In [1]:
```
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_
from sklearn.model_selection import GridSearchCV, cross_validate

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

data overview

In [2]:
```
# Step 1: Data Loading and Understanding

df = pd.read_excel('/kaggle/input/ecommerce-customer-churn-analysis-and-predic
df.head()
```

Out[2]:

	CustomerID	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPayn
0	50001	1	4.0	Mobile Phone	3	6.0	I
1	50002	1	NaN	Phone	1	8.0	
2	50003	1	NaN	Phone	1	30.0	I
3	50004	1	0.0	Phone	3	15.0	I
4	50005	1	0.0	Phone	1	12.0	

In [3]:
```
df.shape
```

Out[3]: (5630, 20)

In [4]:
```
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
```

In [5]:
```
df.nunique()
```

Out[5]:
```
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
```

In [6]:
```
# colums to list
columns = df.columns.to_list()
columns
```

Out[6]:
```
['CustomerID',
 'Churn',
 'Tenure',
 'PreferredLoginDevice',
 'CityTier',
 'WarehouseToHome',
 'PreferredPaymentMode',
 'Gender',
 'HourSpendOnApp',
 'NumberOfDeviceRegistered',
 'PreferedOrderCat',
 'SatisfactionScore',
 'MaritalStatus',
 'NumberOfAddress',
 'Complain',
 'OrderAmountHikeFromlastYear',
 'CouponUsed',
 'OrderCount',
 'DaySinceLastOrder',
 'CashbackAmount']
```

In [7]:
```
df.select_dtypes(exclude=np.number).columns
```

Out[7]:
```
Index(['PreferredLoginDevice', 'PreferredPaymentMode', 'Gender',
       'PreferedOrderCat', 'MaritalStatus'],
      dtype='object')
```

In [8]:
```
df.describe(include='O').style.background_gradient(axis=None , cmap = "Blues"
```

Out[8]:

	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

In [9]:
```
# 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']
PreferredLoginDevice
Mobile Phone    2765
Computer        1634
Phone           1231
Name: count, dtype: int64
_____________________________________________________________________________
___
PreferredPaymentMode : ['Debit Card' 'UPI' 'CC' 'Cash on Delivery' 'E wallet'
'COD' 'Credit Card']
PreferredPaymentMode
Debit Card          2314
Credit Card         1501
E wallet             614
UPI                  414
COD                  365
CC                   273
Cash on Delivery     149
Name: count, dtype: int64
_____________________________________________________________________________
___
Gender : ['Female' 'Male']
Gender
Male      3384
Female    2246
Name: count, dtype: int64
_____________________________________________________________________________
___
PreferedOrderCat : ['Laptop & Accessory' 'Mobile' 'Mobile Phone' 'Others' 'Fa
shion' 'Grocery']
PreferedOrderCat
Laptop & Accessory    2050
Mobile Phone          1271
Fashion                826
Mobile                 809
Grocery                410
Others                 264
Name: count, dtype: int64
_____________________________________________________________________________
___
MaritalStatus : ['Single' 'Divorced' 'Married']
MaritalStatus
Married     2986
Single      1796
Divorced     848
Name: count, dtype: int64
_____________________________________________________________________________
___
```

In [10]:
```
df.select_dtypes(include=np.number).columns
```

Out[10]:
```
Index(['CustomerID', 'Churn', 'Tenure', 'CityTier', 'WarehouseToHome',
       'HourSpendOnApp', 'NumberOfDeviceRegistered', 'SatisfactionScore',
       'NumberOfAddress', 'Complain', 'OrderAmountHikeFromlastYear',
       'CouponUsed', 'OrderCount', 'DaySinceLastOrder', 'CashbackAmount'],
      dtype='object')
```

In [11]:
```
df.describe().T.style.bar(subset=['mean']).background_gradient(subset=['std','
```

Out[11]:

	count	mean	std	min	2
CustomerID	5630.000000	52815.500000	1625.385339	50001.000000	51408.250(
Churn	5630.000000	0.168384	0.374240	0.000000	0.000(
Tenure	5366.000000	10.189899	8.557241	0.000000	2.000(
CityTier	5630.000000	1.654707	0.915389	1.000000	1.000(
WarehouseToHome	5379.000000	15.639896	8.531475	5.000000	9.000(
HourSpendOnApp	5375.000000	2.931535	0.721926	0.000000	2.000(
NumberOfDeviceRegistered	5630.000000	3.688988	1.023999	1.000000	3.000(
SatisfactionScore	5630.000000	3.066785	1.380194	1.000000	2.000(
NumberOfAddress	5630.000000	4.214032	2.583586	1.000000	2.000(
Complain	5630.000000	0.284902	0.451408	0.000000	0.000(
OrderAmountHikeFromlastYear	5365.000000	15.707922	3.675485	11.000000	13.000(
CouponUsed	5374.000000	1.751023	1.894621	0.000000	1.000(
OrderCount	5372.000000	3.008004	2.939680	1.000000	1.000(
DaySinceLastOrder	5323.000000	4.543491	3.654433	0.000000	2.000(
CashbackAmount	5630.000000	177.223030	49.207036	0.000000	145.770(

In [12]:
```
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("______________________________________________________________________________
```

```
CustomerID : [50001 50002 50003 ... 55628 55629 55630]
CustomerID
50001    1
53751    1
53759    1
53758    1
53757    1
        ..
51876    1
51875    1
51874    1
51873    1
55630    1
Name: count, Length: 5630, dtype: int64
__________________________________________________________________________
______
Churn : [1 0]
Churn
0    4682
```

In [13]:
```
#As mobile phone and phone are both same so we have merged them
df.loc[df['PreferredLoginDevice'] == 'Phone', 'PreferredLoginDevice' ] = 'Mobi
df.loc[df['PreferedOrderCat'] == 'Mobile', 'PreferedOrderCat' ] = 'Mobile Phon
```

In [14]:
```
df['PreferredLoginDevice'].value_counts()
```

Out[14]:
```
PreferredLoginDevice
Mobile Phone    3996
Computer        1634
Name: count, dtype: int64
```

In [15]:
```
#as cod is also cash on delievery
#as cc is also credit card so i merged them
df.loc[df['PreferredPaymentMode'] == 'COD', 'PreferredPaymentMode' ] = 'Cash o
df.loc[df['PreferredPaymentMode'] == 'CC', 'PreferredPaymentMode' ] = 'Credit
```

In [16]:
```
df['PreferredPaymentMode'].value_counts()
```

Out[16]:
```
PreferredPaymentMode
Debit Card          2314
Credit Card         1774
E wallet             614
Cash on Delivery     514
UPI                  414
Name: count, dtype: int64
```

In [17]:
```
# 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
```

Out[17]:
```
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
```

In [18]:
```
# Categorical cols after Converting
df2.describe(include='O').style.background_gradient(axis=None , cmap = "Blues"
```

Out[18]:

	Churn	PreferredLoginDevice	CityTier	PreferredPaymentMode	Gender	NumberOfDeviceF
count	5630	5630	5630	5630	5630	
unique	2	2	3	5	2	
top	0	Mobile Phone	1	Debit Card	Male	
freq	4682	3996	3666	2314	3384	

In [19]:
```
# Numerical cols after Converting
df2.describe().T.style.bar(subset=['mean']).background_gradient(subset=['std',
```

Out[19]:

	count	mean	std	min	2
CustomerID	5630.000000	52815.500000	1625.385339	50001.000000	51408.250(
Tenure	5366.000000	10.189899	8.557241	0.000000	2.000(
WarehouseToHome	5379.000000	15.639896	8.531475	5.000000	9.000(
HourSpendOnApp	5375.000000	2.931535	0.721926	0.000000	2.000(
OrderAmountHikeFromlastYear	5365.000000	15.707922	3.675485	11.000000	13.000(
CouponUsed	5374.000000	1.751023	1.894621	0.000000	1.000(
OrderCount	5372.000000	3.008004	2.939680	1.000000	1.000(
DaySinceLastOrder	5323.000000	4.543491	3.654433	0.000000	2.000(
CashbackAmount	5630.000000	177.223030	49.207036	0.000000	145.770(

In [20]:
```
df.duplicated().sum()
```

Out[20]: 0

In [21]:
```python
# 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', 'perce

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

Out[21]:
```
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
```

In [22]:
```python
from pandas_profiling import ProfileReport
ProfileReport(df)
```

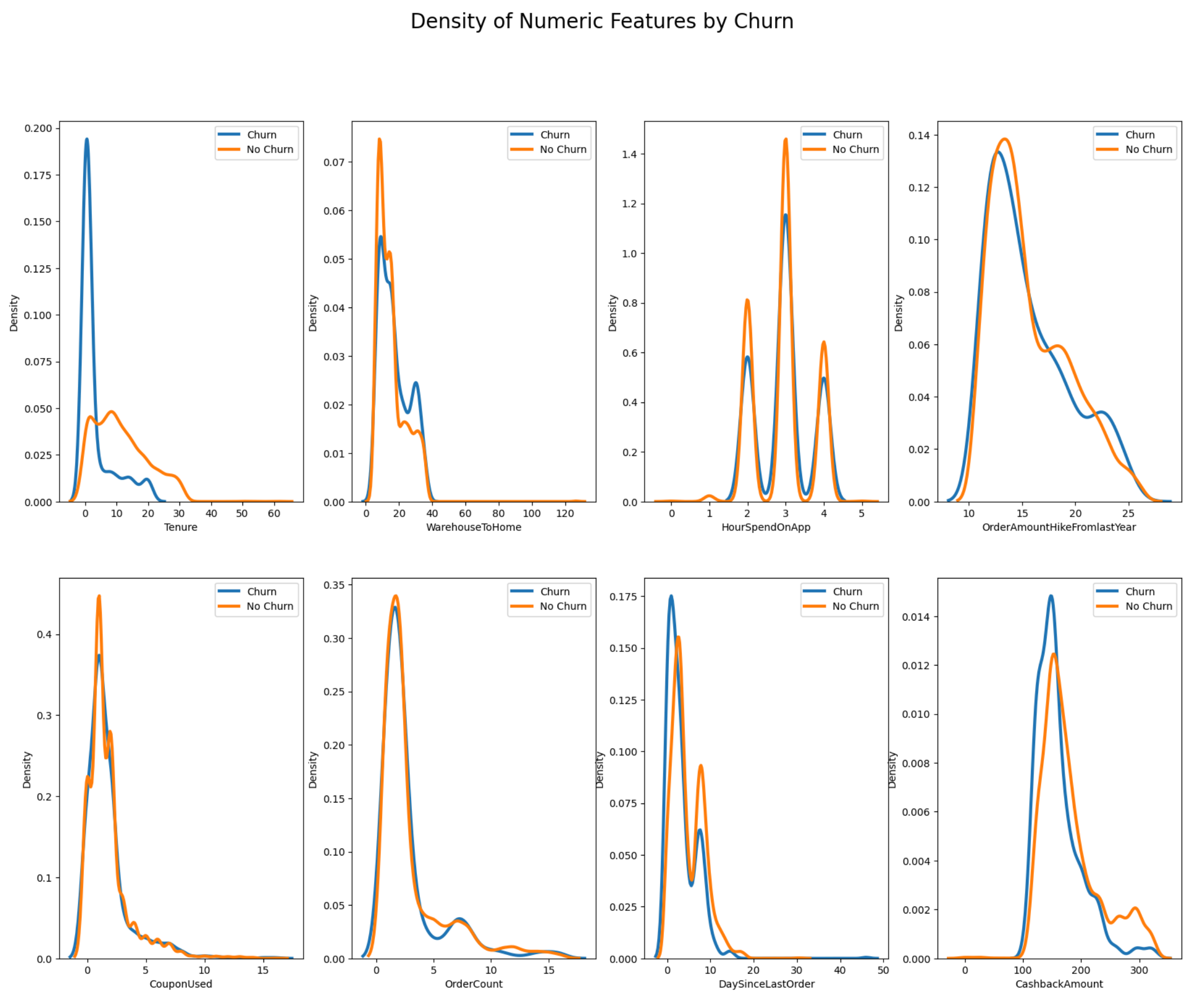
```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Out[22]:

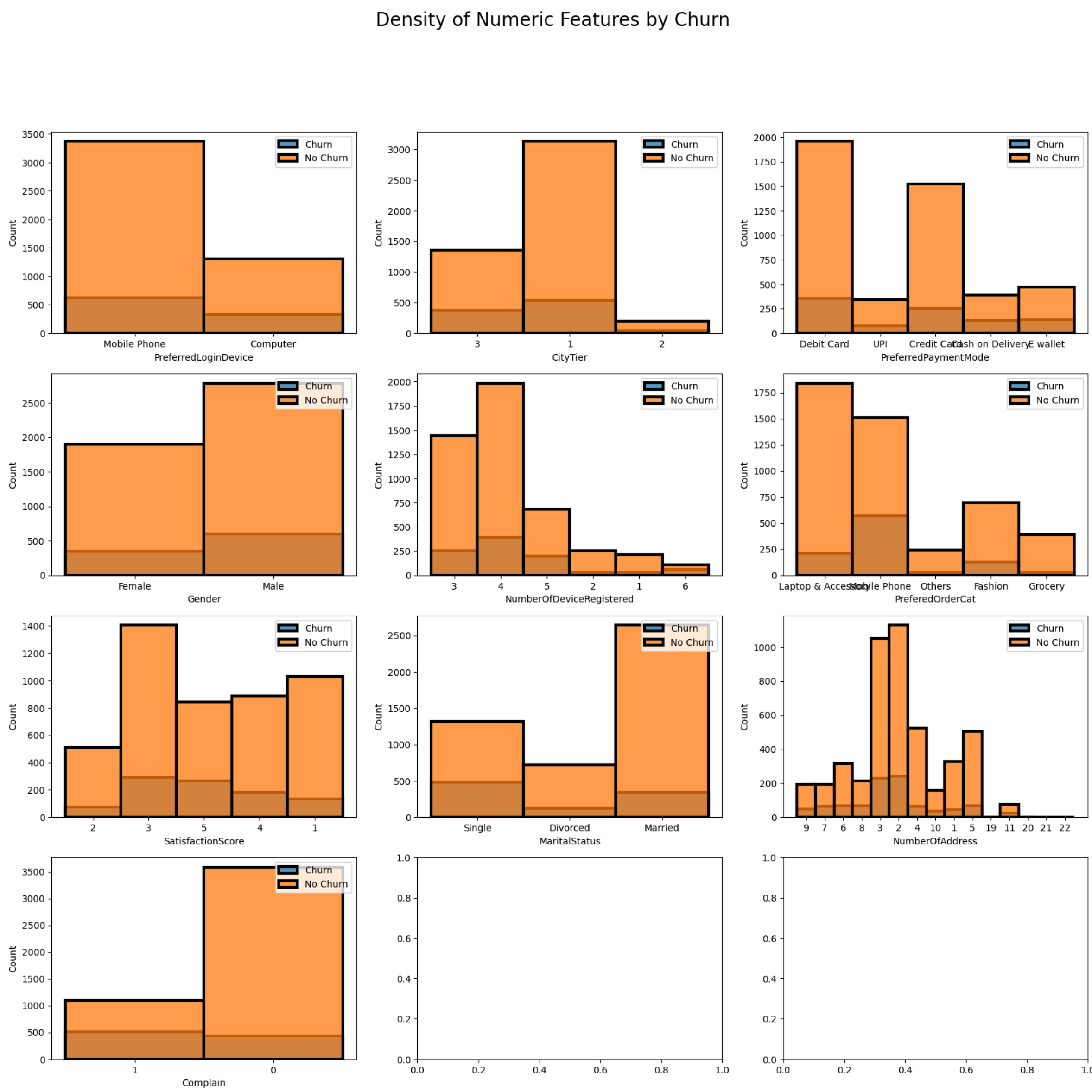
EDA

In [23]:
```
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', 'OrderAmountHikeFro
```

In [24]:
```
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()
```



In [25]:
```
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()
```



In [26]:
```
# 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']
```

relationship between Gender and Churn? & Which Gender has more Orders

In [27]:
```
df['Gender'].value_counts()
```

Out[27]:
```
Gender
Male      3384
Female    2246
Name: count, dtype: int64
```

In [28]:
```
df.groupby("Churn")["Gender"].value_counts() # the churned females ratio 348/2
                                              # the churned males ratio 600/33
```

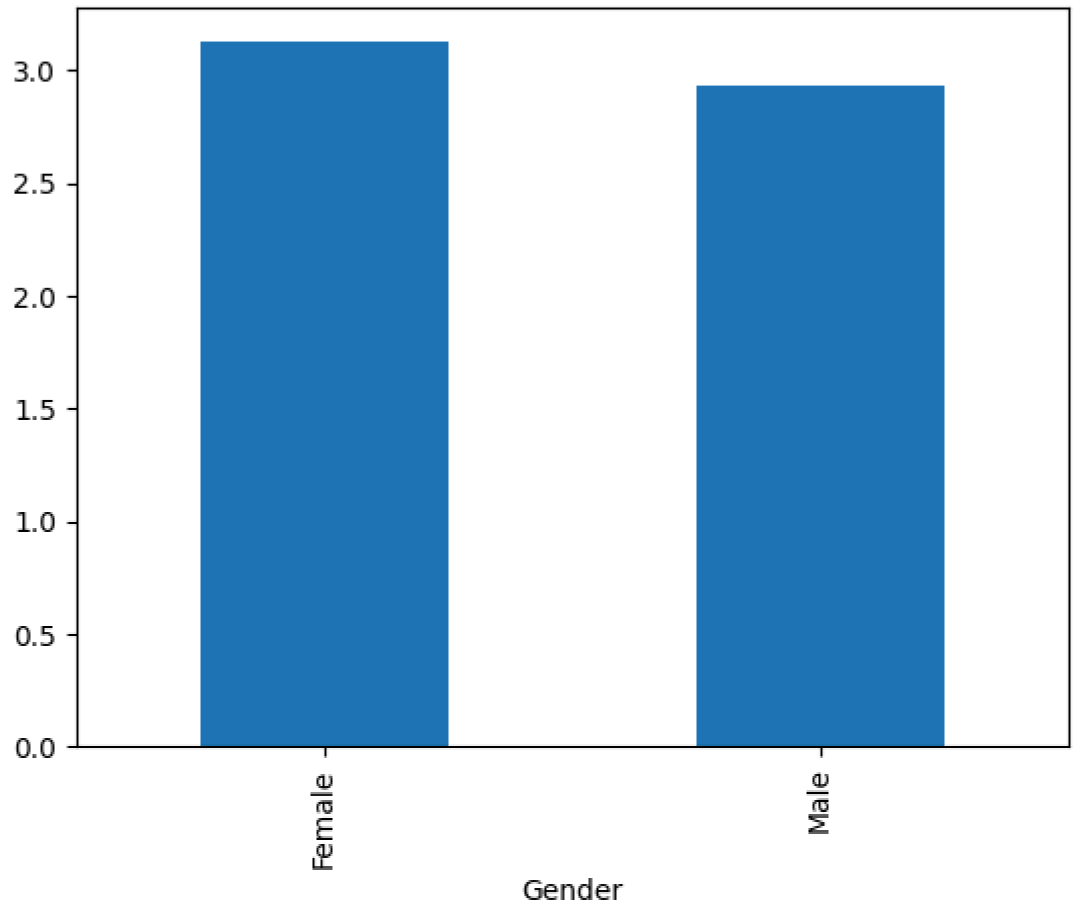
Out[28]:
```
Churn  Gender
0      Male      2784
       Female    1898
1      Male       600
       Female     348
Name: count, dtype: int64
```

In [29]:
```
df.groupby("PreferredLoginDevice")["OrderCount"].value_counts() # the churned
```

Out[29]:
```
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
                      13.0            21
                      10.0            21
                      16.0            19
                      14.0            16
Name: count, dtype: int64
```

In [30]:
```
gender_orders = df.groupby('Gender')['OrderCount'].mean().plot(kind='bar')
gender_orders  # females have more order count avg
```

Out[30]: <Axes: xlabel='Gender'>



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

In [31]:
```
percentageM =600/3384 * 100
percentageM   #the percentage of the leaving males out of the males
```

Out[31]: 17.73049645390071

In [32]:
```
percentageF =348/2246 * 100
percentageF  #the percentage of the leaving females out of the females
```

Out[32]: 15.49421193232413

In [33]:
```
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.

