E-COMMERCE CHURN

ITI 3-Month Program // Data Analysis Track

ABSTRACT

Churn problem refers to the issue of losing customers who were previously active on an online store. The churn rate is an important metric for e-commerce businesses because it can significantly impact their revenue and profitability.

AUTHORS

Omar Hany Mohamed Mohamed

Sandra Ezzat Rezk Moses

Mariam Alaa Eltoukhy

Ahmed Yasser Mohammed

Table of Contents

Ecommerce Churn

Importing libraries & Loading data into a data frame:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from scipy.stats import ttest_ind
import scipy.stats as stats

#for plotting
import matplotlib.pyplot as plt
import seaborn as sns

#warnings
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df = pd.read_csv("/kaggle/input/e-commerce-churn/E Commerce Dataset.csv")
```

EDA & Checking the data using:

Checking data shape and data types and looking for nulls:

```
df.head()
df.info()
df.isnull().sum()
```

Checking the percentage of null values:

```
#percentage of missing data per column
round(df.isnull().sum()/df.shape[0], 2)*100
```

Describing the data numbers:

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

Data Validation & Cleaning:

Dropping 'Unnamed: 0' Column because it's an index column:

```
df.pop(df.columns[0])
```

Checking for duplicates:

```
df.duplicated().sum()
11260
```

Dropping Duplicates:

```
df = df.drop_duplicates()
df.duplicated().sum()
```

Checking distribution of numerical variables by visualizing boxplot & histograms:

```
for i in df.columns:
 if (df[i].dtype != 'object') & (i not in exclude_cols):
    fig, axs = plt.subplots(ncols=2, figsize=(10, 5))

    sns.boxplot(x=df[i], ax=axs[0])
    axs[0].set_title(i + ' (box plot)')

    sns.histplot(x=df[i], kde=False, ax=axs[1])
    axs[1].set_title(i + ' (histogram)')

    plt.show()
```

Numerical variables outliers:

All outliers/distributions are logical except

Tenure : have 100 values less than 0, there's also outliers (50,51,60,61).

Checking unique values & its count for categorical variables:

In [13]:

```
for i in df.columns:
 if(df[i].dtype == 'object'):
    print(df[i].value_counts())
    print('\n')
```

Categorical variables validation:

Gender: needs to be consistent (m-f to male-female)

PreferredLoginDevice: needs to be consistent (mobile phone - phone) 0 category needs to be dealt with (could be imputed with the mode/ or create 'other' category for it)

PreferredPaymentMode: needs to be consistent (CC – COD)

PreferedOrderCat: needs to be consistent (Mobile phone and mobile are the same thing)

SatisfactionScore: '589314' category needs to be dealt with

Gender replacement:

In [15]:
```
df['Gender'] = df['Gender'].replace('m', 'Male').replace('f', 'Female')
```

Preferred Login Device handling:

In [18]:
```
df['PreferredLoginDevice'] = np.where(df['PreferredLoginDevice'] == '0', df['PreferredLoginDevice'].mode() , df['PreferredLoginDevice'])
```

In [19]:
```
df['PreferredLoginDevice'] = df['PreferredLoginDevice'].replace('Phone', 'Mobile Phone')
```

Preferred Order Category handling:

In [22]:
```
df['PreferedOrderCat'] = np.where(df['PreferedOrderCat'] == 'Mobile', 'Mobile Phone' , df['PreferedOrderCat'])
```

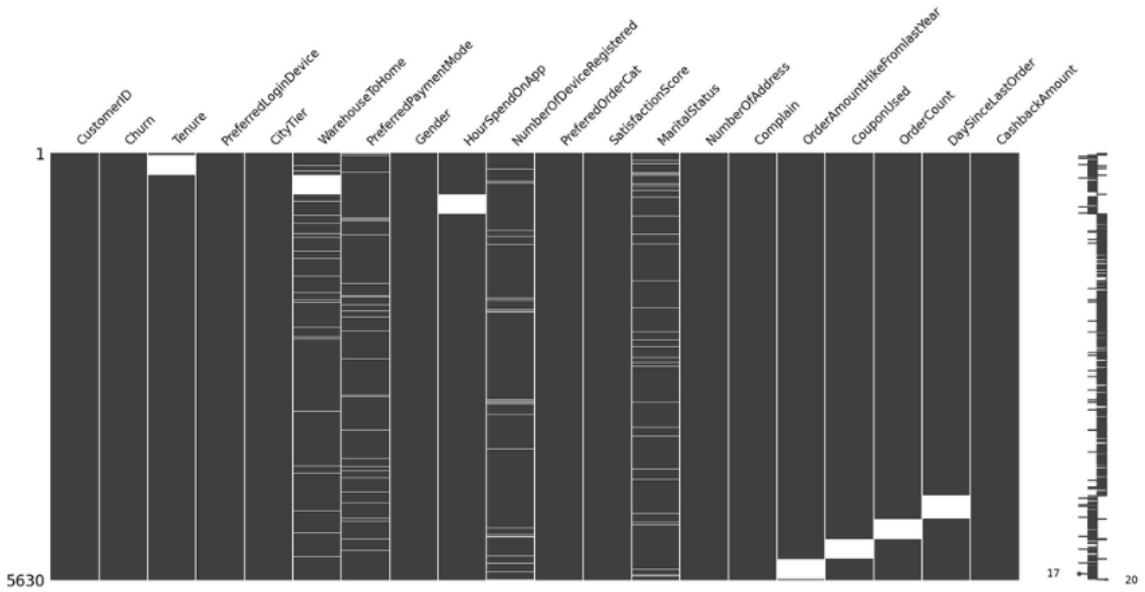
Preferred Payment Mode Handling:

In [25]:
```
df['PreferredPaymentMode'] = df['PreferredPaymentMode'].replace('CC', 'Credit Card').replace('COD', 'Cash on Delivery')
```

Dealing with nulls:

Visualize missing data to check for missing pattern using missingo library and sorting data with cashback amount:

```
In [30]:
print("Cashback Amount Missing Pattern :")
msno.matrix(df.sort_values(by='CashbackAmount'))
plt.show()
```



Finding a pattern:

I found that sorting by the CashbackAmount columns creates a pattern between the missing values and the CashbackAmount columns, which identifies the missingness as missing at random (MAR)

Missing at Random means the propensity for a data point to be missing is not related to the missing data, but it is related to some of the observed data.

if the probability of being missing is the same only within groups defined by the observed data, then the data are missing at random (MAR)

Filling Nulls by Iterative Imputer:

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)

vars_with_missing = ['DaySinceLastOrder', 'OrderCount','CouponUsed','OrderAmountHikeFromlastYear','HourSpendOnApp','Tenure','W
arehouseToHome','NumberOfDeviceRegistered']

df_missing = df[vars_with_missing]

# Create an instance of the IterativeImputer class
imputer = IterativeImputer( estimator=knn,random_state=0)

# Fit the imputer to the data
imputer.fit(df_missing)

# Generate imputed values for each missing data point
imputed_data = imputer.transform(df_missing)

# Combine imputed data sets to obtain a single data set with imputed values
df_imputed = pd.concat([df.drop(vars_with_missing, axis=1), pd.DataFrame(imputed_data, columns=vars_with_missing)], axis=1)
```

Filling Nulls of categorical variables by mode:

```
df_imputed['PreferredPaymentMode'].fillna(df_imputed['PreferredPaymentMode'].mode()[0], inplace=True)
df_imputed['MaritalStatus'].fillna(df_imputed['MaritalStatus'].mode()[0], inplace=True)
```

Validating there's no nulls left:

In [32]:
```
df2 = df_imputed.copy()
df2.isnull().sum()
```

Out[32]:

Handling 'Tenure' outliers:

For this column we chose

Winsorizing:

Where any value of a variable above or below a percentile k on each side of the variables' distribution is replaced with the value of the k-th percentile itself.

Checking ‘Tenure’ quartiles:

In [35]:

```
#Explore different quantiles at the upper end
print('90% quantile:   ', df2['Tenure'].quantile(0.90))
print('92.5% quantile: ', df2['Tenure'].quantile(0.925))
print('95% quantile:   ', df2['Tenure'].quantile(0.95))
print('97.5% quantile: ', df2['Tenure'].quantile(0.975))
print('99% quantile:   ', df2['Tenure'].quantile(0.99))
print('99.9% quantile: ', df2['Tenure'].quantile(0.999))
```

```
90% quantile:    23.0
92.5% quantile:  25.0
95% quantile:    27.0
97.5% quantile:  29.0
99% quantile:    30.0
99.9% quantile:  31.0
```

Using winsorize to clip the outliers on 5th quartile and 95th quartile:

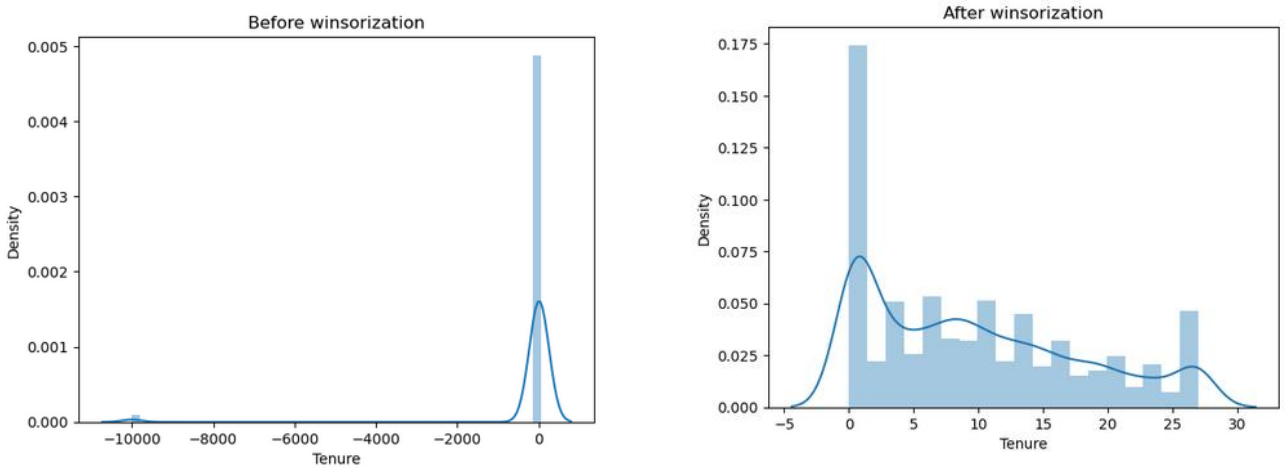
In [36]:

```
from scipy.stats.mstats import winsorize

#Winsorize on right-tail
winsorize(df2['Tenure'], limits=(0.05, 0.05), inplace=True)

df2['Tenure'].describe()
```

