In [ ]:

```
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import plotly.express as px
from sklearn.preprocessing import StandardScaler, MinMaxScaler, PowerTransform
from sklearn.preprocessing import LabelEncoder
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, precision_score, recall_sco
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, auc
from sklearn.metrics import roc_curve
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from yellowbrick.classifier import ROCAUC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier


plt.rcParams['figure.figsize']=[15,6]

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

In [ ]:

```
df=pd.read_csv('train.csv')
pd.set_option('display.max_columns',None)
```

In [ ]:

```
df.head()
```

Out[ ]:

	customer_id	Name	age	gender	security_no	re
0	fffe43004900440036003000030003800	Pattie Morrisey	18	F	XW0DQ7H	
1	fffe43004900440032003100300035003700	Traci Peery	32	F	5K0N3X1	
2	fffe43004900440031003900320036003600	Merideth Mcmeen	44	F	1F2TCL3	
3	fffe43004900440036003000330031003600	Eufemia Cardwell	37	M	VJGJ33N	
4	fffe43004900440031003900350030003600	Meghan Kosak	31	F	SVZXCWB	

In [ ]:
```python
df.shape
print("Number of rows are",df.shape[0])
print("Number of columns are",df.shape[1])
```

```
Number of rows are 36992
Number of columns are 25
```

In [ ]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36992 entries, 0 to 36991
Data columns (total 25 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   customer_id                   36992 non-null  object 
 1   Name                          36992 non-null  object 
 2   age                           36992 non-null  int64  
 3   gender                        36992 non-null  object 
 4   security_no                   36992 non-null  object 
 5   region_category               31564 non-null  object 
 6   membership_category           36992 non-null  object 
 7   joining_date                  36992 non-null  object 
 8   joined_through_referral       36992 non-null  object 
 9   referral_id                   36992 non-null  object 
 10  preferred_offer_types         36704 non-null  object 
 11  medium_of_operation           36992 non-null  object 
 12  internet_option               36992 non-null  object 
 13  last_visit_time               36992 non-null  object 
 14  days_since_last_login         36992 non-null  int64  
 15  avg_time_spent                36992 non-null  float64
 16  avg_transaction_value         36992 non-null  float64
 17  avg_frequency_login_days      36992 non-null  object 
 18  points_in_wallet              33549 non-null  float64
 19  used_special_discount         36992 non-null  object 
 20  offer_application_preference  36992 non-null  object 
 21  past_complaint                36992 non-null  object 
 22  complaint_status              36992 non-null  object 
 23  feedback                      36992 non-null  object 
 24  churn_risk_score              36992 non-null  int64  
dtypes: float64(3), int64(3), object(19)
memory usage: 7.1+ MB
```

In [ ]:
```python
df.describe().T
```

Out[ ]:

	count	mean	std	min	
age	36992.0	37.118161	15.867412	10.000000	23.
days_since_last_login	36992.0	-41.915576	228.819900	-999.000000	8.
avg_time_spent	36992.0	243.472334	398.289149	-2814.109110	60.
avg_transaction_value	36992.0	29271.194003	19444.806226	800.460000	14177.
points_in_wallet	33549.0	686.882199	194.063624	-760.661236	616.
churn_risk_score	36992.0	3.463397	1.409661	-1.000000	3.

In [ ]:

```
df.dtypes
```

Out[ ]:

```
customer_id                      object
Name                             object
age                               int64
gender                           object
security_no                      object
region_category                  object
membership_category              object
joining_date                     object
joined_through_referral          object
referral_id                      object
preferred_offer_types            object
medium_of_operation              object
internet_option                  object
last_visit_time                  object
days_since_last_login             int64
avg_time_spent                  float64
avg_transaction_value           float64
avg_frequency_login_days         object
points_in_wallet                float64
used_special_discount            object
offer_application_preference     object
past_complaint                   object
complaint_status                 object
feedback                         object
churn_risk_score                  int64
dtype: object
```

- joining_date and last_visit_time is in object data type and convert it in to Date data type

Data type Conversion

In [ ]:

```
df[['last_visit_time','joining_date']]=df[['last_visit_time','joining_date']].
```

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

Out[ ]:
```
customer_id                            object
Name                                   object
age                                     int64
gender                                 object
security_no                            object
region_category                        object
membership_category                    object
joining_date                   datetime64[ns]
joined_through_referral                object
referral_id                            object
preferred_offer_types                  object
medium_of_operation                    object
internet_option                        object
last_visit_time                datetime64[ns]
days_since_last_login                   int64
avg_time_spent                        float64
avg_transaction_value                 float64
avg_frequency_login_days               object
points_in_wallet                      float64
used_special_discount                  object
offer_application_preference           object
past_complaint                         object
complaint_status                       object
feedback                               object
churn_risk_score                        int64
dtype: object
```

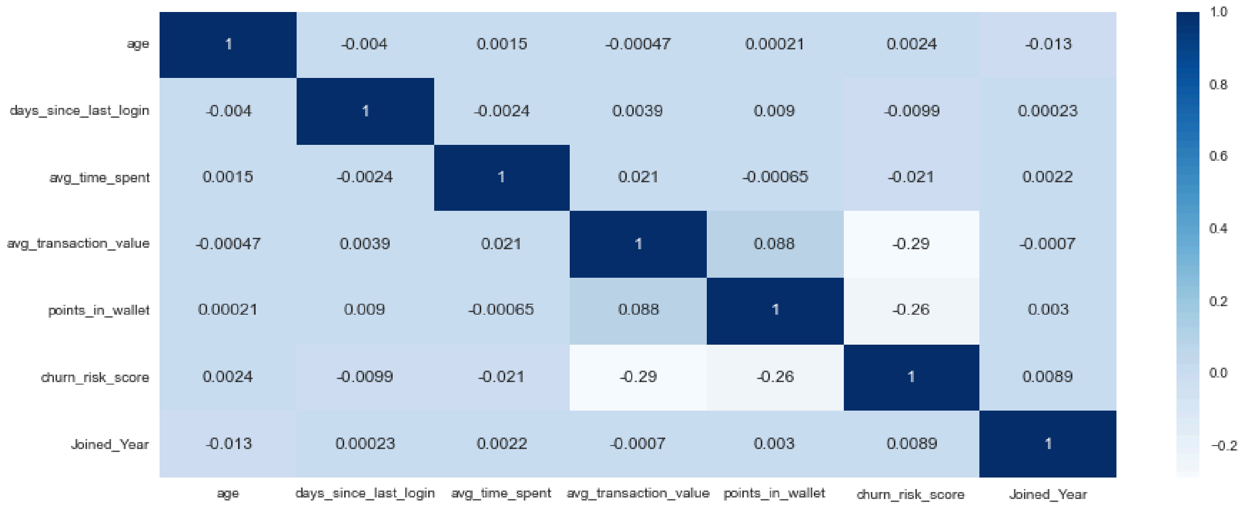
In [ ]:
```
df['Joined_Year']=df.joining_date.dt.year
#df['Joined_Month']=df.joining_date.dt.month_name()
#df['Joined_day']=df.joining_date.dt.day
#df['last_visit_Hour']=df.last_visit_time.dt.hour
```

- Extract a new feature 'joined Year' from joined date

Correlation and Covariance

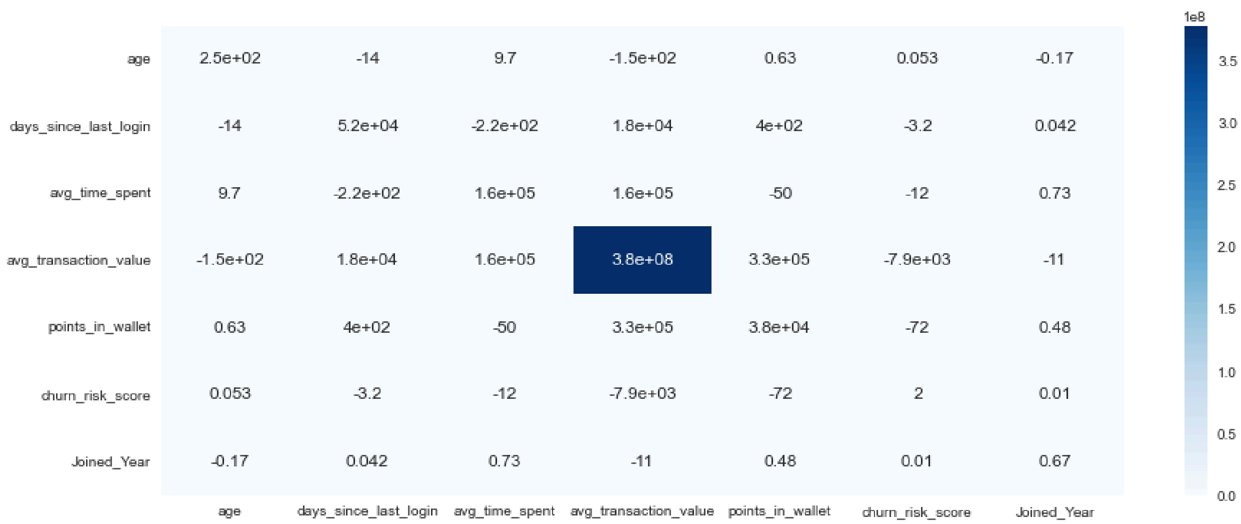
In [ ]:
```
sns.heatmap(df.corr(),annot=True,cmap='Blues')
```

Out[ ]:
```
<AxesSubplot:>
```




```
In [ ]: sns.heatmap(df.cov(),annot=True,cmap='Blues')
```

Out[ ]: <AxesSubplot:>



- From the abovemheatmap we can get to known that there is no correlation.

```
In [ ]: df.describe().T
```

Out[ ]:

	count	mean	std	min	
age	36992.0	37.118161	15.867412	10.000000	23.
days_since_last_login	36992.0	-41.915576	228.819900	-999.000000	8.
avg_time_spent	36992.0	243.472334	398.289149	-2814.109110	60.
avg_transaction_value	36992.0	29271.194003	19444.806226	800.460000	14177.
points_in_wallet	33549.0	686.882199	194.063624	-760.661236	616.
churn_risk_score	36992.0	3.463397	1.409661	-1.000000	3.
Joined_Year	36992.0	2016.006569	0.819384	2015.000000	2015.

It looks like -999 is an 'Error',i.e., the website didn't populate the variable when the data was recorded. Hence replacing it with median as their value so that we can visualize how the data is spread out

In [ ]:
```
df.loc[df['days_since_last_login'] < 0,'days_since_last_login'].value_counts(r
```

Out[ ]:
```
-999    1.0
Name: days_since_last_login, dtype: float64
```

In [ ]:
```
df.loc[df['days_since_last_login'] < 0,'days_since_last_login'] = df['days_sir
```

In [ ]:
```
df.loc[df['days_since_last_login'] < 0,'days_since_last_login']
```

Out[ ]:
```
Series([], Name: days_since_last_login, dtype: int64)
```

In [ ]:
```
'''for i in df.columns:
    if df[i].dtypes in [np.int64, np.number]:
        df.loc[df[i] < 0,i] = df[i].loc[df[i] > 0].median()'''
```

Out[ ]:
```
'for i in df.columns:\n    if df[i].dtypes in [np.int64, np.number]:\n
df.loc[df[i] < 0,i] = df[i].loc[df[i] > 0].median()'
```

Missing_values

In [ ]:
```
# Assuming df is your DataFrame
missing_values = df.isnull().sum()
print(missing_values)
```

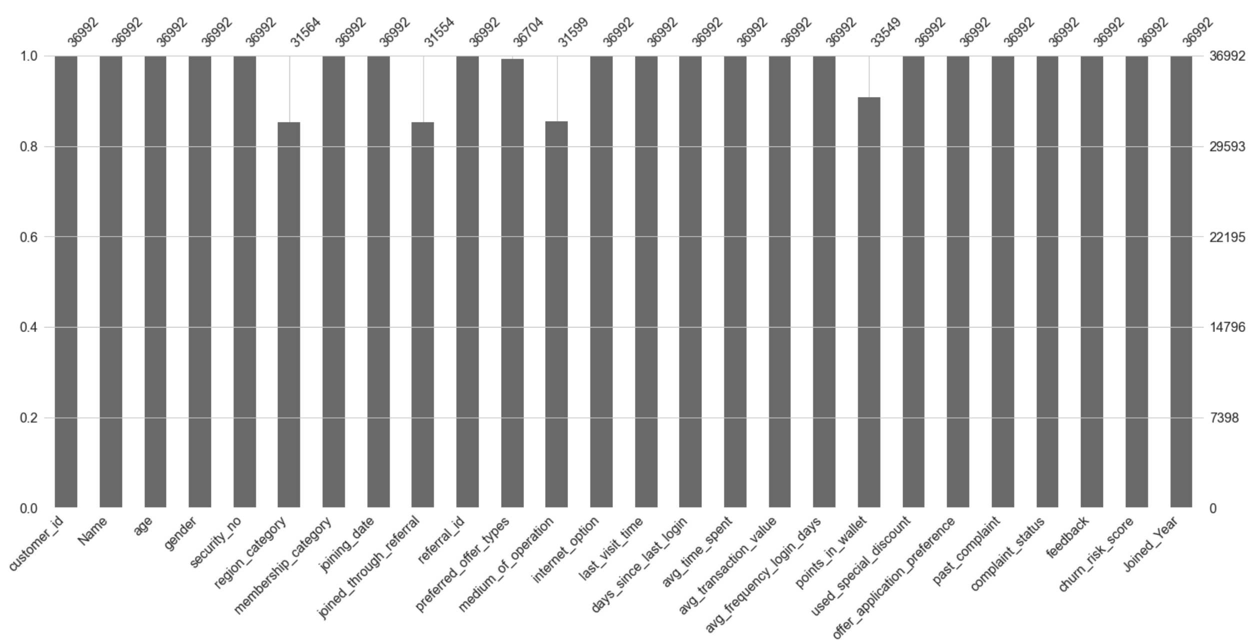
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [167], in <cell line: 1>()
----> 1 df_ = df[df.loc[:].str.contains('?')]
      2 df_

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:5575, in NDFrame.__getattr__(self, name)
   5568 if (
   5569     name not in self._internal_names_set
   5570     and name not in self._metadata
   5571     and name not in self._accessors
   5572     and self._info_axis._can_hold_identifiers_and_holds_name(name)
   5573 ):
   5574     return self[name]
-> 5575 return object.__getattribute__(self, name)

AttributeError: 'DataFrame' object has no attribute 'str'
```

In [ ]:
```python
df.replace({'?':np.nan },inplace=True)
```

- Replace '?' with null values

In [ ]:
```python
msno.bar(df)
```

Out[ ]: <AxesSubplot:>



In [ ]:
```python
df.isnull().sum()[df.isnull().sum()!=0]
```

Out[ ]:
```
region_category           5428
joined_through_referral   5438
preferred_offer_types      288
medium_of_operation       5393
points_in_wallet          3443
dtype: int64
```

From this we observed that the null values present in the columns

- region_category
- point_in_wallet
- joined_through_referral
- preferred_offer_types
- medium_of_operation

In [ ]:
```
Null_values=(df.isnull().sum()/len(df)*100)[(df.isnull().sum()/len(df)*100)!=0
print(Null_values)
```

```
region_category           14.673443
joined_through_referral   14.700476
preferred_offer_types      0.778547
medium_of_operation       14.578828
points_in_wallet           9.307418
dtype: float64
```

Null_Values Handling

- We consider to drop the rows having nul values lessthan 5%
- For Categorical Columns we are imputing with Mode
- For Numerical columns we are imputing with Median

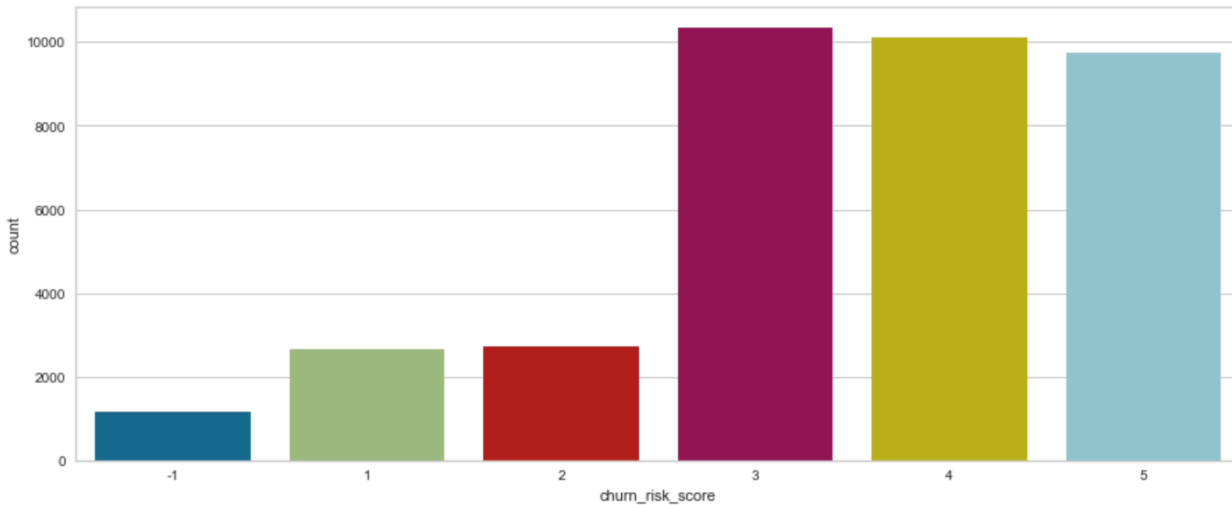
In [ ]:
```
for i in Null_values.index:
    if Null_values[i][Null_values[i]<=5]:
        df.dropna(subset=[i],axis=0,inplace=True)
    elif Null_values[i][Null_values[i]>5]:
        if df[i].dtypes in [np.int64, np.number]:
            df[i].fillna(df[i].median(),inplace=True)
        elif df[i].dtypes == np.object_:
            df[i].fillna(df[i].mode()[0],inplace=True)
```

In [ ]:
```
Null_values=(df.isnull().sum()/len(df)*100)[(df.isnull().sum()/len(df)*100)!=0
print(Null_values)
```

