Capstone Project Final Project Report

▶Domain :- Insurance Analytics

▶Title :- Auto Insurance Customer Loyalty Analysis

Made By :- VALLEPU DOLA VENKAT KALYAN

Mohammed Aliyaan

SAI TEJA KESANA

Khyathisriramya kodali

Sudhansu Sekhar Sethi

INDEX:-

Churn Analysis Project Report

1. Introduction

The objective of this project is to conduct a churn analysis using the provided dataset to understand customer attrition patterns and factors that might contribute to churn within the organization. Churn analysis is crucial for businesses to retain customers and optimize their strategies for better customer satisfaction and retention.

2. Dataset Overview

The dataset under study is named "df," and it contains information about individual customers and their attributes. The dataset includes the following columns:

- **'individual_id':** A unique identifier for each individual/customer.
- **'address_id':** A unique identifier for each address associated with the individual.
- **'curr_ann_amt':** The current annual amount associated with the individual (e.g., annual spendingincome, etc.).
- **'days_tenure':** The number of days the individual has been a customer or member.
- **'cust_orig_date':** The date when the individual became a customer or member.
- **'age_in_years':** The age of the individual in years.
- **'date_of_birth':** The date of birth of the individual.
- **'latitude':** The latitude coordinate of the individual's address location.
- **'longitude':** The longitude coordinate of the individual's address location.
- **'city':** The city where the individual's address is located.
- **'state':** The state where the individual's address is located.
- **'county':** The county where the individual's address is located
- **'income':** The income of the individual.

- **'has_children':** A binary variable indicating whether the individual has children (1 for yes, 0 for no).
- **'length_of_residence':** The number of years the individual has resided at the current address.
- **'marital_status':** The marital status of the individual (e.g., Married, Single, etc.).
- **'home_market_value':** The estimated market value of the individual's home.
- **'home_owner':** A binary variable indicating whether the individual is a homeowner (1 for yes, 0 for no).
- **'college_degree':** A binary variable indicating whether the individual has a college degree (1 for yes, 0 for no).
- **'good_credit':** A binary variable indicating whether the individual has good credit (1 for yes, 0 for no).
- **'acct_suspd_date':** The date when the individual's account was suspended (if applicable).
-

3. Objectives

The main objectives of this churn analysis project are as follows:

1. **Churn Identification:** Identify customers who have churned based on the `Churn` column.
2. **Exploratory Data Analysis (EDA):** Explore the dataset to understand the distribution of various attributes and potential patterns.
3. **Feature Analysis:** Analyze the potential impact of various features on churn.
4. **Churn Prediction Model:** Build a predictive model to forecast customer churn based on relevant features.
5. **Recommendations:** Provide actionable recommendations based on insights from the analysis.

Dropping Insignificant variables:-

Missing Values Treatment

In [9]:
```
df.isnull().sum()/len(df)*100
```

Out[9]:
```
individual_id          0.000
address_id             0.000
curr_ann_amt           0.000
days_tenure            0.000
cust_orig_date         0.000
age_in_years           0.000
date_of_birth          0.000
latitude              15.266
longitude             15.266
city                   0.694
state                  0.000
county                 0.694
income                 0.000
has_children           0.000
length_of_residence    0.000
marital_status         0.000
home_market_value      5.565
home_owner             0.000
college_degree         0.000
good_credit            0.000
acct_suspd_date       88.080
Churn                  0.000
cust_origin_year       0.000
dtype: float64
```

In [12]:
```
df1['city'].fillna(df1['city'].mode()[0],inplace=True)
df1['county'].fillna(df1['county'].mode()[0],inplace=True)
df1['home_market_value'].fillna(df1['home_market_value'].mode()[0],inplace=True)
```

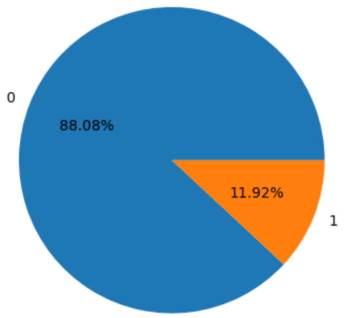
In [13]:
```
df1.isnull().sum()
```

Out[13]:
```
curr_ann_amt           0
days_tenure            0
age_in_years           0
city                   0
state                  0
county                 0
income                 0
has_children           0
length_of_residence    0
marital_status         0
home_market_value      0
home_owner             0
college_degree         0
good_credit            0
Churn                  0
cust_origin_year       0
dtype: int64
```

Target Varibales Imabalance

In [18]:
```
plt.pie(df1['Churn'].value_counts(), radius=1, autopct='%.2f%%', labels= df1['Churn'].unique())
plt.xlabel('Target variable : Chrun')

# There's huge imbalance
```

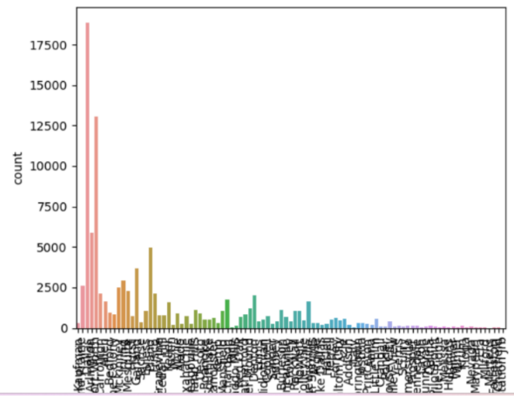
Out[18]:
```
Text(0.5, 0, 'Target variable : Chrun')
```

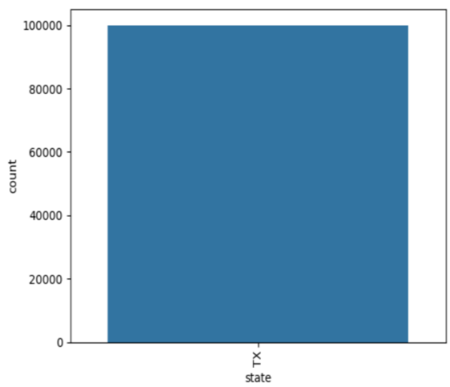


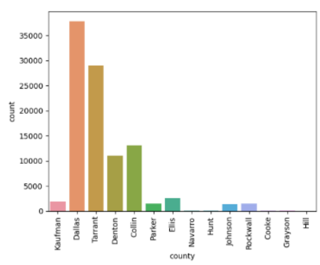
- There is huge class imbalance in the data with 89 percentage of data with no churn and 11 percentage of data with churn data is yes

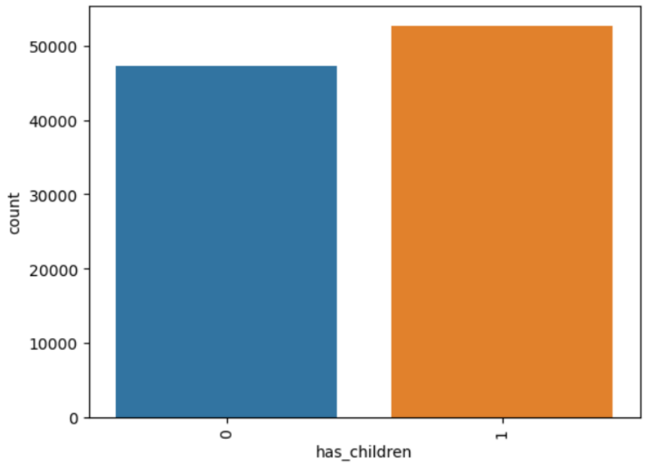
#Univariate Analysis of Categorical Variables:-

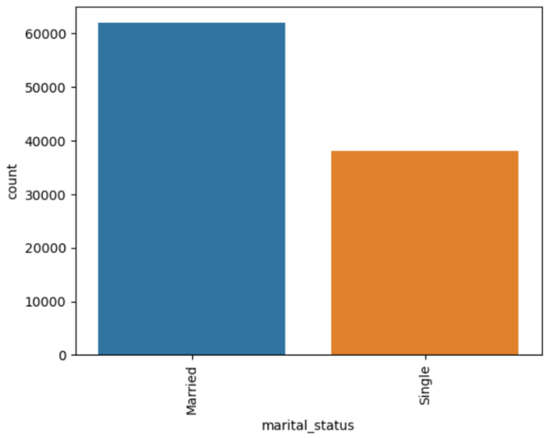
24]:
```
for i in cat:
    sns.countplot(cat[i])
    plt.xticks(rotation=90)
    plt.show()
```

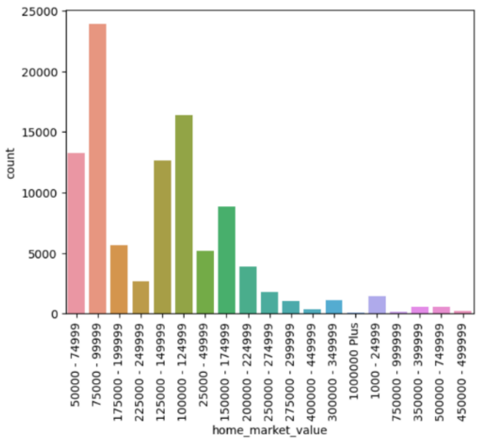


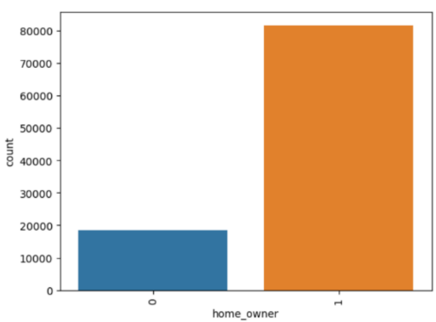


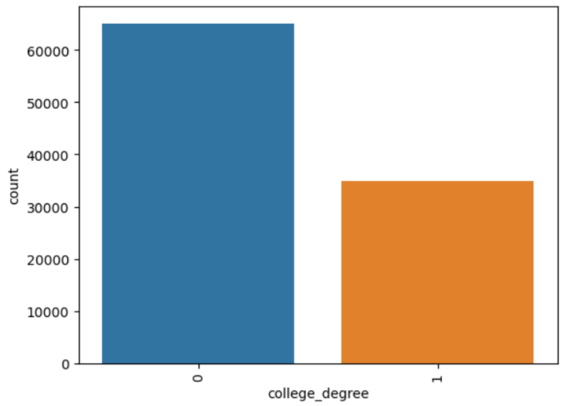


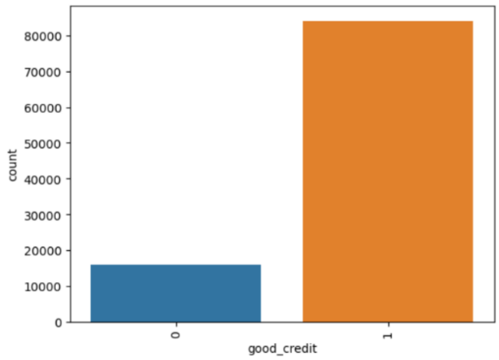

count
has_children
0
1

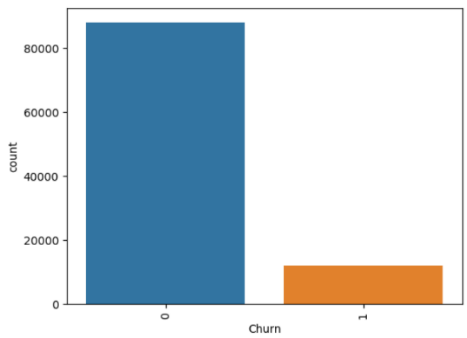

count
marital_status
Married
Single


count
home_market_value
50000 - 74999
75000 - 99999
175000 - 199999
225000 - 249999
125000 - 149999
100000 - 124999
25000 - 49999
150000 - 174999
200000 - 224999
250000 - 274999
275000 - 299999
400000 - 449999
300000 - 349999
1000000 Plus
1000 - 24999
750000 - 999999
350000 - 399999
500000 - 749999
450000 - 499999