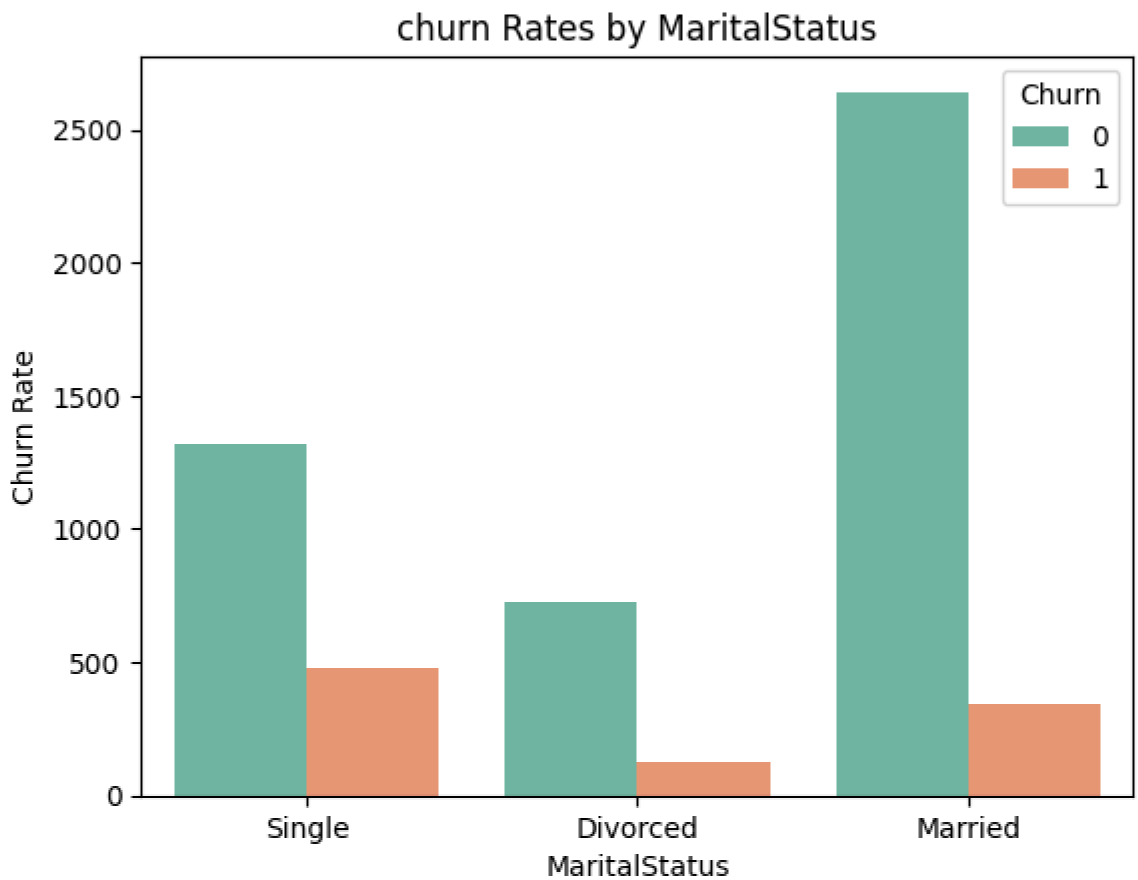
MartialStatus has the highest Churn rate

In [34]:
```
df.groupby("Churn")["MaritalStatus"].value_counts()
```

Out[34]:
```
Churn  MaritalStatus
0      Married          2642
       Single           1316
       Divorced          724
1      Single            480
       Married           344
       Divorced          124
Name: count, dtype: int64
```

In [35]:
```
sns.countplot(x='MaritalStatus',hue='Churn',data=df,palette='Set2')
plt.title("churn Rates by MaritalStatus")
plt.ylabel("Churn Rate")
```

Out[35]: 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

Which CityTier has higher Tenure and OrderCount

In [36]:
```
df_grouped_tenure = df.groupby('CityTier')['Tenure'].agg(['mean', 'max'])
df_grouped_tenure
```

Out[36]:

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

In [37]:
```
df_grouped_OrderCount = df.groupby('CityTier')['OrderCount'].agg(['mean', 'max
df_grouped_OrderCount
```

Out[37]:

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

In [38]:
```
# 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

In [39]:
```
df.groupby("CityTier")["OrderCount"].mean()
```

Out[39]:
```
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

In [40]:
```
df['SatisfactionScore'].dtypes
```

Out[40]:
```
dtype('int64')
```

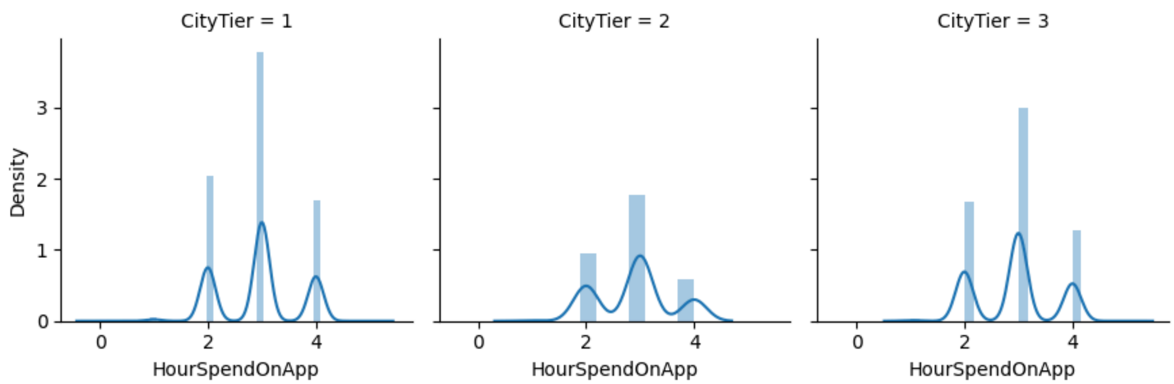
In [41]:
```
import matplotlib.pyplot as plt

# plot
fig = px.histogram(df2, x="HourSpendOnApp", y="SatisfactionScore", orientation

# 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='ba
```

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

Which CityTier has the most HourSpendOnApp

In [42]:
```
g = sns.FacetGrid(df, col='CityTier')
g.map(sns.distplot, 'HourSpendOnApp')
```

Out[42]: <seaborn.axisgrid.FacetGrid at 0x7975df7078e0>



city tier 1 has the most spended hours on the app

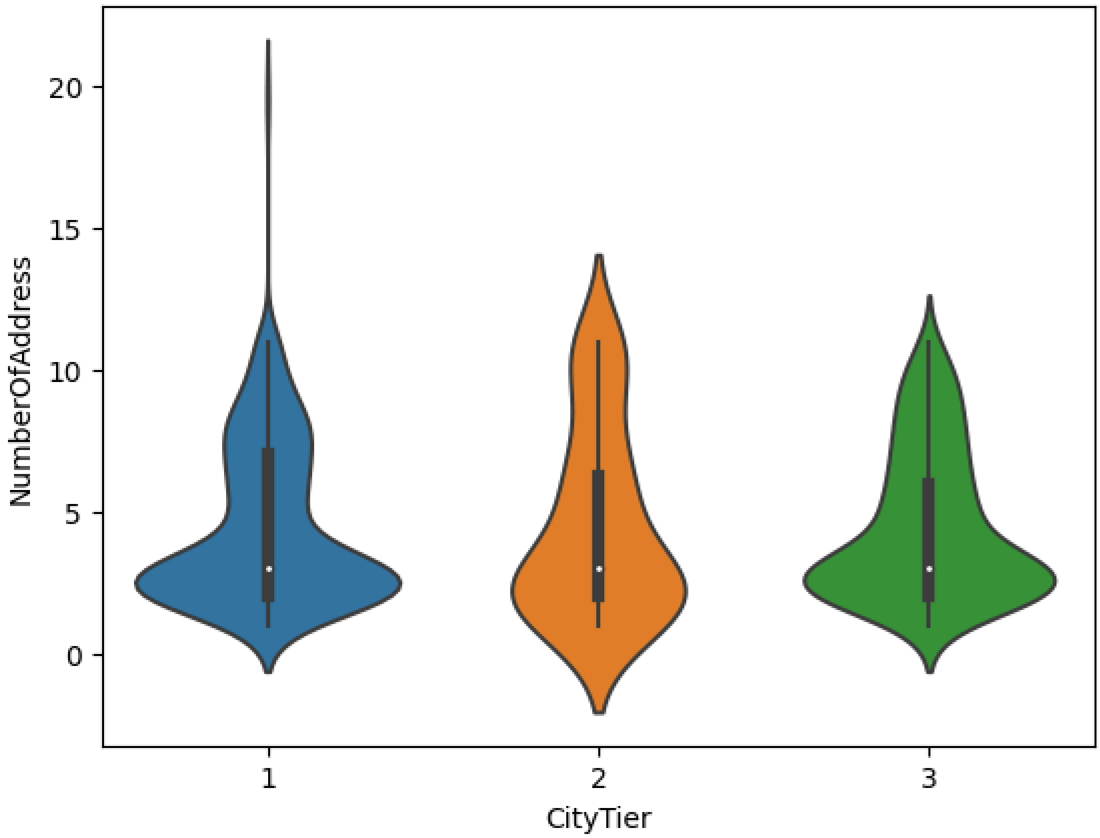
What is the relation between NumberOfAddress and CityTier within the churn segment

In [43]:
```
df.groupby("CityTier")["NumberOfAddress"].value_counts()
```

Out[43]:
```
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
          22                   1
          21                   1
          20                   1
          19                   1
2         2                   61
          3                   43
          5                   30
          1                   23
          6                   21
          4                   16
          10                  13
          8                   10
          7                   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: count, dtype: int64
```

In [44]:
```
# Violin plots
import seaborn as sns
sns.violinplot(x='CityTier', y='NumberOfAddress', data=df[df['Churn']==1])
```

Out[44]: <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?

In [45]:
```
# Pearson correlation
df[['DaySinceLastOrder', 'Complain']].corr()
```

Out[45]:

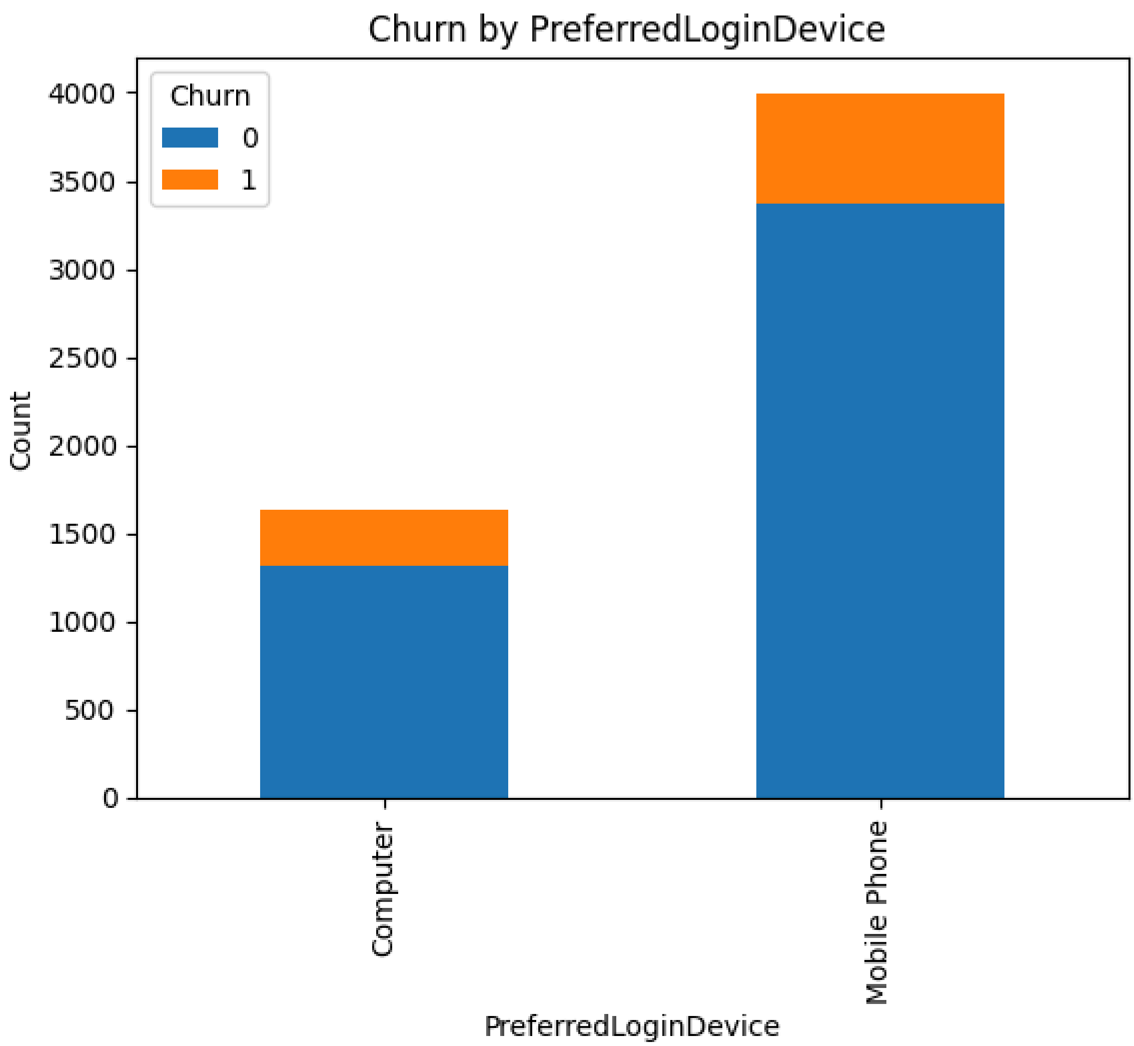
	DaySinceLastOrder	Complain
DaySinceLastOrder	1.000000	-0.043546
Complain	-0.043546	1.000000

In [46]:
```
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?

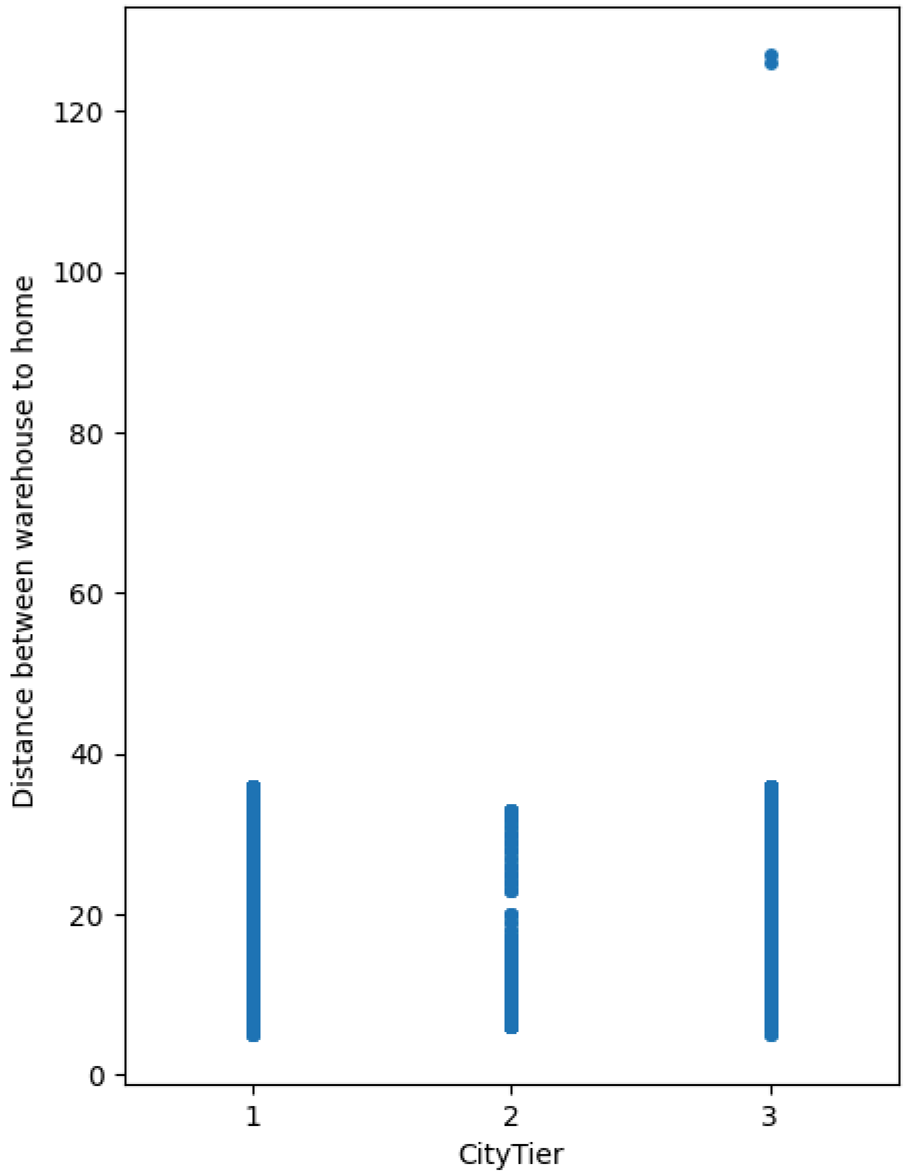
In [47]:
```
# Bar chart with churn rate
import seaborn as sns
# sns.catplot(x='PreferredLoginDevice', y='Churn', data=df, kind='bar')

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

# 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 ?

In [48]:
```
df3 = df.copy()

df3['CityTier'].astype('str')
plt.figure(figsize = (5,7))
sns.stripplot(x = 'CityTier', y = 'WarehouseToHome', data = df3, jitter = Fals
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?

In [49]:

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


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

# 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?

In [50]:
```
df2['PreferredPaymentMode'].value_counts()
```

Out[50]:
```
PreferredPaymentMode
Debit Card          2314
Credit Card         1774
E wallet             614
Cash on Delivery     514
UPI                  414
Name: count, dtype: int64
```

In [51]:
```
df2.groupby('CityTier')[['PreferredPaymentMode']].value_counts()
```

Out[51]:
```
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
Name: count, dtype: int64
```

In [52]:
```
import plotly.express as px

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

# 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?

In [53]:
```
df2.groupby('CityTier')[['OrderCount']].sum()
```

Out[53]:

	OrderCount
CityTier	
1	10298.0
2	615.0
3	5246.0

In [54]:
```
fig = px.histogram(df2, x="OrderCount", y="CityTier", orientation="h", color="

# 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

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

In [55]:
```
df2['OrderAmountHikeFromlastYear'].value_counts()
```

Out[55]:
```
OrderAmountHikeFromlastYear
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: count, dtype: int64
```

In [56]:
```
df2.groupby('OrderAmountHikeFromlastYear')['Churn'].count()
```

Out[56]:
```
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
```

In [57]:
```
comp_ten = df2.groupby(["OrderAmountHikeFromlastYear", "Churn"]).size().reset_

# Create a bubble chart using Plotly
fig_bubble = px.scatter(comp_ten, x="OrderAmountHikeFromlastYear", y="Count",
                        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%

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

In [58]:
```
df_c.groupby('Complain')[['DaySinceLastOrder']].sum()
```

Out[58]:

Complain	DaySinceLastOrder
0	1313.0
1	1580.0

In [59]:
```
fig = px.histogram(df2, x="DaySinceLastOrder", color="Complain",text_auto= Tru
                   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

What is the order counts for customers with high HourSpendOnApp?

In [60]:
```
# we will make binnig for column HourSpendOnApp
df2['HourSpendOnApp'].agg(['min','max'])
```

Out[60]:
```
min    0.0
max    5.0
Name: HourSpendOnApp, dtype: float64
```

In [61]:
```
# 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
```

In [62]:
```
df2.groupby(['HourSpendOnApp_bins','OrderCount'])[['CustomerID']].count()
```

Out[62]:

HourSpendOnApp_bins	OrderCount	CustomerID
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

		CustomerID
HourSpendOnApp_bins	**OrderCount**	
	1.0	1
	2.0	738
	3.0	96
	4.0	34
	5.0	45
	6.0	30
	7.0	25
	8.0	69
high	**9.0**	9
	10.0	15
	11.0	4
	12.0	15
	13.0	6
	14.0	2
	15.0	10
	16.0	10

In [63]:
```
sunbrust_gr = df2.loc[:,['HourSpendOnApp_bins','OrderCount']].dropna()
```

In [64]:
```
fig = px.sunburst(sunbrust_gr,path=['HourSpendOnApp_bins','OrderCount'],title=
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%

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

In [65]:
```
df2.groupby(['PreferedOrderCat' , 'Gender'])[['CustomerID']].count()
```

Out[65]:

PreferedOrderCat	Gender	CustomerID
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

In [66]:
```
# Group and count by 'PreferedOrderCat' and 'Churn'
ordercat_churnrate = pd.DataFrame(df2.groupby('PreferedOrderCat')['Gender'].va
ordercat_churnrate = ordercat_churnrate.rename(columns={'Gender': 'Count'})
ordercat_churnrate = ordercat_churnrate.reset_index()


fig = px.histogram(ordercat_churnrate, x='PreferedOrderCat', y = 'count',color
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()
```

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?

