E-Commerce Customer Churn

Table Of Contents

Introduction

- This is a dataset of leading ecommerce company and we have analysis who are churn(leaving the company service) and have to make predicting churn model.

Features

- CustomerID: Unique customer ID
- Churn: Churn Flag
- Tenure: Tenure of customer in organization
- PreferredLoginDevice: Preferred login device of customer
- CityTier: City tier
- WarehouseToHome: Distance in between warehouse to home of customer
- PreferredPaymentMode: Preferred payment method of customer
- Gender: Gender of customer
- HourSpendOnApp: Number of hours spend on mobile application or website
- NumberOfDeviceRegistered: Total number of deceives is registered on particular customer
- PreferedOrderCat: Preferred order category of customer in last month
- SatisfactionScore: Satisfactory score of customer on service
- MaritalStatus: Marital status of customer
- NumberOfAddress: Total number of added added on particular customer
- OrderAmountHikeFromlastYear: Percentage increases in order from last year
- CouponUsed: Total number of coupon has been used in last month
- OrderCount: Total number of orders has been places in last month
- DaySinceLastOrder: Day Since last order by customer
- CashbackAmount: Average cashback in last month

Goal

- Build a predictive model that can accurately identify customers who are at risk of leaving the company (churn) based on the provided variables. This can help the company take proactive steps to retain these customers and reduce the rate of churn.

- Perform a thorough exploratory analysis of the provided customer data to gain insights into the behavior and characteristics of the customers. This includes analyzing patterns and trends in variables. This analysis can help the

company understand its customers better and inform future decision-making.

Plan

Datasets Overview

- Review the provided customer data to familiarize yourself with the variables and their structure.
- Check the data quality, missing values, and potential errors.
- Determine if any data pre-processing is necessary.

Exploratory Data Analysis

- Analyze the distribution of the variables to identify any outliers or anomalies.
- Investigate the relationship between variables to identify any correlations or patterns.
- Visualize the data to gain insights into the behavior and characteristics of the customers.

Pre-Processing

- Clean the data by handling missing values, converting variables to appropriate data types, and addressing any data quality issues.
- Select the most important variables for building the predictive model.

Machine Learning

- Build a predictive model that can identify customers who are at risk of leaving the company.

Importing Important Libraries

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler , LabelEncoder
from sklearn.svm import SVC

# Additional imports
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score , confusion_matri
from sklearn.model_selection import GridSearchCV, cross_validate

import warnings
warnings.simplefilter(action='ignore')
```

Data Overview

```
# Step 1: Data Loading and Understanding

df = pd.read_excel('E Commerce Dataset.xlsx', sheet_name='E Comm')
df.head()
```

redOrderCat	SatisfactionScore	MaritalStatus	NumberOfAddress	Complain	OrderAmountHikeFromlastYear	CouponUsed
Laptop & Accessory	2	Single	9	1	11.0	1.0
Mobile	3	Single	7	1	15.0	0.0
Mobile	3	Single	6	1	14.0	0.0
Laptop & Accessory	5	Single	8	0	23.0	0.0
Mobile	5	Single	3	0	11.0	1.0

```
df.shape
```

```
(5630, 20)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5630 entries, 0 to 5629
Data columns (total 20 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   CustomerID                   5630 non-null   int64  
 1   Churn                        5630 non-null   int64  
 2   Tenure                       5366 non-null   float64
 3   PreferredLoginDevice         5630 non-null   object 
 4   CityTier                     5630 non-null   int64  
 5   WarehouseToHome              5379 non-null   float64
 6   PreferredPaymentMode         5630 non-null   object 
 7   Gender                       5630 non-null   object 
 8   HourSpendOnApp               5375 non-null   float64
 9   NumberOfDeviceRegistered     5630 non-null   int64  
 10  PreferedOrderCat             5630 non-null   object 
 11  SatisfactionScore            5630 non-null   int64  
 12  MaritalStatus                5630 non-null   object 
 13  NumberOfAddress              5630 non-null   int64  
 14  Complain                     5630 non-null   int64  
 15  OrderAmountHikeFromlastYear  5365 non-null   float64
 16  CouponUsed                   5374 non-null   float64
 17  OrderCount                   5372 non-null   float64
 18  DaySinceLastOrder            5323 non-null   float64
 19  CashbackAmount               5630 non-null   float64
dtypes: float64(8), int64(7), object(5)
memory usage: 879.8+ KB
```

```
df.nunique()
```

```
CustomerID                    5630
Churn                            2
Tenure                          36
PreferredLoginDevice             3
CityTier                         3
WarehouseToHome                 34
PreferredPaymentMode             7
Gender                           2
HourSpendOnApp                   6
NumberOfDeviceRegistered         6
PreferedOrderCat                 6
SatisfactionScore                5
MaritalStatus                    3
NumberOfAddress                 15
Complain                         2
OrderAmountHikeFromlastYear     16
CouponUsed                      17
OrderCount                      16
DaySinceLastOrder               22
CashbackAmount                2586
dtype: int64
```

```
# colums to list
columns = df.columns.to_list()
columns
```

```
['CustomerID',
 'Churn',
 'Tenure',
 'PreferredLoginDevice',
 'CityTier',
 'WarehouseToHome',
 'PreferredPaymentMode',
 'Gender',
 'HourSpendOnApp',
 'NumberOfDeviceRegistered',
 'PreferedOrderCat',
 'SatisfactionScore',
 'MaritalStatus',
 'NumberOfAddress',
 'Complain',
 'OrderAmountHikeFromlastYear',
 'CouponUsed',
 'OrderCount',
 'DaySinceLastOrder',
 'CashbackAmount']
```

```
df.select_dtypes(exclude=np.number).columns
```

```
Index(['PreferredLoginDevice', 'PreferredPaymentMode', 'Gender',
       'PreferedOrderCat', 'MaritalStatus'],
      dtype='object')
```

```
df.describe(include='O').style.background_gradient(axis=None , cmap = "Blues" , vmin = 0 , vmax = 9000  )
```

	PreferredLoginDevice	PreferredPaymentMode	Gender	PreferedOrderCat	MaritalStatus
count	5630	5630	5630	5630	5630
unique	3	7	2	6	3
top	Mobile Phone	Debit Card	Male	Laptop & Accessory	Married
freq	2765	2314	3384	2050	2986

```
# Show the unique values on each column.
for col in df.columns:
    if df[col].dtype == object:
        print(str(col) + ' : ' + str(df[col].unique()))
        print(df[col].value_counts())
        print("_____________________________________________________________________________")
```

```
PreferredLoginDevice : ['Mobile Phone' 'Phone' 'Computer']
Mobile Phone    2765
Computer        1634
Phone           1231
Name: PreferredLoginDevice, dtype: int64
_____________________________________________________________________________
PreferredPaymentMode : ['Debit Card' 'UPI' 'CC' 'Cash on Delivery' 'E wallet' 'COD' 'Credit Card']
Debit Card          2314
Credit Card         1501
E wallet             614
UPI                  414
COD                  365
CC                   273
Cash on Delivery     149
Name: PreferredPaymentMode, dtype: int64
_____________________________________________________________________________
Gender : ['Female' 'Male']
Male      3384
Female    2246
Name: Gender, dtype: int64
_____________________________________________________________________________
PreferedOrderCat : ['Laptop & Accessory' 'Mobile' 'Mobile Phone' 'Others' 'Fashion' 'Grocery']
Laptop & Accessory    2050
Mobile Phone          1271
Fashion                826
Mobile                 809
Grocery                410
Others                 264
Name: PreferedOrderCat, dtype: int64
_____________________________________________________________________________
MaritalStatus : ['Single' 'Divorced' 'Married']
Married     2986
Single      1796
Divorced     848
Name: MaritalStatus, dtype: int64
_____________________________________________________________________________
```

```
df.select_dtypes(include=np.number).columns
```

```
Index(['CustomerID', 'Churn', 'Tenure', 'CityTier', 'WarehouseToHome',
       'HourSpendOnApp', 'NumberOfDeviceRegistered', 'SatisfactionScore',
       'NumberOfAddress', 'Complain', 'OrderAmountHikeFromlastYear',
       'CouponUsed', 'OrderCount', 'DaySinceLastOrder', 'CashbackAmount'],
      dtype='object')
```

```
df.describe().T.style.bar(subset=['mean']).background_gradient(subset=['std','50%','max'])
```

	count	mean	std	min	25%	50%	
CustomerID	5630.000000	52815.500000	1625.385339	50001.000000	51408.250000	52815.500000	542
Churn	5630.000000	0.168384	0.374240	0.000000	0.000000	0.000000	
Tenure	5366.000000	10.189899	8.557241	0.000000	2.000000	9.000000	
CityTier	5630.000000	1.654707	0.915389	1.000000	1.000000	1.000000	
WarehouseToHome	5379.000000	15.639896	8.531475	5.000000	9.000000	14.000000	
HourSpendOnApp	5375.000000	2.931535	0.721926	0.000000	2.000000	3.000000	
NumberOfDeviceRegistered	5630.000000	3.688988	1.023999	1.000000	3.000000	4.000000	
SatisfactionScore	5630.000000	3.066785	1.380194	1.000000	2.000000	3.000000	
NumberOfAddress	5630.000000	4.214032	2.583586	1.000000	2.000000	3.000000	
Complain	5630.000000	0.284902	0.451408	0.000000	0.000000	0.000000	
OrderAmountHikeFromlastYear	5365.000000	15.707922	3.675485	11.000000	13.000000	15.000000	
CouponUsed	5374.000000	1.751023	1.894621	0.000000	1.000000	1.000000	
OrderCount	5372.000000	3.008004	2.939680	1.000000	1.000000	2.000000	
DaySinceLastOrder	5323.000000	4.543491	3.654433	0.000000	2.000000	3.000000	
CashbackAmount	5630.000000	177.223030	49.207036	0.000000	145.770000	163.280000	1

```
for col in df.columns:
    if df[col].dtype == float or df[col].dtype == int:
        print(str(col) + ' : ' + str(df[col].unique()))
        print(df[col].value_counts())
        print("_____________________________________________________________________________")
```

```
127.74    1
145.05    1
174.28    1
173.78    1
Name: CashbackAmount, Length: 2586, dtype: int64
```

```
#As mobile phone and phone are both same so we have merged them
df.loc[df['PreferredLoginDevice'] == 'Phone', 'PreferredLoginDevice' ] = 'Mobile Phone'
df.loc[df['PreferedOrderCat'] == 'Mobile', 'PreferedOrderCat' ] = 'Mobile Phone'
```

```
df['PreferredLoginDevice'].value_counts()
```

```
Mobile Phone    3996
Computer        1634
Name: PreferredLoginDevice, dtype: int64
```

```
#as cod is also cash on delievery
#as cc is also credit card so i merged them
df.loc[df['PreferredPaymentMode'] == 'COD', 'PreferredPaymentMode' ] = 'Cash on Delivery'   # uses loc function
df.loc[df['PreferredPaymentMode'] == 'CC', 'PreferredPaymentMode' ] = 'Credit Card'
```

```
df['PreferredPaymentMode'].value_counts()
```

```
Debit Card          2314
Credit Card         1774
E wallet             614
Cash on Delivery     514
UPI                  414
Name: PreferredPaymentMode, dtype: int64
```

```
# convert num_cols to categories
df2 = df.copy()
for col in df2.columns:
  if col == 'CustomerID':
    continue

  else:
    if df2[col].dtype == 'int':
      df2[col] = df[col].astype(str)
```

```
df2.dtypes
```

```
CustomerID                     int64
Churn                         object
Tenure                       float64
PreferredLoginDevice          object
CityTier                      object
WarehouseToHome              float64
PreferredPaymentMode          object
Gender                        object
HourSpendOnApp               float64
NumberOfDeviceRegistered      object
PreferedOrderCat              object
SatisfactionScore             object
MaritalStatus                 object
NumberOfAddress               object
Complain                      object
OrderAmountHikeFromlastYear  float64
CouponUsed                   float64
OrderCount                   float64
DaySinceLastOrder            float64
CashbackAmount               float64
dtype: object
```

```
# Categorical cols after Converting
df2.describe(include='O').style.background_gradient(axis=None , cmap = "Blues" , vmin = 0 , vmax = 9000  )
```

	Churn	PreferredLoginDevice	CityTier	PreferredPaymentMode	Gender	NumberOfDeviceRegistered	PreferedO
count	5630	5630	5630	5630	5630	5630	
unique	2	2	3	5	2	6	
top	0	Mobile Phone	1	Debit Card	Male	4	Mobi
freq	4682	3996	3666	2314	3384	2377	

```
# Numerical cols after Converting
df2.describe().T.style.bar(subset=['mean']).background_gradient(subset=['std','50%','max'])
```

	count	mean	std	min	25%	50%	
CustomerID	5630.000000	52815.500000	1625.385339	50001.000000	51408.250000	52815.500000	542
Tenure	5366.000000	10.189899	8.557241	0.000000	2.000000	9.000000	
WarehouseToHome	5379.000000	15.639896	8.531475	5.000000	9.000000	14.000000	
HourSpendOnApp	5375.000000	2.931535	0.721926	0.000000	2.000000	3.000000	
OrderAmountHikeFromlastYear	5365.000000	15.707922	3.675485	11.000000	13.000000	15.000000	
CouponUsed	5374.000000	1.751023	1.894621	0.000000	1.000000	1.000000	
OrderCount	5372.000000	3.008004	2.939680	1.000000	1.000000	2.000000	
DaySinceLastOrder	5323.000000	4.543491	3.654433	0.000000	2.000000	3.000000	
CashbackAmount	5630.000000	177.223030	49.207036	0.000000	145.770000	163.280000	1

```
df.duplicated().sum()
```

```
0
```

```
# the sum of null values
grouped_data = []
for col in columns:
    n_missing = df[col].isnull().sum()
    percentage = n_missing / df.shape[0] * 100
    grouped_data.append([col, n_missing, percentage])

# Create a new DataFrame from the grouped data
grouped_df = pd.DataFrame(grouped_data, columns=['column', 'n_missing', 'percentage'])

# Group by 'col', 'n_missing', and 'percentage'
result = grouped_df.groupby(['column', 'n_missing', 'percentage']).size()
result
```

```
column                      n_missing  percentage
CashbackAmount              0          0.000000      1
Churn                       0          0.000000      1
CityTier                    0          0.000000      1
Complain                    0          0.000000      1
CouponUsed                  256        4.547069      1
CustomerID                  0          0.000000      1
DaySinceLastOrder           307        5.452931      1
Gender                      0          0.000000      1
HourSpendOnApp              255        4.529307      1
MaritalStatus               0          0.000000      1
NumberOfAddress             0          0.000000      1
NumberOfDeviceRegistered    0          0.000000      1
OrderAmountHikeFromlastYear 265        4.706927      1
OrderCount                  258        4.582593      1
PreferedOrderCat            0          0.000000      1
PreferredLoginDevice        0          0.000000      1
PreferredPaymentMode        0          0.000000      1
SatisfactionScore           0          0.000000      1
Tenure                      264        4.689165      1
WarehouseToHome             251        4.458259      1
dtype: int64
```

```
# from pandas_profiling import ProfileReport
# ProfileReport(df)
```

EDA (Exploratory Data Analysis)

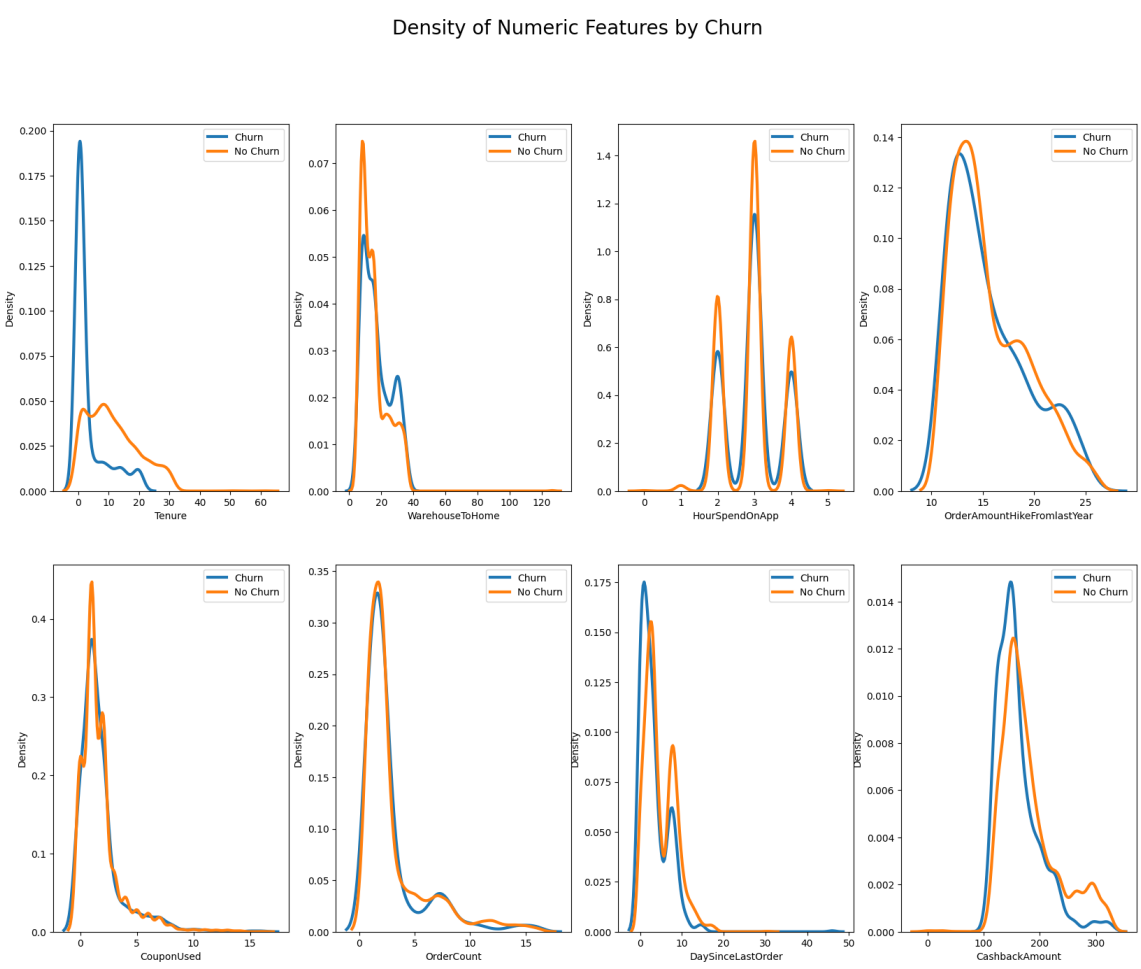
Bussiness Questions

1. Is there a relationship between Gender and Churn? & Which Gender has more Orders?
2. Which MartialStatus has the highest Churn rate?
3. Which CityTier has higher Tenure and OrderCount?
4. Is Customer with High SatisfactionScore have high HourSpendOnApp?
   Is there a correlation between SatisfactionScore and HourSpendOnApp?
5. Which CityTier has the most HourSpendOnApp?
6. What is the relation between NumberOfAddress and CityTier within the churn segment?
7. What is the relation between Complain and DaySinceLastOrder?
8. Is there a relationship between PreferredLoginDevice and Churn?
9. What is the distance between warehouse to customer house in different city tier?

10. Does different CityTiers has different prefered products?
11. What is the preferred payment mode for different CityTiers?
12. Which CityTier has the highest OrderCount?
13. Does the percentage increase in order amount from last year affect churn rate?
14. What is the relation between Complain and DaySinceLastOrder?
15. What is ordercount for customers with high HourSpendOnApp?
16. Is there a relationship between preferred order category and churn rate?
17. Do customers who used more coupons have lower churn rates?
18. Is there a connection between satisfaction score and number of orders in the past month?
19. There is relation between CashbackAmount and order counts within churn?
20. Are customers who complained more likely to churn?

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots
binary_cat_cols = ['Complain']
outcome = ['Churn']
cat_cols = ['PreferredLoginDevice', 'CityTier', 'PreferredPaymentMode',
       'Gender', 'NumberOfDeviceRegistered', 'PreferedOrderCat',
       'SatisfactionScore', 'MaritalStatus', 'NumberOfAddress', 'Complain']
num_cols = ['Tenure', 'WarehouseToHome', 'HourSpendOnApp', 'OrderAmountHikeFromlastYear', 'CouponUsed', 'OrderCount'
```

