Axes: >




```
model.evaluate(X_test_nomulti1, y_test_nomulti1)
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - a=recall: 0.2000 - accuracy: 0.8485 - loss: 0.3578 - precision: 0.6839
[0.3288106918334961, 0.8595238327980042, 0.75, 0.2048780471086502]
```