Out[36]:

```
count    5630.000000
mean        9.677620
std         8.278076
min         0.000000
25%         2.000000
50%         8.000000
75%        15.000000
max        27.000000
Name: Tenure, dtype: float64
```

Checking ‘Tenure’ column before and after winsorization:



In [38]:
```
print(df['Tenure'].mean())
print(df2['Tenure'].mean())
```

```
-176.22648537902776
9.677619893428064
```

Handling ‘SatisfactionScore’ error outliers:

Since it’s an obviously an entry error, we will replace this score with ‘3’ as 3 is a passive value and it wouldn’t affect eh NPS score if we calculated it.

```
df2['SatisfactionScore'] = np.where(df2['SatisfactionScore']==589314, 3, df2['SatisfactionScore'])
```

Working with a clean dataset:

In [43]:
```
new_df = df2.copy()
new_df.head()
```

Out[43]:

	CustomerID	Churn	PreferredLoginDevice	CityTier	PreferredPaymentMode	Gender	PreferedOrderCat	SatisfactionScore	Marital
0	50001	1	Mobile Phone	3	Debit Card	Female	Laptop & Accessory	2	Single
1	50002	1	Mobile Phone	1	UPI	Male	Mobile Phone	3	Single
2	50003	1	Mobile Phone	1	Debit Card	Male	Mobile Phone	3	Single
3	50004	1	Mobile Phone	3	Debit Card	Male	Laptop & Accessory	3	Single
4	50005	1	Mobile Phone	1	Credit Card	Male	Mobile Phone	5	Single

Working on the questions:

Q1: Analyze the number of days since the last order by the customer to create targeted marketing campaigns and offer personalized discounts?

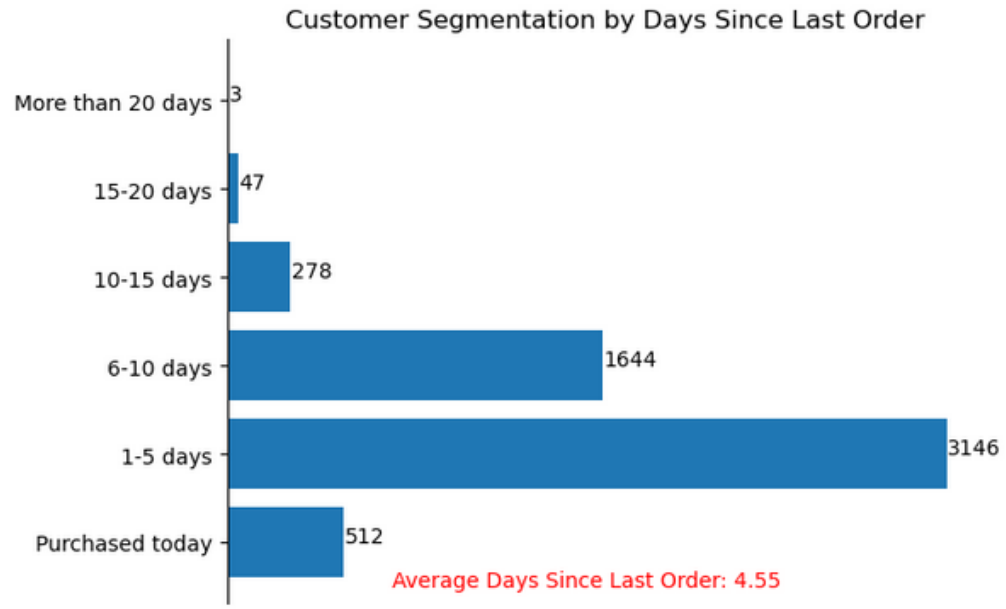
Categorizing DaySinceLastOrder to 5 categories and calculating the mean DaySinceLastOrder:

```
# Calculate the average number of days since the last order for all customers
avg_days_since_last_order = q1['DaySinceLastOrder'].mean()

# Create a new column to segment customers based on the number of days since their last order
customer_count_by_segment = q1['Segment'] = pd.cut(q1['DaySinceLastOrder'], bins=[-1,0,5, 10, 15, 20, 47], labels=['Purchased today','1-5 days', '6-10 days', '10-15 days', '15-20 days', 'More than 20 days'])

# Print the average number of days since the last order for all customers
print('The average number of days since the last order for all customers is:', avg_days_since_last_order)

# Print the number of customers in each segment
print('Number of customers in each segment:')
print(q1['Segment'].value_counts().sort_index())
```

```
The average number of days since the last order for all customers is: 4.551154529307283
Number of customers in each segment:
Purchased today       512
1-5 days             3146
6-10 days            1644
10-15 days            278
15-20 days             47
More than 20 days       3
Name: Segment, dtype: int64
```

Visualizing the categories:

```
counts = q1['Segment'].value_counts().sort_index()

fig, ax = plt.subplots()

ax.barh(counts.index, counts.values)
ax.set_xlabel('Count of Customers')

for i, v in enumerate(counts.values):
    ax.text(v + 3, i, str(v), color='black')#, fontweight='bold')

#ax.set_yticklabels(counts.index, fontsize=12)
#ax.invert_yaxis()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)
ax.xaxis.set_visible(False)
customer_count_by_segment = q1.groupby('Segment').size()
plt.text(x=customer_count_by_segment.max() / 2, y=-0.5, s=f'Average Days Since Last Order: {avg_days_since_last_order:.2
f}', fontsize=10, color='red', ha='center')
plt.title('Customer Segmentation by Days Since Last Order')
```



Considering the customers that didn't made a purchase in the last 15 days to be in risk of churn.
We want to know more about these customers and try to personalize campaigns for them.

Checking Customers that didn't made a purchase in the last 15 days:

How many customer in risk of churn i got and how many have already churned?

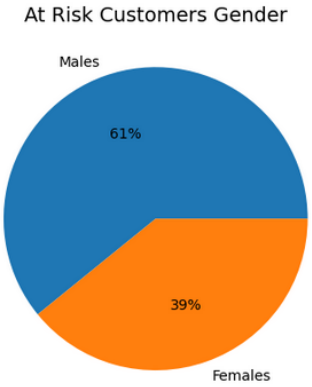
```
q1_risk = q1[q1['DaySinceLastOrder'] >= 15 ]
at_risk_count = q1_risk['DaySinceLastOrder'].count()
print("Count of Customers at risk  = ",at_risk_count)
print("Count of Customers at risk & didn't churn = ",q1_risk['DaySinceLastOrder'][q1_risk['Churn'] == 0].count())
print("Count of Customers at risk & churned already  = ",q1_risk['DaySinceLastOrder'][q1_risk['Churn'] == 1].count())
```

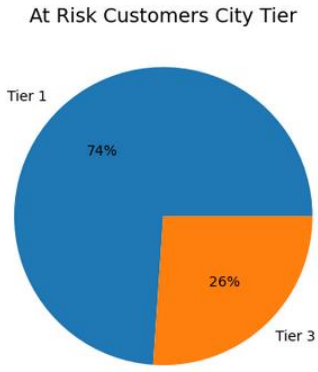
```
Count of Customers at risk  =  69
Count of Customers at risk & didn't churn =  64
Count of Customers at risk & churned already  =  5
```

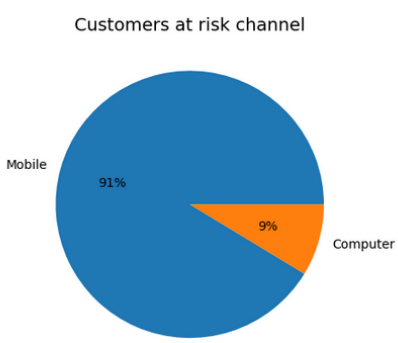
We will work at all the customers at risk (69 customers)

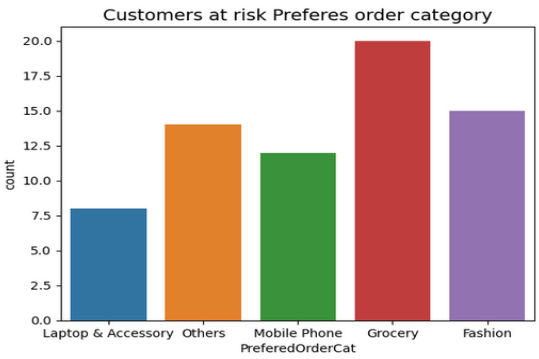
Checking their demographics, behavior, used channels by (value counts) and visualizing it with bar and pie charts:

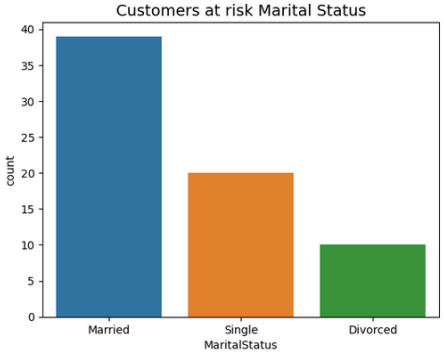
Checking their gender split, in which city tier are the most customers at risk, what are the top categories they are most interested in, What is their marital status? How many of them did raise a complaint in the last month? What is the best channel to communicate with them?











Drawing conclusions and suggesting actions based on a goal:

Actionable Insights:

- The Average days since last order is **4.55**.
- By segmenting the customers upon days since last order: the least **segment** are customers who didn't purchase in **more than 20 days**.
- We will consider the customers who didn't purchase in more than **15 days** are the **customers at risk**, who needs personalized campaigns.
- There Are **69 customers at risk**, 5 of them have already churned.
- **61%** of these customers **are males**.
- **74%** of these customers lives at **City Tier 1**.
- The most preferred order category (from last month) for these customers is **Grocery**
- The majority of them are **married**.
- Only 26% of them had complaints over the last month, so maybe it doesn't affect their buying behavior.
- Over **90%** of them uses **mobile phones**.

Goal: **Lowering the Average days since last order**.

