UNVEILING CUSTOMER CHURN

Problem statement:

An E Commerce company or DTH (you can choose either of these two domains) provider is facing a lot of competition in the current market and it has become a challenge to retain the existing customers in the current situation. Hence, the company wants to develop a model through which they can do churn prediction of the accounts and provide segmented offers to the potential churners. In this company, account churn is a major thing because 1 account can have multiple customers. hence by losing one account the company might be losing more than one customer.

You have been assigned to develop a churn prediction model for this company and provide business recommendations on the campaign. Your campaign suggestion should be unique and be very clear on the campaign offer because your recommendation will go through the revenue assurance team. If they find that you are giving a lot of free (or subsidized) stuff thereby making a loss to the company; they are not going to approve your recommendation.

Hence be very careful while providing campaign recommendation.

Variable	Description
AccountID	account unique identifier
Churn	account churn flag (Target)
Tenure	Tenure of account
City_Tier	Tier of primary customer's city
CC_Contacted_LY	How many times all the customers of the account has contacted customer care in last 12 months
Payment	Preferred Payment mode of the customers in the account
Gender	Gender of the primary customer of the account
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_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
Marital_Status	Marital status of the primary customer of the account
rev_per_month	Monthly average revenue generated by account in last 12 months
Complain_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 13 months)
coupon_used_for_payment	How many times customers have used coupons to do the payment in last 12 months
Day_Since_CC_connect	Number of days since no customers in the account has contacted the customer care
cashback	Monthly average cashback generated by account in last 12 months
Login_device	Preferred login device of the customers in the account

---

Installing and Importing Dependencies

```
1 !pip install ydata_profiling
```

```
Collecting ydata_profiling
  Downloading ydata_profiling-4.8.3-py2.py3-none-any.whl (359 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 359.5/359.5 kB 2.7 MB/s eta 0
Requirement already satisfied: scipy<1.14,>=1.4.1 in /usr/local/lib/python3
Requirement already satisfied: pandas!=1.4.0,<3,>1.1 in /usr/local/lib/pyth
Requirement already satisfied: matplotlib<3.9,>=3.2 in /usr/local/lib/pytho
Requirement already satisfied: pydantic>=2 in /usr/local/lib/python3.10/dis
```

```
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python
Collecting visions[type_image_path]<0.7.7,>=0.7.5 (from ydata_profiling)
  Downloading visions-0.7.6-py3-none-any.whl (104 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 104.8/104.8 kB 7.6 MB/s eta 0
Requirement already satisfied: numpy<2,>=1.16.0 in /usr/local/lib/python3.1
Collecting htmlmin==0.1.12 (from ydata_profiling)
  Downloading htmlmin-0.1.12.tar.gz (19 kB)
  Preparing metadata (setup.py) ... done
Collecting phik<0.13,>=0.11.1 (from ydata_profiling)
  Downloading phik-0.12.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 686.1/686.1 kB 13.5 MB/s eta
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.10
Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/pyth
Collecting multimethod<2,>=1.4 (from ydata_profiling)
  Downloading multimethod-1.11.2-py3-none-any.whl (10 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/pyt
Collecting typeguard<5,>=3 (from ydata_profiling)
  Downloading typeguard-4.2.1-py3-none-any.whl (34 kB)
Collecting imagehash==4.3.1 (from ydata_profiling)
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 296.5/296.5 kB 6.8 MB/s eta 0
Requirement already satisfied: wordcloud>=1.9.1 in /usr/local/lib/python3.1
Collecting dacite>=1.8 (from ydata_profiling)
  Downloading dacite-1.8.1-py3-none-any.whl (14 kB)
Requirement already satisfied: numba<1,>=0.56.0 in /usr/local/lib/python3.1
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/di
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/pytho
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/pyt
Requirement already satisfied: pydantic-core==2.18.2 in /usr/local/lib/pyth
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/p
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/p
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/di
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/di
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/d
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packag
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from ydata_profiling import ProfileReport
```

Loading the data into notebook

```
df = pd.read_excel('/content/drive/MyDrive/Colab Notebooks/Jain/da
```

Few observations on the dataset

```
df.head()
```

	AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Ser
0	20000	1	4	3.0	6.0	Debit Card	Female	
1	20001	1	0	1.0	8.0	UPI	Male	
2	20002	1	0	1.0	30.0	Debit Card	Male	
3	20003	1	0	3.0	15.0	Debit Card	Male	
4	20004	1	0	1.0	12.0	Credit Card	Male	

```
df.columns
```

```
Index(['AccountID', 'Churn', 'Tenure', 'City_Tier', 'CC_Contacted_LY',
       'Payment', 'Gender', 'Service_Score', 'Account_user_count',
       'account_segment', 'CC_Agent_Score', 'Marital_Status',
'rev_per_month',
       'Complain_ly', 'rev_growth_yoy', 'coupon_used_for_payment',
       'Day_Since_CC_connect', 'cashback', 'Login_device'],
      dtype='object')
```

```
df.ndim
```

```
2
```

```
df.size
```

```
213940
```

```
len(df)
```

```
11260
```

```
df.shape
```

```
(11260, 19)
```

Checking the data type of each column

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11260 entries, 0 to 11259
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   AccountID                11260 non-null  int64  
 1   Churn                    11260 non-null  int64  
 2   Tenure                   11158 non-null  object 
 3   City_Tier                11148 non-null  float64
 4   CC_Contacted_LY          11158 non-null  float64
 5   Payment                  11151 non-null  object 
 6   Gender                   11152 non-null  object 
 7   Service_Score            11162 non-null  float64
 8   Account_user_count       11148 non-null  object 
 9   account_segment          11163 non-null  object 
 10  CC_Agent_Score           11144 non-null  float64
 11  Marital_Status           11048 non-null  object 
 12  rev_per_month            11158 non-null  object 
 13  Complain_ly              10903 non-null  float64
 14  rev_growth_yoy           11260 non-null  object 
 15  coupon_used_for_payment  11260 non-null  object 
 16  Day_Since_CC_connect     10903 non-null  object 
 17  cashback                 10789 non-null  object 
 18  Login_device             11039 non-null  object 
dtypes: float64(5), int64(2), object(12)
memory usage: 1.6+ MB
```

> Insights:

There are 7 numerical columns and 12 categorical columns.

Statistical analysis of numeric columns on the dataset

```
1 df.describe().T
```

	count	mean	std	min	25%	50%	
AccountID	11260.0	25629.500000	3250.626350	20000.0	22814.75	25629.5	284
Churn	11260.0	0.168384	0.374223	0.0	0.00	0.0	
City_Tier	11148.0	1.653929	0.915015	1.0	1.00	1.0	
CC_Contacted_LY	11158.0	17.867091	8.853269	4.0	11.00	16.0	
Service_Score	11162.0	2.902526	0.725584	0.0	2.00	3.0	
CC_Agent_Score	11144.0	3.066493	1.379772	1.0	2.00	3.0	
Complain_ly	10903.0	0.285334	0.451594	0.0	0.00	0.0	

Statistical analysis of categorical columns on the dataset

```
1 df.describe(include='O').T
```

	count	unique	top	freq
Tenure	11158	38	1	1351
Payment	11151	5	Debit Card	4587
Gender	11152	4	Male	6328
Account_user_count	11148	7	4	4569
account_segment	11163	7	Super	4062
Marital_Status	11048	3	Married	5860
rev_per_month	11158	59	3	1746
rev_growth_yoy	11260	20	14	1524
coupon_used_for_payment	11260	20	1	4373
Day_Since_CC_connect	10903	24	3	1816
cashback	10789.0	5693.0	155.62	10.0
Login_device	11039	3	Mobile	7482

Data Cleaning

```
for column in df.select_dtypes('object'):
    print(f'{column}: {df[column].nunique()}')
    print(f'{df[column].unique()}')
    print('\n\n')
```

```
[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 nan 27 12 21 17 50 60 31 51 61]



Payment: 5
['Debit Card' 'UPI' 'Credit Card' 'Cash on Delivery' 'E wallet' nan]



Gender: 4
['Female' 'Male' 'F' nan 'M']



Account_user_count: 7
[3 4 nan 5 2 '@' 1 6]



account_segment: 7
['Super' 'Regular Plus' 'Regular' 'HNI' 'Regular +' nan 'Super Plus'
 'Super +']



Marital_Status: 3
['Single' 'Divorced' 'Married' nan]



rev_per_month: 59
[9 7 6 8 3 2 4 10 1 5 '+' 130 nan 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]



rev_growth_yoy: 20
[11 15 14 23 22 16 12 13 17 18 24 19 20 21 25 26 '$' 4 27 28]
```

```
coupon_used_for_payment: 20
[1 0 4 2 9 6 11 7 12 10 5 3 13 15 8 '#' '$' 14 '*' 16]



Day_Since_CC_connect: 24
[5 0 3 7 2 1 8 6 4 15 nan 11 10 9 13 12 17 16 14 30 '$' 46 18 31 47]



cashback: 5693
[159.93 120.9 nan ... 227.36 226.91 191.42]



Login_device: 3
['Mobile' 'Computer' '&&&&' nan]
```

Correcting all the categorical variables with special symbols

Treating 'Tenure' column

- As you can see there is '#' symbol need to be treated as 'np.nan' value and we can perform imputation in future.
- Once the '#s' are converted to 'np.nan', this column will be converted to numerical column.

```
df['Tenure'] = df['Tenure'].replace('#', np.nan)

df['Tenure'].unique()
```

```
array([ 4.,  0.,  2., 13., 11., nan,  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.])
```

Treating 'Gender' column

- As we can see 'F' and 'M' are also known as 'Female' and 'Male'.

```
df['Gender'] = df['Gender'].replace('F', 'Female')
df['Gender'] = df['Gender'].replace('M', 'Male')

df['Gender'].unique()
```

```
array(['Female', 'Male', nan], dtype=object)
```

Treating 'Account_user_count' column

- As you can see there is '@' symbol need to be treated as 'np.nan' value and we can perform imputation in future.
- Once the '@s' are converted to 'np.nan', this column will be converted to numerical column.

```
df['Account_user_count'] = df['Account_user_count'].replace('@', n

df['Account_user_count'].unique()
```

```
array([ 3.,  4., nan,  5.,  2.,  1.,  6.])
```

Treating 'account_segment' column

- As you can see there is 'Regular +' and 'Super +' symbol need to be treated as 'Regular Plus' and 'Super Plus'.
- Since 'Regular +' is nothing but 'Regular Plus' and same in case of 'Super +'.

```
df['account_segment'] = df['account_segment'].replace('Regular +',
df['account_segment'] = df['account_segment'].replace('Super +', '

df['account_segment'].unique()
```

```
array(['Super', 'Regular Plus', 'Regular', 'HNI', nan, 'Super Plus'],
      dtype=object)
```

Treating 'rev_per_month' column

- As you can see there is '+' symbol need to be treated as 'np.nan' value and we can perform imputation in future.
- Once the '+s' are converted to 'np.nan', this column will be converted to numerical column.

```
df['rev_per_month'] = df['rev_per_month'].replace('+', np.nan)

df['rev_per_month'].unique()
```

```
array([  9.,   7.,   6.,   8.,   3.,   2.,   4.,  10.,   1.,   5.,  nan,
       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.])
```

Treating 'rev_growth_yoy' column

- As you can see there is '$' symbol need to be treated as 'np.nan' value and we can perform imputation in future.
- Once the '$s' are converted to 'np.nan', this column will be converted to numerical column.

```
df['rev_growth_yoy'] = df['rev_growth_yoy'].replace('$', np.nan)

df['rev_growth_yoy'].unique()
```

```
array([11., 15., 14., 23., 22., 16., 12., 13., 17., 18., 24., 19., 20.,
       21., 25., 26., nan,  4., 27., 28.])
```

Treating 'coupon_used_for_payment' column

- As you can see there is '#', '$' and '*' symbol need to be treated as 'np.nan' value and we can perform imputation in future.
- Once the '#s', '$s' and '*s' are converted to 'np.nan', this column will be converted to numerical column.

```
df['coupon_used_for_payment'] = df['coupon_used_for_payment'].repl
df['coupon_used_for_payment'] = df['coupon_used_for_payment'].repl
df['coupon_used_for_payment'] = df['coupon_used_for_payment'].repl

df['coupon_used_for_payment'].unique()
```

```
array([ 1.,  0.,  4.,  2.,  9.,  6., 11.,  7., 12., 10.,  5.,  3., 13.,
       15.,  8., nan, 14., 16.])
```

Treating 'Day_Since_CC_connect' column

- As you can see there is '$' symbol need to be treated as 'np.nan' value and we can perform imputation in future.
- Once the '$s' are converted to 'np.nan', this column will be converted to numerical column.

```
df['Day_Since_CC_connect'] = df['Day_Since_CC_connect'].replace('$

df['Day_Since_CC_connect'].unique()
```

```
array([ 5.,  0.,  3.,  7.,  2.,  1.,  8.,  6.,  4., 15., nan, 11., 10.,
        9., 13., 12., 17., 16., 14., 30., 46., 18., 31., 47.])
```

Treating 'cashback' column

- First need to know what is the special character.
- As you can see there is '$' symbol need to be treated as 'np.nan' value and we can perform imputation in future.
- Once the '$s' are converted to 'np.nan', this column will be converted to numerical column.

```
for data in df['cashback'].unique():
    if not isinstance(data, (int, float)):
        print(data)
```

```
$
```

```
df['cashback'] = df['cashback'].replace('$', np.nan)

df['cashback'].unique()
```

```
array([159.93, 120.9 ,    nan, ..., 227.36, 226.91, 191.42])
```

Treating 'Login_device' column

- As you can see there is '&&&&' symbol need to be treated as 'np.nan' value and we can perform imputation in future.
- Once the '&&&&s' are converted to 'np.nan', this column will be converted to numerical column.

```
df['Login_device'] = df['Login_device'].replace('&&&&', np.nan)

df['Login_device'].unique()
```

```
array(['Mobile', 'Computer', nan], dtype=object)
```

Exploratory Data Analysis (EDA)

Pandas Profiling

```
profile = ProfileReport(df)
profile
```

Summarize dataset: 100% 129/129 [00:54<00:00, 1.57it/s, Completed]

Generate report structure: 100% 1/1 [00:14<00:00, 14.08s/it]

Render HTML: 100% 1/1 [00:03<00:00, 3.76s/it]

Overview

Dataset statistics

Number of variables	19
Number of observations	11260
Missing cells	4361
Missing cells (%)	2.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	1.6 MiB
Average record size in memory	152.0 B

Variable types

Numeric	10
Categorical	9

Alerts

`Tenure` has 218 (1.9%) missing values	**Missing**
`Account_user_count` has 444 (3.9%) missing values	**Missing**
`CC_Agent_Score` has 116 (1.0%) missing values	**Missing**
`Marital_Status` has 212 (1.9%) missing values	**Missing**

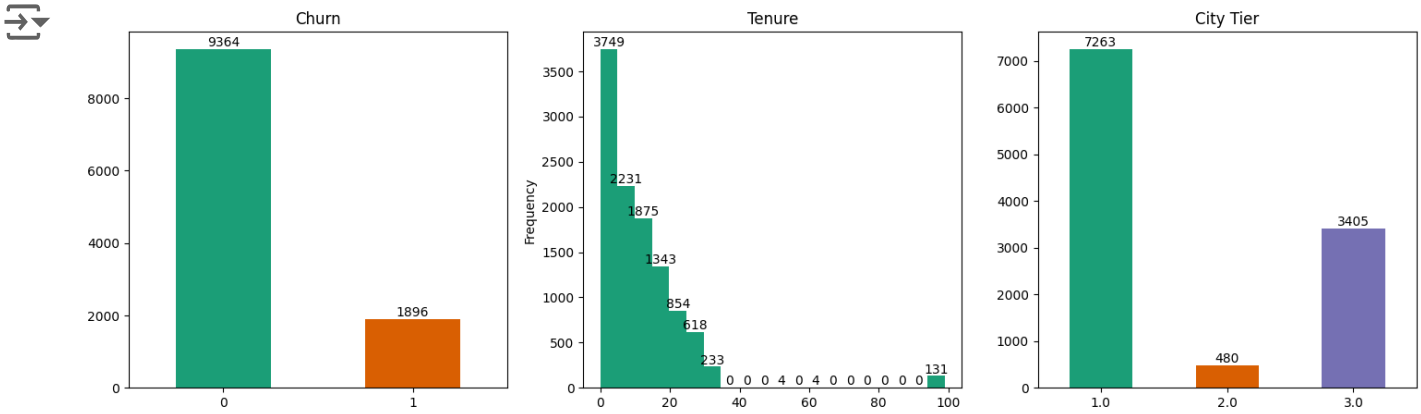
Univariate Analysis

1st Figure

```
# @title 1st Figure

# @markdown **Data Insights:**
# @markdown > Plot 1 - Imbalanc
# @markdown > Plot 2 - More num
# @markdown > Plot 3 - More num


fig = plt.figure(figsize=(18, 5

# 1st Column
plt.subplot(1, 3, 1)
ax = df.groupby('Churn').size()

ax.set_xlabel('')
ax.bar_label(ax.containers[0]);

# 2nd Column
plt.subplot(1, 3, 2)
ax = df['Tenure'].plot(kind='hi
                       color=sr
ax.bar_label(ax.containers[0]);

# 3rd Column
plt.subplot(1, 3, 3)
ax = df.groupby('City_Tier').si

ax.set_xlabel('')
ax.bar_label(ax.containers[0]);
```

Data Insights:

> Plot 1 - Imbalance dataset, as `Churn` samples are less.
> Plot 2 - More number of customers are falling under less `Tenure`.
> Plot 3 - More number of customers are from `City Tier 1` followed by `City Tier 3` and followed by `City Tier 2`.



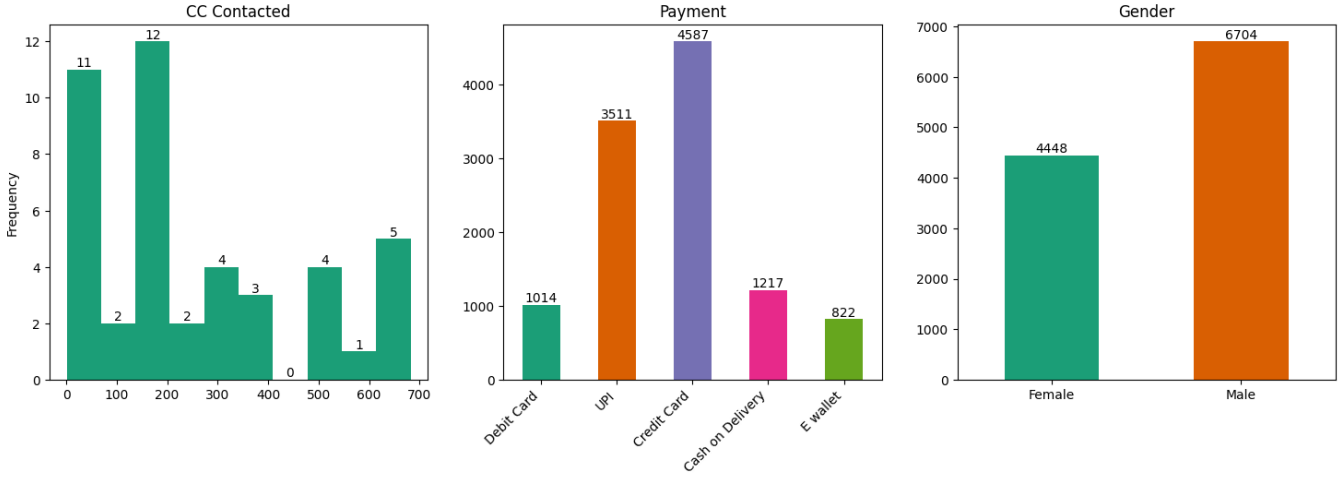
2nd Figure

```
# @title 2nd Figure

# @markdown **Data Insights:**
# @markdown > Plot 1 - `CC_Cont
# @markdown > Plot 2 - Most of
# @markdown > Plot 3 - Most of


fig = plt.figure(figsize=(18, 5

# 1st Column
plt.subplot(1, 3, 1)
ax = df.groupby('CC_Contacted_L


ax.set_xlabel('')
ax.bar_label(ax.containers[0]);


```

Data Insights:

> Plot 1 - `CC_Contacted` doesn't follow any distribution.
> Plot 2 - Most of the customers preferred `Payment` using `credit card` followed by `UPI`.
> Plot 3 - Most of the customers are `Male`.

```
# 2nd Column
plt.subplot(1, 3, 2)
ax = df.groupby('Payment').size

ax.set_xlabel('')
ax.set_xticklabels(df['Payment'
ax.bar_label(ax.containers[0]);


# 3rd Column
plt.subplot(1, 3, 3)
ax = df.groupby('Gender').size(

ax.set_xlabel('')
ax.bar_label(ax.containers[0]);
```



3rd Figure

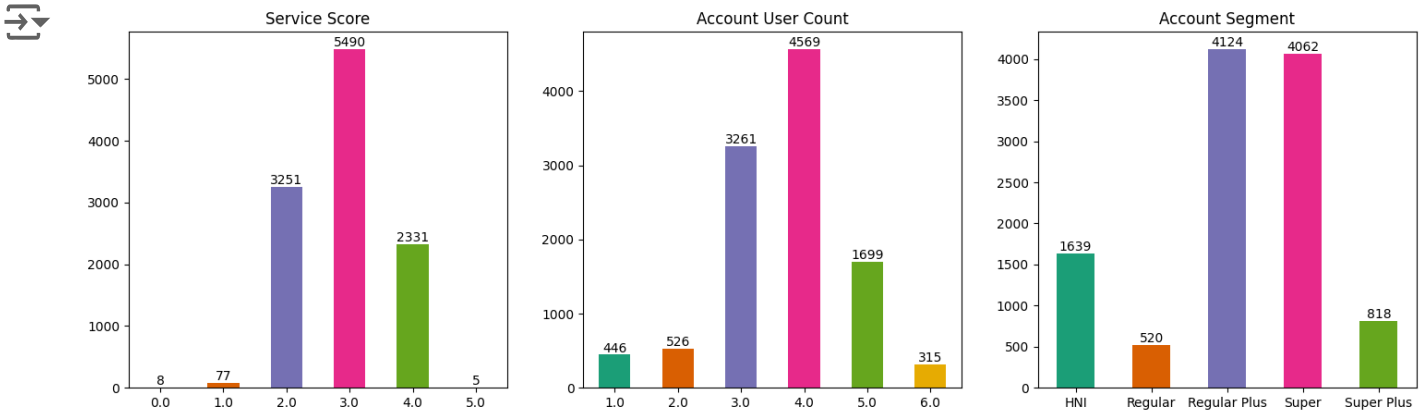
```
# @title 3rd Figure
```

Data Insights:

```
# @title 3rd Figure

# @markdown **Data Insights:**
# @markdown > Plot 1 - Most of
# @markdown > Plot 2 - Most of
# @markdown > Plot 3 - Most of


fig = plt.figure(figsize=(18, 5


# 1st Column
plt.subplot(1, 3, 1)
ax = df.groupby('Service_Score'


ax.set_xlabel('')
ax.bar_label(ax.containers[0]);


# 2nd Column
plt.subplot(1, 3, 2)
ax = df.groupby('Account_user_c


ax.set_xlabel('')
ax.bar_label(ax.containers[0]);


# 3rd Column
plt.subplot(1, 3, 3)
ax = df.groupby('account_segmer

ax.set_xlabel('')
ax.bar_label(ax.containers[0]);
```

Data Insights:

> Plot 1 - Most of the customers provided `Service Score` as `3`.
> Plot 2 - Most of the customers using `4` accounts.
> Plot 3 - Most of the customers are under `Regular Plus` and `Super` Account Segment.



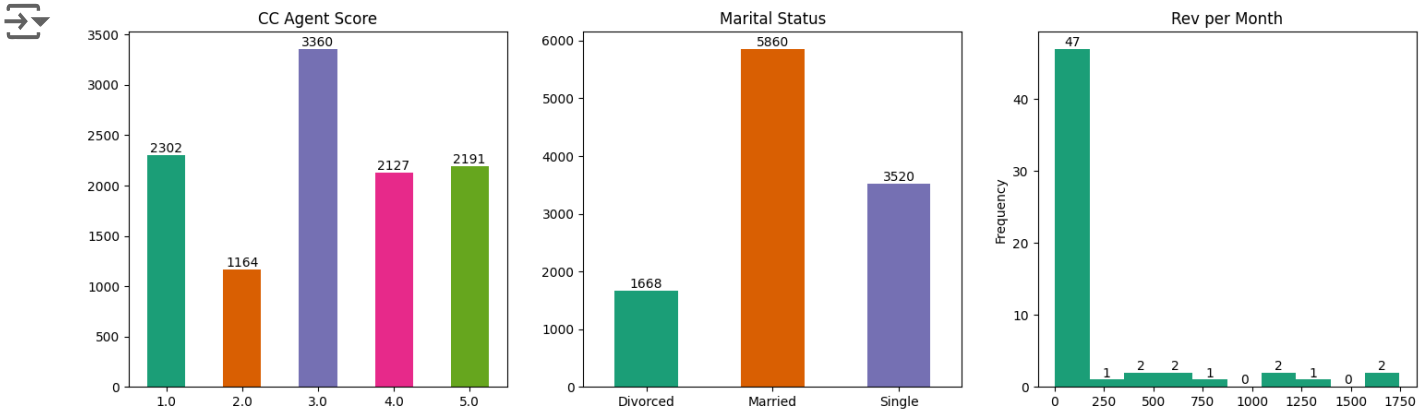
4th Figure

```
# @title 4th Figure

# @markdown **Data Insights:**
# @markdown > Plot 1 - Most of
# @markdown > Plot 2 - Most of
# @markdown > Plot 3 - Most of


fig = plt.figure(figsize=(18, 5

# 1st Column
plt.subplot(1, 3, 1)
ax = df.groupby('CC_Agent_Score

ax.set_xlabel('')
ax.bar_label(ax.containers[0]);


```

Data Insights:

> Plot 1 - Most of the customers provided score as `3` for the `CC Agent`.
> Plot 2 - Most of the customers are under `married` and followed by `Single` users.
> Plot 3 - Most of the customers are under `Regular Plus` and `Super` Account Segment.