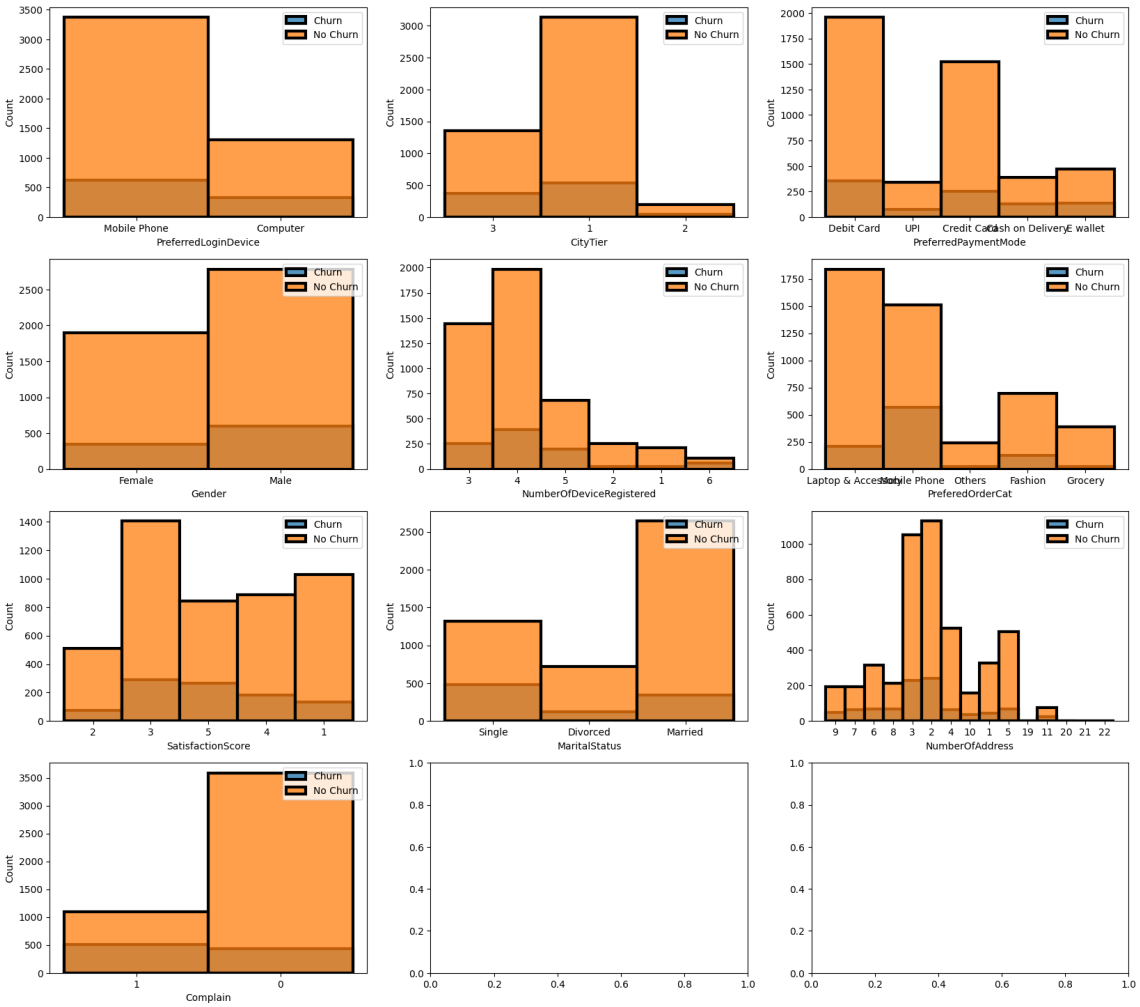
```
df_c = df[df['Churn']==1].copy()
df_nc = df[df['Churn']==0].copy()

fig, ax = plt.subplots(2,4,figsize=(20, 15))
fig.suptitle('Density of Numeric Features by Churn', fontsize=20)
ax = ax.flatten()

for idx,c in enumerate(num_cols):
    sns.kdeplot(df_c[c], linewidth= 3,
             label = 'Churn',ax=ax[idx])
    sns.kdeplot(df_nc[c], linewidth= 3,
             label = 'No Churn',ax=ax[idx])

    ax[idx].legend(loc='upper right')

plt.show()
```



Distributions Insights Of the Numeric Features

- Tenure: Customers with longer tenure seem less likely to churn. Makes sense as longer tenure indicates satisfaction.
- CityTier: Churn rate looks similar across tiers. City tier does not seem predictive of churn.
- WarehouseToHome: Shorter warehouse to home distances have a lower churn rate. Faster deliveries may improve satisfaction.
- HourSpendOnApp: More time spent on app correlates with lower churn. App engagement is a good sign.
- NumberOfDeviceRegistered: More registered devices associates with lower churn. Access across devices improves convenience.
- SatisfactionScore: Higher satisfaction scores strongly associate with lower churn, as expected. Critical driver.
- NumberOfAddress: Slight downward trend in churn as number of addresses increases. More addresses indicates loyalty.
- Complain: More complaints associate with higher churn, though relationship isn't very strong. Complaints hurt satisfaction.
- OrderAmountHikeFromLastYear: Big spenders from last year are less likely to churn. Good to retain big customers.
- CouponUsed: Coupon usage correlates with lower churn. Coupons enhance loyalty.
- OrderCount: Higher order counts associate with lower churn. Frequent usage builds habits.
- DaySinceLastOrder: Longer since last order correlates with higher churn. Recency is a good predictor.

```
df_c = df2[df2['Churn']=='1'].copy()
df_nc = df2[df2['Churn']=='0'].copy()

fig, ax = plt.subplots(4,3,figsize=(20, 18))
fig.suptitle('Density of Numeric Features by Churn', fontsize=20)
ax = ax.flatten()

for idx,c in enumerate(cat_cols):
    sns.histplot(df_c[c], linewidth= 3,
             label = 'Churn',ax=ax[idx])
    sns.histplot(df_nc[c], linewidth= 3,
             label = 'No Churn',ax=ax[idx])

    ax[idx].legend(loc='upper right')

plt.show()
```



```
# color palettes
pie_palette = ['#3E885B','#7694B6','#85BDA6', '#80AEBD', '#2F4B26', '#3A506B']
green_palette = ['#2F4B26', '#3E885B', '#85BDA6', '#BEDCFE', '#C0D7BB']
blue_palette = ['#3A506B', '#7694B6', '#80AEBD', '#5BC0BE', '#3E92CC']
custom_palette = ['#3A506B', '#7694B6', '#80AEBD', '#3E885B', '#85BDA6']
red_palette = ['#410B13', '#CD5D67', '#BA1F33', '#421820', '#91171F']
```

1-Is there a relationship between Gender and Churn? & Which Gender has more Orders?

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

```
Male      3384
Female    2246
Name: Gender, dtype: int64
```

```
df.groupby("Churn")["Gender"].value_counts() # the churned females ratio 348/2246 * 100
                                             # the churned males ratio 600/3384 * 100
```

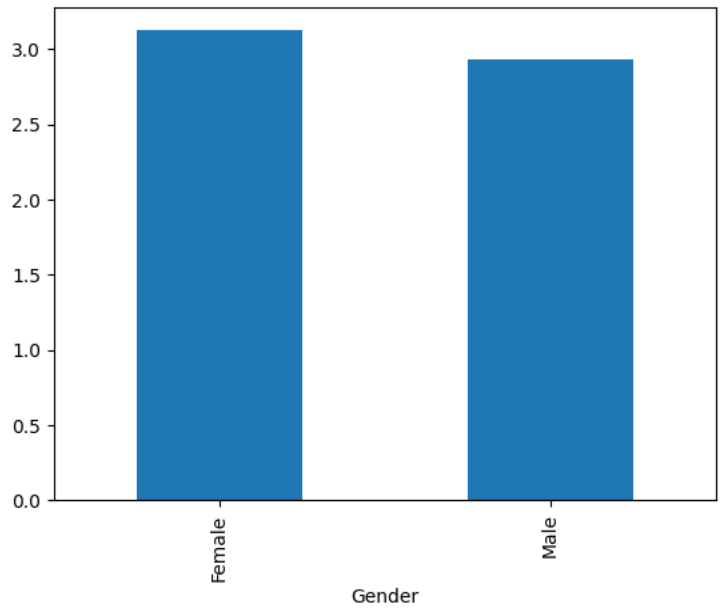
```
Churn  Gender
0      Male      2784
       Female    1898
1      Male       600
       Female     348
Name: Gender, dtype: int64
```

```
df.groupby("PreferredLoginDevice")["OrderCount"].value_counts() # the churned females ratio 348/2246 * 100
```

```
PreferredLoginDevice  OrderCount
Computer              2.0            573
                      1.0            486
                      3.0            132
                      4.0             61
                      7.0             59
                      5.0             48
                      8.0             44
                      6.0             40
                      14.0            20
                      9.0             19
                      11.0            16
                      10.0            15
                      12.0            15
                      13.0             9
                      15.0             8
                      16.0             4
Mobile Phone          2.0           1452
                      1.0           1265
                      3.0            239
                      7.0            147
                      4.0            143
                      5.0            133
                      8.0            128
                      6.0             97
                      9.0             43
                      12.0            39
                      11.0            35
                      15.0            25
                      10.0            21
                      13.0            21
                      16.0            19
                      14.0            16
Name: OrderCount, dtype: int64
```

```
gender_orders = df.groupby('Gender')['OrderCount'].mean().plot(kind='bar')

gender_orders  # females have more order count avg
```

```
<Axes: xlabel='Gender'>
```



there is not a big difference between the males and the femals: avg order

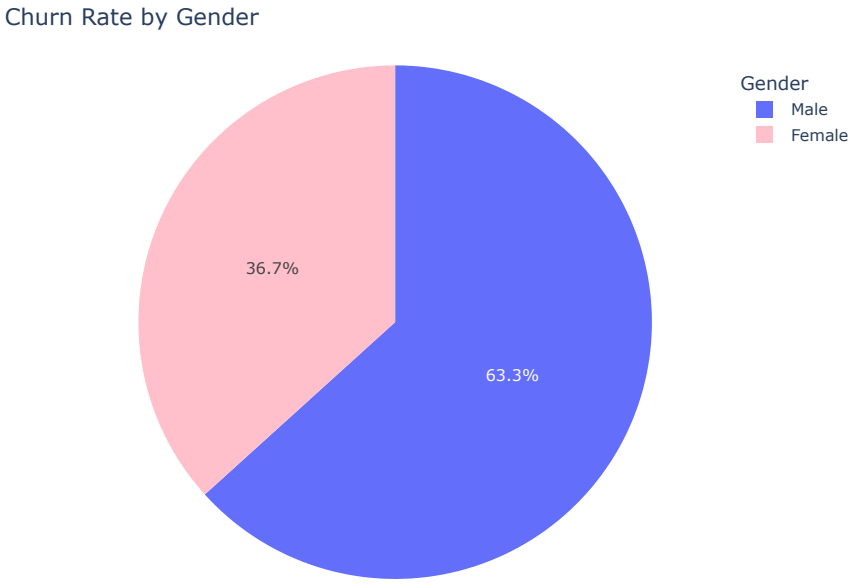
```
percentageM =600/3384 * 100

percentageM   #the percentage of the leaving males out of the males
```

```
17.73049645390071
```

```
percentageF =348/2246 * 100

percentageF  #the percentage of the leaving females out of the females
```

```
15.49421193232413
```

```
import pandas as pd
import plotly.express as px

# Create figure
fig = px.pie(df, values='Churn', names='Gender')
fig.update_traces(marker=dict(colors=['pink ', 'baby blue']))

# Update layout
fig.update_layout(
  title='Churn Rate by Gender',
  legend_title='Gender'
)

# Show plot
fig.show()

# # Create figure
# fig = px.pie(df, values='OrderCount', names='Gender')
# fig.update_traces(marker=dict(colors=['pink ', 'baby blue']))

# # Update layout
# fig.update_layout(
#   title='order Rate by Gender',
#   legend_title='Gender'
# )

# # Show plot
# fig.show()
```



as we see the males are more likely to churn as we have 63.3 % churned males from the app may be the company should consider incresing the products that grap the males interest and so on.. we are going to see if there is another factors that makes the highest segment of churned customers are males.

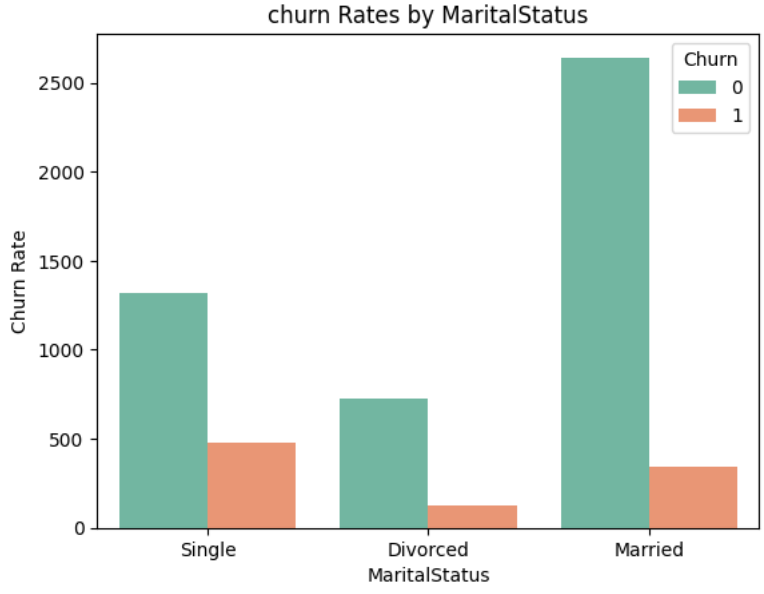
2-Which MartialStatus has the highest Churn rate?

```
df.groupby("Churn")["MaritalStatus"].value_counts()
```

```
Churn  MaritalStatus
0      Married          2642
       Single           1316
       Divorced          724
1      Single            480
       Married           344
       Divorced          124
Name: MaritalStatus, dtype: int64
```

```
sns.countplot(x='MaritalStatus',hue='Churn',data=df,palette='Set2')
plt.title("churn Rates by MaritalStatus")
plt.ylabel("Churn Rate")
```

```
Text(0, 0.5, 'Churn Rate')
```



-the married are the highest customer segment in the comapny may be the comapny should consider taking care of the products that suits the single and the married customers as the singles are the most likely to churn from the app

3-Which CityTier has higher Tenure and OrderCount?

```
df_grouped_tenure = df.groupby('CityTier')['Tenure'].agg(['mean', 'max'])
df_grouped_tenure
```

CityTier	mean	max
1	10.528818	51.0
2	11.169725	31.0
3	9.361740	61.0

```
df_grouped_OrderCount = df.groupby('CityTier')['OrderCount'].agg(['mean', 'max'])
df_grouped_OrderCount
```

CityTier	mean	max
1	2.953255	16.0
2	2.584034	13.0
3	3.185185	16.0

```
# means = df_grouped['Tenure']['mean']
# means.plot(kind='pie',autopct='%1.1f%%')
# plt.xlabel('CityTier')
# plt.ylabel('Mean Tenure')
```

citytier 2 has the highest tenure rate but the tenure rate does not seen to be a strong factor

```
df.groupby("CityTier")["OrderCount"].mean()
```

```
CityTier
1    2.953255
2    2.584034
3    3.185185
Name: OrderCount, dtype: float64
```

citytier 3 has the highest order avg but it not to be a strong factor in the customer churning

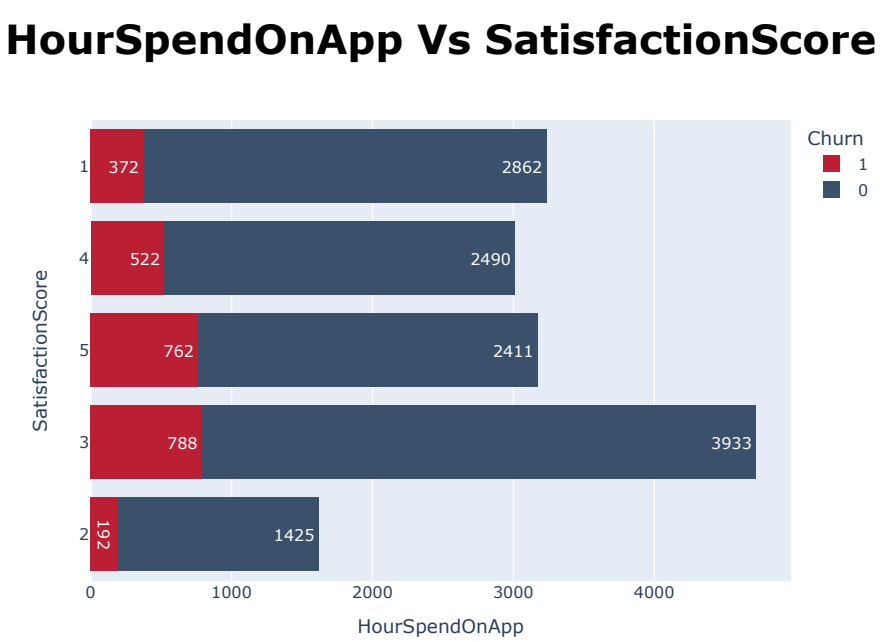
4-Is Customer with High SatisfactionScore have high HourSpendOnApp?