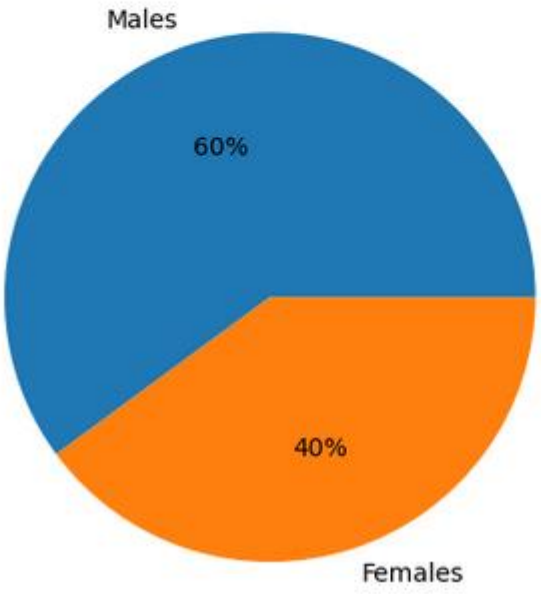
Actions:

- Personalize the campaigns to be more toward males.
- Focus the ads to be geographically towards City Tier 1.
- Offer discounts on "Groceries" to motivate these customers to buy again.
- Offer discounts on products related to married couples.
- Make customer service contact them and make sure all their complains are resolved to avoid churn.
- Do all the communication/discounts/campaigns with them through SMS and calls.

Q2: Is there any difference in the buying behavior of male and female customers?

Checking distribution of males & females:

In [61]:
```
m = (new_df['Gender'] == 'Male').sum()
f =  (new_df['Gender'] == 'Female').sum()
labels = ['Males', 'Females']
plt.pie([m, f], labels= labels, autopct='%.0f%%')
plt.show()
```



Handling the unbalance between them and checking for churn rate:

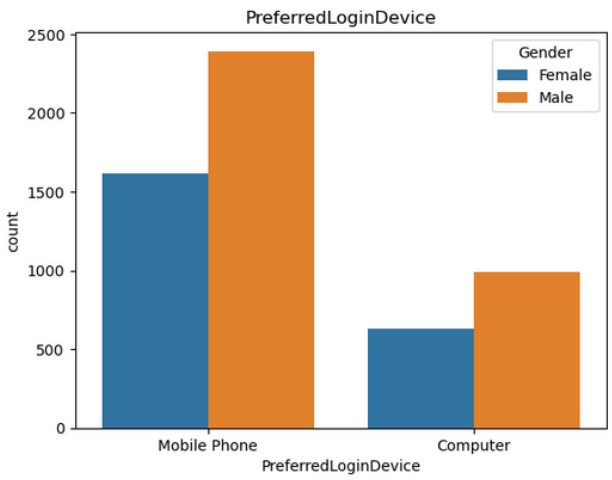
The number of churned males are larger than the churned females And the number of non churned males is also larger than females
But, is this true? or because the percentage of males is dominant next to females in the dataset in the first place 80:60
So, we have to check the rate/percentage.

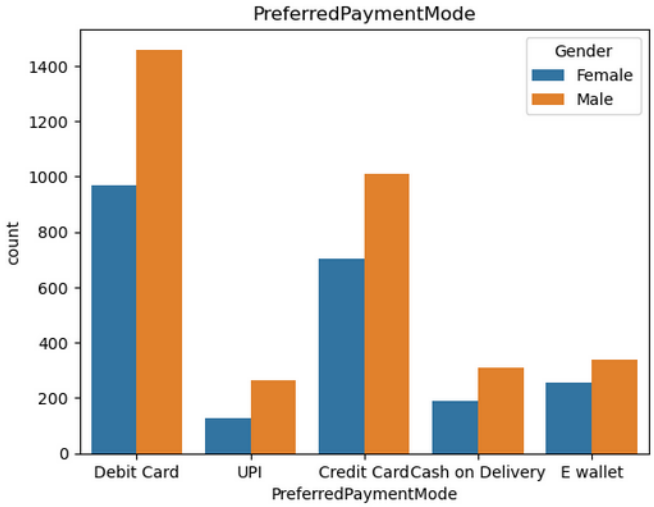
In [63]:
```
churn = (new_df['Churn'] == 1).sum()
nochurn =  (new_df['Churn'] == 0).sum()
males = (new_df['Gender']=="Male").sum()
females = (new_df['Gender']=="Female").sum()
churned_males = ((new_df['Churn'] == 1) & (new_df['Gender']=="Male")).sum()
churned_females = ((new_df['Churn'] == 1) & (new_df['Gender']=="Female")).sum()
nochurn_males = ((new_df['Churn'] == 0) & (new_df['Gender']=="Male")).sum()
nochurn_females = ((new_df['Churn'] == 0) & (new_df['Gender']=="Female")).sum()


males_churn_rate = (churned_males / males)*100
females_churn_rate = (churned_females / females)*100
```

In [64]:
```
print('Churned Males Rate :' ,males_churn_rate)
print('Churned Females Rate :', females_churn_rate)
```

```
Churned Males Rate : 17.73049645390071
Churned Females Rate : 15.49421193232413
```

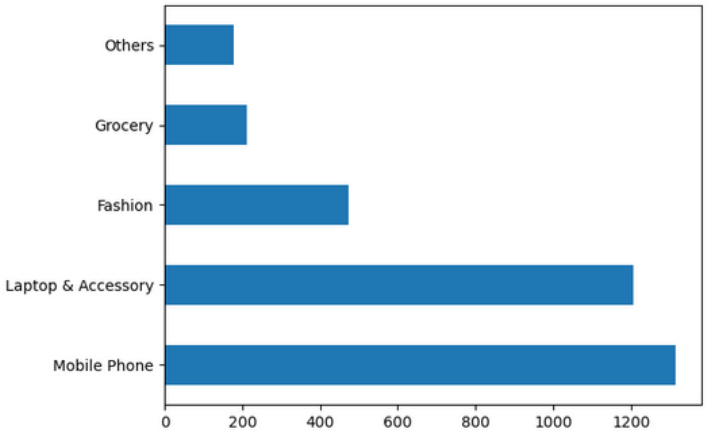
Checking their buying behaviors and difference in demographics:





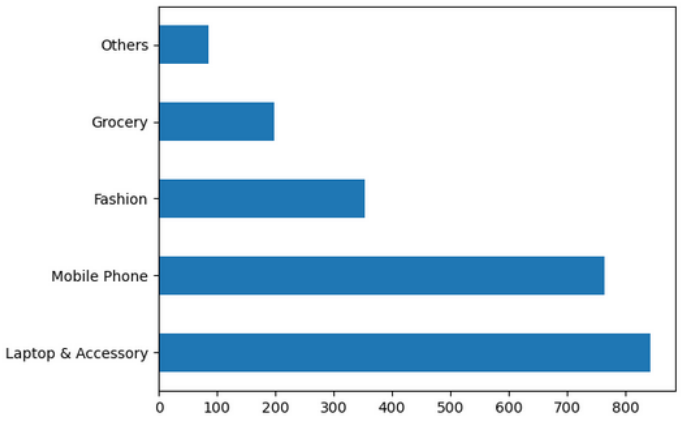

```
m['PreferedOrderCat'].value_counts().plot(kind='barh')
```

```
<AxesSubplot:>
```




```
f['PreferedOrderCat'].value_counts().plot(kind='barh')
```

```
<AxesSubplot:>
```



So, if the top category of **males is mobile phones**, what is the exact % that buys mobile phones over these other things?

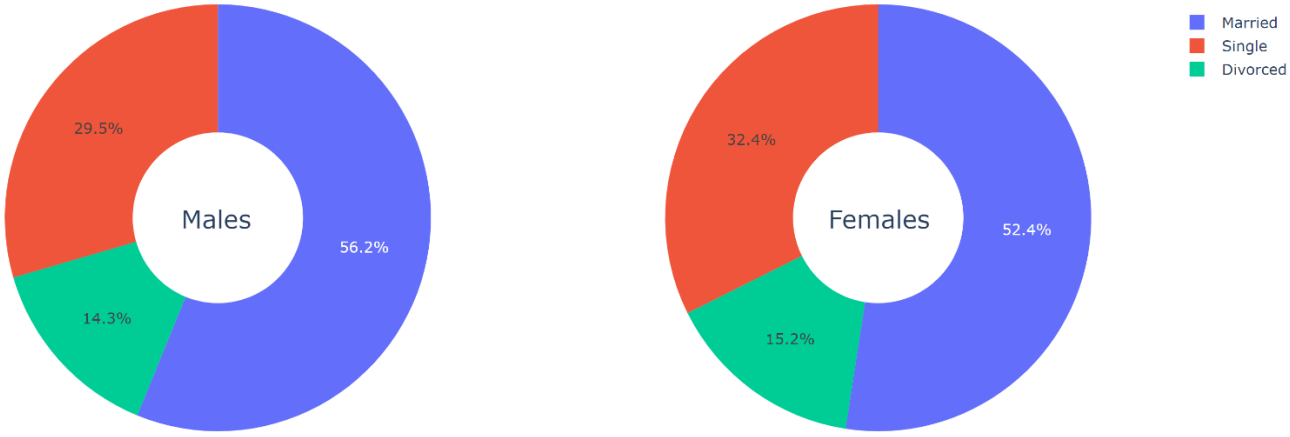
```
In [71]: ((m['PreferedOrderCat']=='Mobile Phone').sum()/males)*100

Out[71]: 38.88888888888889
```

For females, the **laptop and accessory** win the most preferred category

```
In [74]: #percentage of females who prefer Laptop & Accessory over other categories
         ((f['PreferedOrderCat']=='Laptop & Accessory').sum()/females)*100

Out[74]: 37.57791629563669
```

Almost 5% difference between males and females in their preference to Mobile Phones over other categories.