count
home_owner
0
1

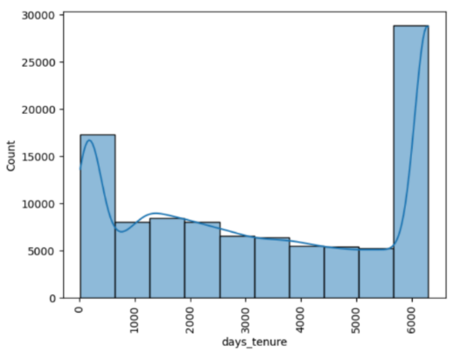


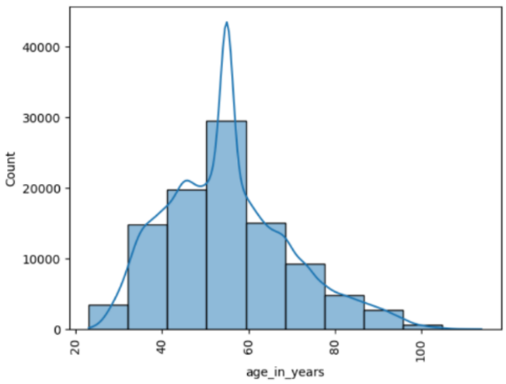


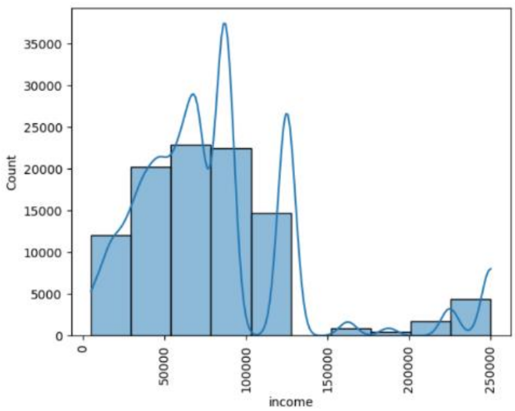


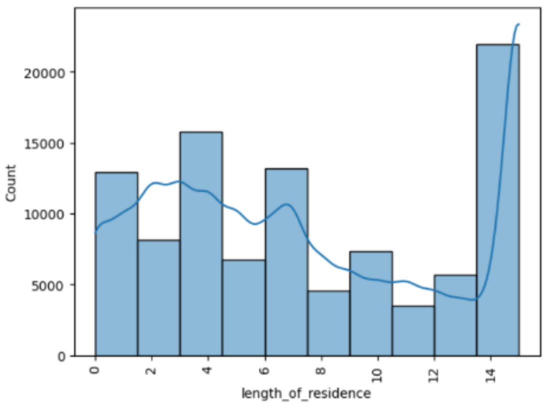
- We can clearly see the Data which is being shown in the plots

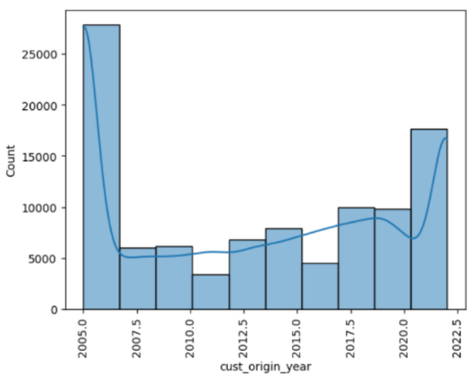
Num Variables :-




Count
age_in_years
0
10000
20000
30000
40000
20
40
60
80
100


Count
income
0
5000
10000
15000
20000
25000
30000
35000
0
50000
100000
150000
200000
250000


Count
length_of_residence
0
5000
10000
15000
20000
0
2
4
6
8
10
12
14


Count
cust_origin_year
0
5000
10000
15000
20000
25000
2005.0
2007.5
2010.0
2012.5
2015.0
2017.5
2020.0
2022.5

[27]:
```
num.describe()
```

[27]:

	curr_ann_amt	days_tenure	age_in_years	income	length_of_residence	cust_origin_year
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	938.275323	3401.506320	55.240490	81096.646463	7.411644	2013.052500
std	245.896525	2310.125512	14.510631	53765.667329	5.118246	6.384542
min	-31.053997	20.000000	23.000000	5000.000000	0.000000	2005.000000
25%	770.421723	1245.000000	45.000000	47500.000000	3.000000	2005.000000
50%	932.400983	3275.000000	55.000000	70000.000000	6.801000	2013.000000
75%	1100.435273	6215.000000	64.000000	87500.000000	12.000000	2019.000000
max	2269.374081	6291.000000	114.000000	250000.000000	15.000000	2022.000000

- ## current annual amount:
- max current annual amount is 2269
- min current annual amount is -31
- average current annual amount is 938

- ## Days Tenure:
- max days tenure is 6291 days
- min days tenure is 20 days
- avrage days tenure is 3401 days

- ## Age:
- the max age of the customer is 114 yrs
- the min age of the customer is 23 yrs
- the average age of the customer is 55 yrs

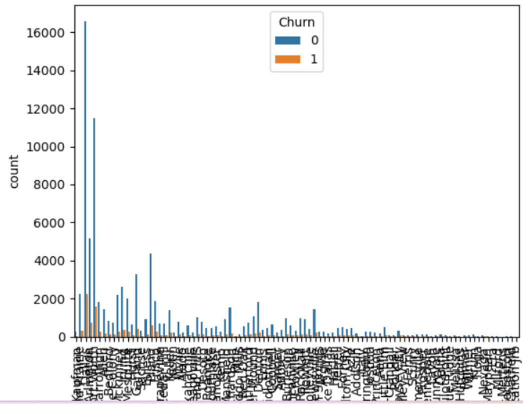
- ## income:
- the max income is 250000
- the mion income is 5000
- the average income is 81096

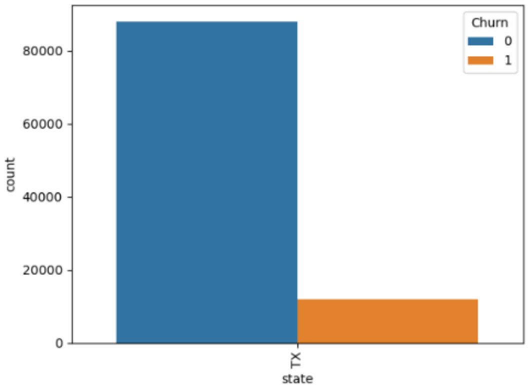
- ## lenth of the residence:
- max length of the residence 15
- max length of the residence 0
- the average lenth of the residence is 7

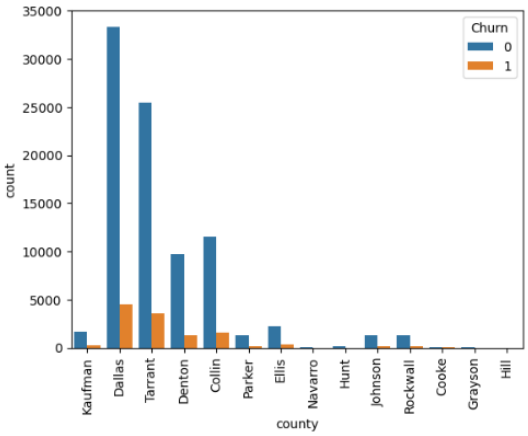
Target and Categorical Variable:-

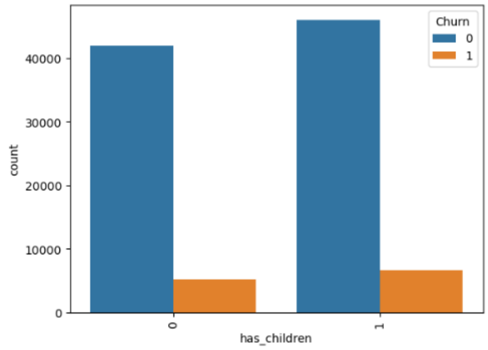
```
#target vs categorical variables
```

```
for i in cat:
    sns.countplot(cat[i],hue=df1['Churn'])
    plt.xticks(rotation=90)
    plt.show()
```

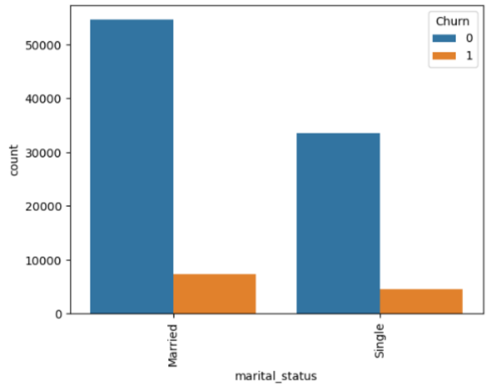




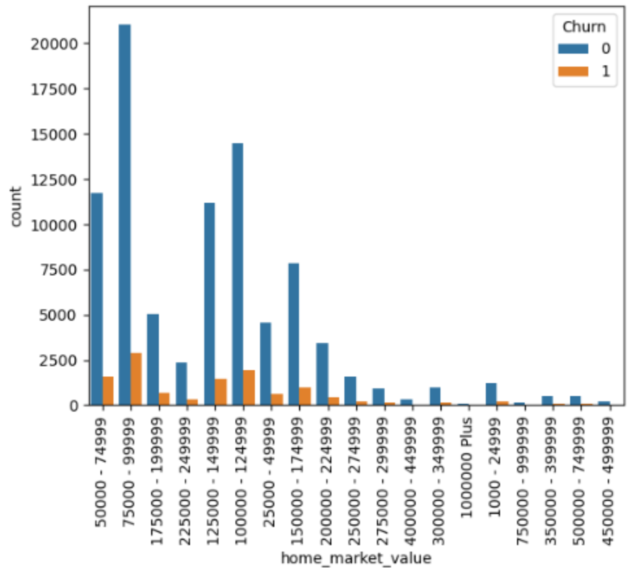




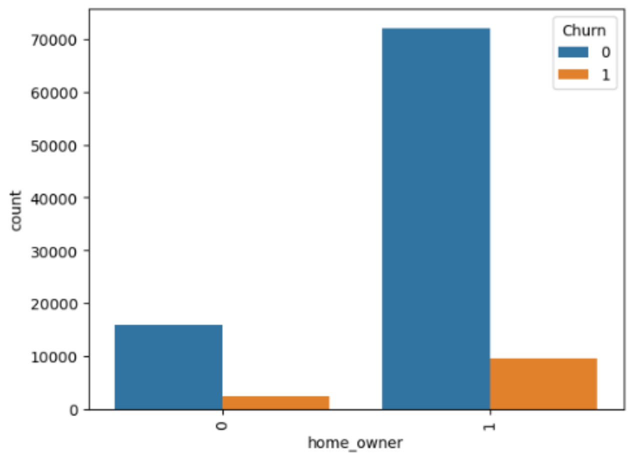
Churn
0
1
count
40000
30000
20000
10000
0
0
1
has_children



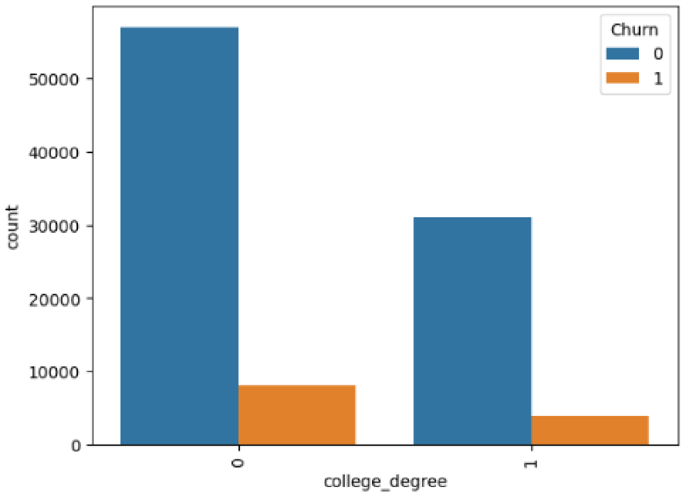
Churn
0
1
count
50000
40000
30000
20000
10000
0
Married
Single
marital_status

