University of Messina

MACHINE LEARNING (Project Report)

BACHELORS IN DATA ANALYSIS (23/24)

(MIFT DEPARTMENT)

Supervisioned by – Prof. Giacomo Fiumara

Student – Vaniprasad Vemuri

INDEX

INTRODUCTION

Customer churn is a major issue for businesses, as losing customers can impact their growth and profitability. By using machine learning to predict when customers are likely to leave, companies can take proactive steps to improve customer satisfaction and retention. This helps businesses develop better strategies to keep their customers and reduce the risk of losing them.

UNDERSTANDING THE DATASET

The main goal of this project is to analyze and identify the factors that contribute to customer churn. To start, we import all the essential libraries that will assist in carrying out the analysis effectively.

Load the Dataset

```
#Load the Dataset
df = pd.read_csv("D:\Data\Churn Dataset fr ML.csv")
df
```

Given Dataset –

It ensures that the dataset is loaded for the upcoming actions and approaches to be performed on the loaded dataset.

```
#Load the Dataset
df = pd.read_csv("D:\Data\Churn Dataset fr ML.csv")
df
```

	Unnamed: 0	CustomerID	Age	Gender	Tenure	Service_Internet	Service_Phone	Service_TV	Contract	PaymentMethod	MonthlyCharges	TotalCharges
0	0	08729464-bde6-43bc-8f63-a357096feab1	56.0	Male	13	DSL	Yes	No	One year	Mailed check	71.88	931.49
1	1	af95bc95-baf4-4318-a21d-70d2ea3148b7	69.0	Male	13	DSL	No	Yes	Two year	Mailed check	110.99	1448.46
2	2	1fe7eee6-2227-4400-9998-4d993f4a60fd	46.0	Male	60	Fiber optic	No	Yes	Month-to-month	Mailed check	116.74	6997.73
3	3	f736fe7b-1b44-4acd-84c2-21c4aef648be	32.0	Female	57	Fiber optic	Yes	Yes	Month-to-month	Bank transfer	78.16	4452.13
4	4	4b40d12d-7633-4309-96b8-aee675ea20ae	60.0	Male	52	Fiber optic	Yes	Yes	Two year	Electronic check	30.33	1569.73
...	...	...	...	...	...	...	...	...	...	...	...	...
3744	3744	7d6b54f3-020a-4606-ac4e-a4c2dd3f1ac8	61.0	Male	59	Fiber optic	Yes	No	One year	Electronic check	61.14	3608.50
3745	3745	b65230c5-d1bf-4789-aefe-7244c6c5c153	36.0	Female	52	DSL	Yes	No	Month-to-month	Electronic check	34.15	1784.38
3746	3746	98313047-1f41-4f20-8f8d-f36bfde0ca1b	29.0	Male	19	NaN	Yes	No	Month-to-month	Credit card	30.79	594.41
3747	3747	e2fb6ab0-61b4-445f-b798-99f809c8eefa	25.0	Male	21	DSL	Yes	No	Month-to-month	Mailed check	80.56	1715.08
3748	3748	a6c038b7-b5c4-4813-9705-6a000a6a7d7f	NaN	Male	22	NaN	Yes	No	One year	Electronic check	102.82	2262.98

3749 rows × 17 columns

Dataset Information

```
#Information of Dataset
df.info()
```

Prints a concise overview of the DataFrame.

Provides details like:

- Number of rows and columns
- Index data type
- Column labels and their data types
- Number of non-null values in each column (to identify missing data)
- Memory usage of the DataFrame

```
#Information of Dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3749 entries, 0 to 3748
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Unnamed: 0        3749 non-null   int64  
 1   CustomerID        3749 non-null   object 
 2   Age               3562 non-null   float64
 3   Gender            3749 non-null   object 
 4   Tenure            3749 non-null   int64  
 5   Service_Internet  3028 non-null   object 
 6   Service_Phone     3749 non-null   object 
 7   Service_TV        3749 non-null   object 
 8   Contract          3749 non-null   object 
 9   PaymentMethod     3562 non-null   object 
 10  MonthlyCharges    3749 non-null   float64
 11  TotalCharges      3749 non-null   float64
 12  StreamingMovies   3749 non-null   object 
 13  StreamingMusic    3749 non-null   object 
 14  OnlineSecurity    3749 non-null   object 
 15  TechSupport       3749 non-null   object 
 16  Churn             3749 non-null   object 
dtypes: float64(3), int64(2), object(12)
memory usage: 498.0+ KB
```

Description of the Dataset(Describing each feature and possible values).

```
#Description of the Dataset
df.describe()
```

This line provides a summary of the main statistical measures of the DataFrame's numerical columns.

df.describe(): This function generates descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset's distribution, excluding NaN values. This thus include **STD**, **mode** , **median** , **mean** etc

```
#Description of the Dataset
df.describe()
```

	Unnamed: 0	Age	Tenure	MonthlyCharges	TotalCharges
count	3749.000000	3562.000000	3749.000000	3749.000000	3749.000000
mean	1874.000000	43.655531	36.264070	75.844318	2718.968266
std	1082.387408	14.914474	20.505528	73.062971	3211.879149
min	0.000000	18.000000	1.000000	20.000000	13.190000
25%	937.000000	31.000000	19.000000	44.570000	1076.240000
50%	1874.000000	44.000000	36.000000	69.590000	2132.260000
75%	2811.000000	56.000000	54.000000	95.540000	3619.710000
max	3748.000000	69.000000	71.000000	1179.300000	79951.800000

Columns and Shape of the Dataset.

Going across all the existing columns in the dataset, retrieve and shows what columns(ex- **Age**, **Tenure** etc) exist and shape implies how many columns and rows (i.e the number of rows and columns) that the dataset contains.

```
In [6]: #Retrieving all columns in the dataset
        df.columns

Out[6]: Index(['Unnamed: 0', 'CustomerID', 'Age', 'Gender', 'Tenure',
               'Service_Internet', 'Service_Phone', 'Service_TV', 'Contract',
               'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'StreamingMovies',
               'StreamingMusic', 'OnlineSecurity', 'TechSupport', 'Churn'],
              dtype='object')

In [7]: #Shape of the Data set (Rows,columns)
        df.shape

Out[7]: (3749, 17)
```

Check for missing values (if any)

- It shows three of the columns containing missing values **'Age'**, '**Service_Internet',PaymentMethod** respectively.
- We check for missing values in this case because missing values might indicate potential issues with data collection or storage. By identifying them and their patterns will help us understand the quality and reliability of our dataset and also many machine learning algorithms cannot handle missing data directly. They might require complete information for each datapoint to function correctly.
- Hence we do the Imputation for missing values in the data. Dropping the missing values from the dataset might potentially impact the reliability for our analysis , because there will be so much of data loss.

```
#Checking for the missing Values in the Dataset
df.isnull().sum()
```

```
Unnamed: 0           0
CustomerID           0
Age                187
Gender               0
Tenure               0
Service_Internet   721
Service_Phone        0
Service_TV           0
Contract             0
PaymentMethod      187
MonthlyCharges       0
TotalCharges         0
StreamingMovies      0
StreamingMusic       0
OnlineSecurity       0
TechSupport          0
Churn                0
dtype: int64
```

- Shows that in the certain columns(i.e **Age**, **Service_Internet**, **PaymentMethod**) have missing values , for which I used imputation in the next step to handle them.

DATA PREPROCESSING

Handling the missing values by imputation and removing unnecessary column (in the columns and cleaning the dataset)

- Initially I handled the missing values in the dataset in specific columns using Imputation procedure. Coming to the columns with missing values , the **'Age'** column (numerical type data) I replaced the missing values using the median age of the column. And for the other columns with missing values **'Service_Internet'** and **'PaymentMethod'** columns , I fill the missing values with the most frequent category(i.e mode), ensuring most common value is used as replacement.