Is there a correlation between SatisfactionScore and HourSpendOnApp?

```
df['SatisfactionScore'].dtypes
```

```
dtype('int64')
```

```
import matplotlib.pyplot as plt

# plot
fig = px.histogram(df2, x="HourSpendOnApp", y="SatisfactionScore", orientation="h", color="Churn" ,text_auto= True ,

# Customize the plot
fig.update_layout(hovermode='x',title_font_size=30)
fig.update_layout(
title_font_color="black",
template="plotly",
title_font_size=30,
hoverlabel_font_size=20,
title_x=0.5,
xaxis_title='HourSpendOnApp',
yaxis_title='SatisfactionScore',
)
fig.show()



# sns.barplot(x='SatisfactionScore',y='HourSpendOnApp',data=df)
# ax = df[['SatisfactionScore','HourSpendOnApp']].value_counts().plot(kind='bar')
```

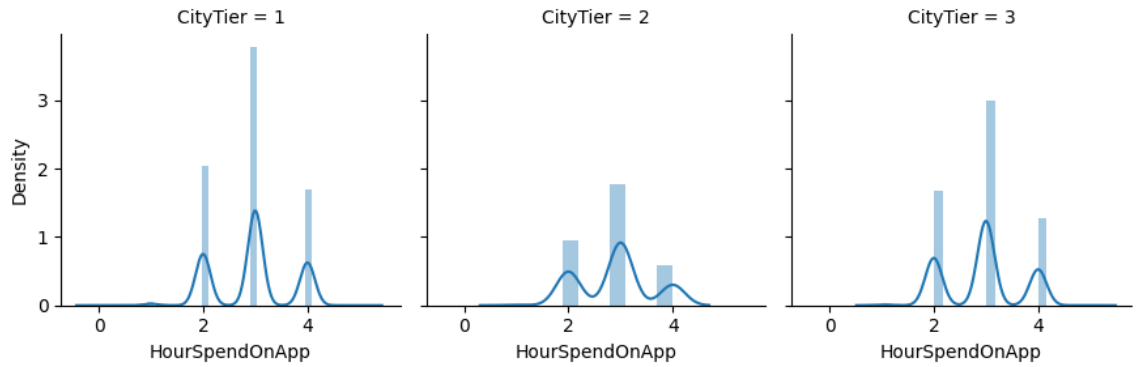


as we see people with less satisfaction score spend less time on the app than the people of satisfaction score 5 but also i do not think there is any realation between the satisfaction score and people's spent time on the app

5-Which CityTier has the most HourSpendOnApp?

```
g = sns.FacetGrid(df, col='CityTier')
g.map(sns.distplot, 'HourSpendOnApp')
```

```
<seaborn.axisgrid.FacetGrid at 0x79c719685c60>
```



city tier 1 has the most spended hours on the app

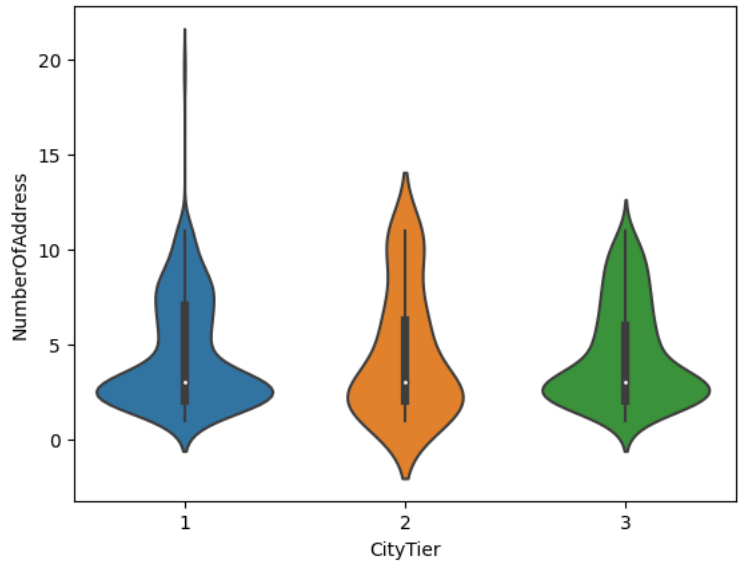
6-What is the relation between NumberOfAddress and CityTier within the churn segment?

```
df.groupby("CityTier")["NumberOfAddress"].value_counts()
```

```
CityTier  NumberOfAddress
1         2                 871
          3                 832
          4                 397
          5                 377
          6                 247
          1                 228
          8                 187
          7                 173
          9                 150
          10                129
          11                 71
          19                  1
          20                  1
          21                  1
          22                  1
2         2                  61
          3                  43
          5                  30
          1                  23
          6                  21
          4                  16
          10                 13
          7                  10
          8                  10
          11                  9
          9                   6
3         2                 437
          3                 403
          4                 175
          5                 164
          1                 120
          6                 114
          8                  83
          9                  83
          7                  73
          10                 52
          11                 18
Name: NumberOfAddress, dtype: int64
```

```
# Violin plots
import seaborn as sns
sns.violinplot(x='CityTier', y='NumberOfAddress', data=df[df['Churn']==1])
```

```
<Axes: xlabel='CityTier', ylabel='NumberOfAddress'>
```



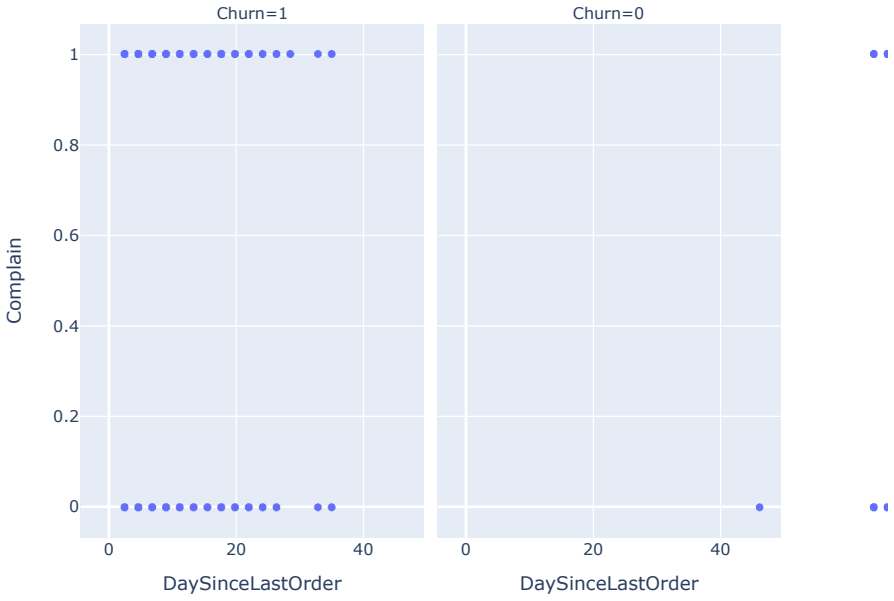
There is a negative correlation between CityTier and NumberOfAddress. Higher CityTiers are associated with lower average NumberOfAddress and a more concentrated distribution. Customers in larger cities (CityTier 1) tend to have more addresses on average compared to smaller cities and towns in lower tiers. The relationship suggests address density and type of locality (metro vs smaller cities vs towns) impacts how many addresses customers have across city types.

7-What is the relation between Complain and DaySinceLastOrder?

```
# Pearson correlation
df[['DaySinceLastOrder', 'Complain']].corr()
```

	DaySinceLastOrder	Complain
DaySinceLastOrder	1.000000	-0.043546
Complain	-0.043546	1.000000

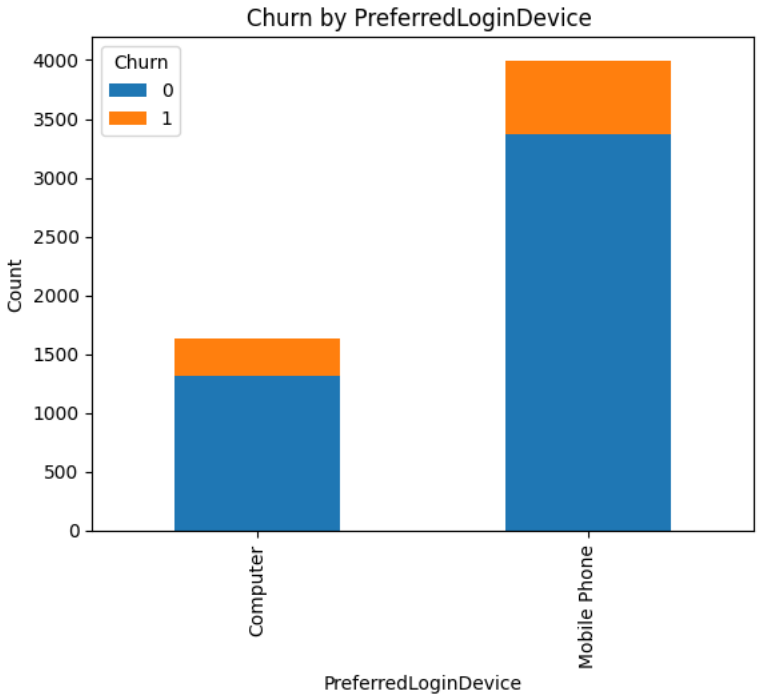
```
import plotly.express as px

fig = px.scatter(df, x='DaySinceLastOrder', y='Complain', facet_col='Churn')
fig.update_layout(hovermode='closest')
fig.show()
```



there is a weak negative relation between complainig and the number of dayes since last order

8-Is there a relationship between PreferredLoginDevice and churn?

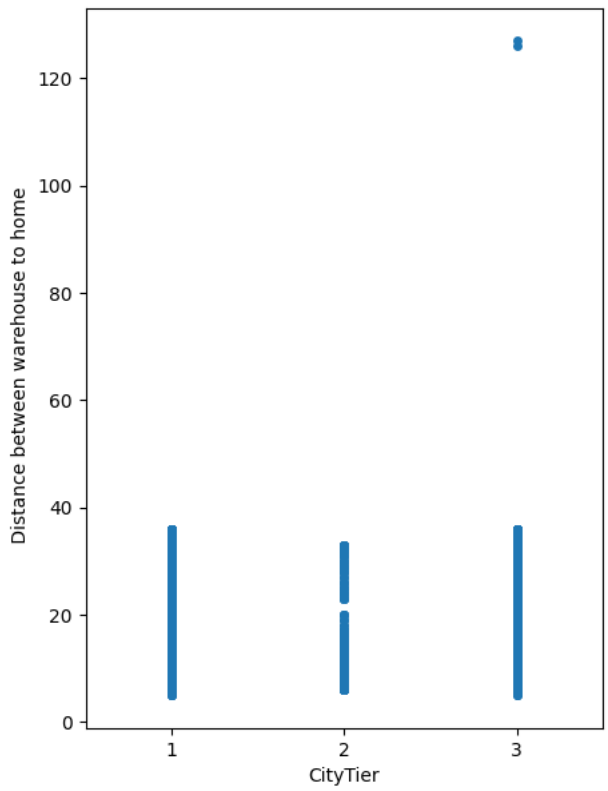
```
# Bar chart with churn rate
import seaborn as sns
# sns.catplot(x='PreferredLoginDevice', y='Churn', data=df, kind='bar')

# Group the data by 'OverTime' and 'Attrition', and calculate the count
grouped_data = df.groupby(['PreferredLoginDevice', 'Churn']).size().unstack().plot(kind='bar', stacked=True)

# Set the plot title, x-label, and y-label
plt.title('Churn by PreferredLoginDevice ')
plt.xlabel('PreferredLoginDevice')
plt.ylabel('Count')

# Show the plot
plt.show()
```



mobile phone users are likely to churn may be this indicates a problem on the app user experience on the app mobile version

9-What is distancebetween warehosue to customer house in different city tier ?

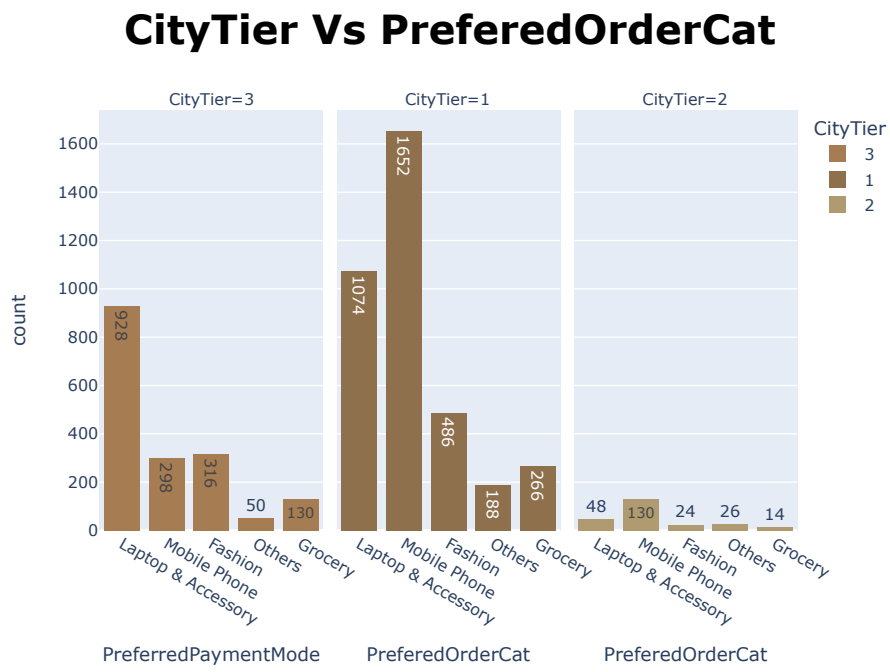
```
df3 = df.copy()

df3['CityTier'].astype('str')
plt.figure(figsize = (5,7))
sns.stripplot(x = 'CityTier', y = 'WarehouseToHome', data = df3, jitter = False)
plt.ylabel(' Distance between warehouse to home');
```



Inference: As the distance from warehouse to home is similar in all city tier which means company had build warehouse in lower city tier also.

10-Does different citytiers has different prefered products?

```
import plotly.express as px
earth_palette = ["#A67C52", "#8F704D", "#B09B71", "#7E786E"]


fig=px.histogram(df,x="PreferedOrderCat",facet_col="CityTier",color="CityTier",color_discrete_sequence=earth_palette

# Customize the plot
fig.update_layout(hovermode='x',title_font_size=30)
fig.update_layout(
title_font_color="black",
template="plotly",
title_font_size=30,
hoverlabel_font_size=20,
title_x=0.5,
xaxis_title='PreferredPaymentMode',
yaxis_title='count',
)
fig.show()
```



laptop & accesories and mobile phones are the prefered category for all the city tiers

11- What is the preferred payment mode for different CityTiers?

```
df2['PreferredPaymentMode'].value_counts()
```

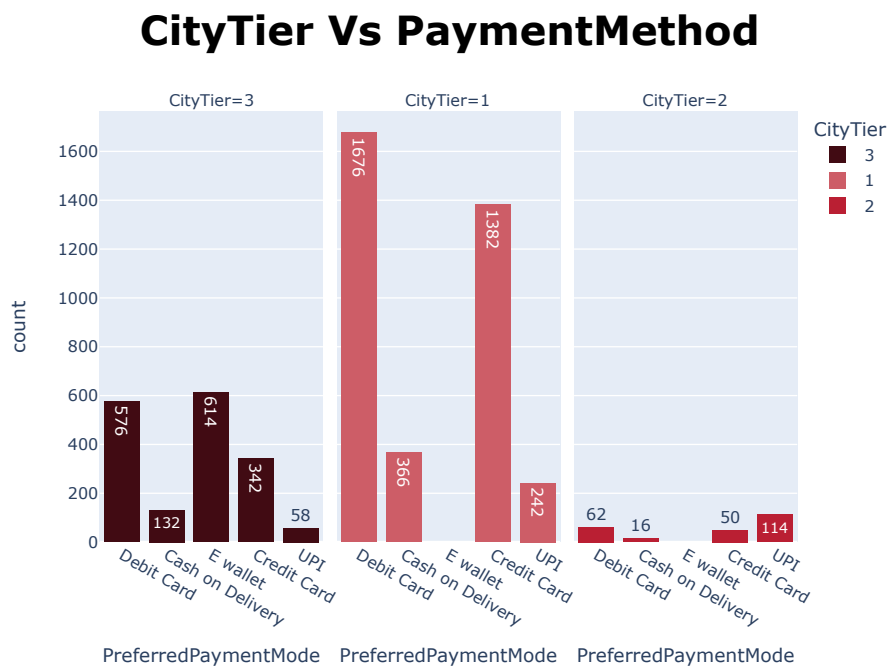
```
Debit Card          2314
Credit Card         1774
E wallet             614
Cash on Delivery     514
UPI                  414
Name: PreferredPaymentMode, dtype: int64
```

```
df2.groupby('CityTier')[['PreferredPaymentMode']].value_counts()
```

```
CityTier  PreferredPaymentMode
1         Debit Card             1676
          Credit Card            1382
          Cash on Delivery        366
          UPI                     242
2         UPI                     114
          Debit Card               62
          Credit Card              50
          Cash on Delivery         16
3         E wallet                614
          Debit Card              576
          Credit Card             342
          Cash on Delivery        132
```

```
UPI                    58
dtype: int64
```

```
import plotly.express as px

fig=px.histogram(df2,x="PreferredPaymentMode",facet_col="CityTier",color="CityTier",color_discrete_sequence=red_pale

# Customize the plot
fig.update_layout(hovermode='x',title_font_size=30)
fig.update_layout(
title_font_color="black",
template="plotly",
title_font_size=30,
hoverlabel_font_size=20,
title_x=0.5,
xaxis_title='PreferredPaymentMode',
yaxis_title='count',
)
fig.show()
```



preferred payment method for CityTier '1' ==> DebitCard

preferred payment method for CityTier '2' ==> UPI

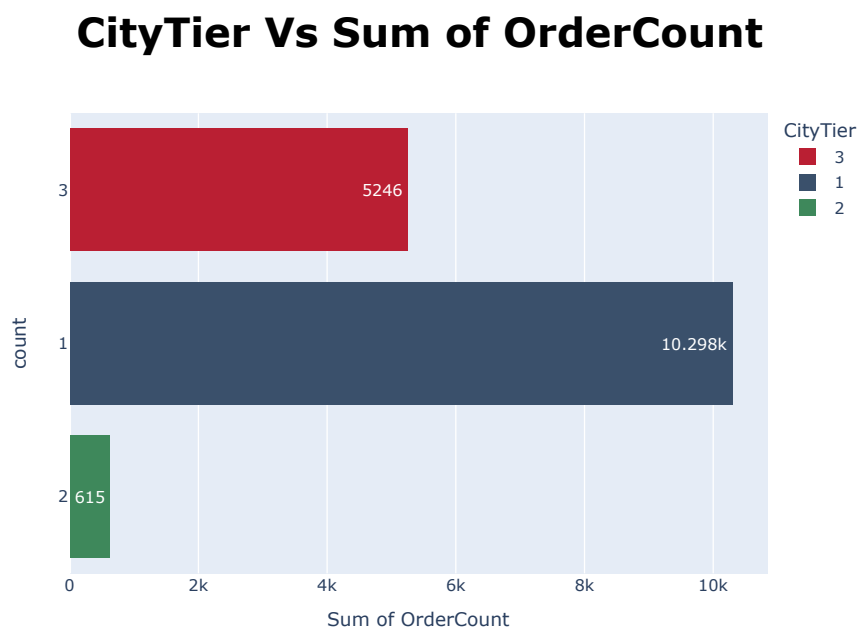
preferred payment method for CityTier '3' ==> E wallet

12-Which CityTier has the highest OrderCount?

```
df2.groupby('CityTier')[['OrderCount']].sum()
```

CityTier	OrderCount
1	10298.0
2	615.0
3	5246.0

```
fig = px.histogram(df2, x="OrderCount", y="CityTier", orientation="h", color="CityTier" ,text_auto= True , title="<b

# Customize the plot
fig.update_layout(hovermode='x',title_font_size=30)
fig.update_layout(
title_font_color="black",
template="plotly",
title_font_size=30,
hoverlabel_font_size=20,
title_x=0.5,
xaxis_title='Sum of OrderCount',
yaxis_title='count',
)
fig.show()
```



CityTier '1' has highest order count with 10298 orders

13-Does the percentage increase in order amount from last year affect churn rate?

```
df2['OrderAmountHikeFromlastYear'].value_counts()
```