```
# 2nd Column
plt.subplot(1, 3, 2)
ax = df.groupby('Marital_Status
ax.set_xlabel('')
ax.bar_label(ax.containers[0]);


# 3rd Column
plt.subplot(1, 3, 3)
ax = df.groupby('rev_per_month'
ax.set_xlabel('')
ax.bar_label(ax.containers[0]);
```



5th Figure

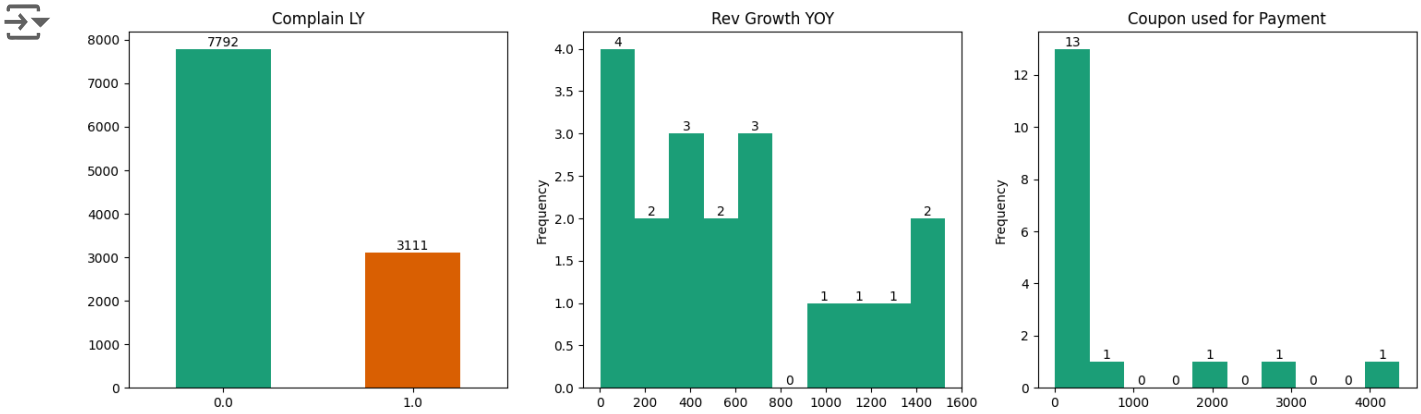
```
# @title 5th Figure

# @markdown **Data Insights:**
# @markdown > Plot 1 - `Complai
```

Data Insights:

> Plot 1 - `Complain_ly`

```
# @markdown > Plot 2 - `Rev_grc
# @markdown > Plot 3 - Most of


fig = plt.figure(figsize=(18, 5


# 1st Column
plt.subplot(1, 3, 1)
ax = df.groupby('Complain_ly').


ax.set_xlabel('')
ax.bar_label(ax.containers[0]);


# 2nd Column
plt.subplot(1, 3, 2)
ax = df.groupby('rev_growth_yoy


ax.set_xlabel('')
ax.bar_label(ax.containers[0]);


# 3rd Column
plt.subplot(1, 3, 3)
ax = df.groupby('coupon_used_fo


ax.set_xlabel('')
ax.bar_label(ax.containers[0]);
```

has less complains.
Plot 2 - `Rev_growth_yoy` does seems like normal distribution.
Plot 3 - Most of the customers used `Coupon for payment` between `0` and `500`.



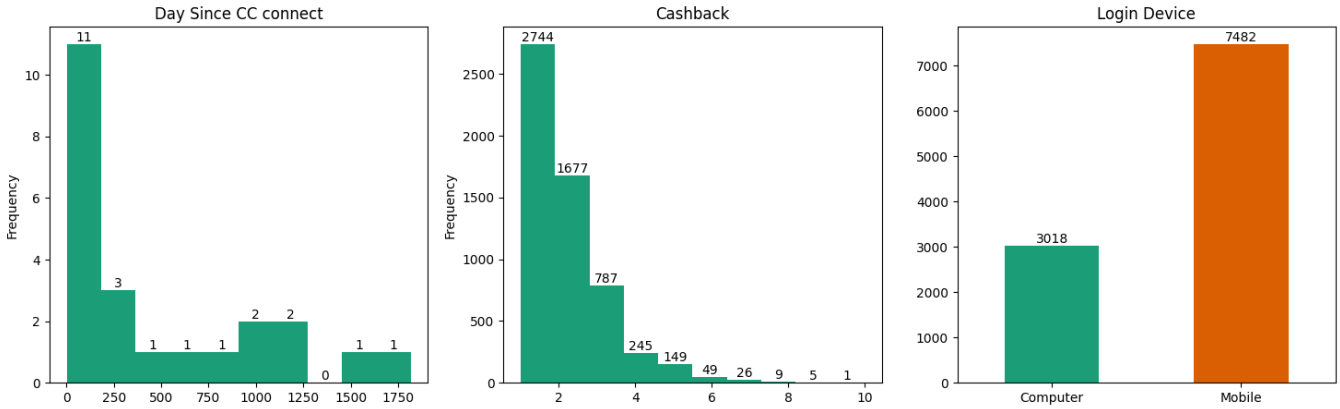
6th Figure

```
# @title 6th Figure

# @markdown **Data Insights:**
# @markdown > Plot 1 - Most of
# @markdown > Plot 2 - Most of
# @markdown > Plot 3 - Most of


fig = plt.figure(figsize=(18, 5

# 1st Column
plt.subplot(1, 3, 1)
ax = df.groupby('Day_Since_CC_c


ax.set_xlabel('')
ax.bar_label(ax.containers[0]);


```

Data Insights:

> Plot 1 - Most of the customers has not contacted `CC connect`.
> Plot 2 - Most of the customers got `Cashback` which is less in amount.
> Plot 3 - Most of the customers `Login Device` is `Mobile` compared to `Computer`.

```
# 2nd Column
plt.subplot(1, 3, 2)
ax = df.groupby('cashback').siz


ax.set_xlabel('')
ax.bar_label(ax.containers[0]);


# 3rd Column
plt.subplot(1, 3, 3)
ax = df.groupby('Login_device')


ax.set_xlabel('')
ax.bar_label(ax.containers[0]);
```



Bivariate Analysis

Warning

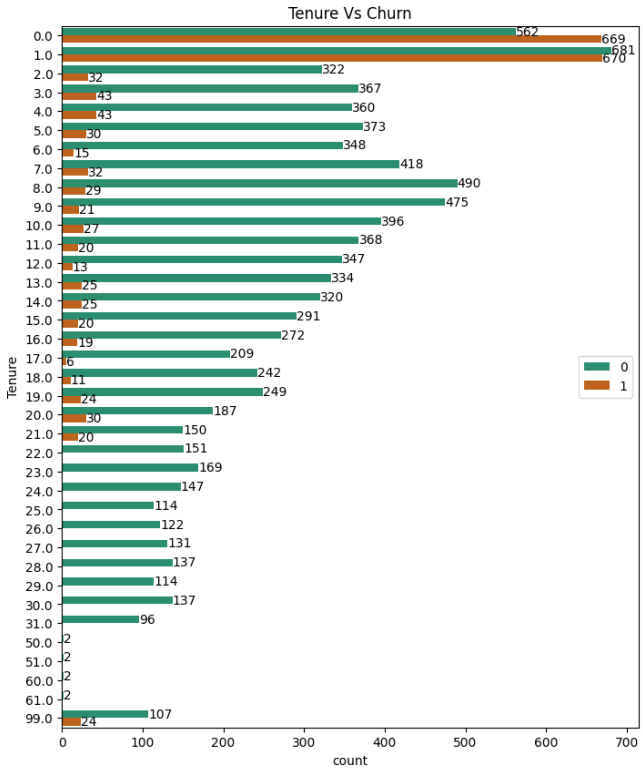
```
# @title Warning

import warnings

warnings.filterwarnings('ignore')
```

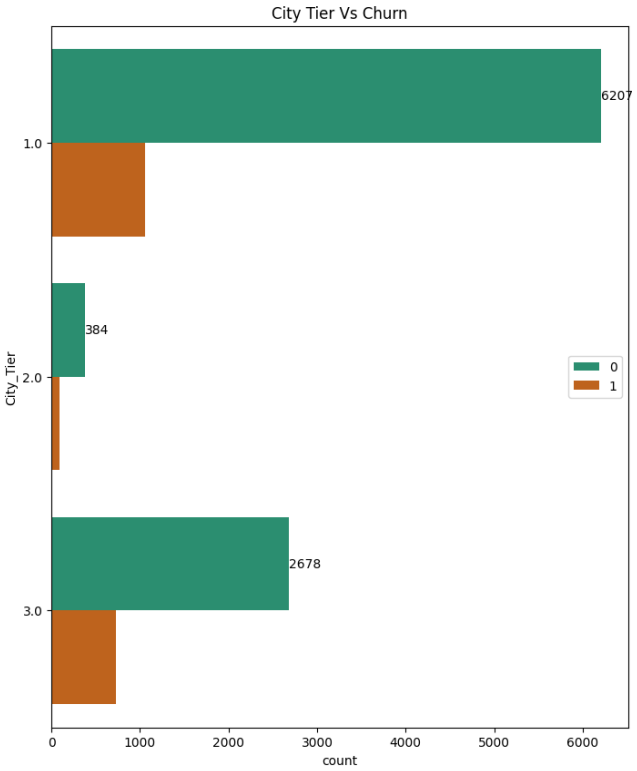
Tenure Vs Churn and City Tier Vs Churn

```
# @title `Tenure` Vs `Churn` ar
# @markdown **Data Insights:**
# @markdown > Plot 1 - If the `
# @markdown > Plot 2 - `City_Ti

fig = plt.figure(figsize=(18, 1

plt.subplot(121)
ax = sns.countplot(df, y='Tenur
ax.legend(loc='center right')
ax.set_title('Tenure Vs Churn')
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])

plt.subplot(122)
ax = sns.countplot(df, y='City_
ax.legend(loc='center right')
ax.set_title('City Tier Vs Chur
ax.bar_label(ax.containers[0]);
```

Data Insights:

> Plot 1 - If the `tenure` is less, the `Churn` is more.
> Plot 2 - `City_Tier` 2 and 3 has comparably more number of `Churn` rate.





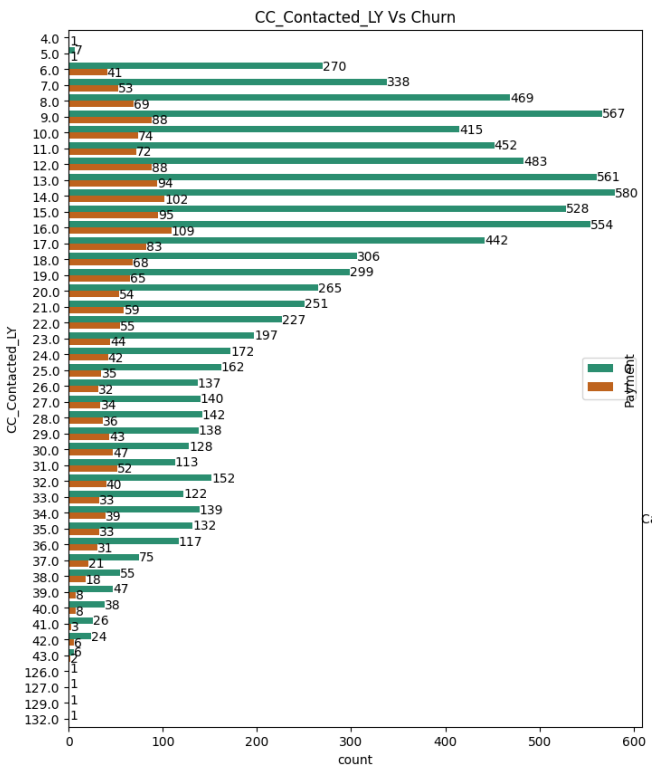
CC_Contacted_LY Vs Churn and Payment Vs Churn

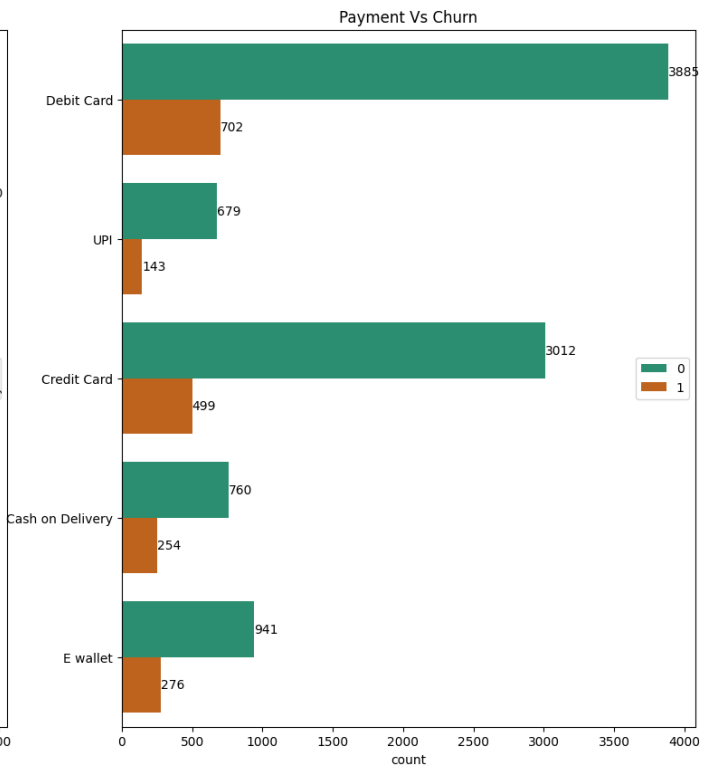
```
# @title `CC_Contacted_LY` Vs `
# @markdown **Data Insights:**
```

Data Insights:

```
# @markdown > Plot 1 - If the `
# @markdown > Plot 2 - Except `

fig = plt.figure(figsize=(18, 1

plt.subplot(121)
ax = sns.countplot(df, y='CC_Co
ax.legend(loc='center right')
ax.set_title('CC_Contacted_LY V
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])

plt.subplot(122)
ax = sns.countplot(df, y='Payme
ax.legend(loc='center right')
ax.set_title('Payment Vs Churn'
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1]);
```

Plot 1 - If the `CC_Contacted_LY` is more, the `Churn` is also more.
Plot 2 - Except `Debit and Credit` card all other payment methods are having high `Churn` rate.





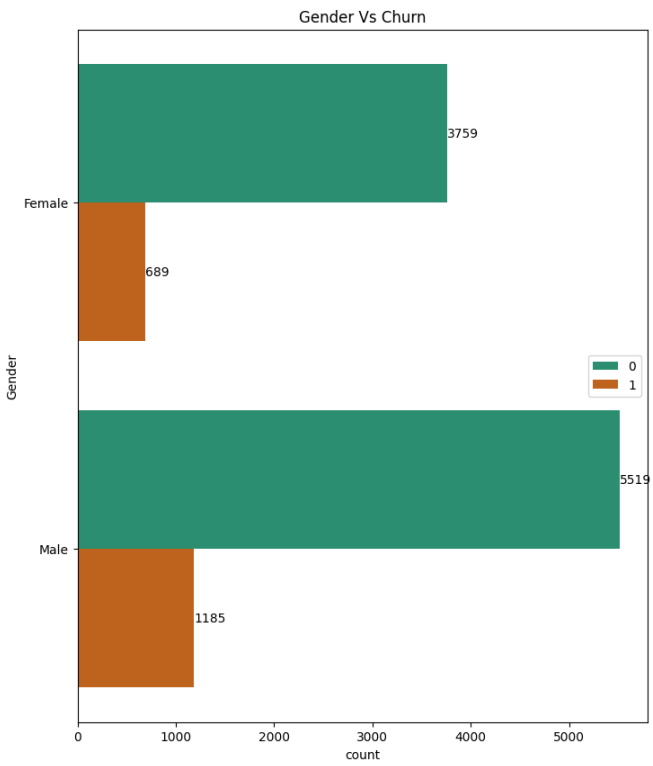
Gender Vs Churn and Service_Score Vs Churn

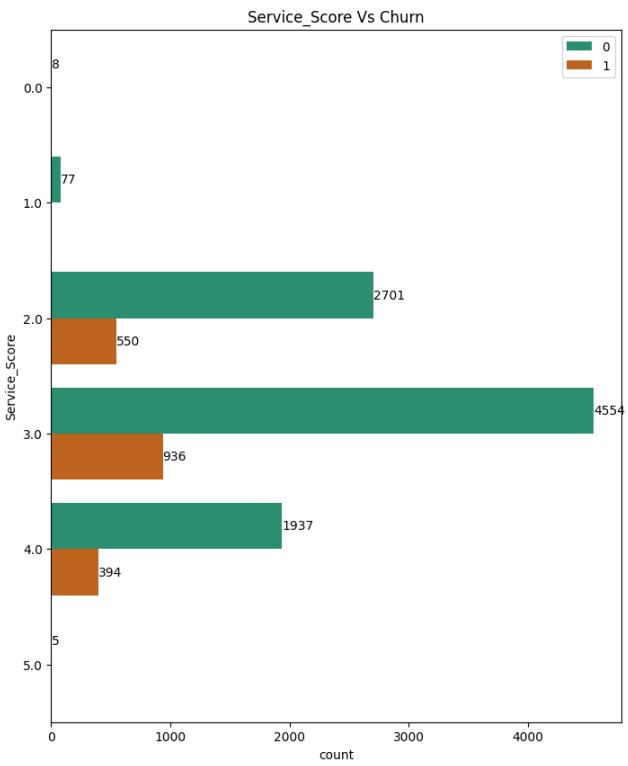
```
# @title `Gender` Vs `Churn` ar
# @markdown **Data Insights:**
```

Data Insights:

```
# @markdown > Plot 1 - `Gender`
# @markdown > Plot 2 - It seems

fig = plt.figure(figsize=(18, 1

plt.subplot(121)
ax = sns.countplot(df, y='Gende
ax.legend(loc='center right')
ax.set_title('Gender Vs Churn')
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])

plt.subplot(122)
ax = sns.countplot(df, y='Servi
ax.legend(loc='upper right')
ax.set_title('Service_Score Vs
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1]);
```

Plot 1 - `Gender` doesn't show any dependency over `Churn`.
Plot 2 - It seems like `Service_Score` has a positive relation with `Churn`.





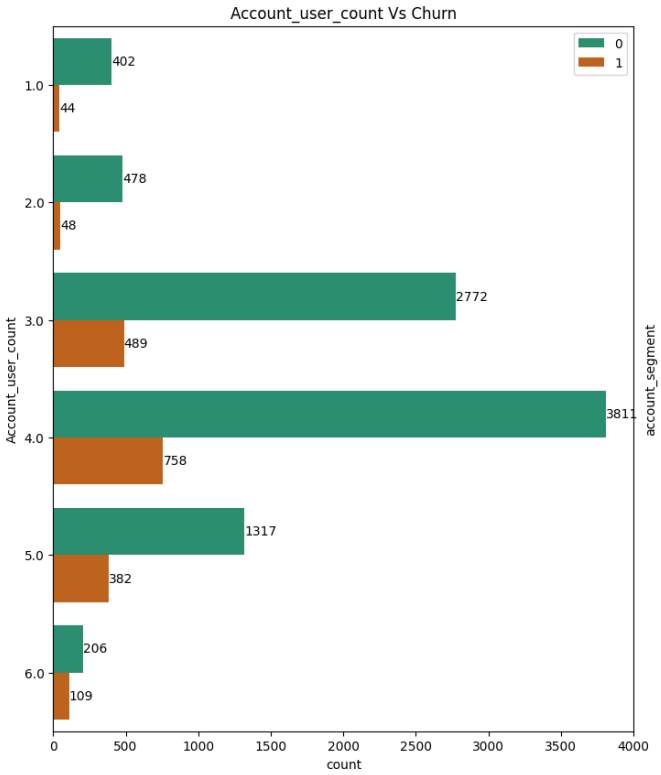
Account_user_count Vs Churn and account_segment Vs Churn

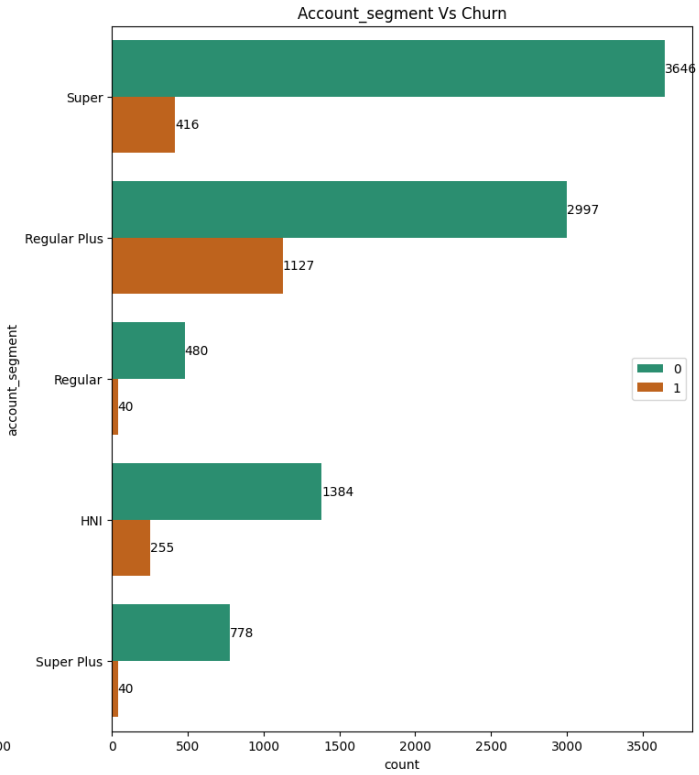
```
# @title `Account_user_count` Vs `Churn` and `account_segment` V
# @markdown **Data Insights:**
# @markdown > Plot 1 - `Account_user_count` shows positive relat
```