- After I perform these imputations, I made sure that there are no missing values in the dataset
- Additionally I also did the dropping of the unnecessary column, **'Unnamed:0'** as it was redundant.and made sure data is cleant , as it also doesn't provide such meaningful information.
- Finally the imputations and cleaning I did , was saved it into new csv file

```
# IMPUTATION OF Missing Values AND Drop of a column(Data Cleaning)

# Dropping the 'Unnamed: 0' column
df = df.drop(columns=['Unnamed: 0'])

# Imputation of missing values in 'Age' with the median
df['Age'] = df['Age'].fillna(df['Age'].median())

# Imputation of missing values in 'Service_Internet' with mode(most frequent)
most_frequent_service_internet = df['Service_Internet'].mode()[0]
df['Service_Internet'] = df['Service_Internet'].fillna(most_frequent_service_internet)

# Imputation missing values in 'PaymentMethod' with mode(most frequent)
most_frequent_payment_method = df['PaymentMethod'].mode()[0]
df['PaymentMethod'] = df['PaymentMethod'].fillna(most_frequent_payment_method)

# Verifying that there are no more missing values
missing_values_after_inputation = df.isnull().sum()
print(missing_values_after_inputation)

# Save the updated DataFrame to a new CSV file
df.to_csv('MLImputedDataset.csv', index=False)
```

```
CustomerID          0
Age                 0
Gender              0
Tenure              0
Service_Internet    0
Service_Phone       0
Service_TV          0
Contract            0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
StreamingMovies     0
StreamingMusic      0
OnlineSecurity      0
TechSupport         0
Churn               0
dtype: int64
```

Verifying the Dataframe Structure After Handling Missing Values.

```
In [10]: # Verifying the dataframe structure after dropping the column and imputation of missing values
df.head(30)
```

Out[10]:

	CustomerID	Age	Gender	Tenure	Service_Internet	Service_Phone	Service_TV	Contract	PaymentMethod	MonthlyCharges	TotalCharges	Streaming
0	08729464-bde6-43bc-8f63-a357096feab1	56.0	Male	13	DSL	Yes	No	One year	Mailed check	71.88	931.49	
1	af95bc95-baf4-4318-a21d-70d2ea3148b7	69.0	Male	13	DSL	No	Yes	Two year	Mailed check	110.99	1448.46	
2	1fe7eee6-2227-4400-9998-4d993f4a60fd	46.0	Male	60	Fiber optic	No	Yes	Month-to-month	Mailed check	116.74	6997.73	
3	f736fe7b-1b44-4acd-84c2-21c4aef648be	32.0	Female	57	Fiber optic	Yes	Yes	Month-to-month	Bank transfer	78.16	4452.13	
	4b40d12d-											

Label Encoding (encoding categorical variables).

- I applied label for categorical variables encoding to transform categorical variables into numerical form.
- This transformation is mainly essential for the machine learning models.
- I used the **LabelEncoder** from the sklearn library to convert all object type – columns except for the target variable Churn into numeric values.

```
In [11]: #Encoding of variables(Label Encoding)
         label_encoders = {}
         for column in df.select_dtypes(include=['object']).columns:
             if column != 'Churn':
                 le = LabelEncoder()
                 df[column] = le.fit_transform(df[column])
                 label_encoders[column] = le
```

- The target variable '**Churn**' was separately encoded, with 0 representing "No" and 1 representing "Yes".
- This transformation is crucial for the machine learning models to correctly interpret the categorical data and make accurate and best predictions.

```
#Encoding the target variable (Churn)
le_churn = LabelEncoder()
df['Churn'] = le_churn.fit_transform(df['Churn'])
```

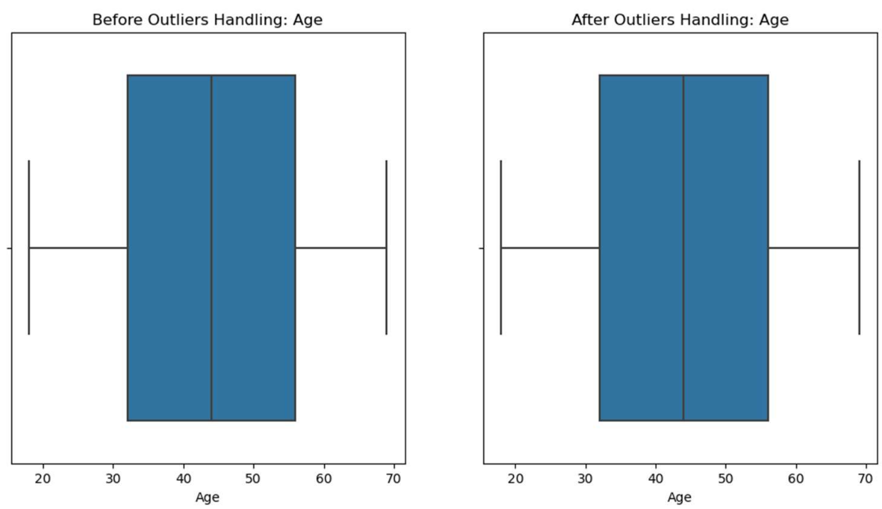
- Later on , using the **df.head()** function , which helps to print the first few rows of the dataset , for the verification if the performed label encoding was successful or not .