```
Series([], dtype: float64)
```

In [ ]:
```
sns.countplot(df.churn_risk_score)
```

Out[ ]: `<AxesSubplot:xlabel='churn_risk_score', ylabel='count'>`



Feature Engineering

In [ ]:
```
pd.crosstab(df.churn_risk_score,df.feedback)
```

Out[ ]:

feedback	No reason specified	Poor Customer Service	Poor Product Quality	Poor Website	Products always in Stock	Quality Customer Care	R
churn_risk_score							
-1	214	195	197	208	36	40	
1	0	0	0	0	675	626	
2	0	0	0	0	660	688	
3	2062	2041	2015	2080	0	0	
4	1977	2024	2100	2047	0	0	
5	1981	1935	1992	1891	0	0	

Churn risk rate -1 is not feasible value so we have done feature engineering to impute the value Compare the feedback and assign accordingly

In [ ]:
```
def feed(x, y):
    l1 = ['Poor  Quality','Too many ads','Poor Website','Poor Customer Service
    if y == -1:
        if x in l1:
            return 5
        else:
            return 1
    else:
        return y

df["churn_risk_score"] = df.apply(lambda x: feed(x['feedback'],x['churn_risk_s
```

In [ ]:
```
df.churn_risk_score.value_counts()
```

Out[ ]:
```
5    10341
3    10339
4    10098
1     3207
2     2719
Name: churn_risk_score, dtype: int64
```

Labelling

- We have Churn risk rate from 1 to 5
- we are bucketing the label in order for better prediction
- risk rate 1 and 2 are low risk so we assign those to label 0
- risk rate 3 falls in both low and high, so we keep it as a standalone label 1
- risk rate 4 and 5 are high risk rate so we assing it to label 2

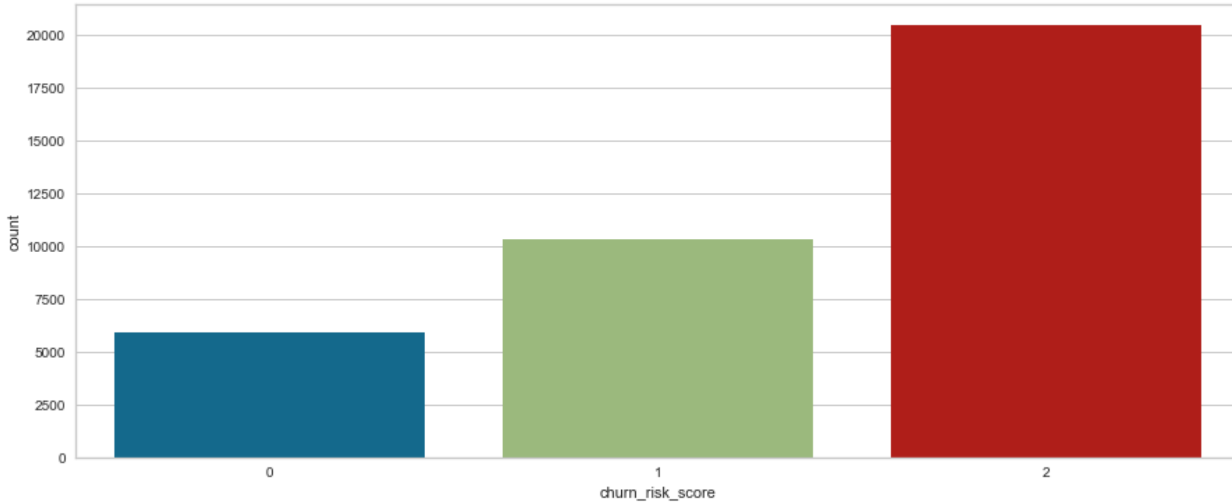
In [ ]:
```
def bucket(x):
    if (x == 1) | (x == 2):
        return 0
    elif (x == 3):
        return 1
    else:
        return 2
df["churn_risk_score"] = df.apply(lambda x: bucket(x['churn_risk_score']), axi
```

In [ ]:
```
df.churn_risk_score.value_counts()
```

Out[ ]:
```
2    20439
1    10339
0     5926
Name: churn_risk_score, dtype: int64
```

In [ ]:
```
sns.countplot(df.churn_risk_score)
```

Out[ ]:
```
<AxesSubplot:xlabel='churn_risk_score', ylabel='count'>
```



Dropping In-significant variables

In [ ]:
```
df.avg_frequency_login_days.value_counts()
```

Out[ ]:
```
Error                   3496
13.0                    1382
19.0                    1351
8.0                     1350
14.0                    1349
                        ...
28.191570401129514         1
41.73357294995208          1
-11.515939810499656        1
45.71683637272365          1
27.8399274405269           1
Name: avg_frequency_login_days, Length: 1632, dtype: int64
```

Looks like avg_freq_login_days(Represents the no. of times a customer has logged in to the website) variable is holding numeric datatype. Hence converted to float.

ERROR value infers that the website was unable to register the avg_freq_login_days. It could be due to various factors like software glitches, etc. Also, the variable days since last login and average frequency login days holds redundancy in terms of their usage. Hence dropping the variable.

Customer-id, Name, security_no are unique variables. referral_id is completely irrelavant to the dataset. Hence, dropping the above mentioned variables.

In [ ]:
```
df=df.drop(columns=['customer_id','Name','security_no','referral_id','avg_freq
```

In [ ]:
```
df['churn_risk_score'] = df['churn_risk_score'].astype('object')
```

Univariate_analysis

In [ ]:
```
Caterogical_columns=df.select_dtypes(include=np.object_).columns
print(Caterogical_columns)
```

```
Index(['gender', 'region_category', 'membership_category',
       'joined_through_referral', 'preferred_offer_types',
       'medium_of_operation', 'internet_option', 'used_special_discount',
       'offer_application_preference', 'past_complaint', 'complaint_status',
       'feedback', 'churn_risk_score'],
      dtype='object')
```

In [ ]:
```
Numerical_columns=df.select_dtypes(include=np.number).columns
print(Numerical_columns)
len(Numerical_columns)
```

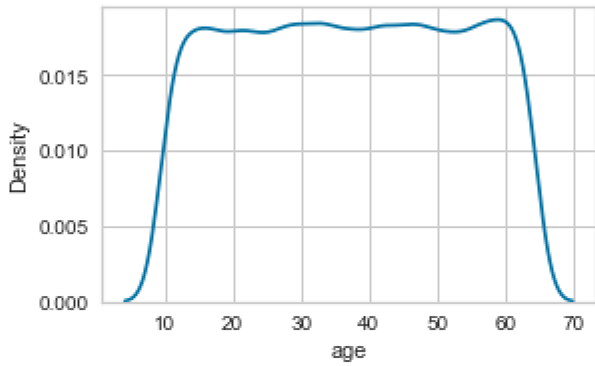
```
Index(['age', 'days_since_last_login', 'avg_time_spent',
       'avg_transaction_value', 'points_in_wallet', 'Joined_Year'],
      dtype='object')
```

Out[ ]: 6

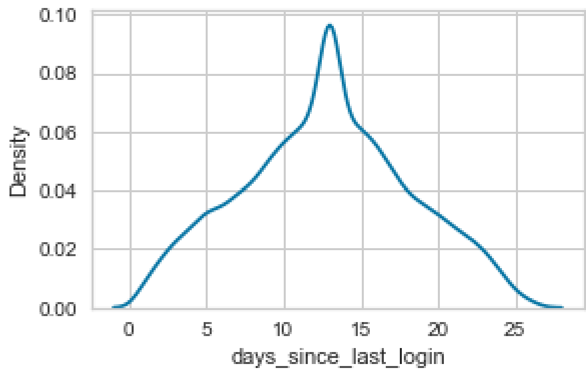
In [ ]:
```
df.dtypes
```

Out[ ]:
```
age                                int64
gender                            object
region_category                   object
membership_category               object
joined_through_referral           object
preferred_offer_types             object
medium_of_operation               object
internet_option                   object
days_since_last_login              int64
avg_time_spent                   float64
avg_transaction_value            float64
points_in_wallet                 float64
used_special_discount             object
offer_application_preference      object
past_complaint                    object
complaint_status                  object
feedback                          object
churn_risk_score                  object
Joined_Year                        int64
dtype: object
```

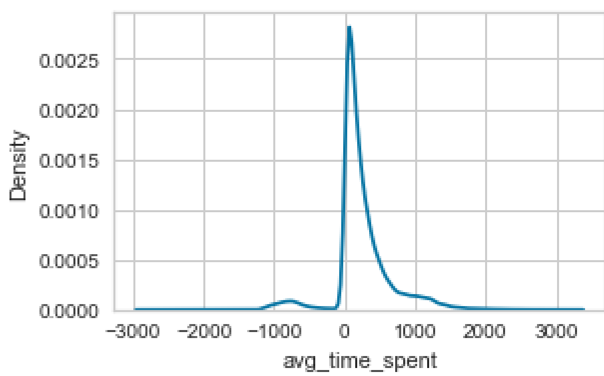
In [ ]:
```
nrows=2
ncols=3
iterator=1

for i in Numerical_columns:
    plt.subplot(nrows,ncols,iterator)
    sns.kdeplot(df.loc[:,i])
    plt.show()
    iterator+=1

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



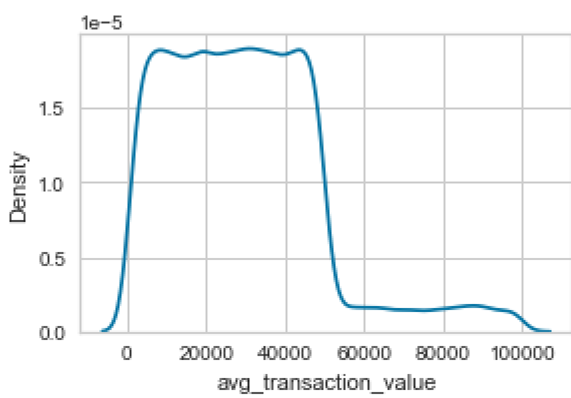
Density
0.000
0.005
0.010
0.015
10
20
30
40
50
60
70
age



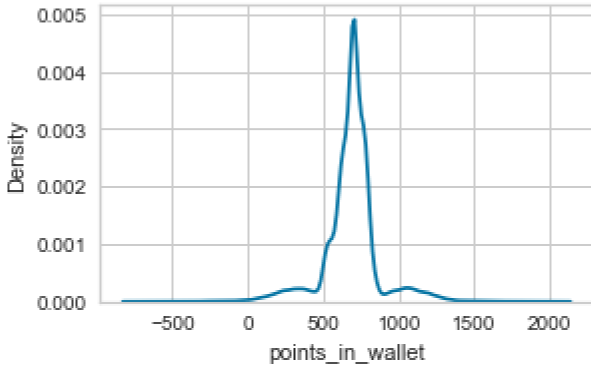
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
5
10
15
20
25
days_since_last_login

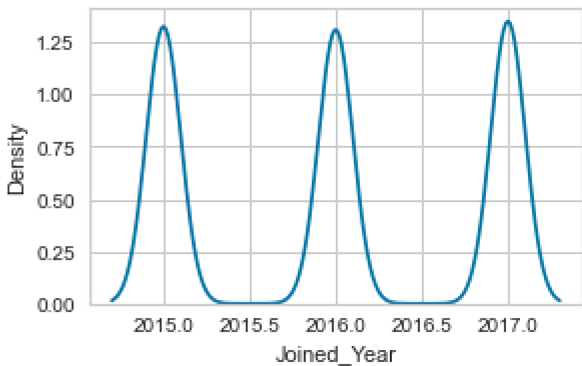


Density
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
−3000
−2000
−1000
0
1000
2000
3000
avg_time_spent



1e−5
Density
0.0
0.5
1.0
1.5
0
20000
40000
60000
80000
100000
avg_transaction_value



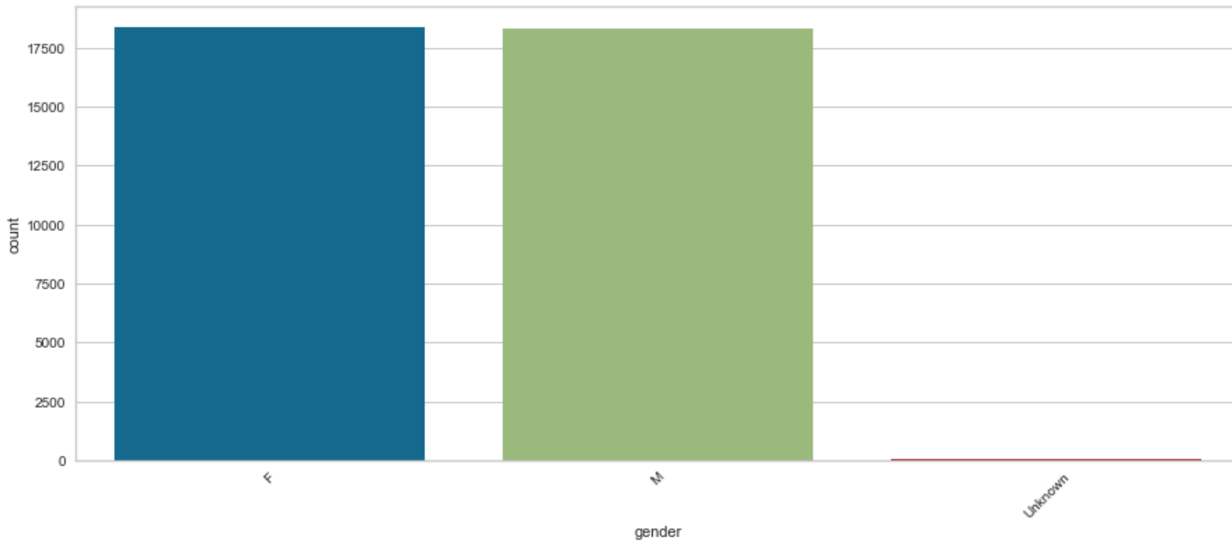



```
<Figure size 1080x432 with 0 Axes>
```

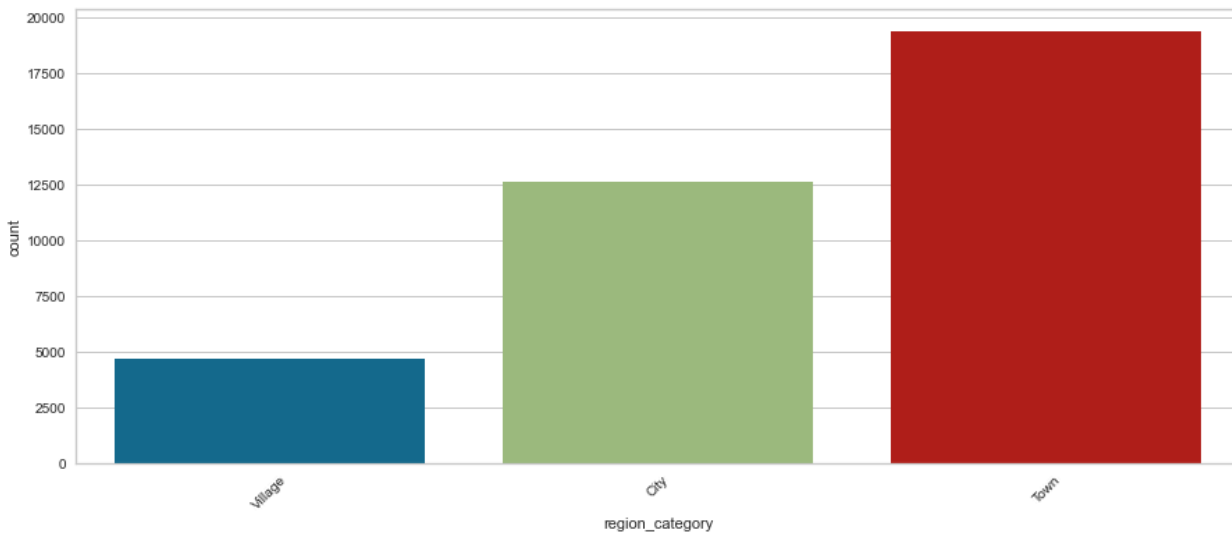
From the above figure we can get to know about the distribution of the data for all the numerical columns.

In [ ]:

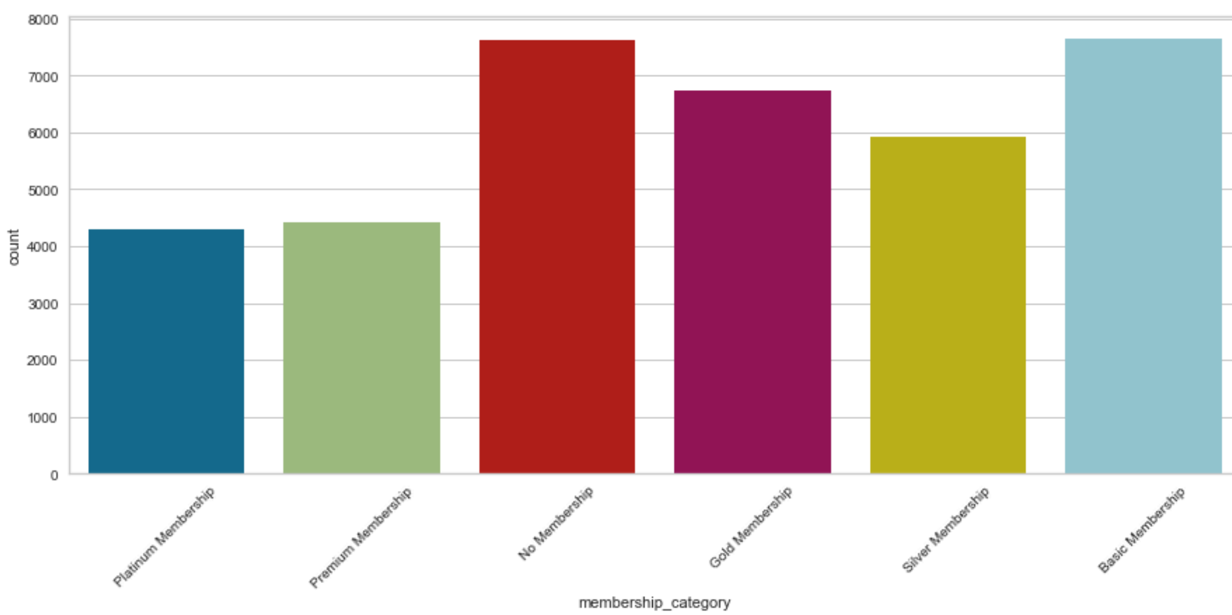
```
nrows=9
ncols=2
iterator=1

for i in Caterogical_columns:
    #plt.subplot(nrows,ncols,iterator)
    sns.countplot(df.loc[:,i])
    #iterator+=1
    plt.xticks(rotation=45)
    plt.show()
```



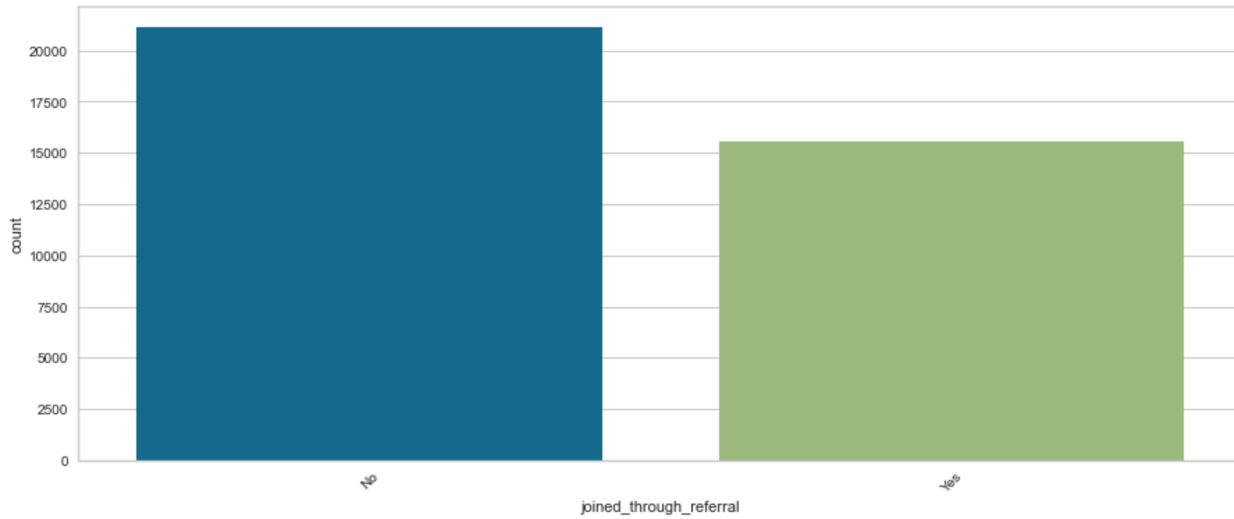
count
17500
15000
12500
10000
7500
5000
2500
0
F
M
Unknown
gender