```
# @markdown > Plot 2 - It seems like `Regular Plus` `Segment` us

fig = plt.figure(figsize=(18, 10))

plt.subplot(121)
ax = sns.countplot(df, y='Account_user_count', hue='Churn', pale
ax.legend(loc='upper right')
ax.set_title('Account_user_count Vs Churn')
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])

plt.subplot(122)
ax = sns.countplot(df, y='account_segment', hue='Churn', palette
ax.legend(loc='center right')
ax.set_title('Account_segment Vs Churn')
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1]);
```

Plot 1 - `Account_user_count` shows positive relation with `Churn` rate.

Plot 2 - It seems like `Regular Plus` `Segment` user are more likely to `Churn`.





CC_Agent_Score Vs Churn and Marital_Status Vs Churn

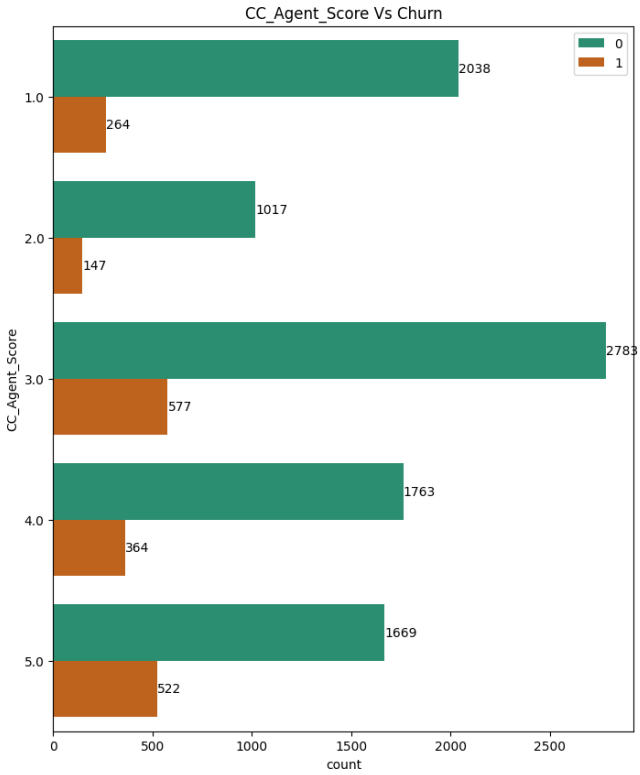
```
# @title `CC_Agent_Score` Vs `Churn` and `Marital_Status` Vs `Ch
# @markdown **Data Insights:**
```

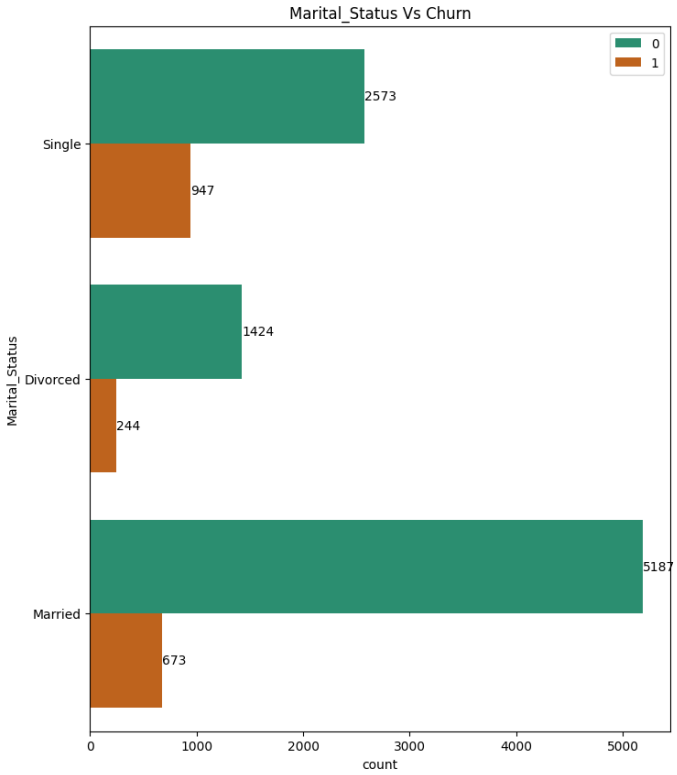
Data Insights:

```
# @markdown > Plot 1 - `CC_Agent_Score` shows no relation with `
# @markdown > Plot 2 - It seems like `Single` `Marital_status` u

fig = plt.figure(figsize=(18, 10))

plt.subplot(121)
ax = sns.countplot(df, y='CC_Agent_Score', hue='Churn', palette=
ax.legend(loc='upper right')
ax.set_title('CC_Agent_Score Vs Churn')
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])

plt.subplot(122)
ax = sns.countplot(df, y='Marital_Status', hue='Churn', palette=
ax.legend(loc='upper right')
ax.set_title('Marital_Status Vs Churn')
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1]);
```

Plot 1 - `CC_Agent_Score` shows no relation with `Churn` rate.

Plot 2 - It seems like `Single` `Marital_status` user are more likely to `Churn`.





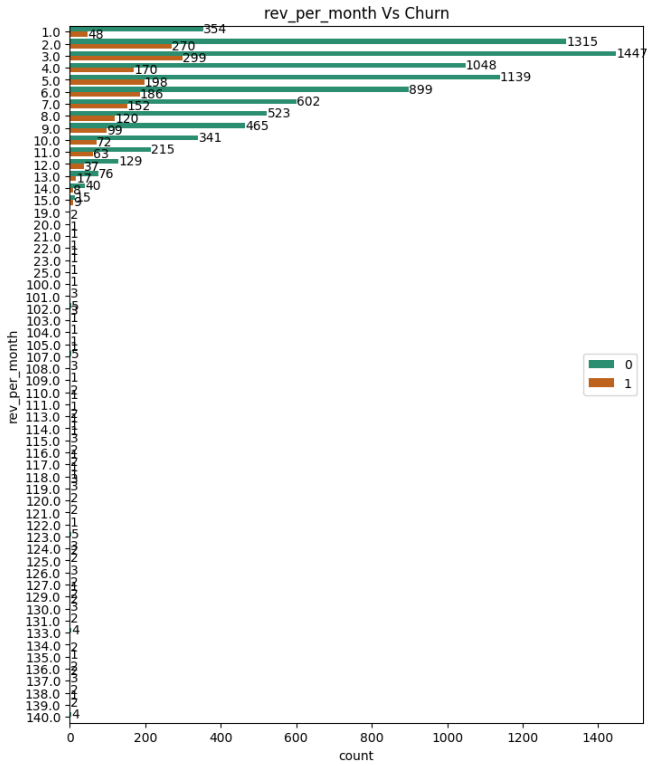
rev_per_month Vs Churn and Complain_ly Vs Churn

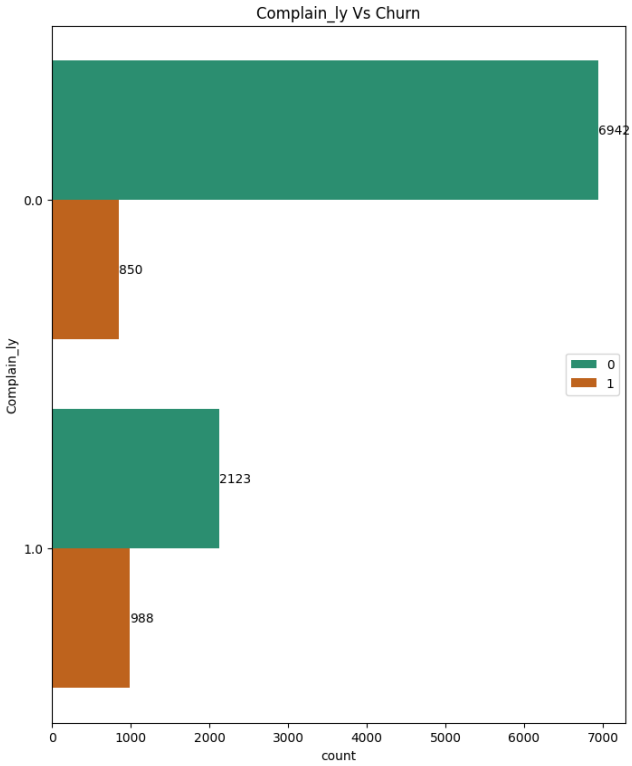
```
# @title `rev_per_month` Vs `Churn` and `Complain_ly` Vs `Churn`
# @markdown **Data Insights:**
```

```
# @markdown > Plot 1 - `rev_per_month` shows positive relation w
# @markdown > Plot 2 - Positive `Complain_ly` has more likely to

fig = plt.figure(figsize=(18, 10))

plt.subplot(121)
ax = sns.countplot(df, y='rev_per_month', hue='Churn', palette=s
ax.legend(loc='center right')
ax.set_title('rev_per_month Vs Churn')
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])

plt.subplot(122)
ax = sns.countplot(df, y='Complain_ly', hue='Churn', palette=sns
ax.legend(loc='center right')
ax.set_title('Complain_ly Vs Churn')
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1]);
```

Plot 1 - `rev_per_month` shows positive relation with `Churn` rate.

Plot 2 - Positive `Complain_ly` has more likely to `Churn`.



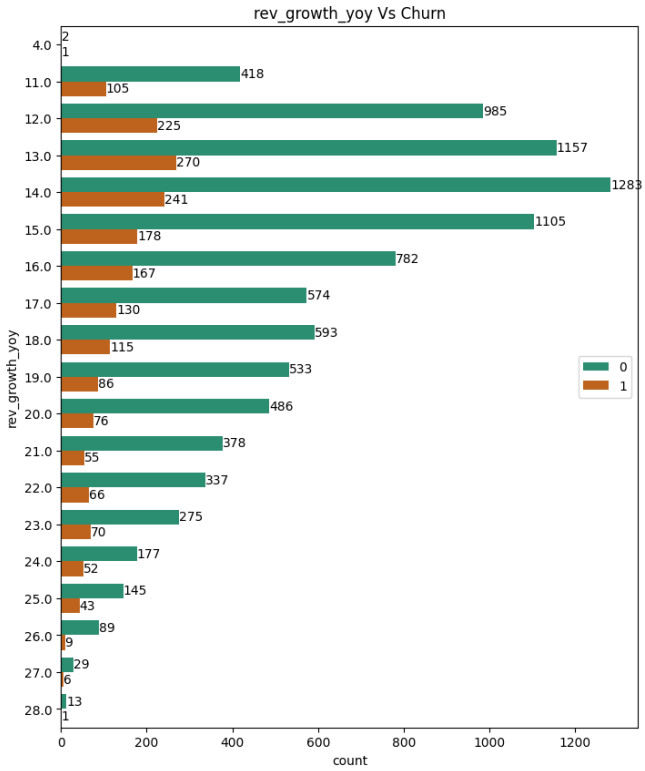


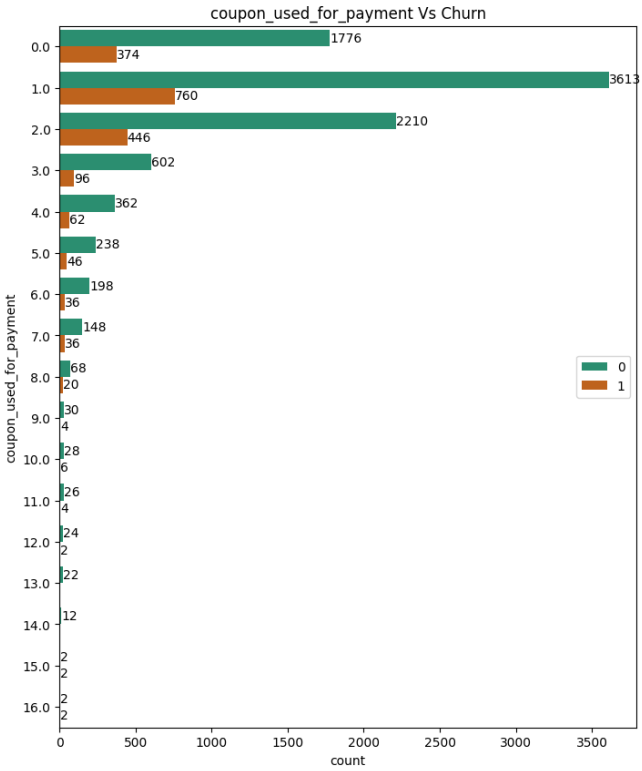
rev_growth_yoy Vs Churn and coupon_used_for_payment Vs Churn

```
# @title `rev_growth_yoy` Vs `Churn` and `coupon_used_for_paymen
```

Data Insights:

```
# @markdown **Data Insights:**
# @markdown > Plot 1 - `rev_growth_yoy` shows positive relation with `Churn` rate.
# @markdown > Plot 2 - `coupon_used_for_payment` shows positive relation with `Churn` rate.

fig = plt.figure(figsize=(18, 10))

plt.subplot(121)
ax = sns.countplot(df, y='rev_growth_yoy', hue='Churn', palette='
ax.legend(loc='center right')
ax.set_title('rev_growth_yoy Vs Churn')
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])

plt.subplot(122)
ax = sns.countplot(df, y='coupon_used_for_payment', hue='Churn',
ax.legend(loc='center right')
ax.set_title('coupon_used_for_payment Vs Churn')
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1]);
```





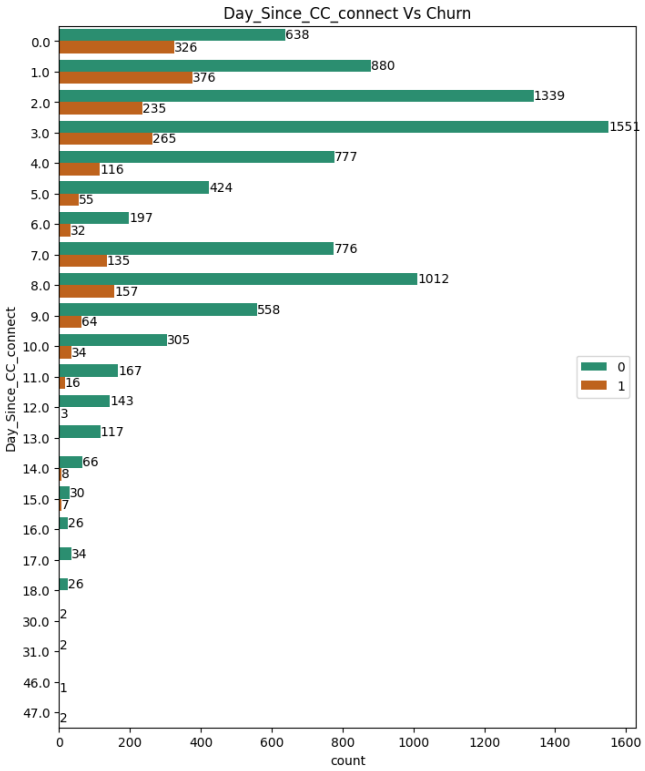
Day_Since_CC_connect Vs Churn and Login_device Vs Churn

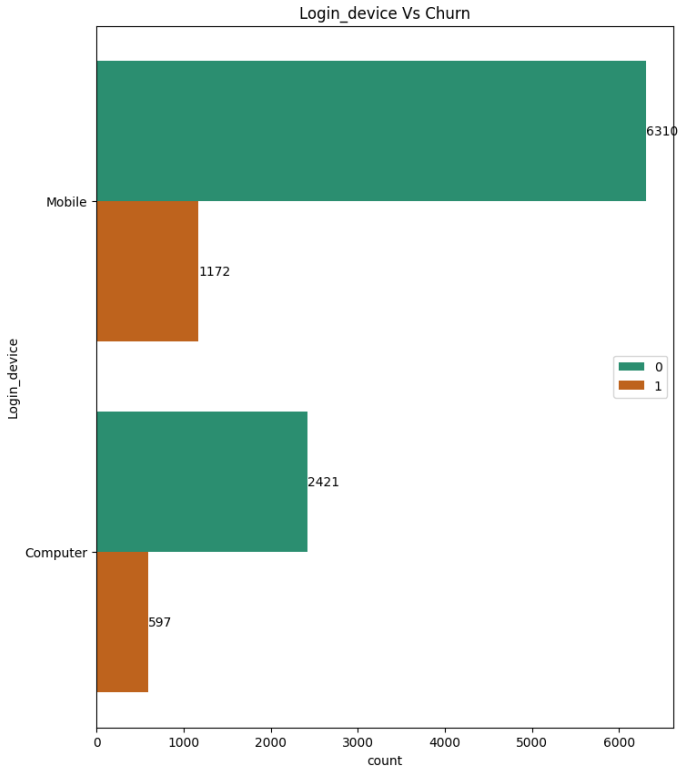
```
# @title `Day_Since_CC_connect` Vs `Churn` and `Login_device` Vs
# @markdown **Data Insights:**
# @markdown > Plot 1 - `Day_Since_CC_connect` shows positive rel
```

```
# @markdown > Plot 1 - `Day_Since_CC_connect` shows positive rel
# @markdown > Plot 2 - `Login_device` shows not much of the rela

fig = plt.figure(figsize=(18, 10))

plt.subplot(121)
ax = sns.countplot(df, y='Day_Since_CC_connect', hue='Churn', pa
ax.legend(loc='center right')
ax.set_title('Day_Since_CC_connect Vs Churn')
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])

plt.subplot(122)
ax = sns.countplot(df, y='Login_device', hue='Churn', palette=sn
ax.legend(loc='center right')
ax.set_title('Login_device Vs Churn')
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1]);
```

Plot 1 - `Day_Since_CC_connect` shows positive relation with `Churn` rate.

Plot 2 - `Login_device` shows not much of the relation with `Churn` rate.

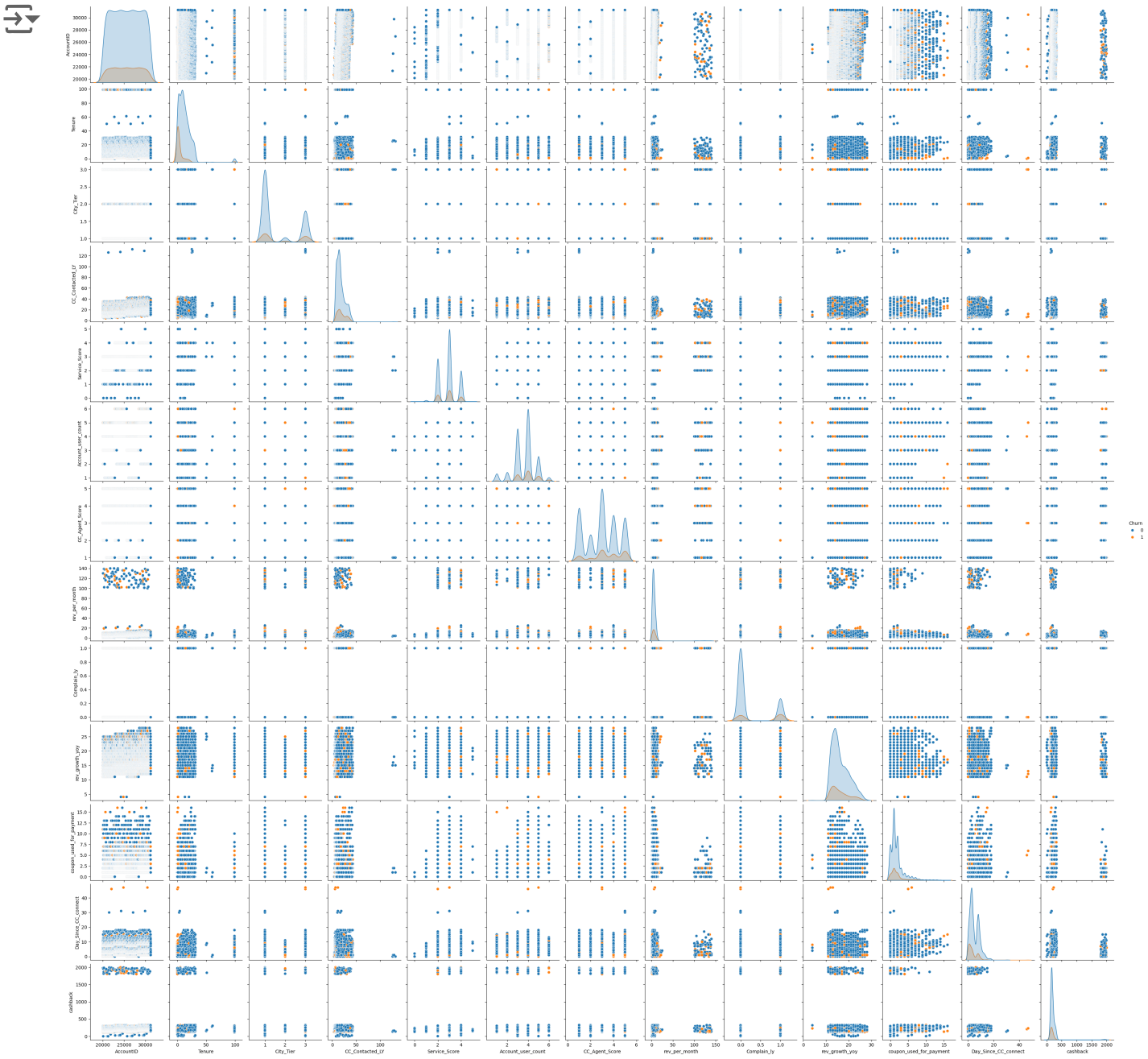




Multivariate Analysis

Pair plot of the dataset

```
sns.pairplot(df, hue='Churn', diag_kind='kde');
```

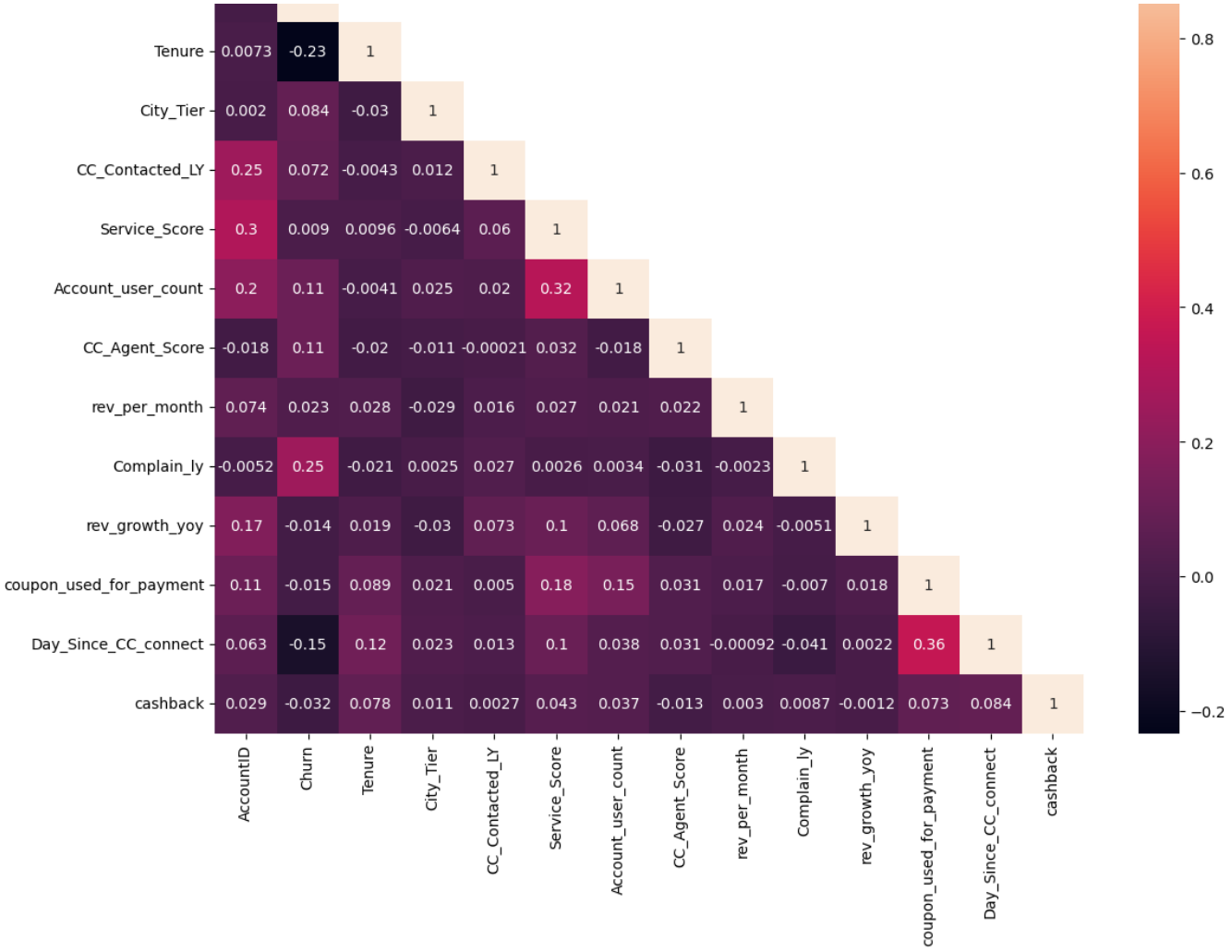


> **Data Insights:**

The pair-plot shown above indicated that the independent variable are week or poor predictors of target variable as we the density of independent variable overlaps with the density of target variable.

Correlation Matrix

```
cor = df.select_dtypes(exclude='O').corr()
mask = np.triu(cor, +1)
fig,ax= plt.subplots(figsize=(13, 10))
sns.heatmap(cor, mask=mask, annot=True, fmt='.2g');
```


Tenure 0.0073 -0.23 1
City_Tier 0.002 0.084 -0.03 1
CC_Contacted_LY 0.25 0.072 -0.0043 0.012 1
Service_Score 0.3 0.009 0.0096 -0.0064 0.06 1
Account_user_count 0.2 0.11 -0.0041 0.025 0.02 0.32 1
CC_Agent_Score -0.018 0.11 -0.02 -0.011 -0.00021 0.032 -0.018 1
rev_per_month 0.074 0.023 0.028 -0.029 0.016 0.027 0.021 0.022 1
Complain_ly -0.0052 0.25 -0.021 0.0025 0.027 0.0026 0.0034 -0.031 -0.0023 1
rev_growth_yoy 0.17 -0.014 0.019 -0.03 0.073 0.1 0.068 -0.027 0.024 -0.0051 1
coupon_used_for_payment 0.11 -0.015 0.089 0.021 0.005 0.18 0.15 0.031 0.017 -0.007 0.018 1
Day_Since_CC_connect 0.063 -0.15 0.12 0.023 0.013 0.1 0.038 0.031 -0.00092 -0.041 0.0022 0.36 1
cashback 0.029 -0.032 0.078 0.011 0.0027 0.043 0.037 -0.013 0.003 0.0087 -0.0012 0.073 0.084 1
AccountID
Churn
Tenure
City_Tier
CC_Contacted_LY
Service_Score
Account_user_count
CC_Agent_Score
rev_per_month
Complain_ly
rev_growth_yoy
coupon_used_for_payment
Day_Since_CC_connect
cashback
0.8
0.6
0.4
0.2
0.0
−0.2

> **Data Insights:**

The Data doesn't shows much of the correlation with each other except few columns (i.e., `Day_Since_CC_connect` with `coupon_used_for_payment`, `Account_user_count` with `Service_Score` and negative correlation with `Tenure` and `Churn` along with `Day_Since_CC_connect`.

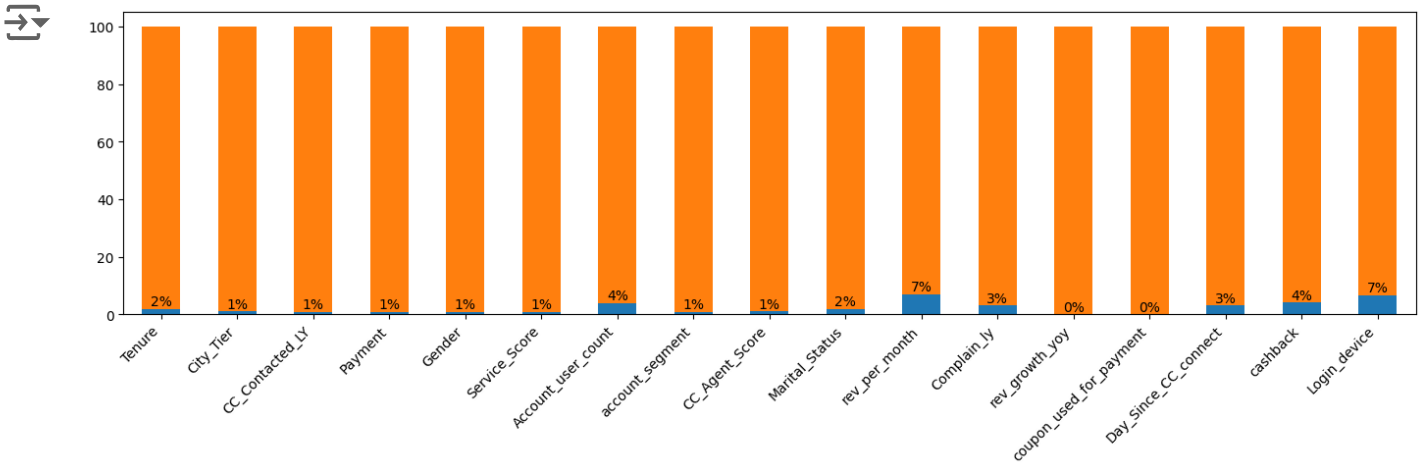
Removing Unwanted column

```
df = df.drop('AccountID', axis=1)

df.head()
```

Handling missing values

Before Treatment

Show code



After Treatment

```
# @title After Treatment

# @markdown > * Out of 19 variables we have null values in 17 va
# @markdown > * Using "Median" to impute null values where varia
# @markdown > * Using "Mode: to impute null values where variabl


for column in df.select_dtypes(exclude='O'):
    df[column].fillna(df[column].median(), inplace=True)


for column in df.select_dtypes(include='O'):
    df[column].fillna(df['Gender'].mode()[0], inplace=True)
```

- Out of 19 variables we have null values in 17 variables
- Using Median to impute null values where variable is continuous in nature because Median is less prone to outliers

```
    df[column].fillna(df['Gender'].mode()[0], inplace=True)


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

when compared with mean.