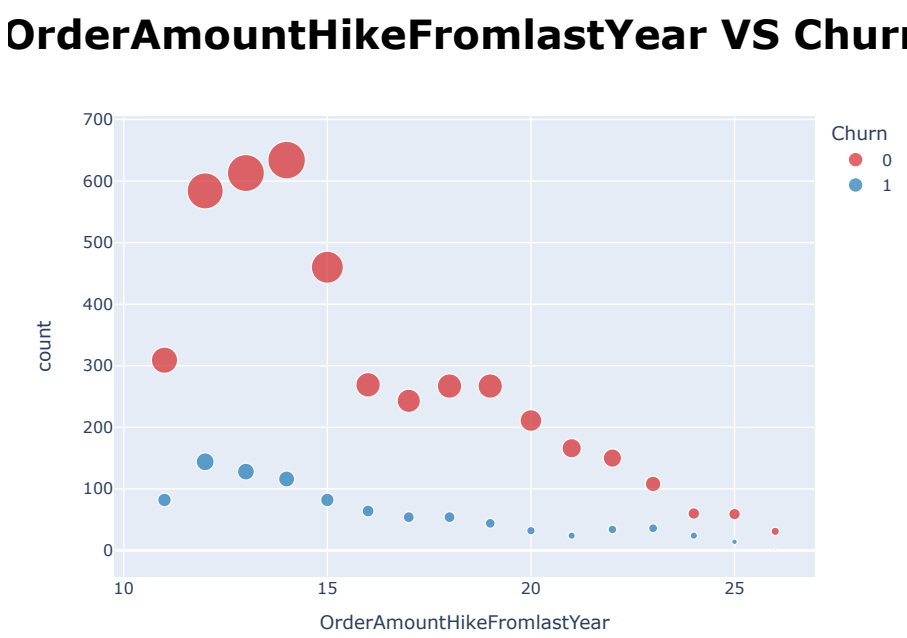
```
14.0    750
13.0    741
12.0    728
15.0    542
11.0    391
16.0    333
18.0    321
19.0    311
17.0    297
20.0    243
21.0    190
22.0    184
23.0    144
24.0     84
25.0     73
26.0     33
Name: OrderAmountHikeFromlastYear, dtype: int64
```

```
df2.groupby('OrderAmountHikeFromlastYear')['Churn'].count()
```

```
OrderAmountHikeFromlastYear
11.0    391
12.0    728
13.0    741
14.0    750
15.0    542
16.0    333
17.0    297
18.0    321
19.0    311
20.0    243
```

```
21.0    190
22.0    184
23.0    144
24.0     84
25.0     73
26.0     33
Name: Churn, dtype: int64
```

```
comp_ten = df2.groupby(["OrderAmountHikeFromlastYear", "Churn"]).size().reset_index(name="Count")

# Create a bubble chart using Plotly
fig_bubble = px.scatter(comp_ten, x="OrderAmountHikeFromlastYear", y="Count", size="Count", color="Churn", title="<b
                        color_discrete_sequence=["#d62728", "#1f77b4"])

# Customize the plot
fig_bubble.update_layout(hovermode='x',title_font_size=30)
fig_bubble.update_layout(
title_font_color="black",
template="plotly",
title_font_size=30,
hoverlabel_font_size=20,
title_x=0.5,
xaxis_title='OrderAmountHikeFromlastYear',
yaxis_title='count',
)
fig_bubble.show()
```



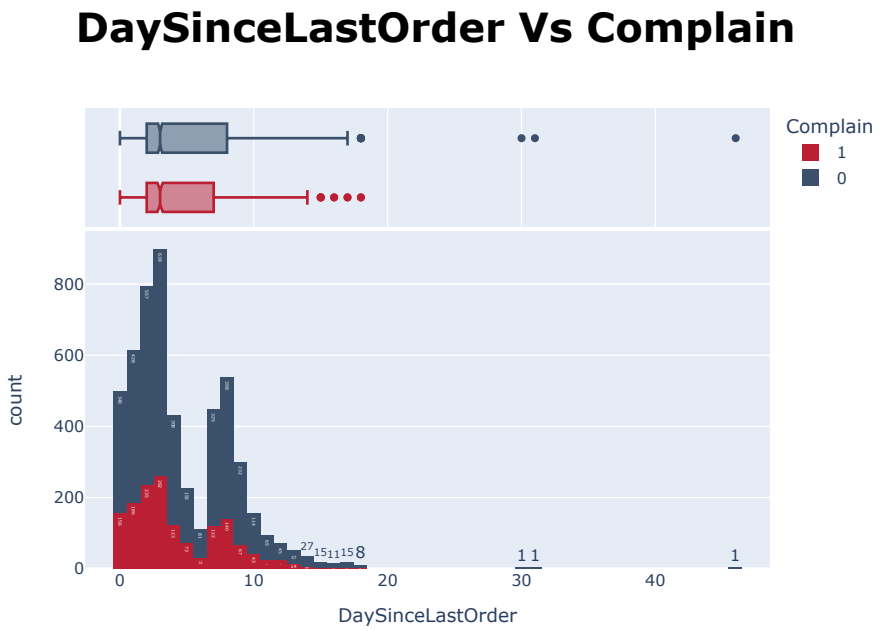
Graph Show when the percentage of order last year increase the churn rate decrease so OrderAmountHikeFromlastYear has postive effect on Churn rate and we need to focus when customer has percentage 12% - 14%

14-What is the relation between Complain and DaySinceLastOrder for churned customers?

```
df_c.groupby('Complain')[['DaySinceLastOrder']].sum()
```

Complain	DaySinceLastOrder
0	1313.0
1	1580.0

```
fig = px.histogram(df2, x="DaySinceLastOrder", color="Complain",text_auto= True , title="<b>DaySinceLastOrder Vs Com
                   marginal="box") # or violin, rug)

# Customize the plot
fig.update_layout(hovermode='x',title_font_size=30)
fig.update_layout(
title_font_color="black",
template="plotly",
title_font_size=30,
hoverlabel_font_size=20,
title_x=0.5,
xaxis_title='DaySinceLastOrder',
yaxis_title='count',
)
fig.show()
```



customers who didn't made complain has higher DaySinceLastOrder , however it's only one customer so its an outlier if we remove it we will customers with no complain has lower DaySinceLastOrder

15-What is the order counts for customers with high HourSpendOnApp?

```
# we will make binnig for column HourSpendOnApp
df2['HourSpendOnApp'].agg(['min','max'])
```

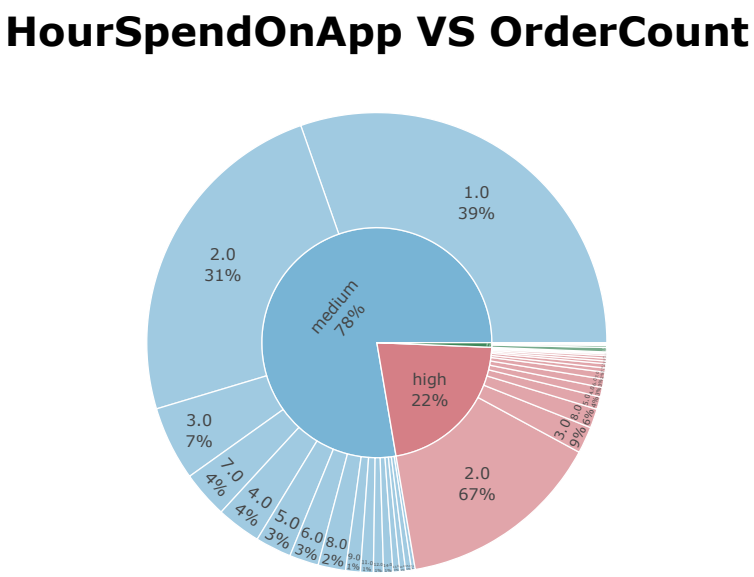
```
min    0.0
max    5.0
Name: HourSpendOnApp, dtype: float64
```

```
# Define the bin range
bins = [0 , 1 , 3 , 6]
label = ['low' , 'medium' , 'high']
# Create a new column 'HourSpendOnApp_bins' with the binned values
df2['HourSpendOnApp_bins'] = pd.cut(df2['HourSpendOnApp'], bins=bins , labels = label)
```

```
df2.groupby(['HourSpendOnApp_bins','OrderCount'])[['CustomerID']].count()
```

		CustomerID
HourSpendOnApp_bins	OrderCount	
low	1.0	16
	2.0	7
	3.0	1
	4.0	3
	5.0	0
	6.0	0
	7.0	4
	8.0	0
	9.0	0
	10.0	0
	11.0	1
	12.0	1
	13.0	0
	14.0	0
	15.0	0
	16.0	0
medium	1.0	1553
	2.0	1242
	3.0	267
	4.0	160
	5.0	130
	6.0	105
	7.0	169
	8.0	99
	9.0	53
	10.0	21
	11.0	46
	12.0	36
	13.0	24
	14.0	34
	15.0	21
	16.0	13
high	1.0	1
	2.0	738
	3.0	96
	4.0	34
	5.0	45
	6.0	30
	7.0	25
	8.0	69
	9.0	9
	10.0	15
	11.0	4
	12.0	15
	13.0	6
	14.0	2
	15.0	10
	16.0	10

```
sunbrust_gr = df2.loc[:,['HourSpendOnApp_bins','OrderCount']].dropna()


fig = px.sunburst(sunbrust_gr,path=['HourSpendOnApp_bins','OrderCount'],title="<b>"+'HourSpendOnApp VS OrderCount',te
fig.update_layout(hovermode='x',title_font_size=30)
fig.update_layout(
title_font_color="black",
template="plotly",
title_font_size=30,
hoverlabel_font_size=20,
title_x=0.5,
)
fig.update_traces(textinfo="label+percent parent")

fig.show()
```



Segment of customers has high spendtime on App has OrderCount 2 with percentage 67%

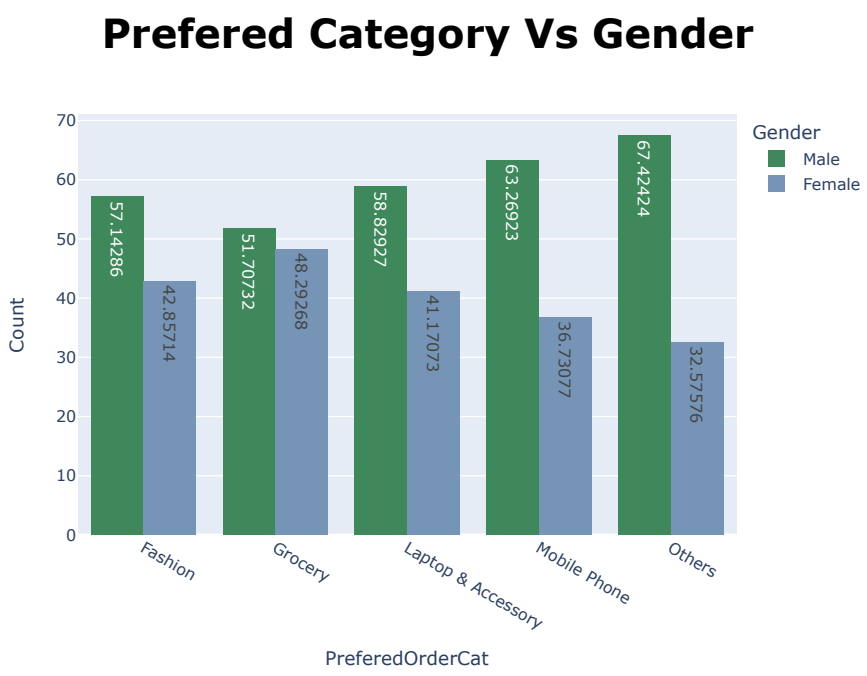
16-Is there a relationship between preferred order category and churn rate?

```
df2.groupby(['PreferedOrderCat' , 'Gender'])[['CustomerID']].count()
```

		CustomerID
PreferedOrderCat	Gender	
Fashion	Female	354
	Male	472
Grocery	Female	198
	Male	212
Laptop & Accessory	Female	844
	Male	1206
Mobile Phone	Female	764
	Male	1316
Others	Female	86
	Male	178

```
# # Group and count by 'PreferedOrderCat' and 'Churn'
# ordercat_churnrate = pd.DataFrame(df2.groupby('PreferedOrderCat')['Gender'].value_counts())
# ordercat_churnrate = ordercat_churnrate.rename(columns={'Gender': 'Count'})
# ordercat_churnrate = ordercat_churnrate.reset_index()
```

```
# fig = px.histogram(ordercat_churnrate, x='PreferedOrderCat', y = 'count',color='Gender', barmode='group',color_disc
# fig.update_layout(hovermode='x',title_font_size=30)
# fig.update_layout(
# title_font_color="black",
# template="plotly",
# title_font_size=30,
# hoverlabel_font_size=20,
# title_x=0.5,
# xaxis_title='PreferedOrderCat',
# yaxis_title='count',
# )
# fig.show()

fig = px.histogram(ordercat_churnrate,
                   x='PreferedOrderCat',
                   y='Count',  # Corrected column name here
                   color='Gender',
                   barmode='group',
                   color_discrete_sequence=pie_palette,
                   title="<b>"+'Prefered Category Vs Gender',
                   barnorm="percent",
                   text_auto=True)

fig.update_layout(hovermode='x', title_font_size=30)
fig.update_layout(
    title_font_color="black",
    template="plotly",
    title_font_size=30,
    hoverlabel_font_size=20,
    title_x=0.5,
    xaxis_title='PreferedOrderCat',
    yaxis_title='Count',  # Corrected y-axis label here
)
fig.show()
```



Top 2 Preferd Category For Males == > [Others , Mobile Phone]

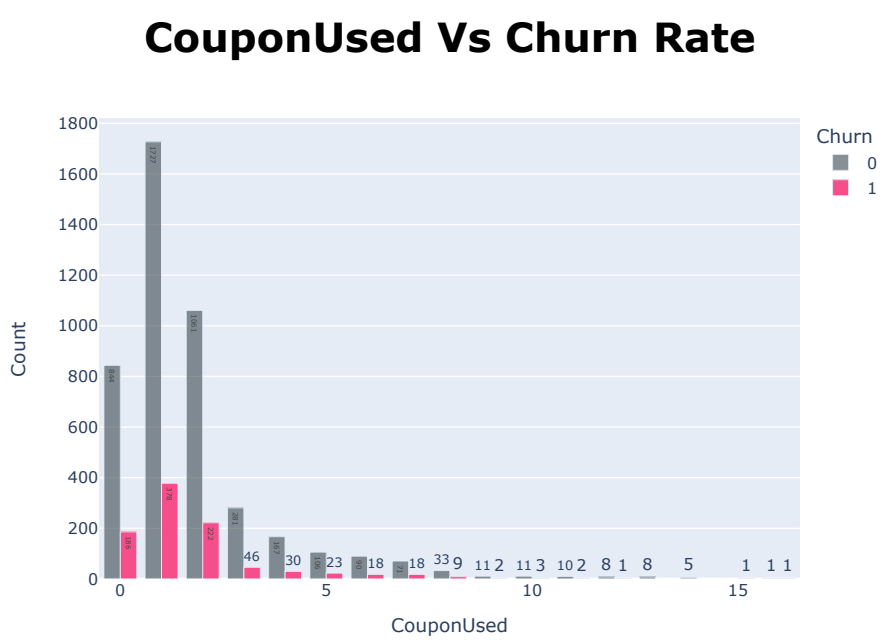
Top 2 Preferd Category For Females == > [Grocery , Fashion]

17-Do customers who used more coupons have lower churn rates?

```
df2.groupby(['CouponUsed' , 'Churn'])[['CustomerID']].count()
```

		CustomerID
CouponUsed	Churn	
0.0	0	844
	1	186
1.0	0	1727
	1	378
2.0	0	1061
	1	222
3.0	0	281
	1	46
4.0	0	167
	1	30
5.0	0	106
	1	23
6.0	0	90
	1	18
7.0	0	71
	1	18
8.0	0	33
	1	9
9.0	0	11
	1	2
10.0	0	11
	1	3
11.0	0	10
	1	2
12.0	0	8
	1	1
13.0	0	8
14.0	0	5
15.0	1	1
16.0	0	1
	1	1

```
fig = px.bar(coupoun_churnrate,
             x='CouponUsed',
             y='Count',  # Corrected column name here
             color='Churn',
             barmode='group',
             color_discrete_sequence=['rgba(58, 71, 80, 0.6)' ,'rgba(246, 78, 139, 1.0)'],
             title="<b>"+'CouponUsed Vs Churn Rate',
             text_auto=True)

fig.update_layout(hovermode='x', title_font_size=30)
fig.update_layout(
    title_font_color="black",
    template="plotly",
    title_font_size=30,
    hoverlabel_font_size=20,
    title_x=0.5,
    xaxis_title='CouponUsed',
    yaxis_title='Count',
)
fig.show()
```



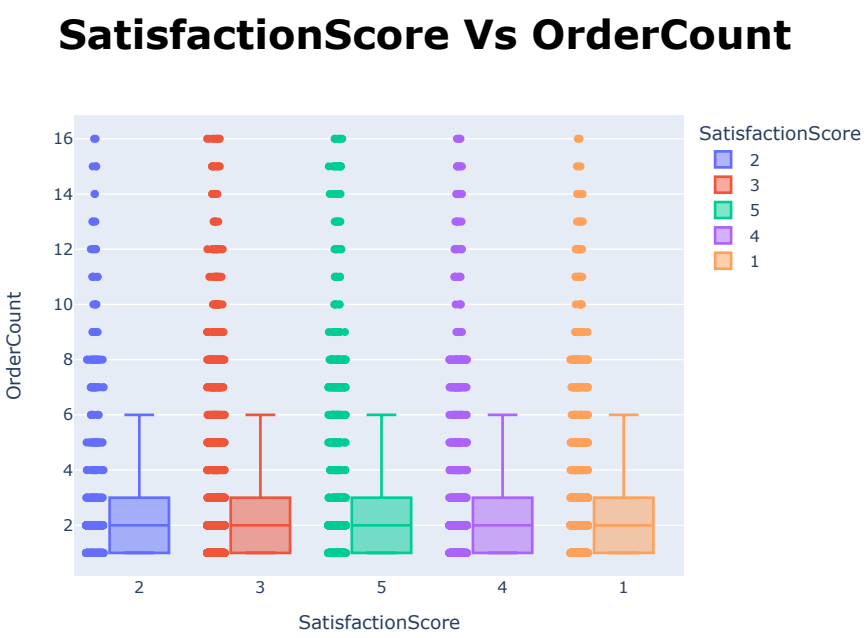
Grpah shows Churn become less when more coupons used

18-Is there a connection between satisfaction score and number of orders in the past month?

```
df2.groupby('SatisfactionScore')[['OrderCount']].count()
```

SatisfactionScore	OrderCount
1	1120
2	558
3	1618
4	1020
5	1056

```
fig = px.box(df2, y="OrderCount", x='SatisfactionScore', color="SatisfactionScore", title="<b>"+'SatisfactionScore Vs
             boxmode="overlay", points='all')
fig.update_layout(hovermode='x',title_font_size=30)
fig.update_layout(
title_font_color="black",
template="plotly",
title_font_size=30,
hoverlabel_font_size=20,
title_x=0.5,
xaxis_title='SatisfactionScore',
yaxis_title='OrderCount',
)
fig.show()
```



StatisfactionScore doesn't have affect on OrderCount

19-There is relation between CashbackAmount and order counts within churn?

