Data science

PROJECT-Churn customer prediction

A training report

Submitted in partial fulfillment of the requirements for the award of degree of

B.TECH- CSE

LOVELY PROFESSIONAL UNIVERSITY

…………PHAGWARA, PUNJAB……

SUBMITTED BY

Name of Student: SHIVAM KULSHRESHTHA

Registration Number: 12108892

Signature of Student: shivam

INTRODUCTION

In the current market, there is fierce competition for E-commerce companies and DTH providers (you can select any of these two areas). As a result, it has become difficult to hold onto current clients. As a result, the business is trying to create a model that will allow them to predict account churn and send targeted offers to those who might consider canceling. Because a single account can have several customers, account churn is a big concern for this organization. Therefore, the business may lose more than one customer as a result of losing one account.

ABOUT: DATASET

The main columns that are commonly present in PG churn prediction datasets are listed below, along with an explanation of their importance:

Each client record in the dataset is uniquely identified by its customer ID (unique identifier).

Details on the demographics:

Age: Can reveal a customer's changing wants and preferences as they get older.

Gender: May have an impact on how a product is used or how responsive it is to promotions.

Location (nation, area, or city): The behavior of customers and pricing policies can be influenced by geographic considerations.

Details of the Subscription:

Start Date of Subscription: The amount of time since the start of the subscription can indicate trends in churn risk.

Subscription Plan (basic, premium, free trial, etc.): The likelihood of cancellation may differ throughout plans.

Payment Method (debit, credit, etc.): The ease of use or complexity of the payment method may have an impact on customer attrition.

Length of Contract (monthly, yearly, etc.): Durations of contracts have an impact on exit points and churn probability.

How to import data set

3. Data Ingestion

Importing necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warningsS
warnings.filterwarnings("ignore")
```

In [2]:
```
cf=pd.read_excel('Customer Churn Data.xlsx',sheet_name='data') # reading the data set
```

HOW TO FETCH TOP AND LAST ROWS

By the use of head() and tail() function

cf.head(10) #top 5 rows get displayed

```
In [4]: cf.tail() #last bottom 5 rows get displayed
```

Out[4]:

	AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Account_user_count	account_segment	CC_Agent_Score	Mari
11255	31255	0	10	1.0	34.0	Credit Card	Male	3.0	2	Super	1.0	
11256	31256	0	13	1.0	19.0	Credit Card	Male	3.0	5	HNI	5.0	
11257	31257	0	1	1.0	14.0	Debit Card	Male	3.0	2	Super	4.0	
11258	31258	0	23	3.0	11.0	Credit Card	Male	4.0	5	Super	4.0	
11259	31259	0	8	1.0	22.0	Credit Card	Male	3.0	2	Super	3.0	

Data cleaning

- Common Data Quality Issues
- Missing Values: Instances where data points are absent.
- Duplicate Entries: Repeated data entries that can skew analysis.
- Inconsistent Data: Data that is not uniform, such as different formats for dates.
- Outliers: Data points that deviate significantly from the rest of the dataset.
- Incorrect Data: Errors in data entry that lead to invalid data points.
- Irrelevant Data: Data that does not contribute to the analysis or model.

4. Data Cleaning

Checking for null values and duplicate values

In [5]:
```
cf.dtypes #  data types of each individual column get printed
```

Out[5]:
```
AccountID                    int64
Churn                        int64
Tenure                      object
City_Tier                  float64
CC_Contacted_LY            float64
Payment                     object
Gender                      object
Service_Score              float64
Account_user_count          object
account_segment             object
CC_Agent_Score             float64
Marital_Status              object
rev_per_month               object
Complain_ly                float64
rev_growth_yoy              object
coupon_used_for_payment     object
Day_Since_CC_connect        object
cashback                    object
Login_device                object
dtype: object
```

Here it is observed that there are 12 feature that are categorical, five are float type and remaining are of integer type only.

In [6]:
```
cf.isnull().sum() # checking each individual column that there are Null values or not
```

Out[6]:
```
AccountID                    0
Churn                        0
Tenure                     102
City_Tier                  112
CC_Contacted_LY            102
Payment                    109
Gender                     108
Service_Score               98
Account_user_count         112
account_segment             97
CC_Agent_Score             116
Marital_Status             212
rev_per_month              102
Complain_ly                357
rev_growth_yoy               0
coupon_used_for_payment      0
Day_Since_CC_connect       357
cashback                   471
Login_device               221
dtype: int64
```

Here the column which are null values we are going to replace with median and mode ,i.e categorical by mode and numerical by mean

In [7]:
```
cf.duplicated().sum() #there no duplicated rows
```

Out[7]:
```
0
```

In [18]:
```python
#now lets fill object column null values with mode
object_columns = cf.select_dtypes(include=['object']).columns
cf[object_columns] = cf[object_columns].fillna(cf[object_columns].mode().iloc[0])
```

In [19]:
```python
#now lets check again that there is null counts or not
cf.isnull().sum()
```

Out[19]:
```
Churn                  0
Tenure                 0
City_Tier              0
CC_Contacted_LY        0
Service_Score          0
Account_user_count     0
account_segment        0
CC_Agent_Score         0
rev_per_month          0
Complain_ly            0
rev_growth_yoy         0
Day_Since_CC_connect   0
dtype: int64
```

In [20]:
```python
# Check unique values in each column
for column in cf.columns:
    unique_values = cf[column].unique()
    print(f"Unique values in {column}: {unique_values}")
```

