Customer Churn Analysis

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

Data source: https://www.kaggle.com/datasets/ankitverma2010/ecommerce-customer-churn-analysis-and-prediction/data

1. Data Loading:

```
c=pd.read_csv('/content/customer .csv')
```

```
c_copy=c.copy() # copy of the original dataset is created which can be used for future references
```

```
c.head(2) # reading first 2 rows
```

	CustomerID	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	Prefer
0	50001	1	4.0	Mobile Phone	3	6.0	
1	50002	1	NaN	Phone	1	8.0	

```
c.tail(2) #reading last 2 rows
```

	CustomerID	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	Pre
5628	55629	0	23.0	Computer	3	9.0	
5629	55630	0	8.0	Mobile Phone	1	15.0	

2. Exploratory Data Analysis

```
c.shape # reading dimensions of the dataset
```

```
(5630, 20)
```

```
c.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   int64  
dtypes: float64(7), int64(8), object(5)
memory usage: 879.8+ KB
```

```
c.describe() # statistical summary
```

	CustomerID	Churn	Tenure	CityTier	WarehouseToHome	HourSpend(
count	5630.000000	5630.000000	5366.000000	5630.000000	5379.000000	5375.0(
mean	52815.500000	0.168384	10.189899	1.654707	15.639896	2.9:
std	1625.385339	0.374240	8.557241	0.915389	8.531475	0.7:
min	50001.000000	0.000000	0.000000	1.000000	5.000000	0.0(
25%	51408.250000	0.000000	2.000000	1.000000	9.000000	2.0(
50%	52815.500000	0.000000	9.000000	1.000000	14.000000	3.0(
75%	54222.750000	0.000000	16.000000	3.000000	20.000000	3.0(
max	55630.000000	1.000000	61.000000	3.000000	127.000000	5.0(

```
c.columns
```

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

```
c.nunique() # count of unique values per column
```

```
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                  220
dtype: int64
```

a. No.of customers based on login device

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

```
Mobile Phone    2765
Computer        1634
Phone           1231
Name: PreferredLoginDevice, dtype: int64
```

Most of the customers prefer mobile phone

b. Customers based on gender

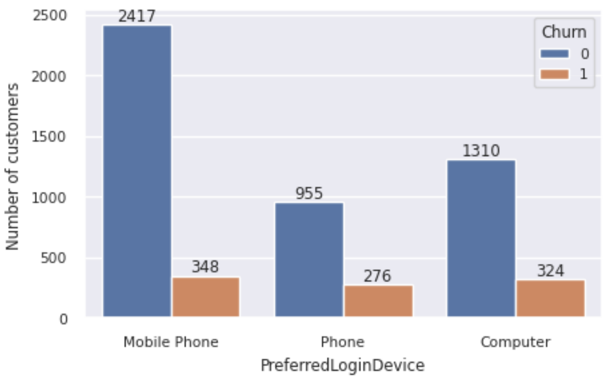
```
c['Gender'].value_counts()
```

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

Comparatively, male customers are more

c. customers churned based on login device

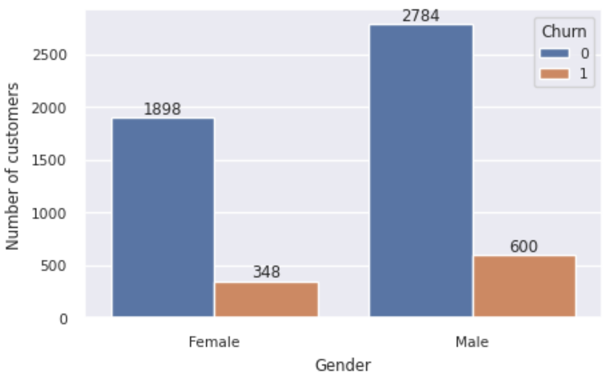
```
plt.figure(figsize=(5,3))
sns.set(font_scale=0.7)
s=sns.countplot(data=c,x='PreferredLoginDevice',hue='Churn')
plt.ylabel('Number of customers')
for i in s.containers:
  s.bar_label(i)
```