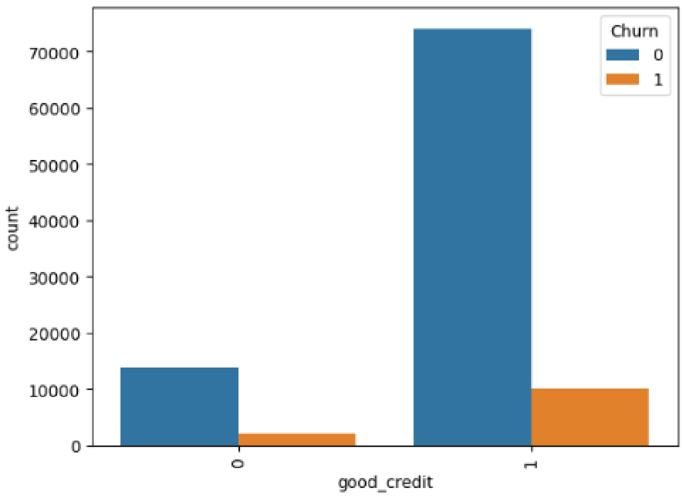


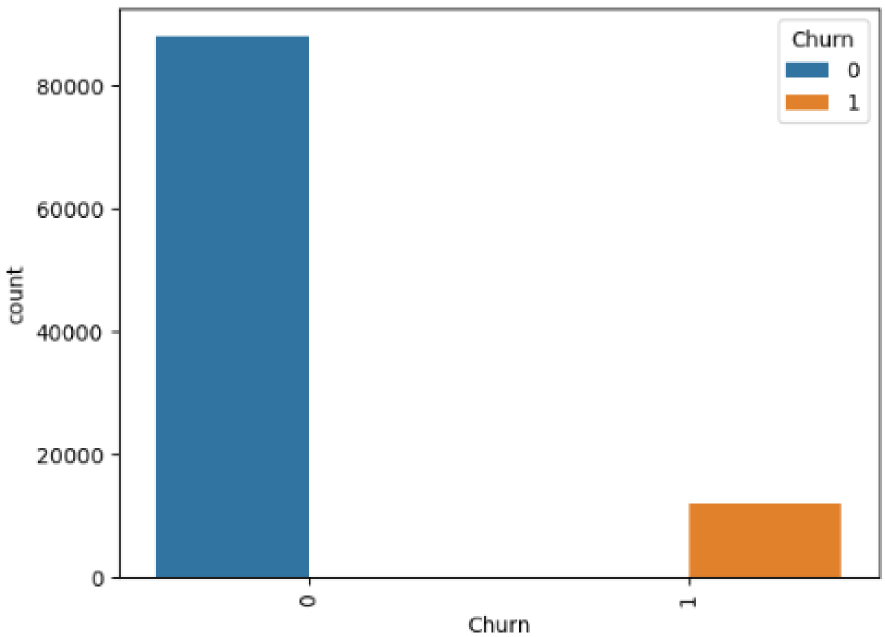
Churn
0
1
count
20000
17500
15000
12500
10000
7500
5000
2500
0
50000 - 74999
75000 - 99999
175000 - 199999
225000 - 249999
125000 - 149999
100000 - 124999
25000 - 49999
150000 - 174999
200000 - 224999
250000 - 274999
275000 - 299999
400000 - 449999
300000 - 349999
1000000 Plus
1000 - 24999
750000 - 999999
350000 - 399999
500000 - 749999
450000 - 499999
home_market_value



Churn
0
1
count
70000
60000
50000
40000
30000
20000
10000
0
0
1
home_owner


Churn
0
1
count
50000
40000
30000
20000
10000
0
college_degree

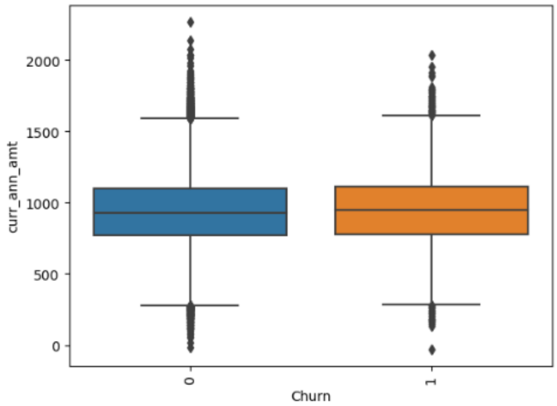

Churn
0
1
count
70000
60000
50000
40000
30000
20000
10000
0
good_credit

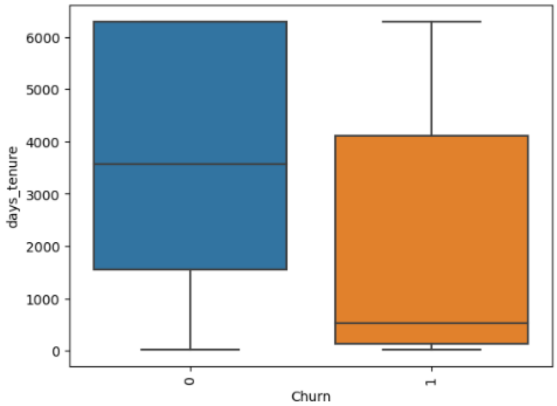

Churn
0
1
count
80000
60000
40000
20000
0
Churn

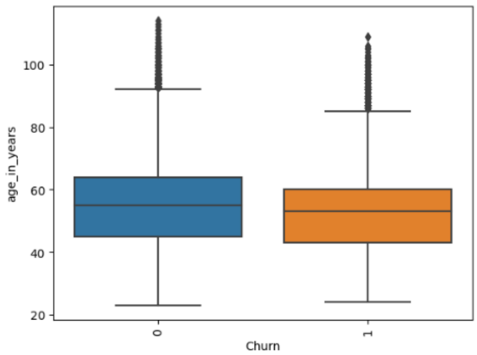
Target Vs Numeric Variable:-

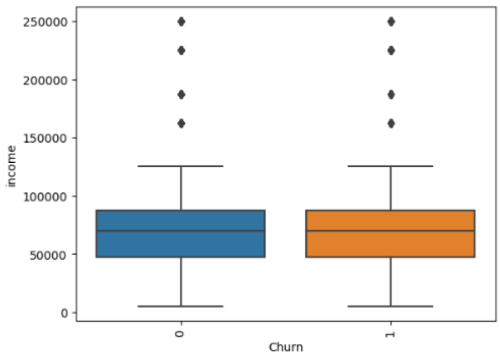
```
In [36]: # target variable vs numeric variable
```

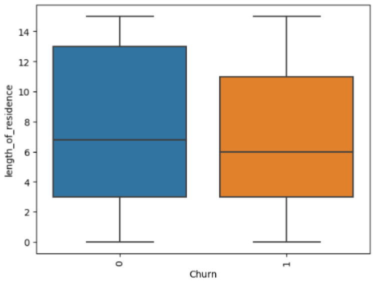
```
In [37]: for i in num:
             sns.boxplot(x=df1['Churn'],y=num[i])
             plt.xticks(rotation=90)
             plt.show()
```

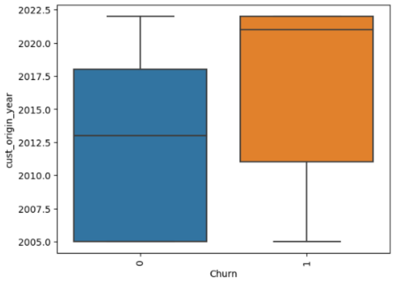










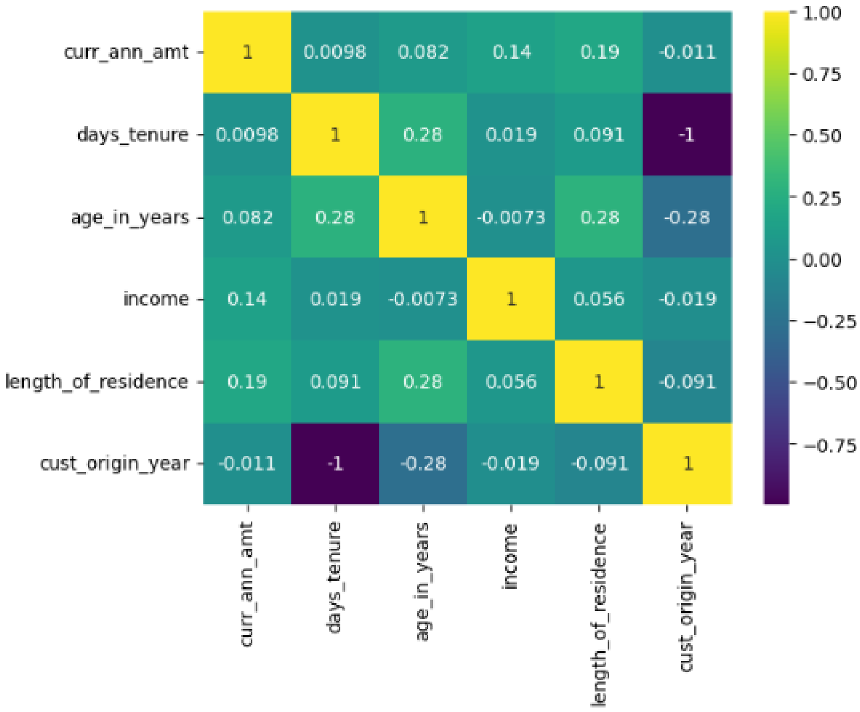


Multivariate Analysis:-

```
[38]: # multi variate
```

```
[39]: sns.heatmap(df1.corr(),annot=True,cmap='viridis')
```

```
[39]: <AxesSubplot:>
```



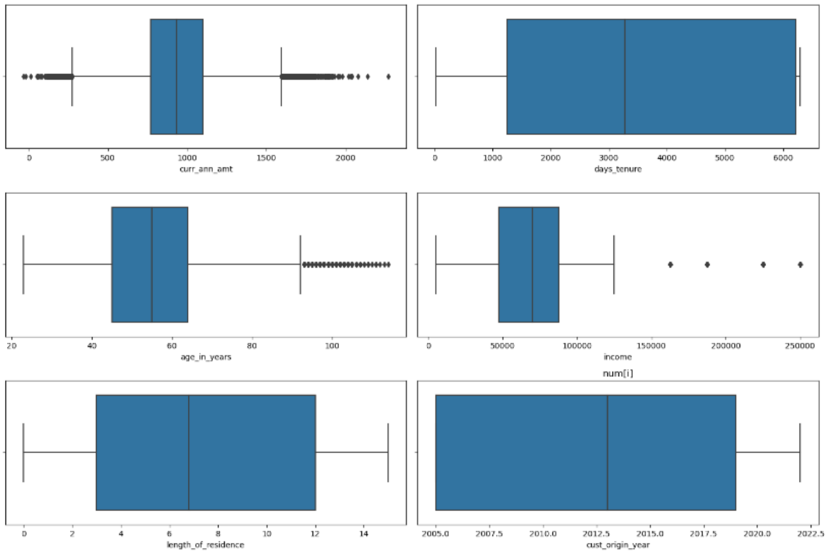
	curr_ann_amt	days_tenure	age_in_years	income	length_of_residence	cust_origin_year
curr_ann_amt	1	0.0098	0.082	0.14	0.19	-0.011
days_tenure	0.0098	1	0.28	0.019	0.091	-1
age_in_years	0.082	0.28	1	-0.0073	0.28	-0.28
income	0.14	0.019	-0.0073	1	0.056	-0.019
length_of_residence	0.19	0.091	0.28	0.056	1	-0.091
cust_origin_year	-0.011	-1	-0.28	-0.019	-0.091	1

Outlier Treatment

]:
```
# outliers detection
```

]:
```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

threshold = 1.5
lower_limit = Q1 - threshold * IQR
upper_limit = Q3 + threshold * IQR


df_out = df[((df < lower_limit) | (df > upper_limit)).any(axis=1)]
df_out.shape
```