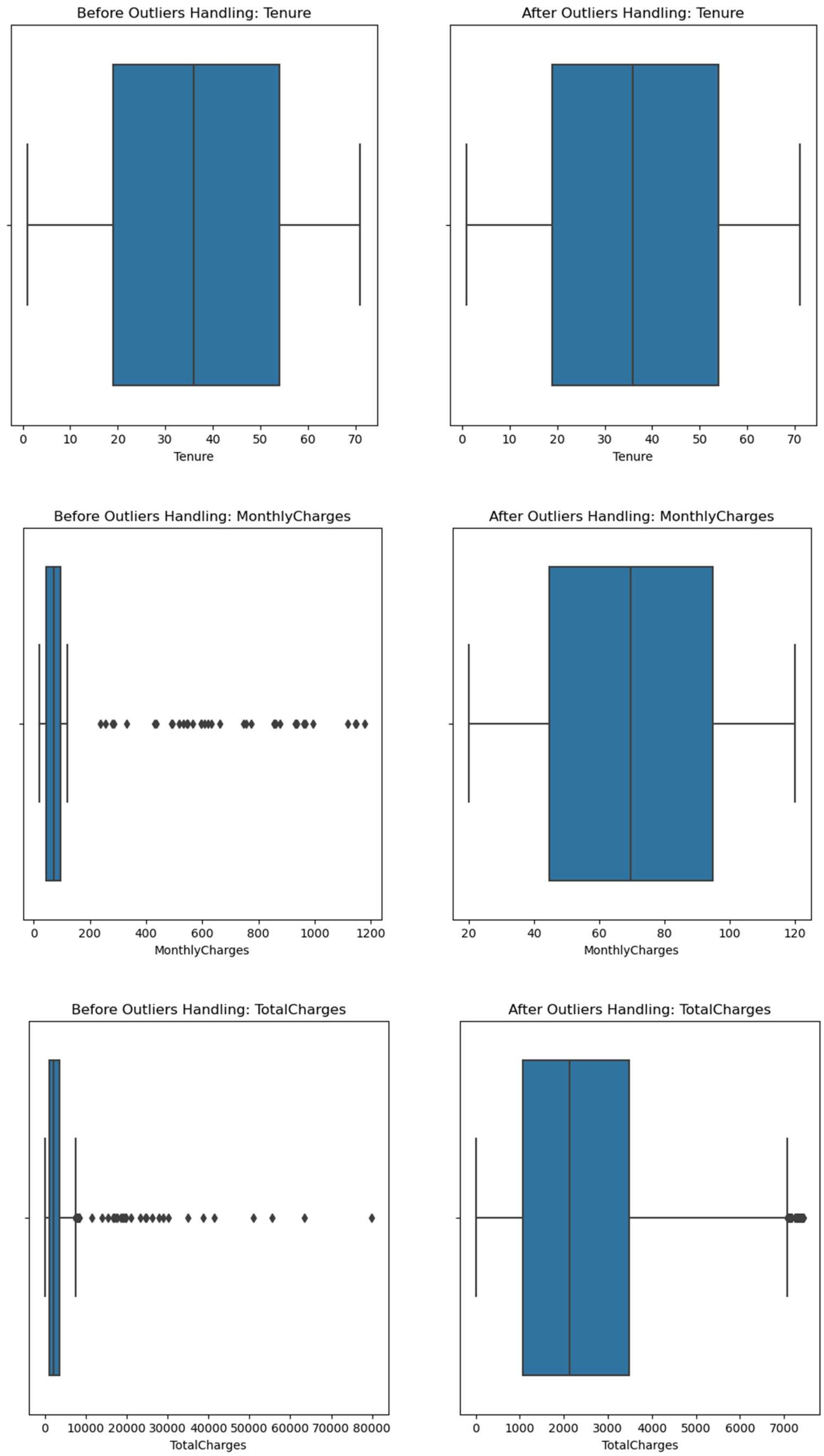
```
In [12]: df.head(10)
Out[12]:
```

	CustomerID	Age	Gender	Tenure	Service_Internet	Service_Phone	Service_TV	Contract	PaymentMethod	MonthlyCharges	TotalCharges	StreamingMovies
0	114	56.0	1	13	0	1	0	1	3	71.88	931.49	0
1	2555	69.0	1	13	0	0	1	2	3	110.99	1448.46	1
2	459	46.0	1	60	1	0	1	0	3	116.74	6997.73	1
3	3609	32.0	0	57	1	1	1	0	0	78.16	4452.13	0
4	1053	60.0	1	52	1	1	1	2	2	30.33	1569.73	1
5	225	25.0	0	53	1	1	1	0	0	77.33	4107.57	1
6	2040	38.0	1	43	1	1	0	2	3	79.76	3437.48	1
7	2643	56.0	1	42	0	0	1	0	0	98.92	4154.29	0
8	575	36.0	0	26	0	1	1	0	2	24.82	662.63	0
9	2344	40.0	1	23	1	1	1	2	3	66.63	1536.62	0

Outliers detection and Treatment of Outliers

- In this analysis , I used an **Interquartile Range(IQR)** method to detect outliers in the dataset. The function **handle_outliers()** was designed to detect and treat outliers in a dataset by replacing them with the median value of the respective column.
- Outliers are the data points that significantly deviate from the majority of data , which can affect the analysis
- The function calculates the **first quartile (Q1)** and **third quartile (Q3)** for the specified column and determines the **Interquartile Range (IQR**) as the difference between Q3 and Q1.
- Values below the lower bound or above the upper bound are replaced with the median value of the column.
- I also created a visual comparison of data of the features before and after treatment of outliers with using the help of the function **plot_outliers_before_after()** . It mainly helps in understanding the impact of Outlier treatment.
- I applied the function to the numerical features which are **'Age'** , **'Tenure'** **'MonthlyCharges'** and **'TotalCharges'** , to understand the outlier treatment performed on the data from the graphical representation (i.e outliers before and after they are handled)




Before Outliers Handling: Tenure
0 10 20 30 40 50 60 70
Tenure
After Outliers Handling: Tenure
0 10 20 30 40 50 60 70
Tenure
Before Outliers Handling: MonthlyCharges
0 200 400 600 800 1000 1200
MonthlyCharges
After Outliers Handling: MonthlyCharges
20 40 60 80 100 120
MonthlyCharges
Before Outliers Handling: TotalCharges
0 10000 20000 30000 40000 50000 60000 70000 80000
TotalCharges
After Outliers Handling: TotalCharges
0 1000 2000 3000 4000 5000 6000 7000
TotalCharges

EDA (EXPLORATORY DATA ANALYSIS)

Exploratory Data Analysis (EDA) is the process of analyzing and summarizing the main characteristics of a dataset, often using visual methods. It helps to uncover patterns, spot anomalies, test hypotheses, and check assumptions. EDA typically involves techniques such as descriptive statistics, data visualization (e.g., histograms, scatter plots, box plots), and correlation analysis. The goal is to better understand the structure of the data, identify relationships between variables, and guide further analysis or model-building steps.

Our dataset is consists of 3749 entries with 17 columns, excluding 'Unnamed: 0 , which is not so relevant for our analysis.

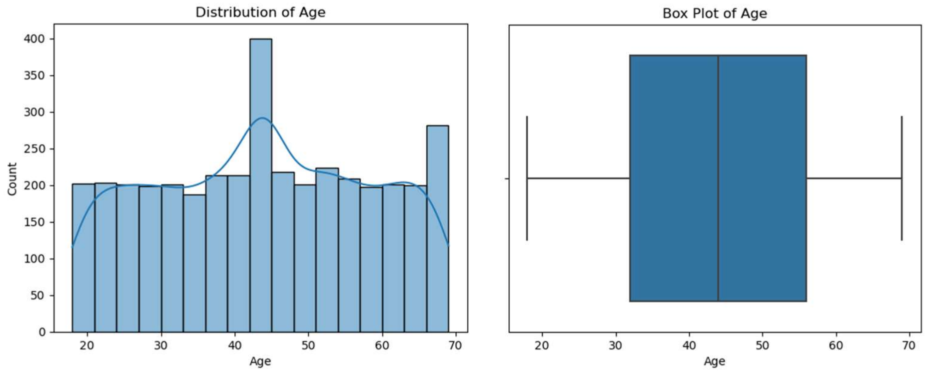
Visualising the Numerical Features

- This is to visualise the numerical features distribution and spread of those numerical features in the dataset using histograms and boxplots
- The features are **'Age'**, **'Tenure'**, **'MonthlyCharges'**, **'TotalCharges'** (Numerical features)
- Histograms(i.e using the **sns.histplot())** help to visualise the distribution of data along kernel density to represent data's probability density, which mainly helps in understanding the overall distribution of each feature .
- Boxplots**(sns.boxplot())** help to visualise spread of data as well in a different form , identifying median, quartiles and potential outliers.
- Comprehensive visual analysis of distribution and presence of outliers using histograms and boxplots is hence performed.

Age-

The histogram displays the age distribution in the dataset, with the x-axis representing age groups and the y-axis showing the number of individuals in each group. For the most part, the age distribution appears relatively uniform, except for a significant peak around the age of 40, where there is a noticeably higher number of individuals (i.e most of them are above middle aged). There's also a smaller rise in the number of people around the age of 70. Overall, the ages are fairly evenly distributed, with these two age ranges standing out.

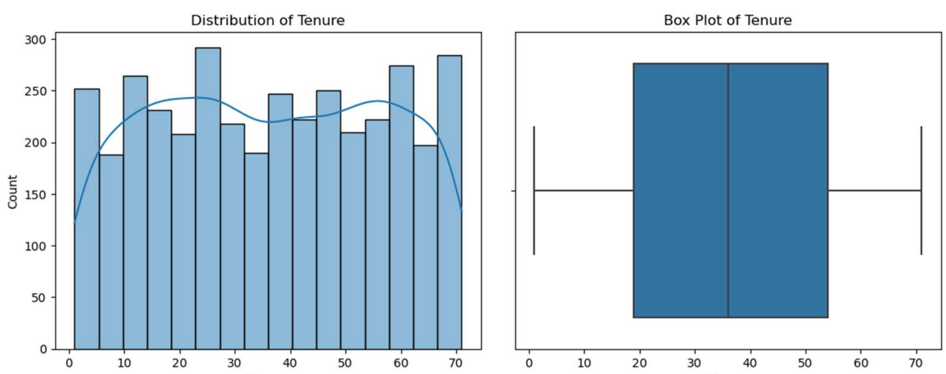
The box plot shows the distribution of age in the dataset. The box represents the middle 50% of the data (the interquartile range), with the median age indicated by the line inside the box, which is around 45 years. The "whiskers" extending from the box show the range of the data, with ages starting 20 to 70. There are no outliers, as no points fall outside the whiskers. This plot gives a basic view of the central tendency and spread of the ages in the dataset.