In [67]:
```
df2.groupby(['CouponUsed' , 'Churn'])[['CustomerID']].count()
```

Out[67]:

CouponUsed	Churn	CustomerID
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

In [68]:
```
# Group and count by 'Coup' and 'Churn'
coupoun_churnrate = pd.DataFrame(df2.groupby('CouponUsed')['Churn'].value_coun
coupoun_churnrate = coupoun_churnrate.rename(columns={'Churn': 'Count'})
coupoun_churnrate = coupoun_churnrate.reset_index()


fig = px.bar(coupoun_churnrate, x='CouponUsed', y = 'count',color='Churn', bar
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?

In [69]:
```
df2.groupby('SatisfactionScore')[['OrderCount']].count()
```

Out[69]:

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

In [70]:
```
fig = px.box(df2, y="OrderCount", x='SatisfactionScore', color="SatisfactionSc
             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?

In [71]:
```
df_c.groupby(['OrderCount','CashbackAmount'])[['Churn']].count()
```

Out[71]:

OrderCount	CashbackAmount	Churn
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

In [72]:
```
# fig = px.density_contour(df2, x="HourSpendOnApp", y="OrderCount", color = 'c
#                           title="<b>"+'HourSpendOnApp Vs OrderCount within ch
#                            color_discrete_sequence=["#d62728", "#1f77b4"]
#                          )
fig = px.histogram(df2, x='CashbackAmount', y='OrderCount' ,color = 'Churn', t

# 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()
```

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?

In [73]:
```
df2.groupby('Complain')[['Churn']].count()
```

Out[73]:

Complain	Churn
0	4026
1	1604

In [74]:

```
comp_churn = pd.DataFrame(df2.groupby('Complain')['Churn'].value_counts())
comp_churn = comp_churn.rename(columns={'Churn': 'Count'})
comp_churn = comp_churn.reset_index()
print(comp_churn)

comp_churn['Complain'].replace('0' , 'No Complain' , inplace = True)
comp_churn['Complain'].replace('1' , 'Complain' , inplace = True)
comp_churn['Churn'].replace('0' , 'No Churn' , inplace = True)
comp_churn['Churn'].replace('1' , 'Churn' , inplace = True)
print(comp_churn)

# Tree map
fig = px.treemap(comp_churn, path=[px.Constant("all"), 'Complain', 'Churn'], v
fig.update_traces(textinfo="label+percent parent+value" ,root_color="lightgrey
fig.update_layout(margin = dict(t=70, l=25, r=25, b=25))

# red_palette = ['#410B13', '#CD5D67', '#BA1F33', '#421820', '#91171F']
# 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 Churn  count
0        0     0   3586
1        0     1    440
2        1     0   1096
3        1     1    508
      Complain     Churn  count
0  No Complain  No Churn   3586
1  No Complain     Churn    440
2     Complain  No Churn   1096
3     Complain     Churn    508
```

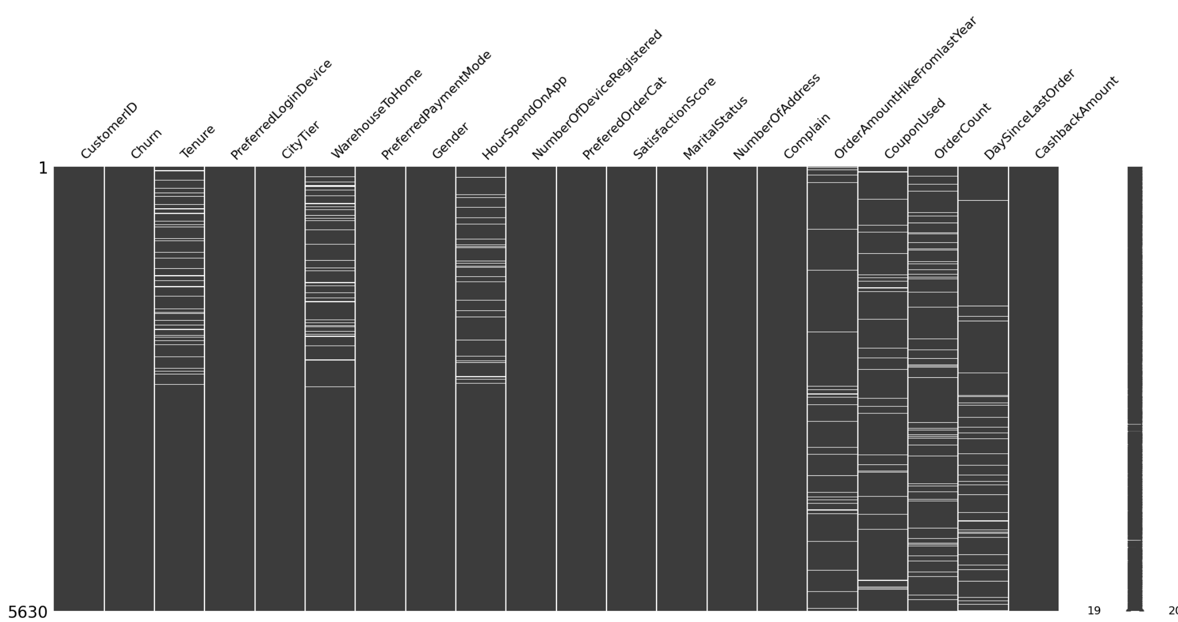
Missing Values

In [75]:
```
round((df.isnull().sum()*100 / df.shape[0]),2)
```

Out[75]:
```
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
```

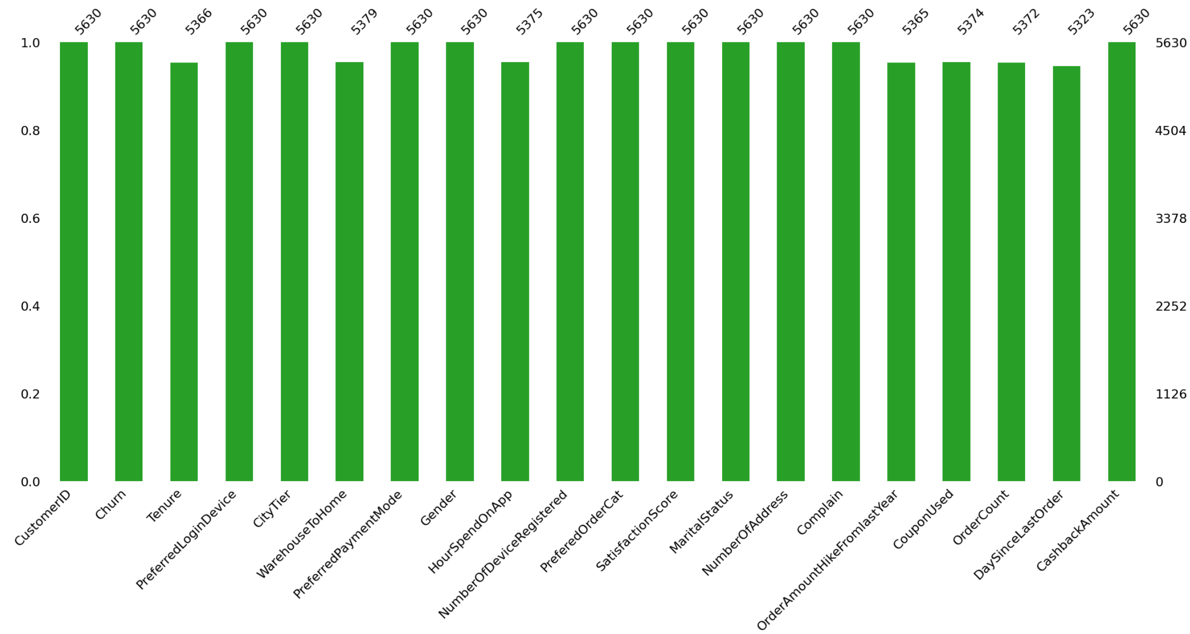
In [76]:
```
msno.matrix(df)
```

Out[76]:
```
<Axes: >
```



In [77]:
```
msno.bar(df , color="tab:green")
```

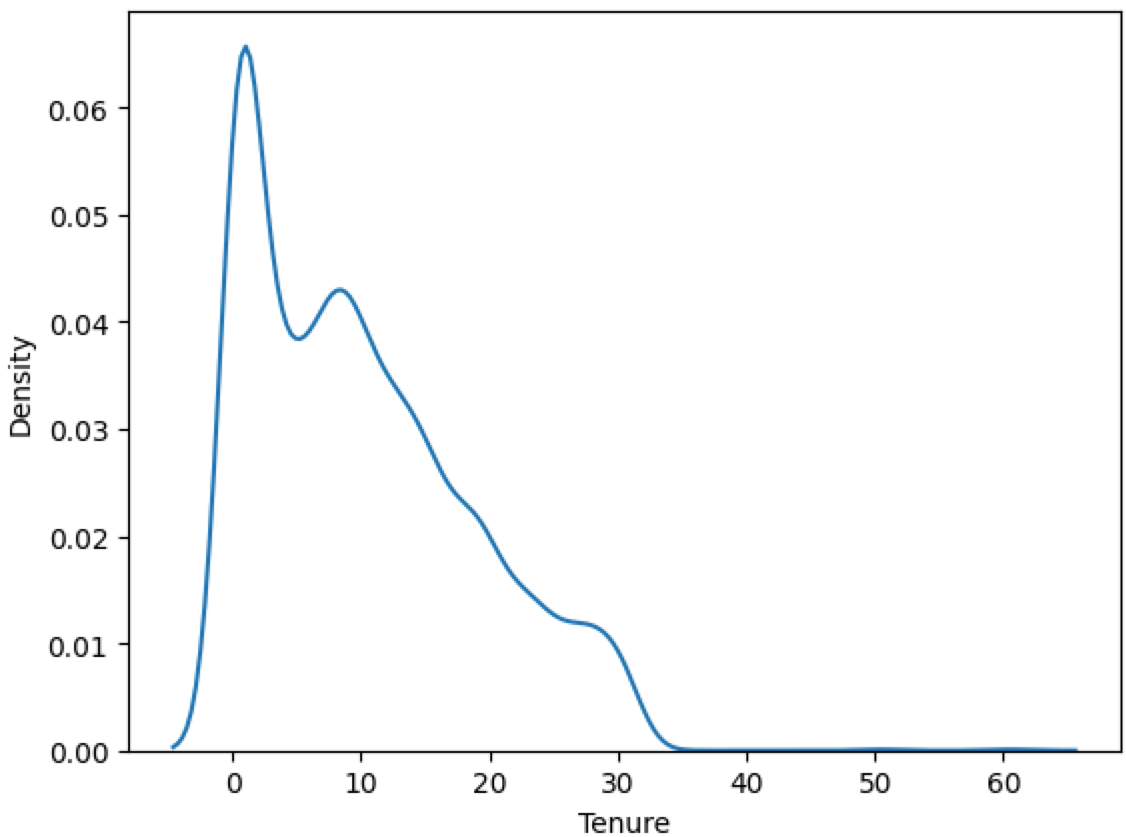
Out[77]: <Axes: >



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

In [78]:
```
sns.kdeplot(df , x='Tenure')
```

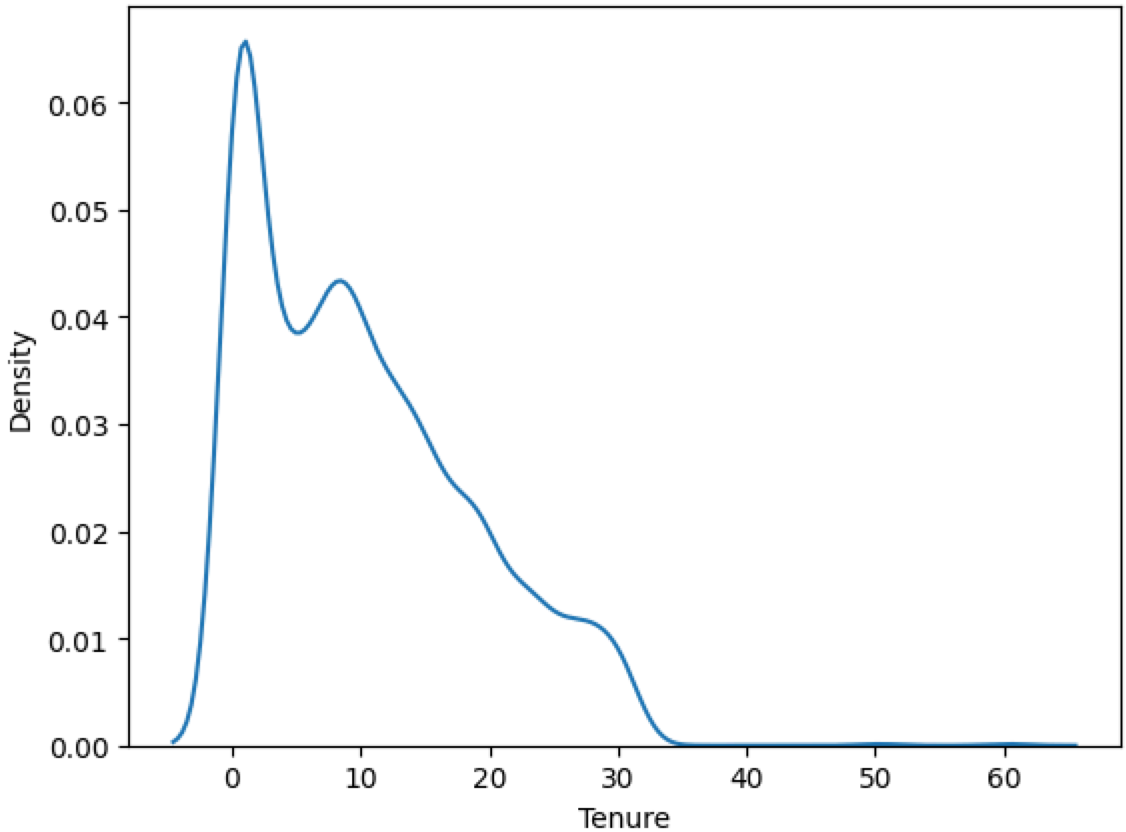
Out[78]: <Axes: xlabel='Tenure', ylabel='Density'>



In [79]:
```
# impute with bfill Method
df['Tenure'] = df['Tenure'].fillna(method = 'bfill')
```

In [80]:
```
sns.kdeplot(df , x='Tenure')
```

Out[80]: <Axes: xlabel='Tenure', ylabel='Density'>

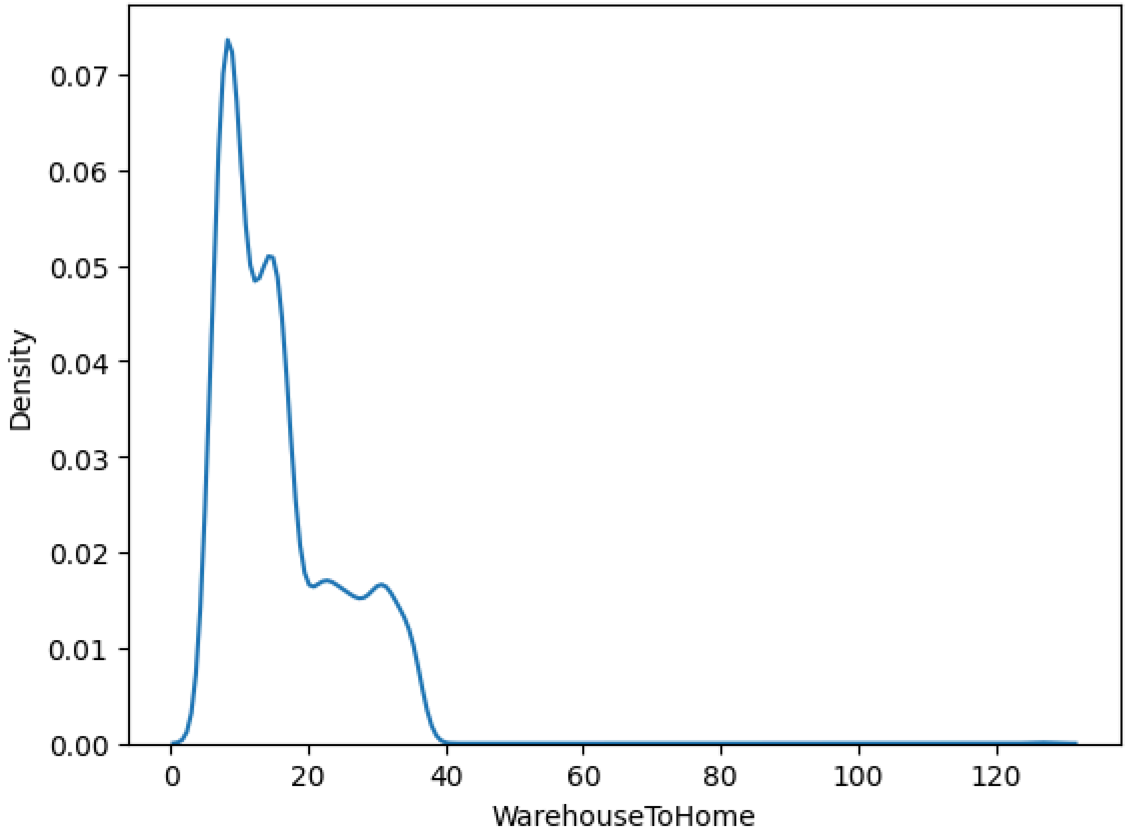


In [81]:
```
df['Tenure'].isnull().sum()
```

Out[81]: 0

In [82]:
```
sns.kdeplot(df , x='WarehouseToHome')
```

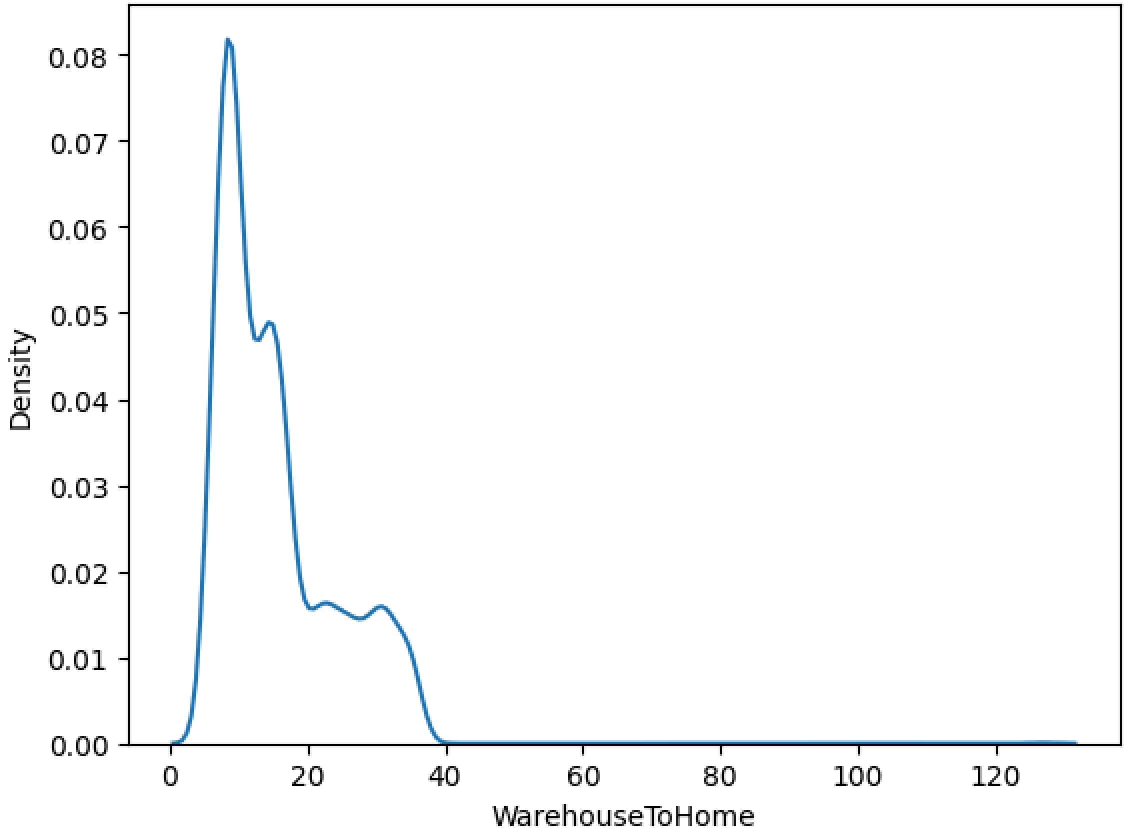
Out[82]:
```
<Axes: xlabel='WarehouseToHome', ylabel='Density'>
```