```
Unique values in Churn: [1. 0.]
Unique values in Tenure: [ 4.  0.  2. 13. 11.  9. 99. 19. 20. 14.  8. 26. 18.  5. 30.  7.  1. 23.
  3. 29.  6. 28. 24. 25. 16. 10. 15. 22. 27. 12. 21. 17. 50. 60. 31. 51.
 61.]
Unique values in City_Tier: [3. 1. 2.]
Unique values in CC_Contacted_LY: [  6.   8.  30.  15.  12.  22.  11.   9.  31.  18.  13.  20.  29.  28.
  26.  14.  10.  25.  27.  17.  23.  33.  19.  35.  24.  16.  32.  21.
  34.   5.   4. 126.   7.  36. 127.  42.  38.  37.  39.  40.  41. 132.
  43. 129.]
Unique values in Service_Score: [3. 2. 1. 0. 4. 5.]
Unique values in Account_user_count: [3. 4. 5. 2. 1. 6.]
Unique values in account_segment: ['Super' 'Regular Plus' 'Regular' 'HNI' 'Regular +' 'Super Plus' 'Super +']
Unique values in CC_Agent_Score: [2. 3. 5. 4. 1.]
Unique values in rev_per_month: [  9.   7.   6.   8.   3.   2.   4.  10.   1.   5. 130.  19. 139. 102.
 120. 138. 127. 123. 124. 116.  21. 126. 134. 113. 114. 108. 140. 133.
 129. 107. 118.  11. 105.  20. 119. 121. 137. 110.  22. 101. 136. 125.
  14.  13.  12. 115.  23. 122. 117. 131. 104.  15.  25. 135. 111. 109.
 100. 103.]
Unique values in Complain_ly: [1. 0.]
Unique values in rev_growth_yoy: [11. 15. 14. 23. 22. 16. 12. 13. 17. 18. 24. 19. 20. 21. 25. 26.  4. 27.
 28.]
Unique values in Day_Since_CC_connect: [ 5.  0.  3.  7.  2.  1.  8.  6.  4. 15. 11. 10.  9. 13. 12. 17. 16. 14.
 30. 46. 18. 31. 47.]
```

Exploratory Data Analysis

- The significance of EDA
- Data Understanding: Offers a thorough understanding of the dataset, covering its relationships, primary features, and structure.
- Finding trends, patterns, and outliers that might not be immediately obvious is made easier with the aid of insight generation.
- Assumption Validation: Verifies the validity of the underlying assumptions in statistical models.
- Selecting features: Assists in determining the most pertinent variables to include in a model.

- Data cleaning: Makes it easier to find errors and abnormalities that should be fixed before doing more analysis.

5. EDA (Exploratory Data Analysis)

In [21]:
```
cf.sample(6) #this line will randomly select any five column from dataset
```

Out[21]:

	Churn	Tenure	City_Tier	CC_Contacted_LY	Service_Score	Account_user_count	account_segment	CC_Agent_Score	rev_per_month	Complain_ly	rev_g
3916	0.0	6.0	1.0	17.0	3.0	4.0	Regular Plus	3.0	101.0	0.0	
4746	0.0	1.0	1.0	16.0	3.0	5.0	Regular Plus	4.0	3.0	0.0	
4547	0.0	14.0	3.0	9.0	3.0	4.0	Super	4.0	9.0	0.0	
9804	1.0	9.0	1.0	37.0	3.0	5.0	Regular Plus	3.0	8.0	1.0	
6675	0.0	0.0	1.0	13.0	2.0	3.0	Regular +	1.0	5.0	0.0	
8073	0.0	13.0	1.0	21.0	3.0	4.0	Super	1.0	5.0	1.0	

Attributes information :¶

- churn: account churn flag (Target)
- Tenure:Tenure of account
- City_Tier:Tier of primary customer's city
- cc_contacted_Ly:How many time the customer of the account has contacted customer care in last 12months
- Service_Score:Satisfaction score given by customers of the account on service provided by company
- Account_user_count:Number of customers tagged with this account
- account_segment:Account segmentation on the basis of spend
- CC_Agent_Score:Satisfaction score given by customers of the account on customer care service provided by company
- rev_per_month:Monthly average revenue generated by account in last 12 months
- compplain_ly:Any complaints has been raised by account in last 12 months
- rev_growth_yoy:revenue growth percentage of the account (last 12 months vs last 24 to 13month)
- Day_Since_CC_connect:Number of days since no customers in the account has contacted the customer care

In [22]:
```
cf.shape #{checking the number of rows and column}
```

Out[22]:
```
(11260, 12)
```

In [23]:
```
cf.info() #{ checking the datatypes and count of null values if present}
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11260 entries, 0 to 11259
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Churn                 11260 non-null  float64
 1   Tenure                11260 non-null  float64
 2   City_Tier             11260 non-null  float64
 3   CC_Contacted_LY       11260 non-null  float64
 4   Service_Score         11260 non-null  float64
 5   Account_user_count    11260 non-null  float64
 6   account_segment       11260 non-null  object
 7   CC_Agent_Score        11260 non-null  float64
 8   rev_per_month         11260 non-null  float64
 9   Complain_ly           11260 non-null  float64
 10  rev_growth_yoy        11260 non-null  float64
 11  Day_Since_CC_connect  11260 non-null  float64
dtypes: float64(11), object(1)
memory usage: 1.0+ MB
```

In [24]:
```
cf['Churn'].value_counts() # Here it can be seen that the number of 1's is very less as compared to number of 0's. So that datas
```

Out[24]:
```
0.0    9364
1.0    1896
Name: Churn, dtype: int64
```

Here it can be seen that the number of 1's is very less as compared to number of 0's. So that dataset is imbalanced dataset.

Separating numerical and categorical features

In [25]:
```
num_feature= [fea for fea in cf.columns if cf[fea].dtype !=object]
cat_feature= [fea for fea in cf.columns if cf[fea].dtype==object]
```

In [26]:
```
print("We have {} Numerical features : {}".format(len(num_feature),num_feature))
print()
print("We have {} Categorical features : {}".format(len(cat_feature),cat_feature))
```