Tenure-

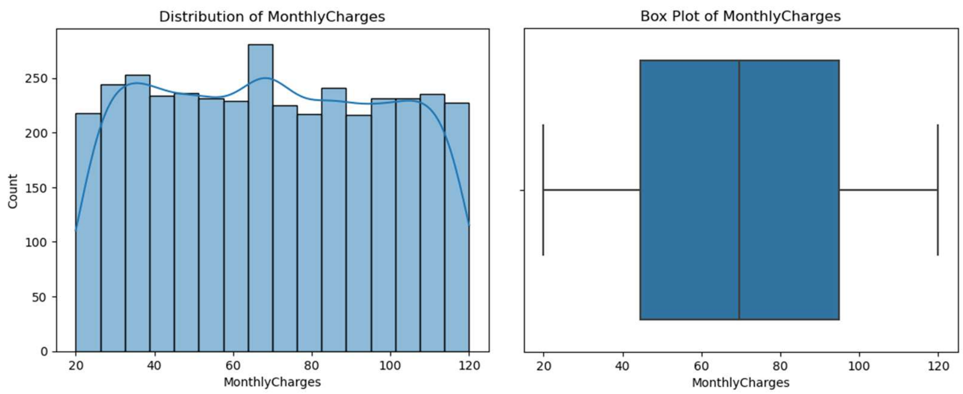
This histogram shows how long customers have stayed with the company. The x-axis shows different lengths of time, and the y-axis shows how many people fit into each group. The bars are mostly even, meaning the number of people in each group is similar. There are a few more customers around 10, 20, and 70 months, but fewer in the middle. Overall, customer stay times are quite different from each other.

Same like the histograms , box plot is another form of visualising the numerical features which helps in understanding the dataset and its features better , The box represents the middle 50% of the data. The line inside the box is the median, which is the middle value of the data. The whiskers represent the range of the data, excluding the outliers. The outliers are the points that are outside of the whiskers. we can see that the median tenure is around 40. The majority of people have a tenure between 20 and 50.



MonthlyCharges**-**

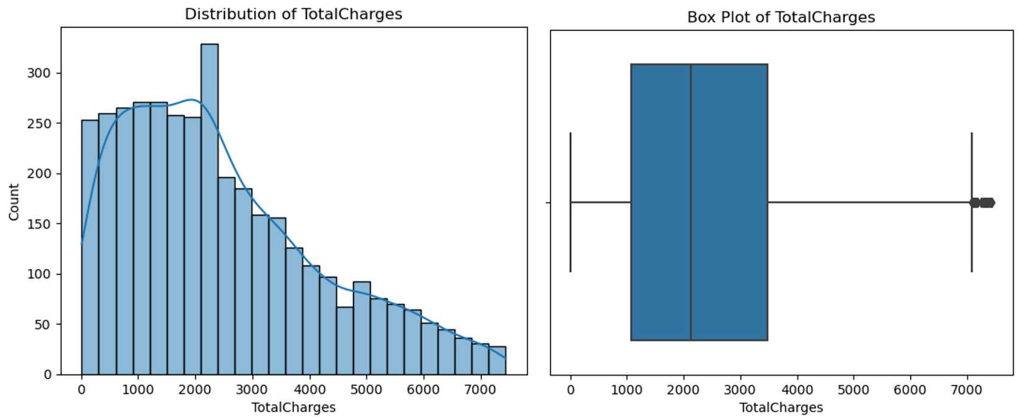
From both plots, we can see that the monthly charges are distributed fairly evenly between 20 and 120. The median monthly charge is around 80. The customers have a wide variety of service plans, with no plan being more popular than others.



TotalCharges**-**

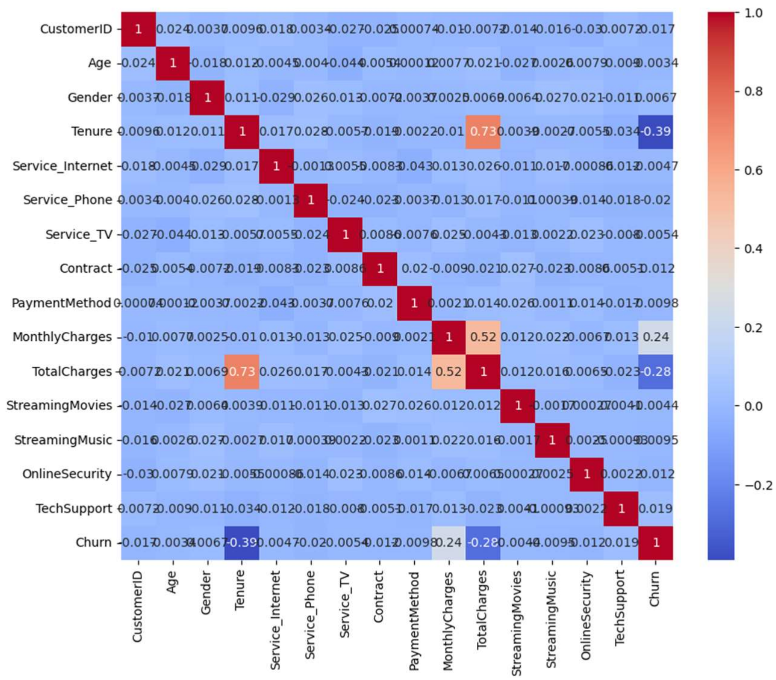
The total charges are skewed to the right, with a large number of customers having total charges below 3000. This skewness suggests there is room for improvement in customer retention. Most customers are paying between 1000 and 3000 in **TotalCharges.**

The two plots show the distribution of **TotalCharges**. The histogram reveals that most people pay between 1000 and 3000, with the highest number of people around 2000. As the charges increase, fewer people are paying higher amounts. The box plot supports this by showing that the middle 50% of people fall between about 1000 and 3000, while a few outliers (represented by dots) are paying much more than the rest. Overall, the graphs indicates that most payments are in the lower range, with only a less number of people paying very high charges.



Correlation

- I visualised the correlation between different features (**numerical and categorical**) variables in the dataset using heatmap to identify the strength and direction of relationships between variables and hence helps in guiding feature selection for modelling.
- Mainly by understanding these correlation between features helps us to identify which features have stronger influence on predicting the target variable **(Churn**) and helps in reducing multicollinearity by avoiding highly correlated features
- **Tenure** and **TotalCharges** have a strong positive correlation , meaning as a customer stays longer, their total charges increase accordingly
- **MonthlyCharges** and **TotalCharges** also show a moderate correlation.
- **Churn** is negatively correlated with **Tenure** , meaning customers who stay longer are less likely to leave
- **Contract** or **TechSupport**, have weaker relationships with **Churn**.



FEATURE ENGINEERING (Addition of new features)

- I added some new features to the data (i.e new columns of data) which help in prediction and further analysis. Feature creation will lead us to help in understanding customer behaviour and improve churn predictability.
- Mainly addition of new features is to capture the additional relations among the variables present which will help in improving the predictiveness of the models .
- Four features added are **'Monthly_Tenure_Ratio'**, **' Phone_Internet_Both'**, **'TotalCharges_Age_Ratio'** and '**Uses_streaming'**
- '**Monthly_Tenure_Ratio'** depends on **'MonthlyCharges'** and **'Tenure'** as it is ratio of both of them , involves providing the details into how much the customer will pay monthly over the customer's tenure. **'Phone_Internet_Both'** which will say that if the customer uses both phone and Internet services by checking the **Service_Internet** and **Service_Phone** (1 – Yes if customer uses both services , 0 – No if he/she doesnot). **'Total_Charges_Age_Ratio'** is the ratio of **TotalCharges** to that of **Age** , which will give the estimation of how much each customer has paid relative to their age. '**Uses_Streaming'** , this feature shows that if any of the customer using any of the streaming services (movies or music) . If they are using represented by 1 , if not used represented by 0. Its based on the customer subscription if he/she has or had subscribed to the either **'StreamingMovies'** or **'StreamingMusic'** or both.
- Then the whole dataframe after implementing the Feature Engineering is saved into a csv file.

```
In [16]: # Feature Engineering (Adding New interaction features)
df['Monthly_Tenure_Ratio'] = df['MonthlyCharges'] / (df['Tenure'] + 1)
df['Phone_Internet_Both'] = ((df['Service_Internet'] != 0) & (df['Service_Phone'] != 0)).astype(int)
df['TotalCharges_Age_Ratio'] = df['TotalCharges'] / (df['Age'] + 1)
df['Uses_Streaming'] = ((df['StreamingMovies'] != 0) | (df['StreamingMusic'] != 0)).astype(int)


# Saving the updated DataFrame to a new CSV file
df.to_csv('ML_updatedFeatureEngineeringDataset.csv', index=False)
```