Martial state difference between both genders:



Order count by marital status for each gender:

In [77]:
```
#So, what is the order count for each martial state?
#males
m.groupby('MaritalStatus').agg(orders_count=('OrderCount','sum')).reset_index()
```

Out[77]:

	MaritalStatus	orders_count
0	Divorced	1513.0
1	Married	5753.0
2	Single	2709.0

In [78]:
```
#females
f.groupby('MaritalStatus').agg(orders_count=('OrderCount','sum')).reset_index()
```

Out[78]:

	MaritalStatus	orders_count
0	Divorced	1084.0
1	Married	3784.0
2	Single	2137.0

Median number of addresses for each gender:

In [81]:
```
print('Avg No of Addresses for Males: ', m['NumberOfAddress'].median())
print('Avg No of Addresses for Females: ', f['NumberOfAddress'].median())
```

```
Avg No of Addresses for Males:  3.0
Avg No of Addresses for Females:  3.0
```

Difference between complaints for each gender:

In [83]:
```
complain = (new_df['Complain'] == 1).sum()
no_complain =  (new_df['Complain'] == 0).sum()

males = (new_df['Gender']=="Male").sum()
females = (new_df['Gender']=="Female").sum()

comp_males = ((new_df['Complain'] == 1) & (new_df['Gender']=="Male")).sum()
comp_females = ((new_df['Complain'] == 1) & (new_df['Gender']=="Female")).sum()

males_comp_rate = (comp_males / complain)*100
females_comp_rate = (comp_females / complain)*100

print('From all complains:')
print( males_comp_rate, '% were males')
print(females_comp_rate, '% were females')
```

```
From all complains:
56.98254364089775 % were males
43.01745635910225 % were females
```

Difference in coupon usage:

```
m['no_coupon'] = m['CouponUsed']==0
m['coupon'] = m['CouponUsed']!=0

f['no_coupon'] = f['CouponUsed']==0
f['coupon'] = f['CouponUsed']!=0
```

```
print('%of males who havent used coupons vs all males', (m['no_coupon'].sum())/males *100)
print('%of females who havent used coupons vs all females', (f['no_coupon'].sum())/females *10
0)

print('-------------------------------------------------------')

print('%of males who have used coupons vs all males', (m['coupon'].sum())/males *100)
print('%of females who have used coupons vs all females', (f['coupon'].sum())/females *100)
```

```
%of males who havent used coupons vs all males 19.326241134751772
%of females who havent used coupons vs all females 19.01157613535174
-------------------------------------------------------
%of males who have used coupons vs all males 80.67375886524822
%of females who have used coupons vs all females 80.98842386464827
```

Difference in order amount hike:

In [88]:
```
#OrderAmountHikeFromlastYear
print(m['OrderAmountHikeFromlastYear'].mean())
print(f['OrderAmountHikeFromlastYear'].mean())
```

```
15.651004728132389
15.663401602849511
```

Difference in hours spent on app:

In [90]:
```
#HourSpendOnApp
print(m['HourSpendOnApp'].mean())
print(f['HourSpendOnApp'].mean())
```

```
2.911052009456265
2.93766696349065
```

Drawing conclusions on question 2:

Insights:

- Rate of churn in males is **17.7%** while the rate of churn in females is **15.4%**.
- Both prefer to login on **Mobile Phone** more.
- When they buy? How do they pay?

The first payment method being **Debit Cards** for both genders, then Credit Cards comes next.

- The preferred ordered Category is **Mobile Phones**, while for females it is **Laptop & Accessory**.

Almost **5% difference between males and females** in their preference to Mobile Phones over other categories.

- It appears that the **order count for Married Couples are the largest** sector for both males and females.
- The median number of addresses is **3** for both males and females
- There were **1604 complains** in our dataset where:

males' complains were **~57%**
where the females' are **~43%**

- For the coupons I divided the columns into those who used coupons and thos who haven't

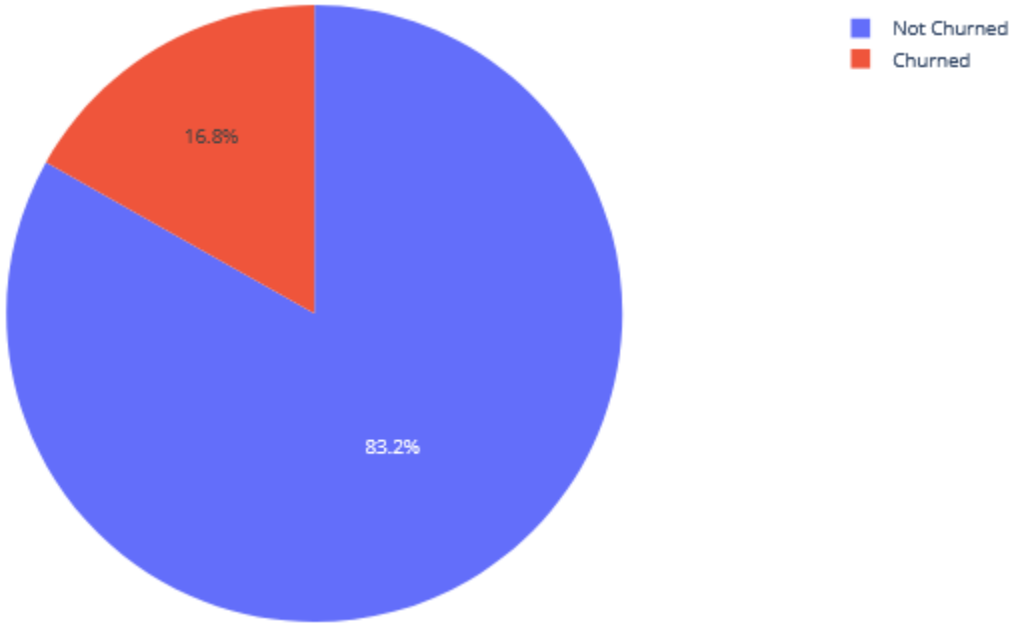
there are **4549 coupon users** while 1081 used zero coupons
and the difference in behaviour between the males and females is very little in that matter
only **0.3% diff** between those who used coupons between the two
Almost 81% for both used coupons, while 19% haven't

Q3: Provide key insights on why our customers churn and possible churn indicators

Checking distribution of churned vs non churned:

In [91]:

```
import plotly.express as px

Churn_values = new_df['Churn'].value_counts()
sizes = Churn_values.values

fig = px.pie(new_df, values=sizes, names=['Not Churned','Churned'] )
fig.show()
```



Checking every column to see if it affects the churn rate or not:

We conducted statistical tests to test the hypothesis of that there's a difference in churn between multiple groups/ between numerical variable.

Choosing a statistical test for categorical variables: (two categorical variables)

To compare the churn rates of different groups we used **Chi-square test** which is used to compare the proportion of categorical outcomes across multiple groups.

For example:

City Tier

In [92]:

```
# Create a contingency table of CityTier and Churn
contingency_table = pd.crosstab(df['CityTier'], df['Churn'])

# Print the contingency table
print(contingency_table)
print("\n")
# Conduct a chi-square test of independence
chi2, p_value, dof, expected = stats.chi2_contingency(contingency_table)

# Print the test statistics and p-value
print('Chi-square statistic:', chi2)
print('Degrees of freedom:', dof)
print('p-value:', p_value)
```

```
Churn        0    1
CityTier
1         3134  532
2          194   48
3         1354  368


Chi-square statistic: 40.982404247736355
Degrees of freedom: 2
p-value: 1.2612000812079956e-09
```

Since p-value is less than 0.05 We can reject the null hypothesis that there is no difference in churn rate between CityTier groups, and conclude that CityTier is likely an indicator of churn.

Choosing a statistical test for numerical variables: (one categorical variable and one numerical variable)

For skewed numerical variables: We used *Mann-Whitney U test* to compare the means or medians of numerical variables between the two groups (churned/not churned), because it doesn't assume normality of the data like t-test or ANOVA test. For low skewed data we used t-test.

For example:

Number of address

In [114]:

```
from scipy.stats import mannwhitneyu

# Generate two groups of non-normally distributed data
churned_column = new_df[new_df['Churn'] == 1]['NumberOfAddress']
non_churned_column = new_df[new_df['Churn'] == 0]['NumberOfAddress']

# Perform the Mann-Whitney U test
statistic, p_value = mannwhitneyu(churned_column, non_churned_column)

# Print the results
print("Mann-Whitney U test statistic:", statistic)
print("P-value:", p_value)
```

```
Mann-Whitney U test statistic: 2316556.5
P-value: 0.03049724563723119
```

Since p-value is less than 0.05 We can reject the null hypothesis that there is no difference in the average Number of addresses between churned and non-churned customers.

Checking churn rate for categorical variables which had statistically significant effect on churn:

Out[93]:

	CityTier	Churn
0	1	14.51
1	2	19.83
2	3	21.37

Out[96]:

	MaritalStatus	Churn
0	Divorced	14.79
1	Married	11.92
2	Single	26.59

Out[99]:

	Gender	Churn
0	Female	15.49
1	Male	17.73

Out[151]:

	Complain	Churn
0	0	10.93
1	1	31.67

Out[102]:

	PreferredLoginDevice	Churn
0	Computer	19.69
1	Mobile Phone	15.69

Out[105]:

	PreferredPaymentMode	Churn
0	Cash on Delivery	25.00
1	Credit Card	14.19
2	Debit Card	15.40
3	E wallet	22.61
4	UPI	18.11

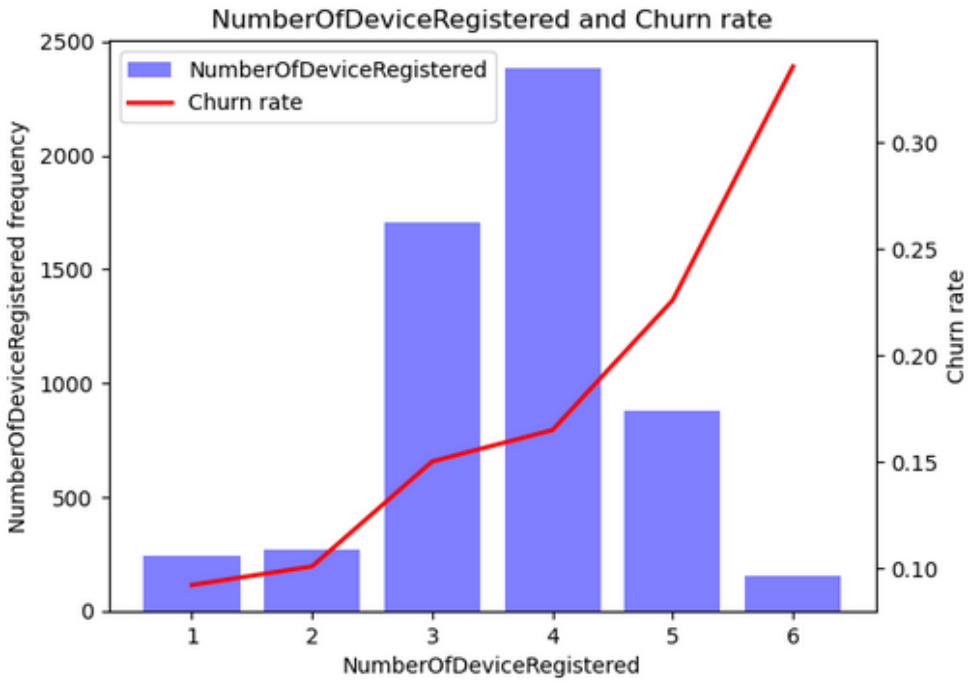
Out[118]:

	PreferedOrderCat	Churn
0	Fashion	15.50
1	Grocery	4.88
2	Laptop & Accessory	10.24
3	Mobile Phone	27.40
4	Others	7.58

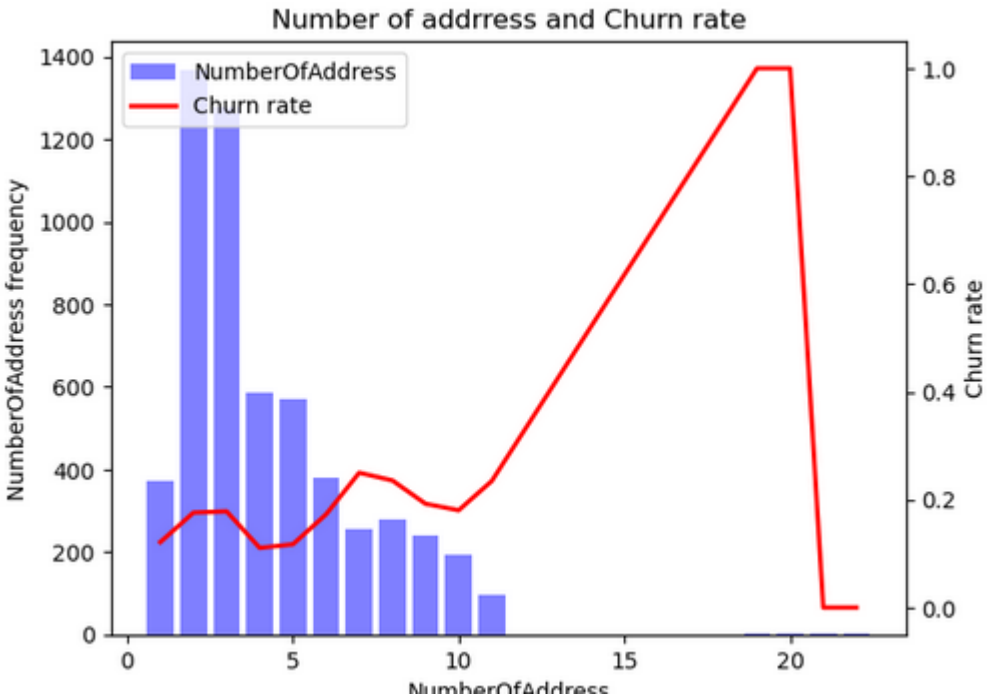
Checking churn rate for numerical variables which had statistically significant effect on churn:

Number of devices registered:

	NumberOfDeviceRegistered	Churn
0	1.0	9.21
1	2.0	10.07
2	3.0	15.01
3	4.0	16.48
4	5.0	22.58
5	6.0	33.55



Number of addresses:

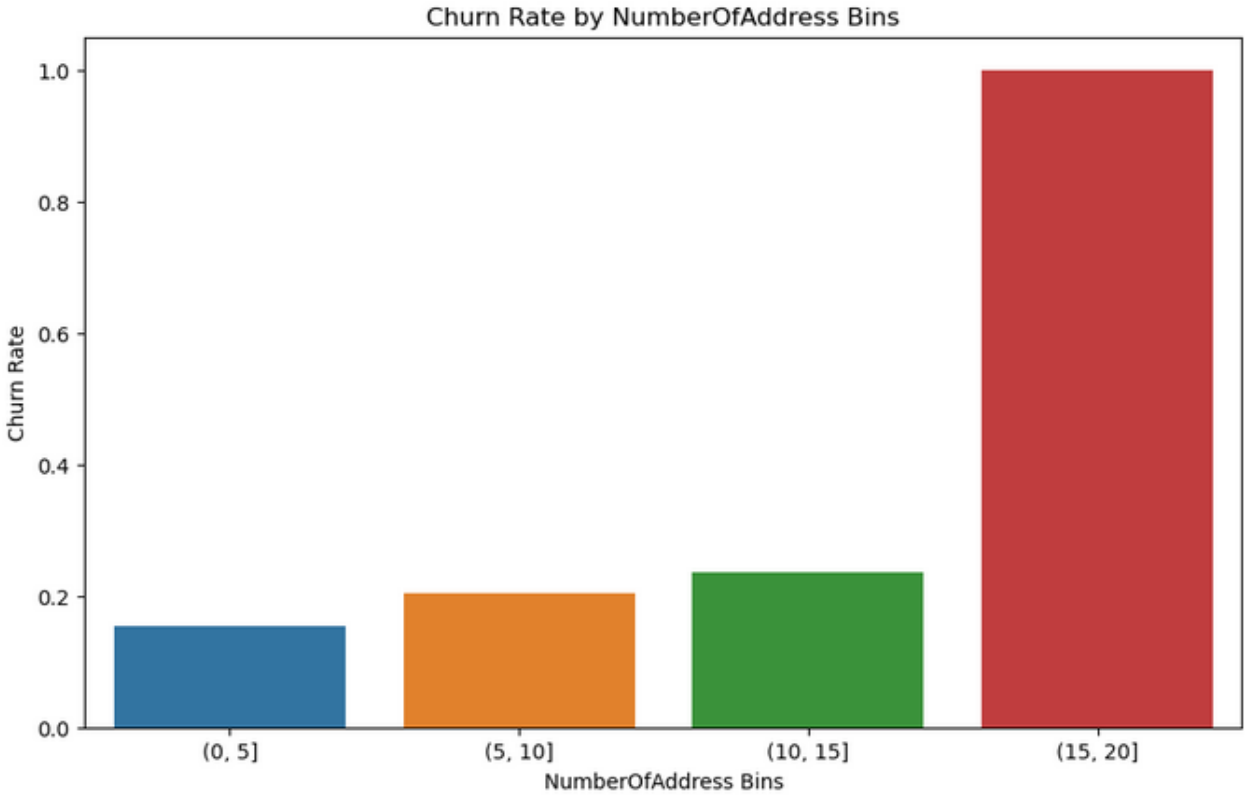


Binning number of addresses:

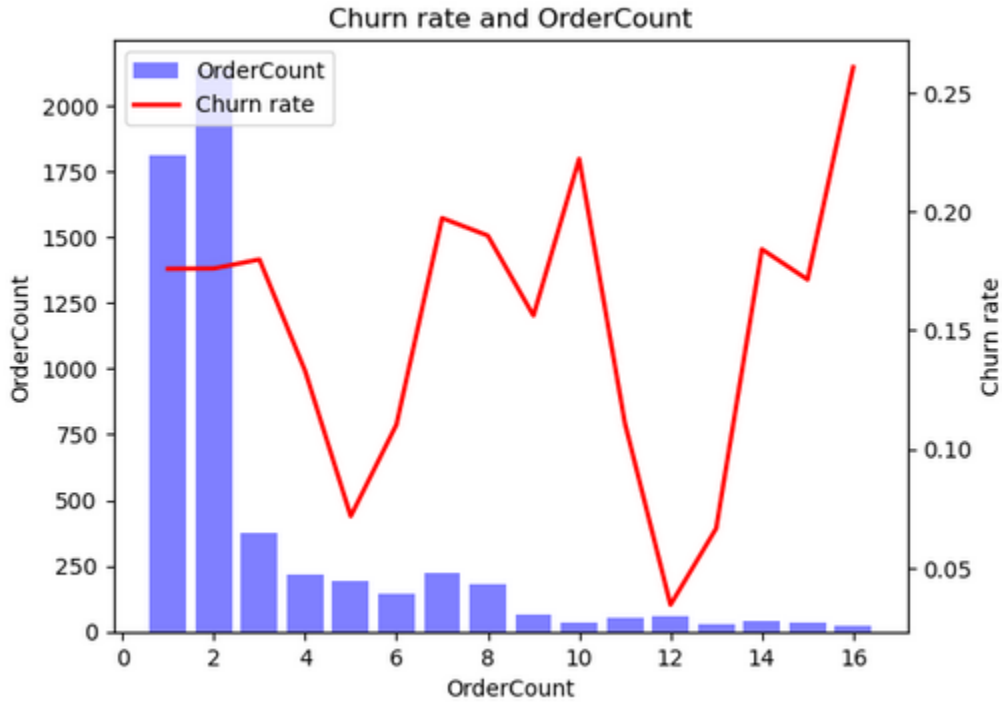
In [116]:

```
# Create bins for Tenure
bins = np.arange(0, new_df['NumberOfAddress'].max() + 1, 5)
new_df['NumberOfAddress_bins'] = pd.cut(new_df['NumberOfAddress'], bins)

# Calculate churn rate for each bin
churn_rate_by_bin = new_df.groupby('NumberOfAddress_bins')['Churn'].mean()
print(churn_rate_by_bin)

# Visualize churn rate by Tenure bin
plt.figure(figsize=(10, 6))
sns.barplot(x=churn_rate_by_bin.index, y=churn_rate_by_bin.values)
plt.ylabel('Churn Rate')
plt.xlabel('NumberOfAddress Bins')
plt.title('Churn Rate by NumberOfAddress Bins')
plt.show()
```