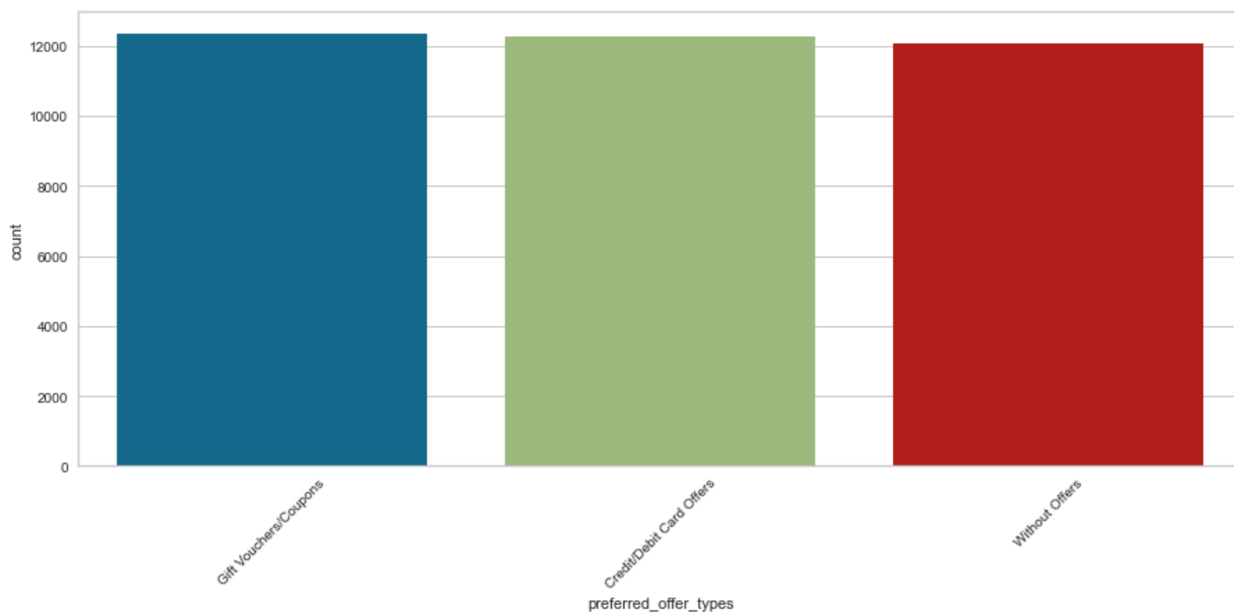


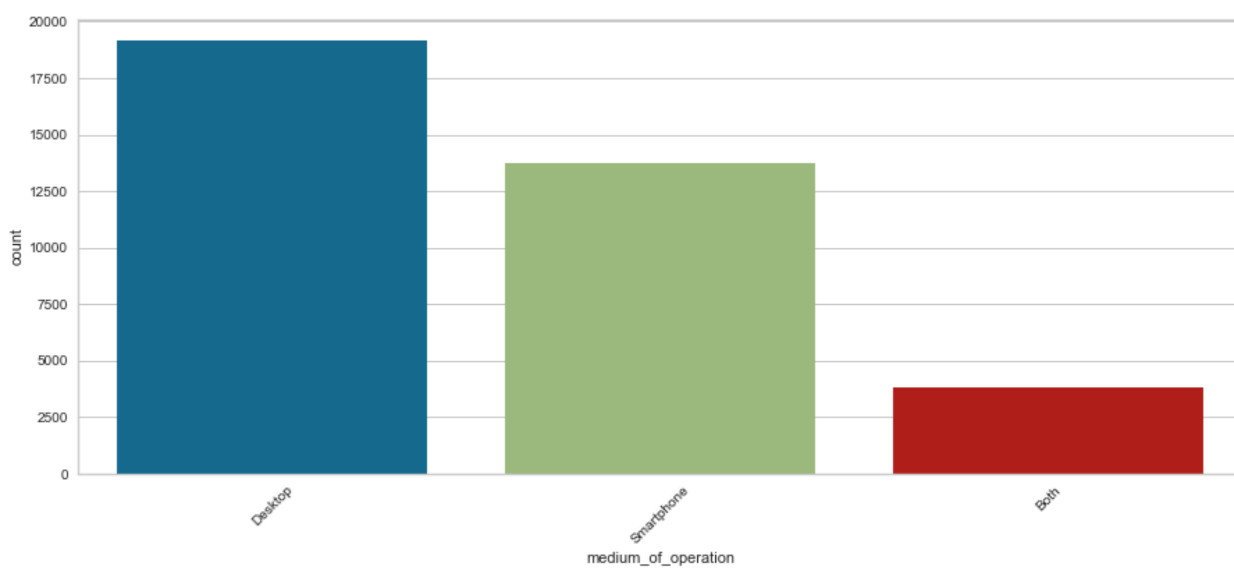
count
20000
17500
15000
12500
10000
7500
5000
2500
0
Village
City
Town
region_category

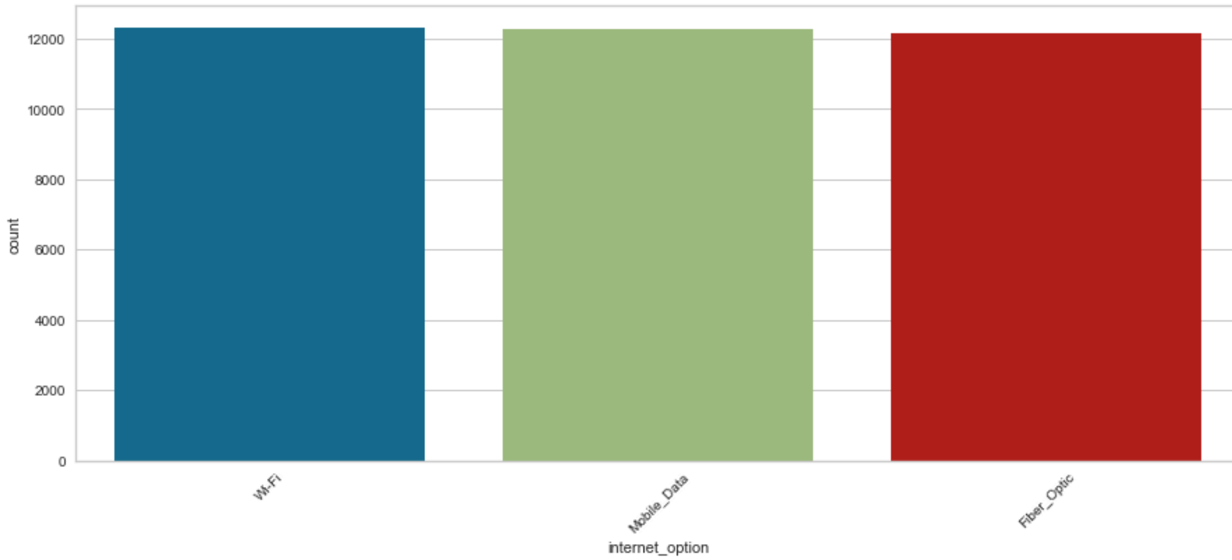


count
8000
7000
6000
5000
4000
3000
2000
1000
0
Platinum Membership
Premium Membership
No Membership
Gold Membership
Silver Membership
Basic Membership
membership_category

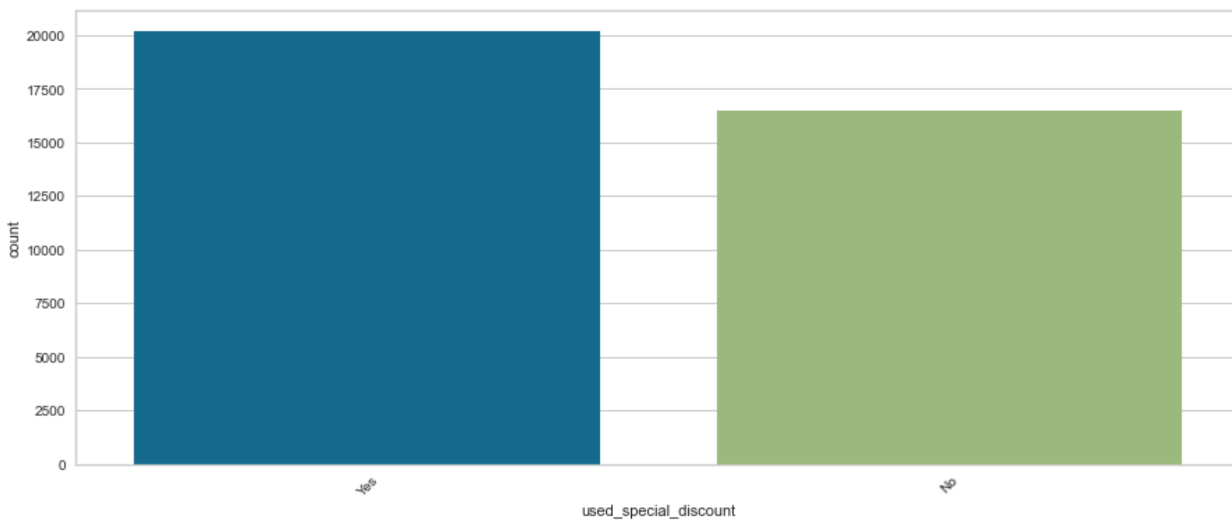

count
0
2500
5000
7500
10000
12500
15000
17500
20000
No
Yes
joined_through_referral


count
0
2000
4000
6000
8000
10000
12000
Gift Vouchers/Coupons
Credit/Debit Card Offers
Without Offers
preferred_offer_types

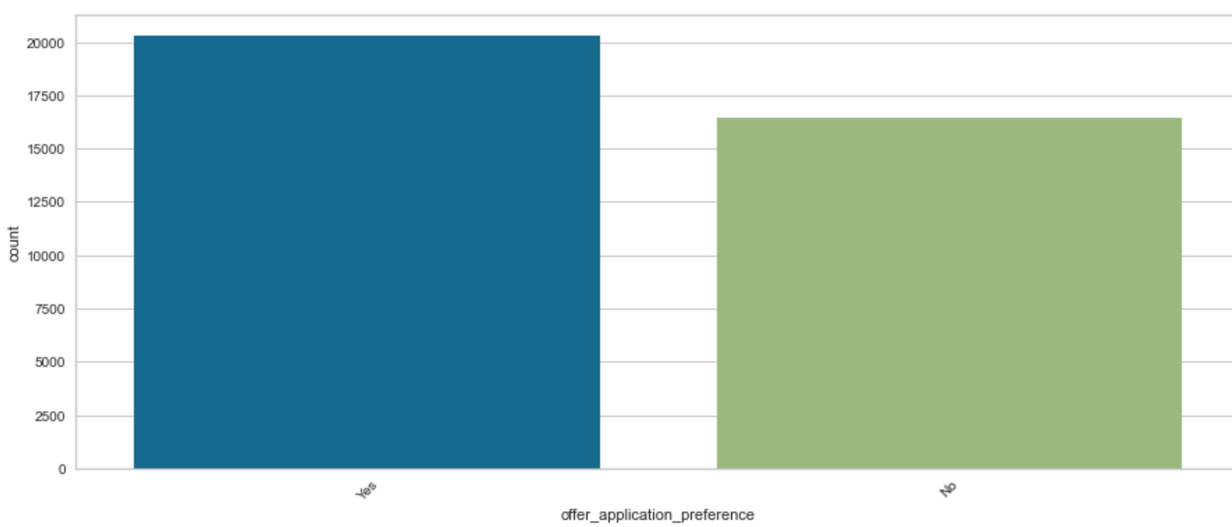

count
0
2500
5000
7500
10000
12500
15000
17500
20000
Desktop
Smartphone
Both
medium_of_operation



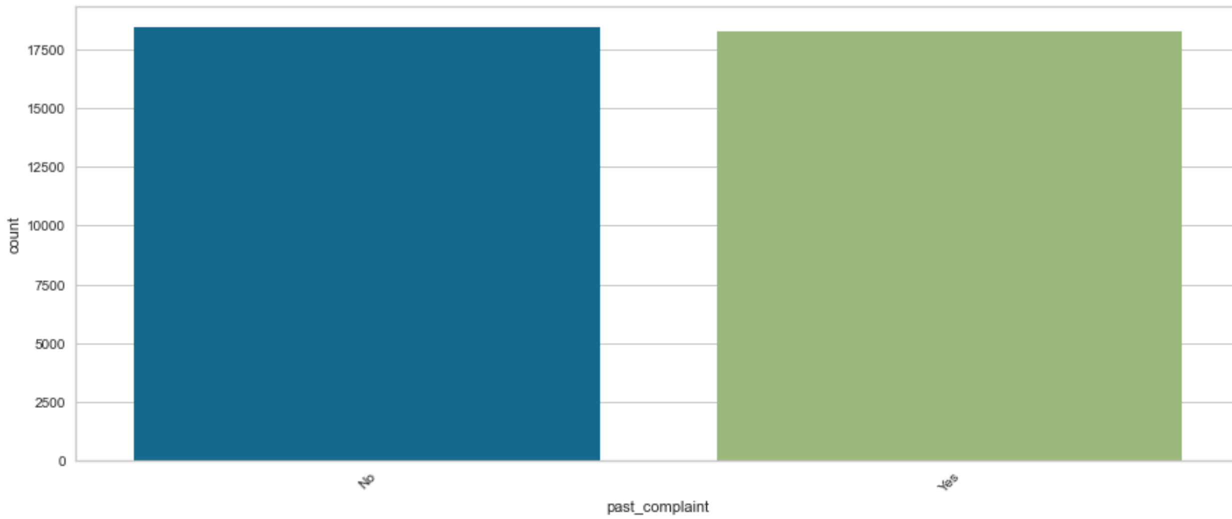
count
12000
10000
8000
6000
4000
2000
0
Wi-Fi
Mobile_Data
Fiber_Optic
internet_option

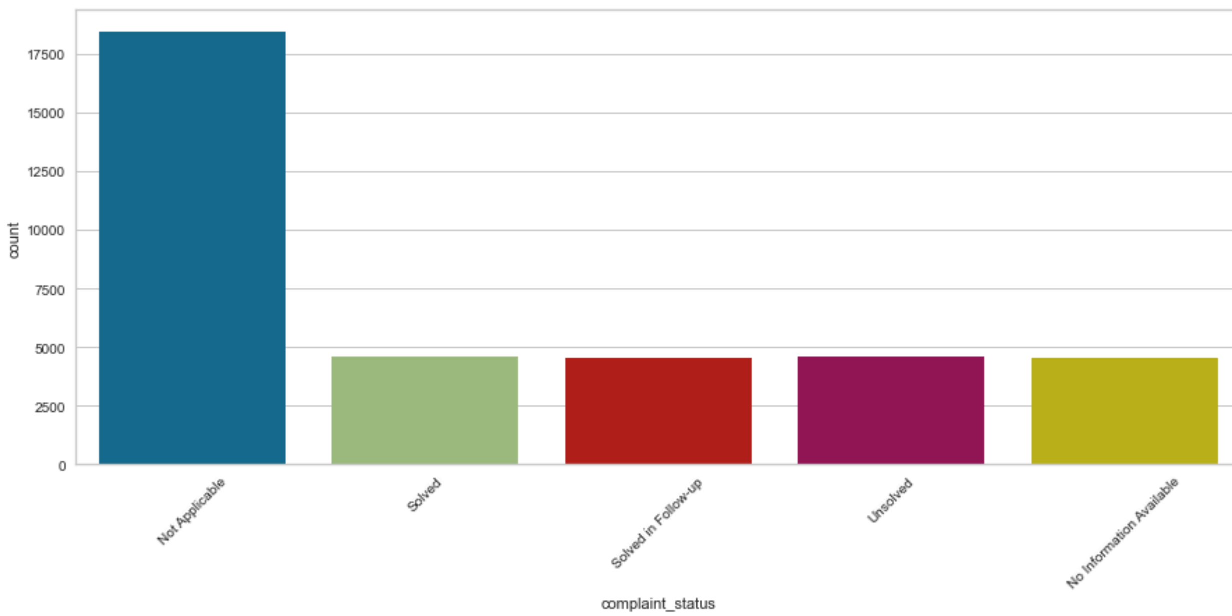


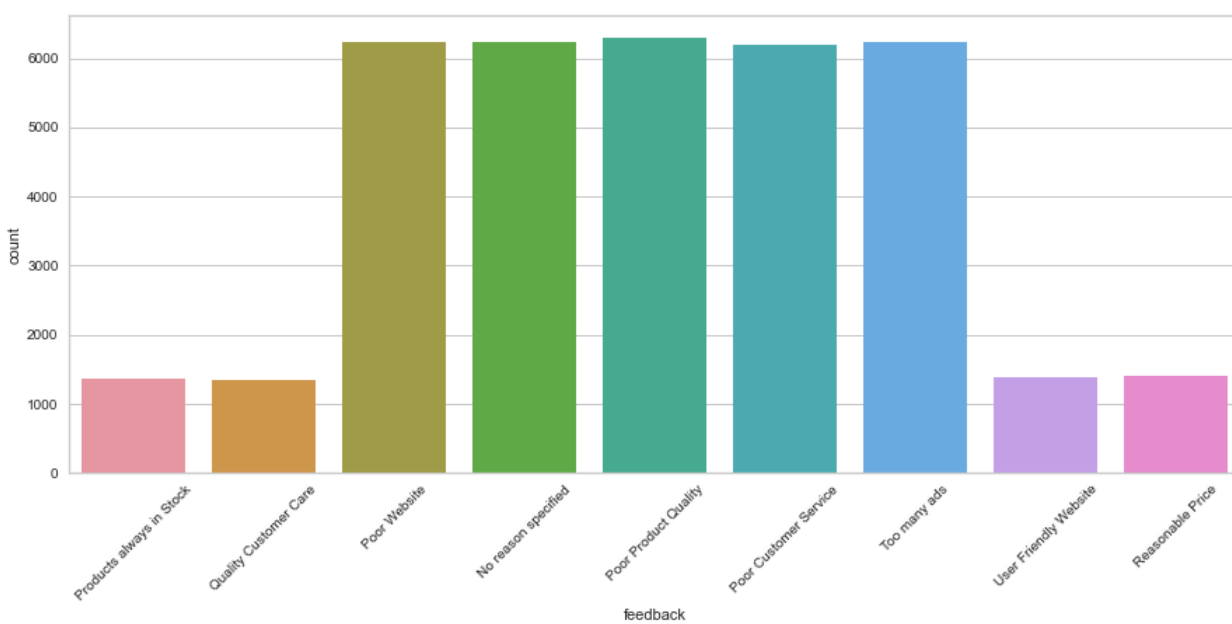
count
20000
17500
15000
12500
10000
7500
5000
2500
0
Yes
No
used_special_discount

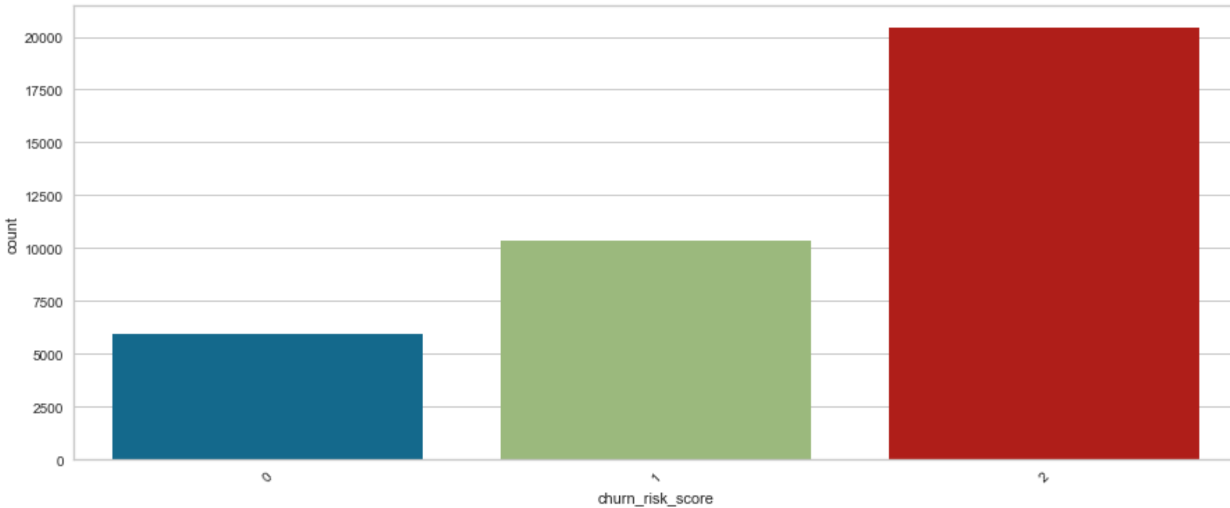


count
20000
17500
15000
12500
10000
7500
5000
2500
0
Yes
No
offer_application_preference


count
17500
15000
12500
10000
7500
5000
2500
0
No
Yes
past_complaint


count
17500
15000
12500
10000
7500
5000
2500
0
Not Applicable
Solved
Solved in Follow-up
Unsolved
No Information Available
complaint_status


count
6000
5000
4000
3000
2000
1000
0
Products always in Stock
Quality Customer Care
Poor Website
No reason specified
Poor Product Quality
Poor Customer Service
Too many ads
User Friendly Website
Reasonable Price
feedback



The above plots represents the data available in categorical columns and their distribution.