Only small amount of customers have churned

d. gender vs churn

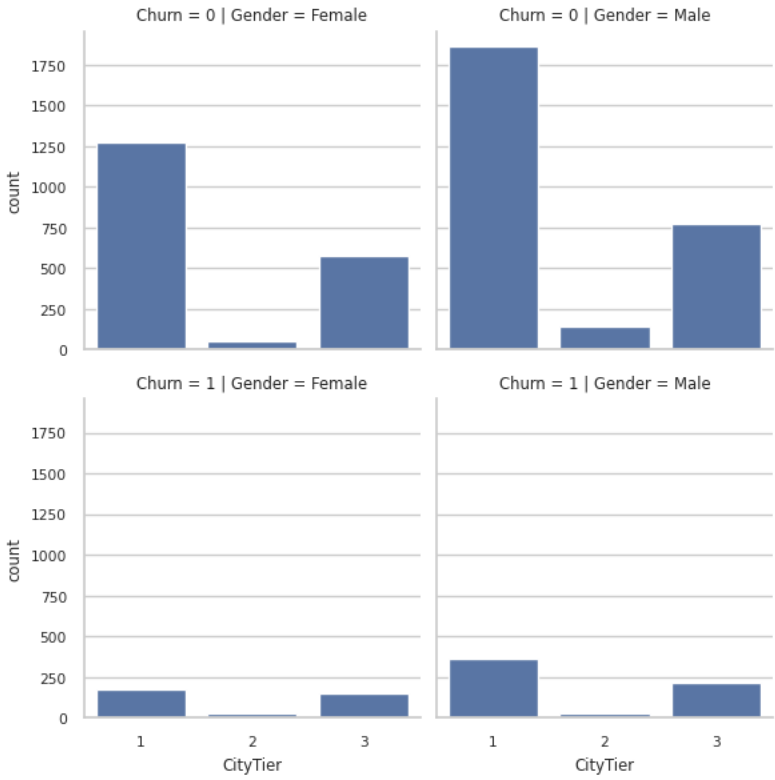
```
plt.figure(figsize=(5,3))
sns.set(font_scale=0.7)
s=sns.countplot(data=c,x='Gender',hue='Churn')
plt.ylabel('Number of customers')
for i in s.containers:
  s.bar_label(i)
```



e. customers churned from different cities

```
sns.set_style('whitegrid')
f=sns.FacetGrid(data=c,col ='Gender',row='Churn')
f.map_dataframe(sns.countplot,x='CityTier')
```

```
<seaborn.axisgrid.FacetGrid at 0x7c9cde55c670>
```