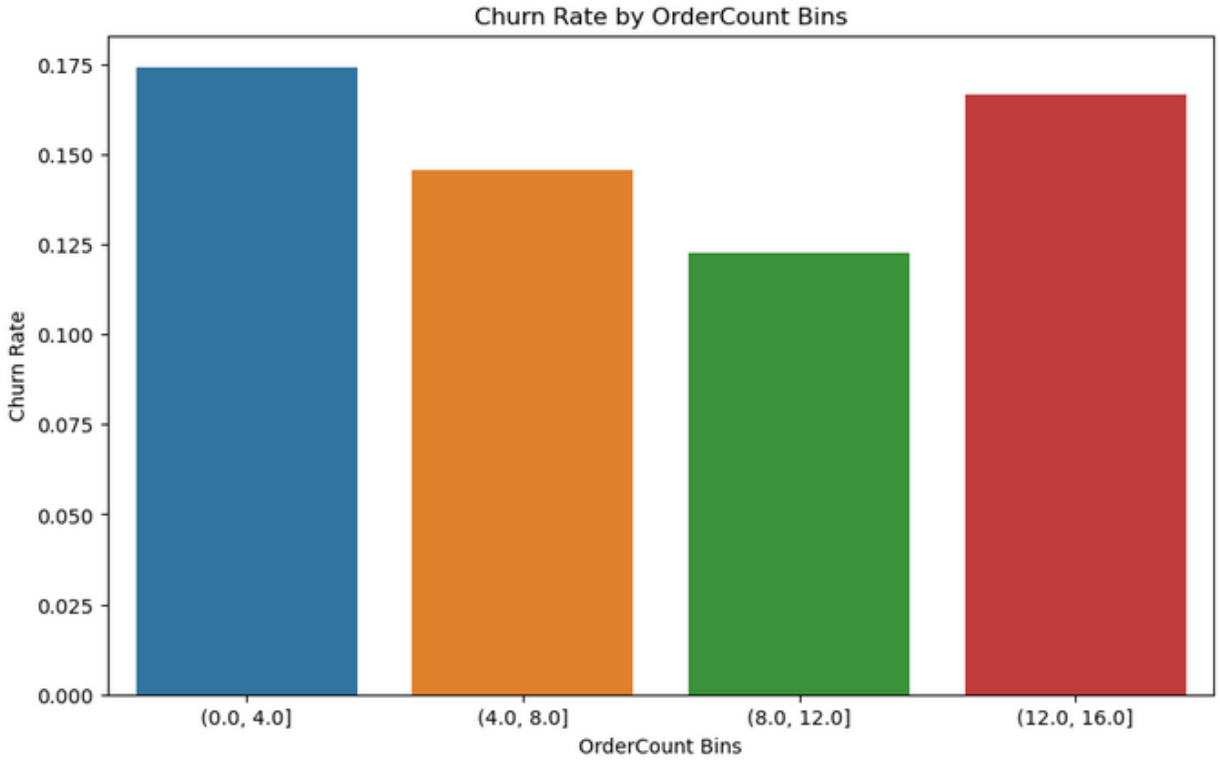
```
NumberOfAddress_bins
(0, 5]      0.154656
(5, 10]     0.205033
(10, 15]    0.234694
(15, 20]    1.000000
Name: Churn, dtype: float64
```



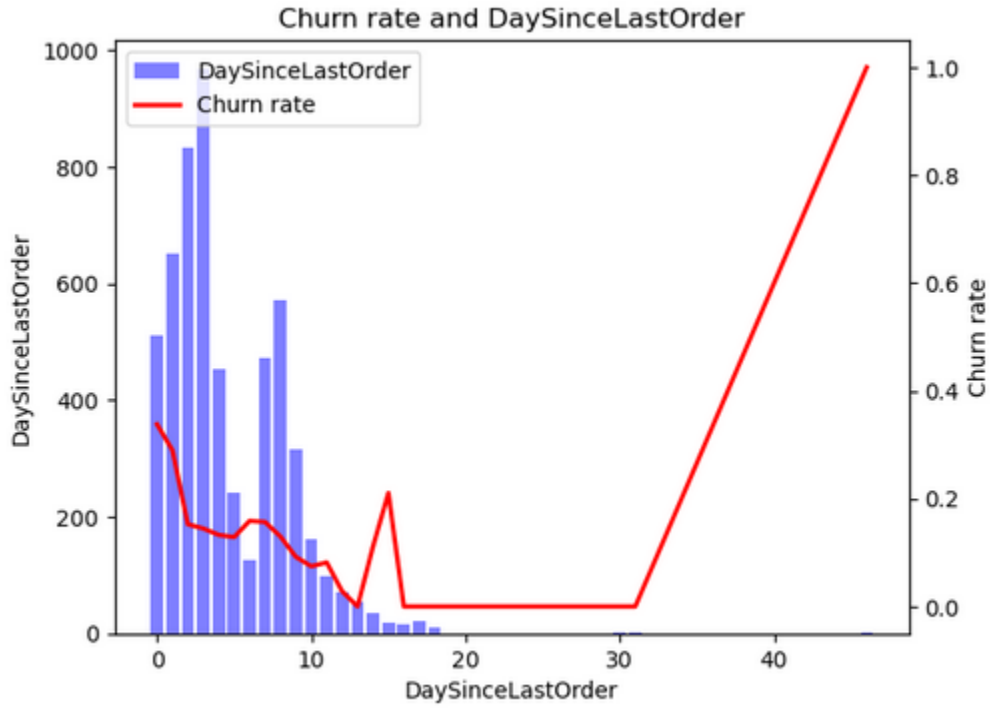
Order count:



Binning Order count:



Days since last order:



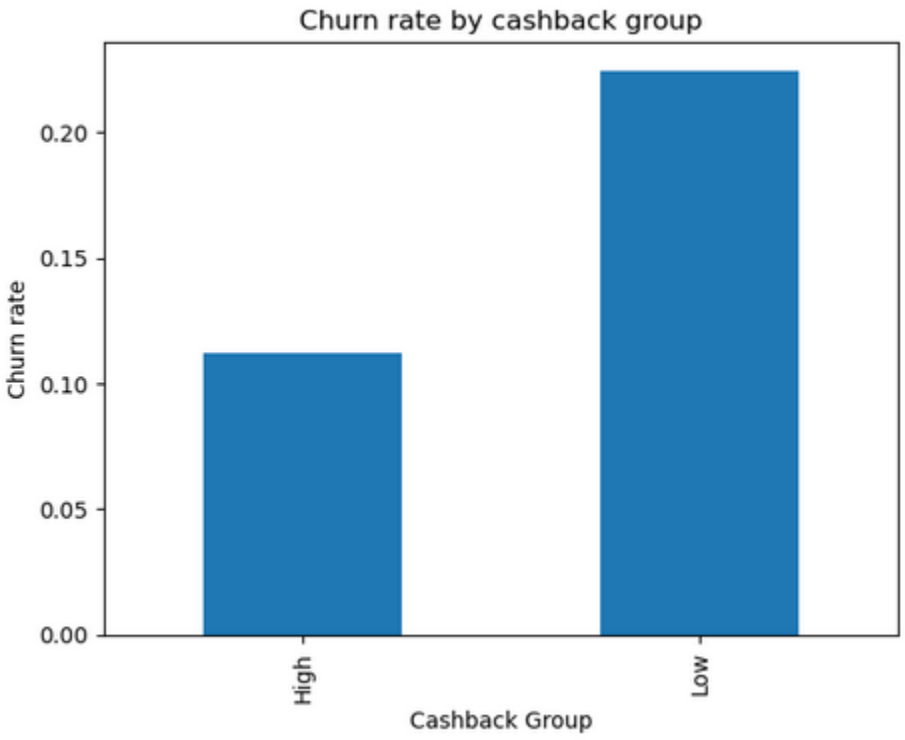
Cashback Amount

Splitting cashback amount into two categories (bases on the median)

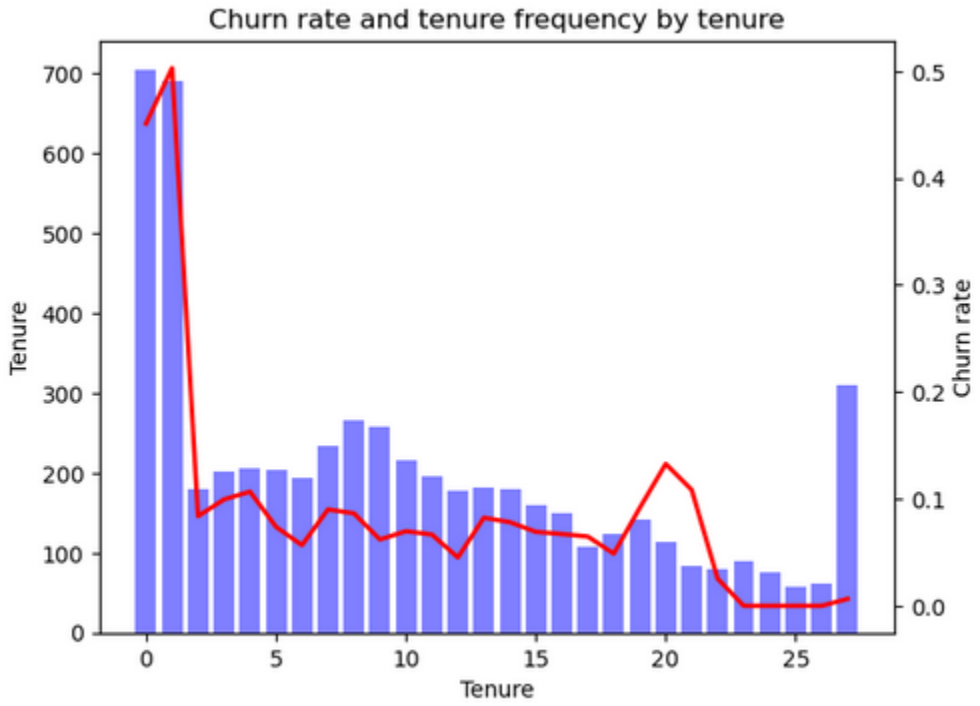
In [132]:

```
cashback_median = new_df['CashbackAmount'].median()
new_df['CashbackGroup'] = np.where(new_df['CashbackAmount'] <= cashback_median, 'Low', 'High')
churn_rate_by_cashback = new_df.groupby('CashbackGroup')['Churn'].mean()
print(churn_rate_by_cashback)

churn_rate_by_cashback.plot(kind='bar')
plt.xlabel('Cashback Group')
plt.ylabel('Churn rate')
plt.title('Churn rate by cashback group')
plt.show()
```

```
CashbackGroup
High    0.112256
Low     0.224512
Name: Churn, dtype: float64
```