In [83]:
```
# 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']
```

In [84]:
```
sns.kdeplot(df , x='WarehouseToHome')
```

Out[84]:
```
<Axes: xlabel='WarehouseToHome', ylabel='Density'>
```

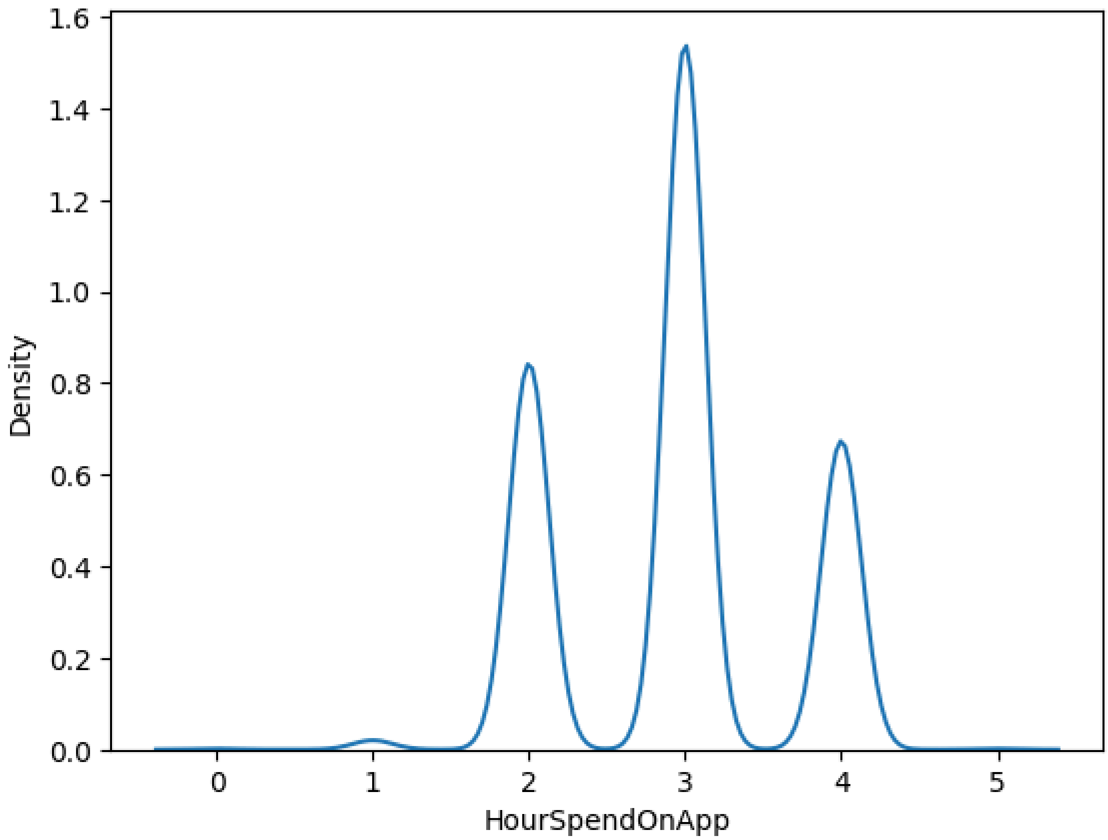


In [85]:
```
df['WarehouseToHome'].isnull().sum()
```

Out[85]:
```
0
```

In [86]:
```
sns.kdeplot(df , x='HourSpendOnApp')
```

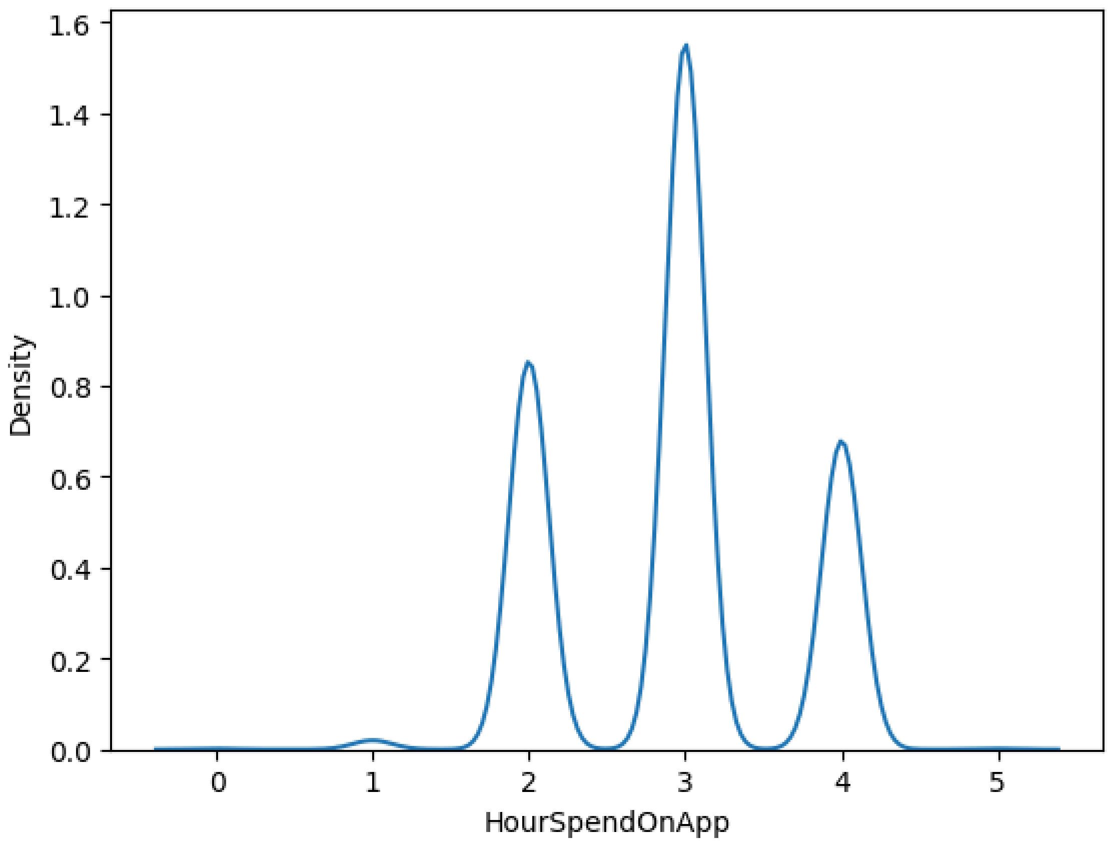
Out[86]: <Axes: xlabel='HourSpendOnApp', ylabel='Density'>



In [87]:
```
fill_list = df['HourSpendOnApp'].dropna()
df['HourSpendOnApp'] = df['HourSpendOnApp'].fillna(pd.Series(np.random.choice(
```

In [88]:
```
sns.kdeplot(df , x='HourSpendOnApp')
```

Out[88]: <Axes: xlabel='HourSpendOnApp', ylabel='Density'>

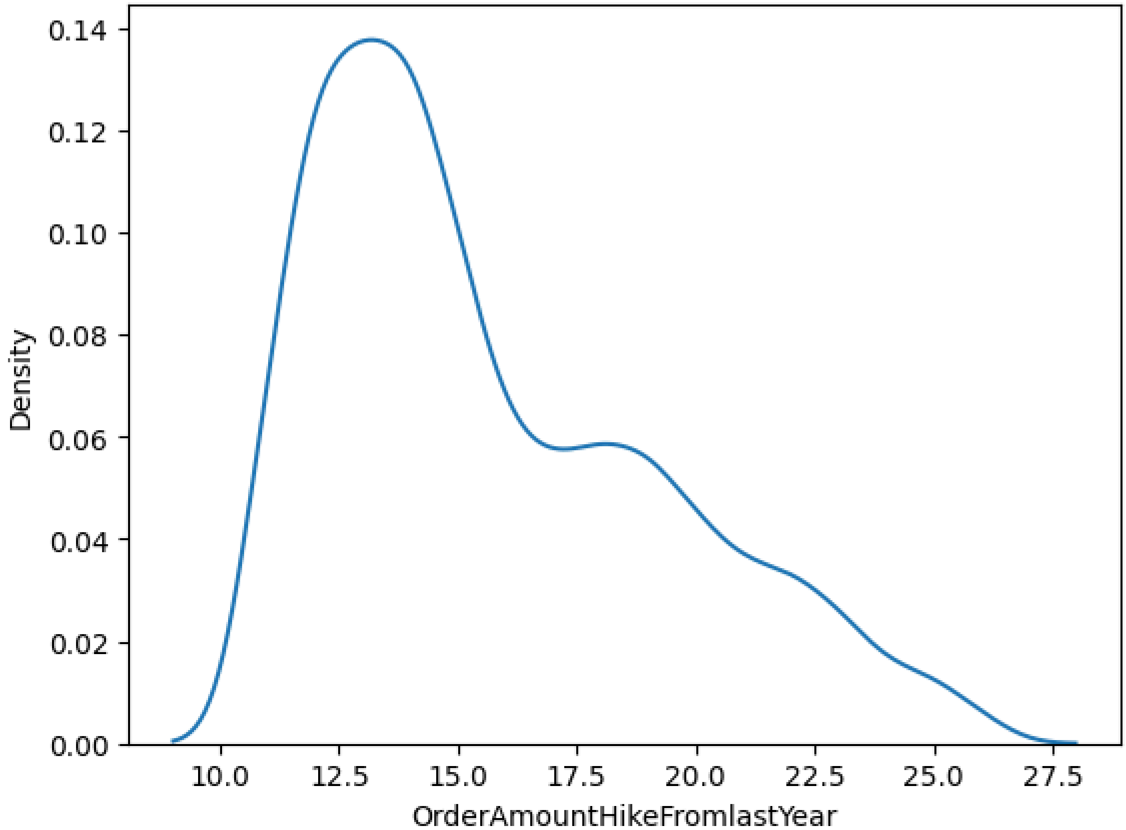


In [89]:
```
df['HourSpendOnApp'].isnull().sum()
```

Out[89]: 0

In [90]:
```
sns.kdeplot(df , x='OrderAmountHikeFromlastYear')
```

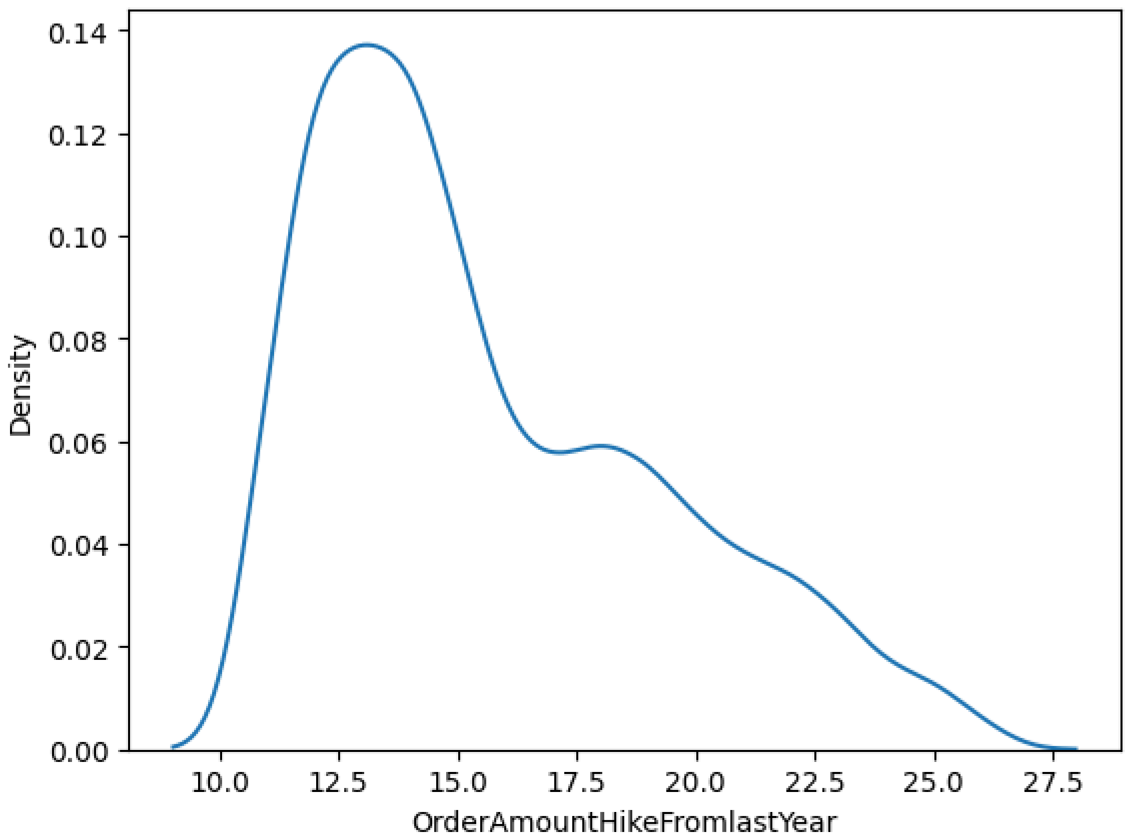
Out[90]:
```
<Axes: xlabel='OrderAmountHikeFromlastYear', ylabel='Density'>
```



In [91]:
```
# impute with ffill method
df['OrderAmountHikeFromlastYear'] = df['OrderAmountHikeFromlastYear'].fillna(m
```

In [92]:
```
sns.kdeplot(df , x='OrderAmountHikeFromlastYear')
```

Out[92]:
```
<Axes: xlabel='OrderAmountHikeFromlastYear', ylabel='Density'>
```

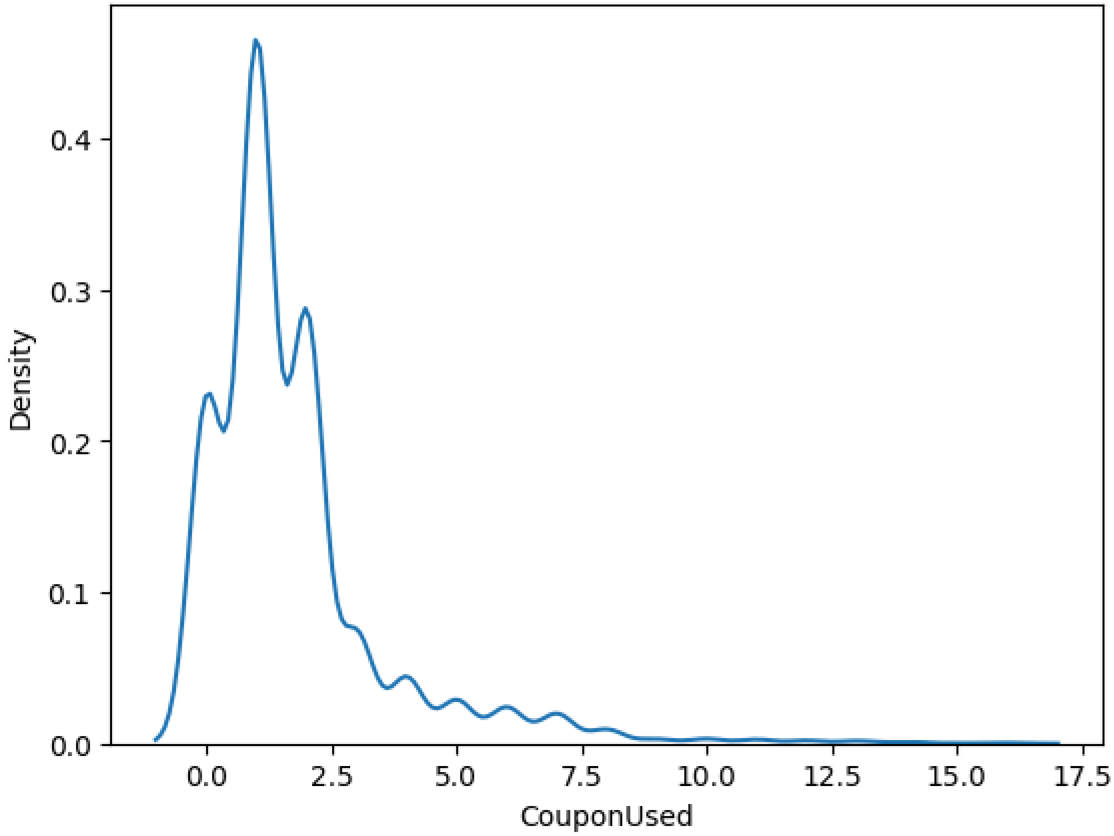


In [93]:
```
df['OrderAmountHikeFromlastYear'].isnull().sum()
```

Out[93]:
```
0
```

In [94]:
```
sns.kdeplot(df , x='CouponUsed')
```

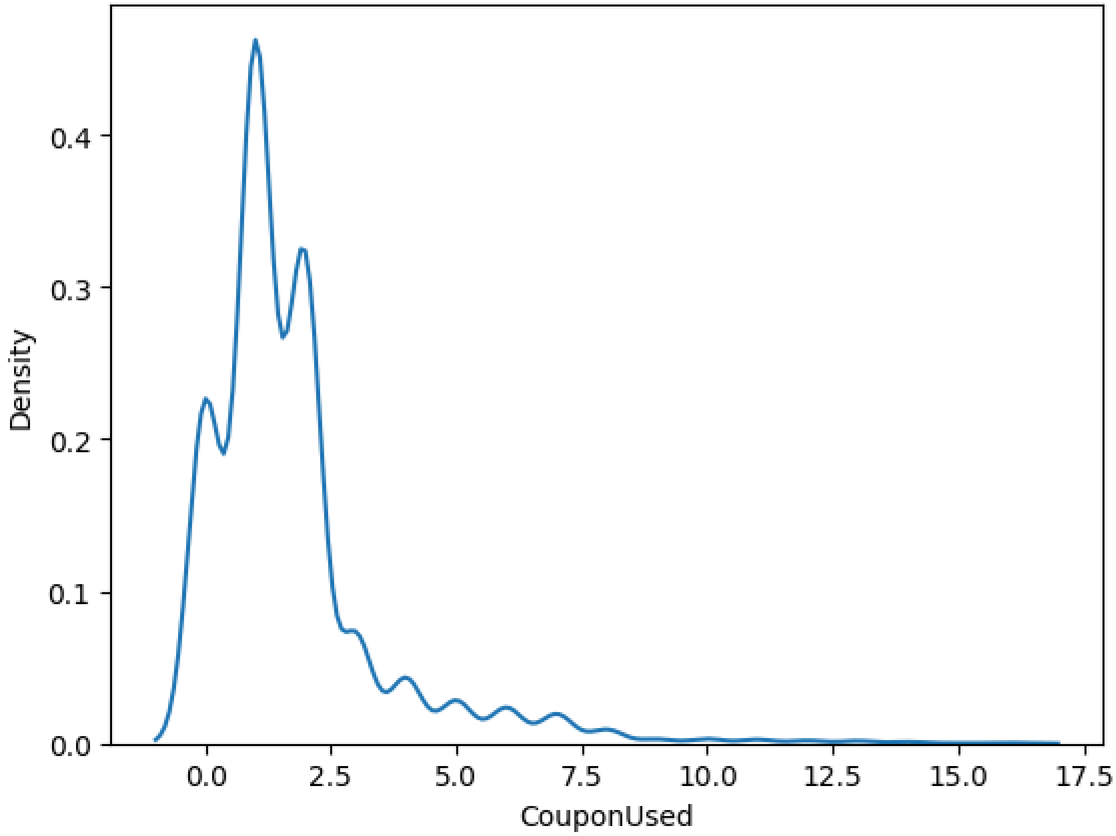
Out[94]: <Axes: xlabel='CouponUsed', ylabel='Density'>