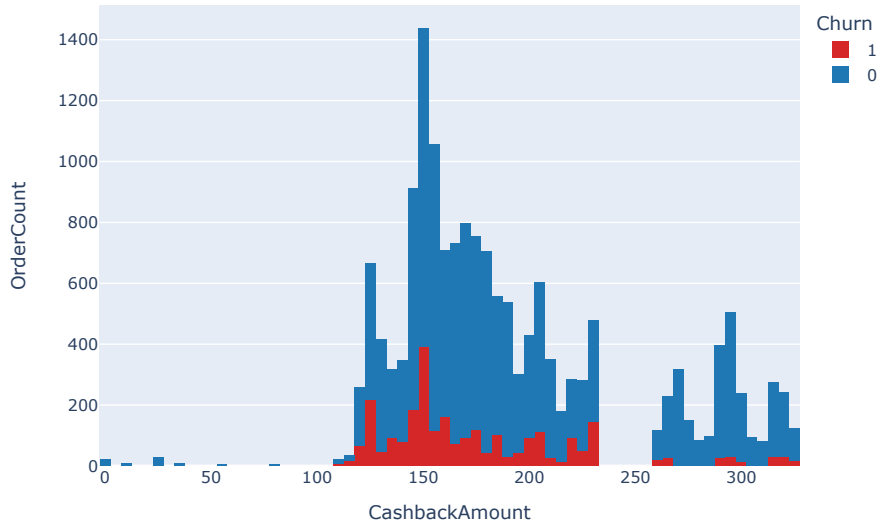
```
df_c.groupby(['OrderCount','CashbackAmount'])[['Churn']].count()
```

		Churn
OrderCount	CashbackAmount	
1.0	110.09	2
	110.81	2
	110.91	2
	111.02	2
	111.18	2
...	...	...
15.0	203.12	2
	295.45	2
16.0	152.43	2
	228.12	2
	320.45	2

461 rows × 1 columns

```
# fig = px.density_contour(df2, x="HourSpendOnApp", y="OrderCount", color = 'churn',
#                          title="<b>"+'HourSpendOnApp Vs OrderCount within churn',
#                           color_discrete_sequence=["#d62728", "#1f77b4"]
#                         )
fig = px.histogram(df2, x='CashbackAmount', y='OrderCount' ,color = 'Churn', title="<b>"+'CashbackAmount Vs OrderCou

# Customize the plot
fig.update_layout(hovermode='x',title_font_size=30)
fig.update_layout(
title_font_color="black",
template="plotly",
title_font_size=30,
hoverlabel_font_size=20,
title_x=0.5,
xaxis_title='CashbackAmount',
yaxis_title='OrderCount',
)
fig.show()
```

shbackAmount Vs OrderCount within chu



Graphs shows there is no relation between cash back amount and ordercount and there is postive relation between cashback amount and churn rate

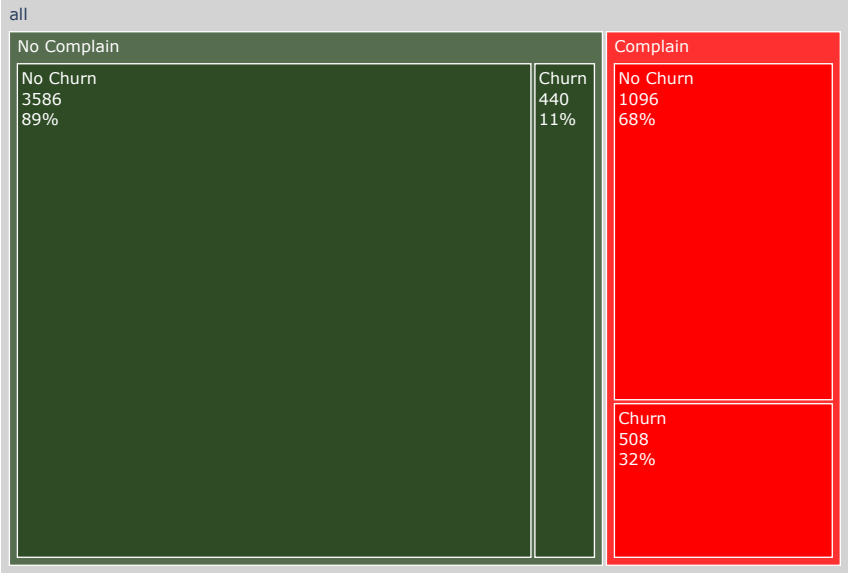
20-Are customers who complained more likely to churn?

```
df2.groupby('Complain')[['Churn']].count()
```

Complain	Churn
0	4026
1	1604

```
# Tree map
fig = px.treemap(comp_churn,
                 path=[px.Constant("all"), 'Complain', 'Churn'],
                 values='Count',  # Corrected column name here
                 color_discrete_sequence=["#2F4B26" , '#FF0000'],
                 title="<b>"+'Complain Vs Churn')

fig.update_traces(textinfo="label+percent parent+value" ,root_color="lightgrey")
fig.update_layout(margin=dict(t=70, l=25, r=25, b=25))

# Customize the plot
fig.update_layout(hovermode='x', title_font_size=30)
fig.update_layout(
    title_font_color="black",
    template="plotly",
    title_font_size=30,
    hoverlabel_font_size=20,
    title_x=0.5,
)

fig.show()
```

Complain Vs Churn



No complain doesn't affects on Churn as graph shows customers which made complains 68% doesn't make Churn

All EDA Insights

- There is not a big difference between the males and the females: avg order
- The males are more likely to churn as we have 63.3 % churned males from the app may be the company should consider incresing the products that grap the males interest and so on.. we are going to see if there is another factors that makes the highest segment of churned customers are males.
- The married are the highest customer segment in the comapny may be the comapny should consider taking care of the products that suits the single and the married customers as the singles are the most likely to churn from the app
- Citytier 2 has the highest tenure rate but the tenure rate does not seen to be a strong factor
- Citytier 3 has the highest order avg but it not to be a strong factor in the customer churning
- People with less satisfaction score spend less time on the app than the people of satisfaction score 5 but also i do not think there is any realation between the satisfaction score and people's spent time on the app
- City tier 1 has the most spended hours on the app
- There is a negative correlation between CityTier and NumberOfAddress. Higher CityTiers are associated with lower average NumberOfAddress and a more concentrated distribution, Customers in larger cities (CityTier 1) tend to have more addresses on average compared to smaller cities and towns in lower tiers, The relationship suggests address density and type of locality (metro vs smaller cities vs towns) impacts how many addresses customers have across city types.
- There is a weak negative relation between complainig and the number of days since last order
- mobile phone users are likely to churn may be this indicates a problem on the app user experience on the app mobile version
- Inference: As the distance from warehouse to home is similar in all city tier which means company had build warehouse in lower city tier also.
- laptop & accesories and mobile phones are the prefered category for all the city tiers
- Preferred payment method for CityTier '1' ==> DebitCard
  Preferred payment method for CityTier '2' ==> UPI
  Preferred payment method for CityTier '3' ==> E wallet
  There is big common in debit card method in tiers
- CityTier '1' has highest order count with 10298 orders
  CityTier '3' has highest mean ordercount that means CityTier '3' thier count small and they have many orders 'richTier'
- When the percentage of order last year increase the churn rate decrease so OrderAmountHikeFromlastYear has postive effect on Churn rate and we need to focus when customer has percentage 12% - 15%
- customers who didn't made complain has higher DaySinceLastOrder , however it's only one customer so its an outlier if we remove it we will customers with no complain has lower DaySinceLastOrder
- Segment of customers has high spendtime on App has OrderCount 2 with percentage 67%

- Top 2 Preferd Category For Males == > [Others , Mobile Phone]
  Top 2 Preferd Category For Females == > [Grocery , Fashion]
- Churn become less when more coupons used
- StatisfactionScore doesn't have affect on OrderCount
- There is no relation between cash back amount and ordercount and there is postive relation between cashback amount and churn rate
- Complain doesn't affects on Churn, Customers which made complains 68% doesn't make Churn

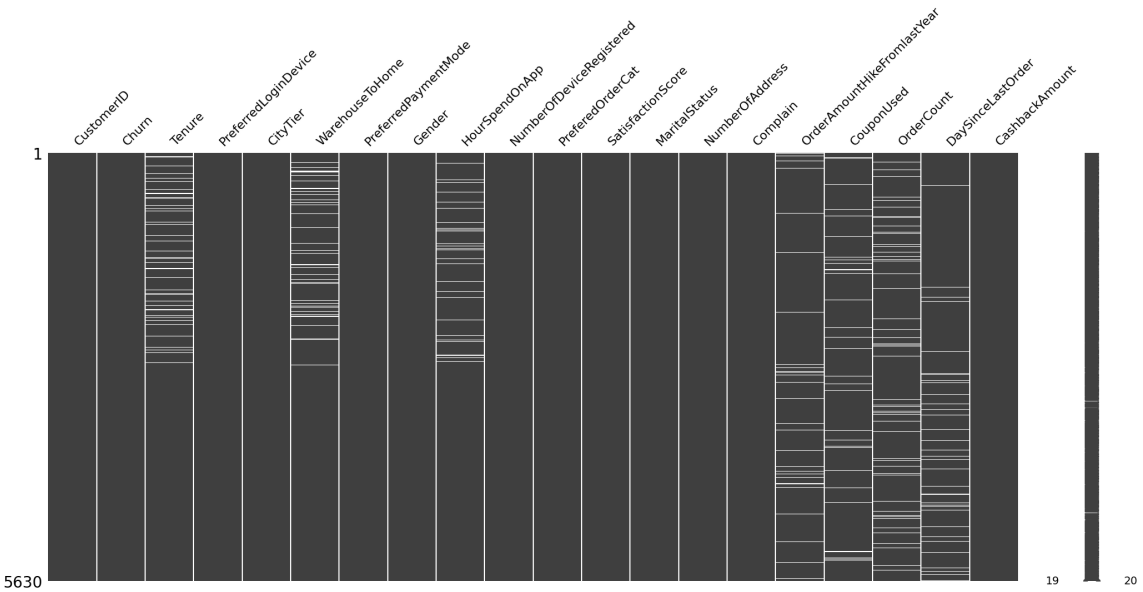
Data Preprocessing

Handling Missing Values

```
round((df.isnull().sum()*100 / df.shape[0]),2)
```

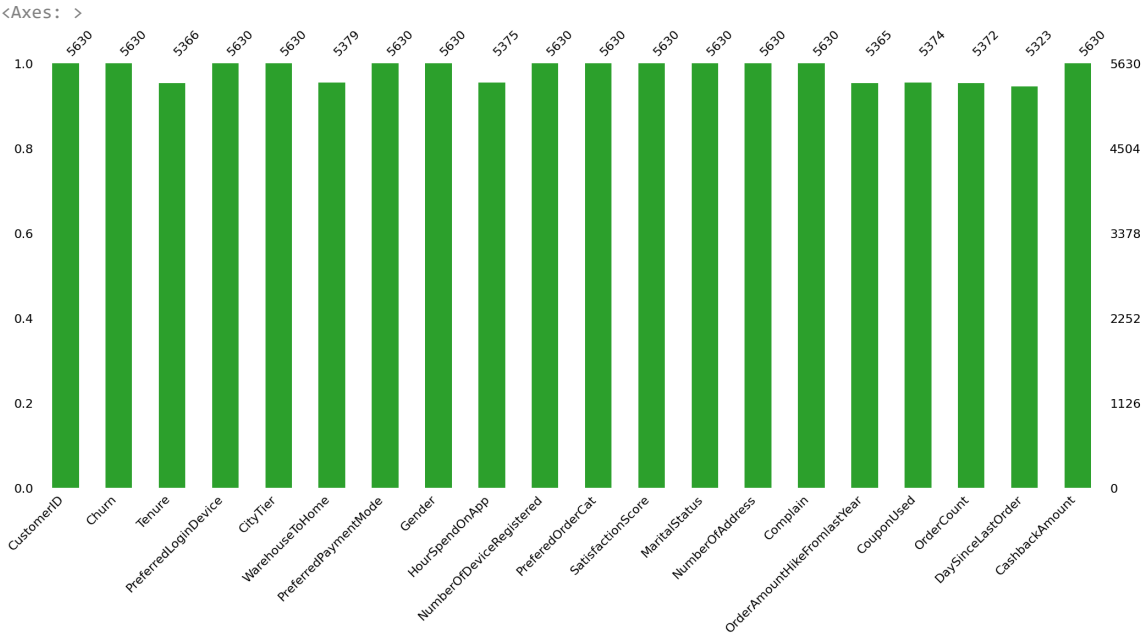
```
CustomerID                   0.00
Churn                        0.00
Tenure                       4.69
PreferredLoginDevice         0.00
CityTier                     0.00
WarehouseToHome              4.46
PreferredPaymentMode         0.00
Gender                       0.00
HourSpendOnApp               4.53
NumberOfDeviceRegistered     0.00
PreferedOrderCat             0.00
SatisfactionScore            0.00
MaritalStatus                0.00
NumberOfAddress              0.00
Complain                     0.00
OrderAmountHikeFromlastYear  4.71
CouponUsed                   4.55
OrderCount                   4.58
DaySinceLastOrder            5.45
CashbackAmount               0.00
dtype: float64
```

```
msno.matrix(df)
```




```
msno.bar(df , color="tab:green")
```

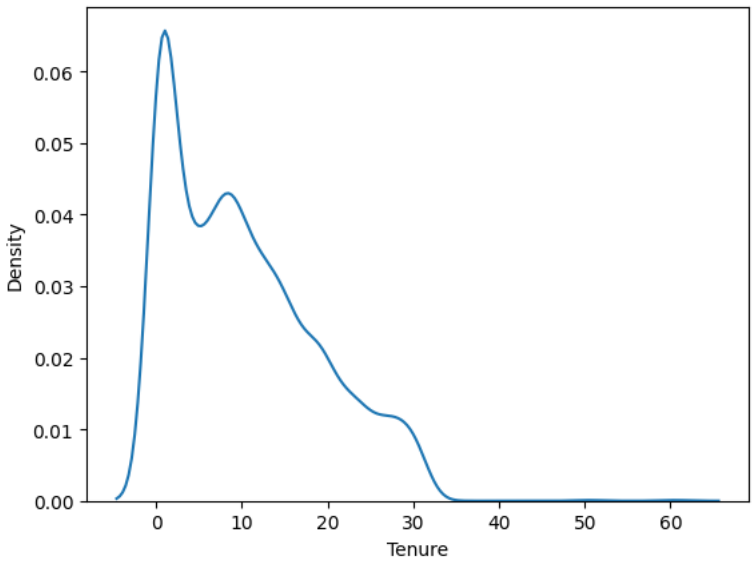
```
<Axes: >
```



All Missing values less than 6% so we can impute them

```
sns.kdeplot(df , x='Tenure')
```

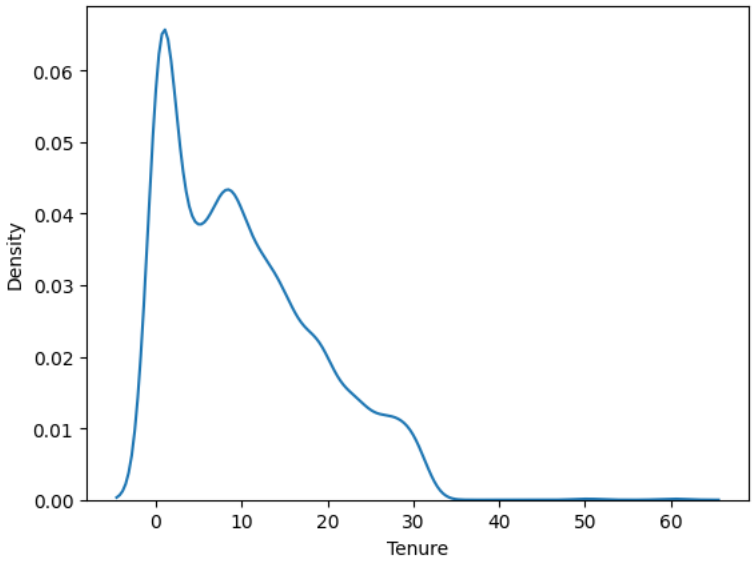
```
<Axes: xlabel='Tenure', ylabel='Density'>
```




```
# impute with bfill Method
df['Tenure'] = df['Tenure'].fillna(method = 'bfill')
```

```
sns.kdeplot(df , x='Tenure')
```

```
<Axes: xlabel='Tenure', ylabel='Density'>
```

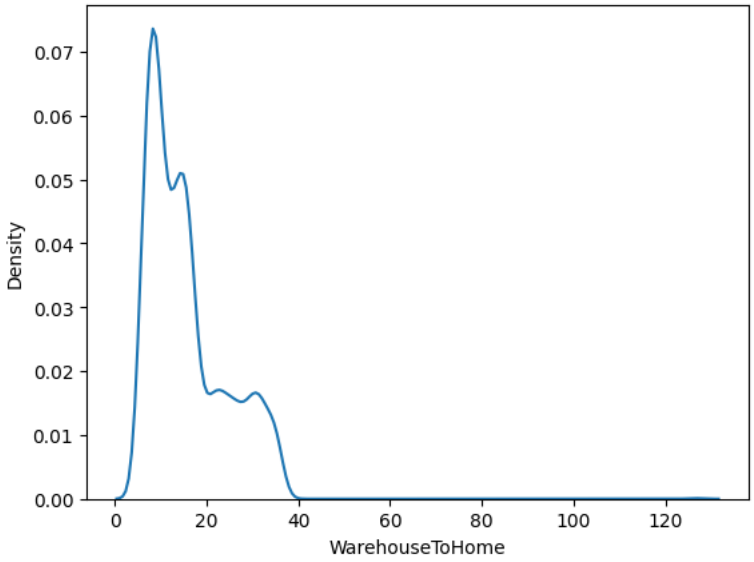



```
df['Tenure'].isnull().sum()
```

```
0
```

```
sns.kdeplot(df , x='WarehouseToHome')
```

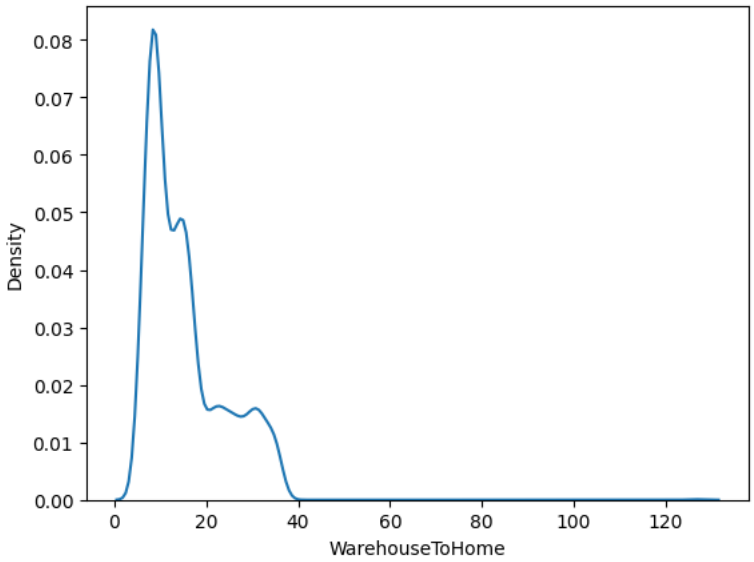
```
<Axes: xlabel='WarehouseToHome', ylabel='Density'>
```




```
# Impute with simple imputer
from sklearn.impute import SimpleImputer
s_imp = SimpleImputer(missing_values=np.nan , strategy = 'most_frequent')
df['WarehouseToHome'] = s_imp.fit_transform(pd.DataFrame(df['WarehouseToHome']))
```

```
sns.kdeplot(df , x='WarehouseToHome')
```

```
<Axes: xlabel='WarehouseToHome', ylabel='Density'>
```