Inference

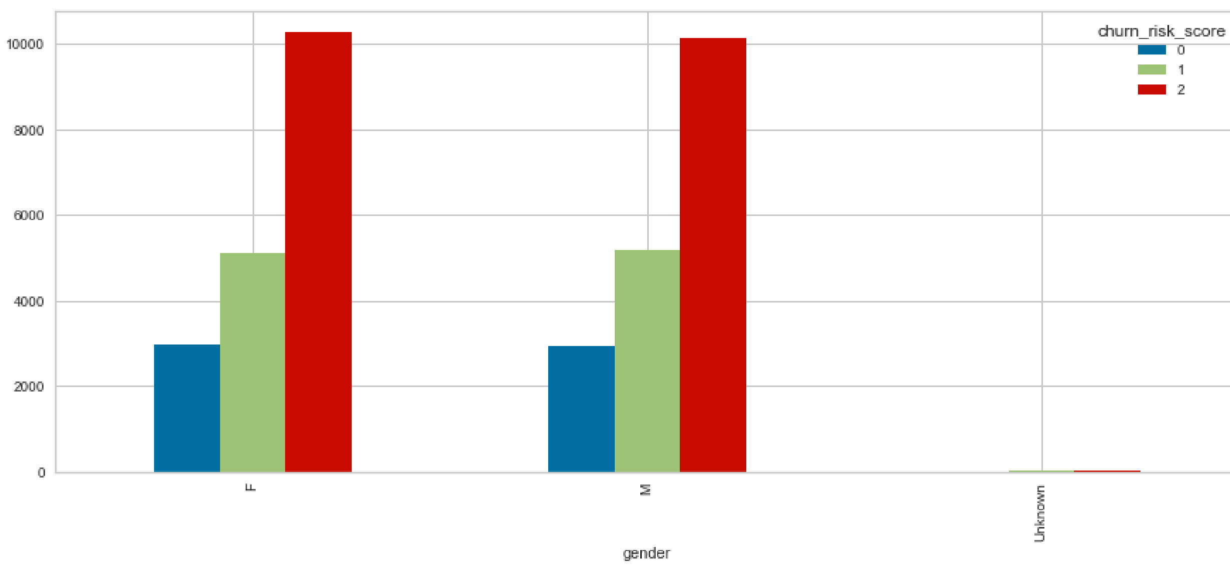
- The Age category shows that the Age is widely spread from 10-64 with almost equal weightage.
- The Gender is almost equally spread in the data except the unknown subclass
- The region category Town is having the maximum counts. And the region Village is having the minimum counts. The plot shows that the town population is attracted to this particular e-commerce site.
- The Membership that the Basic Membership and No Membership are having the highest count. With the platinum membership being the lowest in count
- The preferred_offer_types is almost equally spread in the data except the missing values.We cannot impute missing values as this the variable is related to personal information.
- From the Barplot, we can infer that both Desktop and Smartphone is spread equally. We can also infer that only 10% of people using both Smartphone and Desktop.
- The Internet option is showing equal weightage to all the subclasses being Wi-Fi, Mobile, Fiber_Optic.
- The days_since_last_login variable is holding the number of days since the customer has logged in. The plot shows that the average lies around 13.The maximum days since logged-in is 31. And the minimum is 1.
- Maximum number of negative feedback for the variable is poor product

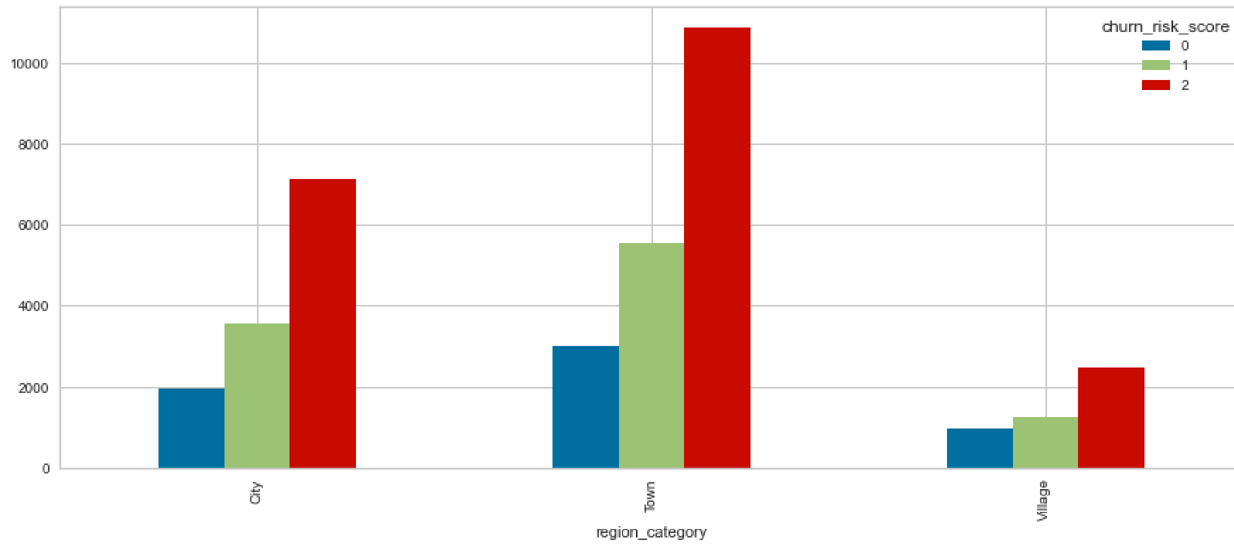
quality and maximum number of positive feedback for the variable is user friendly website and reasonable price.

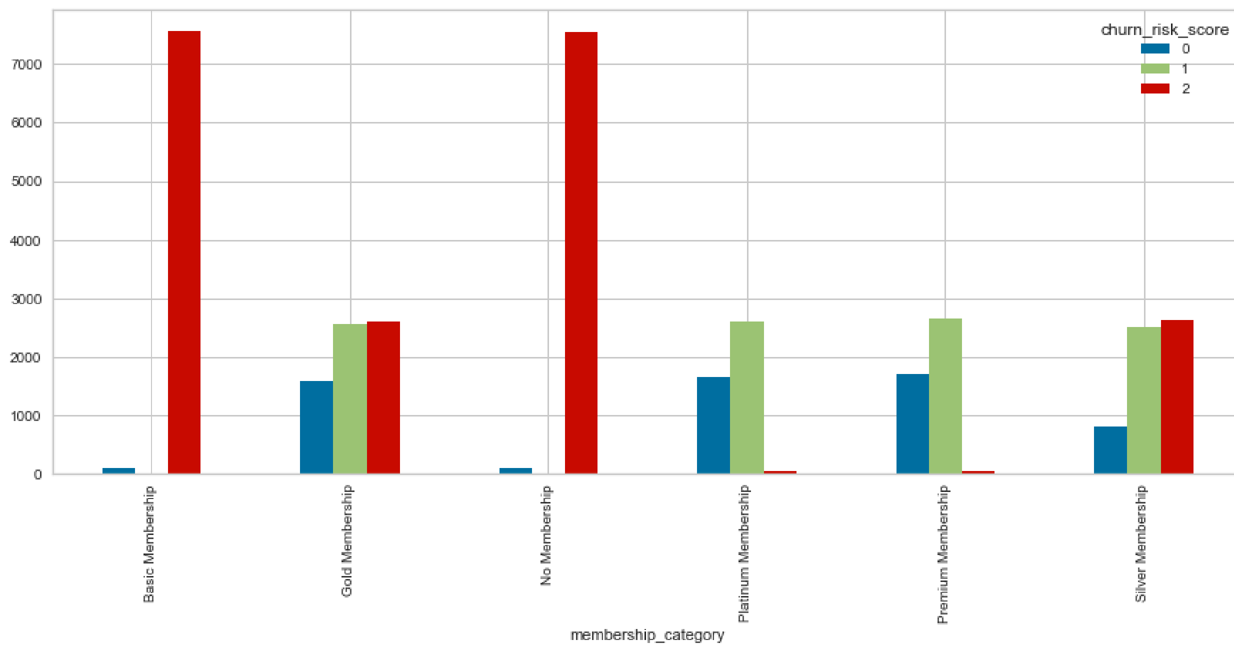
- The complaint status' subclasses holds almost equal weights except 'Not Applicable'.
- In Medium of operation Desktop users are more than smartphone
- Used special discount column has shos that most people are not using special discount offers
- Point in wallet has the maximum value of 500 to 800

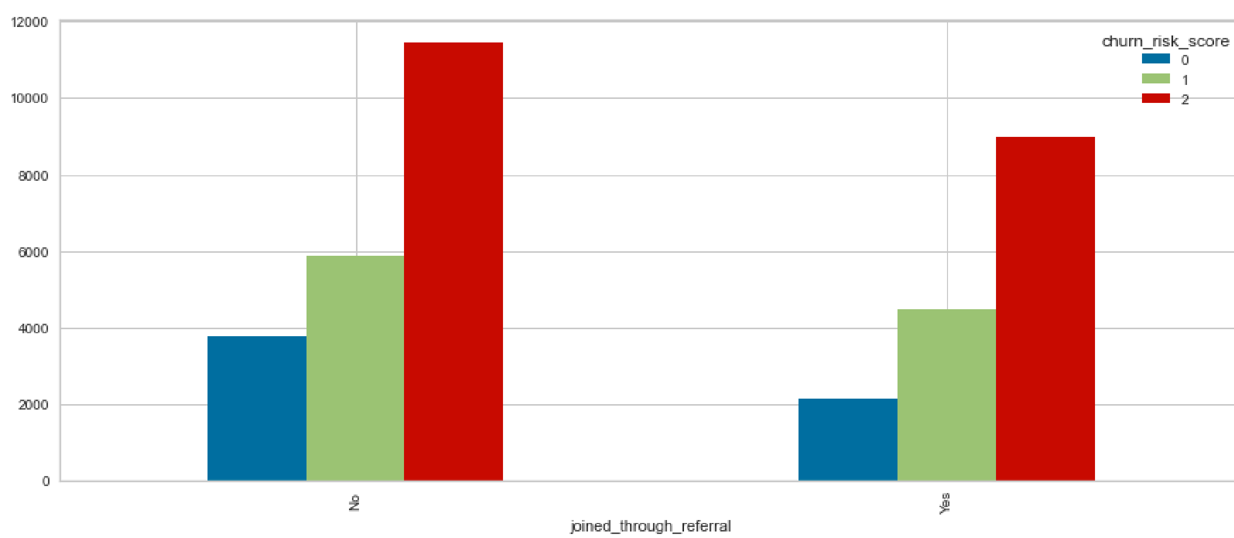
Bivariate analysis

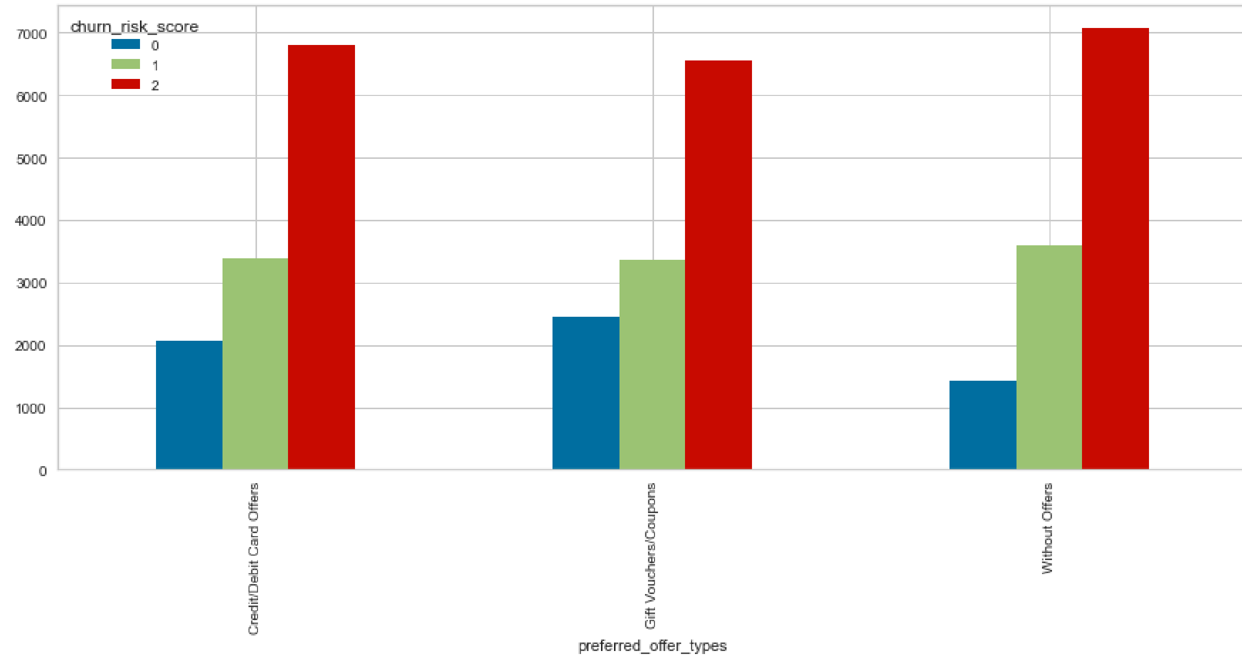
In [ ]:
```
nrows=9
ncols=2
iterator=1


for i in Caterogical_columns:
    #plt.subplot(nrows,ncols,iterator)
    pd.crosstab(df.loc[:,i],df.churn_risk_score).plot(kind='bar')
    #sns.boxplot(x=df.loc[:,i],y=df.churn_risk_score)
    #iterator+=1
    plt.show()
```

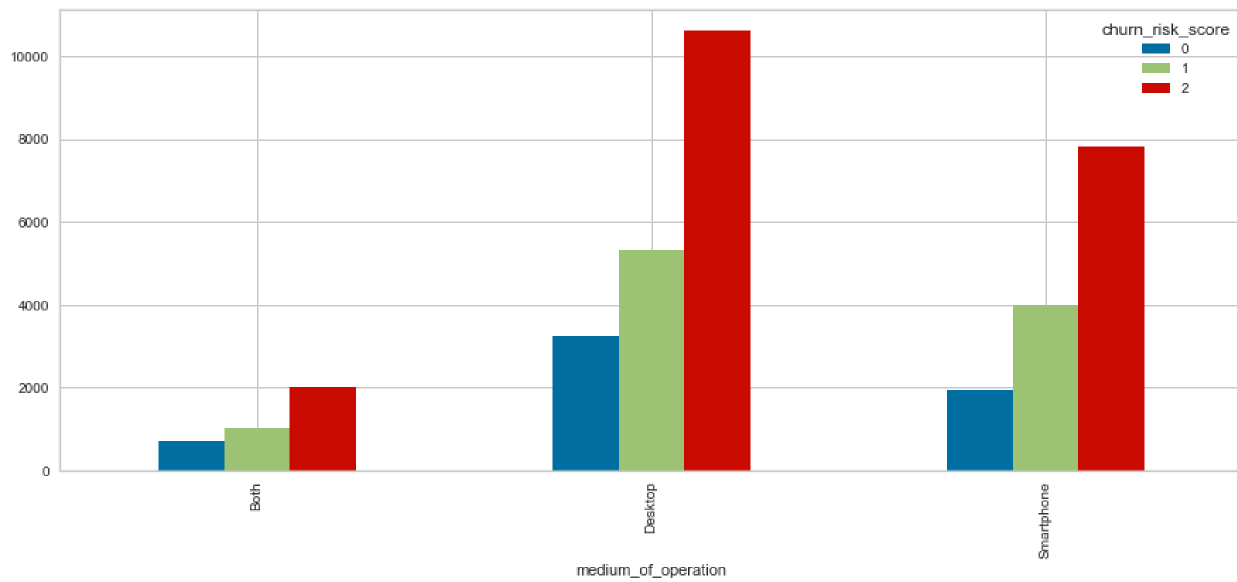


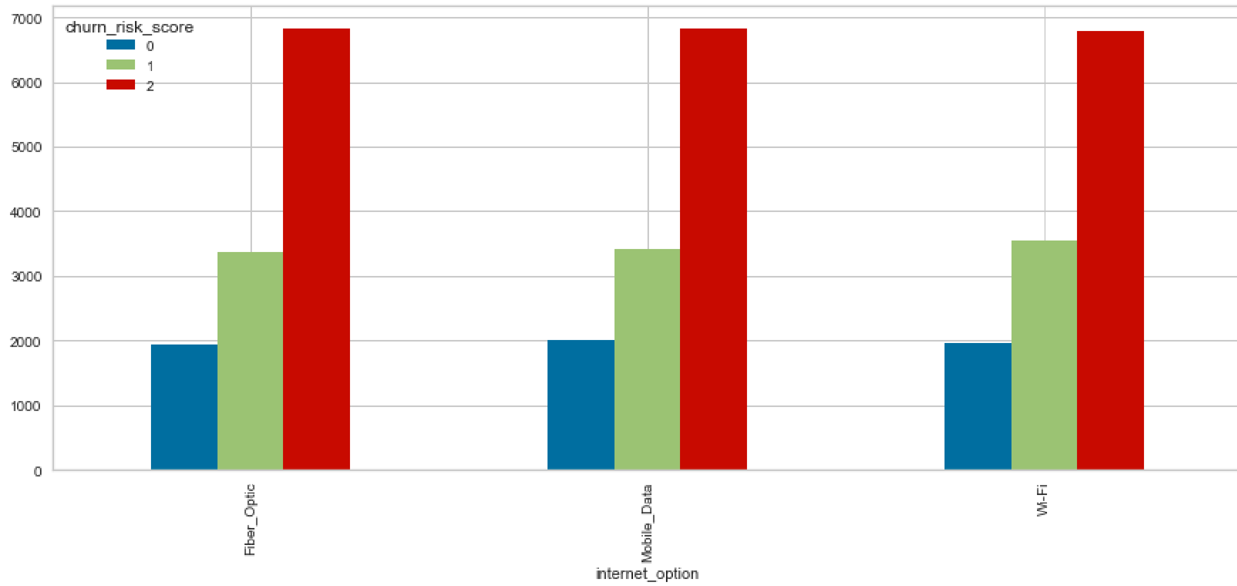

churn_risk_score
0
1
2
10000
8000
6000
4000
2000
0
City
Town
Village
region_category

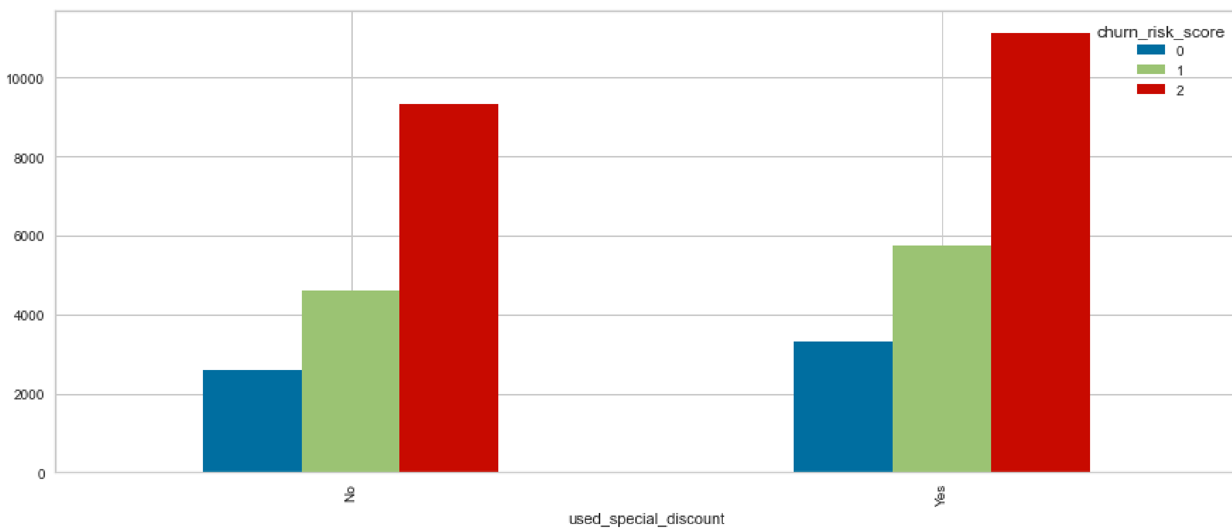

churn_risk_score
0
1
2
7000
6000
5000
4000
3000
2000
1000
0
Basic Membership
Gold Membership
No Membership
Platinum Membership
Premium Membership
Silver Membership
membership_category