- Then I used the **df.head()** function which prints the first few rows of the features to check if the feature engineering was successful or not .

```python
# Displaying the dataframe with the new features added
print(df.head())
```

```
   CustomerID   Age  Gender  Tenure  Service_Internet  Service_Phone  \
0         114  56.0       1      13                 0              1
1        2555  69.0       1      13                 0              0
2         459  46.0       1      60                 1              0
3        3609  32.0       0      57                 1              1
4        1053  60.0       1      52                 1              1

   Service_TV  Contract  PaymentMethod  MonthlyCharges  TotalCharges  \
0           0         1              3           71.88        931.49
1           1         2              3          110.99       1448.46
2           1         0              3          116.74       6997.73
3           1         0              0           78.16       4452.13
4           1         2              2           30.33       1569.73

   StreamingMovies  StreamingMusic  OnlineSecurity  TechSupport  Churn  \
0                0               0               1            0      0
1                1               1               0            0      0
2                1               1               0            0      0
3                0               1               0            1      0
4                1               0               1            1      0

   Monthly_Tenure_Ratio  Phone_Internet_Both  TotalCharges_Age_Ratio  \
0              5.134286                    0               16.341930
1              7.927857                    0               20.692286
2              1.913770                    0              148.887872
3              1.347586                    1              134.913030
4              0.572264                    1               25.733279

   Uses_Streaming
0               0
1               1
2               1
3               1
```

DATA MODELLING AND MODEL INTERPRETATION

Splitting the Dataset for Model Training

- I performed the Data splitting of the data in the dataset for machine learning (Splitting it into the features(X) and the target variable(y)). Then Performed the Train-Test Split to evaluate the models performance.
- All the columns except the target column **('Churn')** are assigned to X , which are used as the input features of the model.
- The **'Churn'** column is assigned to y , which represents the target variable we aim to predict.
- And using of **train_test_split()** splits the data into training and testing sets. In which 80 percent of data is allocated to training as obvious (training set – X_train, y_train) to train the model and 20 percent is allocated to testing as the test set (X_test, y_test) to evaluate how well the model will evaluate and generalise the unseen data

```
In [17]: # Assigning X as the DataFrame with the remaining  columns (excluding 'Churn')
X = df.drop(columns=['Churn'])

# Assigning y as the 'Churn' column
y = df['Churn']

# Print X and y
print("X (Columns DataFrame):")
print(X)
print("\nY (Churn Column):")
print(y)

#Train-Test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Saving the split data into CSV files
X_train.to_csv('X_train.csv', index=False)
X_test.to_csv('X_test.csv', index=False)
y_train.to_csv('y_train.csv', index=False)
y_test.to_csv('y_test.csv', index=False)

print("Train-Test split data saved to CSV files.")
```

- The data split into the following Training sets (X_train, y_train) and testing sets (X_test , y_test are saved into csv files. I attached the csv files here for reference

X_train.csv y_train.csv X_test.csv y_test.csv

Feature Selection

I performed three feature selection techniques on the data to see which features play a key role in predicting the churn which is the target variable .

Feature Selection Using Chi-Square Test**-**

I used the **SelectKBest** function to select the top features with highest Chi-Square Scores . I used the score_func=chi2 argument which uses the **Chi-Square statistical test** to rank the features. The scores of each feature are stored in Dataframe **(features_SKB**) along with the feature names and then the output will be sorted by the Chi-Square Ranking in descending order to show the most important features.

	Feature	Ranking
10	TotalCharges	370537.790336
15	Monthly_Tenure_Ratio	15025.094858
17	TotalCharges_Age_Ratio	9355.287944
3	Tenure	6710.526939
9	MonthlyCharges	2539.519831
0	CustomerID	643.854528
14	TechSupport	0.780247
7	Contract	0.534231
5	Service_Phone	0.439102
13	OnlineSecurity	0.339420
8	PaymentMethod	0.229804
16	Phone_Internet_Both	0.216927
1	Age	0.214609
12	StreamingMusic	0.171462
2	Gender	0.082627
18	Uses_Streaming	0.055767
6	Service_TV	0.045332
11	StreamingMovies	0.036121
4	Service_Internet	0.024715

Feature Selection using Extra – trees Classifier-

I performed another feature selection method using **ExtraTreesClassifier** , which helps to pick the top features predicting the target variable **('Churn')** based on the feature importance scores. Here I created an instance of **ExtraTreesClassifier** with 50 decision trees (n-estimators = 50) and it is fitted to the data X ,y) . This classifier is an ensemble based method that calculates the importance of each feature based on how useful it is in the Decision Trees. **SelectFromModel** is used to select features based on their importance scores. The threshold="median" ensures that only the features with an importance score above the median are selected.

The output is sorted by feature importance in descending order, showing the most important features first.

	Feature	Ranking
15	Monthly_Tenure_Ratio	0.280350
3	Tenure	0.186389
9	MonthlyCharges	0.182425
10	TotalCharges	0.134530
17	TotalCharges_Age_Ratio	0.067241
1	Age	0.020901
0	CustomerID	0.019547
7	Contract	0.014911
8	PaymentMethod	0.013655
2	Gender	0.009960
6	Service_TV	0.009834
13	OnlineSecurity	0.009088
12	StreamingMusic	0.008344
4	Service_Internet	0.008280
14	TechSupport	0.007989
11	StreamingMovies	0.007592
16	Phone_Internet_Both	0.006486
5	Service_Phone	0.006448
18	Uses_Streaming	0.006030

Feature Selection Using Sequential Feature Selection-

I performed yet another feature selection which performs the Feature selection using the **Sequential Feature Selector (SFS)**. It is to select the subset of the most relevant features from the dataset to improve models performance while reducing complexity. In this case, I used the forward selection , where features are added sequentially based on their contribution to model performance. The function takes in the classifier **(clf**), feature matrix **(X**), and target variable **(y).** The Sequential Feature Selector fits the model and identifies the top 5 features that maximize performance.

```
Selected Features:
     Selected Features
0               Tenure
1             Contract
2       MonthlyCharges
3  Monthly_Tenure_Ratio
4   Phone_Internet_Both
```

MODEL EVALUATION

In this section, I implemented a comparison of multiple machine learning models to assess their performance on the given dataset. The models included **Logistic Regression, Decision Tree**, **Random Forest**, **K-Neighbors**, and **Support Vector Machine (SVM with RBF Kernel)**. The purpose of this process was to identify which model works best for the prediction task.

For each model, I applied **cross-validation** with 5 folds, which helps in evaluating the performance of the model by splitting the training data into different subsets and ensuring the model performs well across different portions of the data. This technique also helps to avoid overfitting and gives a more reliable estimate of the model's accuracy.

After performing cross-validation, I saved the model to disk for later use using the pickle module. I then trained the models on the entire training set and evaluated their performance on both the training and testing datasets. The training accuracy and test accuracy were reported for each model, alongside classification reports which provide detailed insights of the **precision**, **recall**, and **F1 scores**.