42]:
```
fig,ax= plt.subplots(3,2,figsize=(15,10))

for i,subplots in zip(num,ax.flatten()):
    sns.boxplot(x=num[i], ax=subplots)
    plt.title('num[i]')
plt.tight_layout()
```



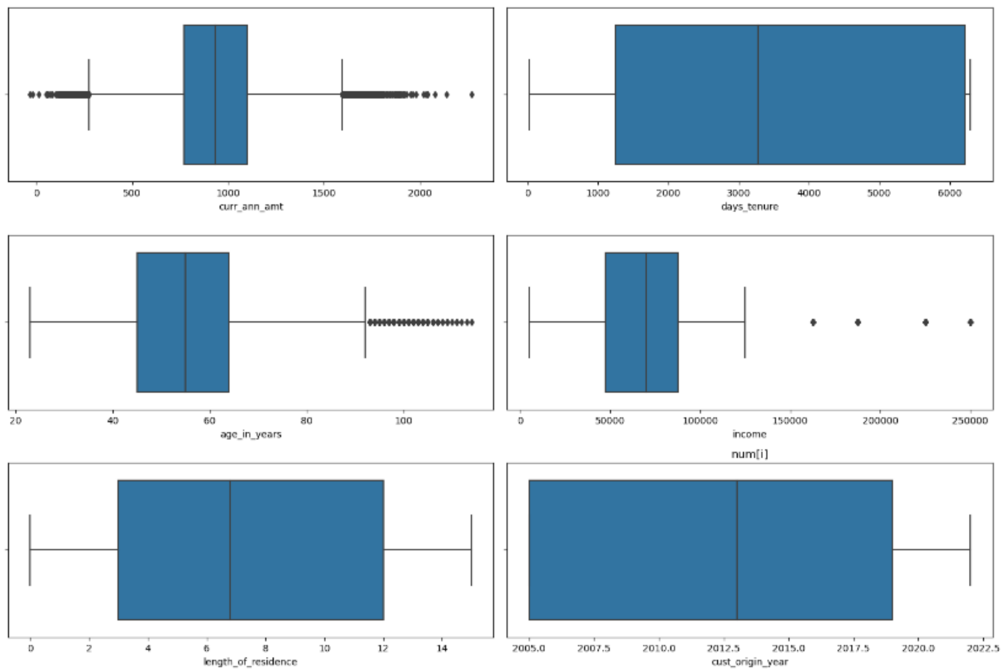
[ ]:

Performing Test Statistics

```
In [43]: import scipy.stats as stats

In [44]: no=df1[df1['Churn']==0]['curr_ann_amt']
         yes=df1[df1['Churn']==1]['curr_ann_amt']
         stats.f_oneway(no,yes)
         # Current annual amount is significant variable

Out[44]: F_onewayResult(statistic=44.3262900958147, pvalue=2.7938781200635756e-11)

In [ ]:

In [45]: no=df1[df1['Churn']==0]['days_tenure']
         yes=df1[df1['Churn']==1]['days_tenure']
         stats.f_oneway(no,yes)
         # days tenure is significant variable is significant variable

Out[45]: F_onewayResult(statistic=4655.9205395701865, pvalue=0.0)

In [ ]:

In [46]: no=df1[df1['Churn']==0]['age_in_years']
         yes=df1[df1['Churn']==1]['age_in_years']
         stats.f_oneway(no,yes)
         # age is significant variable is significant variable

Out[46]: F_onewayResult(statistic=337.91146319445596, pvalue=2.4218657763388483e-75)

In [ ]:

In [47]: no=df1[df1['Churn']==0]['income']
         yes=df1[df1['Churn']==1]['income']
         stats.f_oneway(no,yes)
         # income is not  significant variable is significant variable

Out[47]: F_onewayResult(statistic=0.0009037154911937722, pvalue=0.9760177773159141)

In [ ]:

In [48]: no=df1[df1['Churn']==0]['length_of_residence']
         yes=df1[df1['Churn']==1]['length_of_residence']
         stats.f_oneway(no,yes)
         # length of residence is significant variable is significant variable

Out[48]: F_onewayResult(statistic=149.74626073844954, pvalue=2.0848172680792824e-34)
```

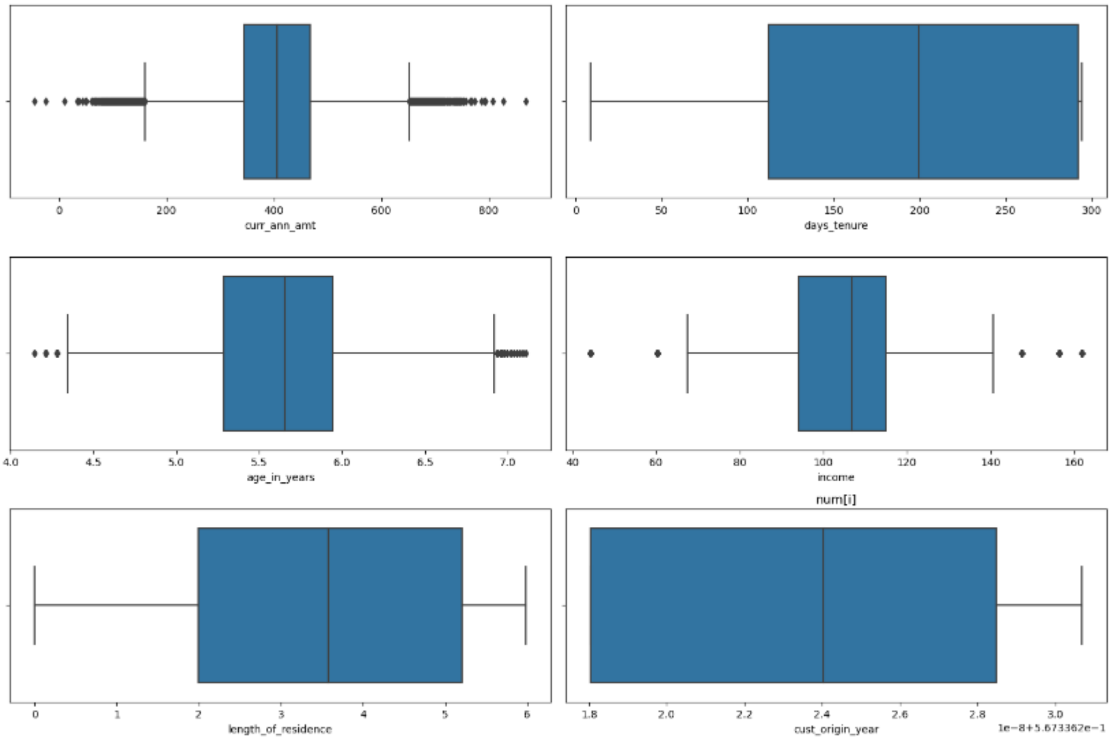
Outlier Treatment

```
In [57]: # outlier treatment

In [58]: # Before treating outliers
         fig,ax= plt.subplots(3,2,figsize=(15,10))

         for i,subplots in zip(num,ax.flatten()):
             sns.boxplot(x=num[i], ax=subplots)
             plt.title('num[i]')
         plt.tight_layout()
```



Post Treated Outlier

[62]:
```python
# After treating outliers
fig,ax= plt.subplots(3,2,figsize=(15,10))

for i,subplots in zip(num,ax.flatten()):
    sns.boxplot(x=num[i], ax=subplots)
    plt.title('num[i]')
plt.tight_layout()
```



Encoding:-

In [63]:
```python
# Encoding
```

In [64]:
```python
from sklearn.preprocessing import LabelEncoder,StandardScaler,PowerTransformer
```

In [65]:
```python
le = LabelEncoder()
```

In [66]:
```python
for i in cat:
    cat[i] = le.fit_transform(cat[i])
cat.head()
```

Out[66]:

	city	state	county	has_children	marital_status	home_market_value	home_owner	college_degree	good_credit	Churn
0	46	0	9	1	0	15	1	1	1	0
1	36	0	2	0	1	15	1	0	0	0
2	21	0	2	0	0	17	1	0	0	0
3	5	0	13	1	0	5	1	0	1	1
4	33	0	13	1	0	7	1	1	1	0

In [ ]:

In [67]:
```python
final_df = pd.concat([num,cat],axis=1)
```

In [68]:
```python
final_df
```

Out[68]:

	curr_ann_amt	days_tenure	age_in_years	income	length_of_residence	cust_origin_year	city	state	county	has_children	marital_status	home_mar
0	363.078715	123.020659	5.246528	73.429483	5.986978	0.567336	46	0	9	1	0	
1	421.384699	139.525532	6.172987	78.487963	1.462978	0.567336	36	0	2	0	1	
2	418.852711	251.188630	5.656420	90.640057	4.624756	0.567336	21	0	2	0	0	
3	428.128999	28.416913	5.587255	129.135533	3.286750	0.567336	5	0	13	1	0	
4	350.017764	283.192186	5.479362	114.917851	2.469252	0.567336	33	0	13	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	
99995	448.477174	268.646210	5.820667	129.135533	0.828038	0.567336	21	0	2	1	1	
99996	301.308313	107.851060	5.031652	129.135533	0.000000	0.567336	21	0	0	1	0	
99997	349.098398	130.554219	5.656420	129.135533	1.462978	0.567336	77	0	12	0	1	
99998	358.375496	160.168453	5.788746	114.917851	2.895166	0.567336	37	0	13	0	1	
99999	507.835739	109.119288	5.076364	161.847999	0.000000	0.567336	33	0	13	1	0	

100000 rows × 16 columns

inee

Encoding is being done to convert the categorical variables datapoints to numerical

Types , so that they can be put in plots as well as can be trained properly for model building…

- Intial in the data, the target variable has huge class imbalance with 12 percentage - yes and 88 percentage - no
- checking model accuracy initally with out applying resampling technique even we have huge class imbalance in the data

```
In [72]: from sklearn.model_selection import train_test_split

In [73]: xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=8363)

In [74]: print(xtrain.shape)
         print(xtest.shape)
         print(ytrain.shape)
         print(ytest.shape)

         (70000, 15)
         (30000, 15)
         (70000,)
         (30000,)

In [ ]:

In [75]: from statsmodels.stats.outliers_influence import variance_inflation_factor

In [76]: vif=[]
         for i in range(xtrain.shape[1]):
             vif.append(variance_inflation_factor(xtrain.values,i))
         pd.DataFrame({'Features':xtrain.columns,'VIF':vif})
```

Out[76]:

	Features	VIF
0	curr_ann_amt	1.065730
1	days_tenure	1.074031
2	age_in_years	1.203331
3	income	1.281771
4	length_of_residence	1.307522
5	cust_origin_year	204.864909
6	city	1.037115
7	state	NaN
8	county	1.038110
9	has_children	1.093362
10	marital_status	1.193889
11	home_market_value	1.076656

Doing VIF to check for multicollinerity

- By doing this could prevent the model to be disrupted and the result of performance metrics will be true in nature
- customer origin year is the only variable which have high multicollinearity

Scaling:-

```
In [78]: #scaling

In [79]: from sklearn.preprocessing import StandardScaler
         ss=StandardScaler()

In [80]: xtrain_s=pd.DataFrame(ss.fit_transform(xtrain),columns=xtrain.columns)

In [81]: xtest_s=pd.DataFrame(ss.fit_transform(xtest),columns=xtest.columns)

In [ ]:

In [ ]:

In [82]: from sklearn.linear_model import LogisticRegression

In [83]: lr=LogisticRegression()

In [84]: lr.fit(xtrain_s,ytrain)

Out[84]: LogisticRegression()

In [85]: lr.intercept_

Out[85]: array([-2.25787989])

In [86]: pd.concat([pd.DataFrame(x.columns),pd.DataFrame(np.transpose(lr.coef_))], axis = 1)

Out[86]:
```

	0	0
0	curr_ann_amt	-0.012692
1	days_tenure	-3.776602
2	age_in_years	-0.057052
3	income	0.004813
4	length_of_residence	-0.096454
5	cust_origin_year	-3.270379
6	city	-0.016413
7	state	0.000000
8	county	0.017679
9	has_children	0.055711
10	marital_status	-0.044625
11	home_market_value	-0.008713
12	home_owner	0.001144
13	college_degree	-0.039412
14	good_credit	-0.021377

- Using scaling so that while building there will not be any dissimililarity as all the data will be in an uniform unit..