In [95]:
```
# Impute with KNN Imputer
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=2)
df['CouponUsed']=imputer.fit_transform(df[['CouponUsed']])
```

In [96]:
```
sns.kdeplot(df , x='CouponUsed')
```

Out[96]: <Axes: xlabel='CouponUsed', ylabel='Density'>

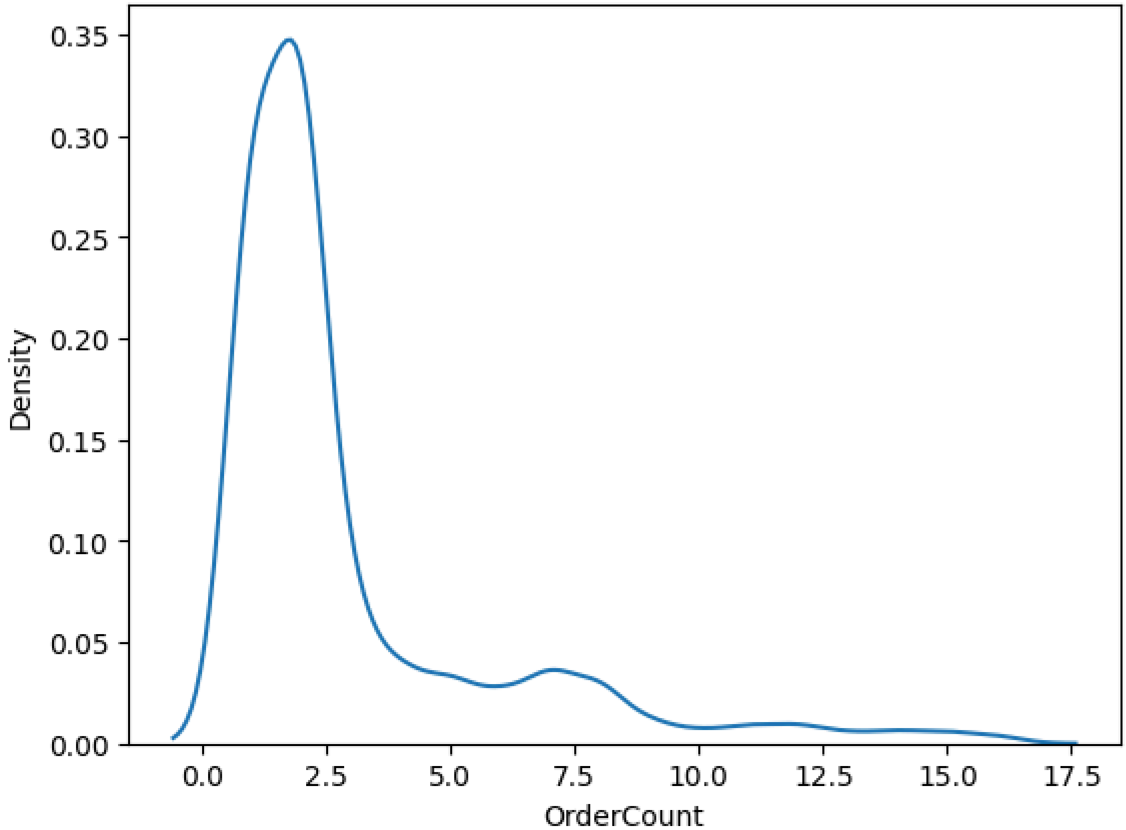


In [97]:
```
df['CouponUsed'].isnull().sum()
```

Out[97]: 0

In [98]:
```
sns.kdeplot(df , x='OrderCount')
```

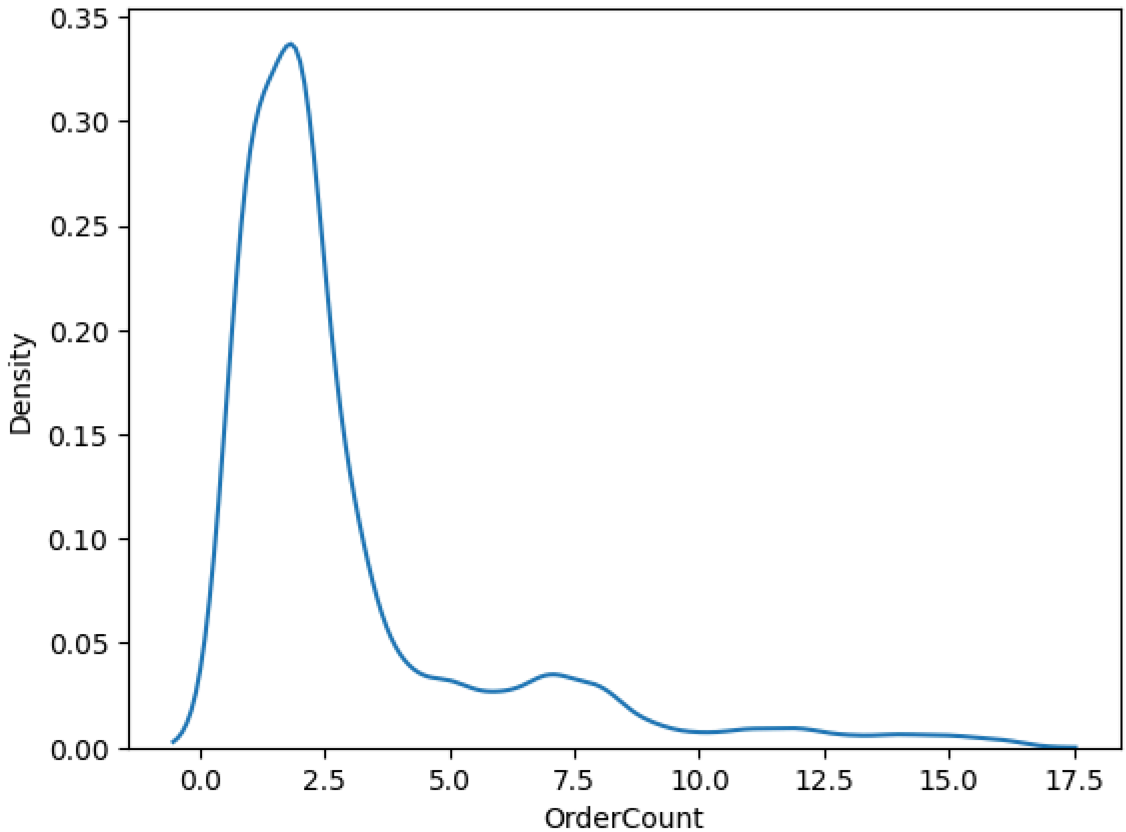
Out[98]: <Axes: xlabel='OrderCount', ylabel='Density'>



In [99]:
```
# Impute with KNN imputer
imputer_2 = KNNImputer(n_neighbors=2)
df['OrderCount']=imputer_2.fit_transform(df[['OrderCount']])
```

In [100]:
```
sns.kdeplot(df , x='OrderCount')
```

Out[100]: <Axes: xlabel='OrderCount', ylabel='Density'>

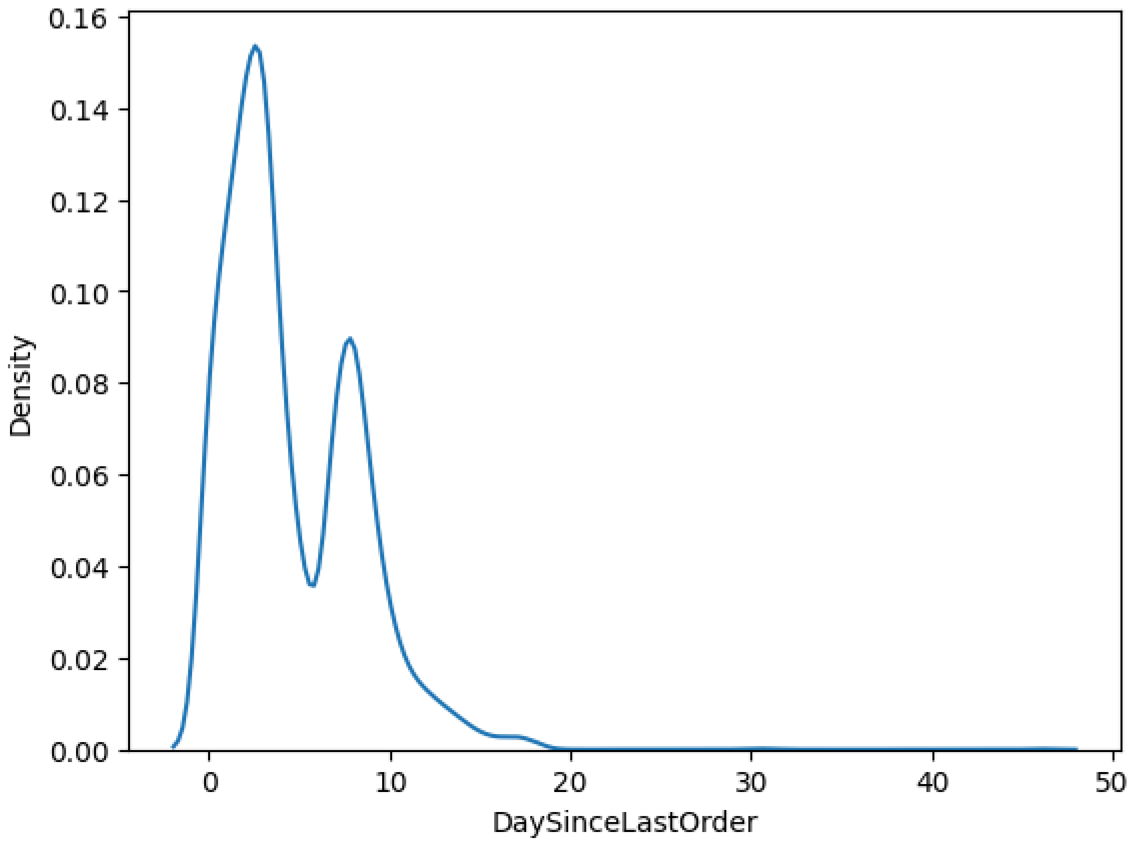


In [101]:
```
df['OrderCount'].isnull().sum()
```

Out[101]: 0

In [102]:
```
sns.kdeplot(df , x='DaySinceLastOrder')
```

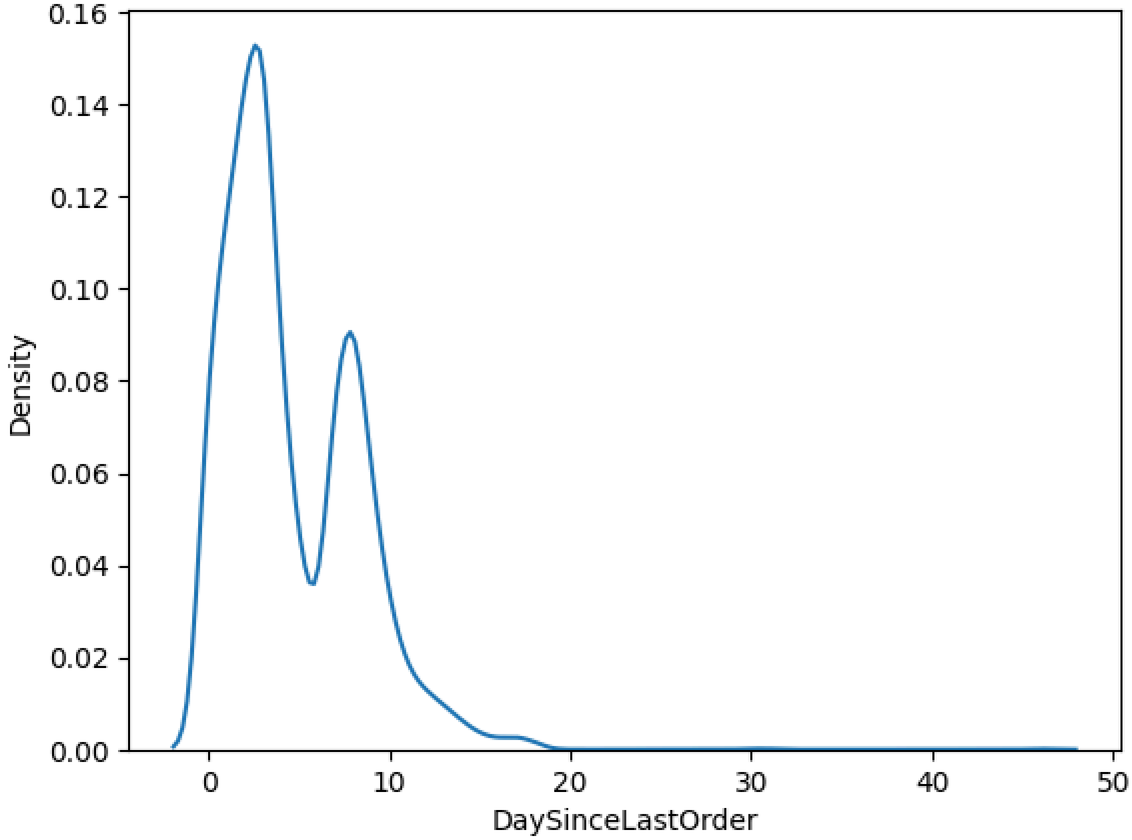
Out[102]:
```
<Axes: xlabel='DaySinceLastOrder', ylabel='Density'>
```



In [103]:
```
# impute with bfill Method
df['DaySinceLastOrder'] = df['DaySinceLastOrder'].fillna(method = 'bfill')
```

In [104]:
```
sns.kdeplot(df , x='DaySinceLastOrder')
```

Out[104]:
```
<Axes: xlabel='DaySinceLastOrder', ylabel='Density'>
```



In [105]:
```
df['DaySinceLastOrder'].isnull().sum()
```

Out[105]:
```
0
```

In [106]:
```
# After we Checked the data the Customer ID Column not important for our Model
df.drop('CustomerID' , axis = 1 , inplace = True)
```

In [107]:
```
df.shape
```

Out[107]:
```
(5630, 19)
```

We Handled Mssing Values

Encoding

In [108]:
```
# 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)
```

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

In [109]:
```
# cat columns
data = df[df.select_dtypes(exclude=np.number).columns]
data
```

Out[109]:

	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

In [110]:
```
le = LabelEncoder()
```

In [111]:
```
# Encode for cat_cols
for i in df.columns:
  if df[i].dtype == 'object':
    df[i] = le.fit_transform(df[i])

df.head(4)
```

Out[111]:

	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPaymentMode	Ge
0	1	4.0	1	3	6.0	2	
1	1	0.0	1	1	8.0	4	
2	1	0.0	1	1	30.0	2	
3	1	0.0	1	3	15.0	2	

In [112]:
```
for i in data.columns:
    data[i] = le.fit_transform(data[i])

data.head(4)
```

Out[112]:

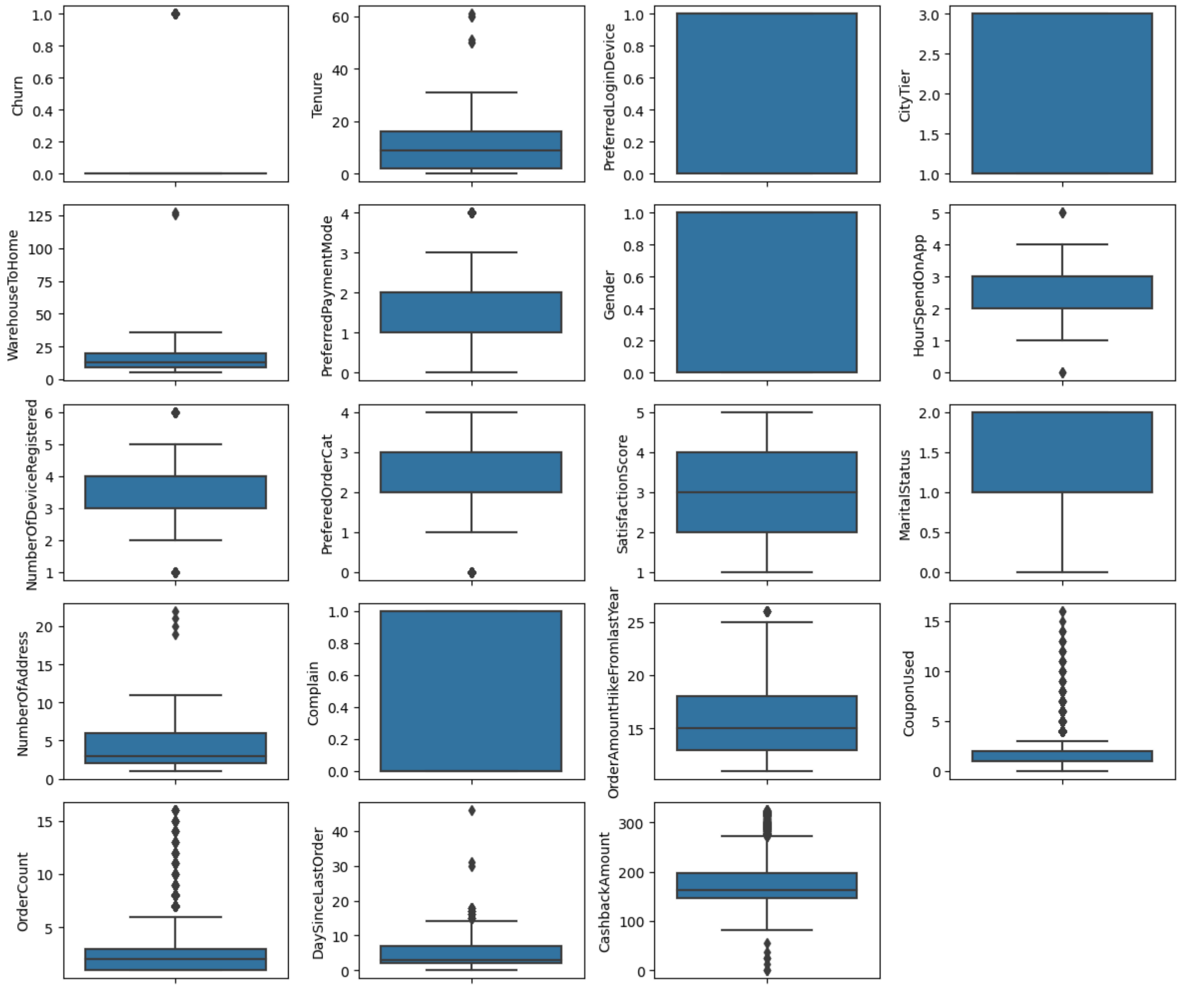
	PreferredLoginDevice	PreferredPaymentMode	Gender	PreferedOrderCat	MaritalStatus
0	1	2	0	2	2
1	1	4	1	3	2
2	1	2	1	3	2
3	1	2	1	2	2