```
We have 11 Numerical features : ['Churn', 'Tenure', 'City_Tier', 'CC_Contacted_LY', 'Service_Score', 'Account_user_count', 'CC_Agent_Score', 'rev_per_month', 'Complain_ly', 'rev_growth_yoy', 'Day_Since_CC_connect']

We have 1 Categorical features : ['account_segment']
```

Statistical Description

In [27]:
```
cf.describe().T
```

Out[27]:

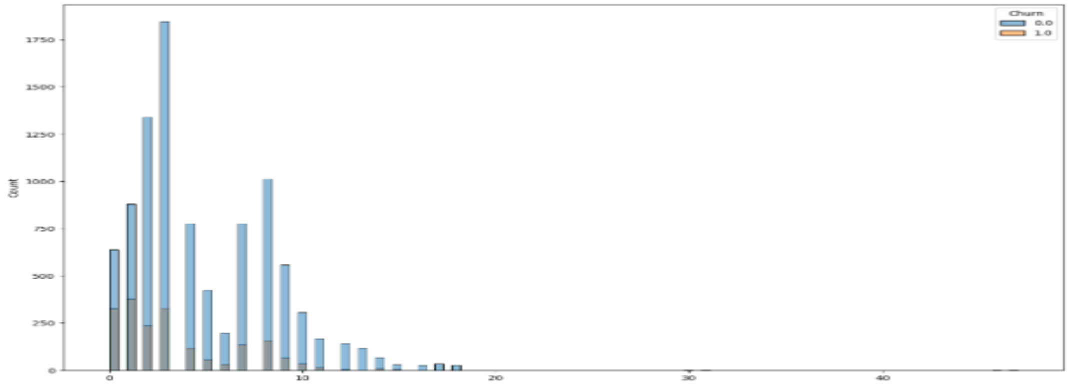
	count	mean	std	min	25%	50%	75%	max
Churn	11260.0	0.168384	0.374223	0.0	0.0	0.0	0.0	1.0
Tenure	11260.0	10.985879	12.757534	0.0	2.0	9.0	16.0	99.0
City_Tier	11260.0	1.647425	0.912763	1.0	1.0	1.0	3.0	3.0
CC_Contacted_LY	11260.0	17.850178	8.814851	4.0	11.0	16.0	23.0	132.0
Service_Score	11260.0	2.903375	0.722476	0.0	2.0	3.0	3.0	5.0
Account_user_count	11260.0	3.704973	1.004383	1.0	3.0	4.0	4.0	6.0
CC_Agent_Score	11260.0	3.065808	1.372663	1.0	2.0	3.0	4.0	5.0
rev_per_month	11260.0	6.266874	11.488990	1.0	3.0	5.0	7.0	140.0