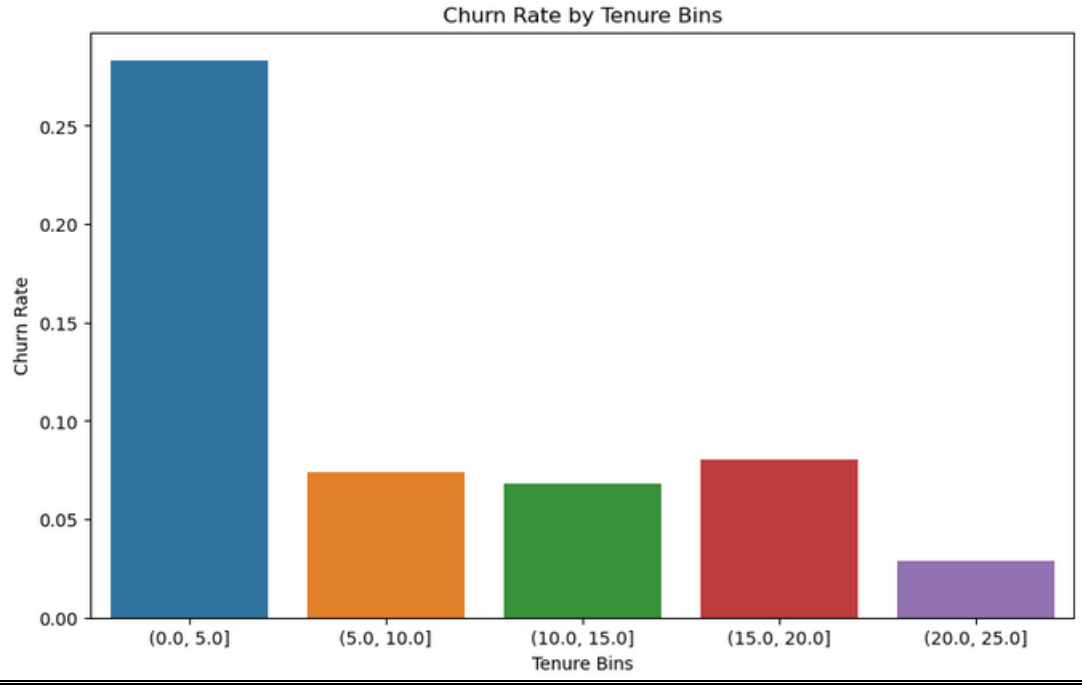


Tenure:

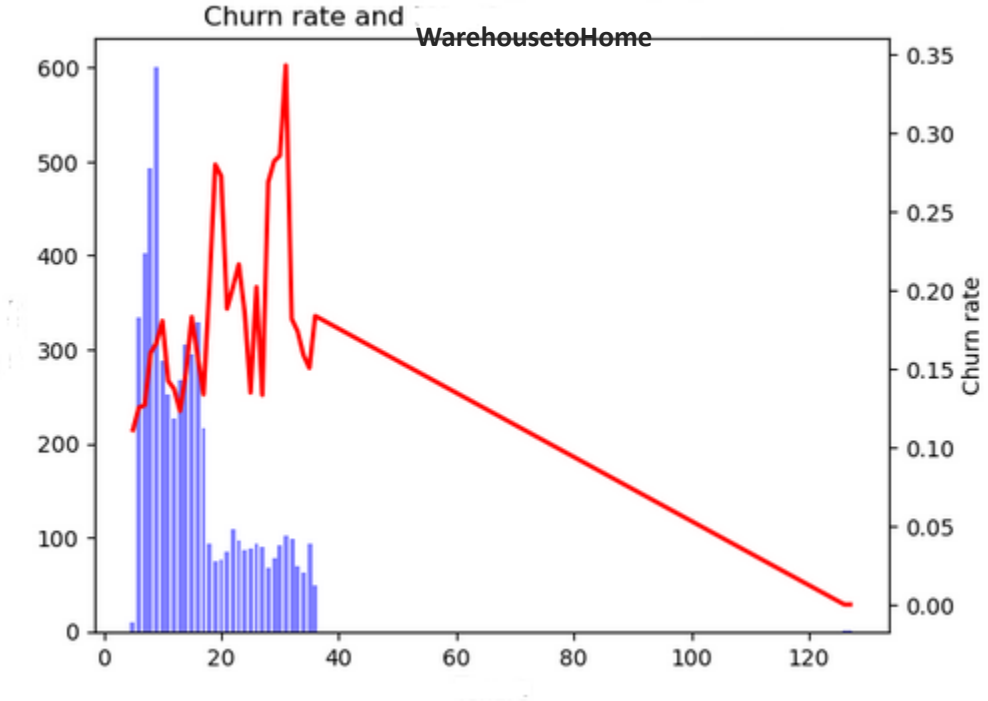


Binning Tenure:

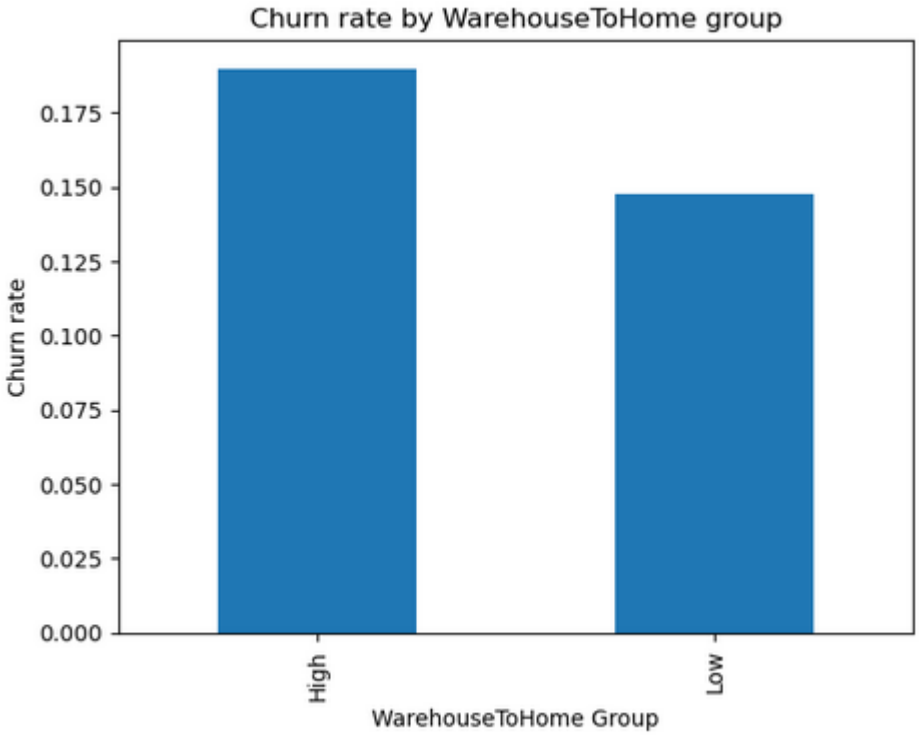
```
Tenure_bins
(0.0, 5.0]      0.283300
(5.0, 10.0]     0.073756
(10.0, 15.0]    0.068386
(15.0, 20.0]    0.080315
(20.0, 25.0]    0.028721
Name: Churn, dtype: float64
```



Warehouse to home:



Splitting warehouse to home into two categories low/high (by the median)



Satisfaction score:

In [148]:
```
SS_mean = round(new_df.groupby("SatisfactionScore")["Churn"].mean()*100,2).reset_index()
SS_mean
```

Out[148]:

	SatisfactionScore	Churn
0	1	11.40
1	2	12.67
2	3	17.28
3	4	17.08
4	5	23.78

Using feature selection to determine which variables has the most effect on the target(churn):

In [160]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

x = data_enc.drop(['Churn'], axis = 1)
y = data_enc['Churn']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 42, strati
fy= y )

from sklearn.feature_selection import SelectFromModel
sel = SelectFromModel(RandomForestClassifier(n_estimators = 100))
sel.fit(x_train, y_train)
```

Out[160]:
```
SelectFromModel(estimator=RandomForestClassifier())
```

In [162]:
```
selected_feat= x_train.columns[(sel.get_support())]
len(selected_feat)
```

Out[162]:
```
8
```

In [163]:
```
selected_feat
```

Out[163]:
```
Index(['SatisfactionScore', 'NumberOfAddress', 'Complain', 'CashbackAmount',
       'DaySinceLastOrder', 'OrderAmountHikeFromlastYear', 'Tenure',
       'WarehouseToHome'],
      dtype='object')
```

Drawing conclusions on question 3:

Churn Indicators:

- City Tier: Where **City Tier 2 & 3** has the most churn rate **~20%**.
- Marital Status: Where **Single customers** are the most churned customers with **~27%** Churn rate.
- Gander: **Males** are most likely to churn than Females, where males has **~18%** Churn rate.
- Preferred Login Device: **Computer Users** are more likely to churn than mobile useres, where computer users have **~20%** Churn rate.
- Preferred Payment Mode: **Cash on Dileviry and E wallet** users are more likely to churn, with churn rate **25%, 22%** respectively.
- NumberOfDeviceRegistered: Churn rate increases as number of devices increases, where customers with **6 devices registers have 33%** Chrun rate.
- NumberOfAddress: Churn **rate increases by number of address increases**, although we cannot conclude that high addresses cause churn because number of customers with high addresses are very little.
- PreferredOrderCategory: **Mobile category** is the most preferred category with churn rate **27%**.
- OrderCount: Order count is an indicator of churn although there's no obvious pattern to indicate how it affect the churn rate.
- DaySinceLastOrder: DaySinceLastOrder is an indicator of churn, there's no obvious pattern but we can say **churn rate increase after 15 Days**.
- CashbackAmount: Customers with **low cashback amount** from last month tends to churn more than customers with high cashback amount.
- Tenure: Customers with **low tenure (0 to 5)** in the organization have the highest churn rate.
- WarehouseToHome: Customers with **far WarehouseToHome** are more likely to churn than customers with near WarehouseToHome.
- SatisfactionScore: Customers with **5 satisfaction score** have the highest churn rate **~23%**.
- Complain: Customers that **raised complains** in the past month have higher churn rate **(~32%)** compared to who didn't.

According to feature selection randm forest the most variables that affect churn are: NumberfAddress -Complain -CashbackAmount -DaySinceLastOrder -Tenure -WarehouseToHome

Q4: Analyze the distance between the warehouse and the customer's home and check if it's related to complains?

Using a statistical test to check the relation:

Using *Mann-Whitney U test* to check if there's difference in mean of warehouse to home between customers who complains and others who don't

```
P-value:  0.039809647504852365
```

Based on the Mann-Whitney U test,we conclude that there is significant difference in the mean WarehouseToHome distance between customers who complain and those who do not complain.

Checking the difference between averages:

In [167]:
```
grouped_data = new_df.groupby('Complain')['WarehouseToHome']
grouped_data.agg(['mean', 'median', 'std'])
```

Out[167]:

	mean	median	std
Complain			
0	15.313959	13.0	8.576889
1	15.709476	13.0	8.188310

Checking the direction and strength of the correlation:

We used point **biserial correlation** which measures the relationship between a binary variable and a continuous variable

In [168]:
```
# point-biserial correlation coefficient, which measures the relationship between a binary variable an
d a continuous variable
from scipy.stats import pointbiserialr

correlation, _ = pointbiserialr(new_df['Complain'], new_df['WarehouseToHome'])
print('Correlation coefficient:', correlation)
```