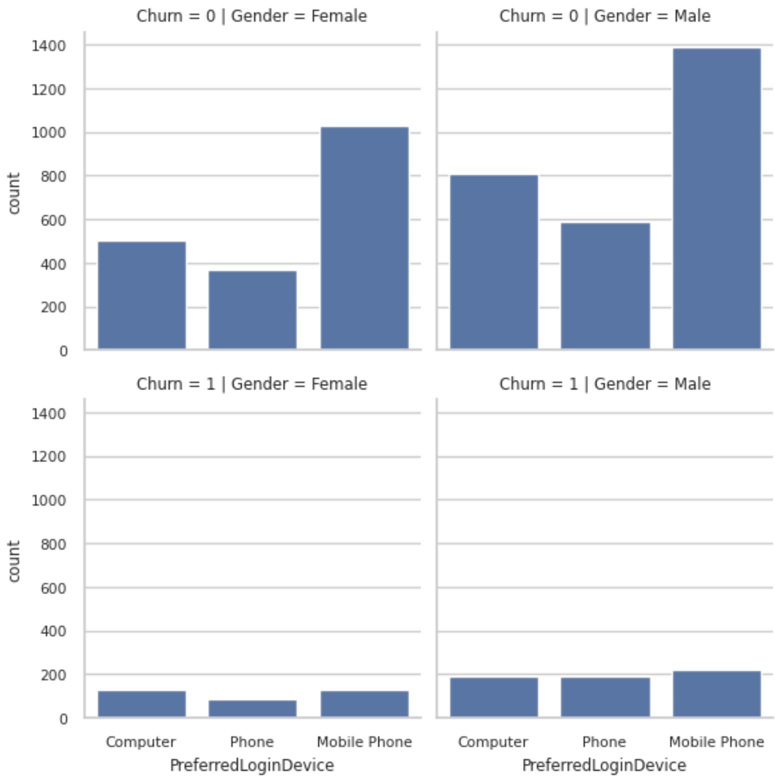


City 1 has more customers than from other 2 city tiers

f. customers churned from different devices used

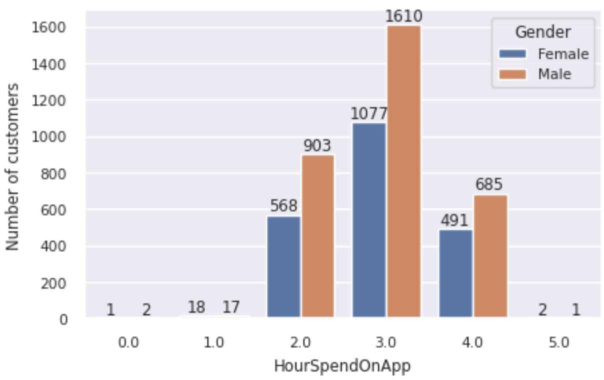
```
sns.set_style('whitegrid')
h=sns.FacetGrid(data=c,col='Gender',row='Churn')
h.map_dataframe(sns.countplot,x='PreferredLoginDevice')
```

```
<seaborn.axisgrid.FacetGrid at 0x7c9cde193d90>
```



g. time spent on app by different customers based on their gender

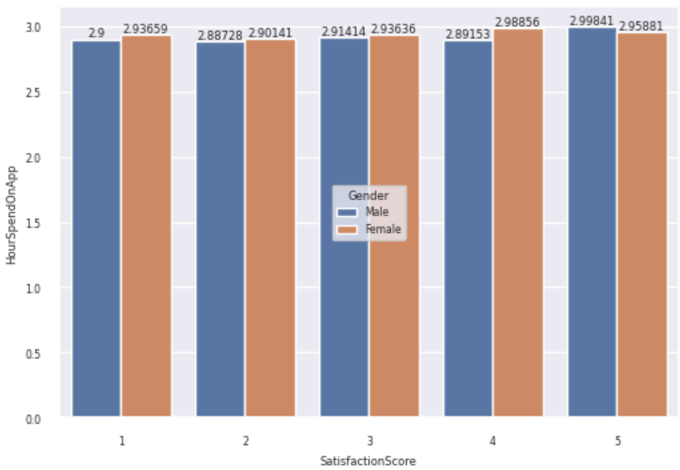
```
plt.figure(figsize=(5,3))
sns.set(font_scale=0.7)
s=sns.countplot(data=c,x='HourSpendOnApp',hue='Gender')
plt.ylabel('Number of customers')
for i in s.containers:
  s.bar_label(i)
```



Maximum hours spent is 4 , but 3 hrs is the maximum duration spent by each customer

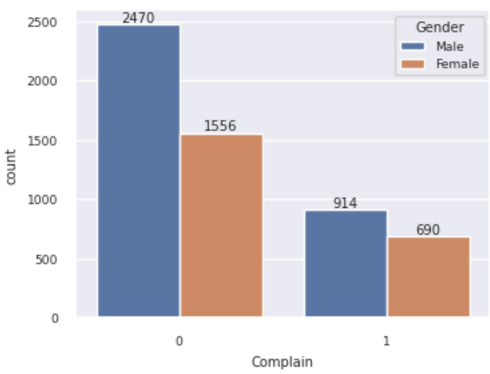
h. time spent vs satisfaction score

```
plt.figure(figsize=(6,4))
sns.set(font_scale=0.5)
s=sns.barplot(data=c,x='SatisfactionScore',y='HourSpendOnApp',hue='Gender',errorbar=None)
sns.move_legend(s,"center")
for i in s.containers:
  s.bar_label(i)
```