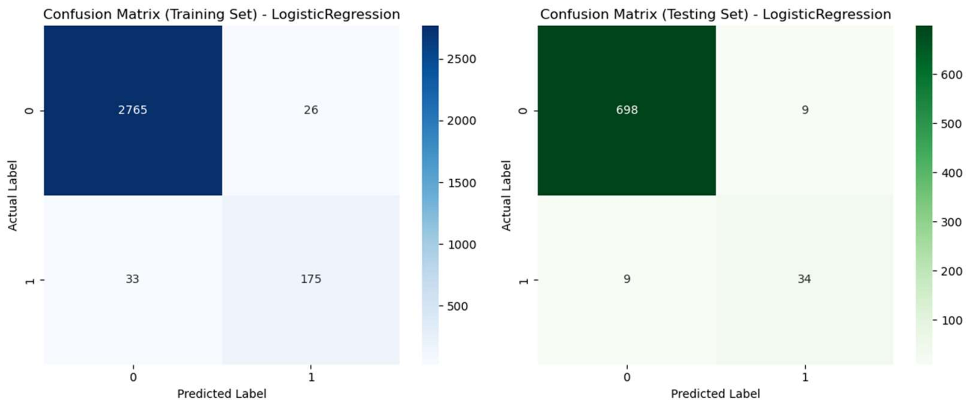
```
In [21]: # Defining the models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'K-Neighbors': KNeighborsClassifier(),
    'SVM (RBF Kernel)': SVC(kernel='rbf', probability=True)
}

# Function to save model using pickle
def save_model(model, model_name):
    with open(f"{model_name}.pkl", 'wb') as file:
        pickle.dump(model, file)
    print(f"Model saved: {model_name}.pkl")

# Function to perform cross-validation and print results
def perform_cross_validation(model, model_name, X_train, y_train, X_test, y_test, cv=5):
    # Perform cross-validation on the training set
    cv_results = cross_val_score(model, X_train, y_train, cv=cv, scoring='accuracy')

    # Printing average training accuracy from cross-validation
    print(f"{model.__class__.__name__}:")
    print(f"Avg Training Accuracy: {cv_results.mean() * 100:.2f}% (+/- {cv_results.std() * 100:.2f}%)")

    # Fit the model on the entire training set
    model.fit(X_train, y_train)

    # Save the model before hyperparameter tuning
    save_model(model, model_name + "_before_tuning")

    # Making predictions on training and test sets
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Calculation of training and test accuracies
    train_accuracy = accuracy_score(y_train, y_train_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)

    # Generate confusion matrices
    train_conf_matrix = confusion_matrix(y_train, y_train_pred)
    test_conf_matrix = confusion_matrix(y_test, y_test_pred)

    # Print training and test accuracies
    print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
    print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
    print()

    # Printing classification reports for both training and testing sets of data
    print("Classification Report - Training Set:")
    print(classification_report(y_train, y_train_pred))
    print("Classification Report - Testing Set:")
    print(classification_report(y_test, y_test_pred))
```

I also computed the **ROC-AUC** score, which is particularly useful for binary classification problems as it measures how well the model distinguishes between the two classes. Finally, I visualized the **confusion matrices** for both the training and testing sets, which helped me understand how well the models were performing in terms of true positive, true negative, false positive, and false negative predictions.

This evaluation process provided a comprehensive comparison of models and their effectiveness in predicting the target variable, allowing for the selection of the best-performing model for further optimization.

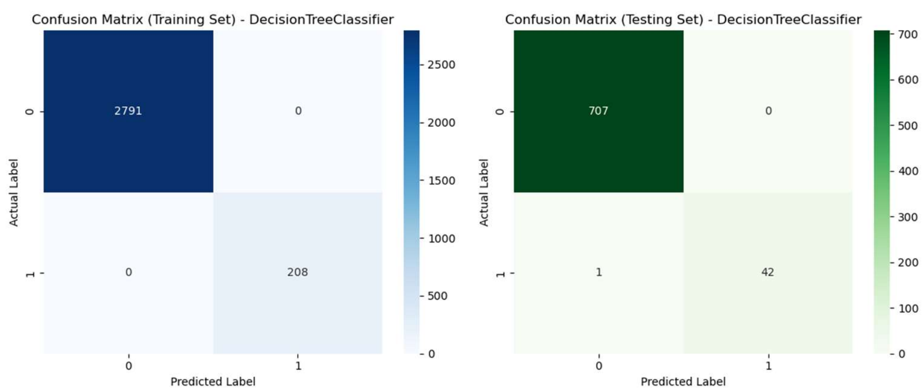
```
    # Calculation and printing of ROC-AUC score
    if len(set(y_test)) == 2:
        test_roc_auc = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])
        print(f"ROC AUC - Testing Set: {test_roc_auc:.2f}")

    # Visualize confusion matrices
    plt.figure(figsize=(12, 5))

    # Training confusion matrix
    plt.subplot(1, 2, 1)
    sns.heatmap(pd.DataFrame(train_conf_matrix), annot=True, cmap='Blues', fmt='g')
    plt.title(f"Confusion Matrix (Training Set) - {model.__class__.__name__}")
    plt.xlabel('Predicted Label')
    plt.ylabel('Actual Label')

    # Testing confusion matrix
    plt.subplot(1, 2, 2)
    sns.heatmap(pd.DataFrame(test_conf_matrix), annot=True, cmap='Greens', fmt='g')
    plt.title(f"Confusion Matrix (Testing Set) - {model.__class__.__name__}")
    plt.xlabel('Predicted Label')
    plt.ylabel('Actual Label')
    plt.tight_layout()
    plt.show()

#  perform crossvalidation function for each model and save separately:
for model_name, model in models.items():
    perform_cross_validation(model, model_name, X_train, y_train, X_test, y_test)
```

1) **Logistic Regression Classifier** with an accuracy of 97.83 %



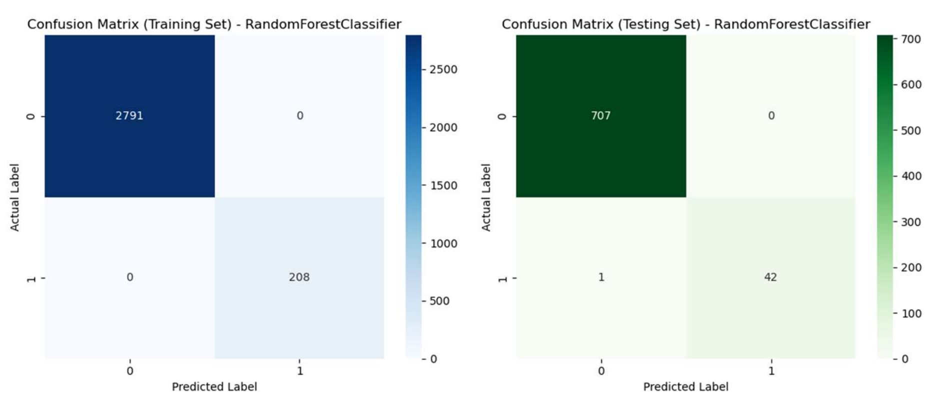
- This gave the moderate accuracy compared to the other models and this is the confusion matrix

2) **Decision Trees Classifier** with an accuracy of 99.87 %



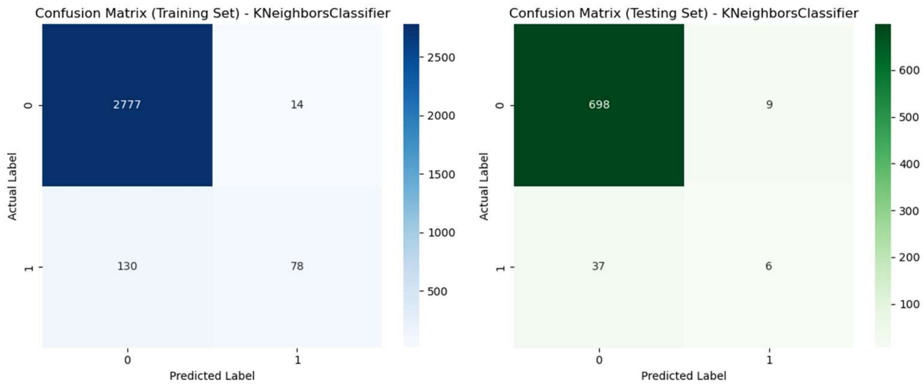
- This model provides a good performance and the confusion matrix is as follows.