Handling Outliers

In [113]:
```
df.dtypes
```

Out[113]:
```
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
```

In [114]:
```
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()
```



In [115]:
```
# 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
```

In [116]:
```
df.columns
```

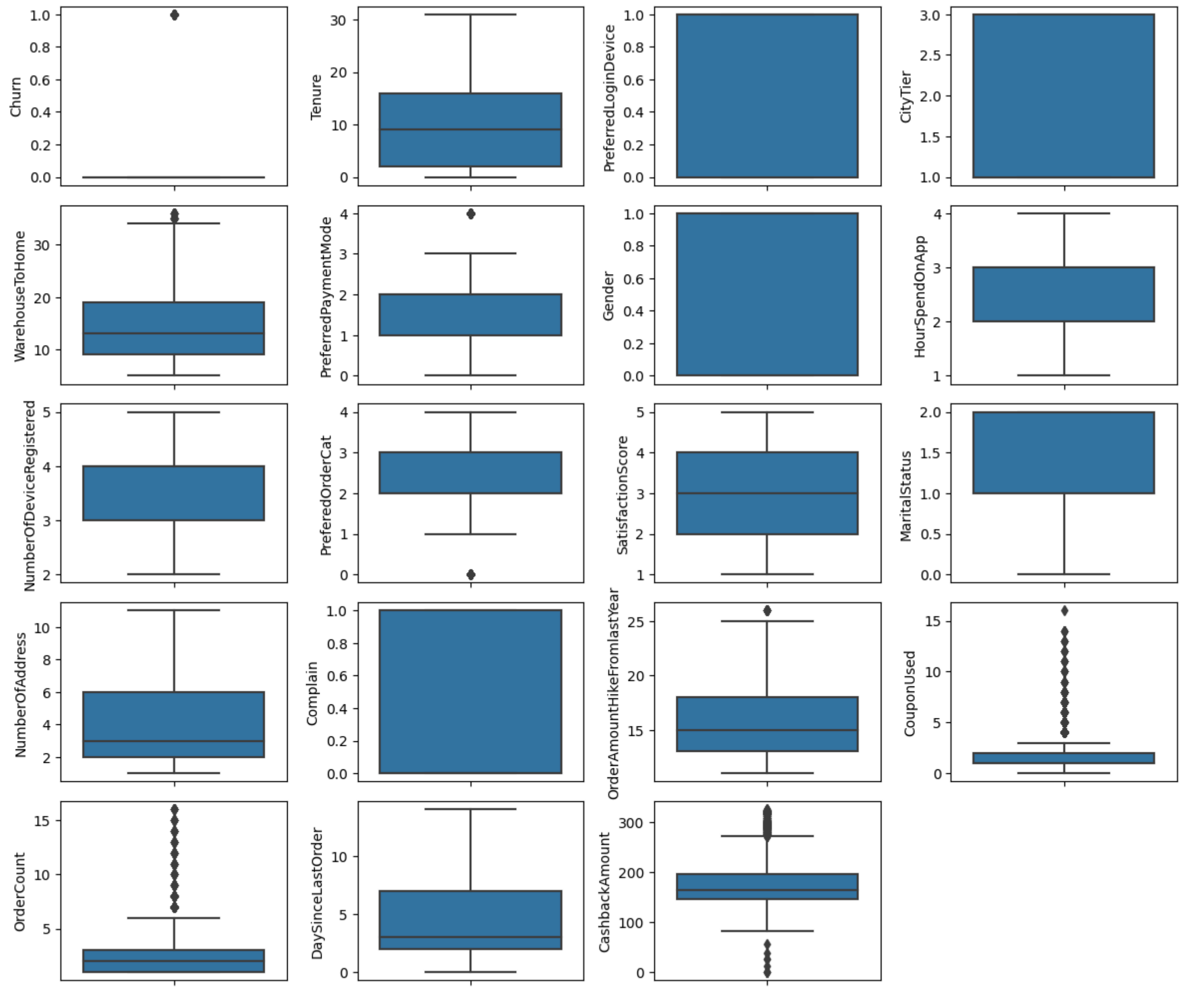
Out[116]:
```
Index(['Churn', 'Tenure', 'PreferredLoginDevice', 'CityTier',
       'WarehouseToHome', 'PreferredPaymentMode', 'Gender', 'HourSpendOnApp',
       'NumberOfDeviceRegistered', 'PreferedOrderCat', 'SatisfactionScore',
       'MaritalStatus', 'NumberOfAddress', 'Complain',
       'OrderAmountHikeFromlastYear', 'CouponUsed', 'OrderCount',
       'DaySinceLastOrder', 'CashbackAmount'],
      dtype='object')
```

In [117]:
```
# lets Give our Functions columns contains outlier
cols_outliers = ['Tenure' , 'WarehouseToHome' , 'NumberOfAddress' , 'DaySinceL

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

df.head(4)
```

Out[117]:

	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPaymentMode	Ge
0	1	4.0	1	3	6.0	2	
1	1	0.0	1	1	8.0	4	
2	1	0.0	1	1	30.0	2	
3	1	0.0	1	3	15.0	2	

In [118]:
```
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

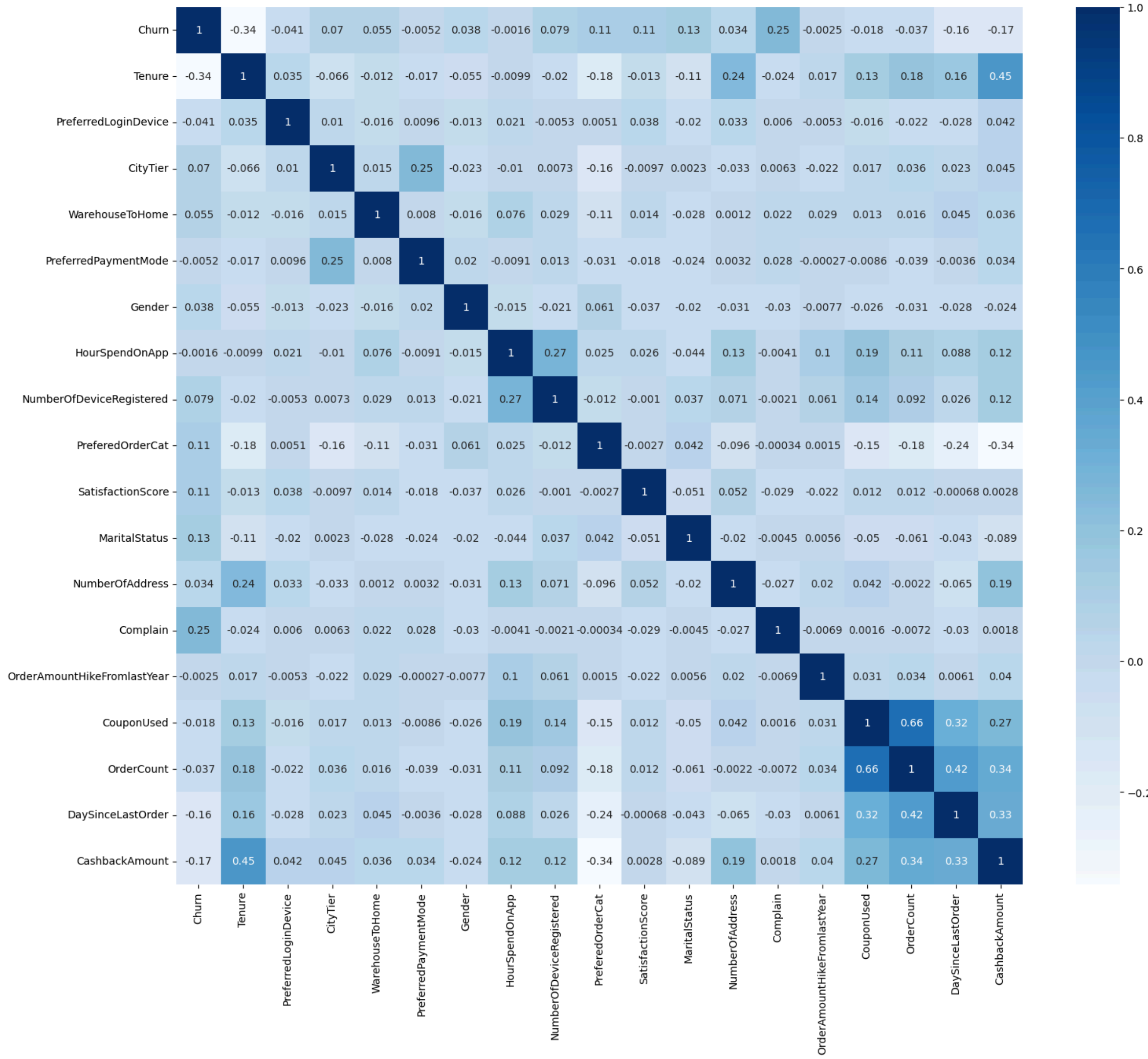
In [119]:
```
corr_matrix = df.corr()
corr_matrix
```

Out[119]:

	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseT
Churn	1.000000	-0.336058	-0.041250	0.069595	0
Tenure	-0.336058	1.000000	0.034596	-0.065933	-0
PreferredLoginDevice	-0.041250	0.034596	1.000000	0.010097	-0
CityTier	0.069595	-0.065933	0.010097	1.000000	0
WarehouseToHome	0.054768	-0.011849	-0.015852	0.014636	1
PreferredPaymentMode	-0.005156	-0.016797	0.009610	0.251539	0
Gender	0.038193	-0.054684	-0.012892	-0.022759	-0
HourSpendOnApp	-0.001624	-0.009877	0.021446	-0.010387	0
NumberOfDeviceRegistered	0.079116	-0.019592	-0.005323	0.007282	0
PreferedOrderCat	0.105149	-0.180637	0.005137	-0.164040	-0
SatisfactionScore	0.108600	-0.013331	0.037642	-0.009735	0
MaritalStatus	0.131982	-0.111074	-0.020207	0.002254	-0
NumberOfAddress	0.033703	0.240939	0.033310	-0.033363	0
Complain	0.252346	-0.023903	0.005983	0.006312	0
OrderAmountHikeFromlastYear	-0.002545	0.017177	-0.005296	-0.022135	0
CouponUsed	-0.017914	0.127314	-0.015940	0.017139	0
OrderCount	-0.036568	0.181138	-0.021975	0.035656	0
DaySinceLastOrder	-0.164448	0.164444	-0.027906	0.023394	0
CashbackAmount	-0.165008	0.453981	0.042321	0.044946	0

In [120]:
```
plt.figure(figsize = (18,15))
sns.heatmap(df.corr() , annot = True , cmap = 'Blues')
```

Out[120]: <Axes: >

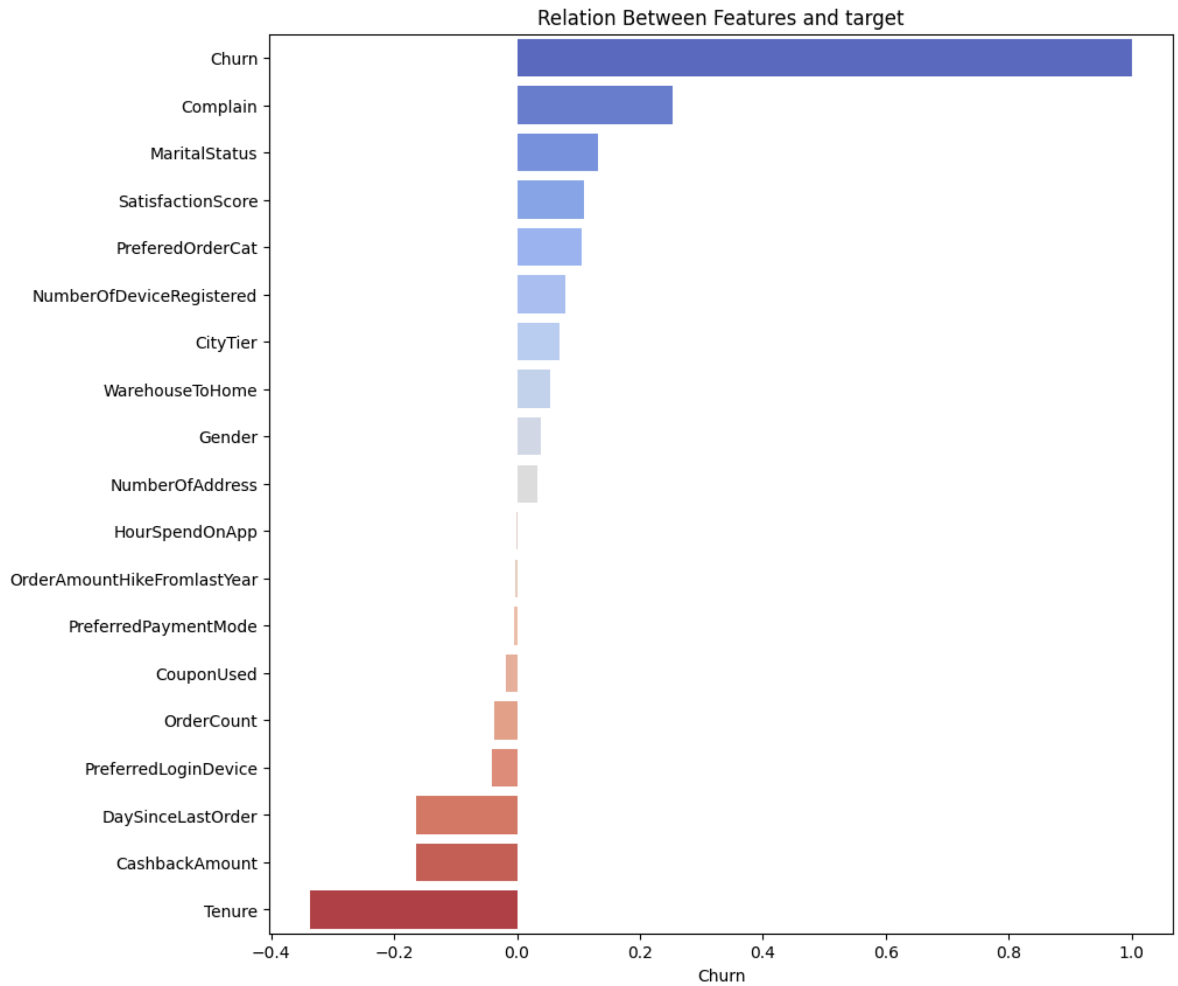


In [121]:
```
churn_corr_vector = corr_matrix['Churn'].sort_values(ascending = False)
churn_corr_vector
```

Out[121]:
```
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                -0.001624
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
```

In [122]:
```
plt.figure(figsize = (10,10))
sns.barplot(x = churn_corr_vector , y = churn_corr_vector.index , palette = 'c
plt.title('Relation Between Features and target')
```

Out[122]:
```
Text(0.5, 1.0, 'Relation Between Features and target')
```



In [123]:
```
fig = px.histogram(df2, x="Churn", color="Churn" ,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='Churn',
yaxis_title='count',
)
fig.show()
```

Handling Imbalanced Data

In [124]:
```
X = df.drop('Churn' , axis = 1)
Y = df['Churn']
```

In [125]:
```
from imblearn.combine import SMOTETomek
```

In [126]:
```
smt = SMOTETomek(random_state=42)
x_over , y_over = smt.fit_resample(X , Y)
```

In [127]:
```
x_over.shape, y_over.shape
```

Out[127]:
```
((8582, 18), (8582,))
```

Split Data

In [128]:
```
x_train , x_test , y_train , y_test = train_test_split(x_over , y_over , test_
```

In [129]:
```
# 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)
```

model building

In [130]:
```
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")
```

In [131]:
```
logisreg_clf = LogisticRegression()
svm_clf = SVC()
dt_clf = DecisionTreeClassifier()
rf_clf = RandomForestClassifier()
XGB_clf = XGBClassifier()
ada_clf = AdaBoostClassifier()
```

In [132]:
```
clf_list = [logisreg_clf, svm_clf, dt_clf, rf_clf, XGB_clf, ada_clf]
clf_name_list = ['Logistic Regression', 'Support Vector Machine', 'Decision Tr

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

In [133]:
```
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: 0.7696021308473447
Test Score: 0.7735922330097087
Acc Train: 0.7696021308473447
Acc Test: 0.7735922330097087

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

Using model: Support Vector Machine
Trainning Score: 0.9052771766272681
Test Score: 0.8807766990291263
Acc Train: 0.9052771766272681
Acc Test: 0.8807766990291263

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

Using model: Decision Tree
Trainning Score: 1.0
Test Score: 0.9409708737864078
Acc Train: 1.0
Acc Test: 0.9409708737864078

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

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: 0.9627184466019417
Acc Train: 1.0
Acc Test: 0.9627184466019417

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

Using model: AdaBoostClassifier
Trainning Score: 0.8763109705343766
Test Score: 0.8520388349514563
Acc Train: 0.8763109705343766
Acc Test: 0.8520388349514563

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

In [134]:
```
# graph to determine best 2 models

all_models = pd.DataFrame({'Train_Accuarcy': train_acc_list , 'Test_Accuarcy'
all_models
```

Out[134]:

	Train_Accuarcy	Test_Accuarcy
Logistic Regression	0.769602	0.773592
Support Vector Machine	0.905277	0.880777
Decision Tree	1.000000	0.940971
Random Forest	1.000000	0.969709
XGBClassifier	1.000000	0.962718
AdaBoostClassifier	0.876311	0.852039

In [135]:
```
# Models vs Train Accuracies
fig = px.bar(all_models, x=all_models['Train_Accuarcy'], y = all_models.index
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 ,
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]