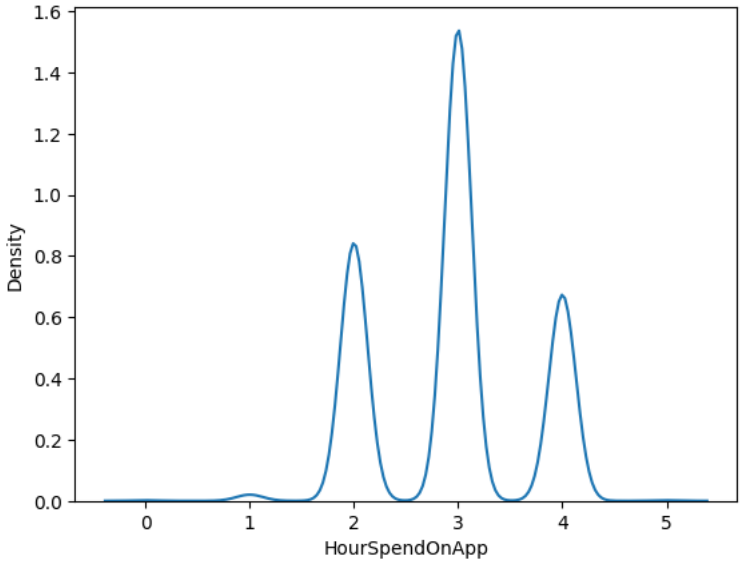



```
df['WarehouseToHome'].isnull().sum()
```

```
0
```

```
sns.kdeplot(df , x='HourSpendOnApp')
```

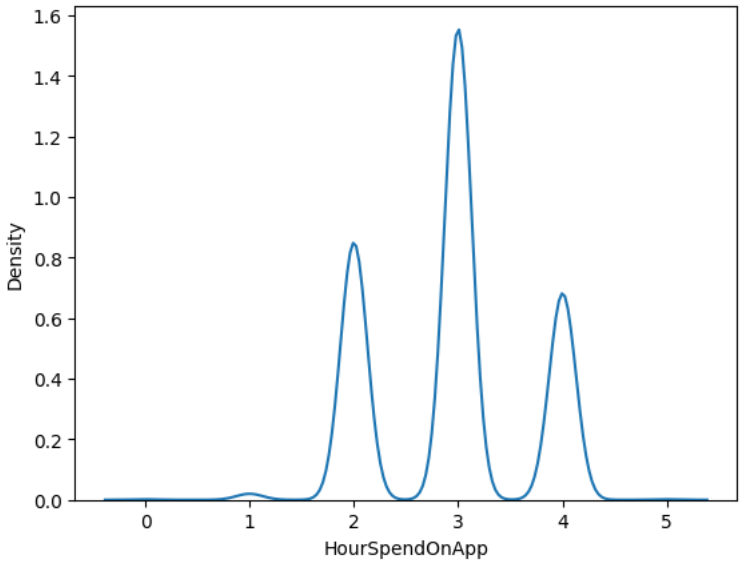
```
<Axes: xlabel='HourSpendOnApp', ylabel='Density'>
```




```
fill_list = df['HourSpendOnApp'].dropna()
df['HourSpendOnApp'] = df['HourSpendOnApp'].fillna(pd.Series(np.random.choice(fill_list , size = len(df['HourSpendOnA
```

```
sns.kdeplot(df , x='HourSpendOnApp')
```

```
<Axes: xlabel='HourSpendOnApp', ylabel='Density'>
```

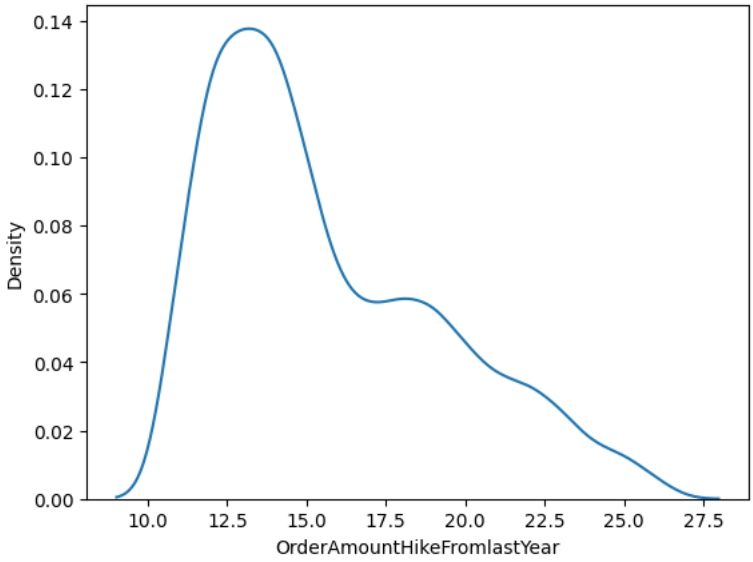



```
df['HourSpendOnApp'].isnull().sum()
```

```
0
```

```
sns.kdeplot(df , x='OrderAmountHikeFromlastYear')
```

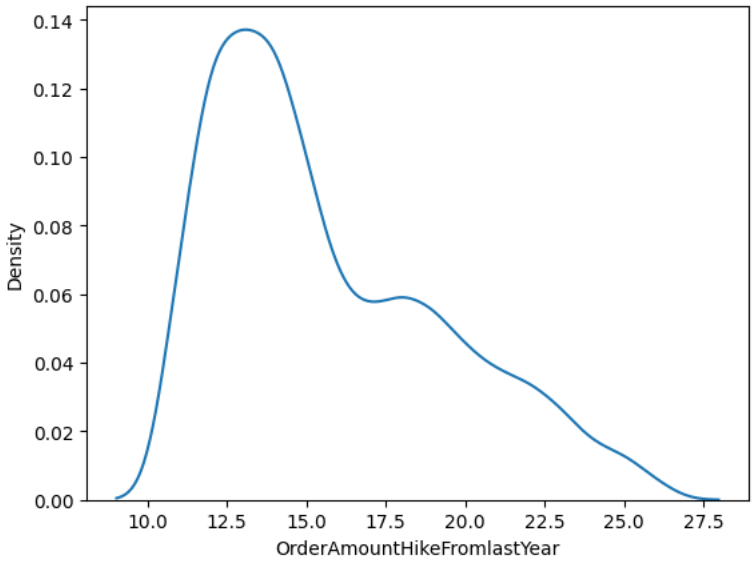
```
<Axes: xlabel='OrderAmountHikeFromlastYear', ylabel='Density'>
```




```
# impute with ffill method
df['OrderAmountHikeFromlastYear'] = df['OrderAmountHikeFromlastYear'].fillna(method = 'ffill')
```

```
sns.kdeplot(df , x='OrderAmountHikeFromlastYear')
```

```
<Axes: xlabel='OrderAmountHikeFromlastYear', ylabel='Density'>
```

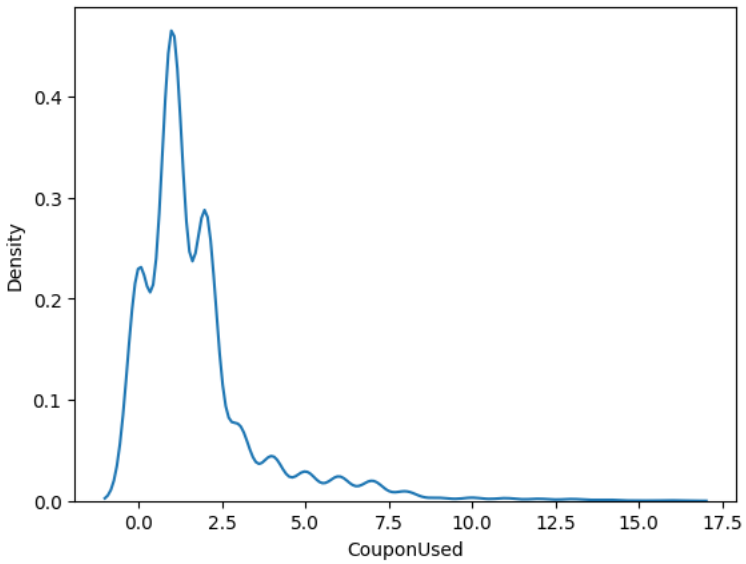



```
df['OrderAmountHikeFromlastYear'].isnull().sum()
```

```
0
```

```
sns.kdeplot(df , x='CouponUsed')
```

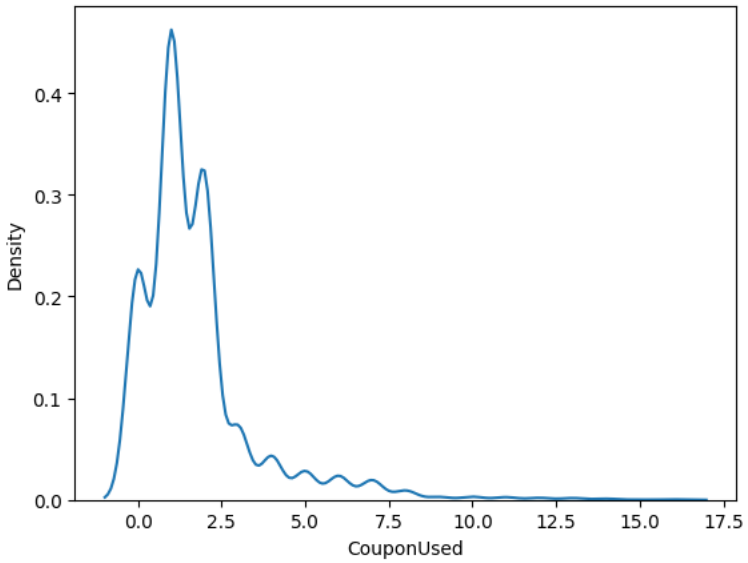
```
<Axes: xlabel='CouponUsed', ylabel='Density'>
```




```
# Impute with KNN Imputer
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=2)
df['CouponUsed']=imputer.fit_transform(df[['CouponUsed']])
```

```
sns.kdeplot(df , x='CouponUsed')
```

```
<Axes: xlabel='CouponUsed', ylabel='Density'>
```

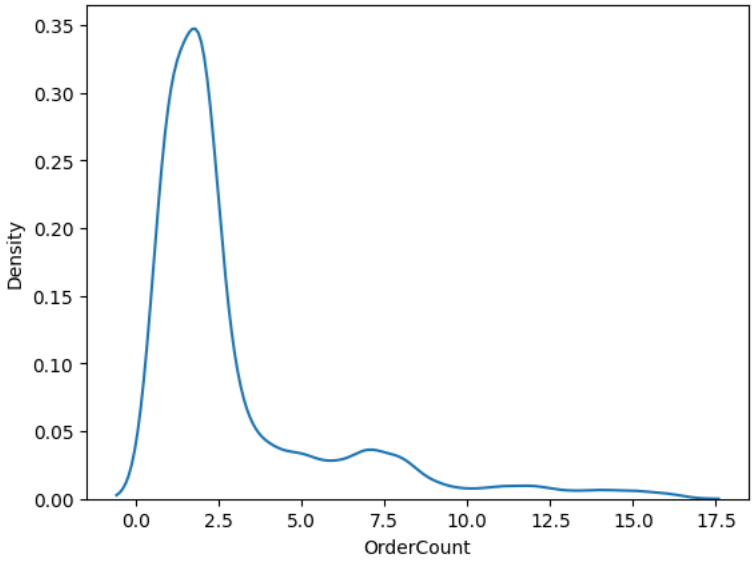



```
df['CouponUsed'].isnull().sum()
```

```
0
```

```
sns.kdeplot(df , x='OrderCount')
```

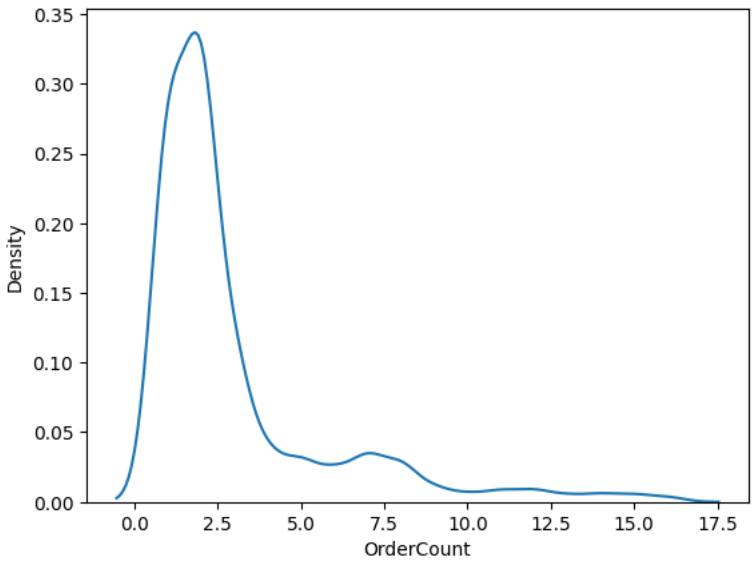
```
<Axes: xlabel='OrderCount', ylabel='Density'>
```




```
# Impute with KNN imputer
imputer_2 = KNNImputer(n_neighbors=2)
df['OrderCount']=imputer_2.fit_transform(df[['OrderCount']])
```

```
sns.kdeplot(df , x='OrderCount')
```

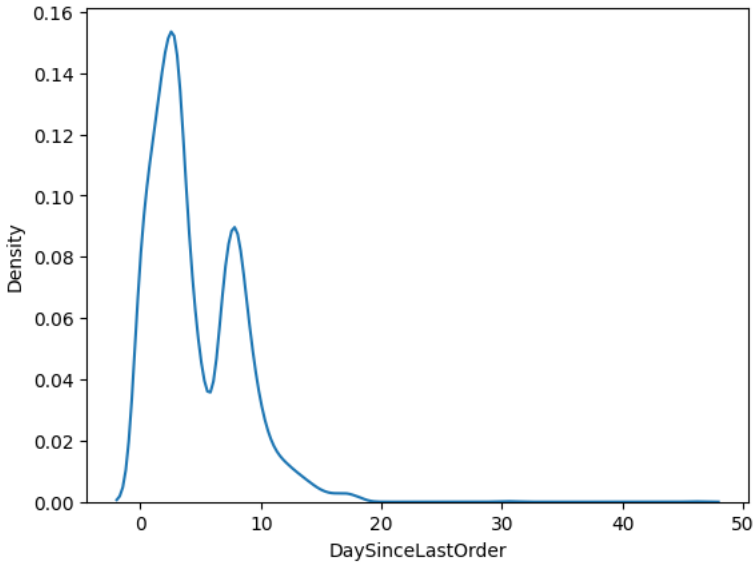
```
<Axes: xlabel='OrderCount', ylabel='Density'>
```




```
df['OrderCount'].isnull().sum()
```

```
sns.kdeplot(df , x='DaySinceLastOrder')
```

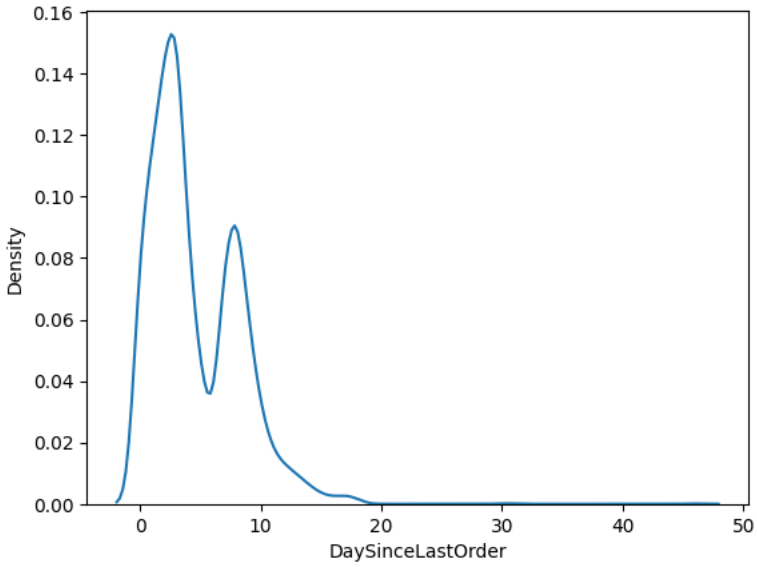
```
<Axes: xlabel='DaySinceLastOrder', ylabel='Density'>
```




```
# impute with bfill Method
df['DaySinceLastOrder'] = df['DaySinceLastOrder'].fillna(method = 'bfill')
```

```
sns.kdeplot(df , x='DaySinceLastOrder')
```

```
<Axes: xlabel='DaySinceLastOrder', ylabel='Density'>
```




```
df['DaySinceLastOrder'].isnull().sum()
```

```
0
```

```
# After we Checked the data the Customer ID Column not important for our Models so We drop it
df.drop('CustomerID' , axis = 1 , inplace = True)
```

```
df.shape
```

```
(5630, 19)
```

We Handled Mssing Values

Encoding

```
# check before encoding that my catogries for my columns are limited
for i in df.columns:
    if df[i].dtype == 'object':
        print(df[i].value_counts())
        print('*' * 40)
```

```
Mobile Phone    3996
Computer        1634
Name: PreferredLoginDevice, dtype: int64
****************************************
Debit Card          2314
Credit Card         1774
E wallet             614
Cash on Delivery     514
UPI                  414
Name: PreferredPaymentMode, dtype: int64
****************************************
Male      3384
Female    2246
Name: Gender, dtype: int64
****************************************
Mobile Phone          2080
Laptop & Accessory    2050
Fashion                826
Grocery                410
Others                 264
Name: PreferedOrderCat, dtype: int64
****************************************
Married     2986
Single      1796
Divorced     848
Name: MaritalStatus, dtype: int64
****************************************
```

```
# cat columns
data = df[df.select_dtypes(exclude=np.number).columns]
data
```

	PreferredLoginDevice	PreferredPaymentMode	Gender	PreferedOrderCat	MaritalStatus
0	Mobile Phone	Debit Card	Female	Laptop & Accessory	Single
1	Mobile Phone	UPI	Male	Mobile Phone	Single
2	Mobile Phone	Debit Card	Male	Mobile Phone	Single
3	Mobile Phone	Debit Card	Male	Laptop & Accessory	Single
4	Mobile Phone	Credit Card	Male	Mobile Phone	Single
...	...	...	...	...	...
5625	Computer	Credit Card	Male	Laptop & Accessory	Married
5626	Mobile Phone	Credit Card	Male	Fashion	Married
5627	Mobile Phone	Debit Card	Male	Laptop & Accessory	Married
5628	Computer	Credit Card	Male	Laptop & Accessory	Married
5629	Mobile Phone	Credit Card	Male	Laptop & Accessory	Married

5630 rows × 5 columns

```
le = LabelEncoder()


# Encode for cat_cols
for i in df.columns:
  if df[i].dtype == 'object':
    df[i] = le.fit_transform(df[i])

df.head(4)
```

	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPaymentMode	Gender	HourSpendOnApp
0	1	4.0	1	3	6.0	2	0	3.0
1	1	0.0	1	1	8.0	4	1	3.0
2	1	0.0	1	1	30.0	2	1	2.0
3	1	0.0	1	3	15.0	2	1	2.0

```
for i in data.columns:
    data[i] = le.fit_transform(data[i])

data.head(4)
```