3) **Random Trees Classifier** with an accuracy of 99.87% similarly like the Decision Trees Classifier



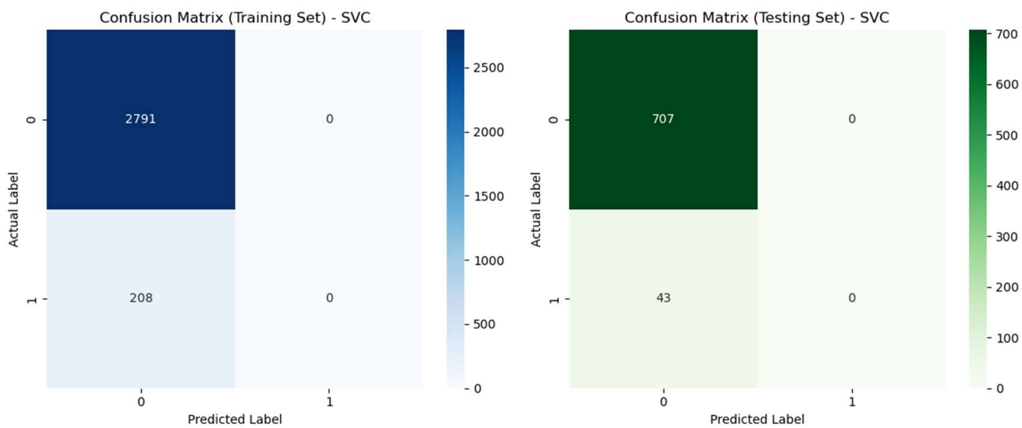
- This model also provides similar performance which is indeed a good performance same as the Decision Trees Classifier and the confusion matrix is as follows.

4) **K-Neigbors Classifier** with an accuracy of 93.87%



- This model also provide moderate performance in predicting and the confusion matrix is as follows.

5) **SVC – Classifier (kernel -rbf)** with an accuracy of 94.27 %



- This model also provides the moderately accurated report in predicting and the confusion matrix is as follows.

MODEL TUNING

In this section, hyperparameter tuning was performed for the machine learning models using GridSearchCV to identify the optimal parameters for each model. The models on which hyperparameter tuning are performed are **Logistic Regression, Decision Tree**, **Random Forest**, **K-Neighbors**, and **Support Vector Machine (SVM with RBF Kernel).**

I implemented the tuning with GridSearchCV for the models which was utilized to systematically explore various hyperparameter configurations for each model.

The following hyperparameters were used –

- For Logistic Regression, parameters such as 'C' (regularization strength) and 'penalty' were tuned.
- For Decisionn Trees , parameters such as ‘max_depth’ , 'min_samples_split', 'min_samples_leaf'.
- For Random Forest, the parameters such as 'n_estimators' (number of trees) and 'max_depth' (maximum depth of the trees)
- For K-Neighbors , parameters such as ‘n_neighbors’ and ‘weights’.
- For SVC (kernel -rbf) , parameters such as ‘C’ and ‘gamma’

Once the hyperparameters were optimized through cross-validation on the training data, the model with the best-performing parameters was saved. This ensures that we utilize the most effective version of each model for making predictions.

In [22]:
```
# Defining the models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'K-Neighbors': KNeighborsClassifier(),
    'SVM (RBF Kernel)': SVC(kernel='rbf', probability=True)
}

# Defining the hyperparameter grids for each model
param_grids = {
    'Logistic Regression': {'C': [0.01, 0.1, 1, 10], 'penalty': ['l2']},
    'Decision Tree': {'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4]},
    'Random Forest': {'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10]},
    'K-Neighbors': {'n_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance']},
    'SVM (RBF Kernel)': {'C': [0.1, 1, 10], 'gamma': ['scale', 'auto']}
}

# Function to save model using pickle
def save_model(model, model_name):
    with open(f"{model_name}_after_tuning.pkl", 'wb') as file:
        pickle.dump(model, file)
    print(f"Model saved: {model_name}_after_tuning.pkl")

# Function to perform hyperparameter tuning, cross-validation, and print results
def perform_hyperparameter_tuning_and_evaluation(model, model_name, param_grid, X_train, y_train, X_test, y_test, cv=5):
# Perform GridSearchCV for hyperparameter tuning
    print(f"Hyperparameter tuning for {model_name}...")
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=cv, scoring='accuracy')
    grid_search.fit(X_train, y_train)

    # To Get the best model after tuning
    best_model = grid_search.best_estimator_

    # Save the model after hyperparameter tuning
    save_model(best_model, model_name)
```

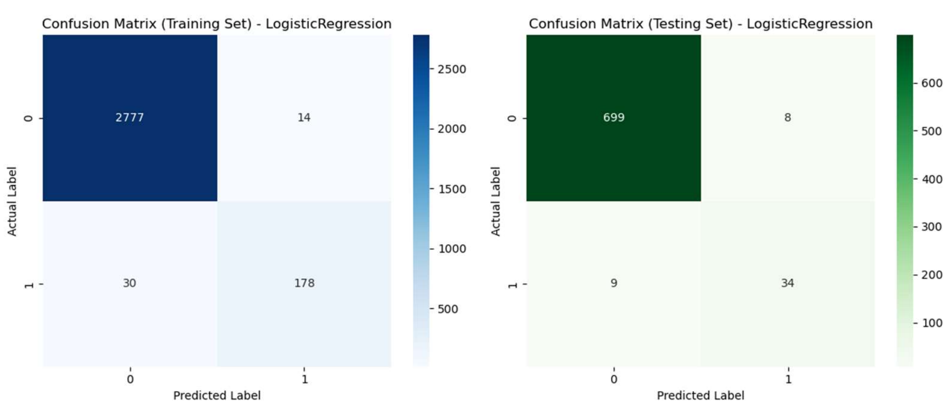
After tuning, each model was evaluated on both the training and testing datasets. This evaluation involved calculating training and test accuracies, generating classification reports (including precision, recall, and F1-score), and visualizing confusion matrices for both datasets. These metrics and visualizations are crucial for understanding how well the models classify instances across different categories.

For binary classification problems, the ROC-AUC score was also computed to assess how effectively the models differentiate between the two classes.This thorough approach allowed for a detailed comparison of model performances, ensuring that the selected models generalize well to new data. The results of this evaluation will inform the choice of the best model for final deployment.