In [136]:
```
!pip install mlxtend
```

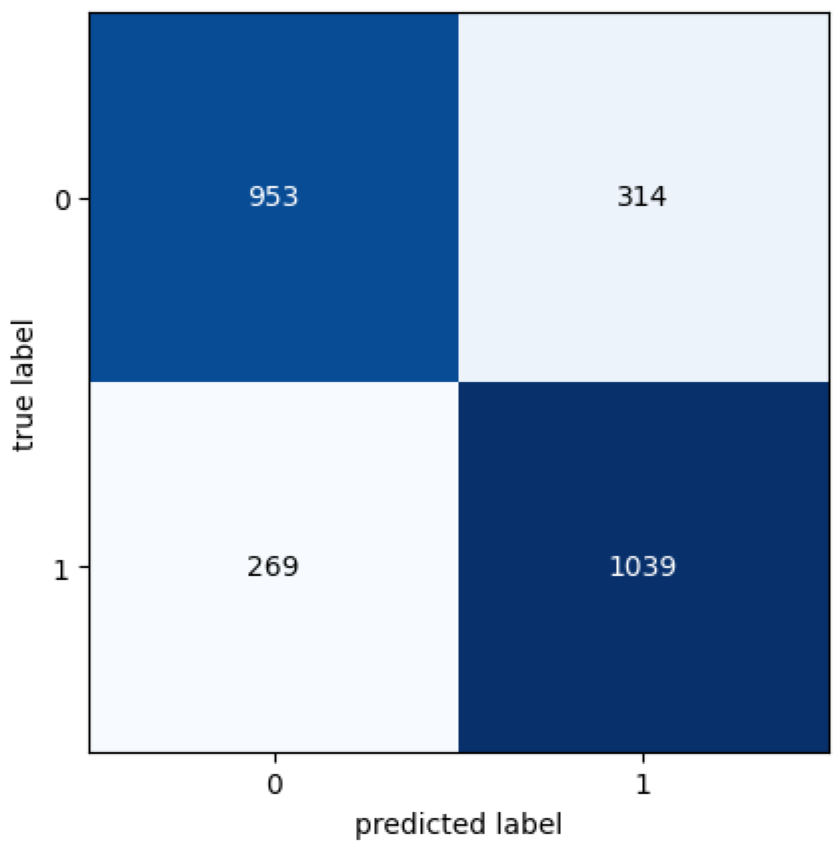
```
Requirement already satisfied: mlxtend in /opt/conda/lib/python3.10/site-pack
ages (0.22.0)
Requirement already satisfied: scipy>=1.2.1 in /opt/conda/lib/python3.10/site
-packages (from mlxtend) (1.11.2)
Requirement already satisfied: numpy>=1.16.2 in /opt/conda/lib/python3.10/sit
e-packages (from mlxtend) (1.23.5)
Requirement already satisfied: pandas>=0.24.2 in /opt/conda/lib/python3.10/si
te-packages (from mlxtend) (2.0.3)
Requirement already satisfied: scikit-learn>=1.0.2 in /opt/conda/lib/python3.
10/site-packages (from mlxtend) (1.2.2)
Requirement already satisfied: matplotlib>=3.0.0 in /opt/conda/lib/python3.1
0/site-packages (from mlxtend) (3.7.2)
Requirement already satisfied: joblib>=0.13.2 in /opt/conda/lib/python3.10/si
te-packages (from mlxtend) (1.3.2)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.10/site-p
ackages (from mlxtend) (68.0.0)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.10/
site-packages (from matplotlib>=3.0.0->mlxtend) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.10/site
-packages (from matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.1
0/site-packages (from matplotlib>=3.0.0->mlxtend) (4.40.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.1
0/site-packages (from matplotlib>=3.0.0->mlxtend) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.10/s
ite-packages (from matplotlib>=3.0.0->mlxtend) (21.3)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.10/sit
e-packages (from matplotlib>=3.0.0->mlxtend) (9.5.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in /opt/conda/lib/python
3.10/site-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python
3.10/site-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site
-packages (from pandas>=0.24.2->mlxtend) (2023.3)
Requirement already satisfied: tzdata>=2022.1 in /opt/conda/lib/python3.10/si
te-packages (from pandas>=0.24.2->mlxtend) (2023.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python
3.10/site-packages (from scikit-learn>=1.0.2->mlxtend) (3.1.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-pac
kages (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxtend) (1.16.0)
```

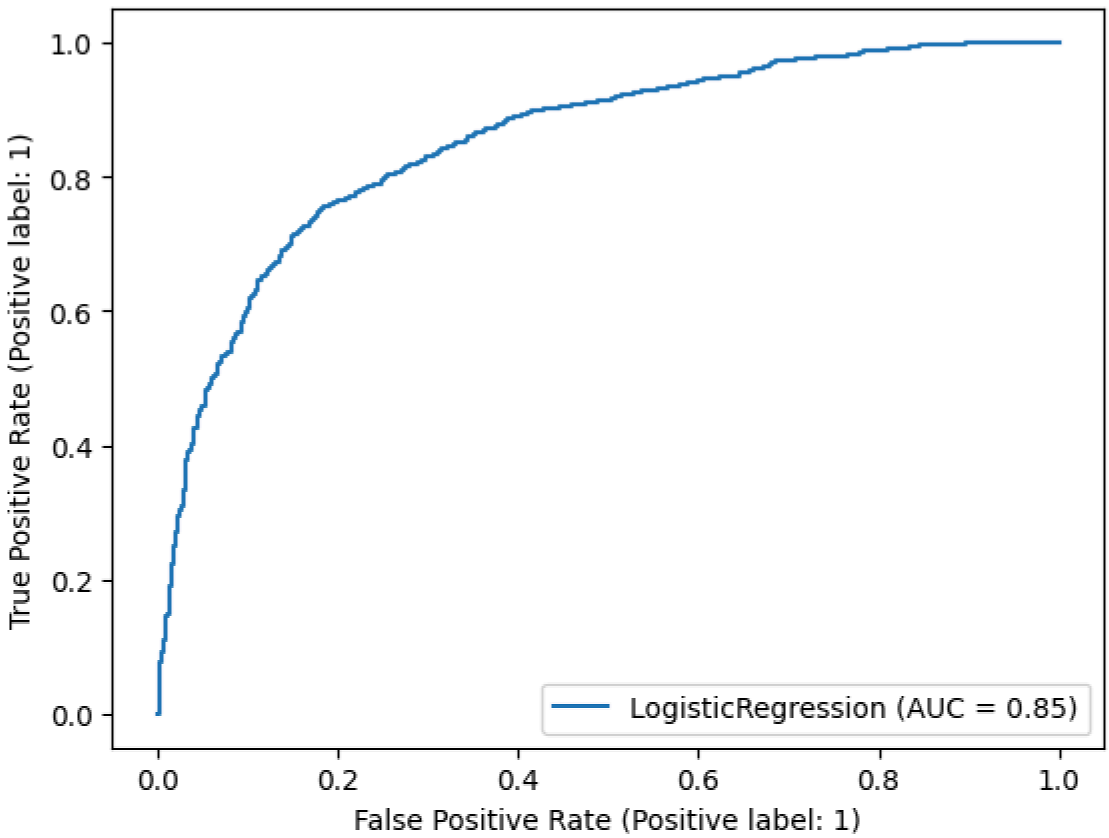
In [137]:
```
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, classification_repo
```

In [138]:
```
# 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 = 0.7735922330097087
ROC Area under Curve = 0.7732564945487547
              precision    recall  f1-score   support

           0    0.77987   0.75217   0.76577      1267
           1    0.76792   0.79434   0.78091      1308

    accuracy                        0.77359      2575
   macro avg    0.77390   0.77326   0.77334      2575
weighted avg    0.77380   0.77359   0.77346      2575

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

Out[138]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7975e09c47c0>



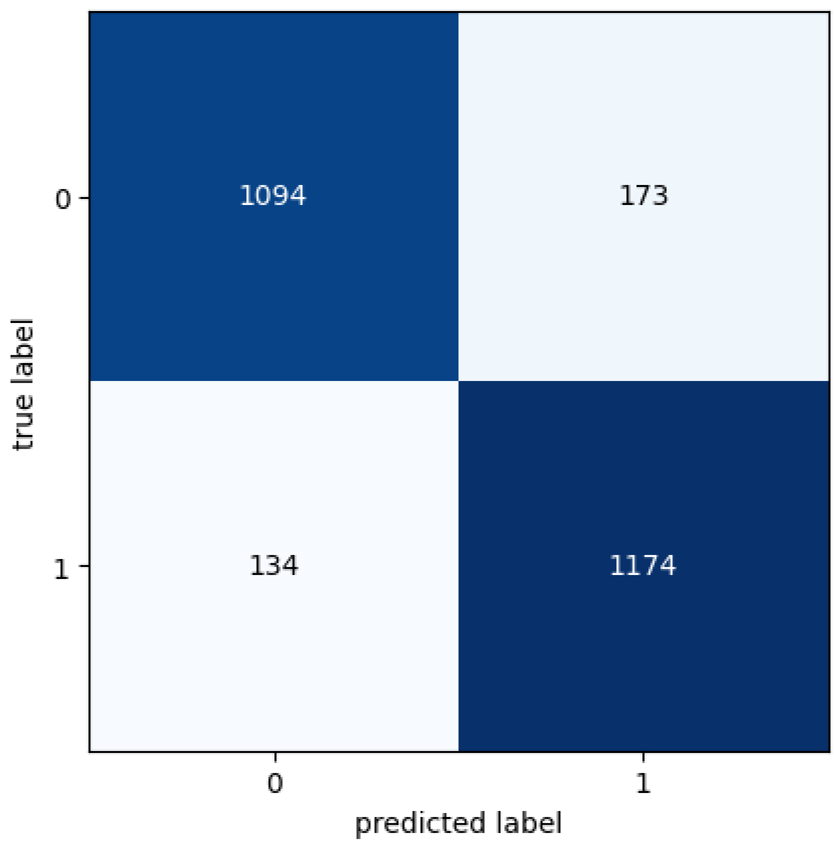


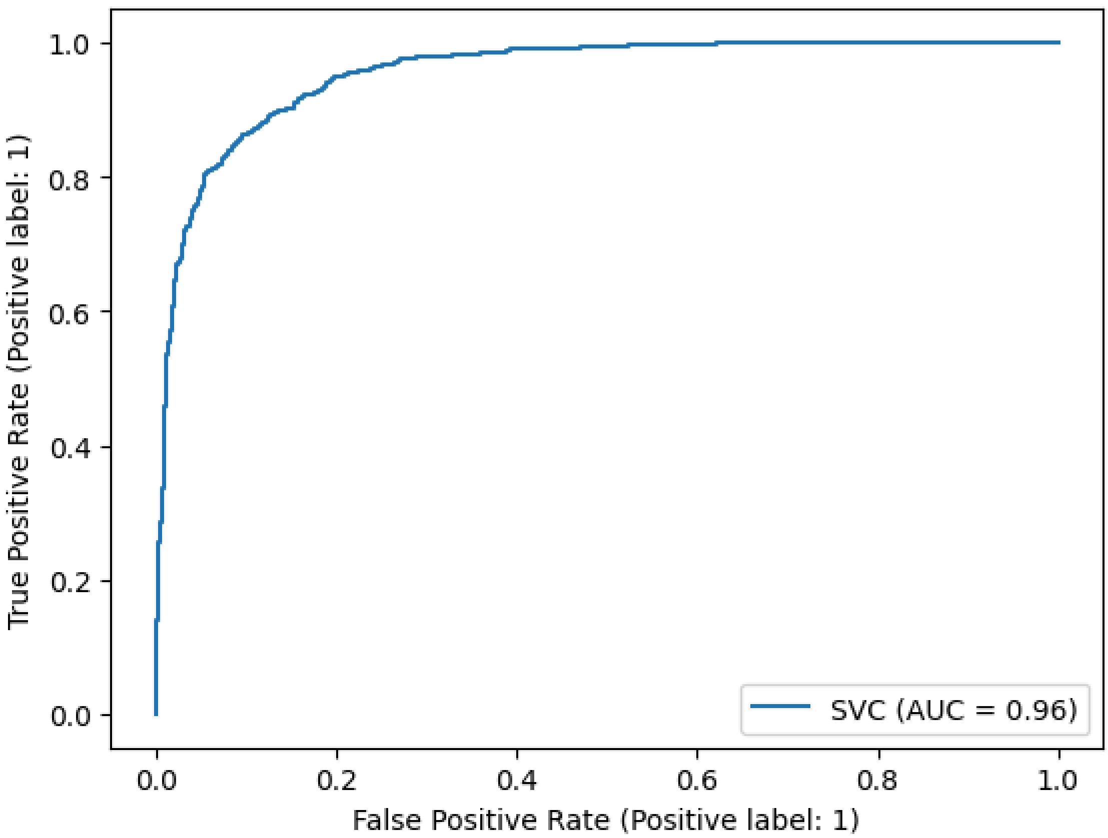
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
LogisticRegression (AUC = 0.85)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)

In [139]:
```
# 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 = 0.8807766990291263
ROC Area under Curve = 0.880505250911759
              precision    recall  f1-score   support

           0    0.89088   0.86346   0.87695      1267
           1    0.87157   0.89755   0.88437      1308

    accuracy                        0.88078      2575
   macro avg    0.88122   0.88051   0.88066      2575
weighted avg    0.88107   0.88078   0.88072      2575
```

Out[139]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7975e00928c0>

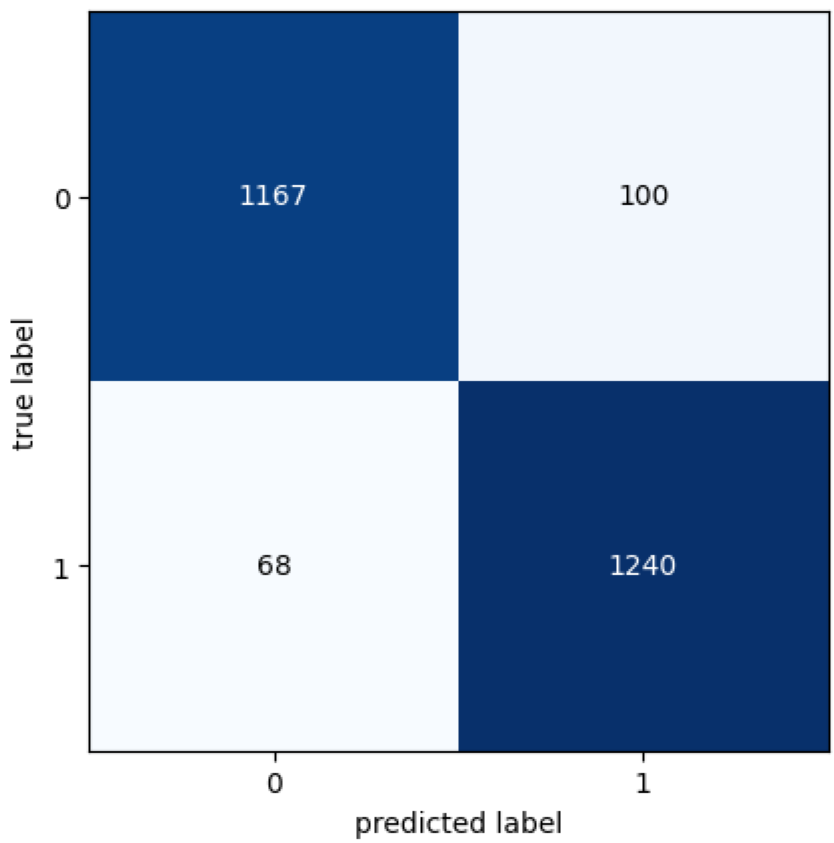


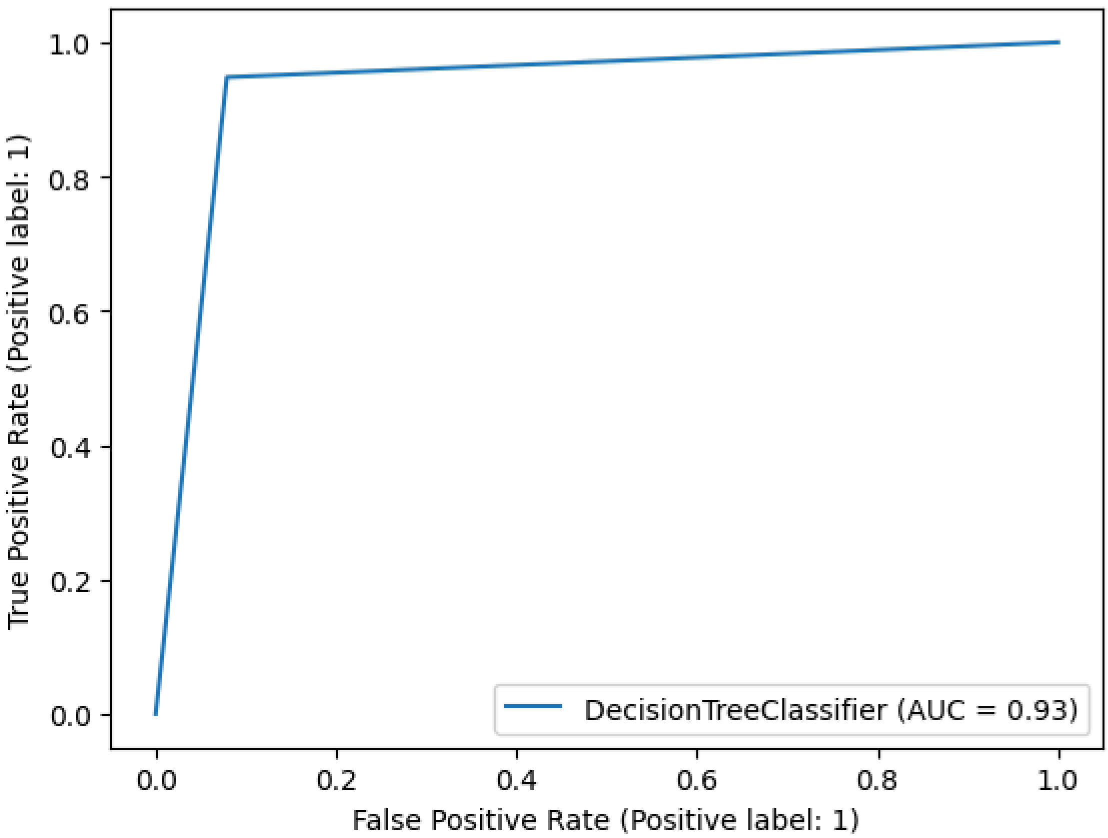

1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
SVC (AUC = 0.96)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)

In [140]:
```
# 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)
```

```
Accuracy = 0.9347572815533981
ROC Area under Curve = 0.9345428170761436
              precision    recall  f1-score   support

           0    0.94494   0.92107   0.93285      1267
           1    0.92537   0.94801   0.93656      1308

    accuracy                        0.93476      2575
   macro avg    0.93516   0.93454   0.93470      2575
weighted avg    0.93500   0.93476   0.93473      2575
```

Out[140]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7975e0659a50>
```