- Using "Mode: to impute null values where variables are categorical in nature. We have treated null values variable by variable as each and every variable is unique in its nature.

```
Churn                     0
Tenure                    0
City_Tier                 0
CC_Contacted_LY           0
Payment                   0
Gender                    0
Service_Score             0
Account_user_count        0
account_segment           0
CC_Agent_Score            0
Marital_Status            0
rev_per_month             0
Complain_ly               0
rev_growth_yoy            0
coupon_used_for_payment   0
Day_Since_CC_connect      0
cashback                  0
Login_device              0
dtype: int64
```

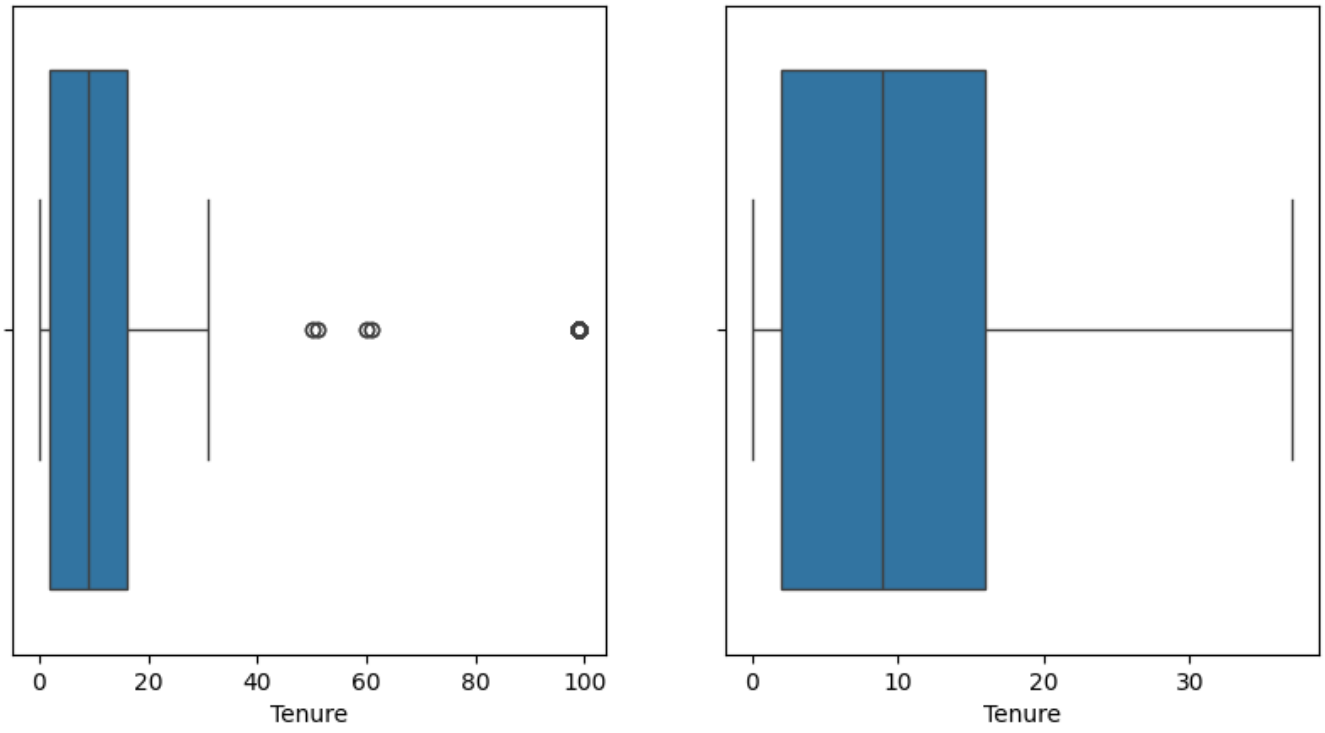
Checking for Outliers on continous variables

```
#treating outlier
def remove_outlier(col):
    sorted(col)
    Q1,Q3=col.quantile([0.25,0.75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
```

Before and After Treatment

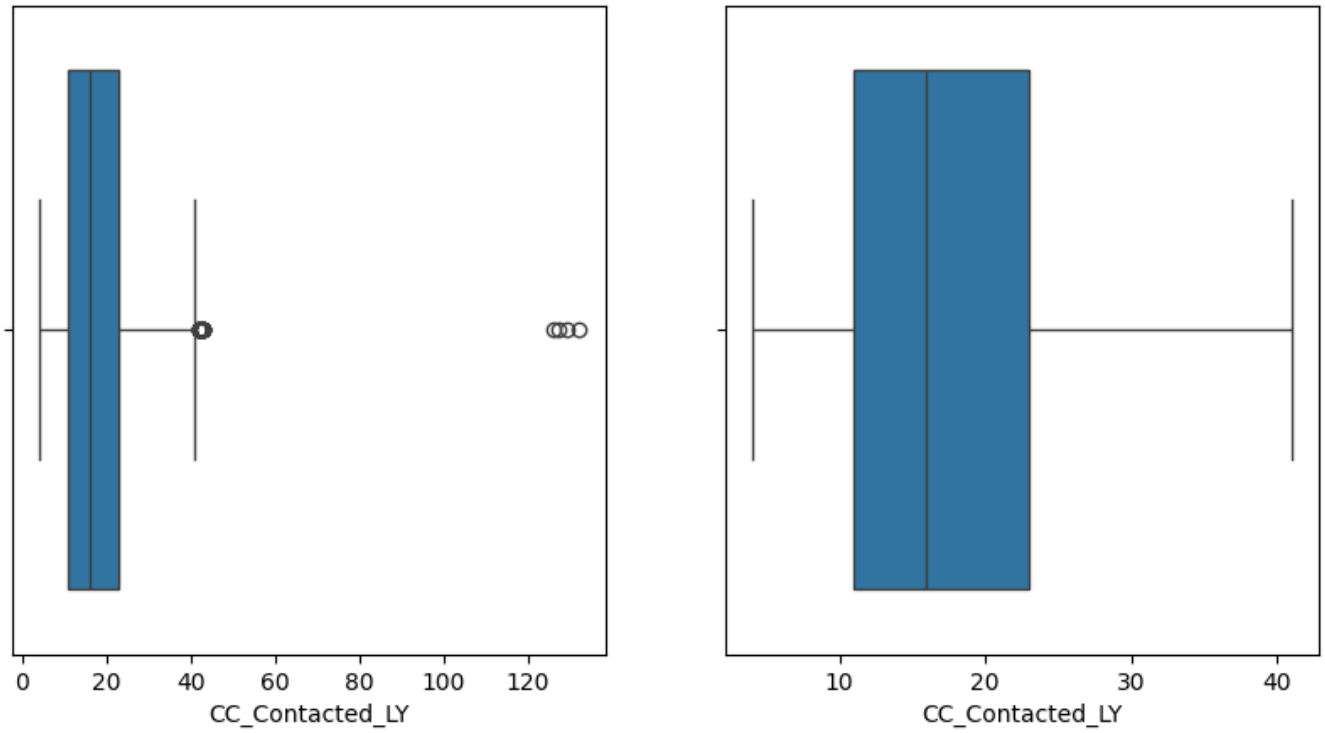
Tenure

```
# @title Tenure

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
sns.boxplot(df['Tenure'], orient='h', ax=ax[0])

lw,up=remove_outlier(df['Tenure'])
df['Tenure']=np.where(df['Tenure']>up,up,df['Tenure'])
df['Tenure']=np.where(df['Tenure']<lw,lw,df['Tenure'])

sns.boxplot(df['Tenure'], orient='h', ax=ax[1]);
```



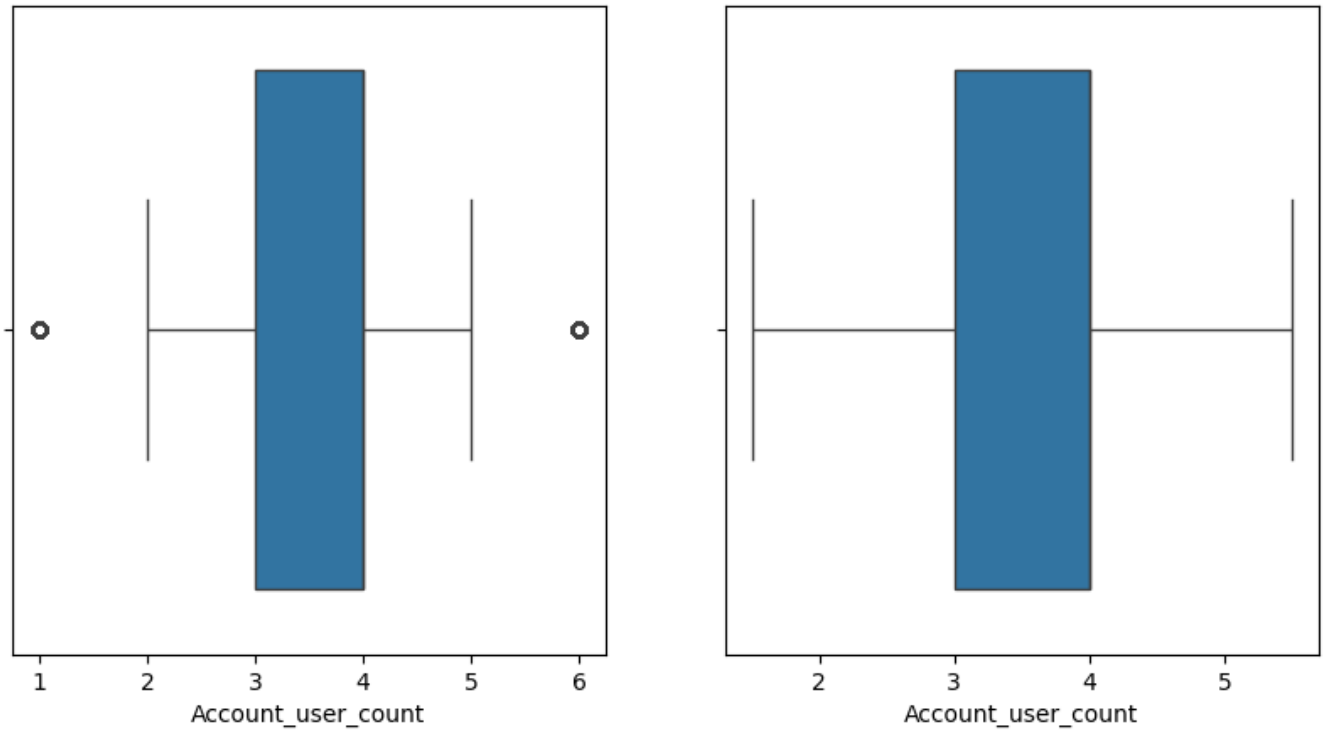
CC_Contacted_LY

```
# @title CC_Contacted_LY

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
sns.boxplot(df['CC_Contacted_LY'], orient='h', ax=ax[0])

lw,up=remove_outlier(df['CC_Contacted_LY'])
df['CC_Contacted_LY']=np.where(df['CC_Contacted_LY']>up,up,df['CC_C
df['CC_Contacted_LY']=np.where(df['CC_Contacted_LY']<lw,lw,df['CC_C

sns.boxplot(df['CC_Contacted_LY'], orient='h', ax=ax[1]);
```



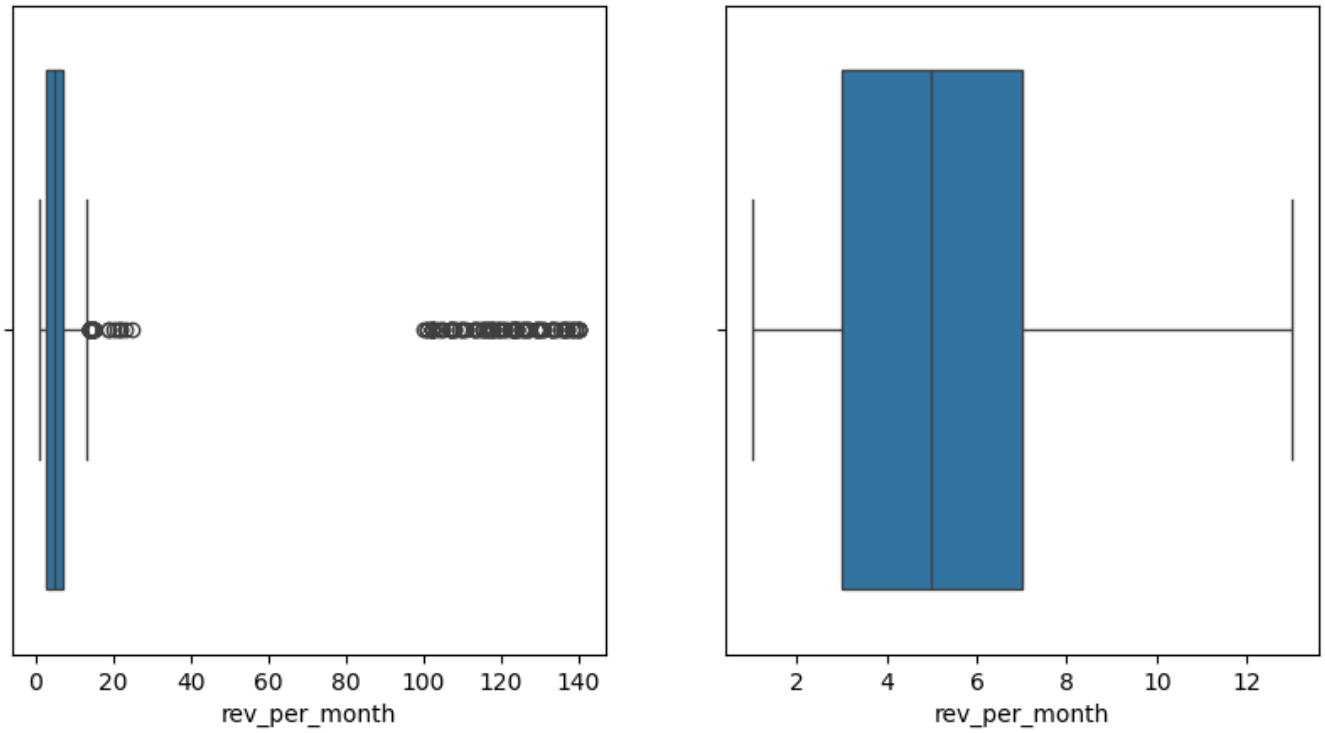
Account_user_count

```
# @title Account_user_count

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
sns.boxplot(df['Account_user_count'], orient='h', ax=ax[0])

lw,up=remove_outlier(df['Account_user_count'])
df['Account_user_count']=np.where(df['Account_user_count']>up,up,d
df['Account_user_count']=np.where(df['Account_user_count']<lw,lw,d

sns.boxplot(df['Account_user_count'], orient='h', ax=ax[1]);
```



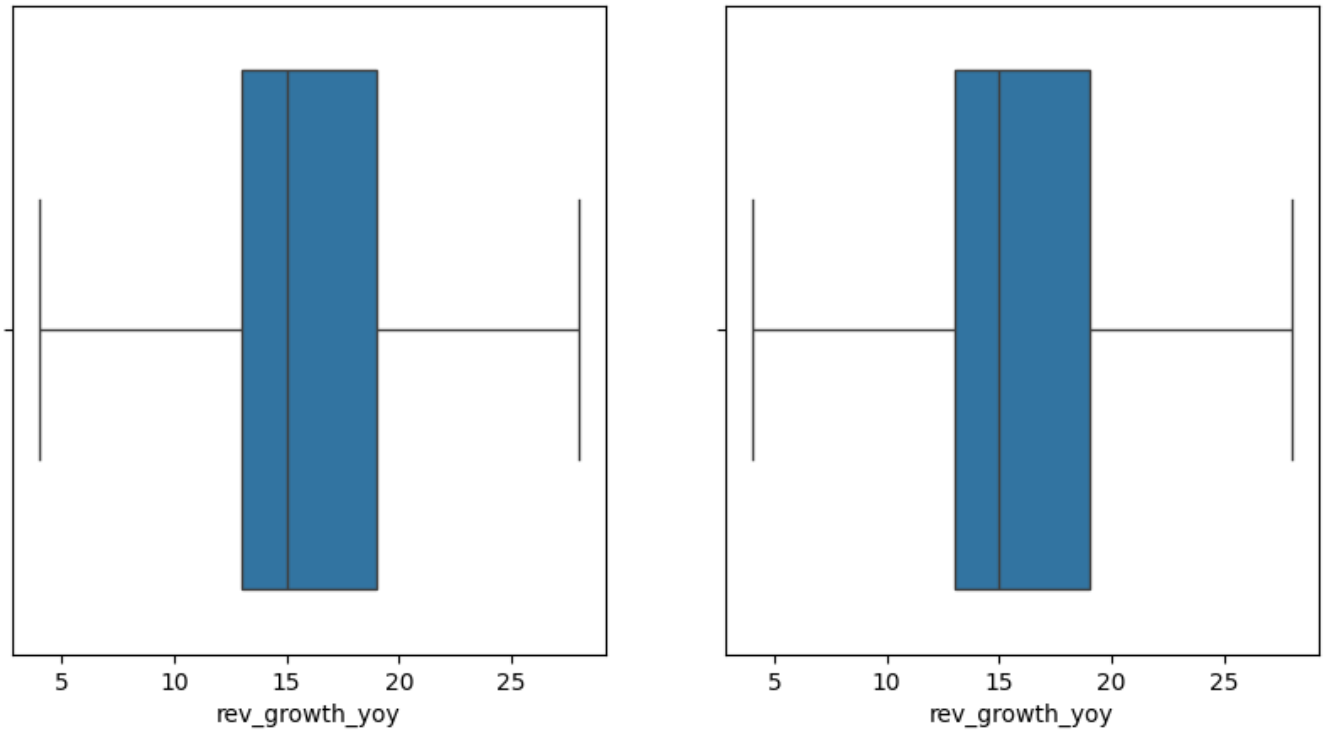
rev_per_month

```
# @title rev_per_month

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
sns.boxplot(df['rev_per_month'], orient='h', ax=ax[0])

lw,up=remove_outlier(df['rev_per_month'])
df['rev_per_month']=np.where(df['rev_per_month']>up,up,df['rev_per
df['rev_per_month']=np.where(df['rev_per_month']<lw,lw,df['rev_per

sns.boxplot(df['rev_per_month'], orient='h', ax=ax[1]);
```



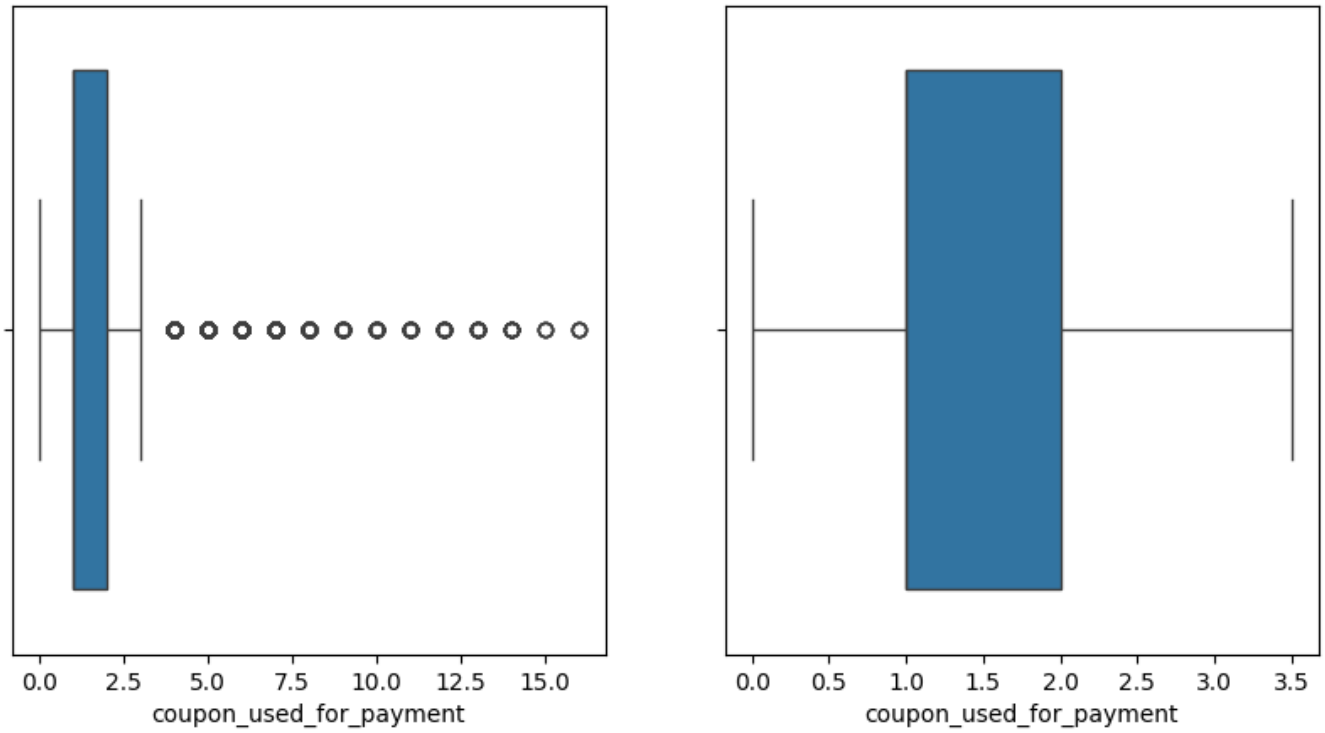
rev_growth_yoy

```
# @title rev_growth_yoy

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
sns.boxplot(df['rev_growth_yoy'], orient='h', ax=ax[0])

lw,up=remove_outlier(df['rev_growth_yoy'])
df['rev_growth_yoy']=np.where(df['rev_growth_yoy']>up,up,df['rev_gr
df['rev_growth_yoy']=np.where(df['rev_growth_yoy']<lw,lw,df['rev_gr

sns.boxplot(df['rev_growth_yoy'], orient='h', ax=ax[1]);
```