```
Correlation coefficient: 0.021081146885559874
```

Drawing conclusion on q4:

WarehouseToHome is related to the complains but the correlation between them is **positive** (where far WarehouseToHome is more associated with complains) but it's **weak** and not noticeable.

Q5: Does the number of addresses added by customers impact the churn rate?

As answered in Q3, NumberOfAddress Churn rate increases as number of address increases, although we cannot conclude that high addresses cause churn because number of customers with high addresses are very few.

Q6: Models

Encoding nominal categorical variables using one hot encoding:

One-hot encoding is a technique used in machine learning to convert categorical data into numerical data that can be used in various algorithms.

In one-hot encoding, each categorical variable is converted into a binary vector of size n, where n is the number of categories. The vector contains all zeros except for the position corresponding to the category, which is set to 1.

In [156]:
```
cat_data = new_df.select_dtypes(include='object')

#encode categorical variables and add it to the normal dataset
encoded = pd.get_dummies(cat_data,drop_first=True)

data_enc = pd.concat([new_df.drop(cat_data.columns, axis=1), encoded], axis=1)
```

1 – nominal variables have “object” data type , so we have selected them using “select_dtypes()” function

2- pd.get_dummies() : is function takes a Pandas DataFrame as input and converts all categorical columns in the DataFrame into a set of binary columns using on-hot encoding . It returns a new DataFrame with binary columns for each unique category in the original categorical column

3- then concat the 2 dataframe : data frame that include encoded data and dataframe that contain the rest of the variables

Splitting data:

]:
```
#selecting features and target variable, and splitting the data
X=data_enc.drop(['Churn'],axis=1)
y=data_enc['Churn']

# Split the dataset into training and temporary sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3 ,stratify = y ,random_state=42)

# Split the temporary set into validation and test sets
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

We split data into 3 sets ; we have worked on training set and validation set , and when we choose the best model “based on f1-score” we apply it on the test set to check its performance .

Training set : 70%

Validation set : 15%

Test Set : 15%

random_state: random seed used for reproducibility

stratify : allow us to perform stratified sampling during the data split to ensure that the proportion of classes in the original dataset is preserved in both the training and test and validation sets since our data set is imbalanced

Training the models:

Decision Tree

+ Code + Markdown

```
Decision_Tree_model = DecisionTreeClassifier(max_depth = 4 , random_state=42)
Decision_Tree_model.fit(X_train , y_train)

# Evaluate the model on the validation set
y_pred_val = Decision_Tree_model.predict(X_val)
print('F1 Score:', round(f1_score(y_val, y_pred_val),2))
```

```
F1 Score: 0.61
```

Random Forest

```
# creating a RF classifier
Random_Forest_model = RandomForestClassifier(max_depth =4, random_state=42)
Random_Forest_model.fit(X_train, y_train)

# Evaluate the model on the validation set
y_pred_val = Random_Forest_model.predict(X_val)
print('F1 Score:', round(f1_score(y_val, y_pred_val),2))
```

```
F1 Score: 0.28
```

AdaBoostClassifier

```
AdaBoost_model = AdaBoostClassifier(n_estimators=200, random_state=42)
AdaBoost_model.fit(X_train, y_train)

# Evaluate the model on the validation set
y_pred_val = AdaBoost_model.predict(X_val)
print('F1 Score:', round(f1_score(y_val, y_pred_val),2))
```

```
F1 Score: 0.67
```

We have used 3 classifers : Decision Tree , Random Forest , Adaboost

When we evaluating the model we have used f1-score because :

1 – we could not use accuracy since the classes are imbalanced . This is because accuracy can be misleading when the classes are imbalanced , since a high accuracy score can be achieved simply by predicting the majority class

2- f1-score would be a good metric because we wanted to balance both precision and recall

Precision : “reducing false positive”; predicting a customer will churn when they actually won’t

REACLL : “ Reducing false negative “ ; predicting customer won’t churn when they actually will

Hyper parameter tuning:

RandomizedSearchCV :-

Is a hyperparameter tuning technique that helps optimize the performance of a machine learing model. It works by randomly selecting combinations of hyperparameters from a defined search space and evaluating the model’s performance using cross-validation. It allows you to efficiently search the hyperparameter space and find the best set of hyperparameters for your model

[361]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]

# Number of features to consider at every split
max_features = ['auto', 'sqrt']

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(4, 20, num =16)]

# Minimum number of samples required to split a node
min_samples_split = randint(2, 20)

# Minimum number of samples required at each leaf node
min_samples_leaf = randint(1, 20)

# Method of selecting samples for training each tree
bootstrap = [True, False]

criterion = ["gini", "entropy"]
```

- We define a search space fot the hyperparameters we want to tune , such as the number of estimators , maximum depth , Number of features to consider at every split , Maximum number of levels in tree (max_depth) , minimum number of levels in tree , minimum number od samples required to split a node minimum number of samples required at each leaf node , method for training samples for training each tree , gini or entropy .

- we use the randint function from scipy.stats to sample the number of estimators from a uniform distribution between2 and 20 , 1 and 20

Decision Tree

```
# Define the parameter space for the randomized search
param_dist = {
    "max_depth": max_depth,
    "min_samples_split": min_samples_split,
    "min_samples_leaf": min_samples_leaf,
    "criterion": criterion,
    "max_features" :max_features
}

# Create a Decision Tree classifier
clf = DecisionTreeClassifier(max_depth = 4,random_state=42)

# Perform the randomized search with 5-fold cross-validation
#search across 100 different combinations, and use all available cores

n_iter_search = 10
random_search = RandomizedSearchCV(clf, param_distributions=param_dist,
                                   n_iter=n_iter_search, cv=5, random_state=42, scoring='f1_macro')
random_search.fit(X_train, y_train)

# Print the best hyperparameters and the corresponding F1 score on the test set
print(f"Best hyperparameters: {random_search.best_params_}")
y_pred = random_search.predict(X_val)
f1 = f1_score(y_val, y_pred, average='macro')
print(f"F1 score on validation set: {f1:.2f}")
```

```
Best hyperparameters: {'criterion': 'gini', 'max_depth': 11, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split': 9}
F1 score on validation set: 0.81
```

We then create a RandomizedSearchCV object and pass the Decision Tree classifier , the hyperparameter search space , number of random searches and cross-validation folds . We set the scoring parameter to “f1_macro” to optimize for the macro-averaged f1-score .

F1-score = 0.87

Random Forest

```
param_dist = {
    "n_estimators":n_estimators ,
    "max_depth": max_depth,
    'min_samples_split': min_samples_split,
    "min_samples_leaf":min_samples_leaf ,
    "criterion": criterion,
    'max_features': max_features,
    'bootstrap': bootstrap
}

# Create a Random Forest classifier
clf = RandomForestClassifier(max_depth = 4,random_state=42)

# Perform the randomized search with 5-fold cross-validation
n_iter_search = 10
random_search = RandomizedSearchCV(clf, param_distributions=param_dist,
                                   n_iter=n_iter_search, cv=5, random_state=42, scoring='f1_macro')
random_search.fit(X_train, y_train)

# Print the best hyperparameters and the corresponding F1 score on the test set
print(f"Best hyperparameters: {random_search.best_params_}")
y_pred = random_search.predict(X_val)
f1 = f1_score(y_val, y_pred, average='macro')
print(f"F1 score on validation set: {f1:.2f}")
```

```
Best hyperparameters: {'bootstrap': True, 'criterion': 'entropy', 'max_depth': 11, 'max_features': 'sqrt', 'min_samples_leaf': 3, 'min_sample
s_split': 3, 'n_estimators': 1600}
F1 score on validation set: 0.85
```

F1-score = 0.85

AdaBoost Classifier

```
# Define the parameter space for the randomized search
param_dist = {
    "base_estimator__max_depth": [1, 2, 3, 4],
    "n_estimators": randint(50, 200),
    "learning_rate": [0.1, 0.5, 1.0, 2.0]
}

# Create an AdaBoost classifier with decision stumps as the weak learners
clf = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), random_state=42)

# Perform the randomized search with 5-fold cross-validation
n_iter_search = 10
random_search = RandomizedSearchCV(clf, param_distributions=param_dist,
                                   n_iter=n_iter_search, cv=5, random_state=42, scoring='f1_macro')
random_search.fit(X_train, y_train)

# Print the best hyperparameters and the corresponding F1 score on the test set
print(f"Best hyperparameters: {random_search.best_params_}")
y_pred = random_search.predict(X_val)
f1 = f1_score(y_val, y_pred, average='macro')
print(f"F1 score on validation set: {f1:.2f}")
```

```
Best hyperparameters: {'base_estimator__max_depth': 3, 'learning_rate': 1.0, 'n_estimators': 121}
F1 score on validation set: 0.89
```

F1-score = 0.92

- Finally , we decided to go with adaboost classifier since it gives as the best f1-score

Try Adaboost classifer on the test set :

Predection on test set

```
y_pred_test = random_search.predict(X_test)
print('F1 Score:', round(f1_score(y_test, y_pred_test),2))
```

```
F1 Score: 0.83
```

We get f-score = 0.83

General insights & KPIS :

NPS Score:

```
promoters = len(new_df[new_df['SatisfactionScore'] >= 4])
detractors = len(new_df[new_df['SatisfactionScore'] <= 2])
neutrals = len(new_df[(new_df['SatisfactionScore'] == 3)])

# Calculate the total number of responses
total_responses = len(new_df)

# Calculate the NPS score
nps_score = 100 * (promoters - detractors) / total_responses

# Print the results
print('Promoters: ', promoters)
print('Detractors: ', detractors)
print('Neutrals: ', neutrals)
print('Total Responses: ', total_responses)
print('NPS Score: ', nps_score)
```

```
Promoters:  2149
Detractors:  1716
Neutrals:  1765
Total Responses:  5630
NPS Score:  7.690941385435169
```

NPS score is so low as according to industry benchmarks, a good NPS score for an ecommerce business is typically around 50 or higher. Scores above 70 are considered excellent, while scores below 30 are considered poor.

Active user rate

```
active_users = (new_df['HourSpendOnApp'] >= 3).sum()
active_user_rate = (active_users / len(new_df['HourSpendOnApp']) * 100)
print(active_user_rate)
```

```
71.56305506216695
```

Customer Complaint Rate

```
num_complained = (new_df['Complain']==1).sum()

num_customers = len(df.index)

complaint_rate = (num_complained / num_customers) * 100

print(complaint_rate)
```

```
28.49023090586146
```

Business should work on lowering the complaint rate