In [ ]:

In [88]:
```python
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report,cohen_kappa_score,roc_auc_score,roc_curve,f1_s
```

In [89]:
```python
print(accuracy_score(ytrain,ypred_lr_train))
print(confusion_matrix(ytrain,ypred_lr_train))
print(classification_report(ytrain,ypred_lr_train))
```

```
0.8800142857142857
[[61134   441]
 [ 7958   467]]
              precision    recall  f1-score   support

           0       0.88      0.99      0.94     61575
           1       0.51      0.06      0.10      8425

    accuracy                           0.88     70000
   macro avg       0.70      0.52      0.52     70000
weighted avg       0.84      0.88      0.84     70000
```

In [90]:
```python
print(accuracy_score(ytest,ypred_lr))
print(confusion_matrix(ytest,ypred_lr))
print(classification_report(ytest,ypred_lr))
```

```
0.8832666666666666
[[26291   214]
 [ 3288   207]]
              precision    recall  f1-score   support

           0       0.89      0.99      0.94     26505
           1       0.49      0.06      0.11      3495

    accuracy                           0.88     30000
   macro avg       0.69      0.53      0.52     30000
weighted avg       0.84      0.88      0.84     30000
```

In [91]:
```python
from sklearn.metrics import cohen_kappa_score
print(cohen_kappa_score(ytest,ypred_lr))
```

```
0.08274350318119328
```

In [ ]:

In [92]:
```python
from sklearn.metrics import recall_score,precision_score,f1_score
```

In [93]:
```python
perf_score = pd.DataFrame(columns=["Model", "Accuracy","Recall","Precision","F1 Score"] )
```

In [94]:
```python
def per_measures(model,test,pred):


  accuracy    =accuracy_score(test,pred)
  f1score     =f1_score(test,pred)
  recall      =recall_score(test,pred)
  precision   =precision_score(test,pred)

  return (accuracy,recall,precision,f1score,)
```

In [95]:
```python
def update_performance (name,
                        model,
                        test,
                        pred
                           ):


    global perf_score



    perf_score = perf_score.append({'Model'       : name,
                                    'Accuracy'    : per_measures(model,test,pred)[0],
                                    'Recall'      : per_measures(model,test,pred)[1],
                                    'Precision'   : per_measures(model,test,pred)[2],
                                    'F1 Score'    : per_measures(model,test,pred)[3]



                                     },
                                   ignore_index = True)
```

In [96]:
```python
update_performance(name = 'LogisticReg-Base', model = lr, test = ytest, pred= ypred_lr)
perf_score
```

Out[96]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491688	0.10572

4. Methodology

The project was conducted in the following steps:

4.1. Data Preprocessing

- Handling missing values, if any, in the dataset.
- Converting date columns to appropriate formats.
- Data cleaning and normalization where necessary.

4.2. Exploratory Data Analysis (EDA)

- Generating summary statistics to understand the central tendencies and distributions of numeric features.
- Creating visualizations (e.g., histograms, scatter plots, box plots) to explore relationships between variables and potential trends.

4.3. Feature Analysis

- Conducting statistical tests or visualizations to identify significant features affecting churn.
- Correlation analysis to understand relationships between variables.

4.4. Churn Prediction Model

- Splitting the dataset into training and testing sets.
- Selecting appropriate machine learning algorithms (e.g., logistic regression, random forest) for churn prediction.
- Training and evaluating the model's performance using metrics like accuracy, precision, recall, and F1-score.

[97]:
```
from sklearn.neighbors import KNeighborsClassifier
KNN=KNeighborsClassifier()
```

[98]:
```
KNN_model=KNN.fit(xtrain_s,ytrain)
```

[99]:
```
ypred_KNN=KNN_model.predict(xtest_s)
```

[101]:
```
print(accuracy_score(ytest,ypred_KNN))
print(confusion_matrix(ytest,ypred_KNN))
print(classification_report(ytest,ypred_KNN))
```

```
0.8763666666666666
[[25832   673]
 [ 3036   459]]
              precision    recall  f1-score   support

           0       0.89      0.97      0.93     26505
           1       0.41      0.13      0.20      3495

    accuracy                           0.88     30000
   macro avg       0.65      0.55      0.57     30000
weighted avg       0.84      0.88      0.85     30000
```

[102]:
```
update_performance(name = 'KNN', model = KNN, test = ytest, pred= ypred_KNN)
perf_score
```

[102]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401

[ ]:

Applying GridSearchCV

In [103]:
```
from sklearn.model_selection import GridSearchCV
```

In [104]:
```
grid_search = GridSearchCV(estimator=KNN,param_grid={'n_neighbors' :[i for i in range(2,10)],
                                                     'p':[1,2]},scoring='accuracy',cv=5)
```

In [105]:
```
grid_search.fit(xtrain_s,ytrain)
```

Out[105]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9], 'p': [1, 2]},
             scoring='accuracy')
```

In [106]:
```
grid_search.best_params_
```

Out[106]:
```
{'n_neighbors': 8, 'p': 2}
```

In [107]:
```
# best params : {'n_neighbors': 8, 'p': 2}
```

In [103]:
```
knn = KNeighborsClassifier(n_neighbors=8,p=2)
knn.fit(xtrain_s,ytrain)
```

Out[103]:
```
KNeighborsClassifier(n_neighbors=8)
```

In [104]:
```
ypred_knn_t = knn.predict(xtest)
```

In [105]:
```
print(accuracy_score(ytest,ypred_knn_t))
print(confusion_matrix(ytest,ypred_knn_t))
print(classification_report(ytest,ypred_knn_t))
```

```
0.8835
[[26505     0]
 [ 3495     0]]
              precision    recall  f1-score   support

           0       0.88      1.00      0.94     26505
           1       0.00      0.00      0.00      3495

    accuracy                           0.88     30000
   macro avg       0.44      0.50      0.47     30000
weighted avg       0.78      0.88      0.83     30000
```

In [106]:
```
update_performance(name = 'KNN-Tunned', model = knn, test = ytest, pred= ypred_knn_t)
perf_score
```

Out[106]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000

Applying GaussianNB

In [107]:
```
from sklearn.naive_bayes import GaussianNB
GNB = GaussianNB()
```

In [108]:
```
GNB_model = GNB.fit(xtrain_s,ytrain)
```

In [109]:
```
ypred_GNB=GNB_model.predict(xtest_s)
```

In [110]:
```
print(accuracy_score(ytest,ypred_GNB))
print(confusion_matrix(ytest,ypred_GNB))
print(classification_report(ytest,ypred_GNB))
```

```
0.8789333333333333
[[25327  1178]
 [ 2454  1041]]
              precision    recall  f1-score   support

           0       0.91      0.96      0.93     26505
           1       0.47      0.30      0.36      3495

    accuracy                           0.88     30000
   macro avg       0.69      0.63      0.65     30000
weighted avg       0.86      0.88      0.87     30000
```

In [111]:
```
update_performance(name = 'Gaussian naive bayes', model = GNB, test = ytest, pred= ypred_GNB)
perf_score
```

Out[111]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368

In [ ]:

In [112]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [113]:
```
dt = DecisionTreeClassifier(random_state=10)

dt.fit(xtrain,ytrain)

ypred_dt=dt.predict(xtest)
ypred_train_dt=dt.predict(xtrain)
```

In [114]:
```
print(accuracy_score(ytest,ypred_dt))
print(confusion_matrix(ytest,ypred_dt))
print(classification_report(ytest,ypred_dt))
```

```
0.8081
[[23322  3183]
 [ 2574   921]]
              precision    recall  f1-score   support

           0       0.90      0.88      0.89     26505
           1       0.22      0.26      0.24      3495

    accuracy                           0.81     30000
   macro avg       0.56      0.57      0.57     30000
weighted avg       0.82      0.81      0.81     30000
```

In [115]:
```
print(accuracy_score(ytrain,ypred_train_dt))
print(confusion_matrix(ytrain,ypred_train_dt))
print(classification_report(ytrain,ypred_train_dt))
```

```
1.0
[[61575     0]
 [    0  8425]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     61575
           1       1.00      1.00      1.00      8425

    accuracy                           1.00     70000
   macro avg       1.00      1.00      1.00     70000
weighted avg       1.00      1.00      1.00     70000
```

[116]:
```
update_performance(name = 'DecisionTreeClassifier', model = dt, test = ytest, pred= ypred_dt)
perf_score
```

t[116]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400

In [117]:
```
from sklearn.model_selection import GridSearchCV
```

In [153]:
```
tuned_paramaters = [{'criterion': ['entropy','gini'],
                     'max_depth': [2,3,5,6,7,8,9,10],
                     'max_leaf_nodes': [5,8],
                     'min_samples_leaf': [1,5,9],
                     'max_features': ["sqrt", "log2"],
                     'min_samples_split': [2,5,8]

}]
```

In [154]:
```
dt =DecisionTreeClassifier(random_state=10)
tree_grid = GridSearchCV(estimator=dt,param_grid=tuned_paramaters,cv=5)
```

In [155]:
```
tree_grid_model = tree_grid.fit(xtrain, ytrain)
print('Best parameters for decision tree classifier: ', tree_grid_model.best_params_, '\n')
```

```
Best parameters for decision tree classifier:  {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'sqrt', 'max_leaf_node
s': 5, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

In [ ]:

```
Best parameters for decision tree classifier:  {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'sqrt',
'max_leaf_nodes': 5, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

In [ ]:

In [118]:
```
dt_grid_model = DecisionTreeClassifier(criterion = 'entropy',
                                       max_depth = 2,
                                       max_features = 'sqrt',
                                       max_leaf_nodes = 5,
                                       min_samples_leaf = 1,
                                       min_samples_split = 2,
                                       random_state = 10)
```

In [119]:
```
dt_grid_model = dt_grid_model.fit(xtrain,ytrain)
```

In [120]:
```
ypred_dt_tp = dt_grid_model.predict(xtest)
```

In [121]:
```
print(accuracy_score(ytest,ypred_dt_tp))
print(confusion_matrix(ytest,ypred_dt_tp))
print(classification_report(ytest,ypred_dt_tp))
```

```
0.8835
[[26505     0]
 [ 3495     0]]
              precision    recall  f1-score   support