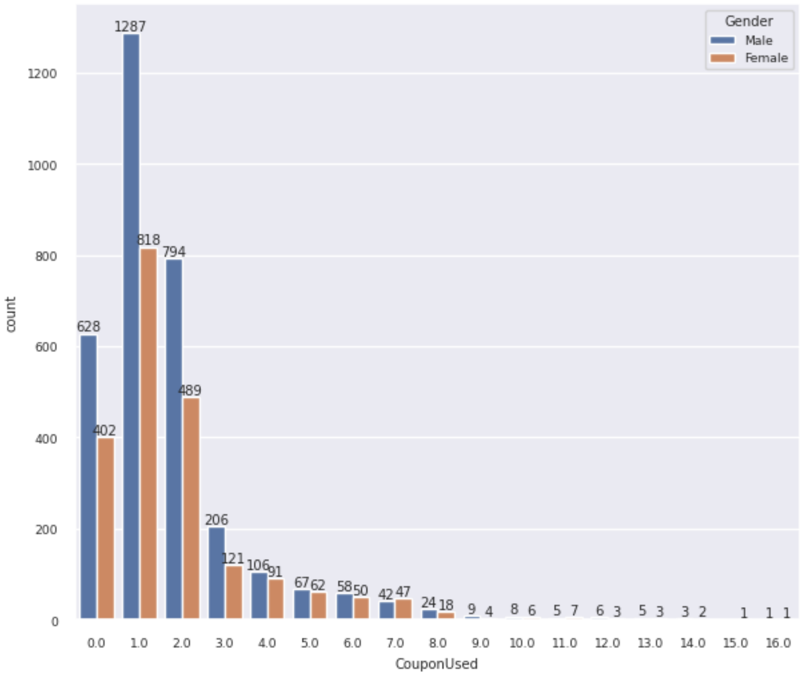
i.no.of complains raised by customers

```
plt.figure(figsize=(4,3))
sns.set(font_scale=0.6)
s=sns.countplot(data=c,x='Complain',hue='Gender')
for i in s.containers:
  s.bar_label(i)
```



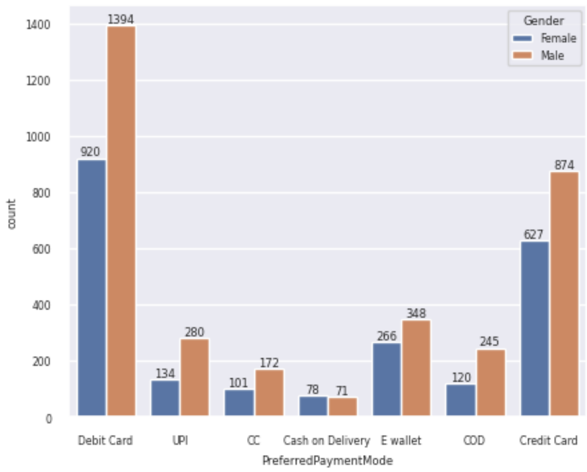
j. coupons used

```
plt.figure(figsize=(7,6))
sns.set(font_scale=0.6)
s=sns.countplot(data=c,x='CouponUsed',hue='Gender')
for i in s.containers:
  s.bar_label(i)
```



k. payment mode vs gender

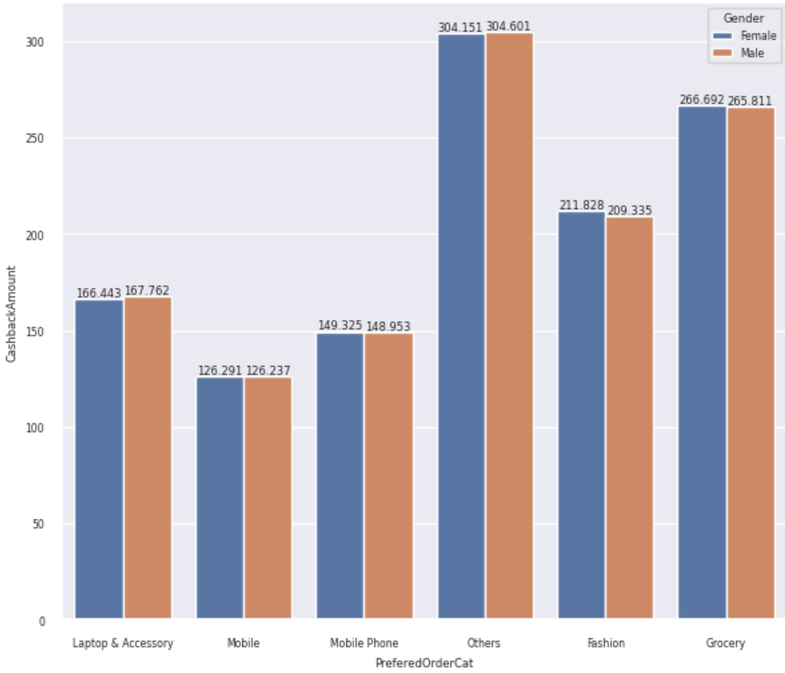
```
plt.figure(figsize=(5,4))
sns.set(font_scale=0.5)
s=sns.countplot(data=c,x='PreferredPaymentMode',hue='Gender')
for i in s.containers:
  s.bar_label(i)
```