Complain_ly	11260.0	0.276288	0.447181	0.0	0.0	0.0	1.0	1.0
rev_growth_yoy	11260.0	16.193073	3.757271	4.0	13.0	15.0	19.0	28.0
Day_Since_CC_connect	11260.0	4.581261	3.649643	0.0	2.0	3.0	7.0	47.0

ANALYSIS with the help of graph

- Histograms:

  A single numerical variable's frequency distribution should be displayed.helpful in comprehending the distribution and central tendency of the data.

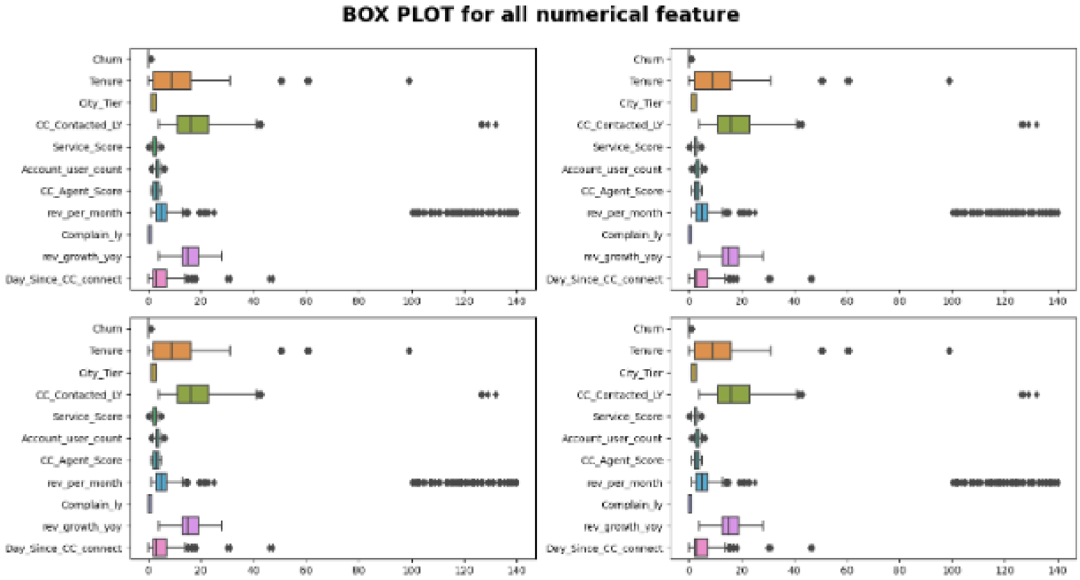
In [34]:
```
plt.figure(figsize=(15,10))
sns.histplot(x='Day_Since_CC_connect',hue='Churn',data=cf)
plt.show()
```



- Plots in boxes:

  Show a dataset's variability, central value, and distribution.
  efficient in comparing distributions between groups and identifying outliers.

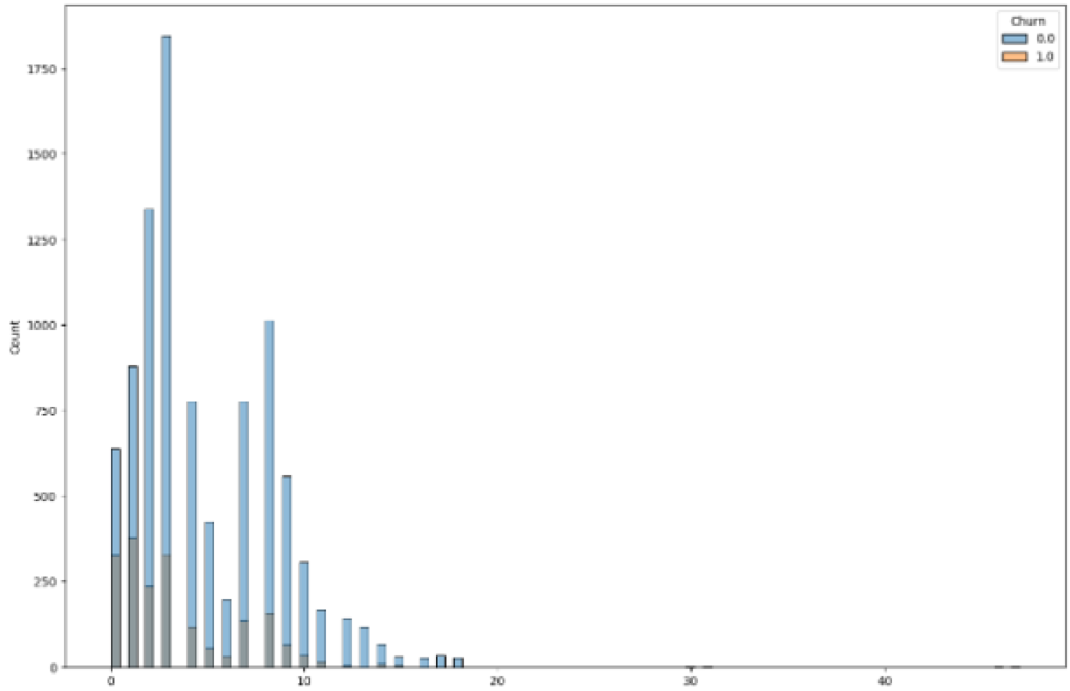
In [37]:
```
plt.figure(figsize=(15,30))
plt.suptitle('BOX PLOT for all numerical feature',fontsize=20,fontweight='bold',y=1)