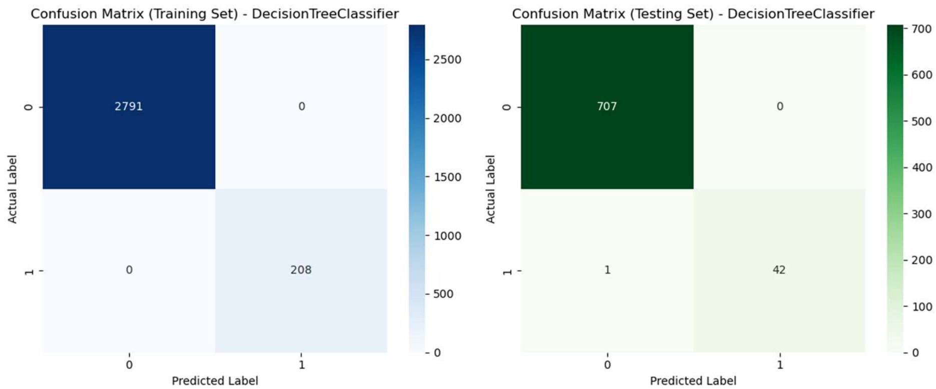
```python
    # To Perform cross-validation on the training set with the best model
    cv_results = cross_val_score(best_model, X_train, y_train, cv=cv, scoring='accuracy')

    # Print average training accuracy from cross-validation
    print(f"{model.__class__.__name__}:")
    print(f"Avg Training Accuracy: {cv_results.mean() * 100:.2f}% (+/- {cv_results.std() * 100:.2f}%)")

    # Making predictions on training and test sets
    y_train_pred = best_model.predict(X_train)
    y_test_pred = best_model.predict(X_test)

    # Calculate training and test accuracies
    train_accuracy = accuracy_score(y_train, y_train_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)

    # Generation of confusion matrices
    train_conf_matrix = confusion_matrix(y_train, y_train_pred)
    test_conf_matrix = confusion_matrix(y_test, y_test_pred)

    # Print training and test accuracies
    print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
    print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
    print()

    # Printing classification reports for both training and testing sets of data
    print("Classification Report - Training Set:")
    print(classification_report(y_train, y_train_pred))
    print("Classification Report - Testing Set:")
    print(classification_report(y_test, y_test_pred))

    # Calculation and printing of ROC-AUC score
    if len(set(y_test)) == 2:  # Check if it's a binary classification problem
        test_roc_auc = roc_auc_score(y_test, best_model.predict_proba(X_test)[:, 1])
        print(f"ROC AUC - Testing Set: {test_roc_auc:.2f}")

    # Visualize confusion matrices
    plt.figure(figsize=(12, 5))

    # Training confusion matrix
    plt.subplot(1, 2, 1)
    sns.heatmap(pd.DataFrame(train_conf_matrix), annot=True, cmap='Blues', fmt='g')
    plt.title(f"Confusion Matrix (Training Set) - {model.__class__.__name__}")
    plt.xlabel('Predicted Label')
    plt.ylabel('Actual Label')

    # Testing confusion matrix
    plt.subplot(1, 2, 2)
    sns.heatmap(pd.DataFrame(test_conf_matrix), annot=True, cmap='Greens', fmt='g')
    plt.title(f"Confusion Matrix (Testing Set) - {model.__class__.__name__}")
    plt.xlabel('Predicted Label')
    plt.ylabel('Actual Label')

    plt.tight_layout()
    plt.show()

# Perform hyperparameter tuning and cross-validation for each model and save the best models:

for model_name, model in models.items():
    perform_hyperparameter_tuning_and_evaluation(model, model_name, param_grids[model_name], X_train, y_train, X_test, y_test)
```

After performing the Hyperparameter tuning , couple of models like **Logistic Regression, K-neighbors** performances have been improved a bit , while other models remain unchanged .

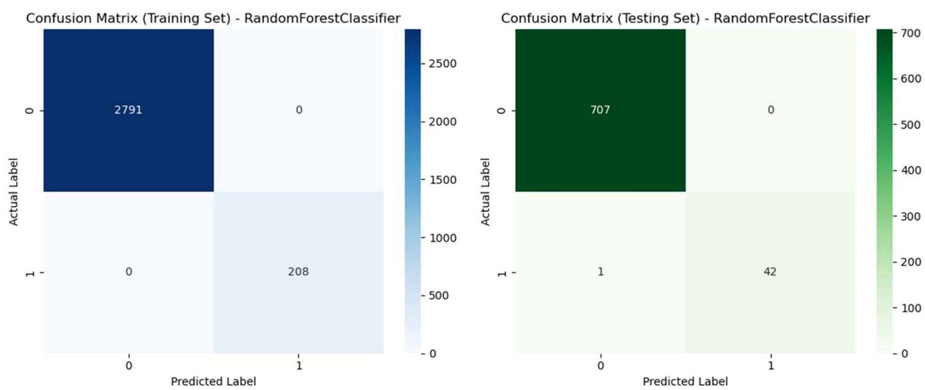
1) **Logistic Regression** with an accuracy of 97.73 % , the performance has increased a bit after the tuning.



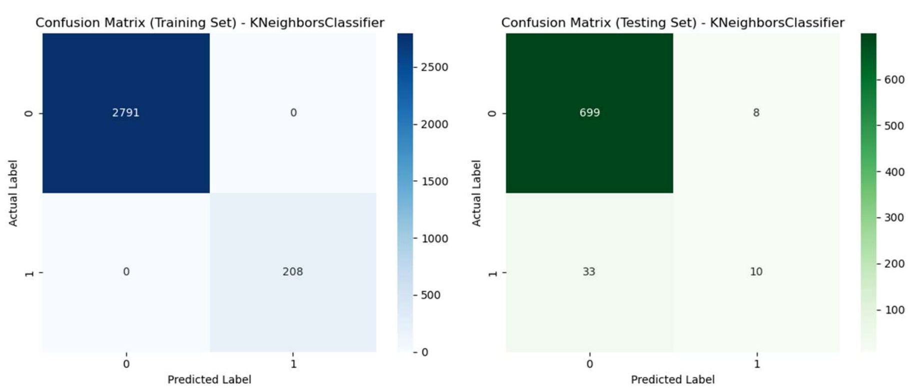
2) **Decision Trees** model with the same accuracy before and after tuning .



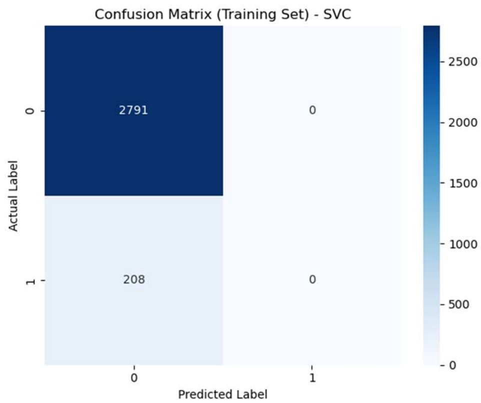
3) **Random Forest Model** also with same accuracy like the decision trees before and after tuning.

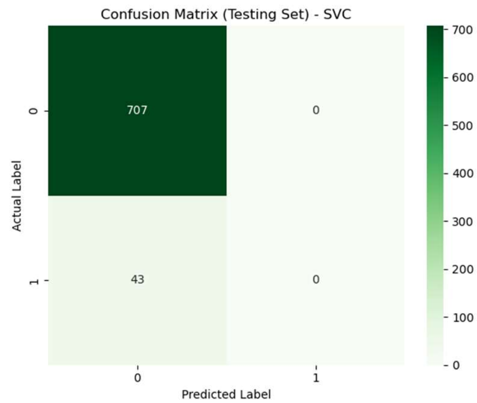


4) **K-Neighbors Model** model with the improved accuracy of 93.83%



5) **SVC (kernel = 'rbf') Model** with the similar accuracy before and after training.





<u>**CONCLUSION**</u>

The main aim of this is to predict the Customer's Churn Variable. This project is about predicting customer churn using machine learning. It explains the steps taken to prepare the dataset and build models that can make accurate predictions.

First, the dataset was cleaned to fix missing values. For numerical data, the missing values were filled using the median, while for categorical data, the most frequent value (mode) was used. Unnecessary column is also removed to make the data simpler and easier to analyze and the categorical variables are encoded for better interpretation by the models.

Next, outliers, or extreme values, were handled using the Interquartile Range (IQR) method. Outliers were replaced with the median value to ensure that the dataset remained consistent. Visualizations were created to see the effect of this on key features.

Feature engineering was an important step to improve the dataset. New features were added to better capture relationships between the data, which helped the models perform better. The dataset was then split into training and testing sets, which helped in evaluating the models.

Several machine learning models are tested, such as Logistic Regression, Decision Trees, Random Forest, K-Neighbors, and Support Vector Machines. These models are evaluated here using cross-validation, accuracy metrics, confusion matrices, and ROC-AUC scores. Decision Trees and Random Forest performed the best, with the highest accuracy rates Hyperparameter tuning was also done to improve model performance. This process involved trying different settings for each model, which slightly improved some models, while others stayed the same.

This hence effectively predicted customer churn by implementing a thorough process of data preprocessing,feature engineering, feature selection, and model evaluation. The Decision Trees and Random Forest models delivered the best results, making them strong candidates for churn prediction tasks