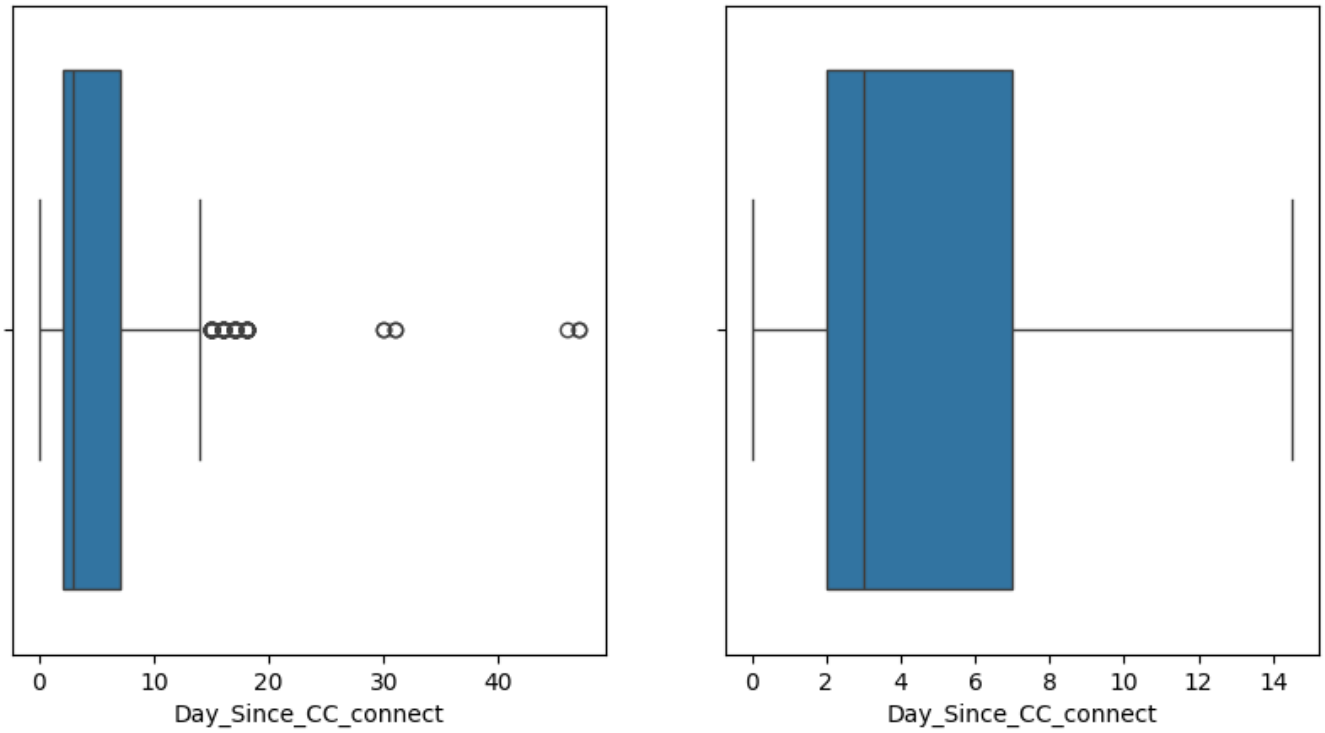
coupon_used_for_payment

```
# @title coupon_used_for_payment

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
sns.boxplot(df['coupon_used_for_payment'], orient='h', ax=ax[0])

lw,up=remove_outlier(df['coupon_used_for_payment'])
df['coupon_used_for_payment']=np.where(df['coupon_used_for_payment'
df['coupon_used_for_payment']=np.where(df['coupon_used_for_payment'

sns.boxplot(df['coupon_used_for_payment'], orient='h', ax=ax[1]);
```



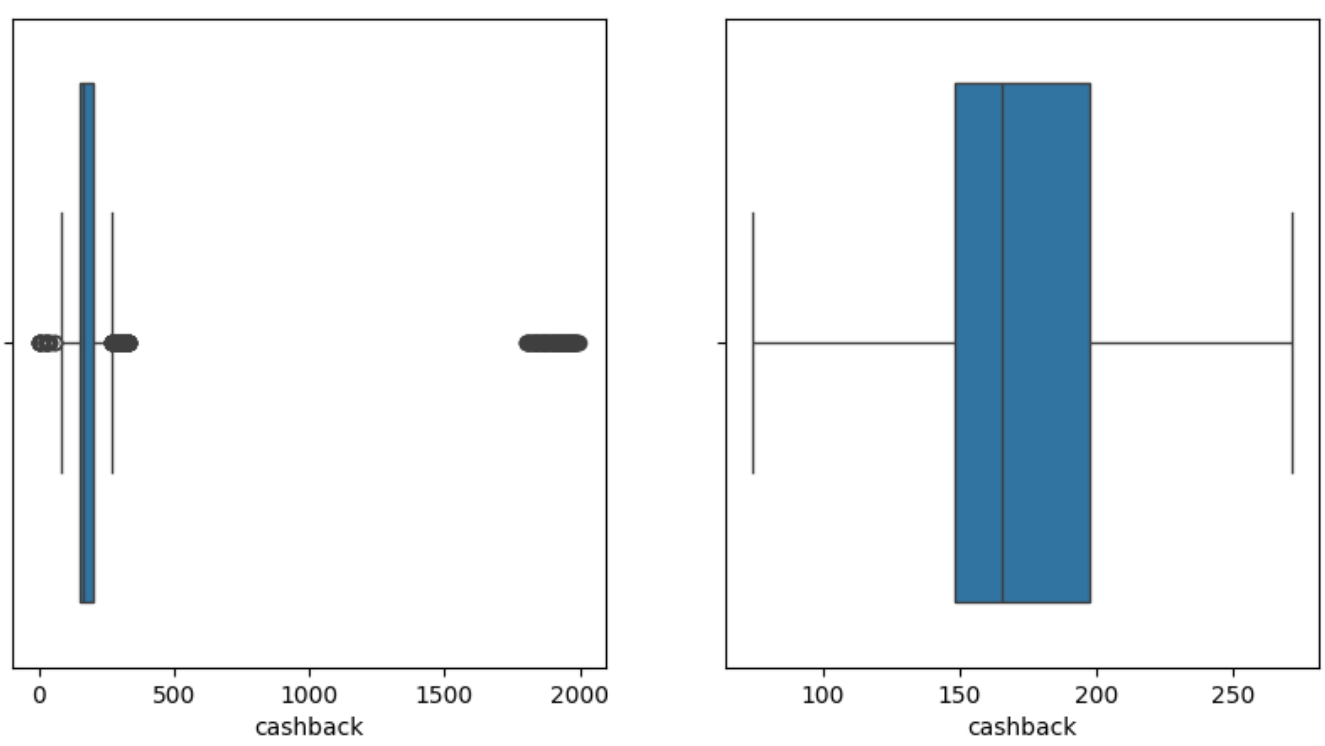
Day_Since_CC_connect

```
# @title Day_Since_CC_connect

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
sns.boxplot(df['Day_Since_CC_connect'], orient='h', ax=ax[0])

lw,up=remove_outlier(df['Day_Since_CC_connect'])
df['Day_Since_CC_connect']=np.where(df['Day_Since_CC_connect']>up,u
df['Day_Since_CC_connect']=np.where(df['Day_Since_CC_connect']<lw,l

sns.boxplot(df['Day_Since_CC_connect'], orient='h', ax=ax[1]);
```



cashback

```
# @title cashback

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
sns.boxplot(df['cashback'], orient='h', ax=ax[0])

lw,up=remove_outlier(df['cashback'])
df['cashback']=np.where(df['cashback']>up,up,df['cashback'])
df['cashback']=np.where(df['cashback']<lw,lw,df['cashback'])

sns.boxplot(df['cashback'], orient='h', ax=ax[1]);
```



Data Preprocessing

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from imblearn.combine import SMOTEENN
from sklearn.cluster import KMeans
```

Splitting the Dataset

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

> Dividing the dataset into X and y. Where X containing independent variables and y containing dependent variable.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((9008, 17), (2252, 17), (9008, 1), (2252, 1))
```

> Splitting the dataset into `80%` of train and `20%` of test dataset.

Encoding the categorical variables using `LabelEncoder`

```
obj_cols = X_train.select_dtypes('object').columns

for col in X_train.columns:
    if col in obj_cols:
        encoder = LabelEncoder()
        encoder.fit(X_train[col])
        X_train[col] = encoder.transform(X_train[col])
        X_test[col] = encoder.transform(X_test[col])


X_train.head()
```

	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Acc
6848	2.0	1.0	18.0	2	1	2.0	
8131	13.0	1.0	23.0	2	0	3.0	
8139	13.0	1.0	29.0	2	0	2.0	
9984	19.0	1.0	14.0	1	1	4.0	
7701	4.0	1.0	8.0	2	0	3.0	

> Selecting all the columns which are of `object` type such as (`Payment`, `Gender`, `Marital_Status`, etc.,) and converting into numerical data using `LabelEncoder` from `scikit-learn` library.

Scaling the data using `MinMaxScaler`

```
scaling = MinMaxScaler()
scaling.fit(X_train)
X_train = pd.DataFrame(scaling.transform(X_train), columns=X_train
X_test = pd.DataFrame(scaling.transform(X_test), columns=X_test.co
```

```
1 X_train.head()
```

	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Ac
6848	0.054054	0.0	0.378378	0.4	1.0	0.4	
8131	0.351351	0.0	0.513514	0.4	0.0	0.6	
8139	0.351351	0.0	0.675676	0.4	0.0	0.4	
9984	0.513514	0.0	0.270270	0.2	1.0	0.8	
7701	0.108108	0.0	0.108108	0.4	0.0	0.6	

> **Data Insights**

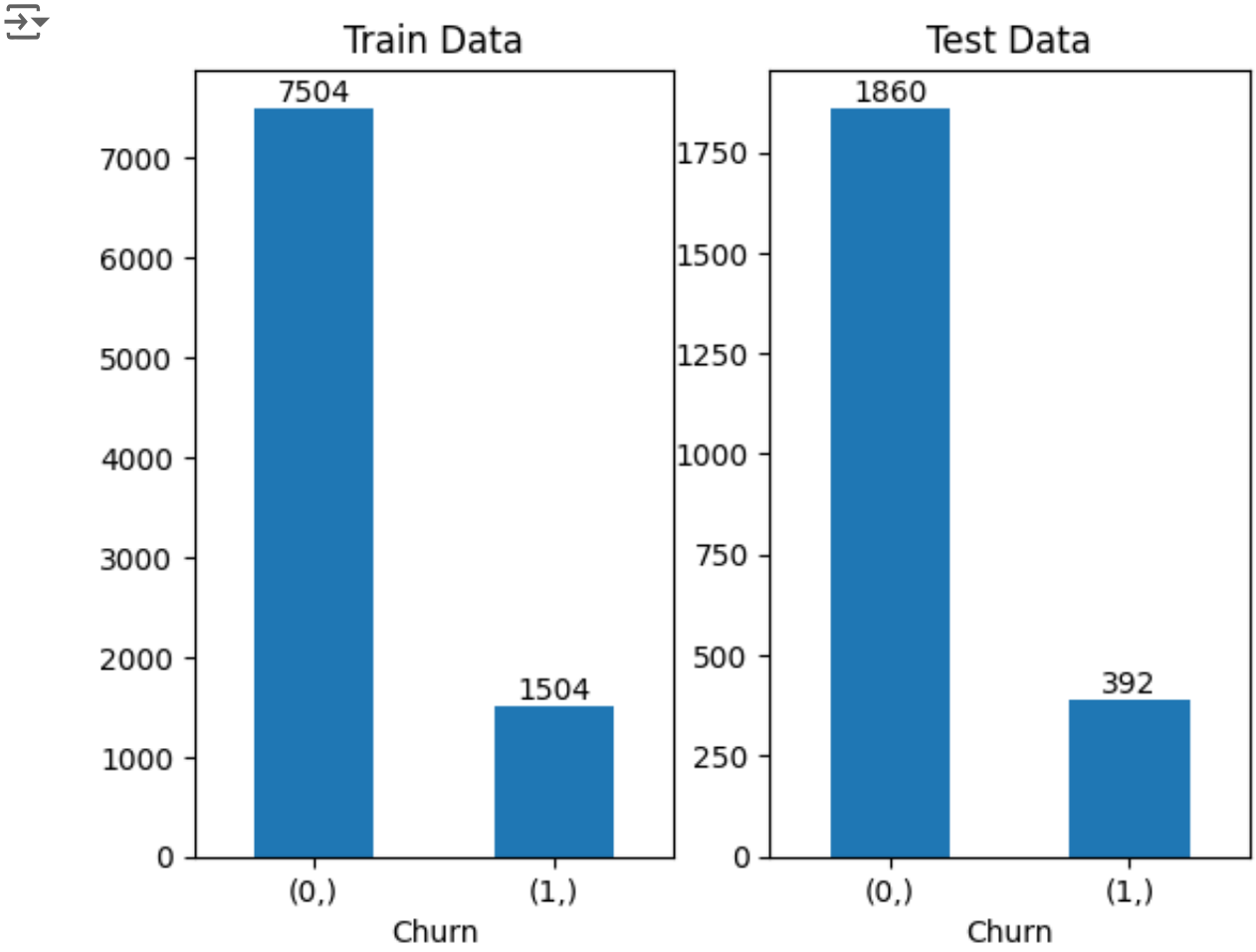
Since the data doesn't follow uniform distribution or bell curve shape, we need to use normalization instead of standardization for better results.

Data balancing using `SMOTE`

> Since, the data is imbalanced we need to try to balance the `Churn`.

Before sampling

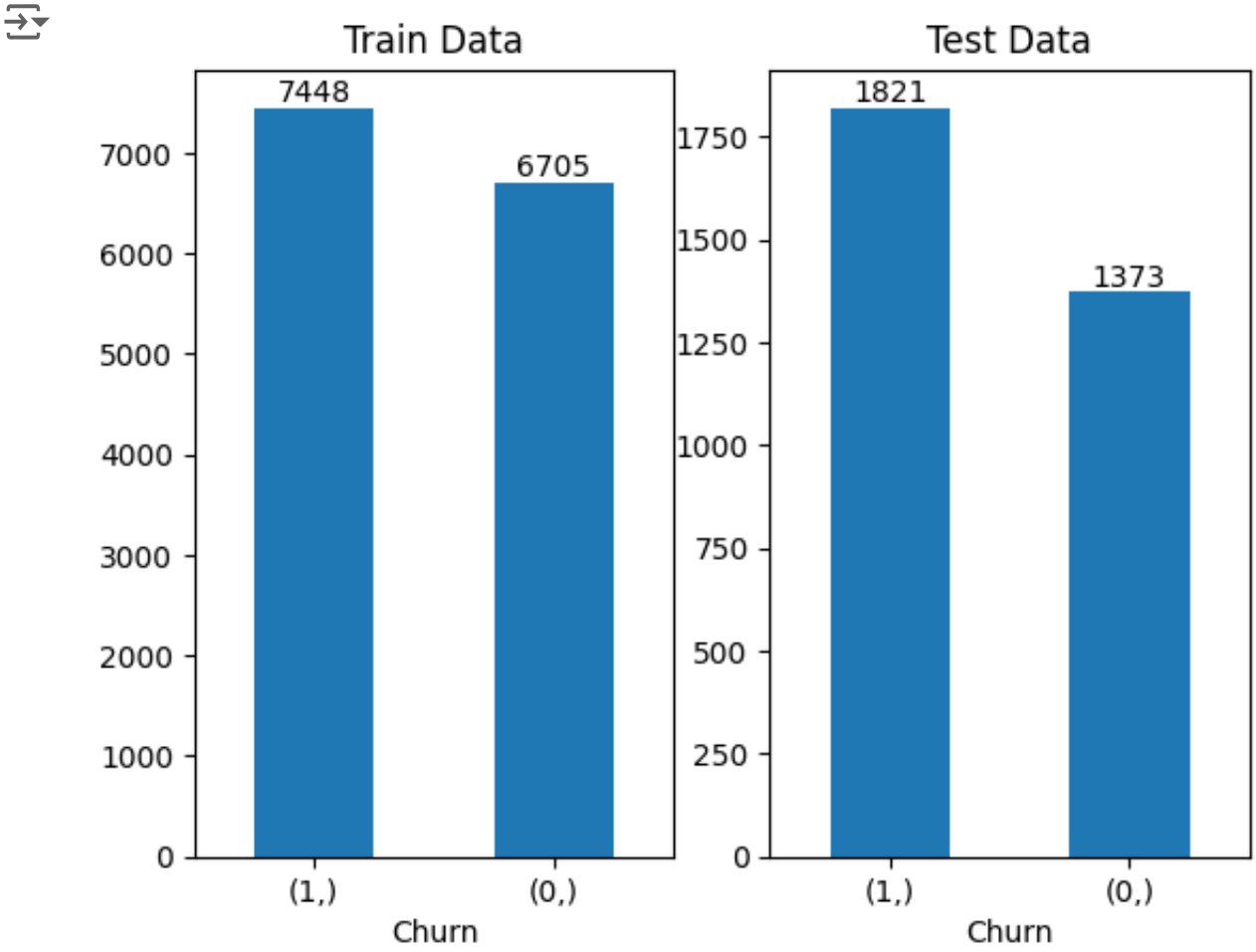
```
# @title Before sampling

plt.subplot(121)
ax = y_train.value_counts().plot(kind='bar', rot='horizontal')
ax.bar_label(ax.containers[0])
ax.set_title('Train Data')

plt.subplot(122)
ax = y_test.value_counts().plot(kind='bar', rot='horizontal')
ax.bar_label(ax.containers[0])
ax.set_title('Test Data');
```




```
sm = SMOTEENN(random_state=8)
sm.fit(X_train, y_train)
X_train_sm, y_train_sm = sm.fit_resample(X_train, y_train)
X_test_sm, y_test_sm = sm.fit_resample(X_test, y_test)
```

After Sampling

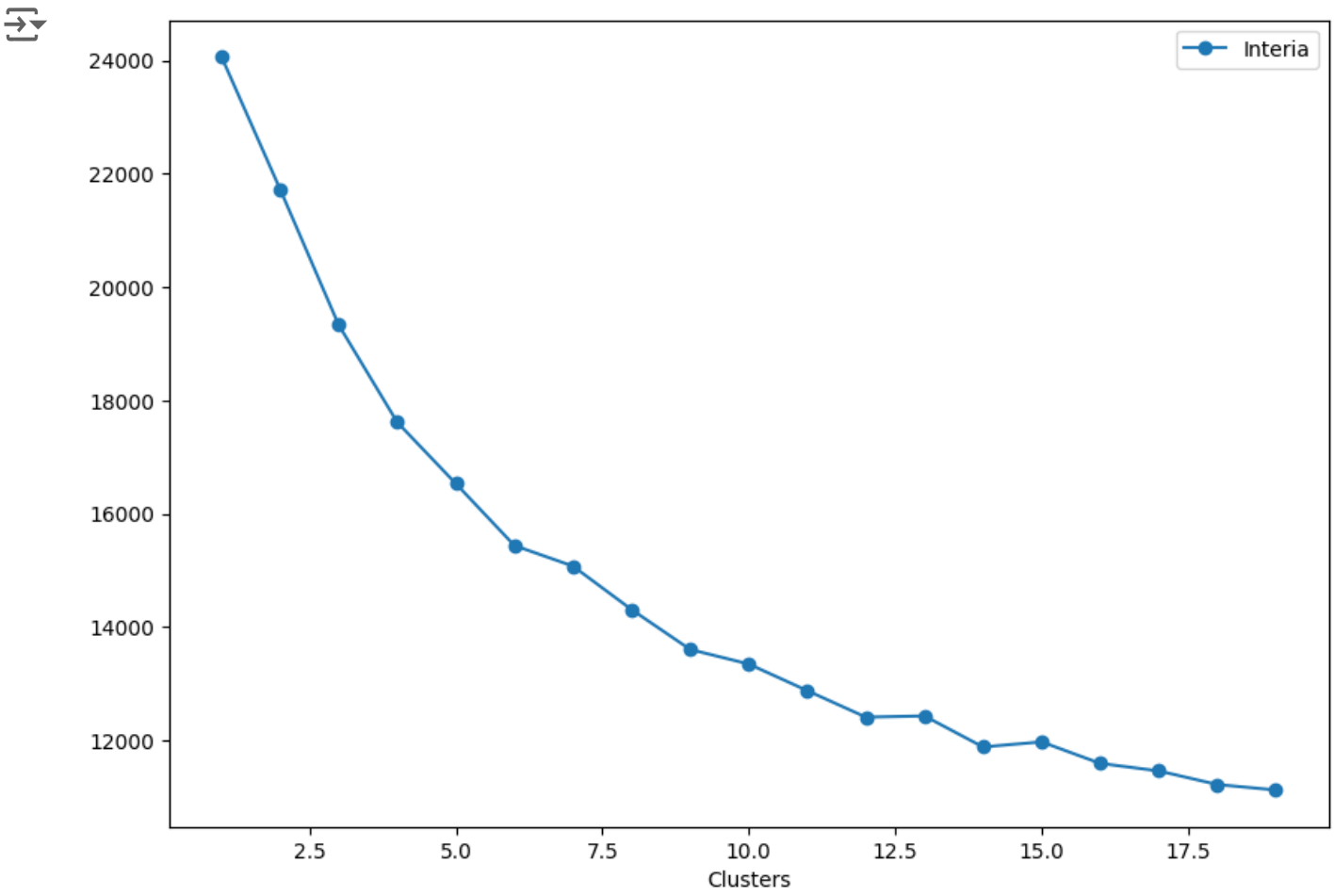
```
# @title After Sampling

plt.subplot(121)
ax = y_train_sm.value_counts().plot(kind='bar', rot='horizontal')
ax.bar_label(ax.containers[0])
ax.set_title('Train Data')

plt.subplot(122)
ax = y_test_sm.value_counts().plot(kind='bar', rot='horizontal')
ax.bar_label(ax.containers[0])
ax.set_title('Test Data');
```



Clustering using k-means

```
interia = []

for i in range(1, 20):
    k_means = KMeans(n_clusters=i, n_init='auto')
    k_means.fit(X_train_sm)
    inter = k_means.inertia_
    interia.append(inter)
```

```
cluster = pd.DataFrame({
    'Clusters': list(range(1, 20)),
    'Interia': interia

})

cluster.plot(x='Clusters', y='Interia', marker='o', figsize=(10, 7
```



```
k_means = KMeans(n_clusters=7, random_state=8, n_init='auto')
k_means.fit(X_train_sm)
train_labels = k_means.predict(X_train_sm)
train_labels
```

```
array([2, 0, 5, ..., 4, 4, 6], dtype=int32)
```

```
test_labels = k_means.predict(X_test_sm)
test_labels[:10]
```

```
array([3, 1, 4, 3, 4, 3, 4, 3, 2, 1], dtype=int32)
```

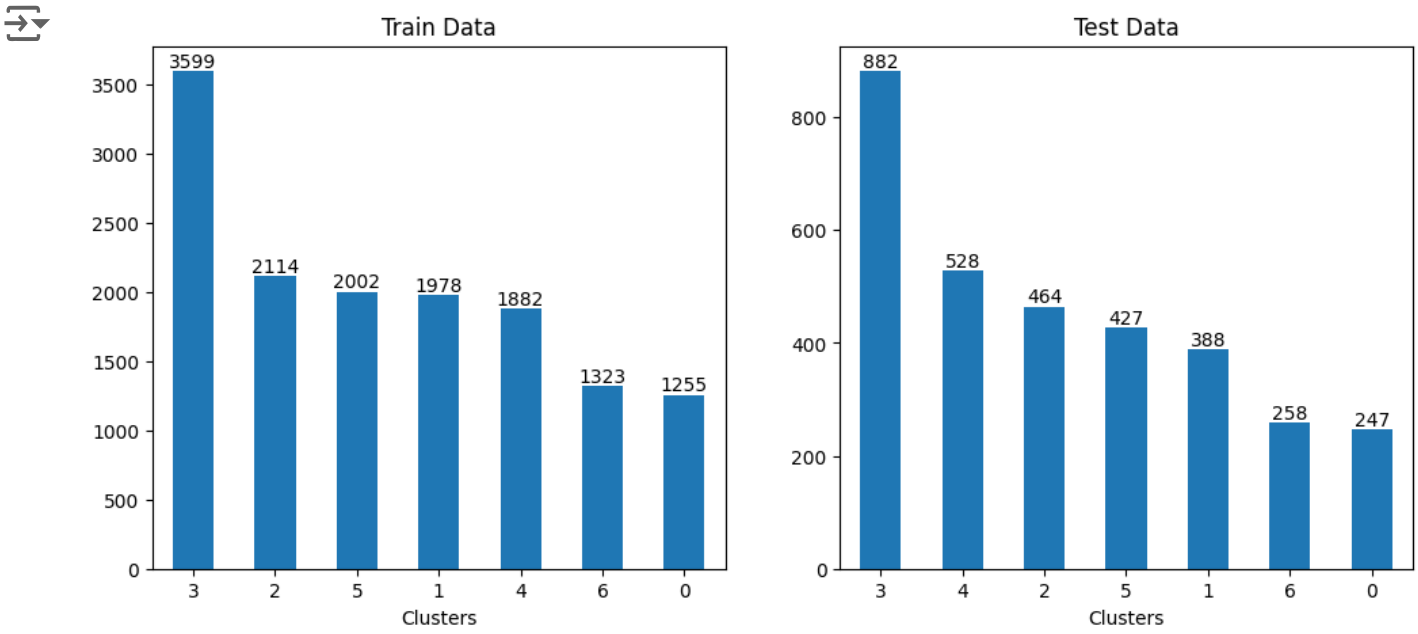
```
X_train_means = X_train_sm.copy()
X_train_means['Clusters'] = train_labels

X_test_means = X_test_sm.copy()
X_test_means['Clusters'] = test_labels
```

```
X_train_final = X_train_means.copy()
X_test_final = X_test_means.copy()

y_train_final = y_train_sm.copy()
y_test_final = y_test_sm.copy()
```

```
X_train_final.head()
```