           0       0.88      1.00      0.94     26505
           1       0.00      0.00      0.00      3495

    accuracy                           0.88     30000
   macro avg       0.44      0.50      0.47     30000
weighted avg       0.78      0.88      0.83     30000
```

In [122]:
```
update_performance(name = 'DecisionTreeClassifier tunned', model = dt_grid_model, test = ytest, pred= ypred_dt_tp)
perf_score
```

Out[122]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000

In [ ]:

In [123]:
```
perf_score.loc[:,:]
```

Out[123]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000

Inference of Non Resampled models:

- All the above models and related score are based on with out resampling
- even there is huge class imbalance
- from the above models Gaussian navie bayes is performing well compared to all others

Resampling With SMOTE

In [127]:
```
# Resampling using SMOTE
```

In [128]:
```
from imblearn.over_sampling import SMOTE
xr,yr=SMOTE(sampling_strategy=0.5).fit_resample(x,y)
dfs=pd.concat([xr,pd.DataFrame(yr)],axis=1)
dfs
```

Out[128]:

	curr_ann_amt	days_tenure	age_in_years	income	length_of_residence	cust_origin_year	city	state	county	has_children	marital_status	home_ma
0	363.078715	123.020659	5.246528	73.429483	5.986978	0.567336	46	0	9	1	0	
1	421.384699	139.525532	6.172987	78.487963	1.462978	0.567336	36	0	2	0	1	
2	418.852711	251.188630	5.656420	90.640057	4.624756	0.567336	21	0	2	0	0	
3	428.128999	28.416913	5.587255	129.135533	3.286750	0.567336	5	0	13	1	0	
4	350.017764	283.192186	5.479362	114.917851	2.469252	0.567336	33	0	13	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	
132115	450.328467	27.449545	5.539041	96.180701	4.267735	0.567336	39	0	13	1	0	
132116	441.805198	215.968399	6.381152	88.360497	1.822081	0.567336	33	0	13	0	0	
132117	372.897000	32.493621	5.890872	159.402845	2.209440	0.567336	68	0	2	0	0	
132118	470.237505	17.221729	4.781144	73.429483	2.721392	0.567336	33	0	13	0	1	
132119	196.981209	225.442097	4.882671	126.521672	4.468451	0.567336	30	0	3	1	0	

132120 rows × 16 columns

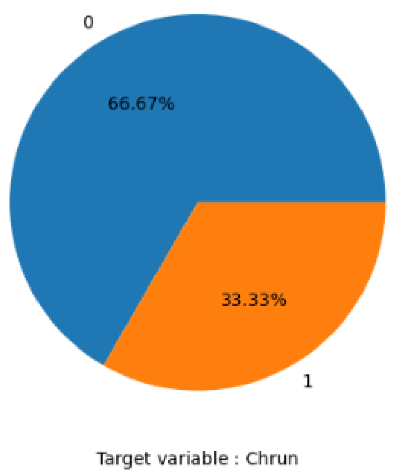
In [ ]:

After Resampling

In [129]:
```
# After resampling using smote
```

In [130]:
```
plt.pie(dfs['Churn'].value_counts(), radius=1, autopct='%.2f%%', labels= dfs['Churn'].unique())
plt.xlabel('Target variable : Chrun')
```

Out[130]: Text(0.5, 0, 'Target variable : Chrun')



Inference:

- Before resampling of data there is huge class imbalance in target variable with 12 percentage of '1' and 88 percentage of '0'
- After resampling of the data, using Smote the class '1' in taget variable is increased to 33.33 percentage (or) 1/3 rd of the Data

```
In [135]: from sklearn.linear_model import LogisticRegression

In [136]: lr_RS=LogisticRegression()

In [137]: lr_RS.fit(x_train_s,y_train)
Out[137]: LogisticRegression()

In [138]: lr_RS.intercept_
Out[138]: array([-0.88216217])

In [139]: pd.concat([pd.DataFrame(x.columns),pd.DataFrame(np.transpose(lr_RS.coef_))], axis = 1)
Out[139]:
```

	0	0
0	curr_ann_amt	-0.009589
1	days_tenure	-3.840723
2	age_in_years	-0.134167
3	income	0.066627
4	length_of_residence	-0.031716
5	cust_origin_year	-3.201692
6	city	-0.044201
7	state	0.000000
8	county	-0.047495
9	has_children	-0.351523
10	marital_status	-0.542966
11	home_market_value	-0.096311

```
In [ ]:

In [140]: ypred_lr_train_RS=lr_RS.predict(x_train_s)
          ypred_lr_RS=lr_RS.predict(x_test_s)

In [ ]:

In [141]: print(accuracy_score(y_train,ypred_lr_train_RS))
          print(confusion_matrix(y_train,ypred_lr_train_RS))
          print(classification_report(y_train,ypred_lr_train_RS))

0.770003460057956
[[55811  5914]
 [15357 15402]]
              precision    recall  f1-score   support

           0       0.78      0.90      0.84     61725
           1       0.72      0.50      0.59     30759

    accuracy                           0.77     92484
   macro avg       0.75      0.70      0.72     92484
weighted avg       0.76      0.77      0.76     92484


In [142]: print(accuracy_score(y_test,ypred_lr_RS))
          print(confusion_matrix(y_test,ypred_lr_RS))
          print(classification_report(y_test,ypred_lr_RS))

0.7715965284085176
[[23920  2435]
 [ 6618  6663]]
              precision    recall  f1-score   support

           0       0.78      0.91      0.84     26355
           1       0.73      0.50      0.60     13281

    accuracy                           0.77     39636
   macro avg       0.76      0.70      0.72     39636
weighted avg       0.77      0.77      0.76     39636
```

In [143]:
```
# Resample(R.s)
update_performance(name = 'Logistic Regression (R.S)', model = lr_RS, test = y_test, pred= ypred_lr_RS)
perf_score
```

Out[143]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469

Applying KNN:-

In [144]:
```
from sklearn.neighbors import KNeighborsClassifier
KNN_RS=KNeighborsClassifier()
```

In [145]:
```
KNN_model_RS=KNN_RS.fit(x_train_s,y_train)
```

In [146]:
```
ypred_KNN_train_RS=KNN_model_RS.predict(x_train_s)
ypred_KNN_RS=KNN_model_RS.predict(x_test_s)
```

In [ ]:

In [ ]:

In [147]:
```
print(accuracy_score(y_train,ypred_KNN_train_RS))
print(confusion_matrix(y_train,ypred_KNN_train_RS))
print(classification_report(y_train,ypred_KNN_train_RS))
```

```
0.8549911336014878
[[55249  6476]
 [ 6935 23824]]
              precision    recall  f1-score   support

           0       0.89      0.90      0.89     61725
           1       0.79      0.77      0.78     30759

    accuracy                           0.85     92484
   macro avg       0.84      0.83      0.84     92484
weighted avg       0.85      0.85      0.85     92484
```

In [148]:
```
print(accuracy_score(y_test,ypred_KNN_RS))
print(confusion_matrix(y_test,ypred_KNN_RS))
print(classification_report(y_test,ypred_KNN_RS))
```

```
0.7855232616812998
[[22282  4073]
 [ 4428  8853]]
              precision    recall  f1-score   support

           0       0.83      0.85      0.84     26355
           1       0.68      0.67      0.68     13281

    accuracy                           0.79     39636
   macro avg       0.76      0.76      0.76     39636
weighted avg       0.78      0.79      0.78     39636
```

In [149]:
```
update_performance(name = 'KNeighborsClassifier(R.S)', model = KNN_RS, test = y_test, pred= ypred_KNN_RS)
perf_score
```

Out[149]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621

In [ ]:

In [ ]:
```
# tunned params for Knn: {'n_neighbors': 2, 'p': 1}
```

In [150]:
```
knn_RS = KNeighborsClassifier(n_neighbors=2,p=1)
knn_RS.fit(x_train_s,y_train)
```

Out[150]:
```
KNeighborsClassifier(n_neighbors=2, p=1)
```

In [151]:
```
ypred_knn_RS_t = knn_RS.predict(x_test)
```

In [152]:
```
print(accuracy_score(y_test,ypred_knn_RS_t))
print(confusion_matrix(y_test,ypred_knn_RS_t))
print(classification_report(y_test,ypred_knn_RS_t))
```

```
0.6644969219901l
[[26315    40]
 [13258    23]]
              precision    recall  f1-score   support

           0       0.66      1.00      0.80     26355
           1       0.37      0.00      0.00     13281

    accuracy                           0.66     39636
   macro avg       0.52      0.50      0.40     39636
weighted avg       0.56      0.66      0.53     39636
```

In [153]:
```
update_performance(name = 'Knn(R.S) tunned', model = knn_RS, test = y_test, pred= ypred_knn_RS_t)
perf_score
```

Out[153]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621
8	Knn(R.S) tunned	0.664497	0.001732	0.365079	0.003447

Applying DecisionTree:-

In [ ]:

In [154]:
```
from sklearn.tree import DecisionTreeClassifier
dt_RS = DecisionTreeClassifier(random_state=10)
```

In [155]:
```
dt_RS.fit(x_train,y_train)
```

Out[155]:
```
DecisionTreeClassifier(random_state=10)
```

In [156]:
```
ypred_dt_RS=dt_RS.predict(x_test)
ypred_train_dt_RS=dt_RS.predict(x_train)
```

In [ ]:

In [157]:
```
print(accuracy_score(y_test,ypred_dt_RS))
print(confusion_matrix(y_test,ypred_dt_RS))
print(classification_report(y_test,ypred_dt_RS))
```

```
0.819759814310223
[[22613  3742]
 [ 3402  9879]]
              precision    recall  f1-score   support

           0       0.87      0.86      0.86     26355
           1       0.73      0.74      0.73     13281

    accuracy                           0.82     39636
   macro avg       0.80      0.80      0.80     39636
weighted avg       0.82      0.82      0.82     39636
```

In [158]:
```
print(accuracy_score(y_train,ypred_train_dt_RS))
print(confusion_matrix(y_train,ypred_train_dt_RS))
print(classification_report(y_train,ypred_train_dt_RS))
```

```
1.0
[[61725     0]
 [    0 30759]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     61725
           1       1.00      1.00      1.00     30759

    accuracy                           1.00     92484
   macro avg       1.00      1.00      1.00     92484
weighted avg       1.00      1.00      1.00     92484
```

In [159]:
```
update_performance(name = 'DecisionTreeClassifier(R.S) ', model = dt_RS, test = y_test, pred= ypred_dt_RS)
perf_score
```

Out[159]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621
8	Knn(R.S) tunned	0.664497	0.001732	0.365079	0.003447
9	DecisionTreeClassifier(R.S)	0.819760	0.743845	0.725277	0.734444

In [163]:
```
print(accuracy_score(y_test,ypred_dt_RS_tp))
print(confusion_matrix(y_test,ypred_dt_RS_tp))
print(classification_report(y_test,ypred_dt_RS_tp))
```

```
0.7244424260773035
[[25151  1204]
 [ 9718  3563]]
              precision    recall  f1-score   support

           0       0.72      0.95      0.82     26355
           1       0.75      0.27      0.39     13281

    accuracy                           0.72     39636
   macro avg       0.73      0.61      0.61     39636
weighted avg       0.73      0.72      0.68     39636
```

In [ ]:

In [164]:
```
update_performance(name = 'DecisionTreeClassifier(R.S) tunned ', model = dt_grid_model_RS, test = y_test, pred= ypred_dt_RS_tp)
perf_score
```

Out[164]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621
8	Knn(R.S) tunned	0.664497	0.001732	0.365079	0.003447
9	DecisionTreeClassifier(R.S)	0.819760	0.743845	0.725277	0.734444
10	DecisionTreeClassifier(R.S) tunned	0.724442	0.268278	0.747430	0.394836

In [ ]:

Applying RandomForest:-

In [165]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [166]:
```
rf=RandomForestClassifier(random_state=10)
rf.fit(x_train,y_train)
```

Out[166]: RandomForestClassifier(random_state=10)

In [167]:
```
ypred_rf = rf.predict(x_test)
ypred_rf_train=rf.predict(x_train)
```

In [168]:
```
print(accuracy_score(y_train,ypred_rf_train))
print(confusion_matrix(y_train,ypred_rf_train))
print(classification_report(y_train,ypred_rf_train))
```

```
0.9999783746377752
[[61725     0]
 [    2 30757]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     61725
           1       1.00      1.00      1.00     30759

    accuracy                           1.00     92484
   macro avg       1.00      1.00      1.00     92484
weighted avg       1.00      1.00      1.00     92484
```

In [169]:
```
print(accuracy_score(y_test,ypred_rf))
print(confusion_matrix(y_test,ypred_rf))
print(classification_report(y_test,ypred_rf))
```

```
0.8714552427086487
[[24824  1531]
 [ 3564  9717]]
              precision    recall  f1-score   support

           0       0.87      0.94      0.91     26355
           1       0.86      0.73      0.79     13281

    accuracy                           0.87     39636
   macro avg       0.87      0.84      0.85     39636
weighted avg       0.87      0.87      0.87     39636
```

In [170]:
```
update_performance(name = 'Random-Forest (RS)', model = rf, test = y_test, pred=ypred_rf)
perf_score
```

Out[170]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621
8	Knn(R.S) tunned	0.664497	0.001732	0.365079	0.003447
9	DecisionTreeClassifier(R.S)	0.819760	0.743845	0.725277	0.734444
10	DecisionTreeClassifier(R.S) tunned	0.724442	0.268278	0.747430	0.394836
11	Random-Forest (RS)	0.871455	0.731647	0.863887	0.792287

Applying Bagging

In [171]:
```
# Bagging Classifier
```

In [172]:
```
from sklearn.ensemble import BaggingClassifier
```

In [173]:
```
dt = DecisionTreeClassifier(random_state=10)
bc=BaggingClassifier(dt)
bc.fit(x_train,y_train)

ypred_bc=bc.predict(x_test)

print(accuracy_score(y_test,ypred_bc))
print(confusion_matrix(y_test,ypred_bc))
print(classification_report(y_test,ypred_bc))
```