Majority prefers debit card for payment

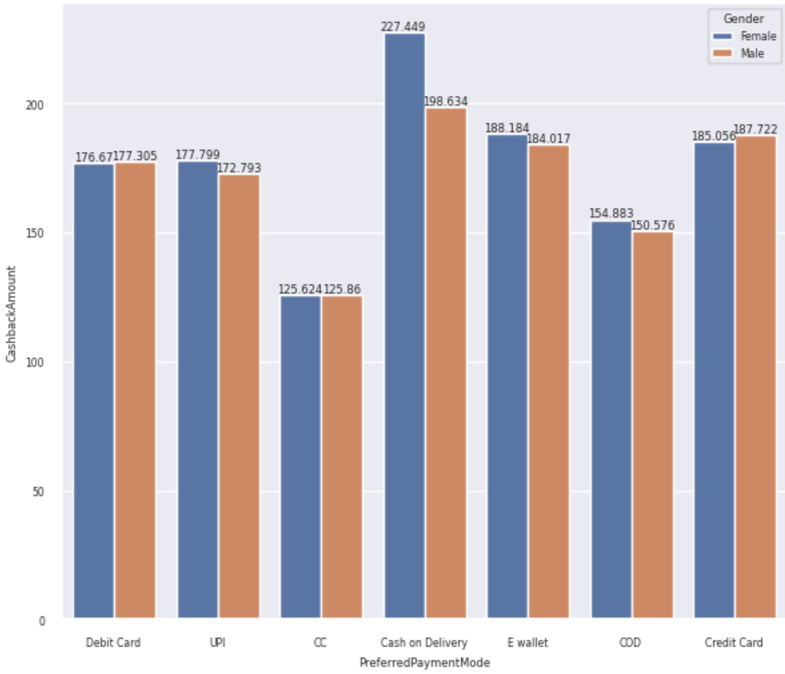
l. cashback amount vs order categories based on gender

```
plt.figure(figsize=(7,6))
sns.set(font_scale=0.5)
s=sns.barplot(data=c,x='PreferedOrderCat',y='CashbackAmount',hue='Gender',errorbar=None)
for i in s.containers:
  s.bar_label(i)
```



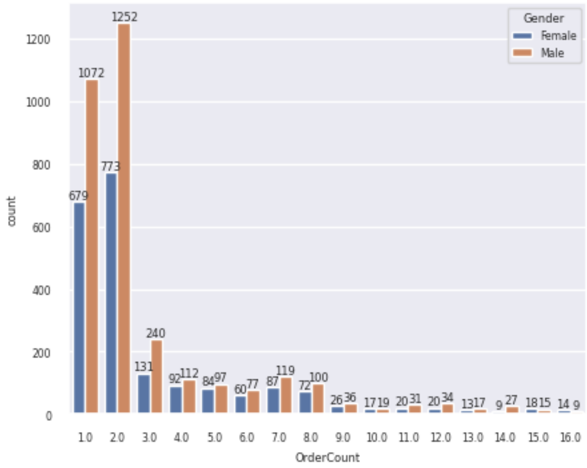
m. payment mode vs cashback amt

```
plt.figure(figsize=(7,6))
sns.set(font_scale=0.5)
s=sns.barplot(data=c,x='PreferredPaymentMode',y='CashbackAmount',hue='Gender',errorbar=None)
for i in s.containers:
  s.bar_label(i)
```



n. order count vs gender

```
plt.figure(figsize=(5,4))
sns.set(font_scale=0.5)
s=sns.countplot(data=c,x='OrderCount',hue='Gender')
for i in s.containers:
  s.bar_label(i)
```



Data Cleaning

```
print('null values:',c.isnull().any().sum())
print('nan values:',c.isna().any().sum())
print('duplicates:',c.duplicated().any().sum())
```

```
null values: 7
nan values: 7
duplicates: 0
```

Null values are dropped

```
c=c.dropna()
```

```
l=[]
for i in range(len(c)):
  l.append(i)
c.index=l
```

```
print('null values:',c.isnull().any().sum())
print('nan values:',c.isna().any().sum())
print('duplicates:',c.duplicated().any().sum())
```

```
null values: 0
nan values: 0
duplicates: 0
```