for i in range(0,len(num_feature)):
    plt.subplot(8,2,i+1)
    sns.boxplot(data=cf[num_feature],orient='h') # checking the outliers are present or not
    plt.tight_layout()
```



- Bar Diagrams:

  Use rectangular bars to show categorical data.
  helpful in comparing the number or frequency of groups.

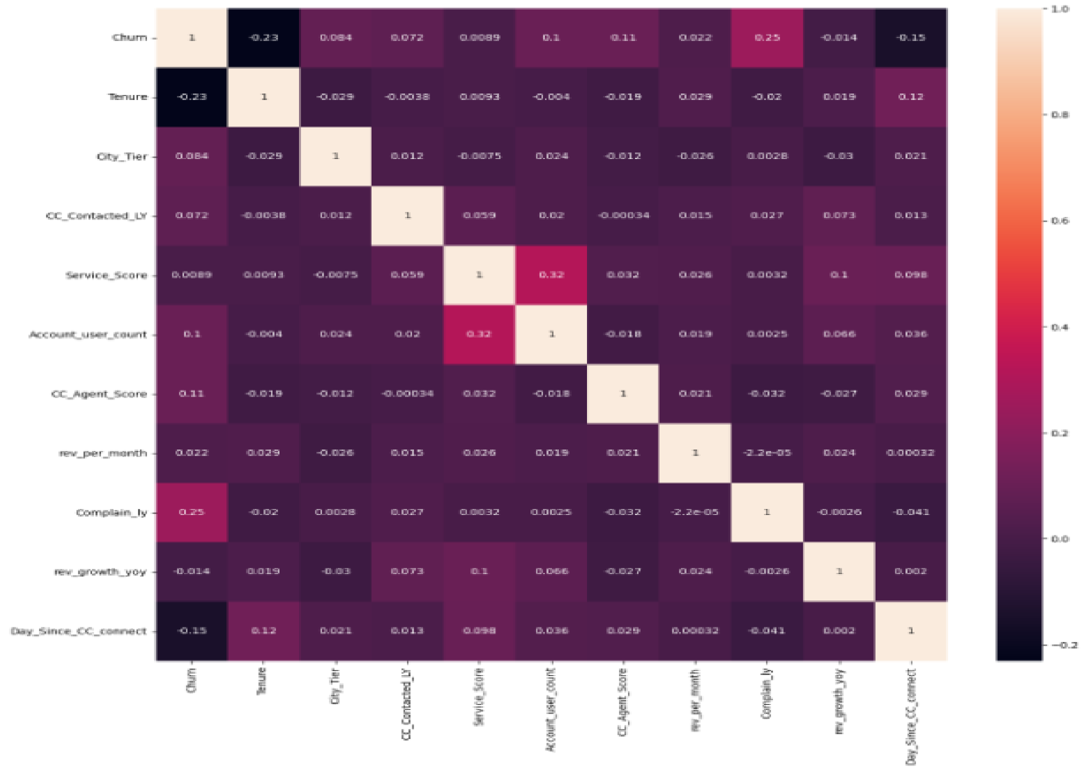
In [34]:
```
plt.figure(figsize=(15,10))
sns.histplot(x='Day_Since_CC_connect',hue='Churn',data=cf)
plt.show()
```



- Heat maps:

  Make use of color gradients to visualize the variables' correlation matrix.
  Excellent for identifying robust correlations and patterns among several variables.

In [33]:
```
plt.figure(figsize=(15,14))
sns.heatmap(cf.corr(),annot=True)
plt.show()
```

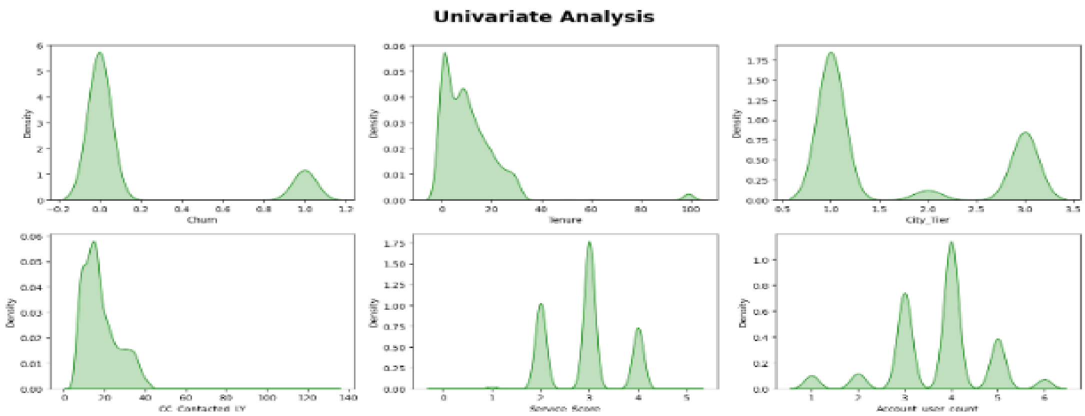


- Line Diagrams:

  Monitor a variable's changes over time.
  beneficial for identifying cycles, trends, and seasonal patterns in time-series data

Univariate Analysis

In [28]:
```
plt.figure(figsize=(15,17))
plt.suptitle('Univariate Analysis',fontsize=20,fontweight='bold',y=1)