	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Accou
0	0.351351	0.0	0.513514	0.4	0.0	0.6	
1	0.351351	0.0	0.675676	0.4	0.0	0.4	
2	0.513514	0.0	0.270270	0.2	1.0	0.8	
3	0.108108	0.0	0.108108	0.4	0.0	0.6	
4	0.216216	0.0	0.243243	0.4	1.0	0.4	

```
fig = plt.figure(figsize=(12, 5))

plt.subplot(121)
ax = X_train_final['Clusters'].value_counts().plot(kind='bar', rot
ax.bar_label(ax.containers[0])
ax.set_title('Train Data')

plt.subplot(122)
ax = X_test_final['Clusters'].value_counts().plot(kind='bar', rot=
ax.bar_label(ax.containers[0])
ax.set_title('Test Data');
```



Training the model

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import (GradientBoostingClassifier,
                              HistGradientBoostingClassifier,
                              RandomForestClassifier,
                              AdaBoostClassifier,
                              BaggingClassifier,
                              StackingClassifier)
from xgboost import XGBClassifier


from sklearn.model_selection import cross_val_score, GridSearchCV,
from sklearn.metrics import roc_curve, RocCurveDisplay, roc_auc_sc
from sklearn.metrics import confusion_matrix, ConfusionMatrixDispl
from sklearn.metrics import precision_recall_fscore_support, accur
```

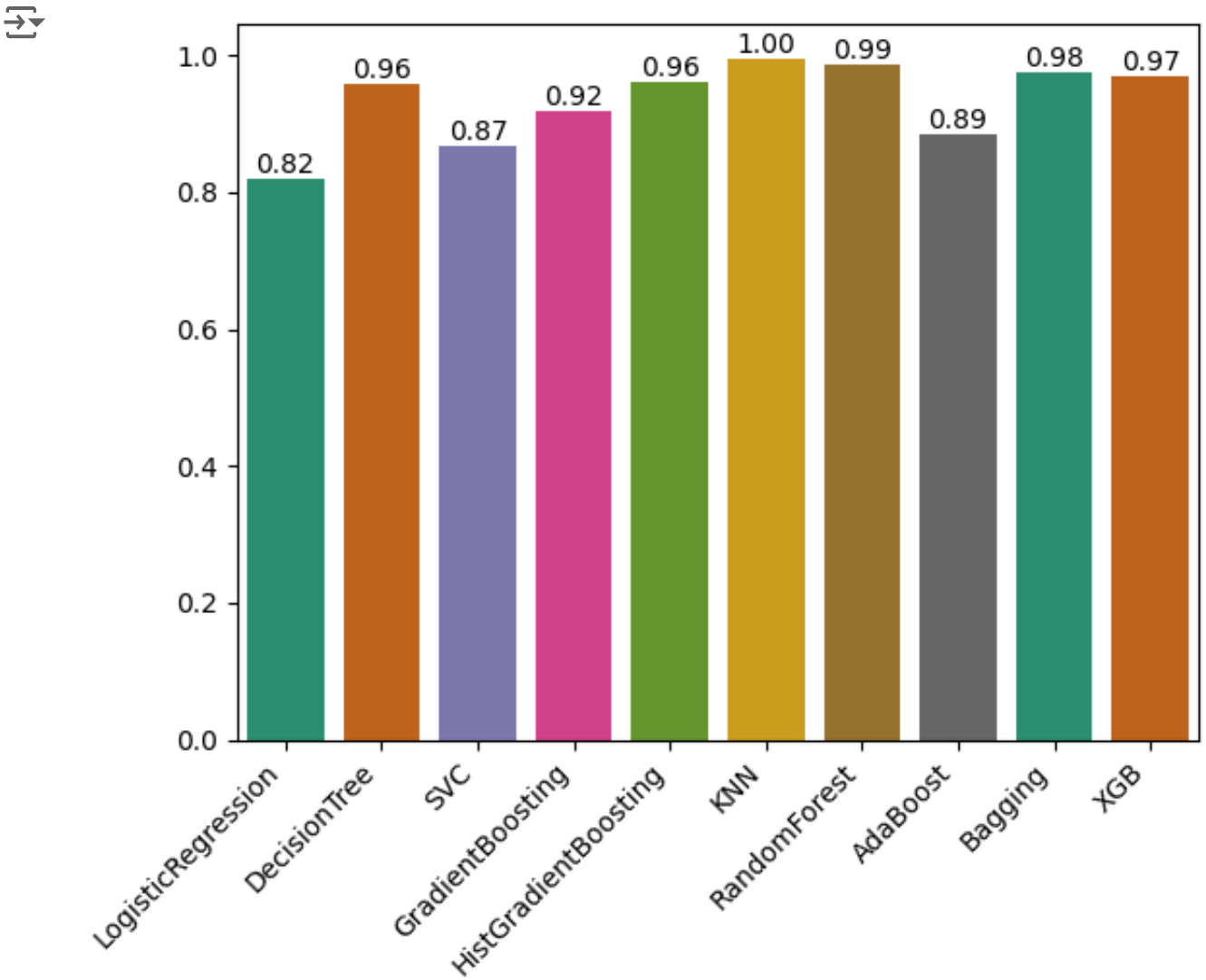
```
models = {
    'LogisticRegression': LogisticRegression(random_state=8),
    'DecisionTree': DecisionTreeClassifier(random_state=8),
    'SVC': SVC(random_state=8),
    'GradientBoosting': GradientBoostingClassifier(random_state=8)
    'HistGradientBoosting': HistGradientBoostingClassifier(random_
    'KNN': KNeighborsClassifier(n_neighbors=2),
    'RandomForest': RandomForestClassifier(random_state=8),
    'AdaBoost': AdaBoostClassifier(random_state=8),
    'Bagging': BaggingClassifier(random_state=8),
    'XGB': XGBClassifier(),
}
```

```
# Calculates the accuracy, precision, recall, and f1-score score
def calculate_results(name, y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision, recall, fscore, _ = precision_recall_fscore_support


    return {
        'Model Name': name,
        'Accuracy score': accuracy,
        'Precision score': precision,
        'Recall score': recall,
        'F1 score': fscore,
    }
```

```
train_results = []
test_results = []

for name, model in models.items():
    result = cross_val_score(model, X=X_train_final, y=y_train_fin
    train_results.append(np.mean(result))

    model.fit(X_train_final, y_train_final)
    y_pred = model.predict(X_test_final)

    test_results.append(calculate_results(name, y_test_final, y_pr
```

Train data score

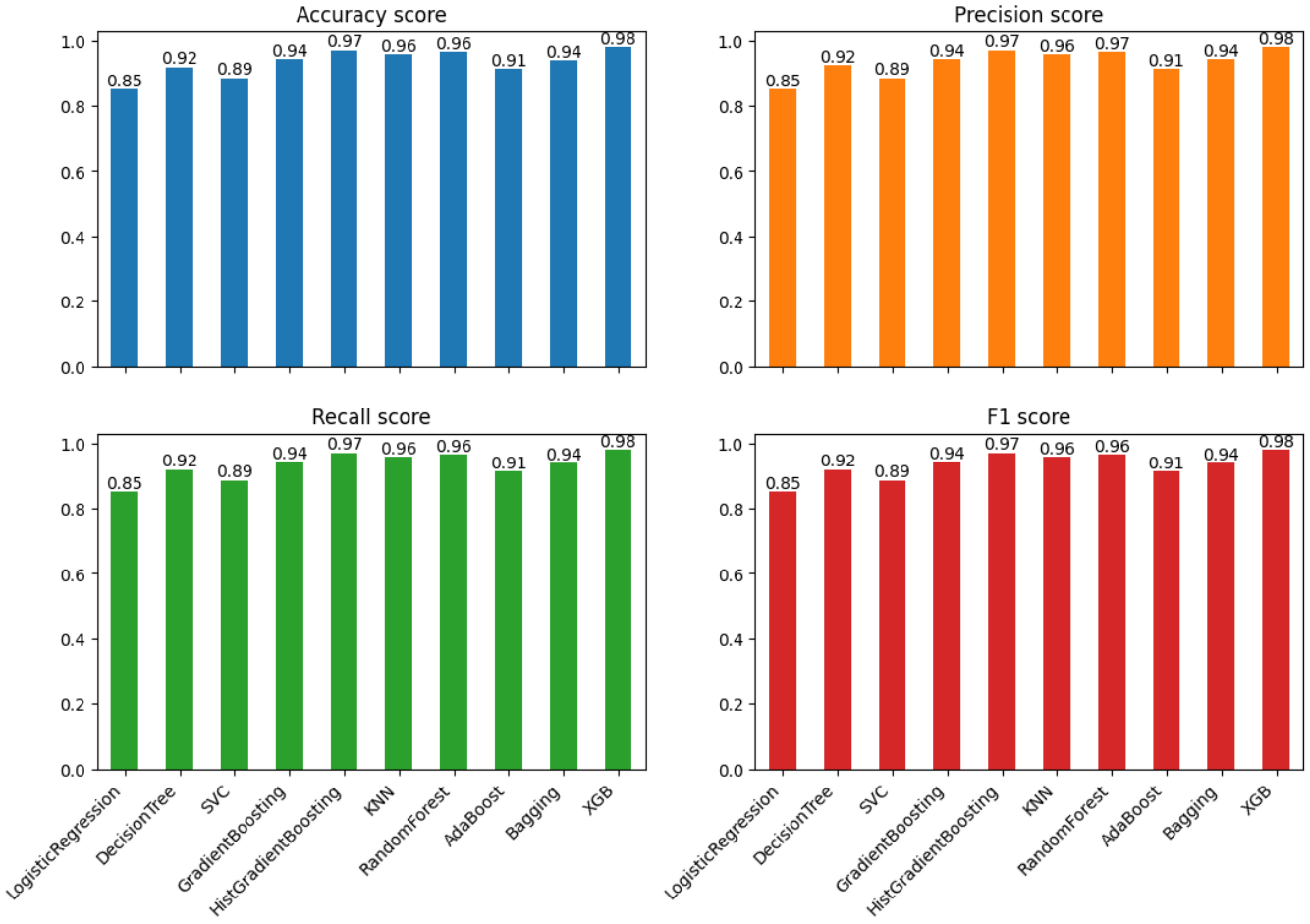
```
# @title Train data score

ax = sns.barplot(x=list(models.keys()), y=train_results, palette='
for container in ax.containers:
    ax.bar_label(container, fmt='%.2f')
ax.set_xticklabels(list(models.keys()), rotation=45, ha='right');
```



Test data score

```
# @title Test data score

axis = pd.DataFrame(test_results).plot(kind='bar', subplots=True,
for ax in axis.flatten():
    ax.get_legend().remove()
```

```
    ax.set_xticklabels(list(models.keys()), rotation=45, ha='right
    ax.bar_label(ax.containers[0], fmt='%.2f');
```



About Data:

> Best Models:
>
> - $DecisionTree$
> - $GradientBoosting$
> - $HistGradientBoosting$
> - KNN
> - $RandomForest$
> - $Bagging$
> - XGB

```
best_models = ['DecisionTree', 'GradientBoosting', 'HistGradientBo
best_models_tuned = ['DecisionTreeTuned', 'GradientBoostingTuned',
```

```
train_results, test_results  = [], []
```

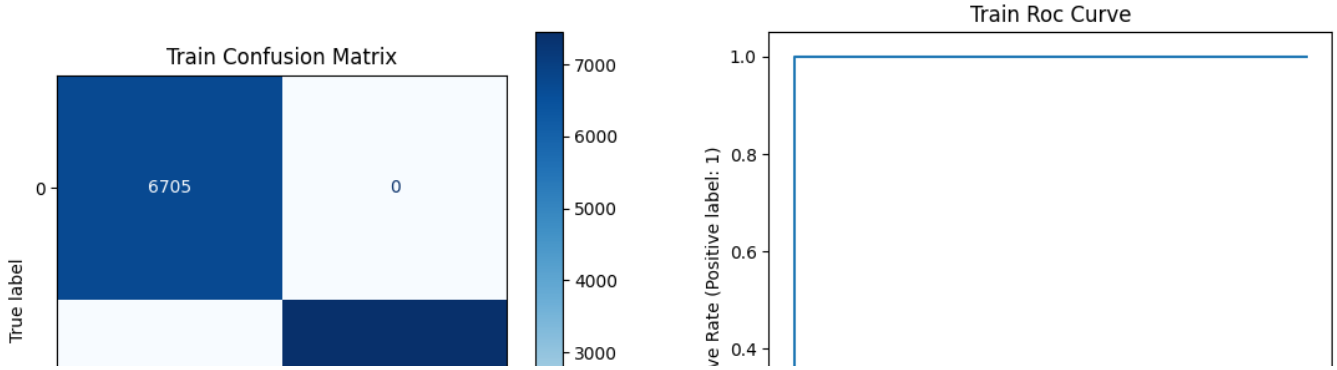
```
def pre_result_calculator(name, model):
    train_pred = model.predict(X_train_final)
    test_pred = model.predict(X_test_final)

    train_score = calculate_results(name, y_train_final, train_pre
    test_score = calculate_results(name, y_test_final, test_pred)

    train_results.append(train_score)
    test_results.append(test_score)

    return (y_train_final, train_pred), \
     (y_test_final, test_pred), \
     pd.DataFrame(train_score, index=[0]), \
     pd.DataFrame(test_score, index=[0])
```

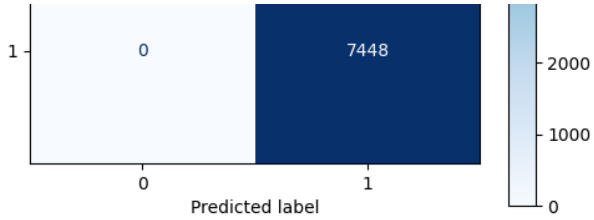
```
def plot(train, test):
    fig, ax = plt.subplots(2, 2, figsize=(12, 12))
    fig.set_tight_layout({'pad': 5.0,
                          'w_pad': 5.0,
                          'h_pad': 5.0})
    ax = ax.flatten()

    ConfusionMatrixDisplay.from_predictions(train[0], train[1], cm
    ax[0].set_title('Train Confusion Matrix')
    RocCurveDisplay.from_predictions(train[0], train[1], ax=ax[1])
    ax[1].set_title('Train Roc Curve')

    ConfusionMatrixDisplay.from_predictions(test[0], test[1], cmap
    ax[2].set_title('Test Confusion Matrix')
    RocCurveDisplay.from_predictions(test[0], test[1], ax=ax[3])
    ax[3].set_title('Test Roc Curve')
```

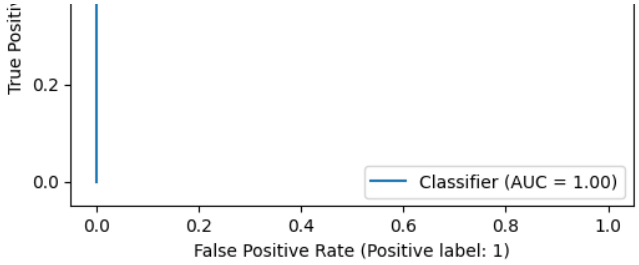
Model 0 - DecisionTree

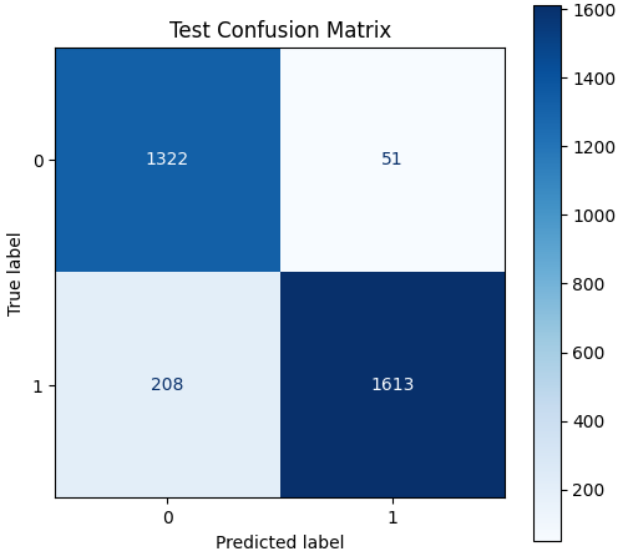
```
dt = DecisionTreeClassifier(random_state=8)
dt.fit(X_train_final, y_train_final)

results = pre_result_calculator(best_models[0], dt)
train, test = results[0], results[1]
train_score, test_score = results[2], results[3]

print(train_score)
print(test_score)
plot(train, test)
```

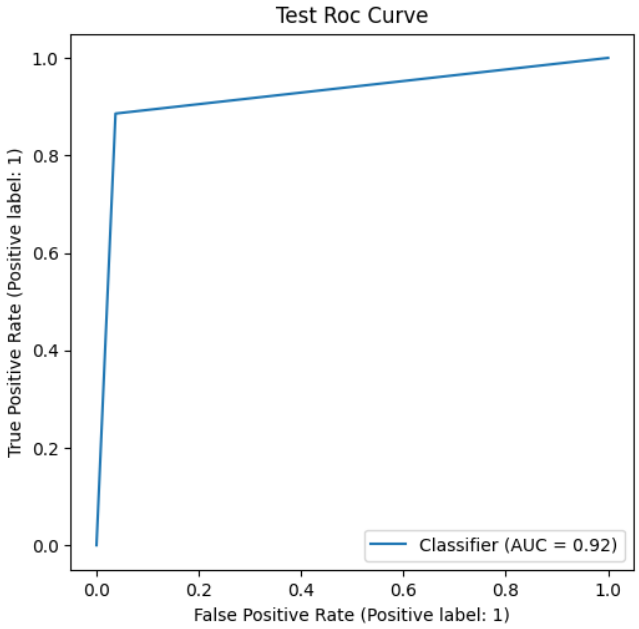
```
      Model Name  Accuracy score  Precision score  Recall score  F1 score
0  DecisionTree             1.0              1.0           1.0       1.0
      Model Name  Accuracy score  Precision score  Recall score  F1 score
0  DecisionTree         0.91891         0.924086       0.91891  0.919277
```




1
0
7448
0
1
Predicted label
2000
1000
0


True Positi
0.2
0.0
Classifier (AUC = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)

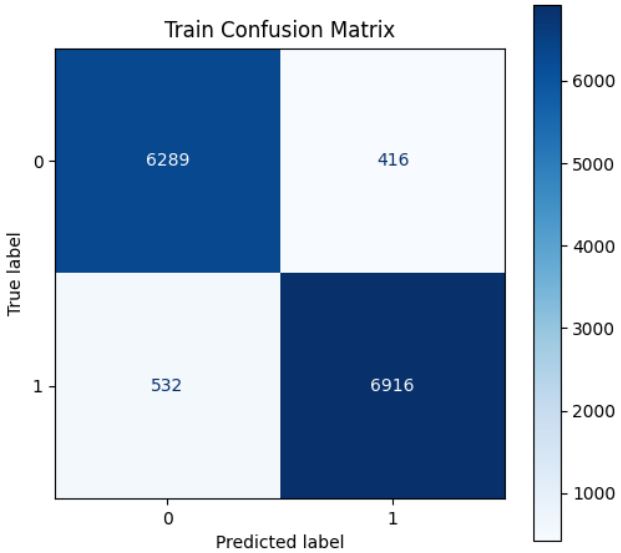

Test Confusion Matrix
0
1322
51
True label
1
208
1613
0
1
Predicted label
1600
1400
1200
1000
800
600
400
200

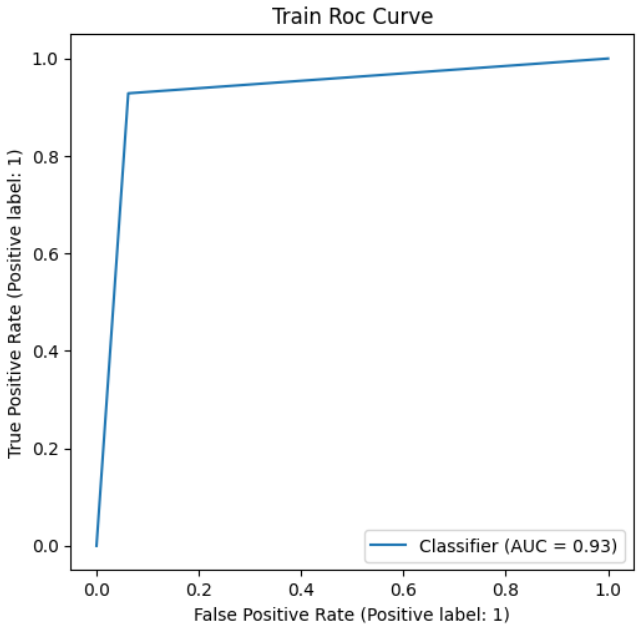

Test Roc Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
Classifier (AUC = 0.92)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)

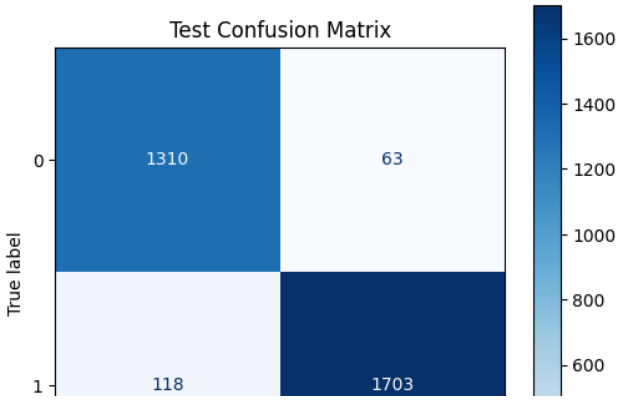
Model 1 - GradientBoosting

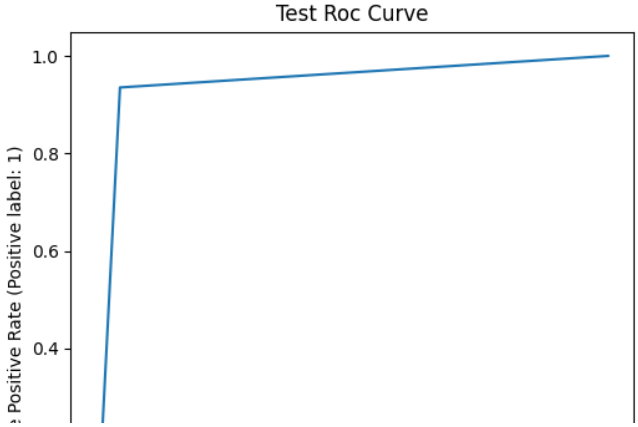
```
gb = GradientBoostingClassifier(random_state=8)
gb.fit(X_train_final, y_train_final)

results = pre_result_calculator(best_models[1], gb)
train, test = results[0], results[1]
train_score, test_score = results[2], results[3]