```
0.8762740942577455
[[25015  1340]
 [ 3564  9717]]
              precision    recall  f1-score   support

           0       0.88      0.95      0.91     26355
           1       0.88      0.73      0.80     13281

    accuracy                           0.88     39636
   macro avg       0.88      0.84      0.85     39636
weighted avg       0.88      0.88      0.87     39636
```

In [174]:
```
update_performance(name = 'BaggingClassifier DT (RS)', model = bc, test = y_test, pred=ypred_bc)
perf_score
```

Out[174]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621
8	Knn(R.S) tunned	0.664497	0.001732	0.365079	0.003447
9	DecisionTreeClassifier(R.S)	0.819760	0.743845	0.725277	0.734444
10	DecisionTreeClassifier(R.S) tunned	0.724442	0.268278	0.747430	0.394836
11	Random-Forest (RS)	0.871455	0.731647	0.863887	0.792287
12	BaggingClassifier DT (RS)	0.876274	0.731647	0.878810	0.798504

In [175]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [176]:
```
knn = KNeighborsClassifier()
bag_knn=BaggingClassifier(knn)
bag_knn.fit(x_train,y_train)

ypred_bag_knn=bag_knn.predict(x_test)

print(accuracy_score(y_test,ypred_bag_knn))
print(confusion_matrix(y_test,ypred_bag_knn))
print(classification_report(y_test,ypred_bag_knn))
```

```
0.812190937531537
[[21661  4694]
 [ 2750 10531]]
              precision    recall  f1-score   support

           0       0.89      0.82      0.85     26355
           1       0.69      0.79      0.74     13281

    accuracy                           0.81     39636
   macro avg       0.79      0.81      0.80     39636
weighted avg       0.82      0.81      0.81     39636
```

In [177]:
```
update_performance(name = 'BaggingClassifier-KNN (RS)', model = bag_knn, test = y_test, pred=ypred_bag_knn)
perf_score
```

Out[177]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621
8	Knn(R.S) tunned	0.664497	0.001732	0.365079	0.003447
9	DecisionTreeClassifier(R.S)	0.819760	0.743845	0.725277	0.734444
10	DecisionTreeClassifier(R.S) tunned	0.724442	0.268278	0.747430	0.394836
11	Random-Forest (RS)	0.871455	0.731647	0.863887	0.792287
12	BaggingClassifier DT (RS)	0.876274	0.731647	0.878810	0.798504
13	BaggingClassifier-KNN (RS)	0.812191	0.792937	0.691691	0.738862

In [ ]:

Applying Log.Regression:-

In [178]:
```
logr=LogisticRegression()
bag_logr = BaggingClassifier(logr,random_state=10)
bag_logr.fit(x_train,y_train)

ypred_bag_logr=bag_logr.predict(x_test)

print(accuracy_score(y_test,ypred_bag_logr))
print(confusion_matrix(y_test,ypred_bag_logr))
print(classification_report(y_test,ypred_bag_logr))
```

```
0.7505550509637703
[[23395  2960]
 [ 6927  6354]]
              precision    recall  f1-score   support

           0       0.77      0.89      0.83     26355
           1       0.68      0.48      0.56     13281

    accuracy                           0.75     39636
   macro avg       0.73      0.68      0.69     39636
weighted avg       0.74      0.75      0.74     39636
```

In [179]:
```
update_performance(name = 'BaggingClassifier-logr (RS)', model = bag_logr, test = y_test, pred=ypred_bag_logr)
perf_score
```

Out[179]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621
8	Knn(R.S) tunned	0.664497	0.001732	0.365079	0.003447
9	DecisionTreeClassifier(R.S)	0.819760	0.743845	0.725277	0.734444
10	DecisionTreeClassifier(R.S) tunned	0.724442	0.268278	0.747430	0.394836
11	Random-Forest (RS)	0.871455	0.731647	0.863887	0.792287
12	BaggingClassifier DT (RS)	0.876274	0.731647	0.878810	0.798504
13	BaggingClassifier-KNN (RS)	0.812191	0.792937	0.691691	0.738862
14	BaggingClassifier-logr (RS)	0.750555	0.478428	0.682199	0.562425

Applying AdaBoost:-

In [180]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [181]:
```
abcl = AdaBoostClassifier(dt,random_state=10)
abcl.fit(x_train,y_train)
```

Out[181]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(random_state=10),
                   random_state=10)
```

In [182]:
```
ypred_abcl=abcl.predict(x_test)
```

In [183]:
```
print(accuracy_score(y_test,ypred_abcl))
print(confusion_matrix(y_test,ypred_abcl))
print(classification_report(y_test,ypred_abcl))
```

```
0.8198355030780099
[[22581  3774]
 [ 3367  9914]]
              precision    recall  f1-score   support

           0       0.87      0.86      0.86     26355
           1       0.72      0.75      0.74     13281

    accuracy                           0.82     39636
   macro avg       0.80      0.80      0.80     39636
weighted avg       0.82      0.82      0.82     39636
```

In [184]:
```
update_performance(name = 'AdaBoostClassifier-Descision tree (RS)', model = abcl, test = y_test, pred=ypred_abcl)
perf_score
```

Out[184]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621
8	Knn(R.S) tunned	0.664497	0.001732	0.365079	0.003447
9	DecisionTreeClassifier(R.S)	0.819760	0.743845	0.725277	0.734444
10	DecisionTreeClassifier(R.S) tunned	0.724442	0.268278	0.747430	0.394836
11	Random-Forest (RS)	0.871455	0.731647	0.863887	0.792287
12	BaggingClassifier DT (RS)	0.876274	0.731647	0.878810	0.798504
13	BaggingClassifier-KNN (RS)	0.812191	0.792937	0.691691	0.738862
14	BaggingClassifier-logr (RS)	0.750555	0.478428	0.682199	0.562425
15	AdaBoostClassifier-Descision tree (RS)	0.819836	0.746480	0.724284	0.735215

Applying RandomForest:-

In [185]:
```
rf=RandomForestClassifier()
abcl_rf = AdaBoostClassifier(rf,random_state=10)
abcl_rf.fit(x_train,y_train)
ypred_abcl_rf=abcl_rf.predict(x_test)
```

In [186]:
```
print(accuracy_score(y_test,ypred_abcl_rf))
print(confusion_matrix(y_test,ypred_abcl_rf))
print(classification_report(y_test,ypred_abcl_rf))
```

```
0.8704712887274195
[[24830  1525]
 [ 3609  9672]]
              precision    recall  f1-score   support

           0       0.87      0.94      0.91     26355
           1       0.86      0.73      0.79     13281

    accuracy                           0.87     39636
   macro avg       0.87      0.84      0.85     39636
weighted avg       0.87      0.87      0.87     39636
```

In [187]:
```
update_performance(name = 'AdaBoostClassifier-Random Forest(RS)', model = abcl_rf, test = y_test, pred=ypred_abcl_rf)
perf_score
```

Out[187]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621
8	Knn(R.S) tunned	0.664497	0.001732	0.365079	0.003447
9	DecisionTreeClassifier(R.S)	0.819760	0.743845	0.725277	0.734444
10	DecisionTreeClassifier(R.S) tunned	0.724442	0.268278	0.747430	0.394836
11	Random-Forest (RS)	0.871455	0.731647	0.863887	0.792287
12	BaggingClassifier DT (RS)	0.876274	0.731647	0.878810	0.798504
13	BaggingClassifier-KNN (RS)	0.812191	0.792937	0.691691	0.738862
14	BaggingClassifier-logr (RS)	0.750555	0.478428	0.682199	0.562425
15	AdaBoostClassifier-Descision tree (RS)	0.819836	0.746480	0.724284	0.735215
16	AdaBoostClassifier-Random Forest(RS)	0.870471	0.728258	0.863803	0.790261

Applying GradientBoost:-

In [188]:
```
from sklearn.ensemble import GradientBoostingClassifier
gbcl = GradientBoostingClassifier(n_estimators=50,learning_rate=0.5,random_state=10)
gbcl.fit(x_train,y_train)
ypred_gbcl=gbcl.predict(x_test)
print(accuracy_score(y_test,ypred_gbcl))
print(confusion_matrix(y_test,ypred_gbcl))
print(classification_report(y_test,ypred_gbcl))
```

```
0.8950196790796245
[[25324  1031]
 [ 3130 10151]]
              precision    recall  f1-score   support

           0       0.89      0.96      0.92     26355
           1       0.91      0.76      0.83     13281

    accuracy                           0.90     39636
   macro avg       0.90      0.86      0.88     39636
weighted avg       0.90      0.90      0.89     39636
```

In [189]:
```
update_performance(name = 'GradientBoostingClassifier(RS)',model=gbcl,test=y_test,pred=ypred_gbcl)
perf_score
```

Out[189]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621
8	Knn(R.S) tunned	0.664497	0.001732	0.365079	0.003447
9	DecisionTreeClassifier(R.S)	0.819760	0.743845	0.725277	0.734444
10	DecisionTreeClassifier(R.S) tunned	0.724442	0.268278	0.747430	0.394836
11	Random-Forest (RS)	0.871455	0.731647	0.863887	0.792287
12	BaggingClassifier DT (RS)	0.876274	0.731647	0.878810	0.798504
13	BaggingClassifier-KNN (RS)	0.812191	0.792937	0.691691	0.738862
14	BaggingClassifier-logr (RS)	0.750555	0.478428	0.682199	0.562425
15	AdaBoostClassifier-Descision tree (RS)	0.819836	0.746480	0.724284	0.735215
16	AdaBoostClassifier-Random Forest(RS)	0.870471	0.728258	0.863803	0.790261
17	GradientBoostingClassifier(RS)	0.895020	0.764325	0.907798	0.829906

Applying XGBoost:-

```
In [190]: from xgboost import XGBClassifier
```

```
In [191]: xgb=XGBClassifier(random_state=10)
          xgb.fit(x_train,y_train)
```

```
Out[191]: XGBClassifier(base_score=None, booster=None, callbacks=None,
                        colsample_bylevel=None, colsample_bynode=None,
                        colsample_bytree=None, early_stopping_rounds=None,
                        enable_categorical=False, eval_metric=None, feature_types=None,
                        gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                        interaction_constraints=None, learning_rate=None, max_bin=None,
                        max_cat_threshold=None, max_cat_to_onehot=None,
                        max_delta_step=None, max_depth=None, max_leaves=None,
                        min_child_weight=None, missing=nan, monotone_constraints=None,
                        n_estimators=100, n_jobs=None, num_parallel_tree=None,
                        predictor=None, random_state=10, ...)
```

```
In [192]: ypred_xgb=xgb.predict(x_test)
          print(accuracy_score(y_test,ypred_xgb))

          0.9037238873751136
```

```
In [193]: update_performance(name = 'XGB (RS)',model=gbcl,test=y_test,pred=ypred_xgb)
          perf_score
```

Out[193]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.883267	0.059227	0.491686	0.105720
1	KNN	0.876367	0.131330	0.405477	0.198401
2	KNN-Tunned	0.883500	0.000000	0.000000	0.000000
3	Gaussian naive bayes	0.878933	0.297854	0.469130	0.364368
4	DecisionTreeClassifier	0.808100	0.263519	0.224415	0.242400
5	DecisionTreeClassifier tunned	0.883500	0.000000	0.000000	0.000000
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621
8	Knn(R.S) tunned	0.664497	0.001732	0.365079	0.003447
9	DecisionTreeClassifier(R.S)	0.819760	0.743845	0.725277	0.734444
10	DecisionTreeClassifier(R.S) tunned	0.724442	0.268278	0.747430	0.394836
11	Random-Forest (RS)	0.871455	0.731647	0.863887	0.792287
12	BaggingClassifier DT (RS)	0.876274	0.731647	0.878810	0.798504
13	BaggingClassifier-KNN (RS)	0.812191	0.792937	0.691691	0.738862
14	BaggingClassifier-logr (RS)	0.750655	0.478428	0.682199	0.562425
15	AdaBoostClassifier-Descision tree (RS)	0.819836	0.746480	0.724284	0.735215
16	AdaBoostClassifier-Random Forest(RS)	0.870471	0.728258	0.863803	0.790261
17	GradientBoostingClassifier(RS)	0.895020	0.764325	0.907798	0.829906
18	XGB (RS)	0.903724	0.775996	0.924554	0.843786