for i in range(0,len(num_feature)):
    plt.subplot(5,3,i+1)
    sns.kdeplot(x=cf[num_feature[i]],shade=True,color='g')
    plt.xlabel(num_feature[i])
    plt.tight_layout()
```



DATA PRE-PROCESSING

- The Value of Preparing Data
- Enhanced Data Quality: Assures that the information is accurate, dependable, and clean.
- Improved Model Performance: Results in improved machine learning model training.
- Effective Analysis: Reduces complexity of the data, facilitating manipulation and examination.
- Decreased Variance and Bias: Aids in dataset balancing and error reduction.

6. Data Preprocessing

feature encoding to covert the categorical features into numerical values

cf.head() #checking top 5 rows of the data set

In [39]:
```
#Now doing one_hot_encoding to convert categorical features into numerical feature
cf1=pd.get_dummies(data=cf,columns=['account_segment'],drop_first=True)
```

In [40]:
```
cf1.head()
```

Out[40]:

	Churn	Tenure	City_Tier	CC_Contacted_LY	Service_Score	Account_user_count	CC_Agent_Score	rev_per_month	Complain_ly	rev_growth_yoy	Day_Since_
0	1.0	4.0	3.0	6.0	3.0	3.0	2.0	9.0	1.0	11.0	
1	1.0	0.0	1.0	8.0	3.0	4.0	3.0	7.0	1.0	15.0	
2	1.0	0.0	1.0	30.0	2.0	4.0	3.0	6.0	1.0	14.0	
3	1.0	0.0	3.0	15.0	2.0	4.0	5.0	8.0	0.0	23.0	
4	1.0	0.0	1.0	12.0	2.0	3.0	5.0	3.0	0.0	11.0	

Extracting the target column into separate vectors for training set and test set

In [41]:
```
X=cf1.drop(['Churn'],axis=1) #dropping the  label
Y=cf1['Churn']
X.shape,Y.shape #checking shape
```

Out[41]:
```
((11260, 16), (11260,))
```

In [42]:
```
X.head()
```

Out[42]:

	Tenure	City_Tier	CC_Contacted_LY	Service_Score	Account_user_count	CC_Agent_Score	rev_per_month	Complain_ly	rev_growth_yoy	Day_Since_CC_con
0	4.0	3.0	6.0	3.0	3.0	2.0	9.0	1.0	11.0	
1	0.0	1.0	8.0	3.0	4.0	3.0	7.0	1.0	15.0	
2	0.0	1.0	30.0	2.0	4.0	3.0	6.0	1.0	14.0	
3	0.0	3.0	15.0	2.0	4.0	5.0	8.0	0.0	23.0	
4	0.0	1.0	12.0	2.0	3.0	5.0	3.0	0.0	11.0	

In [43]:
```
Y.head()
```

Out[43]:
```
0    1.0
1    1.0
2    1.0
3    1.0
4    1.0
Name: Churn, dtype: float64
```

NOW ABOUT TRAIN AND TEST OF MODEL

- Train-Test Split Model Validation is important because it offers a strong framework for evaluating the model's capacity to generalize to new data.
  Performance Metrics: Provides the ability to compute metrics on unseen data, including accuracy, precision, recall, and F1-score.
  Preventing Overfitting: Assesses the model on an independent test set to assist in identifying and reducing overfitting.

- Steps for Data Splitting in Train-Test Split: splitting the dataset, usually in an 80-20 or 70-30 ratio, into training and testing sets.
  Model Training: Using the training set, which includes most of the data, the model is constructed.
  Model testing is the process of assessing a model's performance using data from the test set that was not observed by the model during training.

Splitting the dataset into train and test data

In [48]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X_res,Y_res,test_size=0.2,random_state=0)
```