Removing unnecessary columns

```
c=c.drop(columns=['CustomerID'])
```

Feature Engineering

```
c1=c.loc[:,['PreferredLoginDevice','PreferredPaymentMode','Gender','PreferedOrderCat','MaritalStatus']]
```

```
c_f=c.drop(columns=c1.columns)
```

```
from sklearn.preprocessing import LabelEncoder
c_cd=c1.apply(LabelEncoder().fit_transform)
```

```
C=pd.concat([c_f,c_cd],axis=1)
```

Feature Selection

```
# separate data into feature and target:

x=C.drop(columns='Churn') # feature
y=C['Churn'] # Target is churn
```

```
correlation_values = x.apply(lambda feature: np.abs(np.corrcoef(feature, y)[0, 1]))
```

```
correlation_values
```

```
Tenure                         0.340013
CityTier                       0.073858
WarehouseToHome                0.087318
HourSpendOnApp                 0.060845
NumberOfDeviceRegistered       0.149041
SatisfactionScore              0.095759
NumberOfAddress                0.076336
Complain                       0.238137
OrderAmountHikeFromlastYear    0.017193
CouponUsed                     0.010982
OrderCount                     0.001962
DaySinceLastOrder              0.139254
CashbackAmount                 0.058866
PreferredLoginDevice           0.003295
PreferredPaymentMode           0.024731
Gender                         0.033792
PreferedOrderCat               0.131023
MaritalStatus                  0.134036
dtype: float64
```

```
sorted_features = correlation_values.sort_values(ascending=False)
```

```
sorted_features
```

```
Tenure                         0.340013
Complain                       0.238137
NumberOfDeviceRegistered       0.149041
DaySinceLastOrder              0.139254
MaritalStatus                  0.134036
PreferedOrderCat               0.131023
SatisfactionScore              0.095759
WarehouseToHome                0.087318
NumberOfAddress                0.076336
CityTier                       0.073858
HourSpendOnApp                 0.060845
CashbackAmount                 0.058866
Gender                         0.033792
PreferredPaymentMode           0.024731
OrderAmountHikeFromlastYear    0.017193
CouponUsed                     0.010982
PreferredLoginDevice           0.003295
OrderCount                     0.001962
dtype: float64
```

```
k = 10
selected_features = sorted_features.index[:k] # why .index because:- x[columns in selected_features. corr gives the values,we need only
```

```
selected_features
```

```
Index(['Tenure', 'Complain', 'NumberOfDeviceRegistered', 'DaySinceLastOrder',
       'MaritalStatus', 'PreferedOrderCat', 'SatisfactionScore',
       'WarehouseToHome', 'NumberOfAddress', 'CityTier'],
      dtype='object')
```

```
sns.heatmap(x[selected_features].corr(), annot=True, fmt='.2f', linewidths=0.5)
```

```
<Axes: >
```


```
sorted_features[:k]
```