- Till this we can see that we have applied all model possible on the dataset with and without using SMOTE i.e resampling techniques and we will be going to summarize that in a plot and with our inference…

Models with their Performance Metrics:-

In [195]:
```
# Resample data models
perf_score.loc[6:,:]
```

Out[195]:

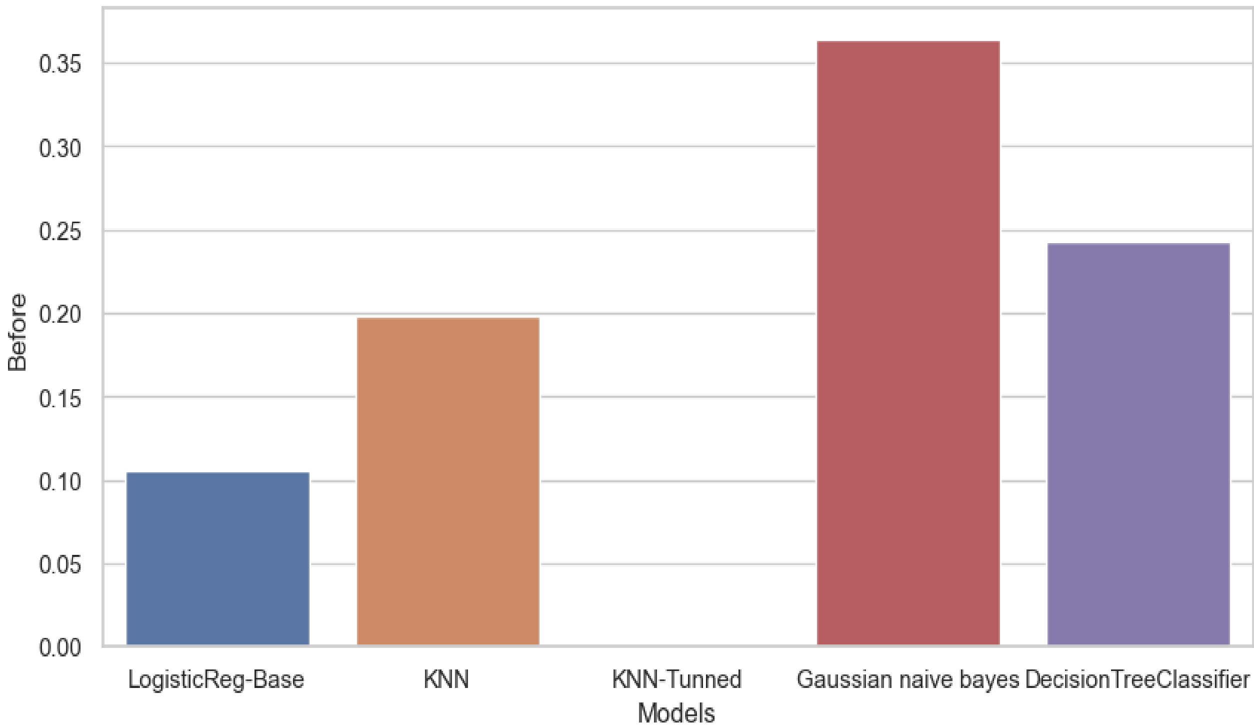
	Model	Accuracy	Recall	Precision	F1 Score
6	Logistic Regression (R.S)	0.771597	0.501694	0.732359	0.595469
7	KNeighborsClassifier(R.S)	0.785523	0.666591	0.684899	0.675621
8	Knn(R.S) tunned	0.664497	0.001732	0.365079	0.003447
9	DecisionTreeClassifier(R.S)	0.819760	0.743845	0.725277	0.734444
10	DecisionTreeClassifier(R.S) tunned	0.724442	0.268278	0.747430	0.394836
11	Random-Forest (RS)	0.871455	0.731647	0.863887	0.792287
12	BaggingClassifier DT (RS)	0.876274	0.731647	0.878810	0.798504
13	BaggingClassifier-KNN (RS)	0.812191	0.792937	0.691691	0.738862
14	BaggingClassifier-logr (RS)	0.750555	0.478428	0.682199	0.562425
15	AdaBoostClassifier-Descision tree (RS)	0.819836	0.746480	0.724284	0.735215
16	AdaBoostClassifier-Random Forest(RS)	0.870471	0.728258	0.863803	0.790261
17	GradientBoostingClassifier(RS)	0.895020	0.764325	0.907798	0.829906
18	XGB (RS)	0.903724	0.775996	0.924554	0.843786

In [ ]:

5. Conclusion

In conclusion, this project successfully conducted a comprehensive churn analysis on the provided dataset. By understanding churn patterns and identifying contributing factors, the organization can take informed steps to retain customers and enhance its customer relationship strategies.

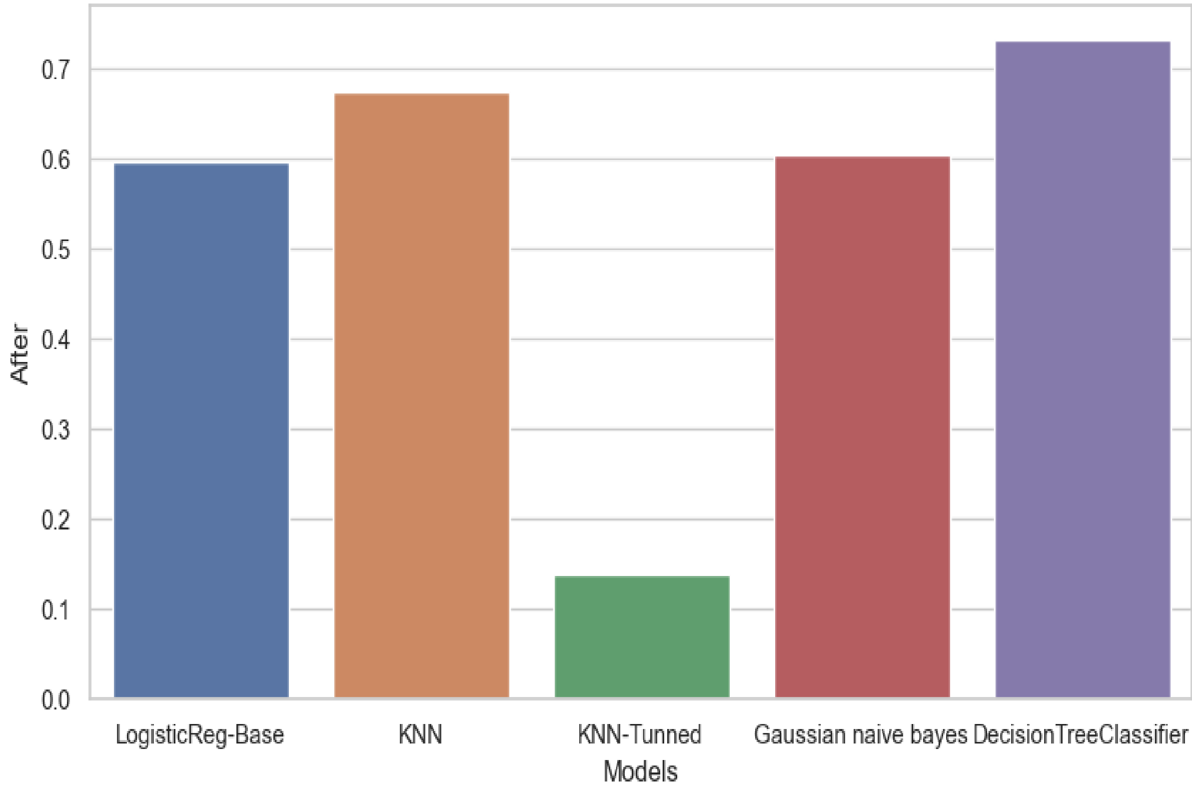
Without SMOTE



- From this we can see the respective F1 score of

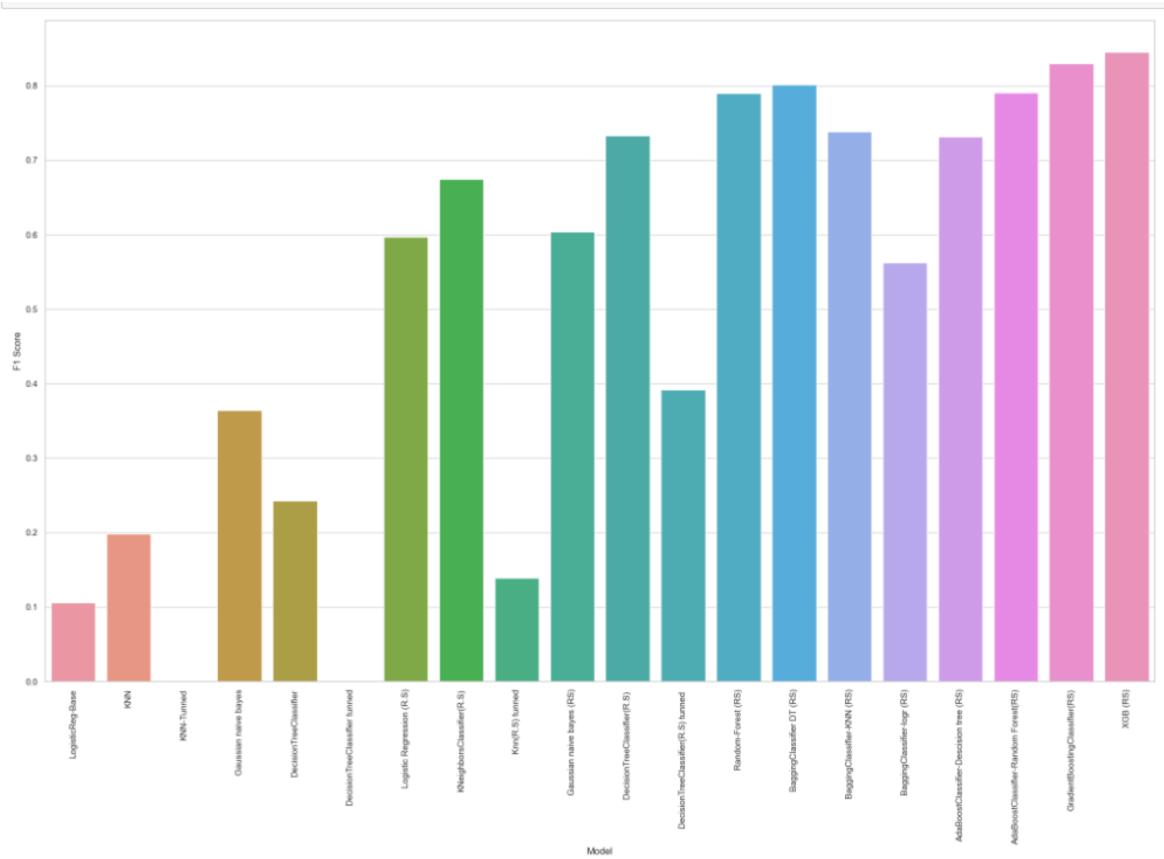
5 models without resampling…in which naïve_bayes with high F1 score..

With The use of SMOTE



- Here we can see that all the models performance have improved drastically after using resampling technique.

All models with their respective F1 score after successful use of SMOTE technique



- **Here we can the XGb has the highest score and its gradually going low as we are moving towards left..**
- **And the initial model are not giving such high high score ..**

THANK YOU !!!