Feature scaling

In [49]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

In [50]:
```
X_train
```
Out[50]:
```
array([[-0.43895116, -0.78523283, -0.07388762, ..., -0.63665676,
        -0.04770982, -0.22312806],
       [ 1.43615636,  1.39689885, -0.38123016, ..., -0.63665676,
        -0.04770982,  4.48173131],
       [ 0.27787186,  1.39689885, -1.05993811, ...,  1.57070507,
        -0.04770982, -0.22312806],
       ...,
       [ 0.81840463, -0.78523283, -0.04187618, ..., -0.63665676,
        -0.04770982, -0.22312806],
       [-0.57153678,  1.39689885,  0.07124181, ..., -0.63665676,
        -0.04770982, -0.22312806],
       [-0.33987988, -0.78523283, -1.28617409, ...,  1.57070507,
        -0.04770982, -0.22312806]])
```

In [51]:
```
X_test
```
Out[51]:
```
array([[ 0.12343392, -0.78523283,  1.5417757 , ..., -0.63665676,
        -0.04770982, -0.22312806],
       [-0.64872193,  1.39689885,  0.63658407, ...,  1.57070507,
        -0.04770982, -0.22312806],
       [-0.10822298, -0.78523283,  0.29747779, ..., -0.63665676,
        -0.04770982, -0.22312806],
       ...,
       [-0.64875575, -0.78523283,  1.20642142, ...,  1.57070507,
        -0.04770982, -0.22312806],
       [-0.57153678, -0.78523283,  1.6418879 , ..., -0.63665676,
        -0.04770982, -0.22312806],
       [ 0.35509082,  1.39689885,  1.76801168, ..., -0.63665676,
        -0.04770982, -0.22312806]])
```

In [52]:
```
X_train.shape,X_test.shape,Y_train.shape,Y_test.shape
```
Out[52]:
```
((14971, 16), (3743, 16), (14971,), (3743,))
```

In [53]:
```
Y_test.value_counts(normalize=True)*100
```
Out[53]:
```
1.0    50.520972
0.0    49.479028
Name: Churn, dtype: float64
```