print(train_score)
print(test_score)
plot(train, test)
```

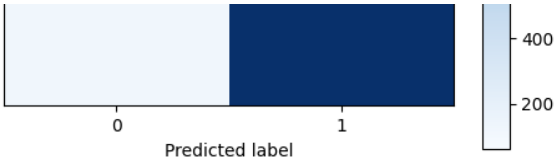
```
          Model Name  Accuracy score  Precision score  Recall score  F1 scor
0  GradientBoosting        0.933018         0.933192      0.933018  0.93304
          Model Name  Accuracy score  Precision score  Recall score  F1 scor
0  GradientBoosting        0.943331          0.94414      0.943331  0.94345
```

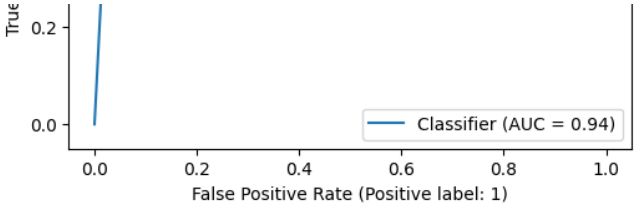










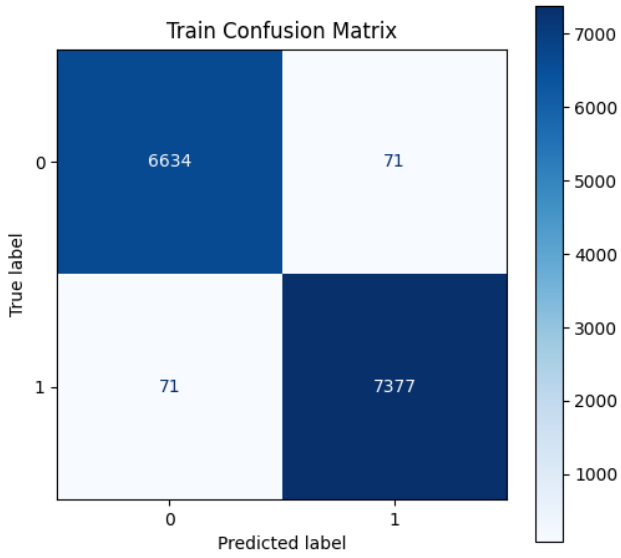


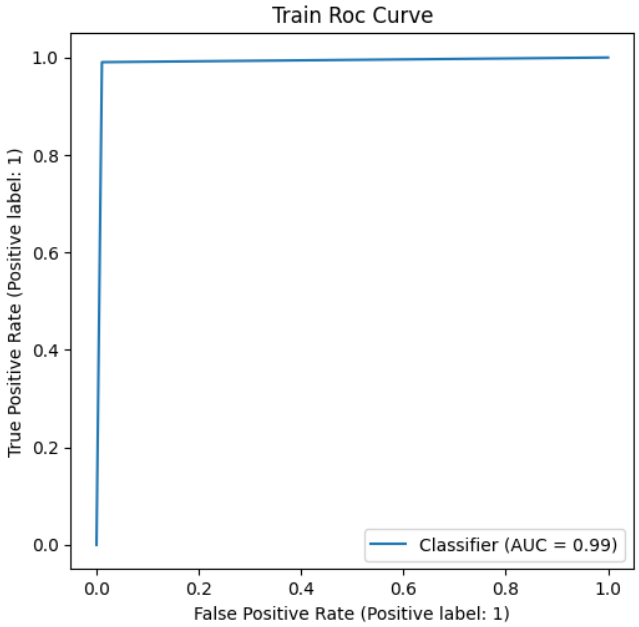
Model 2 - HistGradientBoosting

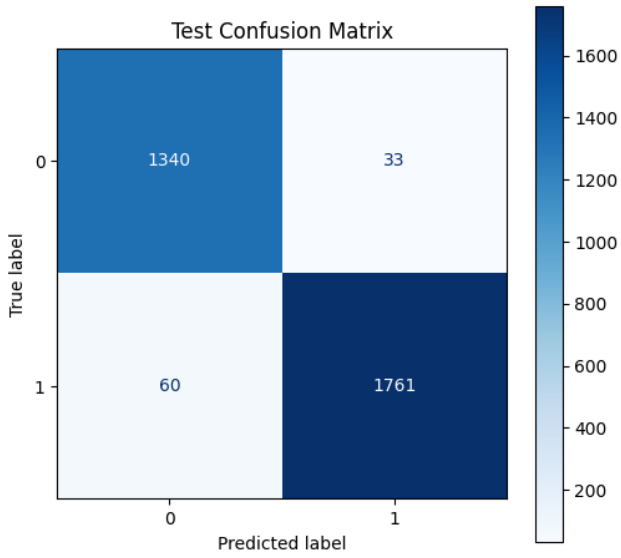
```
hgb = HistGradientBoostingClassifier(random_state=8)
hgb.fit(X_train_final, y_train_final)

results = pre_result_calculator(best_models[2], hgb)
train, test = results[0], results[1]
train_score, test_score = results[2], results[3]

print(train_score)
print(test_score)
plot(train, test)
```

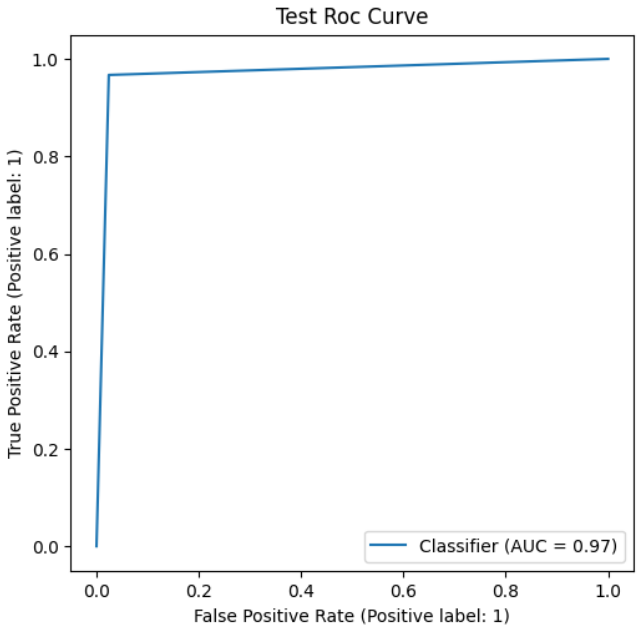
```
              Model Name  Accuracy score  Precision score  Recall score  \
0  HistGradientBoosting        0.989967         0.989967      0.989967
```

```
   F1 score
0  0.989967
            Model Name  Accuracy score  Precision score  Recall score  \
0  HistGradientBoosting        0.970883          0.97109      0.970883

   F1 score
0  0.970916
```





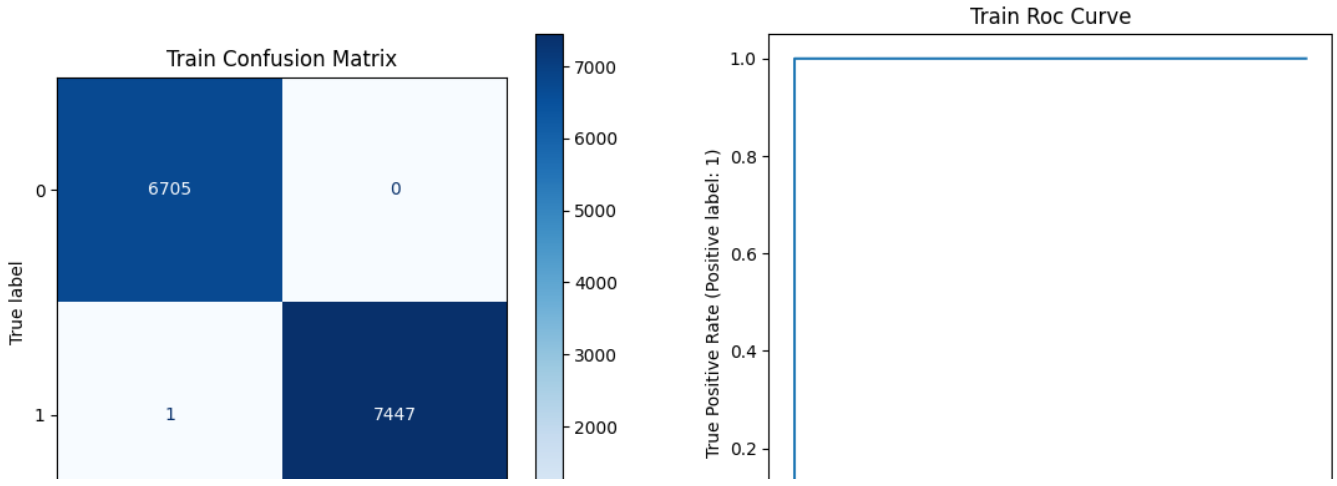




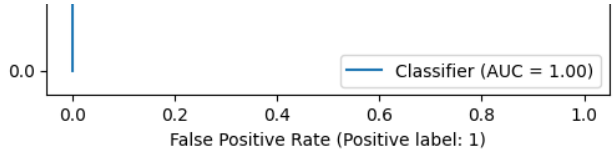
Model 3 - KNN

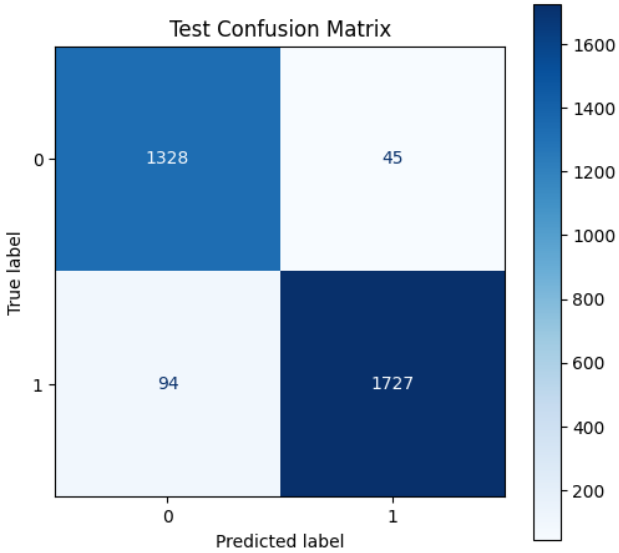
```
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train_final, y_train_final)

results = pre_result_calculator(best_models[3], knn)
train, test = results[0], results[1]
train_score, test_score = results[2], results[3]

print(train_score)
print(test_score)
plot(train, test)
```

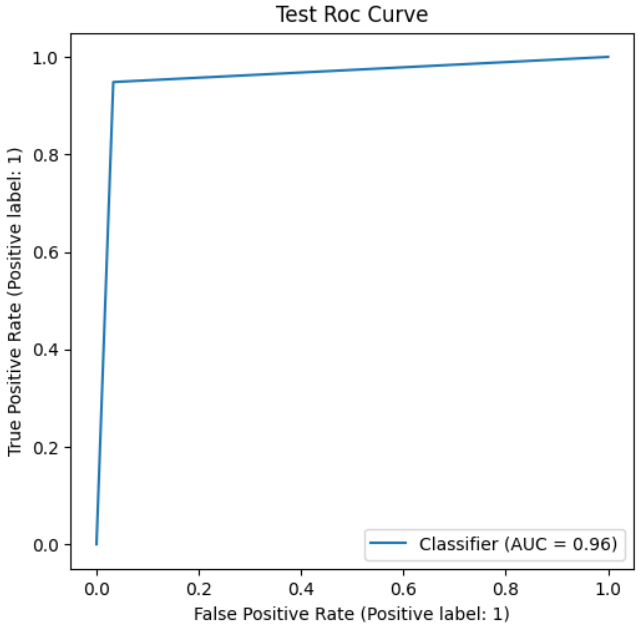
```
  Model Name  Accuracy score  Precision score  Recall score  F1 score
0        KNN        0.999929         0.999929      0.999929  0.999929
  Model Name  Accuracy score  Precision score  Recall score  F1 score
0        KNN        0.956481         0.957105      0.956481  0.956566
```




0
1
Predicted label
1000
0


0.0
Classifier (AUC = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)


Test Confusion Matrix
0
1
True label
1328
45
94
1727
0
1
Predicted label
1600
1400
1200
1000
800
600
400
200

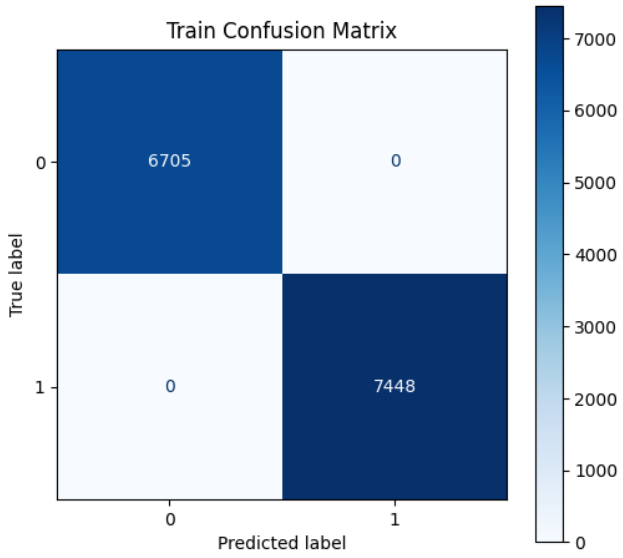

Test Roc Curve
True Positive Rate (Positive label: 1)
1.0
0.8
0.6
0.4
0.2
0.0
Classifier (AUC = 0.96)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)

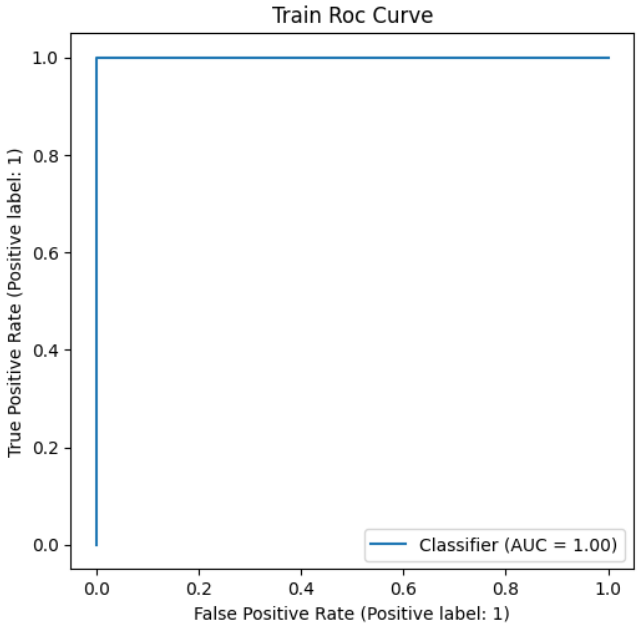
Model 4 - RandomForest

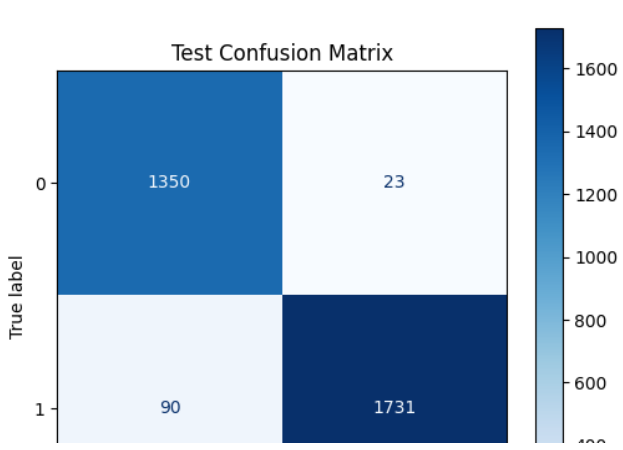
```
rf = RandomForestClassifier(random_state=8)
rf.fit(X_train_final, y_train_final)

results = pre_result_calculator(best_models[4], rf)
train, test = results[0], results[1]
train_score, test_score = results[2], results[3]

print(train_score)
print(test_score)
plot(train, test)
```

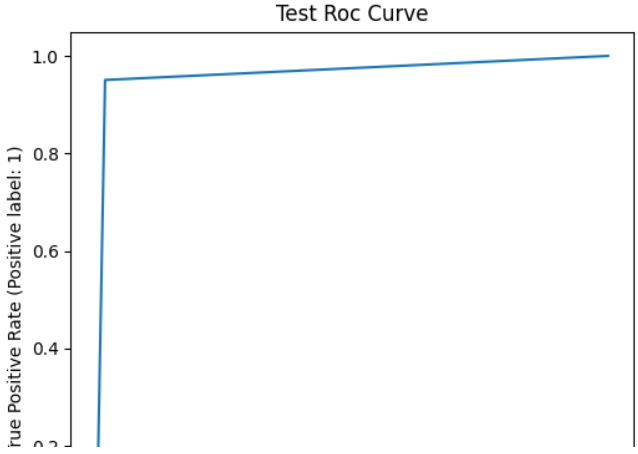
	Model Name	Accuracy score	Precision score	Recall score	F1 score
0	RandomForest	1.0	1.0	1.0	1.0

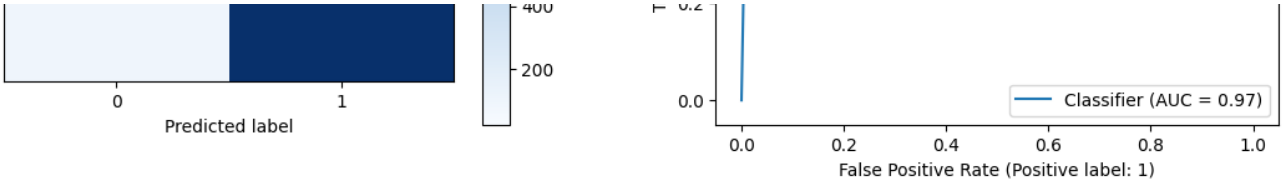
	Model Name	Accuracy score	Precision score	Recall score	F1 score
0	RandomForest	0.964621	0.965657	0.964621	0.964711









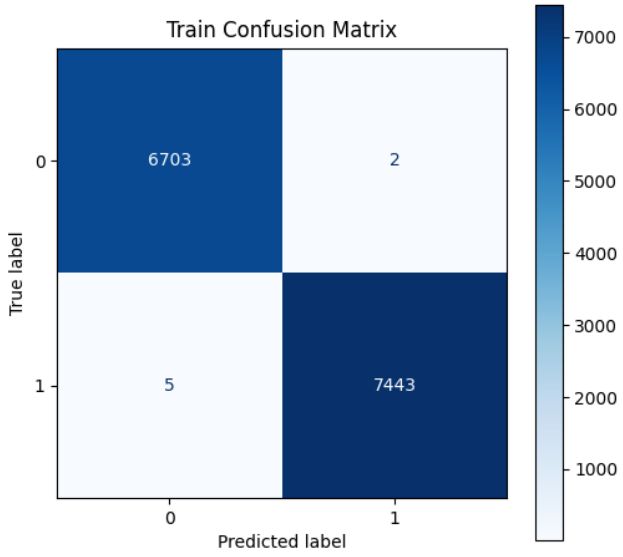


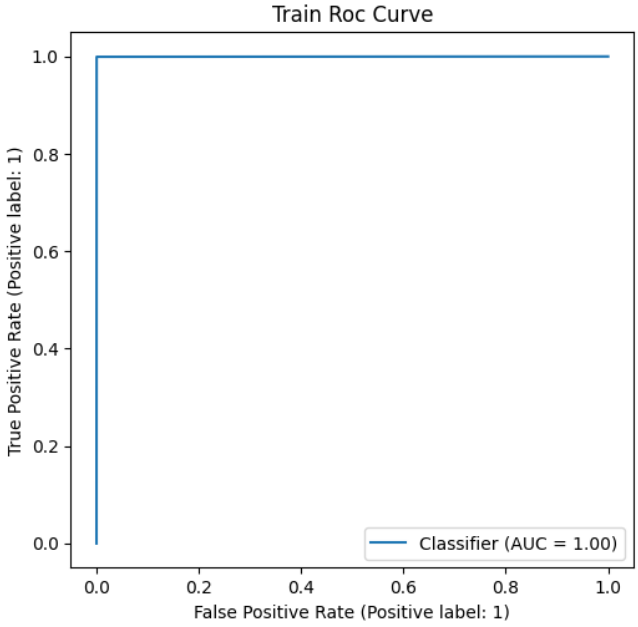
Model 5 - Bagging

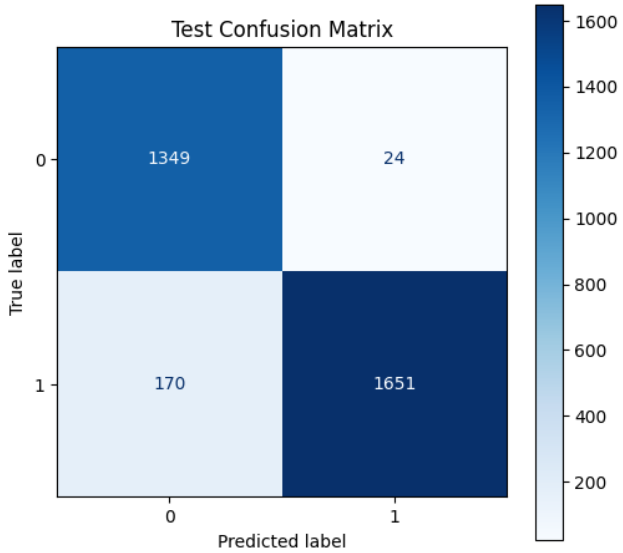
```
bg = BaggingClassifier(random_state=8)
bg.fit(X_train_final, y_train_final)

results = pre_result_calculator(best_models[5], bg)
train, test = results[0], results[1]
train_score, test_score = results[2], results[3]

print(train_score)
print(test_score)
plot(train, test)
```

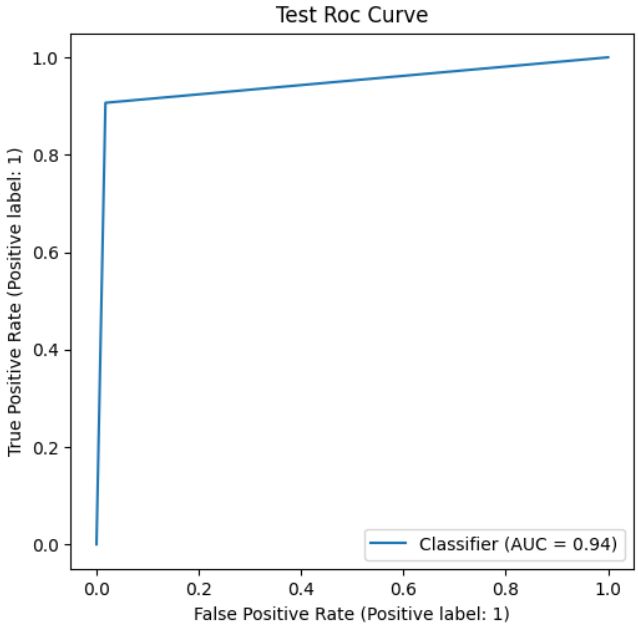
```
  Model Name  Accuracy score  Precision score  Recall score  F1 score
0    Bagging        0.999505         0.999506      0.999505  0.999505
  Model Name  Accuracy score  Precision score  Recall score  F1 score
```