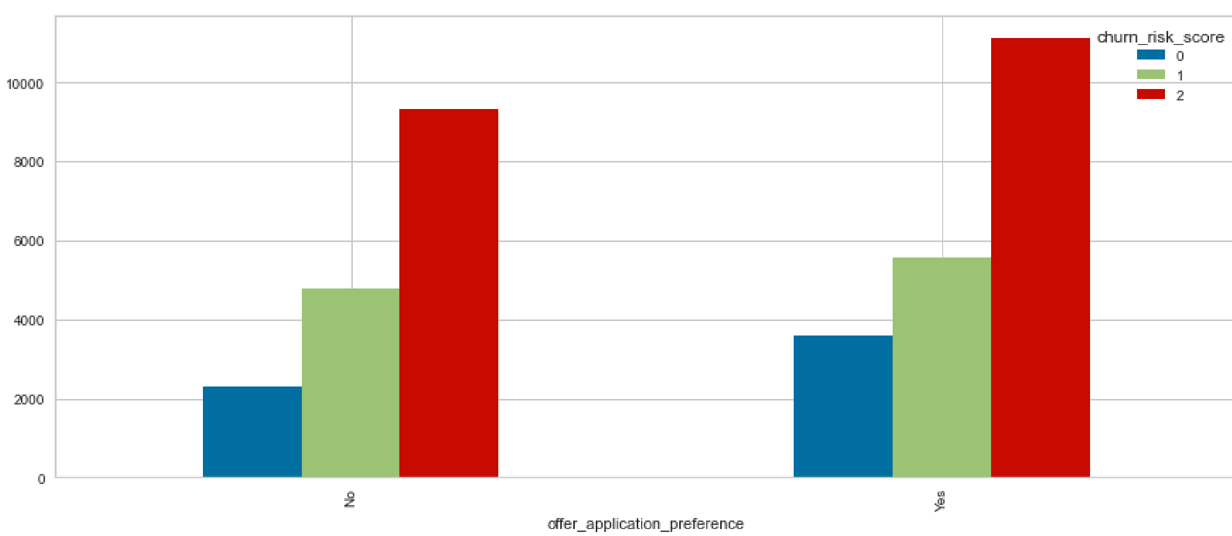

churn_risk_score
0
1
2
12000
10000
8000
6000
4000
2000
0
No
Yes
joined_through_referral

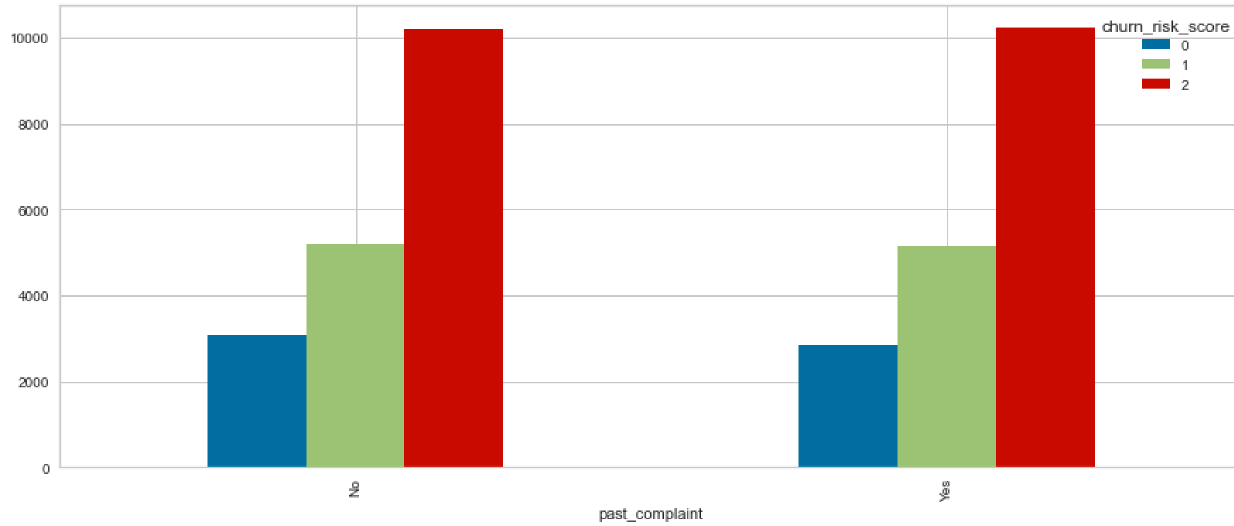

churn_risk_score
0
1
2
7000
6000
5000
4000
3000
2000
1000
0
Credit/Debit Card Offers
Gift Vouchers/Coupons
Without Offers
preferred_offer_types


churn_risk_score
0
1
2
10000
8000
6000
4000
2000
0
Both
Desktop
Smartphone
medium_of_operation

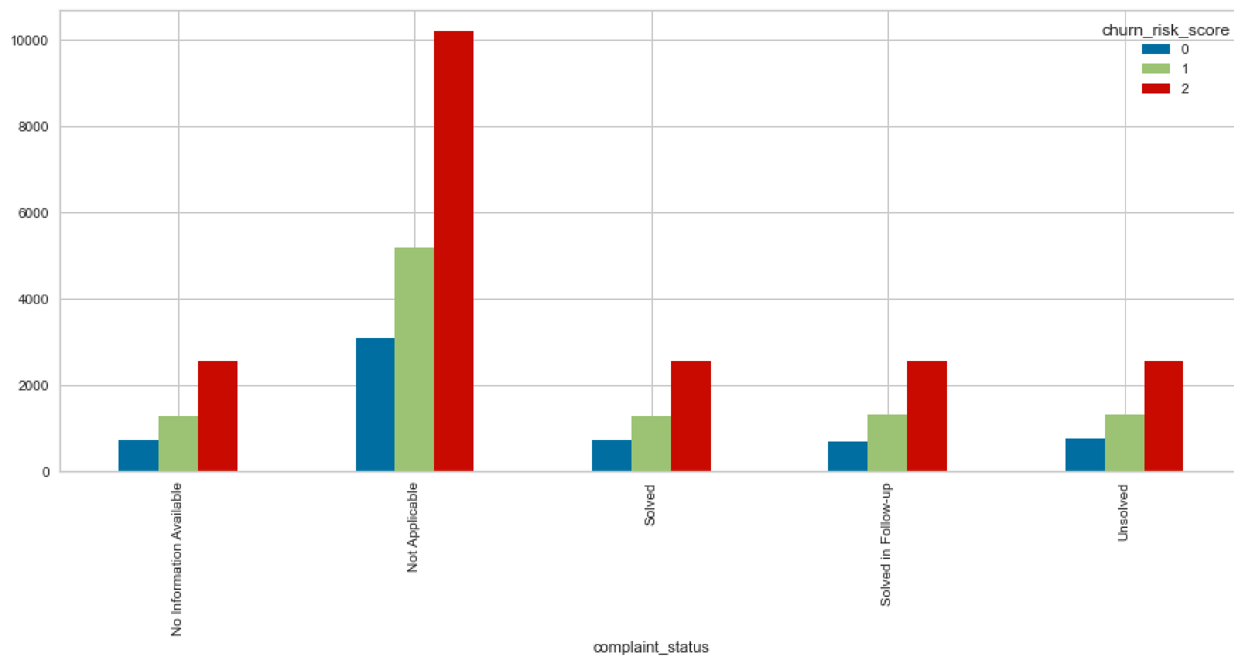

churn_risk_score
0
1
2
7000
6000
5000
4000
3000
2000
1000
0
Fiber_Optic
Mobile_Data
Wi-Fi
internet_option


churn_risk_score
0
1
2
10000
8000
6000
4000
2000
0
No
Yes
used_special_discount

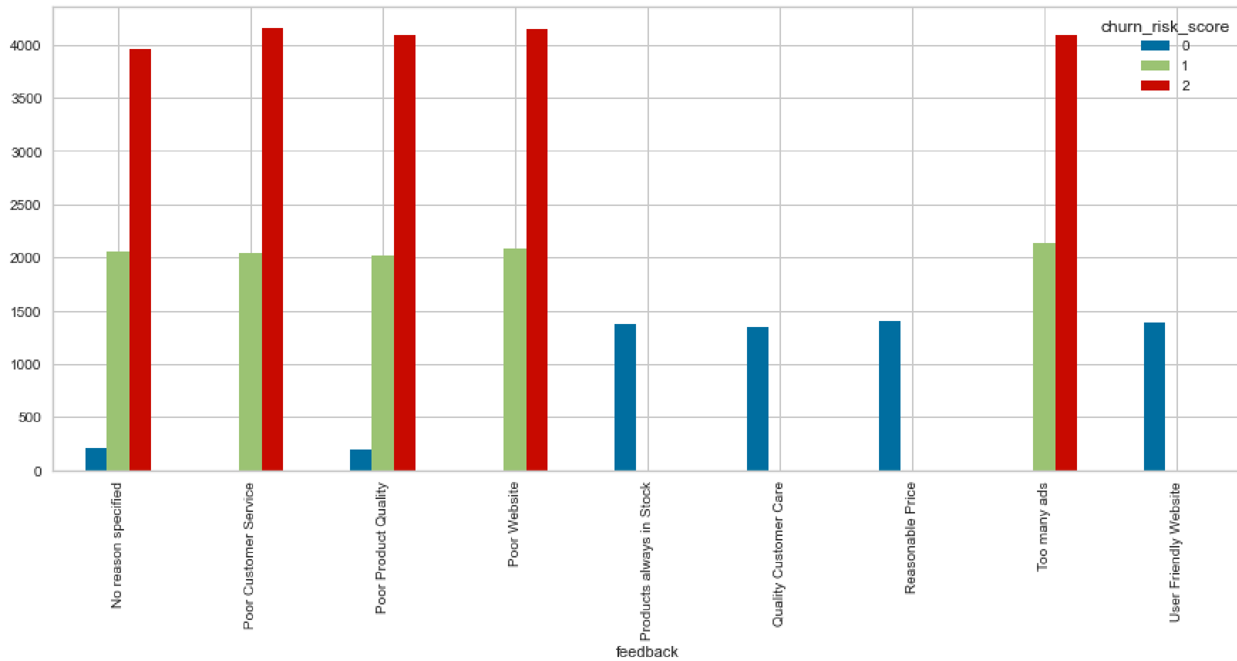

churn_risk_score
0
1
2
10000
8000
6000
4000
2000
0
No
Yes
offer_application_preference

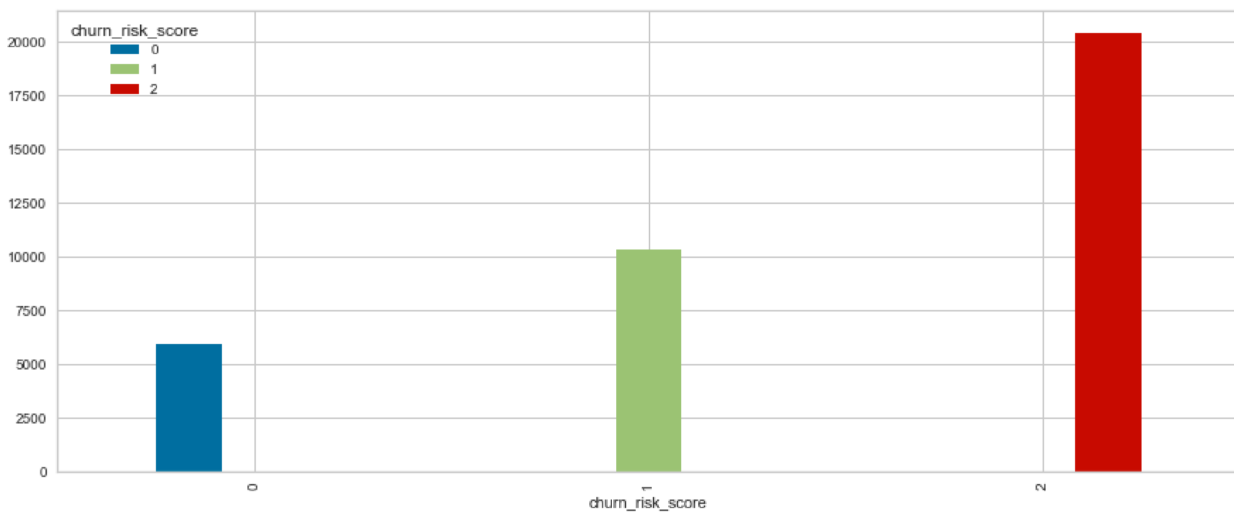


churn_risk_score
0
1
2
10000
8000
6000
4000
2000
0
No
Yes
past_complaint



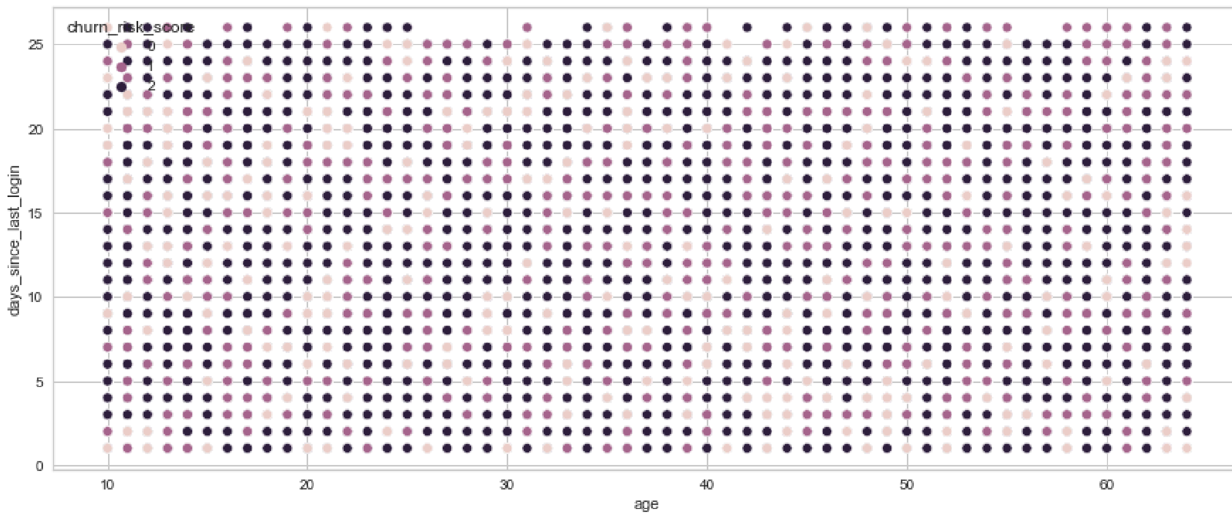
churn_risk_score
0
1
2
10000
8000
6000
4000
2000
0
No Information Available
Not Applicable
Solved
Solved in Follow-up
Unsolved
complaint_status

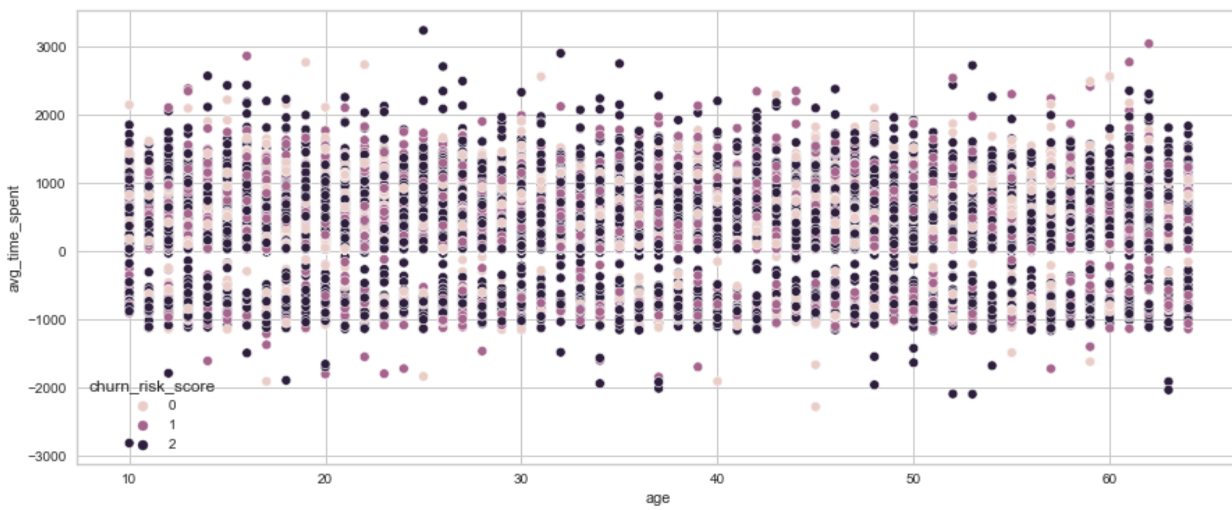


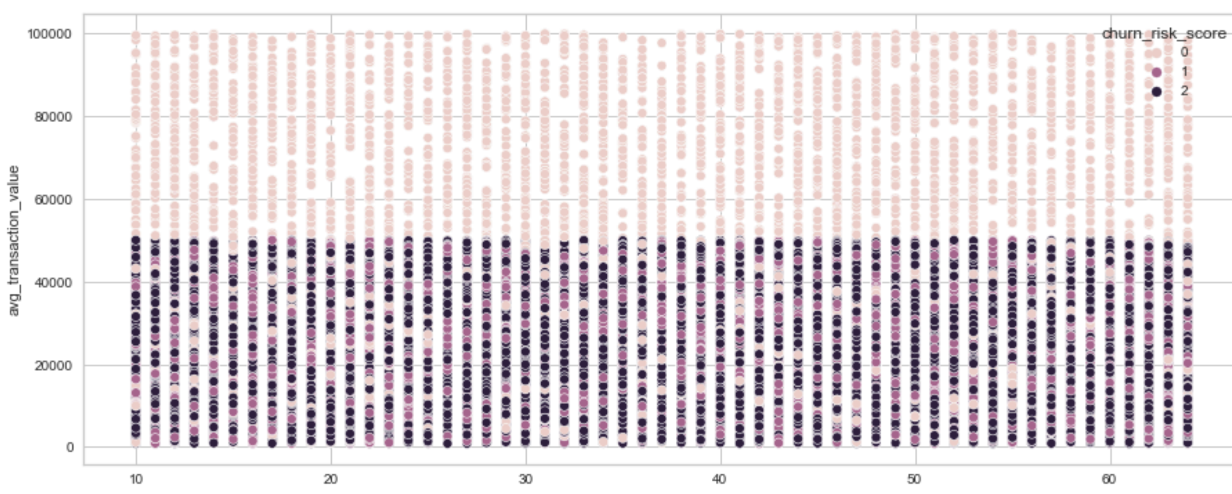


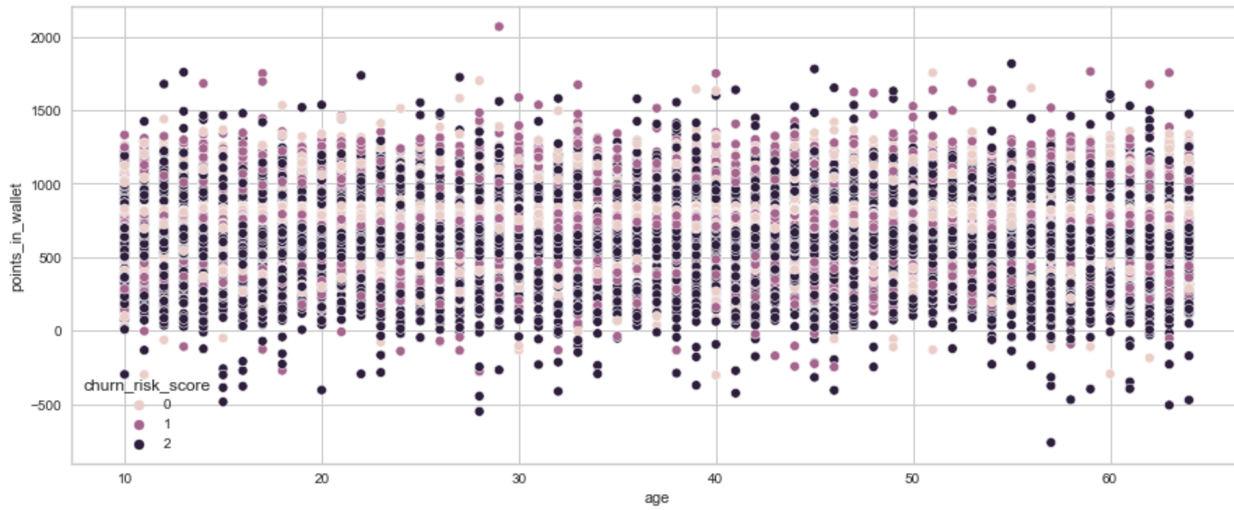
The above plot represents the relationship between the target variable and the other categorical variables.