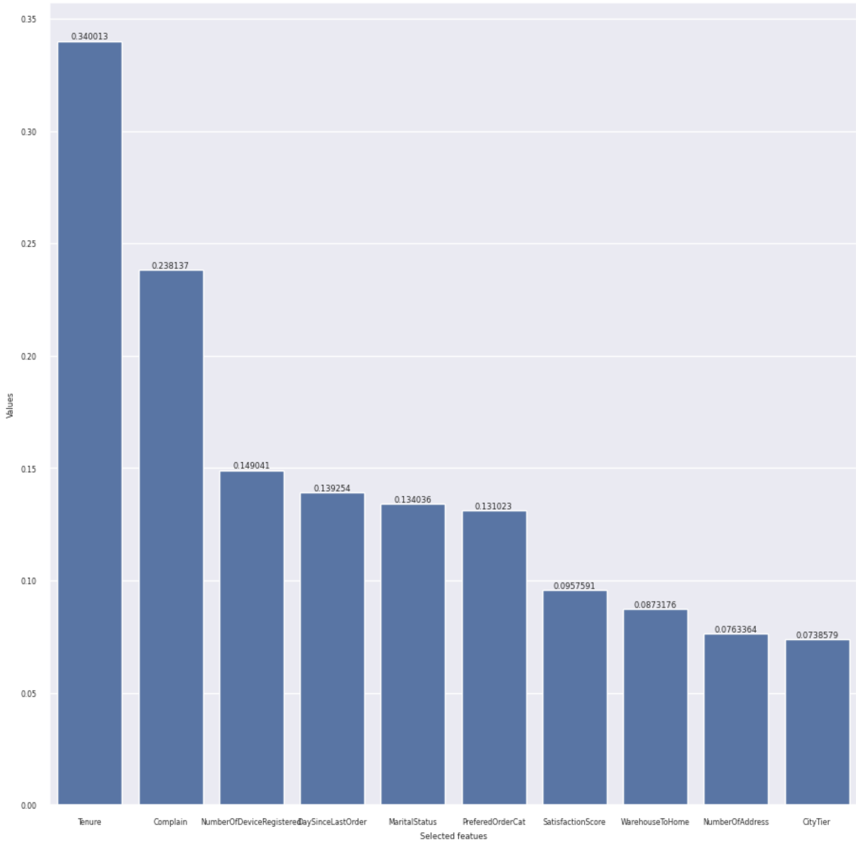
```
Tenure                      0.340013
Complain                    0.238137
NumberOfDeviceRegistered    0.149041
DaySinceLastOrder           0.139254
MaritalStatus               0.134036
PreferedOrderCat            0.131023
SatisfactionScore           0.095759
WarehouseToHome             0.087318
NumberOfAddress             0.076336
CityTier                    0.073858
dtype: float64
```

```
selected_features
```

```
Index(['Tenure', 'Complain', 'NumberOfDeviceRegistered', 'DaySinceLastOrder',
       'MaritalStatus', 'PreferedOrderCat', 'SatisfactionScore',
       'WarehouseToHome', 'NumberOfAddress', 'CityTier'],
      dtype='object')
```

```
plt.figure(figsize=(11,11))
sns.set(font_scale=0.5)
v=sns.barplot(x=selected_features, y=sorted_features[:k])
for i in v.containers:
  v.bar_label(i)
plt.xlabel('Selected featues')
plt.ylabel('Values')
```

```
Text(0, 0.5, 'Values')
```



Above are the top 10 features that act as factors of customers' churn

Model Building

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
selected_features
```

```
Index(['Tenure', 'Complain', 'NumberOfDeviceRegistered', 'DaySinceLastOrder',
       'MaritalStatus', 'PreferedOrderCat', 'SatisfactionScore',
       'WarehouseToHome', 'NumberOfAddress', 'CityTier'],
      dtype='object')
```

the top 10 features are selected for model building

```
x=x.loc[:,['Tenure', 'Complain', 'NumberOfDeviceRegistered', 'DaySinceLastOrder',
       'MaritalStatus', 'PreferedOrderCat', 'SatisfactionScore',
       'WarehouseToHome', 'NumberOfAddress', 'CityTier']]
```

```
std=StandardScaler() # standardising the features
X=std.fit_transform(x)
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

```
print('x_train.shape:',x_train.shape,'x_test.shape:',x_test.shape)
```

```
x_train.shape: (3019, 10) x_test.shape: (755, 10)
```

```
print('y_train.shape:',y_train.shape,'y_test.shape:',y_test.shape)
```

```
y_train.shape: (3019,) y_test.shape: (755,)
```

```
from sklearn.model_selection import GridSearchCV
```

```
models={
    'log_r':{
        'model':LogisticRegression(),
        'params':{

        }
    },
    'KNC':{
        'model':KNeighborsClassifier(),
        'params':{
            'n_neighbors':[2,5,10,12,15,20]
        }
    },
    'RFC':{
        'model':RandomForestClassifier(),
        'params':{
            'n_estimators':[1,2,3,4,5,6,7,8,9,10,12,15,20]
        }
    }
}
```

```
from sklearn.model_selection import ShuffleSplit
```

```
scores=[]
cv=ShuffleSplit(n_splits=5,test_size=0.2,random_state=0)
for i,j in models.items():
  gs=GridSearchCV(j['model'],j['params'],cv=cv,return_train_score=False)
  gs.fit(x,y)
scores.append({
    'model':i,
    'best score':gs.best_score_,
    'best parameter':gs.best_params_
})
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
df=pd.DataFrame(scores,columns=['model','best score','best parameter'])
df
```

	model	best score	best parameter
0	RFC	0.965828	{'n_estimators': 20}