```
0     Bagging        0.939261        0.943722       0.939261   0.939526
```





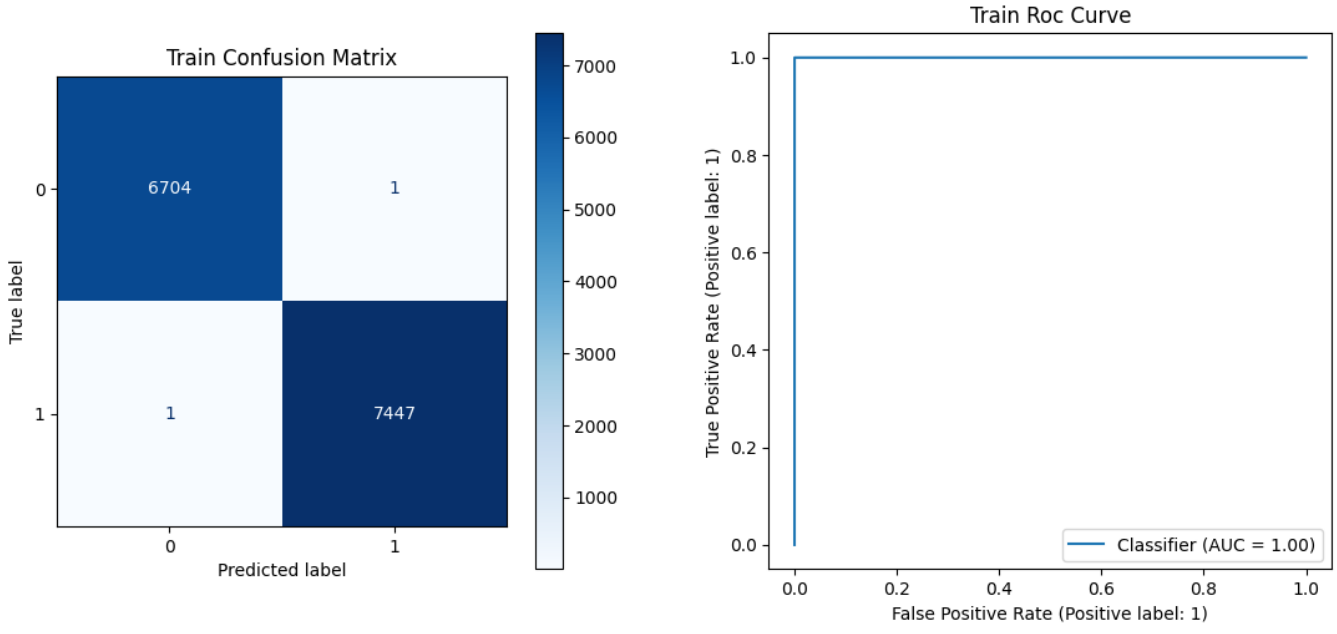


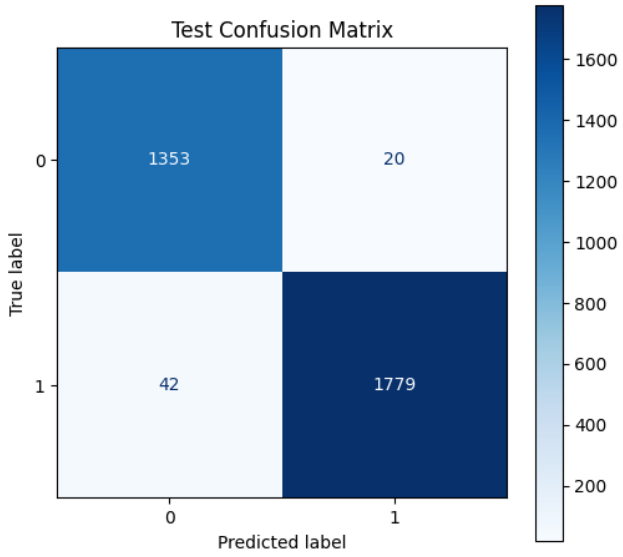


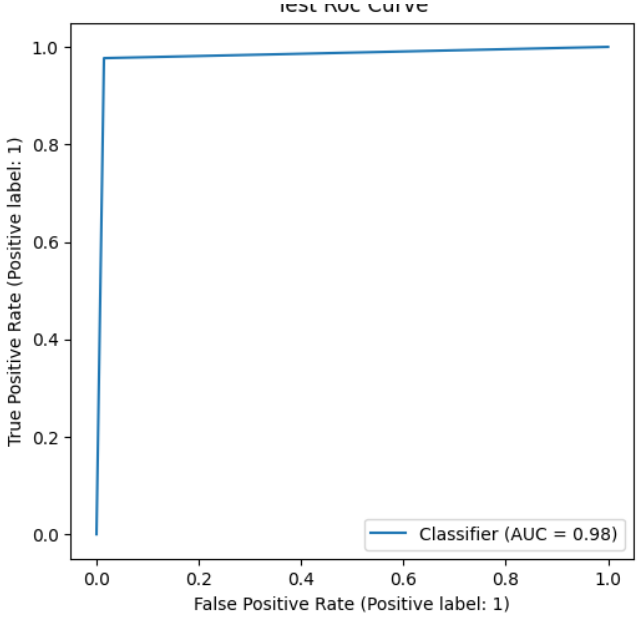
Model 6 - XGB

```
xgb = XGBClassifier(random_state=8)
xgb.fit(X_train_final, y_train_final)

results = pre_result_calculator(best_models[6], xgb)
train, test = results[0], results[1]
train_score, test_score = results[2], results[3]

print(train_score)
print(test_score)
plot(train, test)
```

```
  Model Name  Accuracy score  Precision score  Recall score  F1 score
0        XGB        0.999859         0.999859      0.999859  0.999859
  Model Name  Accuracy score  Precision score  Recall score  F1 score
0        XGB        0.980589         0.980719      0.980589  0.980607
```





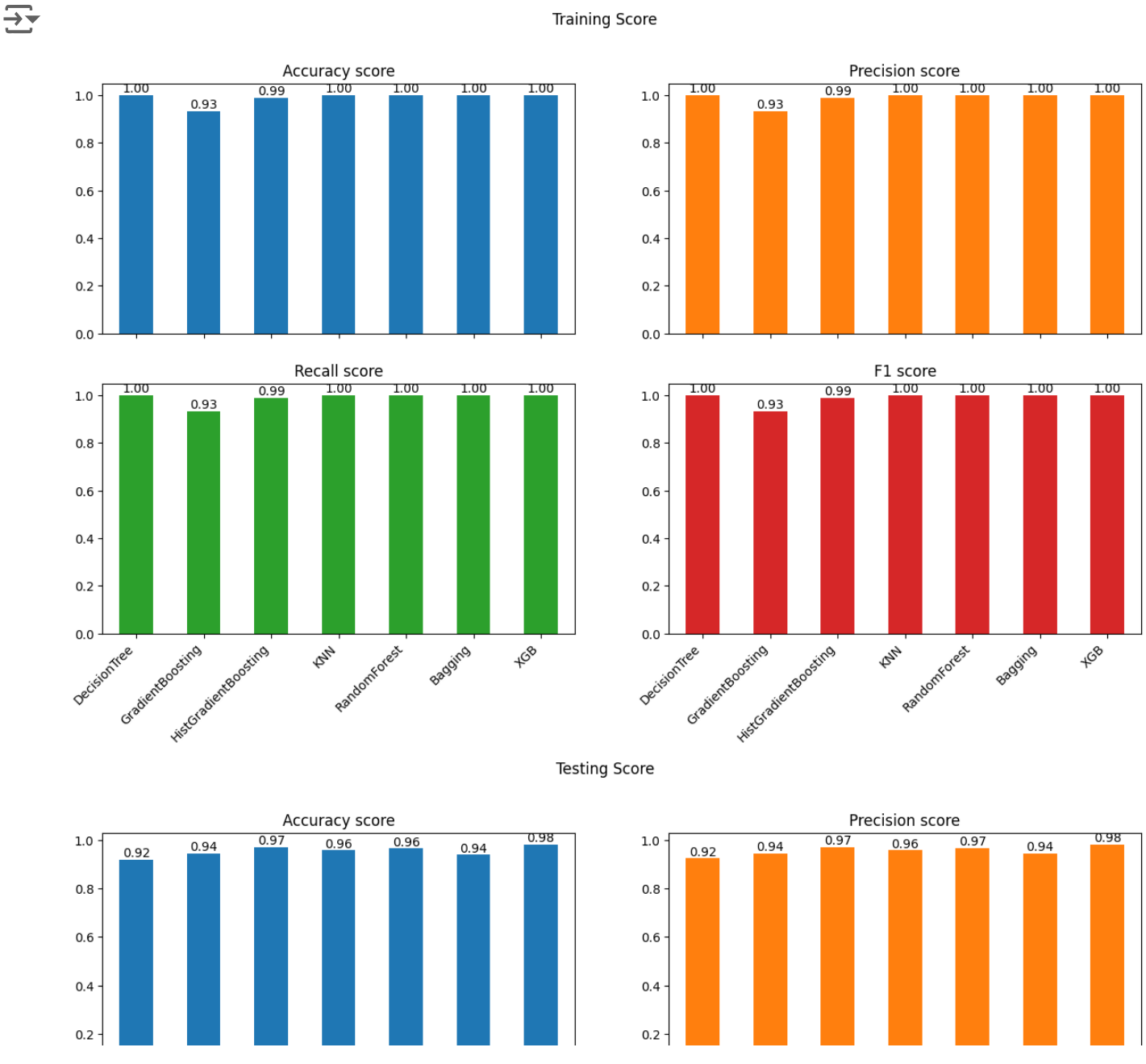


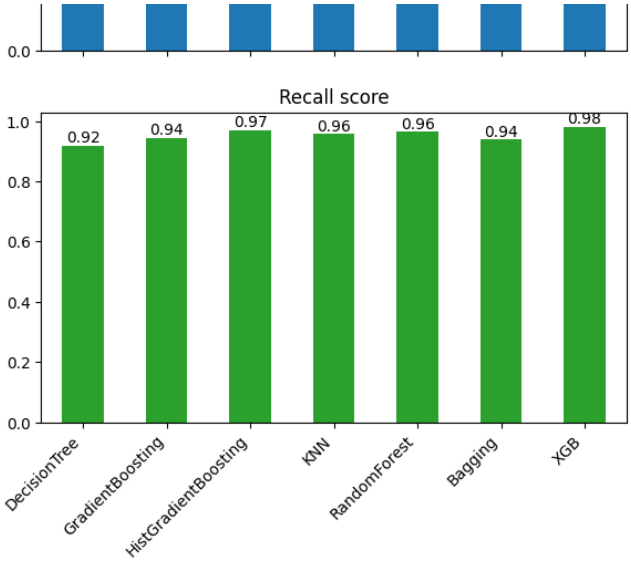
Results

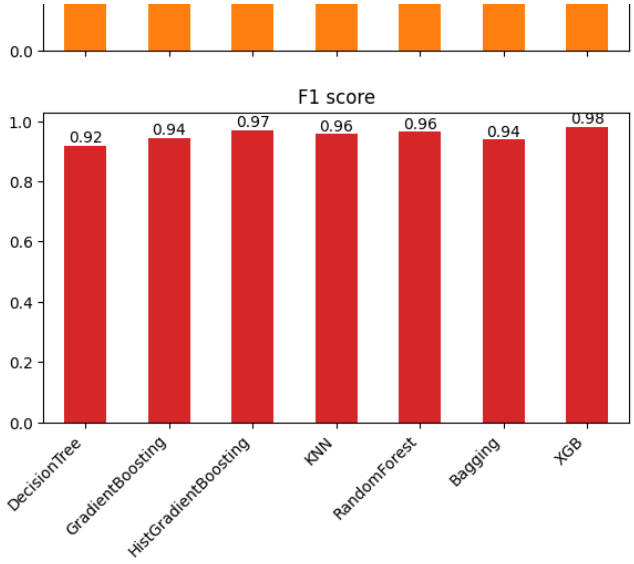
Graph

```
1 # @title Graph
```

```
ax = pd.DataFrame(train_results).plot(kind='bar', subplots=True, la
ax = ax.flatten()
for axes in ax:
    axes.get_legend().remove()
    axes.bar_label(axes.containers[0], fmt='%.2f')
    axes.set_xticklabels(best_models, rotation=45, ha='right')

ax = pd.DataFrame(test_results).plot(kind='bar', subplots=True, lay
ax = ax.flatten()
for axes in ax:
    axes.get_legend().remove()
    axes.bar_label(axes.containers[0], fmt='%.2f')
    axes.set_xticklabels(best_models, rotation=45, ha='right');
```




Recall score
1.0
0.8
0.6
0.4
0.2
0.0
0.92
0.94
0.97
0.96
0.96
0.94
0.98
DecisionTree
GradientBoosting
HistGradientBoosting
KNN
RandomForest
Bagging
XGB


F1 score
1.0
0.8
0.6
0.4
0.2
0.0
0.92
0.94
0.97
0.96
0.96
0.94
0.98
DecisionTree
GradientBoosting
HistGradientBoosting
KNN
RandomForest
Bagging
XGB

Tuning the model

```
train_tuned_results, test_tuned_results = [], []

def pre_tuned_result_calculator(name, model):
    train_pred = model.predict(X_train_final)
    test_pred = model.predict(X_test_final)

    train_score = calculate_results(name, y_train_final, train_pre
    test_score = calculate_results(name, y_test_final, test_pred)

    train_tuned_results.append(train_score)
    test_tuned_results.append(test_score)

    return (y_train_final, train_pred), \
     (y_test_final, test_pred), \
     pd.DataFrame(train_score, index=[0]), \
     pd.DataFrame(test_score, index=[0])
```

GradientBoosting Tuning

```
gb_tuned = GradientBoostingClassifier(random_state=8)

weights = np.linspace(0.0, 0.50, 100)

params = {
    # 'loss': ['log_loss', 'exponential'],
    # 'criterion': ['friedman_mse', 'squared_error'],
    # 'learning_rate': [0.1, 0.01, 0.001],
    'max_depth': [3, 4, 5],
    'max_features': [None, 'auto', 'sqrt'],
    'n_estimators': [100, 120, 140, 160],
}

grid = GridSearchCV(gb_tuned, params, scoring='f1', cv=StratifiedK
grid.fit(X_train_final, y_train_final)
gb_tuned = grid.best_estimator_
gb_tuned
```

```
GradientBoostingClassifier(max_depth=5, max_features='sqrt', n_estimators=1
                           random_state=8)
```

```
GradientBoostingClassifier().get_params()
```

```
{'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'log_loss',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': None,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

```
print(GradientBoostingClassifier.__doc__)
```

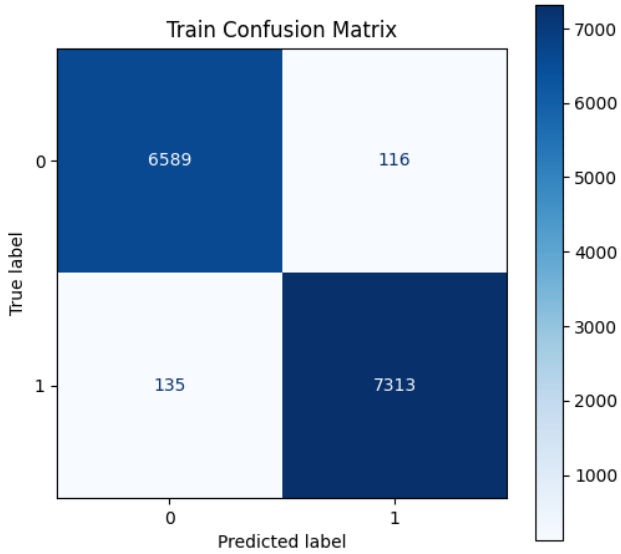
```
print(GradientBoostingClassifier.__doc__)
```

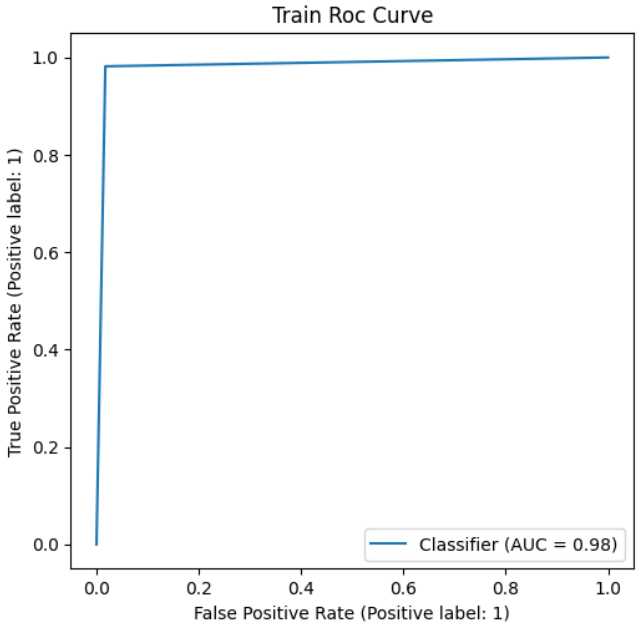
```
Gradient Boosting for classification.

    This algorithm builds an additive model in a forward stage-wise fashion
    allows for the optimization of arbitrary differentiable loss functions.
    each stage ``n_classes_`` regression trees are fit on the negative grad
    of the loss function, e.g. binary or multiclass log loss. Binary
    classification is a special case where only a single regression tree is
    induced.

    :class:`sklearn.ensemble.HistGradientBoostingClassifier` is a much fast
    variant of this algorithm for intermediate datasets (`n_samples >= 10_0

    Read more in the :ref:`User Guide <gradient_boosting>`.

    Parameters
    ----------
    loss : {'log_loss', 'deviance', 'exponential'}, default='log_loss'
        The loss function to be optimized. 'log_loss' refers to binomial an
        multinomial deviance, the same as used in logistic regression.
        It is a good choice for classification with probabilistic outputs.
        For loss 'exponential', gradient boosting recovers the AdaBoost alg

        .. deprecated:: 1.1
            The loss 'deviance' was deprecated in v1.1 and will be removed
            version 1.3. Use `loss='log_loss'` which is equivalent.

    learning_rate : float, default=0.1
        Learning rate shrinks the contribution of each tree by `learning_ra
        There is a trade-off between learning_rate and n_estimators.
        Values must be in the range `[0.0, inf)`.

    n_estimators : int, default=100
        The number of boosting stages to perform. Gradient boosting
        is fairly robust to over-fitting so a large number usually
        results in better performance.
        Values must be in the range `[1, inf)`.

    subsample : float, default=1.0
        The fraction of samples to be used for fitting the individual base
        learners. If smaller than 1.0 this results in Stochastic Gradient
        Boosting. `subsample` interacts with the parameter `n_estimators`.
        Choosing `subsample < 1.0` leads to a reduction of variance
        and an increase in bias.
        Values must be in the range `(0.0, 1.0]`.

    criterion : {'friedman_mse', 'squared_error'}, default='friedman_mse'
        The function to measure the quality of a split. Supported criteria
        'friedman_mse' for the mean squared error with improvement score by
        Friedman, 'squared_error' for mean squared error. The default value
        'friedman_mse' is generally the best as it can provide a better
        approximation in some cases.
```

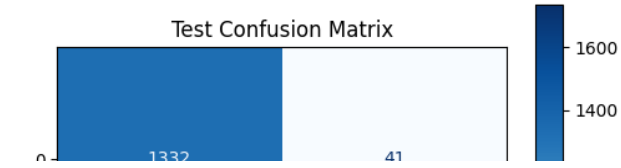
```
        .. versionadded:: 0.18

    min_samples_split : int or float, default=2
        The minimum number of samples required to split an internal node:

        - If int, values must be in the range `[2, inf)`.
        - If float, values must be in the range `(0.0, 1.0]` and `min_sampl
```

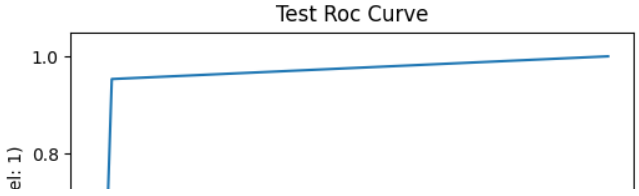
```
gb_tuned.fit(X_train_final, y_train_final)
results = pre_tuned_result_calculator(best_models[1]+'Tuned', gb_t
train, val = results[0], results[1]
train_score, val_score = results[2], results[3]

print(train_score)
print(val_score)
plot(train, val)
```

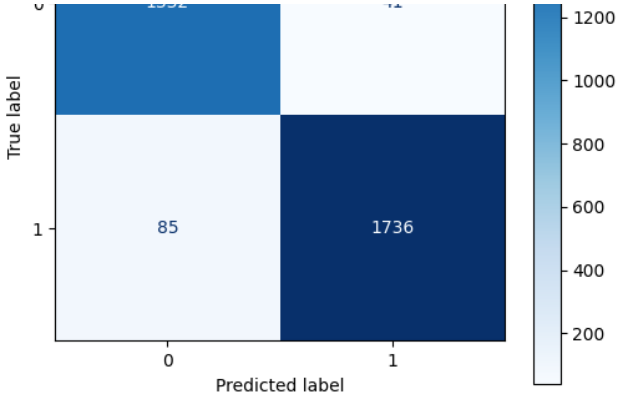
```
                Model Name  Accuracy score  Precision score  Recall score  \
0  GradientBoostingTuned        0.982265         0.982271      0.982265

   F1 score
0  0.982266
                Model Name  Accuracy score  Precision score  Recall score  \
0  GradientBoostingTuned        0.960551          0.96106      0.960551

   F1 score
0  0.960621
```

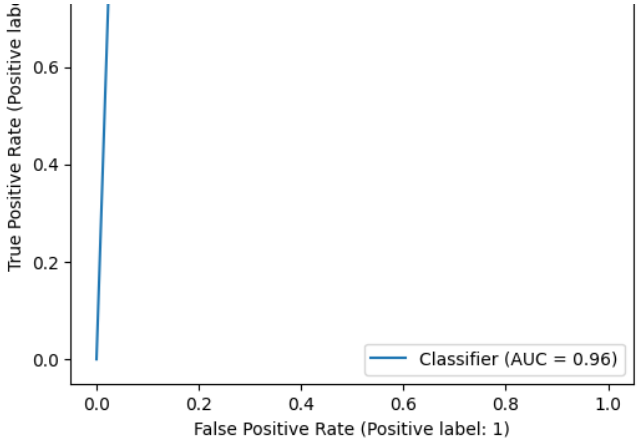












Final Results

```
pd.DataFrame(test_results).sort_values('Recall score', ascending=F
```

	Model Name	Accuracy score	Precision score	Recall score	F1 score
6	XGB	0.980589	0.980719	0.980589	0.980607
2	HistGradientBoosting	0.970883	0.971090	0.970883	0.970916
4	RandomForest	0.964621	0.965657	0.964621	0.964711
3	KNN	0.956481	0.957105	0.956481	0.956566
1	GradientBoosting	0.943331	0.944140	0.943331	0.943453
5	Bagging	0.939261	0.943722	0.939261	0.939526
0	DecisionTree	0.918910	0.924086	0.918910	0.919277

Data Insights (Best Model)

> It's found the best model so far is `XGB`. We got `100%` results on training dataset on all the metrics (Accuracy, Precision, Recall, F1-score) and around `98%` on testing dataset on all the metrics.

Saving the Best Model

```
from joblib import load, dump
```

```
path = '/content/drive/MyDrive/Colab Notebooks/Jain/best_model.job

best_model = dump(xgb, path)
```

Loading the Best Model

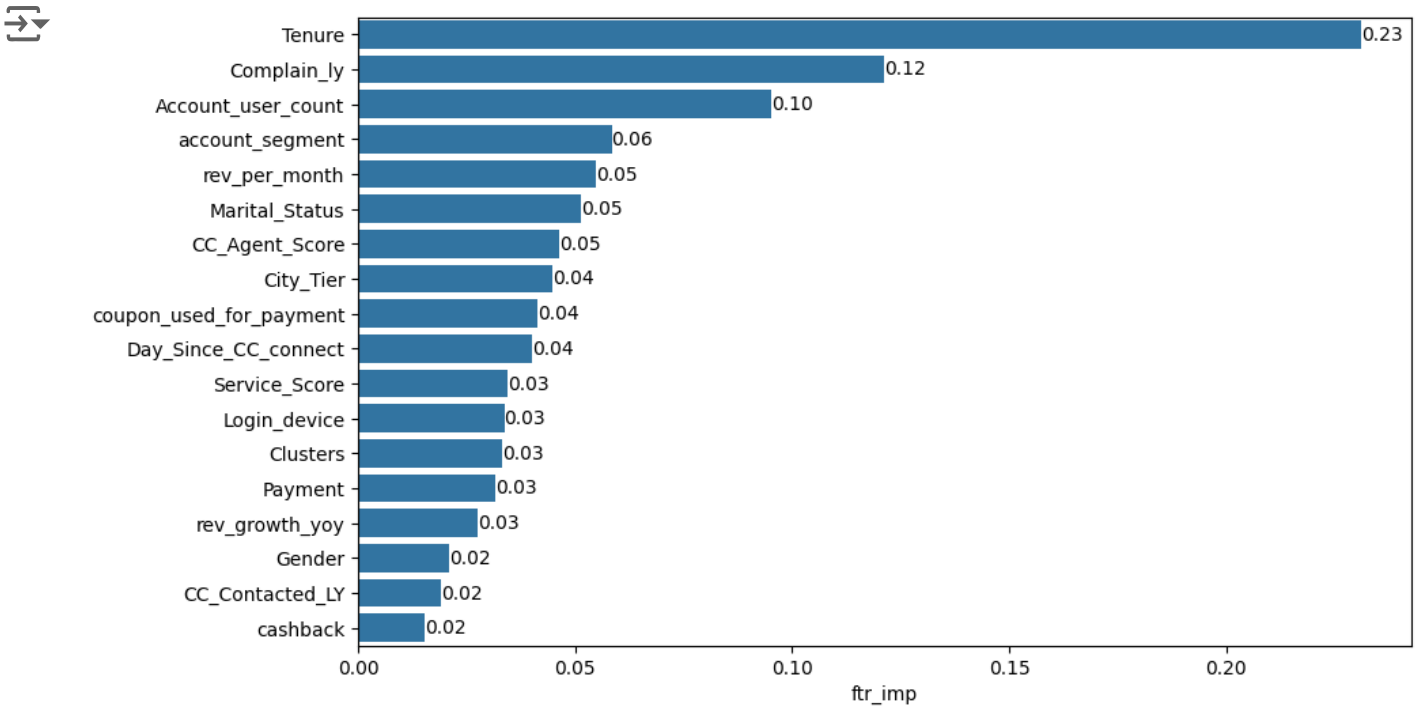
```
saved_model = load(path)
```

```
1 saved_model.score(X_test_final, y_test_final)
```

```
0.9805886036318097
```

Feature Importance

```
ftr_data = pd.DataFrame({'ftr_names': saved_model.feature_names_in
                         'ftr_imp': saved_model.feature_importances

ftr_data = ftr_data.sort_values('ftr_imp', ascending=False).reset_

fig = plt.figure(figsize=(10, 6))
ax = sns.barplot(ftr_data.ftr_imp, orient='h')
ax.set_yticklabels(ftr_data.ftr_names)
ax.bar_label(ax.containers[0], fmt='%.2f');
```



Data Insights

> It is found that the top most important features of the dataset which is contributing most to the model are `Tenure`, `Complain_ly` and `Account_user_count`.