In [ ]:
```
nrows=9
ncols=2
iterator=1
for i in Numerical_columns:
        if i!='age':
            sns.scatterplot(x=df.age,y=df.loc[:,i],hue=df.churn_risk_score)
            plt.show()
```


churn_risk_score
0
1
2
days_since_last_login
age


churn_risk_score
0
1
2
avg_time_spent
age


churn_risk_score
0
1
2
avg_transaction_value
age



The Above plot represents the relationship between the target variable and the other numerical variables.

Inference

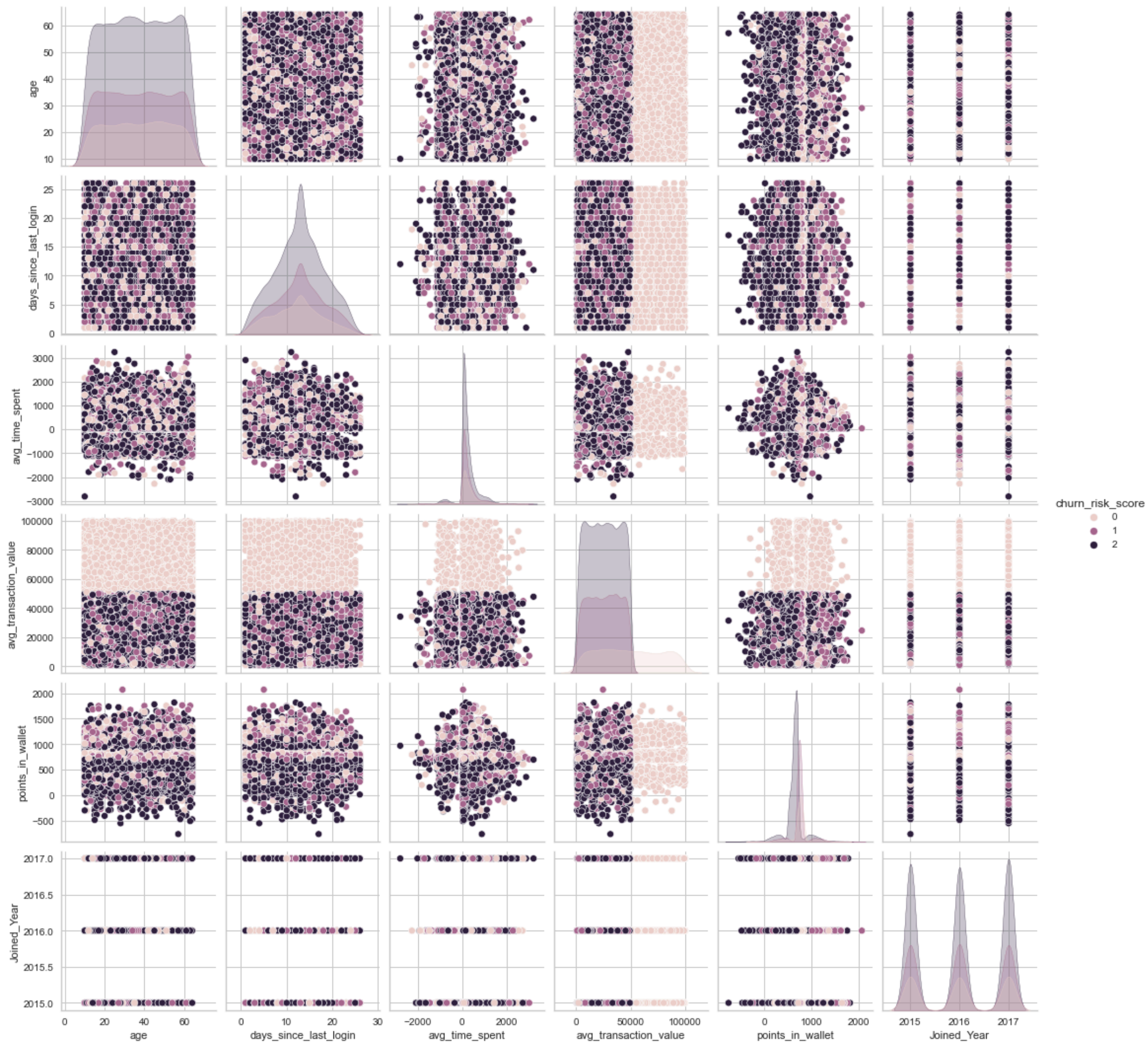
- In Gender Both values holds equal weightage for churn risk rate
- In Region category Churn risk rate is high fot town
- Basic and No membership having high churn risk rate
- Platinum and premium holds low churn risk rate
- Silver and Gold holds both low as well as high
- Without offers customers having high churn score wheareas customer using coupons having low churn risk
- Complaint Status Not applicable holds high churn score
- Postive feedback having low churn score whereas Negative feedback having high churn score
- Customers with average transaction value greater than 50000 holds

low churn score

In [ ]:
```
plt.figure(figsize=(20,20))
sns.pairplot(df, hue='churn_risk_score')
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x1bd4f018fd0>
```

```
<Figure size 1440x1440 with 0 Axes>
```



The average transaction value is holding maximum range for churn risk score 0 irrespective of the age, days_since_last_login, avg_time_Spent, points_in_wallet. And the values are low for churn risk score 1,2. We are able to see the seperation of clusters between 0 and 1,2 after bucketising. The points_in_wallet shows the dominance of cluster 0 for points above 500, whereas cluster 2 is showing its dominance for points below 500.

In [ ]:
```
for col in Caterogical_columns:
    print("--------------------------"+str(col)+"--------------------------")
```

```
    print("unique Values :" ,df[col].unique()) # to print categories name only
    print("Value counts of unique values :\n",df[col].value_counts()) # to pri
    print("-----------------------------------------------------------------------
```

```
-------------------------gender-------------------------
unique Values : ['F' 'M' 'Unknown']
Value counts of unique values :
 F          18348
M          18298
Unknown       58
Name: gender, dtype: int64
--------------------------------------------------------------------------------
----------
-------------------------region_category-------------------------
unique Values : ['Village' 'City' 'Town']
Value counts of unique values :
 Town       19404
City       12635
Village     4665
Name: region_category, dtype: int64
--------------------------------------------------------------------------------
----------
-------------------------membership_category-------------------------
unique Values : ['Platinum Membership' 'Premium Membership' 'No Membership'
 'Gold Membership' 'Silver Membership' 'Basic Membership']
Value counts of unique values :
 Basic Membership       7662
No Membership          7632
Gold Membership        6742
Silver Membership      5935
Premium Membership     4427
Platinum Membership    4306
Name: membership_category, dtype: int64
--------------------------------------------------------------------------------
----------
-------------------------joined_through_referral-------------------------
unique Values : ['No' 'Yes']
Value counts of unique values :
 No     21126
Yes    15578
Name: joined_through_referral, dtype: int64
--------------------------------------------------------------------------------
----------
-------------------------preferred_offer_types-------------------------
unique Values : ['Gift Vouchers/Coupons' 'Credit/Debit Card Offers' 'Without Of
fers']
Value counts of unique values :
 Gift Vouchers/Coupons       12349
Credit/Debit Card Offers    12274
Without Offers              12081
Name: preferred_offer_types, dtype: int64
--------------------------------------------------------------------------------
----------
-------------------------medium_of_operation-------------------------
unique Values : ['Desktop' 'Smartphone' 'Both']
Value counts of unique values :
 Desktop       19154
Smartphone    13766
```

```
Both           3784
Name: medium_of_operation, dtype: int64
--------------------------------------------------------------------------------
----------
--------------------------internet_option--------------------------
unique Values : ['Wi-Fi' 'Mobile_Data' 'Fiber_Optic']
Value counts of unique values :
 Wi-Fi          12310
Mobile_Data    12247
Fiber_Optic    12147
Name: internet_option, dtype: int64
--------------------------------------------------------------------------------
----------
--------------------------used_special_discount--------------------------
unique Values : ['Yes' 'No']
Value counts of unique values :
 Yes    20182
No     16522
Name: used_special_discount, dtype: int64
--------------------------------------------------------------------------------
----------
--------------------------offer_application_preferenc
e--------------------------
unique Values : ['Yes' 'No']
Value counts of unique values :
 Yes    20282
No     16422
Name: offer_application_preference, dtype: int64
--------------------------------------------------------------------------------
----------
--------------------------past_complaint--------------------------
unique Values : ['No' 'Yes']
Value counts of unique values :
 No     18446
Yes    18258
Name: past_complaint, dtype: int64
--------------------------------------------------------------------------------
----------
--------------------------complaint_status--------------------------
unique Values : ['Not Applicable' 'Solved' 'Solved in Follow-up' 'Unsolved'
 'No Information Available']
Value counts of unique values :
 Not Applicable              18446
Unsolved                    4615
Solved                      4579
Solved in Follow-up         4542
No Information Available    4522
Name: complaint_status, dtype: int64
--------------------------------------------------------------------------------
----------
--------------------------feedback--------------------------
unique Values : ['Products always in Stock' 'Quality Customer Care' 'Poor Websi
te'
 'No reason specified' 'Poor Product Quality' 'Poor Customer Service'
```

```
 'Too many ads' 'User Friendly Website' 'Reasonable Price']
Value counts of unique values :
 Poor Product Quality        6304
No reason specified         6234
Too many ads                6230
Poor Website                6226
Poor Customer Service       6195
Reasonable Price            1408
User Friendly Website       1382
Products always in Stock    1371
Quality Customer Care       1354
Name: feedback, dtype: int64
--------------------------------------------------------------------------------
----------
--------------------------churn_risk_score--------------------------
unique Values : [0 2 1]
Value counts of unique values :
 2    20439
1    10339
0     5926
Name: churn_risk_score, dtype: int64
--------------------------------------------------------------------------------
----------
```

In [ ]:
```
#df1.age.describe()# Have to split age in to Teen = 13-19 yrs. Adult = 20-39 y
```

def age_categorize(age):

```
       return('Child')
    elif (age<=19):
        return('Teen')
    elif (age<=39):
        return('Adult')
    elif (age<=59):
        return('Middle_Age_Adult')
    else:
        return('Senior_Adult')
```

df['age_category']=df.age.apply(age_categorize)

Skewness

In [ ]:
```
df.skew()
```

Out[ ]:
```
age                       -0.007368
days_since_last_login      0.021134
avg_time_spent             0.538800
avg_transaction_value      1.009753
points_in_wallet          -0.102518
churn_risk_score          -0.790561
Joined_Year               -0.011602
dtype: float64
```

Data in various columns are postively as well as negatively skewed

Outlier

In [ ]:
```
q1=df.quantile(.25)
q3=df.quantile(.75)
IQR=q3-q1
ll=q1-1.5*IQR
ul=q3+1.5*IQR
wt_outliers=df.loc[((df>ul)|(df<ll)).any(axis=1)]
wt_outliers.shape
```

Out[ ]:
```
(9620, 19)
```

We are having of outliers in our data of around 9620 rows but we keep our outliers in our data

Statistical Test

Null hypothesis (Ho) : Predictor and Target are Independent Alternate hypothesis (Ha) : Predictor and Target are Dependent Confidence_Interval : 0.95 level_of_significance : 0.05

In [ ]:
```
from scipy.stats import shapiro,levene,contingency,chisquare,ttest_ind,f_onewa
```

In [ ]:
```
num_cols=df.select_dtypes(include=np.number).columns
cat_columns=df.select_dtypes(include=np.object_).columns
```

In [ ]:
```
num_cols
```

Out[ ]:
```
Index(['age', 'days_since_last_login', 'avg_time_spent',
       'avg_transaction_value', 'points_in_wallet', 'Joined_Year'],
      dtype='object')
```

In [ ]:
```
import scipy.stats as stats


signif_feats1=[]
test_stats1=[]
p_value1=[]
signif_feats2=[]
```

```
test_stats2=[]
p_value2=[]
for i in num_cols:
    one=df.loc[df.churn_risk_score==0,i]
    two=df.loc[df.churn_risk_score==1,i]
    three=df.loc[df.churn_risk_score==2,i]
    teststats,pvalue=stats.f_oneway(one,two,three)
    if pvalue <0.05:
        signif_feats1.append(i)
        test_stats1.append(teststats)
        p_value1.append(pvalue)
    else:
        signif_feats2.append(i)
        test_stats2.append(teststats)
        p_value2.append(pvalue)

for i in cat_columns:
    if i!='churn_risk_score':
        test_stat, pvalue, dof, expected_value = chi2_contingency(pd.crosstab(
        if pvalue <0.05:
            signif_feats1.append(i)
            test_stats1.append(teststats)
            p_value1.append(pvalue)
        else:
            signif_feats2.append(i)
            test_stats2.append(teststats)
            p_value2.append(pvalue)


Dependent_Features=pd.DataFrame({'Features':signif_feats1,'Test_Statistics':te
Independent_Features=pd.DataFrame({'Features':signif_feats2,'Test_Statistics':
```

In [ ]:
```
print(Dependent_Features)
```

```
                     Features  Test_Statistics        PValue
0         days_since_last_login        34.385545  1.203709e-15
1                avg_time_spent        24.024012  3.743933e-11
2         avg_transaction_value      4749.020578  0.000000e+00
3               points_in_wallet      1781.980356  0.000000e+00
4                region_category         0.783061  2.894598e-18
5            membership_category         0.783061  0.000000e+00
6         joined_through_referral         0.783061  1.832495e-26
7           preferred_offer_types         0.783061  5.892588e-64
8            medium_of_operation         0.783061  6.357438e-17
9          used_special_discount         0.783061  4.818971e-02
10  offer_application_preference         0.783061  2.566210e-20
11                past_complaint         0.783061  4.036414e-02
12                      feedback         0.783061  0.000000e+00
```

In [ ]:
```
print(Independent_Features)
```

```
              Features  Test_Statistics    PValue
0                  age         0.702638  0.495284
1          Joined_Year         0.783061  0.457013
2               gender         0.783061  0.606347
3      internet_option         0.783061  0.314392
4     complaint_status         0.783061  0.195914
```

In [ ]:
```
df_new=df.drop(columns=['age','gender','internet_option','complaint_status','J
```

In [ ]:
```
df_new.shape
```

Out[ ]:
```
(36704, 14)
```

Scaling

In [ ]:
```
df1=df_new.select_dtypes(include=np.number)
ss=StandardScaler()
df_s=ss.fit_transform(df1)
df_s=pd.DataFrame(df_s,columns=df1.columns,index=df1.index)
df_s.head()

df_s
```

Out[ ]:

	days_since_last_login	avg_time_spent	avg_transaction_value	points_in_w
0	0.778932	0.143803	1.220169	0.5(
1	0.594431	0.158149	-0.845189	0.0!
2	0.225429	0.685315	-0.424135	-1.0:
3	-0.328075	-0.477681	-0.207527	-0.6!
4	1.332435	-0.327285	-0.246395	-0.1:
...	...	...	...	
36987	-1.988585	-2.246341	-0.102728	-0.2(
36988	0.040928	-2.214786	-0.936133	-0.8(
36989	-0.143573	-0.222239	0.455167	-0.0
36990	0.409930	0.601022	-1.383012	-2.6!
36991	0.409930	-0.412583	-1.392740	0.1

36704 rows × 4 columns

In [ ]:
```
df_kk=pd.concat([df_s,df.churn_risk_score],axis=1)
df_kk['churn_risk_score']=df_kk.churn_risk_score.astype('int')
```

Encoding

In [ ]:
```
df_cat = df.select_dtypes(include=[np.object])
df_cat=df_cat.drop(['churn_risk_score'],axis=1)
for i in df_cat.columns:
    df_cat[i]=LabelEncoder().fit_transform(df_cat[i])
df_cat.head()
```

Out[ ]:

	gender	region_category	membership_category	joined_through_referral	pref
0	0	2	3	0	
1	0	0	4	0	
2	0	1	2	1	
3	1	0	2	1	
4	0	0	2	0	

In [ ]:
```
#df_cat=pd.get_dummies(df_new,drop_first=True)
#df_cat.drop(columns=['joining_date','last_visit_time'],inplace=True)
```

In [ ]:
```
df_new1=pd.concat([df_kk,df_cat],axis=1)
df_new1
```

Out[ ]:

	days_since_last_login	avg_time_spent	avg_transaction_value	points_in_w
0	0.778932	0.143803	1.220169	0.5
1	0.594431	0.158149	-0.845189	0.0
2	0.225429	0.685315	-0.424135	-1.0
3	-0.328075	-0.477681	-0.207527	-0.6
4	1.332435	-0.327285	-0.246395	-0.1
...	...	...	...	
36987	-1.988585	-2.246341	-0.102728	-0.2
36988	0.040928	-2.214786	-0.936133	-0.8
36989	-0.143573	-0.222239	0.455167	-0.0
36990	0.409930	0.601022	-1.383012	-2.6
36991	0.409930	-0.412583	-1.392740	0.1

36704 rows × 17 columns

In [ ]:
```
X=df_new1.drop('churn_risk_score',1)
y=df.churn_risk_score.astype('int')
```

In [ ]:
```
y.dtype
```
Out[ ]:
```
dtype('int32')
```
In [ ]:
```
print(X.shape,y.shape)
```
```
(36704, 16) (36704,)
```
In [ ]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```
```
(29363, 16)
(7341, 16)
(29363,)
(7341,)
```
In [ ]:
```
X.dtypes
```
Out[ ]:
```
days_since_last_login           float64
avg_time_spent                  float64
avg_transaction_value           float64
points_in_wallet                float64
gender                            int32
region_category                   int32
membership_category               int32
joined_through_referral           int32
preferred_offer_types             int32
medium_of_operation               int32
internet_option                   int32
used_special_discount             int32
offer_application_preference      int32
past_complaint                    int32
complaint_status                  int32
feedback                          int32
dtype: object
```
In [ ]:
```
vif=[]
for i in range (0,df_s.shape[1]):
    vif.append(variance_inflation_factor(df_s.values,i))
pd.DataFrame({'features':df_s.columns,'VIF':vif})
```
Out[ ]:

	features	VIF
0	days_since_last_login	1.008061
1	avg_time_spent	1.008226
2	avg_transaction_value	1.007821
3	points_in_wallet	1.007175

VIF score for all values are less than 10,hence there is no multicollinearity

Model Building

List of Models:

- LogisticRegression
- DecisionTreeClassifier
- RandomForestClassifier
- ExtraTreesClassifier
- XGBClassifier
- LGBMClassifier
- AdaBoostClassifier
- GradientBoostingClassifier