	PreferredLoginDevice	PreferredPaymentMode	Gender	PreferedOrderCat	MaritalStatus
0	1	2	0	2	2
1	1	4	1	3	2
2	1	2	1	3	2
3	1	2	1	2	2

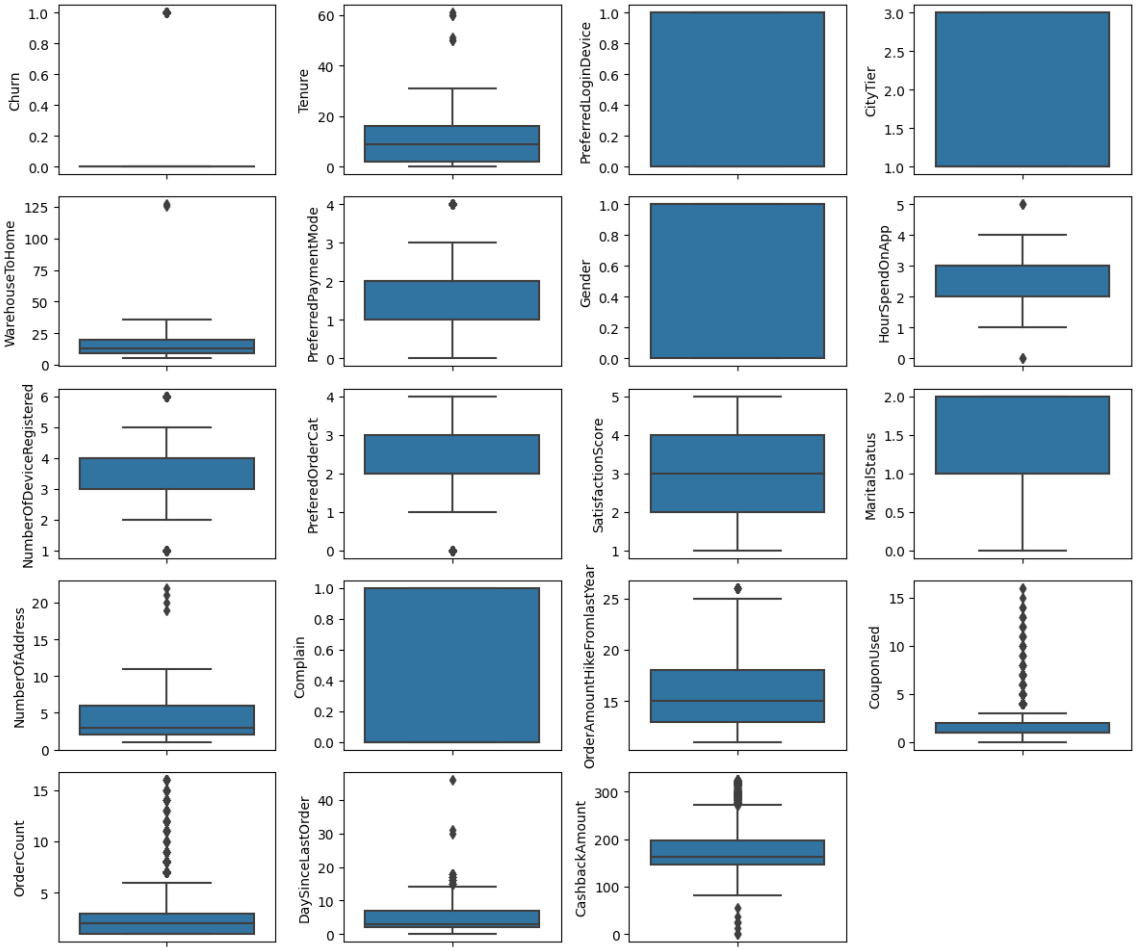
Handling Outliers

```
df.dtypes
```

```
Churn                         int64
Tenure                      float64
PreferredLoginDevice          int64
CityTier                      int64
WarehouseToHome             float64
PreferredPaymentMode          int64
Gender                        int64
HourSpendOnApp              float64
NumberOfDeviceRegistered      int64
PreferedOrderCat              int64
SatisfactionScore             int64
MaritalStatus                 int64
NumberOfAddress               int64
Complain                      int64
OrderAmountHikeFromlastYear float64
CouponUsed                  float64
OrderCount                  float64
DaySinceLastOrder           float64
```

```
CashbackAmount              float64
dtype: object
```

```
fig = plt.figure(figsize=(12,18))
for i in range(len(df.columns)):
    fig.add_subplot(9,4,i+1)
    sns.boxplot(y=df.iloc[:,i])

plt.tight_layout()
plt.show()
```




```
# lets detect True Outliers
def handle_outliers(df , column_name):
  Q1 = df[column_name].quantile(0.25)
  Q3 = df[column_name].quantile(0.75)
  IQR = Q3 - Q1

  # Define Upper and lower boundaries
  Upper = Q3 + IQR * 1.5
  lower = Q1 - IQR * 1.5

  # lets make filter for col values
  new_df = df[ (df[column_name] > lower) & (df[column_name] < Upper) ]

  return new_df
```

```
df.columns
```

```
Index(['Churn', 'Tenure', 'PreferredLoginDevice', 'CityTier',
       'WarehouseToHome', 'PreferredPaymentMode', 'Gender', 'HourSpendOnApp',
       'NumberOfDeviceRegistered', 'PreferedOrderCat', 'SatisfactionScore',
       'MaritalStatus', 'NumberOfAddress', 'Complain',
       'OrderAmountHikeFromlastYear', 'CouponUsed', 'OrderCount',
       'DaySinceLastOrder', 'CashbackAmount'],
      dtype='object')
```

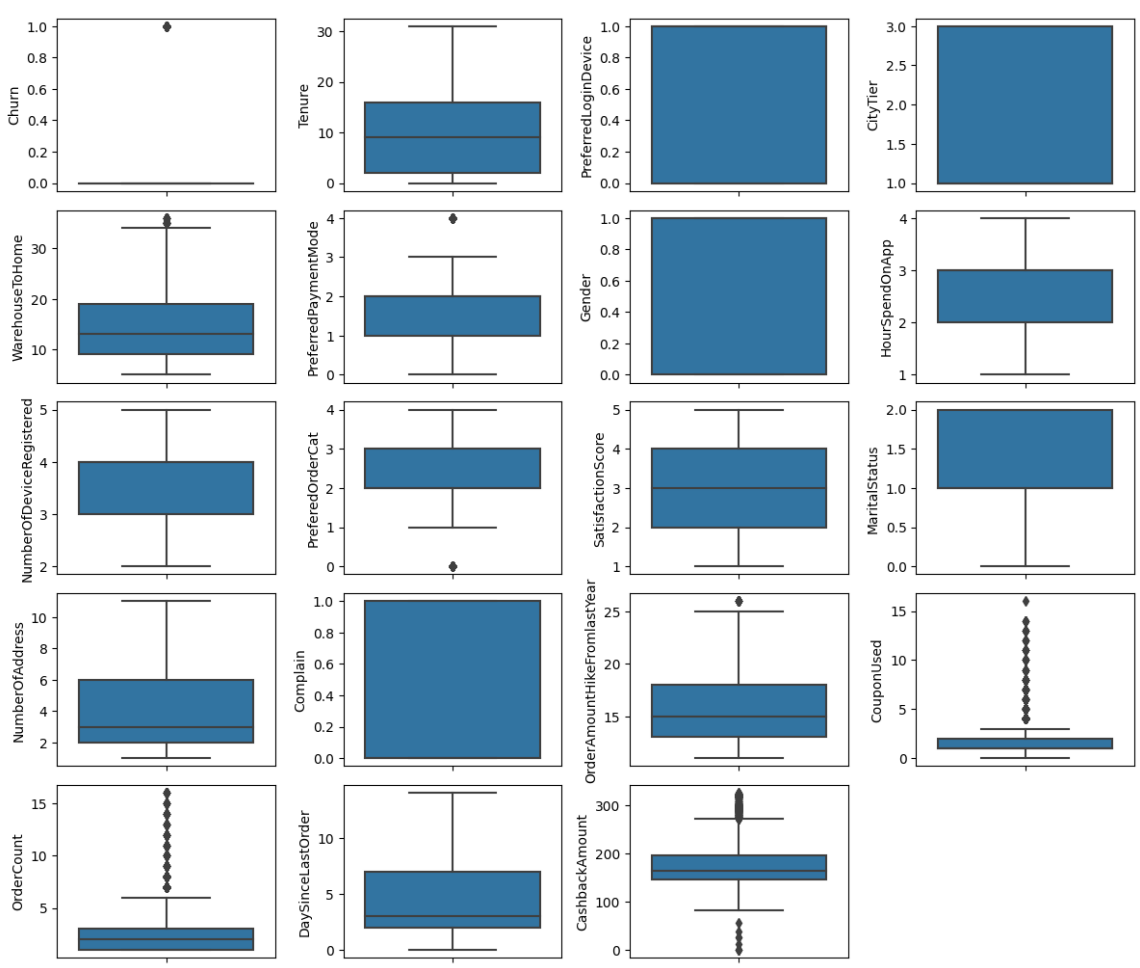
```
# lets Give our Functions columns contains outlier
cols_outliers = ['Tenure' , 'WarehouseToHome' , 'NumberOfAddress' , 'DaySinceLastOrder' , 'HourSpendOnApp' , 'NumberO
```

```
for col in cols_outliers:
    df = handle_outliers(df , col)
```

```
df.head(4)
```

	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPaymentMode	Gender	HourSpendOnApp
0	1	4.0	1	3	6.0	2	0	3.0
1	1	0.0	1	1	8.0	4	1	3.0
2	1	0.0	1	1	30.0	2	1	2.0
3	1	0.0	1	3	15.0	2	1	2.0

```
fig = plt.figure(figsize=(12,18))
for i in range(len(df.columns)):
    fig.add_subplot(9,4,i+1)
    sns.boxplot(y=df.iloc[:,i])

plt.tight_layout()
plt.show()
```



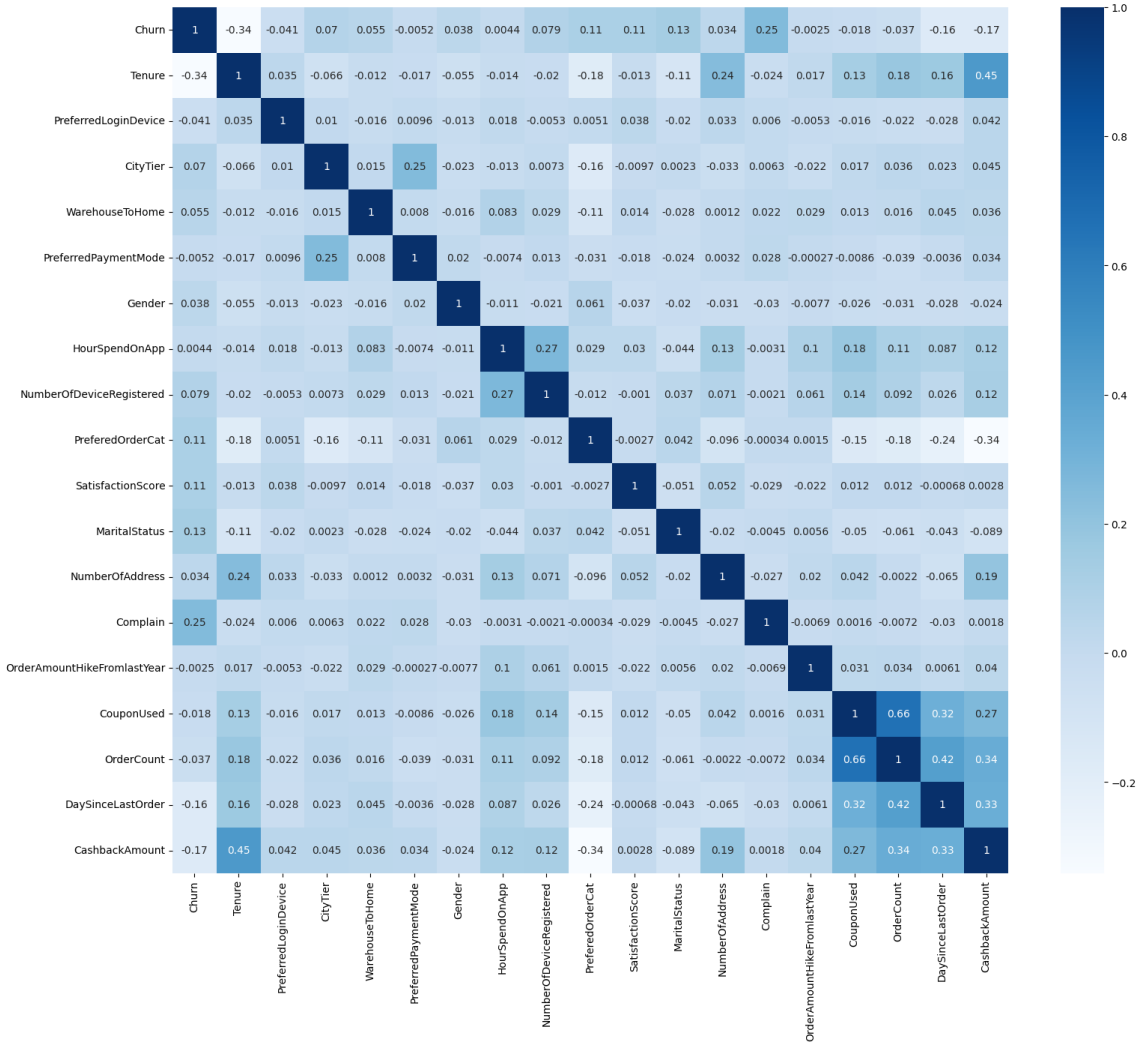
we made Trim on cols that contains outliers but after we check we saw many inforamtion deleted so we made Trimming only on cols that not conatins many outliers

```
corr_matrix = df.corr()
corr_matrix
```

	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPaym
Churn	1.000000	-0.336058	-0.041250	0.069595	0.054768	-(
Tenure	-0.336058	1.000000	0.034596	-0.065933	-0.011849	-(
PreferredLoginDevice	-0.041250	0.034596	1.000000	0.010097	-0.015852	(
CityTier	0.069595	-0.065933	0.010097	1.000000	0.014636	(
WarehouseToHome	0.054768	-0.011849	-0.015852	0.014636	1.000000	(
PreferredPaymentMode	-0.005156	-0.016797	0.009610	0.251539	0.008046	1
Gender	0.038193	-0.054684	-0.012892	-0.022759	-0.015904	(
HourSpendOnApp	0.004377	-0.013747	0.017541	-0.012600	0.082553	-(
NumberOfDeviceRegistered	0.079116	-0.019592	-0.005323	0.007282	0.029049	(
PreferedOrderCat	0.105149	-0.180637	0.005137	-0.164040	-0.114357	-(
SatisfactionScore	0.108600	-0.013331	0.037642	-0.009735	0.013783	-(
MaritalStatus	0.131982	-0.111074	-0.020207	0.002254	-0.028226	-(
NumberOfAddress	0.033703	0.240939	0.033310	-0.033363	0.001173	(
Complain	0.252346	-0.023903	0.005983	0.006312	0.022052	(
OrderAmountHikeFromlastYear	-0.002545	0.017177	-0.005296	-0.022135	0.028927	-(
CouponUsed	-0.017914	0.127314	-0.015940	0.017139	0.013175	-(
OrderCount	-0.036568	0.181138	-0.021975	0.035656	0.016235	-(
DaySinceLastOrder	-0.164448	0.164444	-0.027906	0.023394	0.044883	-(
CashbackAmount	-0.165008	0.453981	0.042321	0.044946	0.036318	(

```
plt.figure(figsize = (18,15))
sns.heatmap(df.corr() , annot = True , cmap = 'Blues')
```

```
<Axes: >
```



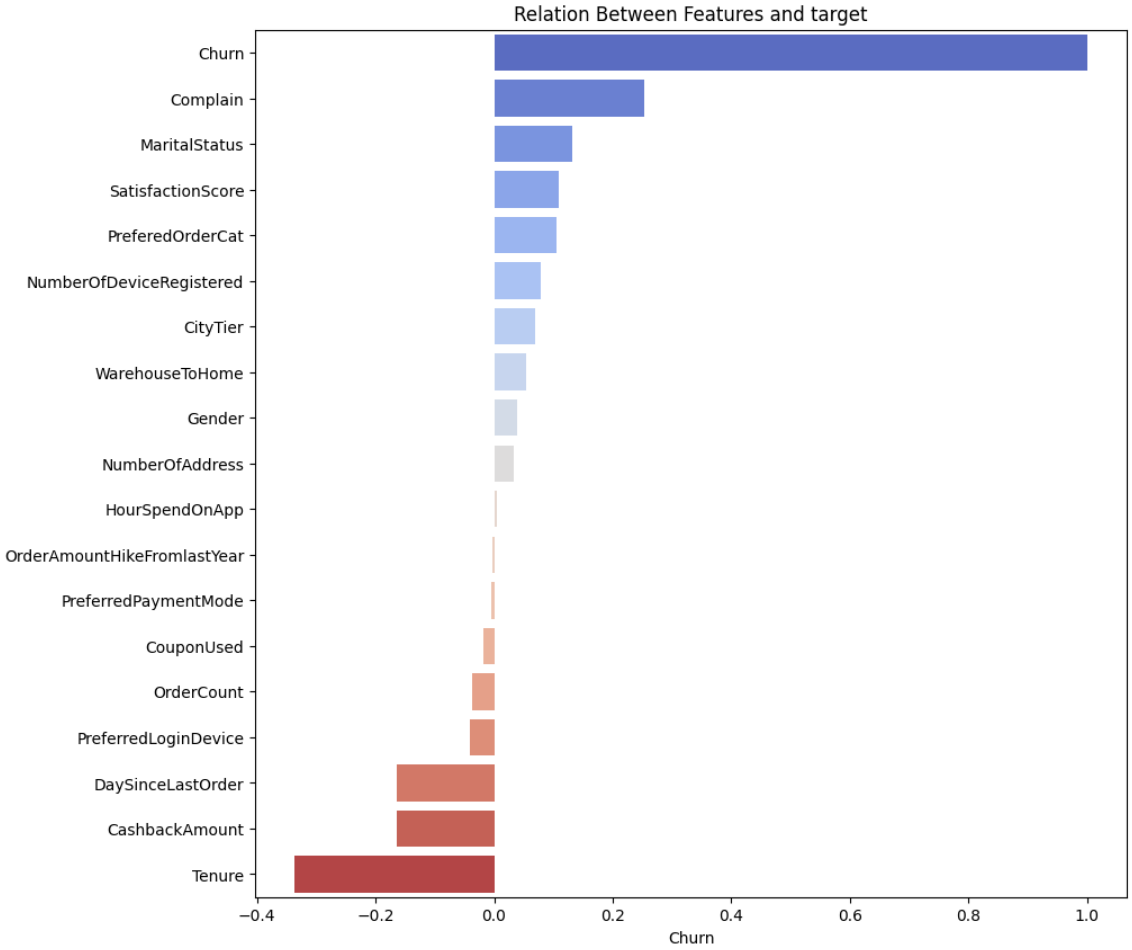

```
churn_corr_vector = corr_matrix['Churn'].sort_values(ascending = False)
churn_corr_vector
```

```
Churn                          1.000000
Complain                       0.252346
MaritalStatus                  0.131982
SatisfactionScore              0.108600
PreferedOrderCat               0.105149
NumberOfDeviceRegistered       0.079116
CityTier                       0.069595
WarehouseToHome                0.054768
Gender                         0.038193
NumberOfAddress                0.033703
HourSpendOnApp                 0.004377
OrderAmountHikeFromlastYear   -0.002545
PreferredPaymentMode          -0.005156
CouponUsed                    -0.017914
OrderCount                    -0.036568
PreferredLoginDevice          -0.041250
DaySinceLastOrder             -0.164448
```