In [54]:
```
Y_train.value_counts(normalize=True)*100
```
Out[54]:
```
0.0    50.130252
1.0    49.869748
Name: Churn, dtype: float64
```

In [59]:
```
model=LogisticRegression(max_iter=1000)
model.fit(X_train,Y_train)
Y_predict=model.predict(X_test)
type(model)
```

Predicting on Training and Test dataset

In [67]:
```
ytest_predict
ytest_predict_prob=best_grid.predict_proba(X_test)
ytest_predict_prob
pd.DataFrame(ytest_predict_prob).head()
```

Out[67]:

	0	1
0	0.877323	0.122677
1	0.061856	0.938144
2	0.873786	0.126214
3	0.219251	0.780749
4	0.019841	0.980159

Model Evaluation

Confusion Matrix for the training data

In [68]:
```
confusion_matrix(Y_train, ytrain_predict)
```

Out[68]:
```
array([[6583,  922],
       [1288, 6178]], dtype=int64)
```

In [69]:
```
#Train Data Accuracy
cart_train_acc=best_grid.score(X_train,Y_train)
cart_train_acc
```

Out[69]: 0.8523812704562154

In [70]:
```
print(classification_report(Y_train, ytrain_predict))
```

```
              precision    recall  f1-score   support

         0.0       0.84      0.88      0.86      7505
         1.0       0.87      0.83      0.85      7466

    accuracy                           0.85     14971
   macro avg       0.85      0.85      0.85     14971
weighted avg       0.85      0.85      0.85     14971
```

In [71]:
```
cart_metrics=classification_report(Y_train, ytrain_predict,output_dict=True)
cf1=pd.DataFrame(cart_metrics).transpose()
cart_train_precision=round(cf1.loc["1.0"][0],2)
cart_train_recall=round(cf1.loc["1.0"][1],2)
cart_train_f1=round(cf1.loc["1.0"][2],2)
print ('cart_train_precision ',cart_train_precision)
print ('cart_train_recall ',cart_train_recall)
print ('cart_train_f1 ',cart_train_f1)
```

```
cart_train_precision  0.87
cart_train_recall  0.83
cart_train_f1  0.85
```

Precision, Recall, and F1 Score in Brief

- Precision:

  The ratio of accurately predicted positive observations to the total number of predicted positives is known as precision.
  It gauges how accurate positive predictions are, which is crucial when the expense of false positives is substantial.

- Recall:

  The ratio of accurately predicted positive observations to all actual positive observations is known as recall.
  It gauges how well the model can find every pertinent instance—a critical function

when it comes to avoiding costly false positives.

- F1 Points:

  The F1 score strikes a balance between recall and precision by taking the harmonic mean of the two.
  It offers a solitary statistic for assessing the model in situations where recall and precision are equally significant.

Out[116]:

	CART Train	CART Test	Random forest train	Neural Network train	Neural Network test
Accuracy	0.85	0.85	0.88	0.88	0.87
Recall	0.83	0.84	0.90	0.90	0.89
Precision	0.87	0.87	0.86	0.86	0.87
f1_Score	0.85	0.85	0.88	0.88	0.88

CONCLUSION

Based on the provided output from the dataset, here is an interpretation of the accuracy scores for different models:

1. Decision Tree (CART) Model:

- Train Data Accuracy: The decision tree model achieved a training accuracy of approximately 86.96% (0.8696). This indicates that the model correctly predicted the churn outcome for about 86.96% of the training data.

2. Random Forest Model:

- Train Data Accuracy: The random forest model achieved a training accuracy of around 85.76% (0.8576). This suggests that the model correctly predicted the churn outcome for approximately 85.76% of the training data.

3. Neural Network Model:

- Train Data Accuracy: The neural network model achieved a training accuracy of approximately 87.73% (0.8773). This indicates that the model correctly predicted the churn outcome for around 87.73% of the training data.

- Test Data Accuracy: The neural network model achieved a test accuracy of approximately 87.54% (0.8754). This suggests that the model correctly predicted the churn outcome for approximately 87.54% of the test data.

Based on above observation our data set is good fit and good model

Based on these results, the neural network model seems to have the highest accuracy both on the training and test data, indicating its better performance compared to the decision tree and random forest models. However, it is important to consider other factors such as model complexity, interpretability, and potential overfitting when selecting the best model for your specific churn prediction task.