In [ ]:
```
score_card = pd.DataFrame(columns=['Model', 'Precision Score', 'Recall Score',
                                   'False Negatives', 'Kappa Score', 'f1-score
```

In [ ]:
```
def update_score_card(model, FN_values, model_name):
    y_pred = model.predict(X_test)
    global score_card
    score_card = score_card.append({'Model': model_name,
                                    'Precision Score': precision_score(y_test,
                                    'Recall Score': recall_score(y_test, y_pre
                                    'False Negatives': FN_values,
                                    'Kappa Score': cohen_kappa_score(y_test, y
                                    'f1-score': f1_score(y_test, y_pred, avera
                                    ignore_index = True)
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
```

In [ ]:
```
LR = LogisticRegression(multi_class='multinomial', solver='lbfgs')
LR_Model=LR.fit(X_train,y_train)
y_pred_xtest=LR_Model.predict(X_test)
print(classification_report(y_test,y_pred_xtest))
```

```
              precision    recall  f1-score   support

           0       0.69      0.58      0.63      1185
           1       0.59      0.47      0.52      2044
           2       0.75      0.85      0.79      4112

    accuracy                           0.70      7341
   macro avg       0.67      0.63      0.65      7341
weighted avg       0.69      0.70      0.69      7341
```

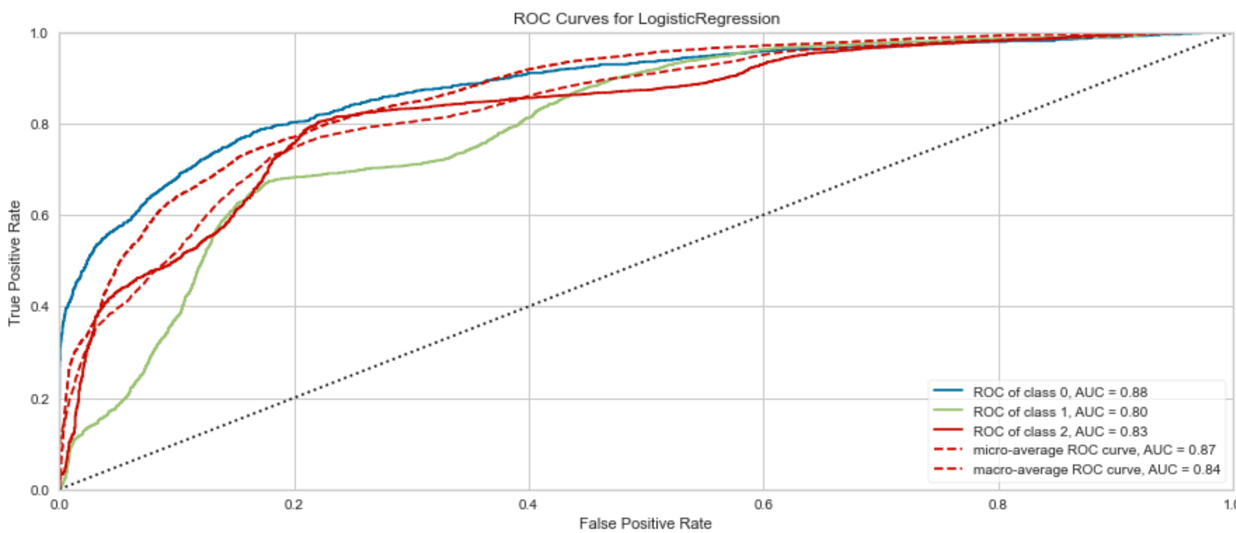
In [ ]:
```
print(accuracy_score(y_test,y_pred_xtest))
```

```
0.7011306361531127
```

In [ ]:
```
print(confusion_matrix(y_test,y_pred_xtest))
```

```
[[ 686  201  298]
 [ 181  967  896]
 [ 134  484 3494]]
```

In [ ]:
```
#For logistic regression model, the roc curve with yellowbrick package
LR_visualizer = ROCAUC(LR_Model)

LR_visualizer.fit(X_train, y_train)        # Fit the training data to the visu
LR_visualizer.score(X_test, y_test)        # Evaluate the model on the test da
LR_visualizer.show()
```

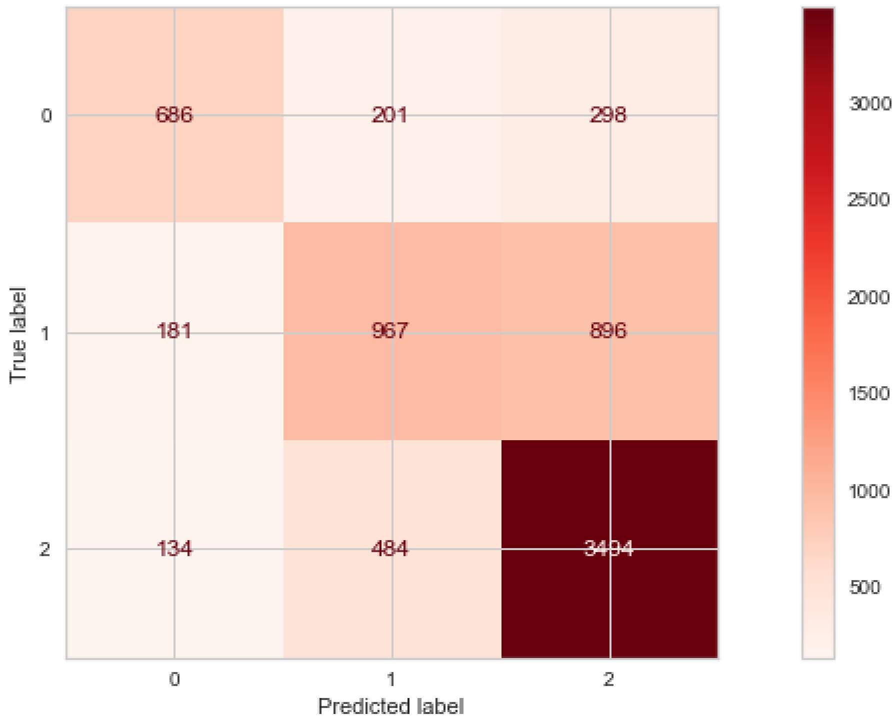


Out[ ]:
```
<AxesSubplot:title={'center':'ROC Curves for LogisticRegression'}, xlabel='Fa
lse Positive Rate', ylabel='True Positive Rate'>
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, auc, mul
```

In [ ]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_xtest, cmap='Reds')
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bd52f0b2
80>
```



In [ ]:
```
LR_mul = multilabel_confusion_matrix(y_test, y_pred_xtest)
LR_mul
```

Out[ ]:
```
array([[[5841,  315],
        [ 499,  686]],

       [[4612,  685],
        [1077,  967]],

       [[2035, 1194],
        [ 618, 3494]]], dtype=int64)
```

In [ ]:
```
LR_FN = LR_mul[2][1][0]
update_score_card(LR_Model, LR_FN,'Logistic Regression')
```

In [ ]:
```
```

In [ ]:
```
```

Decision_Tree

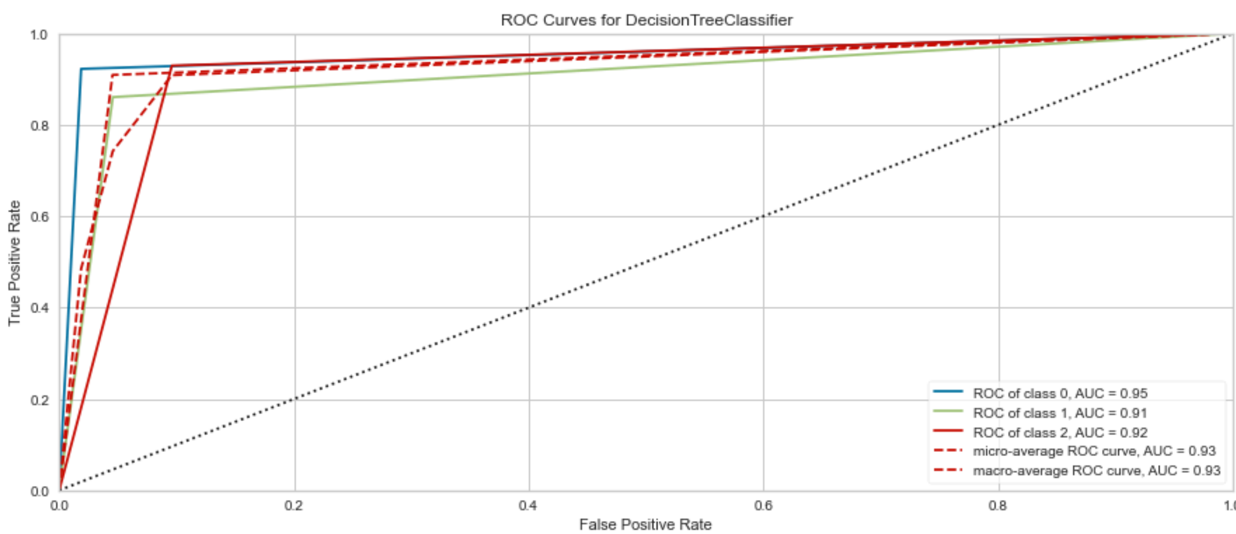
In [ ]:
```
dt=DecisionTreeClassifier()
DT_Model=dt.fit(X_train,y_train)
y_pred_xtest=DT_Model.predict(X_test)
print(classification_report(y_test,y_pred_xtest))
```

```
              precision    recall  f1-score   support

           0       0.90      0.92      0.91      1185
           1       0.88      0.86      0.87      2044
           2       0.92      0.93      0.93      4112

    accuracy                           0.91      7341
   macro avg       0.90      0.90      0.90      7341
weighted avg       0.91      0.91      0.91      7341
```

In [ ]:
```
print(accuracy_score(y_test,y_pred_xtest))
print(confusion_matrix(y_test,y_pred_xtest))
```

```
0.9091404440811879
[[1093   27   65]
 [  40 1759  245]
 [  75  215 3822]]
```

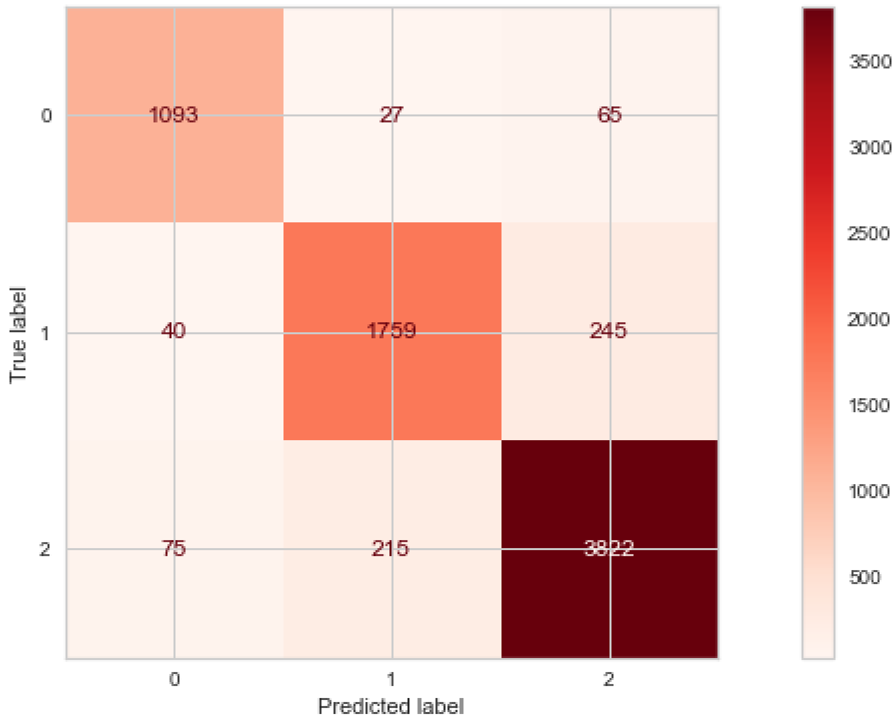
In [ ]:
```
LR_visualizer = ROCAUC(DT_Model)

LR_visualizer.fit(X_train, y_train)        # Fit the training data to the visu
LR_visualizer.score(X_test, y_test)        # Evaluate the model on the test da
LR_visualizer.show()
```



Out[ ]:
```
<AxesSubplot:title={'center':'ROC Curves for DecisionTreeClassifier'}, xlabe
l='False Positive Rate', ylabel='True Positive Rate'>
```

In [ ]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_xtest, cmap='Reds')
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bd55eb7d
60>
```



In [ ]:
```
LR_mul = multilabel_confusion_matrix(y_test, y_pred_xtest)
LR_mul
```

Out[ ]:
```
array([[[6041,  115],
        [  92, 1093]],

       [[5055,  242],
        [ 285, 1759]],

       [[2919,  310],
        [ 290, 3822]]], dtype=int64)
```

In [ ]:
```
LR_FN = LR_mul[2][1][0]
update_score_card(DT_Model, LR_FN,'Decision Tree')
```

RandomForest

In [ ]:
```
rf=RandomForestClassifier()
RF_Model=rf.fit(X_train,y_train)
y_pred_xtest=RF_Model.predict(X_test)
print(classification_report(y_test,y_pred_xtest))
```

```
              precision    recall  f1-score   support

           0       1.00      0.92      0.96      1185
           1       0.88      0.90      0.89      2044
           2       0.94      0.95      0.94      4112

    accuracy                           0.93      7341
   macro avg       0.94      0.92      0.93      7341
weighted avg       0.93      0.93      0.93      7341
```

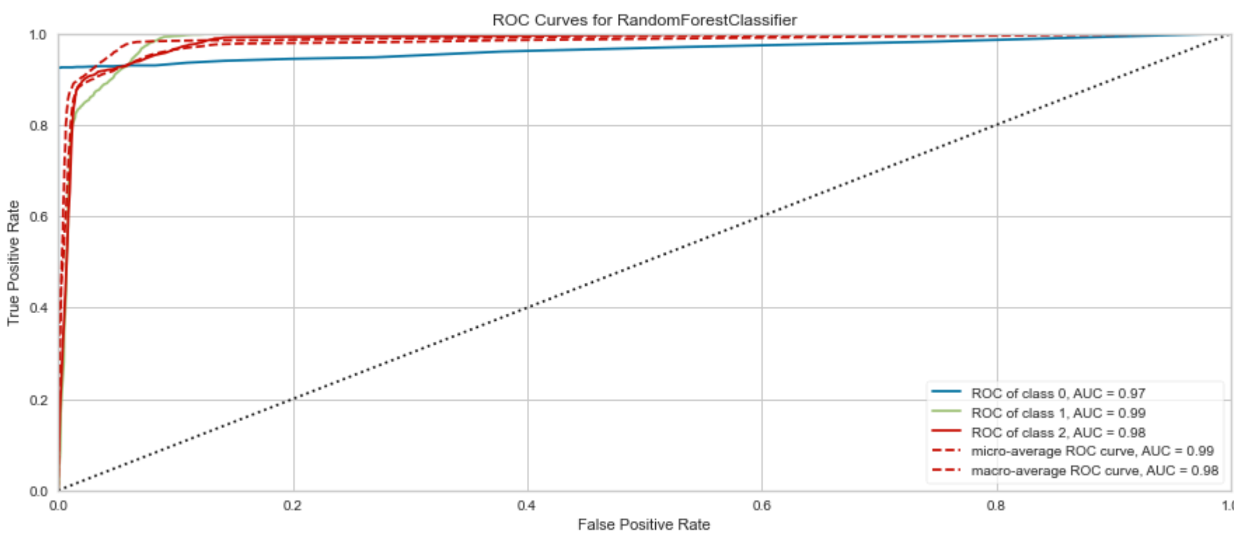
In [ ]:

```
print(accuracy_score(y_test,y_pred_xtest))
print(confusion_matrix(y_test,y_pred_xtest))
```

```
0.9317531671434409
[[1094   31   60]
 [   0 1845  199]
 [   0  211 3901]]
```

In [ ]:

```
LR_visualizer = ROCAUC(RF_Model)

LR_visualizer.fit(X_train, y_train)        # Fit the training data to the visu
LR_visualizer.score(X_test, y_test)        # Evaluate the model on the test da
LR_visualizer.show()
```



Out[ ]:

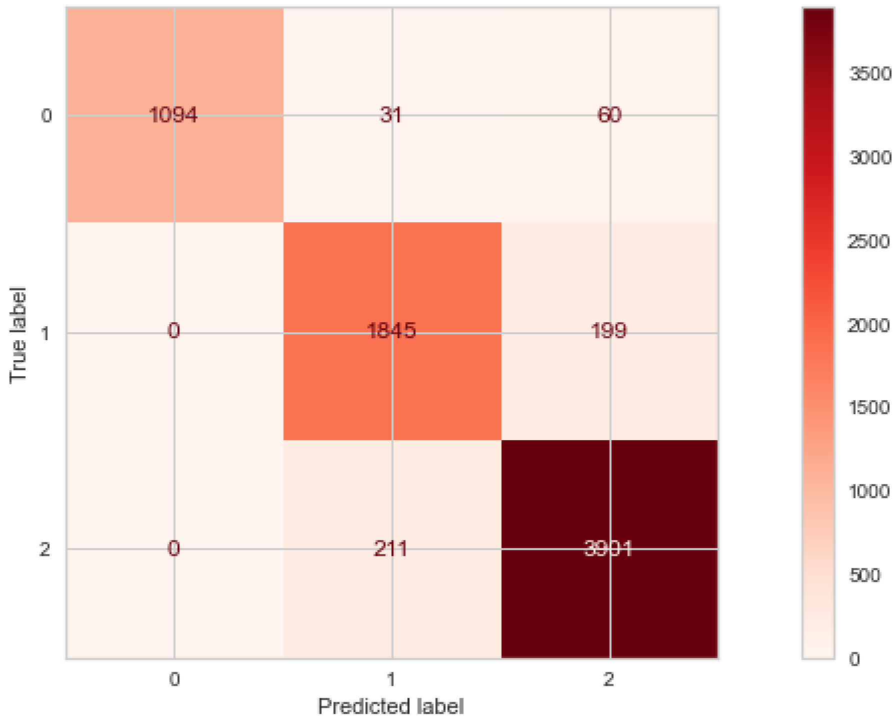
```
<AxesSubplot:title={'center':'ROC Curves for RandomForestClassifier'}, xlabe
l='False Positive Rate', ylabel='True Positive Rate'>
```

In [ ]:

```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_xtest, cmap='Reds')
```

Out[ ]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bd591517
00>
```



In [ ]:
```
LR_mul = multilabel_confusion_matrix(y_test, y_pred_xtest)
LR_mul
```

Out[ ]:
```
array([[[6156,    0],
        [  91, 1094]],