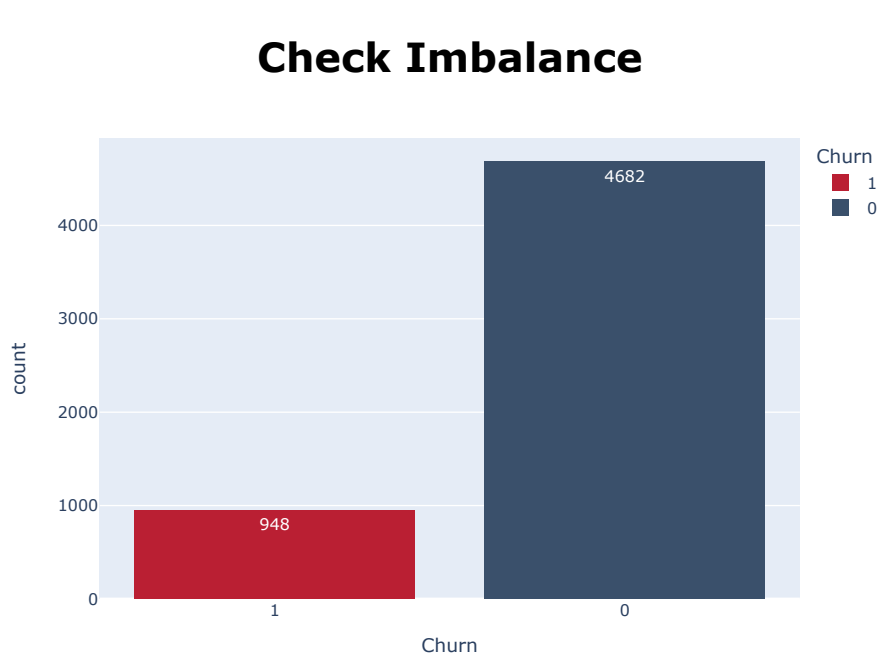
```
CashbackAmount              -0.165008
Tenure                      -0.336058
Name: Churn, dtype: float64
```

```
plt.figure(figsize = (10,10))
sns.barplot(x = churn_corr_vector , y = churn_corr_vector.index , palette = 'coolwarm')
plt.title('Relation Between Features and target')
```

```
Text(0.5, 1.0, 'Relation Between Features and target')
```




```
fig = px.histogram(df2, x="Churn", color="Churn" ,text_auto= True , title="<b>"+'Check Imbalance' , color_discrete_s

# Customize the plot
fig.update_layout(hovermode='x',title_font_size=30)
fig.update_layout(
title_font_color="black",
template="plotly",
title_font_size=30,
hoverlabel_font_size=20,
title_x=0.5,
xaxis_title='Churn',
yaxis_title='count',
)
fig.show()
```



Our Data Imbalanced so lets make Over sample for it Using SMOTETomek

Handling Imbalanced Data

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

```
from imblearn.combine import SMOTETomek
```

```
smt = SMOTETomek(random_state=42)
x_over , y_over = smt.fit_resample(X , Y)
```

```
x_over.shape, y_over.shape
```

```
((8582, 18), (8582,))
```

Split Data

```
x_train , x_test , y_train , y_test = train_test_split(x_over , y_over , test_size = 0.30 , random_state = 42)
```

```
# Now we  will make normalization for all data to make them in commom range
from sklearn.preprocessing import MinMaxScaler , StandardScaler , RobustScaler

MN = MinMaxScaler()
# SC = StandardScaler()
# Rb = RobustScaler()
x_train_scaled = MN.fit_transform(x_train)
x_test_scaled = MN.fit_transform(x_test)
```

Modeling

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
import warnings

warnings.filterwarnings("ignore")


logisreg_clf = LogisticRegression()
svm_clf = SVC()
dt_clf = DecisionTreeClassifier()
rf_clf = RandomForestClassifier()
XGB_clf = XGBClassifier()
ada_clf = AdaBoostClassifier()


clf_list = [logisreg_clf, svm_clf, dt_clf, rf_clf, XGB_clf, ada_clf]
clf_name_list = ['Logistic Regression', 'Support Vector Machine', 'Decision Tree', 'Random Forest', 'XGBClassifier'

for clf in clf_list:
    clf.fit(x_train_scaled,y_train)


train_acc_list = []
test_acc_list = []

for clf,name in zip(clf_list,clf_name_list):
    y_pred_train = clf.predict(x_train_scaled)
    y_pred_test = clf.predict(x_test_scaled)
    print(f'Using model: {name}')
    print(f'Trainning Score: {clf.score(x_train_scaled, y_train)}')
    print(f'Test Score: {clf.score(x_test_scaled, y_test)}')
    print(f'Acc Train: {accuracy_score(y_train, y_pred_train)}')
    print(f'Acc Test: {accuracy_score(y_test, y_pred_test)}')
    train_acc_list.append(accuracy_score(y_train, y_pred_train))
    test_acc_list.append(accuracy_score(y_test, y_pred_test))
    print(' ' * 60)
    print('*' * 60)
    print(' ' * 60)
```

```
 Using model: Logistic Regression
 Trainning Score: 0.7686032961544864
 Test Score: 0.7728155339805826
 Acc Train: 0.7686032961544864
 Acc Test: 0.7728155339805826

 ************************************************************

 Using model: Support Vector Machine
 Trainning Score: 0.9042783419344098
 Test Score: 0.876116504854369
 Acc Train: 0.9042783419344098
 Acc Test: 0.876116504854369

 ************************************************************

 Using model: Decision Tree
 Trainning Score: 1.0
 Test Score: 0.9312621359223301
 Acc Train: 1.0
 Acc Test: 0.9312621359223301

 ************************************************************

 Using model: Random Forest
 Trainning Score: 1.0
 Test Score: 0.9697087378640776
 Acc Train: 1.0
 Acc Test: 0.9697087378640776

 ************************************************************

 Using model: XGBClassifier
 Trainning Score: 1.0
 Test Score: 0.9545631067961166
 Acc Train: 1.0
 Acc Test: 0.9545631067961166

 ************************************************************

 Using model: AdaBoostClassifier
 Trainning Score: 0.874812718495089
 Test Score: 0.8454368932038835
```

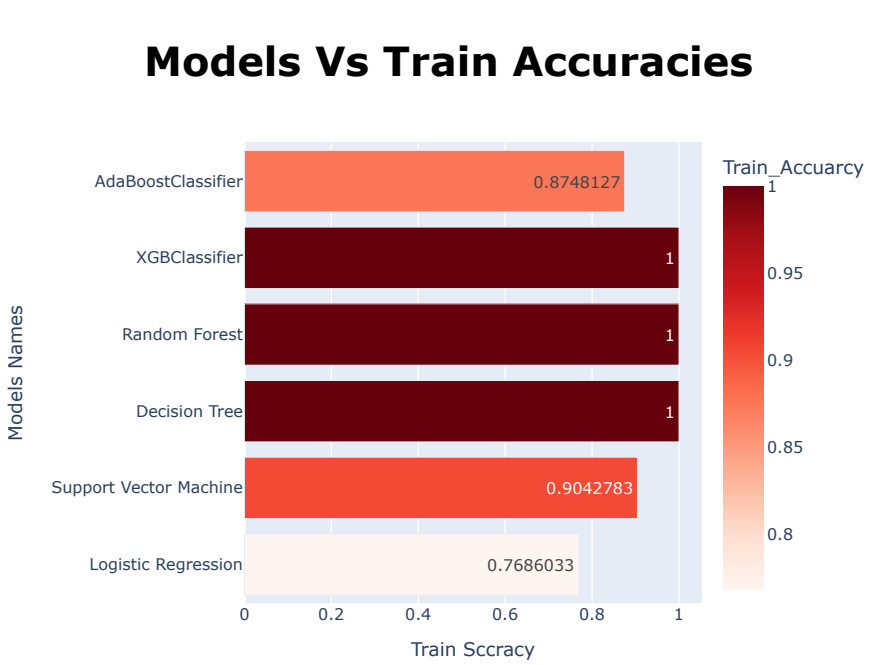
```
Acc Train: 0.874812718495089
Acc Test: 0.8454368932038835

**********************************************************
```

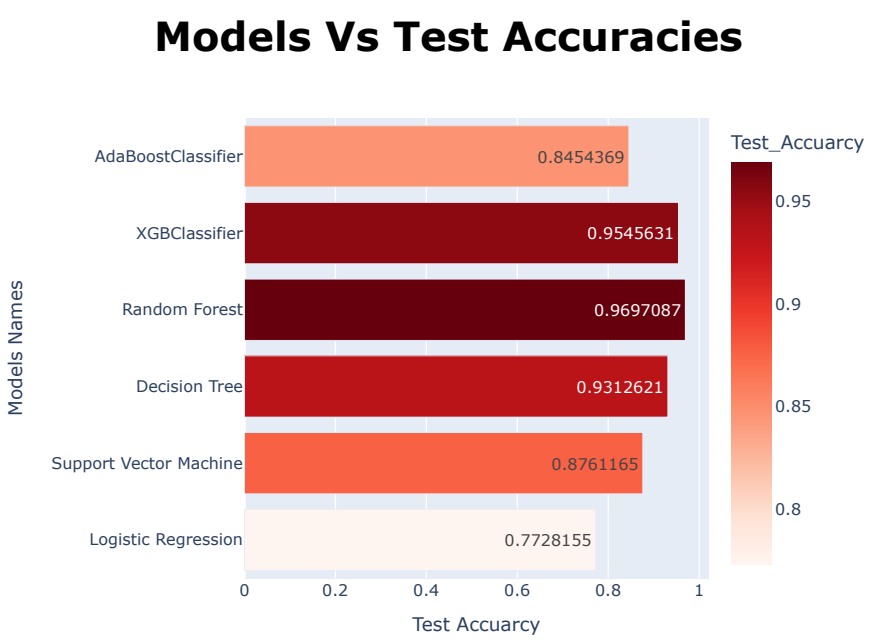
```
# graph to determine best 2 models

all_models = pd.DataFrame({'Train_Accuarcy': train_acc_list , 'Test_Accuarcy' : test_acc_list}  , index = clf_name_li
all_models
```

	Train_Accuarcy	Test_Accuarcy
Logistic Regression	0.768603	0.772816
Support Vector Machine	0.904278	0.876117
Decision Tree	1.000000	0.931262
Random Forest	1.000000	0.969709
XGBClassifier	1.000000	0.954563
AdaBoostClassifier	0.874813	0.845437

```
# Models vs Train Accuracies
fig = px.bar(all_models, x=all_models['Train_Accuarcy'], y = all_models.index ,color=all_models['Train_Accuarcy'],ti
fig.update_layout(hovermode='x',title_font_size=30)
fig.update_layout(
title_font_color="black",
template="plotly",
title_font_size=30,
hoverlabel_font_size=20,
title_x=0.5,
xaxis_title='Train Sccracy',
yaxis_title='Models Names',
)
fig.show()
```

```
# Models vs Test Accuracies
fig = px.bar(all_models, x=all_models['Test_Accuarcy'], y = all_models.index ,color=all_models['Test_Accuarcy'],titl
fig.update_layout(hovermode='x',title_font_size=30)
fig.update_layout(
title_font_color="black",
template="plotly",
title_font_size=30,
hoverlabel_font_size=20,
title_x=0.5,
xaxis_title='Test Accuarcy',
yaxis_title='Models Names',
)
fig.show()
```





from Graphs Best 2 Models in Train and Test are [Random Forest , XGBoost]

```
!pip install mlxtend
```

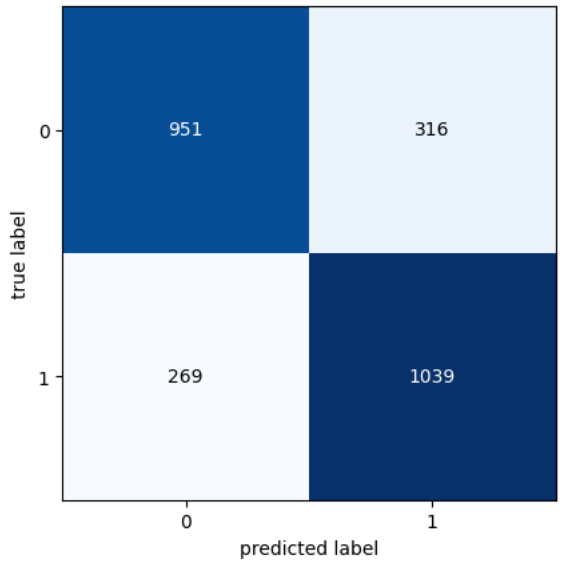
```
Requirement already satisfied: mlxtend in /usr/local/lib/python3.10/dist-packages (0.22.0)
Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.11.4)
Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.23.5)
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.5.3)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.
Requirement already satisfied: matplotlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (3.7.
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.3.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from mlxtend) (67.7.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0-
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->mlx
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-lear
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->m
```

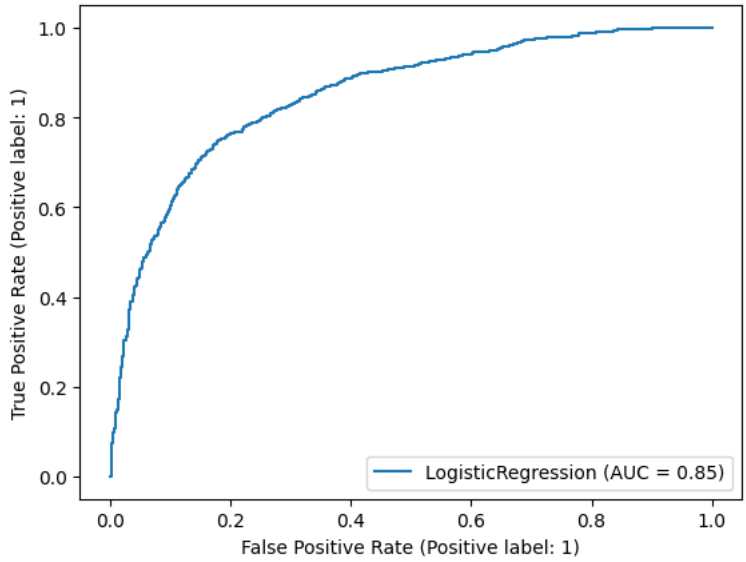
```
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report, RocCurveDisplay
```

Evaluation

```
# Logistic regression
model= LogisticRegression()
model.fit(x_train_scaled,y_train)
y_pred = model.predict(x_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
roc_auc1 = roc_auc_score(y_test, y_pred)
print("Accuracy = {}".format(accuracy))
print("ROC Area under Curve = {}".format(roc_auc1))
print(classification_report(y_test,y_pred,digits=5))
plot_confusion_matrix(confusion_matrix(y_test , y_pred))
print('*' * 70)
RocCurveDisplay.from_estimator(model , x_test_scaled , y_test)
```

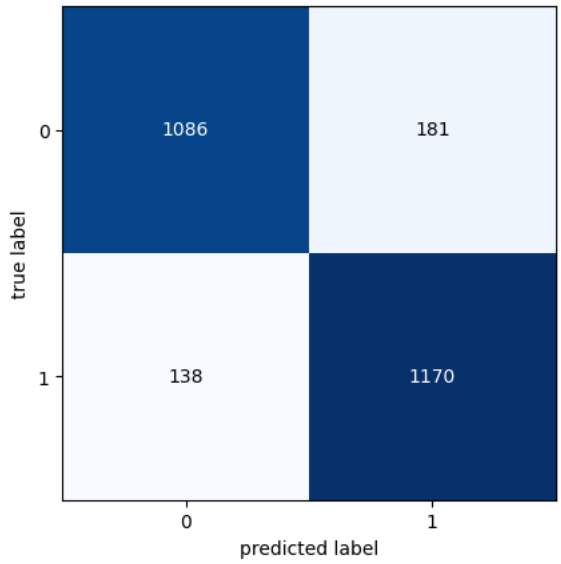
```
Accuracy = 0.7728155339805826
ROC Area under Curve = 0.7724672285661185
              precision    recall  f1-score   support

           0    0.77951   0.75059   0.76478      1267
           1    0.76679   0.79434   0.78032      1308

    accuracy                        0.77282      2575
   macro avg    0.77315   0.77247   0.77255      2575
weighted avg    0.77305   0.77282   0.77267      2575

**********************************************************************
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x79c7150e1ff0>
```

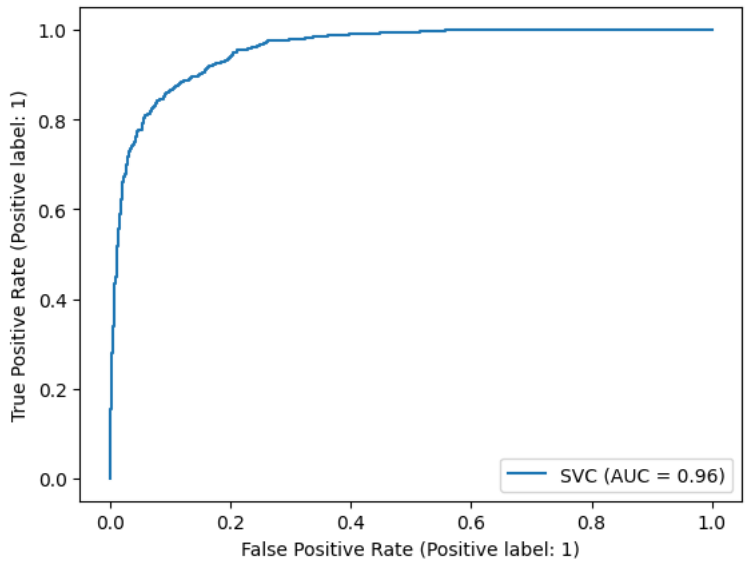




```
# Support Vector Machine
model=SVC()
model.fit(x_train_scaled,y_train)
y_pred = model.predict(x_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
roc_auc2 = roc_auc_score(y_test, y_pred)
print("Accuracy = {}".format(accuracy))
print("ROC Area under Curve = {}".format(roc_auc2))
print(classification_report(y_test,y_pred,digits=5))
plot_confusion_matrix(confusion_matrix(y_test , y_pred))
RocCurveDisplay.from_estimator(model , x_test_scaled , y_test)
```

```
Accuracy = 0.876116504854369
ROC Area under Curve = 0.875819134993447
              precision    recall  f1-score   support

           0    0.88725   0.85714   0.87194      1267
           1    0.86603   0.89450   0.88003      1308

    accuracy                        0.87612      2575
   macro avg    0.87664   0.87582   0.87598      2575
weighted avg    0.87647   0.87612   0.87605      2575

<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x79c71821d4e0>
```






```
# Decision Tree
model=DecisionTreeClassifier()
model.fit(x_train_scaled,y_train)
y_pred = model.predict(x_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
roc_auc3 = roc_auc_score(y_test, y_pred)
print("Accuracy = {}".format(accuracy))
print("ROC Area under Curve = {}".format(roc_auc3))
print(classification_report(y_test,y_pred,digits=5))
plot_confusion_matrix(confusion_matrix(y_test , y_pred))
RocCurveDisplay.from_estimator(model , x_test_scaled , y_test)
```