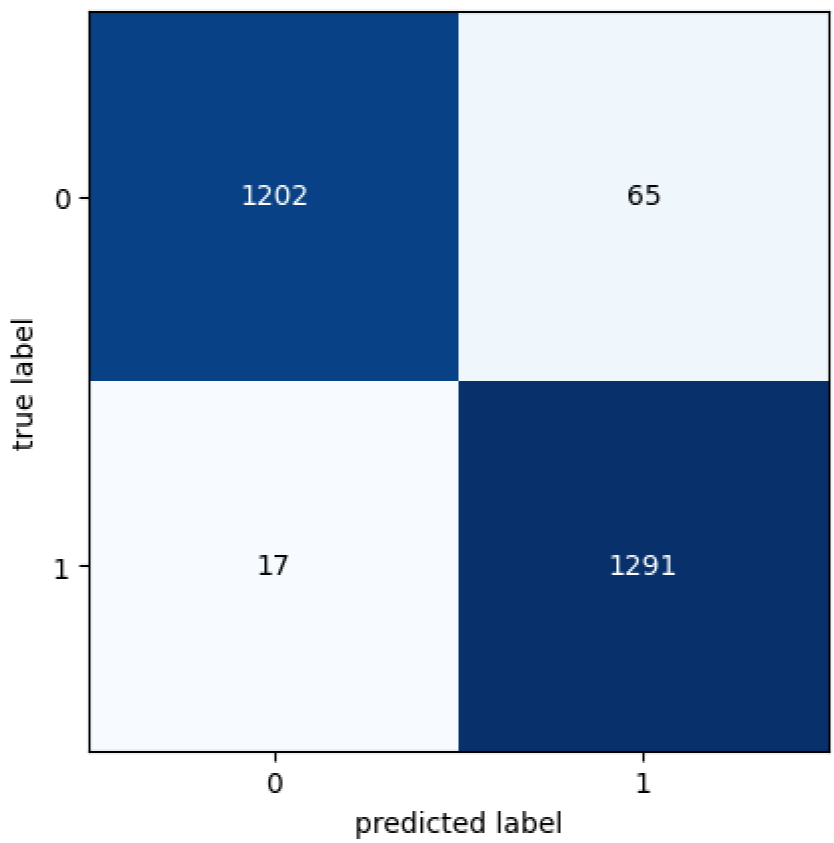


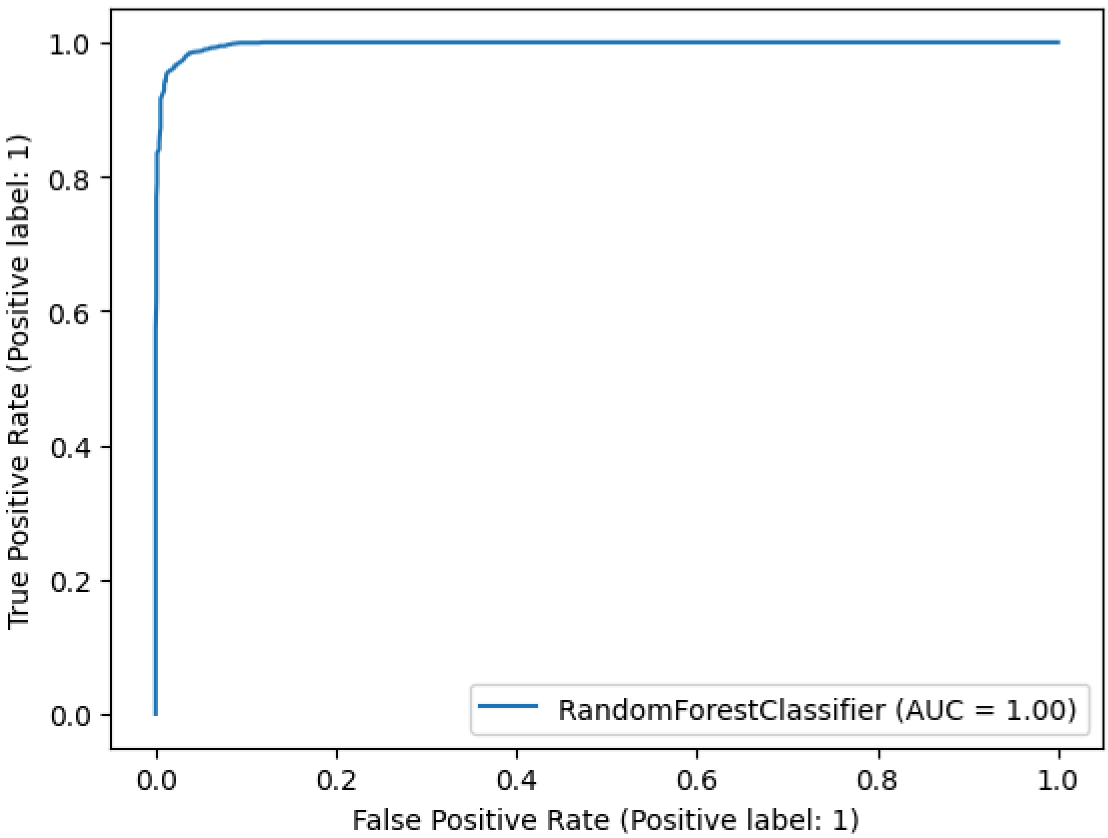
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
DecisionTreeClassifier (AUC = 0.93)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)

In [141]:
```
# random forest
model=RandomForestClassifier()
model.fit(x_train_scaled,y_train)
y_pred = model.predict(x_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
roc_auc4 = roc_auc_score(y_test, y_pred)
print("Accuracy = {}".format(accuracy))
print("ROC Area under Curve = {}".format(roc_auc4))
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 = 0.9681553398058252
ROC Area under Curve = 0.9678503846163129
              precision    recall  f1-score   support

           0    0.98605   0.94870   0.96702      1267
           1    0.95206   0.98700   0.96922      1308

    accuracy                        0.96816      2575
   macro avg    0.96906   0.96785   0.96812      2575
weighted avg    0.96879   0.96816   0.96813      2575
```

Out[141]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7975e73db820>

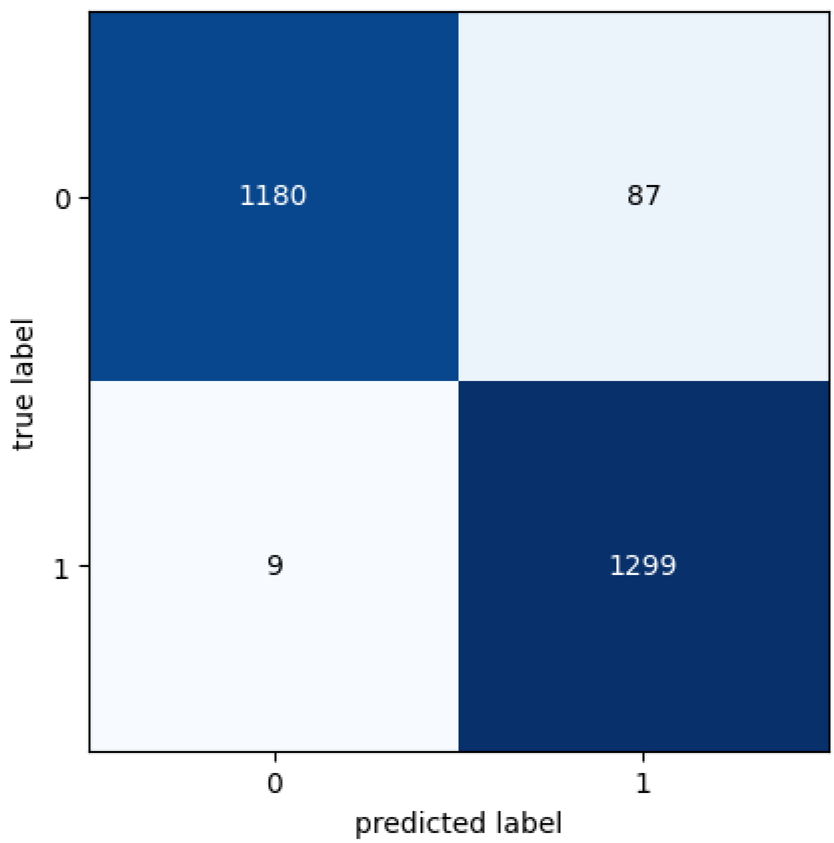


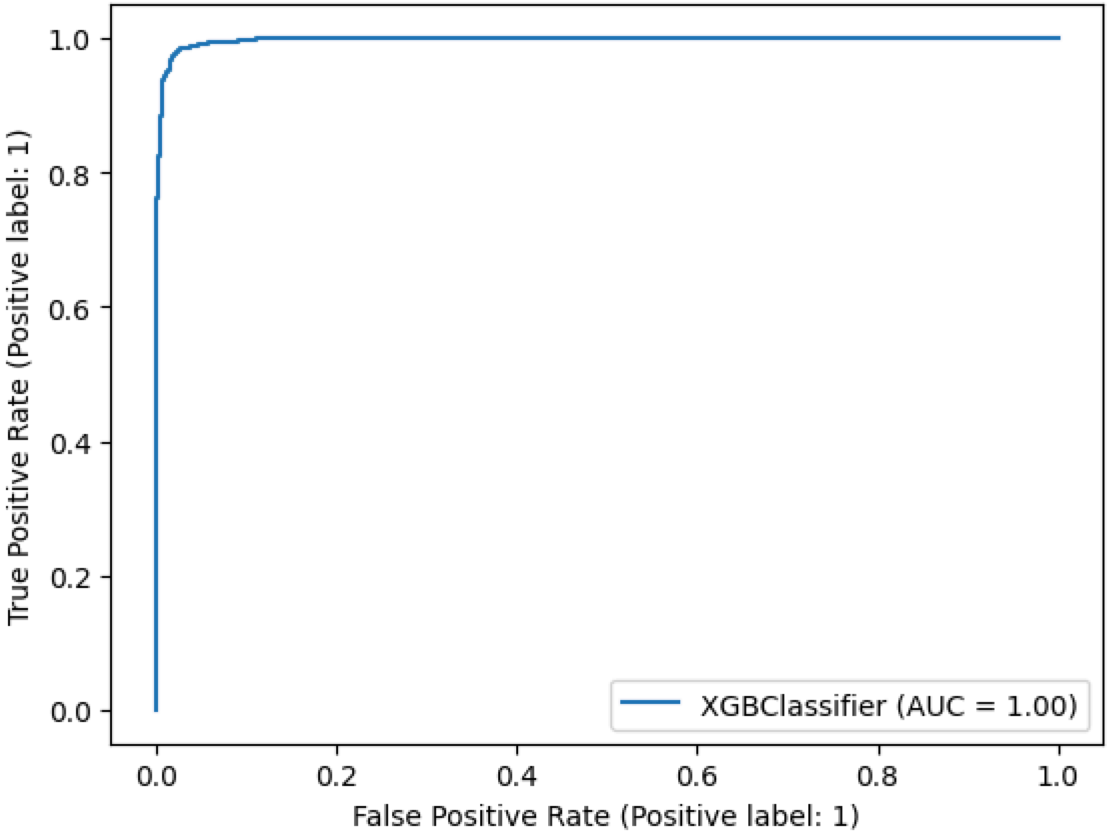

1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
RandomForestClassifier (AUC = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)

In [142]:
```
# XGBoost
model=XGBClassifier()
model.fit(x_train_scaled,y_train)
y_pred = model.predict(x_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
roc_auc5 = roc_auc_score(y_test, y_pred)
print("Accuracy = {}".format(accuracy))
print("ROC Area under Curve = {}".format(roc_auc5))
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 = 0.9627184466019417
ROC Area under Curve = 0.9622265627828505
              precision    recall  f1-score   support

           0    0.99243   0.93133   0.96091      1267
           1    0.93723   0.99312   0.96437      1308

    accuracy                        0.96272      2575
   macro avg    0.96483   0.96223   0.96264      2575
weighted avg    0.96439   0.96272   0.96267      2575
```

Out[142]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7975e0f2d150>



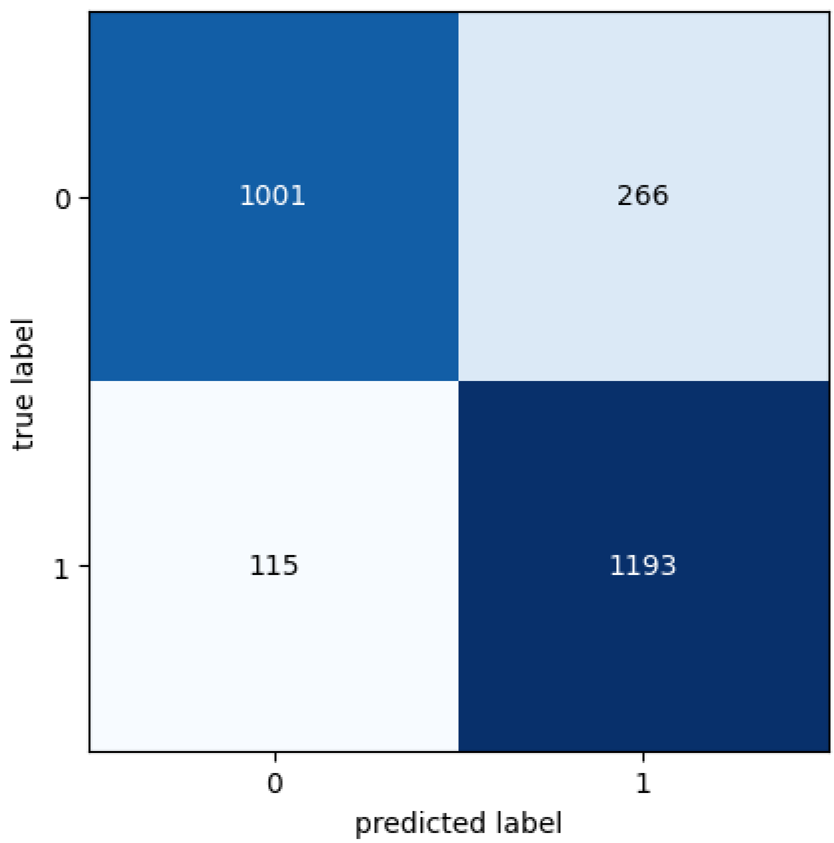


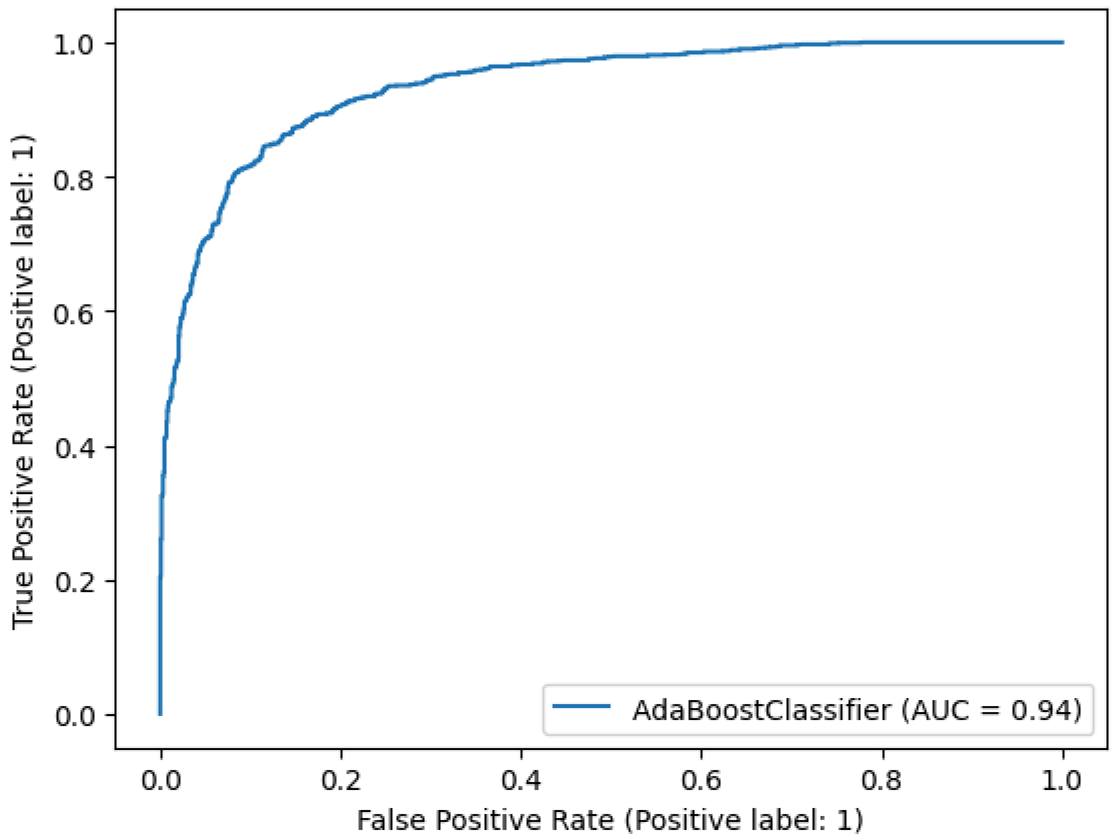
True Positive Rate (Positive label: 1)
False Positive Rate (Positive label: 1)
XGBClassifier (AUC = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0

In [143]:
```
# adaboost
model=AdaBoostClassifier()
model.fit(x_train_scaled,y_train)
y_pred = model.predict(x_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
roc_auc6 = roc_auc_score(y_test, y_pred)
print("Accuracy = {}".format(accuracy))
print("ROC Area under Curve = {}".format(roc_auc6))
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 = 0.8520388349514563
ROC Area under Curve = 0.8510673796610743
              precision    recall  f1-score   support

           0    0.89695   0.79006   0.84012      1267
           1    0.81768   0.91208   0.86231      1308

    accuracy                        0.85204      2575
   macro avg    0.85732   0.85107   0.85121      2575
weighted avg    0.85669   0.85204   0.85139      2575
```

Out[143]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7975e062f2e0>
```





In [144]:
```
pip install pycaret
```

```
Collecting pycaret
  Downloading pycaret-3.1.0-py3-none-any.whl (483 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 483.9/483.9 kB 11.2 MB/s eta
0:00:00
Requirement already satisfied: ipython>=5.5.0 in /opt/conda/lib/python3.1
0/site-packages (from pycaret) (8.14.0)
Requirement already satisfied: ipywidgets>=7.6.5 in /opt/conda/lib/python
3.10/site-packages (from pycaret) (7.7.1)
Requirement already satisfied: tqdm>=4.62.0 in /opt/conda/lib/python3.10/s
ite-packages (from pycaret) (4.66.1)
Requirement already satisfied: numpy<1.24,>=1.21 in /opt/conda/lib/python
3.10/site-packages (from pycaret) (1.23.5)
Collecting pandas<2.0.0,>=1.3.0 (from pycaret)
  Downloading pandas-1.5.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (12.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.1/12.1 MB 48.7 MB/s eta
0:00:00
Requirement already satisfied: jinja2>=1.2 in /opt/conda/lib/python3.10/si
te-packages (from pycaret) (3.1.2)
```

In [145]:
```
from pycaret.classification import *
```

In [146]:
```
# init setup
model_setup = setup(df , target = 'Churn' , train_size=0.7)
```

	Description	Value
0	Session id	1692
1	Target	Churn
2	Target type	Binary
3	Original data shape	(5155, 19)
4	Transformed data shape	(5155, 19)
5	Transformed train set shape	(3608, 19)
6	Transformed test set shape	(1547, 19)
7	Numeric features	18
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Fold Generator	StratifiedKFold
13	Fold Number	10
14	CPU Jobs	-1
15	Use GPU	False
16	Log Experiment	False
17	Experiment Name	clf-default-name
18	USI	9033

In [147]:
```
# model training and selection
best_model = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
xgboost	Extreme Gradient Boosting	0.9587	0.9760	0.8278	0.9175	0.8699	0.8454	0.8473	0.2400
lightgbm	Light Gradient Boosting Machine	0.9587	0.9754	0.8145	0.9307	0.8683	0.8439	0.8469	0.3130
et	Extra Trees Classifier	0.9573	0.9889	0.7815	0.9567	0.8590	0.8342	0.8409	0.2190
rf	Random Forest Classifier	0.9554	0.9811	0.7832	0.9415	0.8543	0.8283	0.8336	0.2640
catboost	CatBoost Classifier	0.9432	0.9755	0.7386	0.9046	0.8128	0.7798	0.7856	2.8620
dt	Decision Tree Classifier	0.9324	0.8807	0.8029	0.7974	0.7997	0.7590	0.7593	0.0250
gbc	Gradient Boosting Classifier	0.9127	0.9294	0.6176	0.8176	0.7028	0.6529	0.6622	0.2500
ada	Ada Boost Classifier	0.8916	0.9003	0.5527	0.7345	0.6298	0.5679	0.5763	0.1250
lr	Logistic Regression	0.8808	0.8660	0.4500	0.7375	0.5563	0.4925	0.5138	0.6580
lda	Linear Discriminant Analysis	0.8792	0.8588	0.4203	0.7476	0.5364	0.4735	0.5003	0.0230
ridge	Ridge Classifier	0.8628	0.0000	0.2053	0.8954	0.3320	0.2881	0.3874	0.0180
knn	K Neighbors Classifier	0.8559	0.8294	0.3906	0.6070	0.4738	0.3953	0.4088	0.0380
nb	Naive Bayes	0.8550	0.8200	0.5911	0.5635	0.5760	0.4888	0.4896	0.0190
qda	Quadratic Discriminant Analysis	0.8537	0.8361	0.6059	0.5581	0.5802	0.4919	0.4930	0.0190
dummy	Dummy Classifier	0.8326	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0380
svm	SVM - Linear Kernel	0.7922	0.0000	0.5202	0.6189	0.4339	0.3400	0.3998	0.0310

```
Processing:   0%|          | 0/69 [00:00<?, ?it/s]
```

In [148]:
```
# evaluate trained model
evaluate_model(best_model)
```

```
interactive(children=(ToggleButtons(description='Plot Type:', icons=('',), op
tions=(('Pipeline Plot', 'pipelin…
```