       [[5055,  242],
        [ 199, 1845]],

       [[2970,  259],
        [ 211, 3901]]], dtype=int64)
```

In [ ]:
```
LR_FN = LR_mul[2][1][0]
update_score_card(RF_Model, LR_FN,'Random Forest')
```

ExtraTreesClassifier

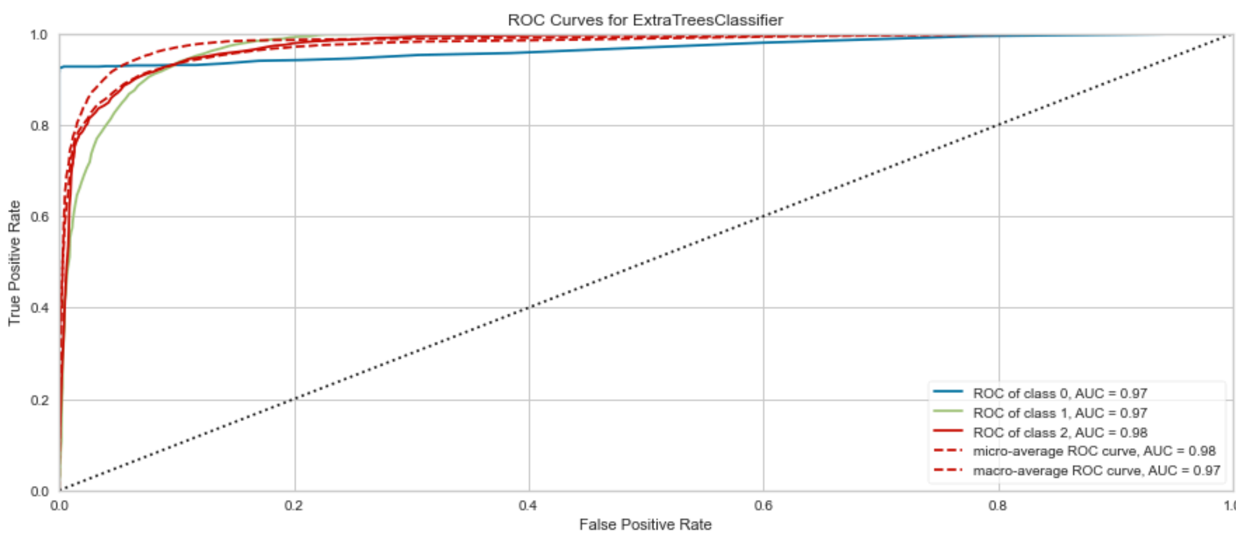
In [ ]:
```
et=ExtraTreesClassifier()
ET_Model=et.fit(X_train,y_train)
y_pred_xtest=ET_Model.predict(X_test)
print(classification_report(y_test,y_pred_xtest))
```

```
              precision    recall  f1-score   support

           0       1.00      0.92      0.96      1185
           1       0.86      0.86      0.86      2044
           2       0.92      0.94      0.93      4112

    accuracy                           0.91      7341
   macro avg       0.93      0.91      0.92      7341
weighted avg       0.92      0.91      0.92      7341
```

In [ ]:
```
print(accuracy_score(y_test,y_pred_xtest))
print(confusion_matrix(y_test,y_pred_xtest))
```

```
0.914997956681651
[[1096   27   62]
 [   0 1765  279]
 [   1  255 3856]]
```

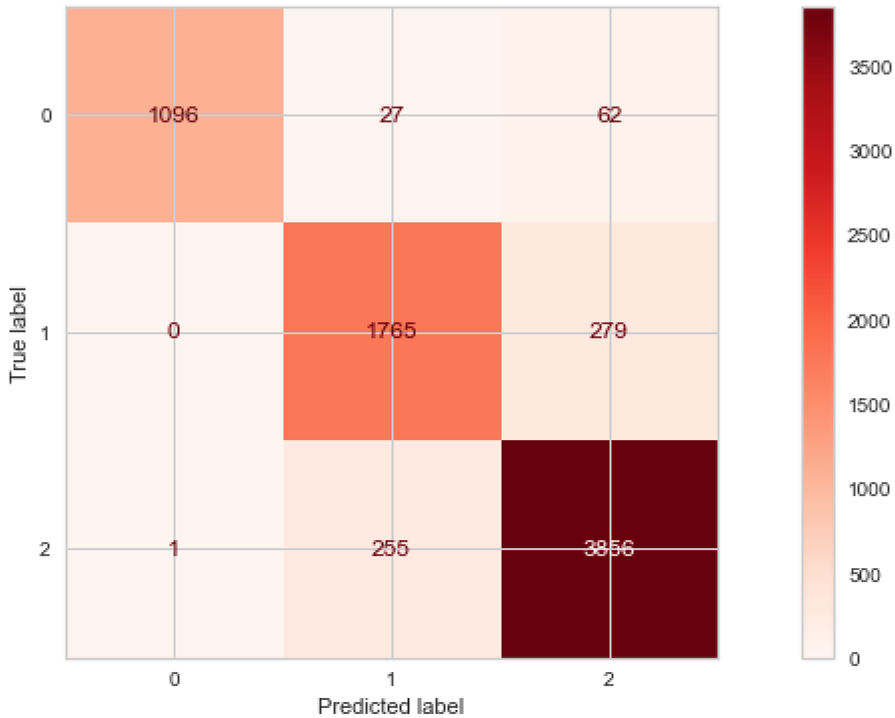
In [ ]:
```
LR_visualizer = ROCAUC(ET_Model)

LR_visualizer.fit(X_train, y_train)        # Fit the training data to the visu
LR_visualizer.score(X_test, y_test)        # Evaluate the model on the test da
LR_visualizer.show()
```



Out[ ]:
```
<AxesSubplot:title={'center':'ROC Curves for ExtraTreesClassifier'}, xlabe
l='False Positive Rate', ylabel='True Positive Rate'>
```

In [ ]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_xtest, cmap='Reds')
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bd5a3a8b
50>
```



In [ ]:
```
LR_mul = multilabel_confusion_matrix(y_test, y_pred_xtest)
LR_mul
```

Out[ ]:
```
array([[[6155,    1],
        [  89, 1096]],

       [[5015,  282],
        [ 279, 1765]],

       [[2888,  341],
        [ 256, 3856]]], dtype=int64)
```

In [ ]:
```
LR_FN = LR_mul[2][1][0]
update_score_card(ET_Model, LR_FN,'Extra Tree')
```

XGBClassifier

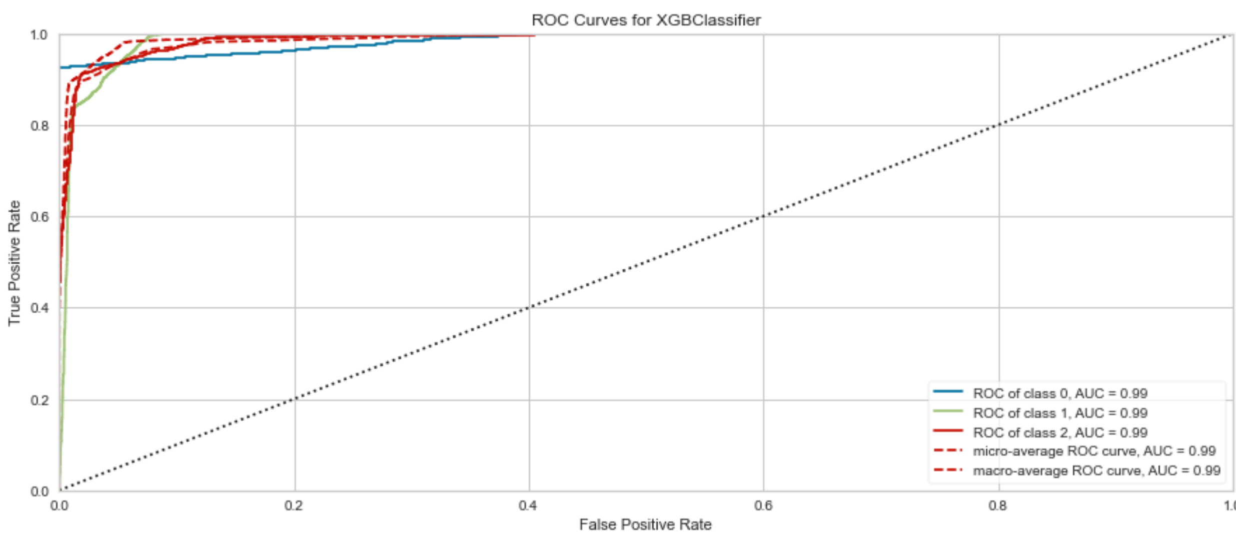
In [ ]:
```
xgb=XGBClassifier()
XGB_Model=xgb.fit(X_train,y_train)
y_pred_xtest=XGB_Model.predict(X_test)
print(classification_report(y_test,y_pred_xtest))
```

```
              precision    recall  f1-score   support

           0       1.00      0.92      0.96      1185
           1       0.88      0.92      0.90      2044
           2       0.95      0.95      0.95      4112

    accuracy                           0.94      7341
   macro avg       0.94      0.93      0.94      7341
weighted avg       0.94      0.94      0.94      7341
```

In [ ]:
```
print(accuracy_score(y_test,y_pred_xtest))
print(confusion_matrix(y_test,y_pred_xtest))
```

```
0.9370657948508377
[[1096   35   54]
 [   0 1887  157]
 [   1  215 3896]]
```

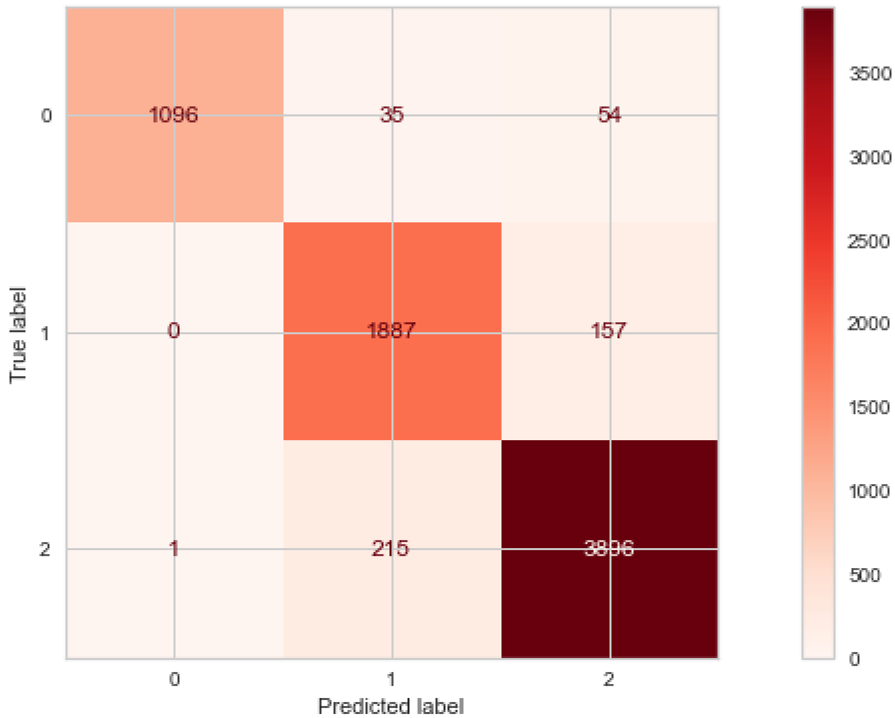
In [ ]:
```
LR_visualizer = ROCAUC(XGB_Model)

LR_visualizer.fit(X_train, y_train)        # Fit the training data to the visu
LR_visualizer.score(X_test, y_test)        # Evaluate the model on the test da
LR_visualizer.show()
```



Out[ ]:
```
<AxesSubplot:title={'center':'ROC Curves for XGBClassifier'}, xlabel='False P
ositive Rate', ylabel='True Positive Rate'>
```

In [ ]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_xtest, cmap='Reds')
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bd5a4517
60>
```




```
In [ ]: LR_mul = multilabel_confusion_matrix(y_test, y_pred_xtest)
        LR_mul
```

```
Out[ ]: array([[[6155,    1],
                [  89, 1096]],

               [[5047,  250],
                [ 157, 1887]],

               [[3018,  211],
                [ 216, 3896]]], dtype=int64)
```

```
In [ ]: LR_FN = LR_mul[2][1][0]
        update_score_card(XGB_Model, LR_FN,'XGB')
```

LGBMClassifier

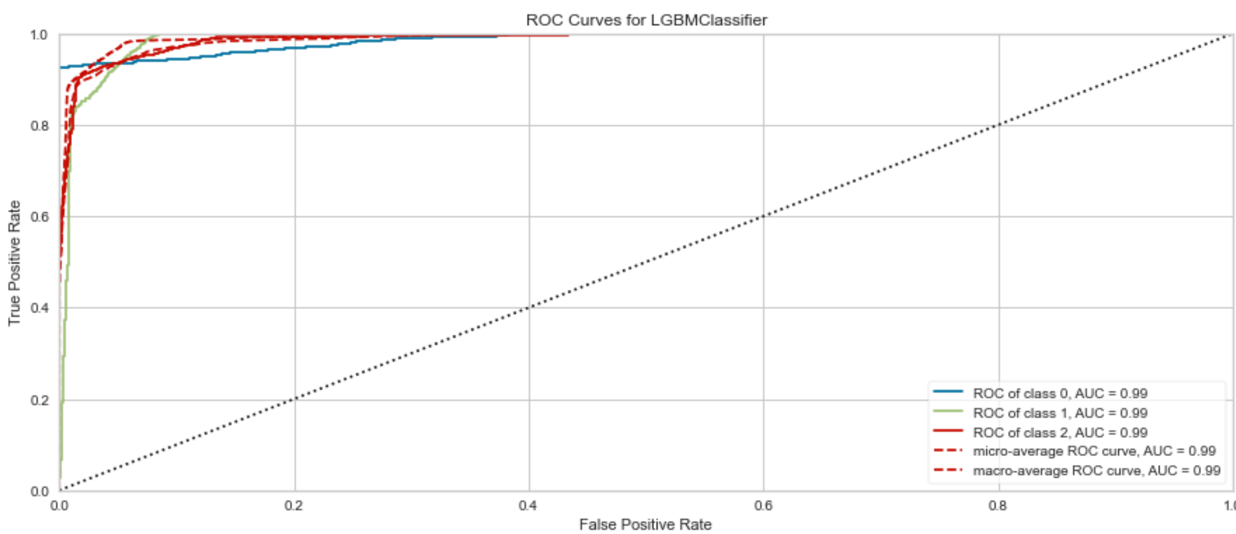
```
In [ ]: lgbm=LGBMClassifier()
        LGBM_Model=lgbm.fit(X_train,y_train)
        y_pred_xtest=LGBM_Model.predict(X_test)
        print(classification_report(y_test,y_pred_xtest))
```

```
              precision    recall  f1-score   support

           0       1.00      0.92      0.96      1185
           1       0.89      0.92      0.90      2044
           2       0.95      0.95      0.95      4112

    accuracy                           0.94      7341
   macro avg       0.94      0.93      0.94      7341
weighted avg       0.94      0.94      0.94      7341
```

In [ ]:
```
print(accuracy_score(y_test,y_pred_xtest))
print(confusion_matrix(y_test,y_pred_xtest))
```

```
0.9361122462879716
[[1096   33   56]
 [   1 1872  171]
 [   0  208 3904]]
```

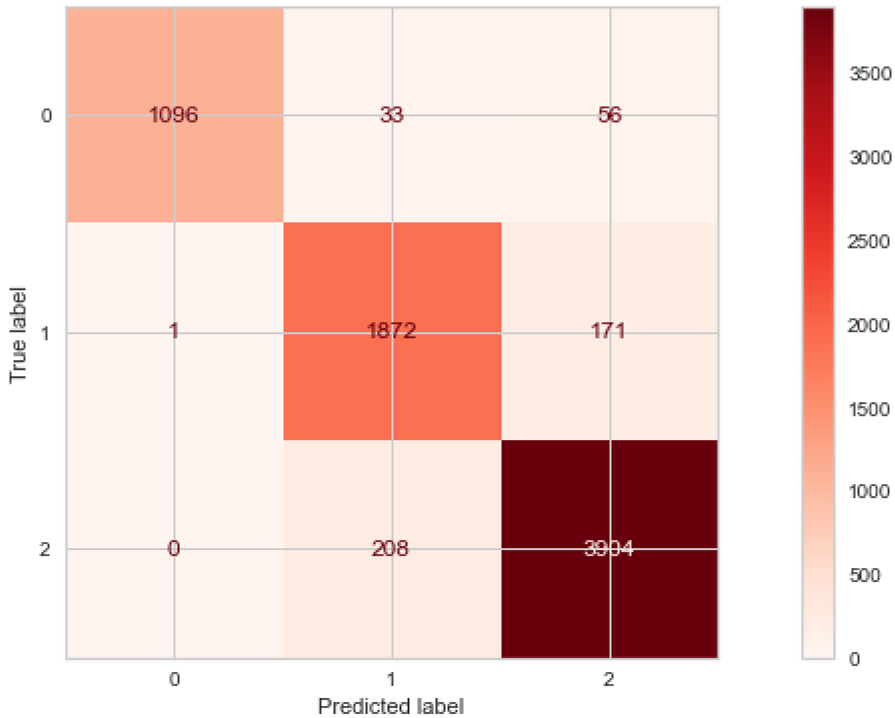
In [ ]:
```
LR_visualizer = ROCAUC(LGBM_Model)

LR_visualizer.fit(X_train, y_train)        # Fit the training data to the visu
LR_visualizer.score(X_test, y_test)        # Evaluate the model on the test da
LR_visualizer.show()
```



Out[ ]: <AxesSubplot:title={'center':'ROC Curves for LGBMClassifier'}, xlabel='False Positive Rate', ylabel='True Positive Rate'>

In [ ]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_xtest, cmap='Reds')
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bd5a8fab80>



In [ ]:
```
LR_mul = multilabel_confusion_matrix(y_test, y_pred_xtest)
LR_mul
```

Out[ ]:
```
array([[[6155,    1],
        [  89, 1096]],

       [[5056,  241],
        [ 172, 1872]],

       [[3002,  227],
        [ 208, 3904]]], dtype=int64)
```

In [ ]:
```
LR_FN = LR_mul[2][1][0]
update_score_card(LGBM_Model, LR_FN,'LGBM')
```

AdaBoostClassifier

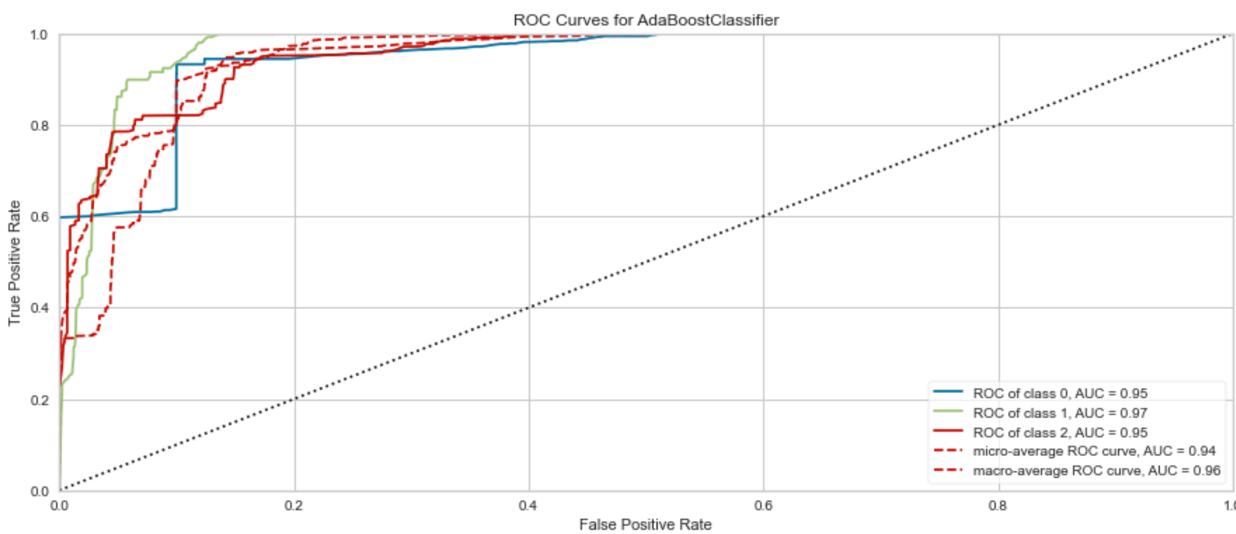
In [ ]:
```
ada=AdaBoostClassifier()
ADA_Model=ada.fit(X_train,y_train)
y_pred_xtest=ADA_Model.predict(X_test)
print(classification_report(y_test,y_pred_xtest))
```

```
              precision    recall  f1-score   support

           0       1.00      0.92      0.96      1185
           1       0.87      0.91      0.89      2044
           2       0.94      0.94      0.94      4112

    accuracy                           0.93      7341
   macro avg       0.94      0.93      0.93      7341
weighted avg       0.93      0.93      0.93      7341
```

In [ ]:
```
print(accuracy_score(y_test,y_pred_xtest))
print(confusion_matrix(y_test,y_pred_xtest))
```

```
0.9298460700177088
[[1096   33   56]
 [   0 1856  188]
 [   0  238 3874]]
```

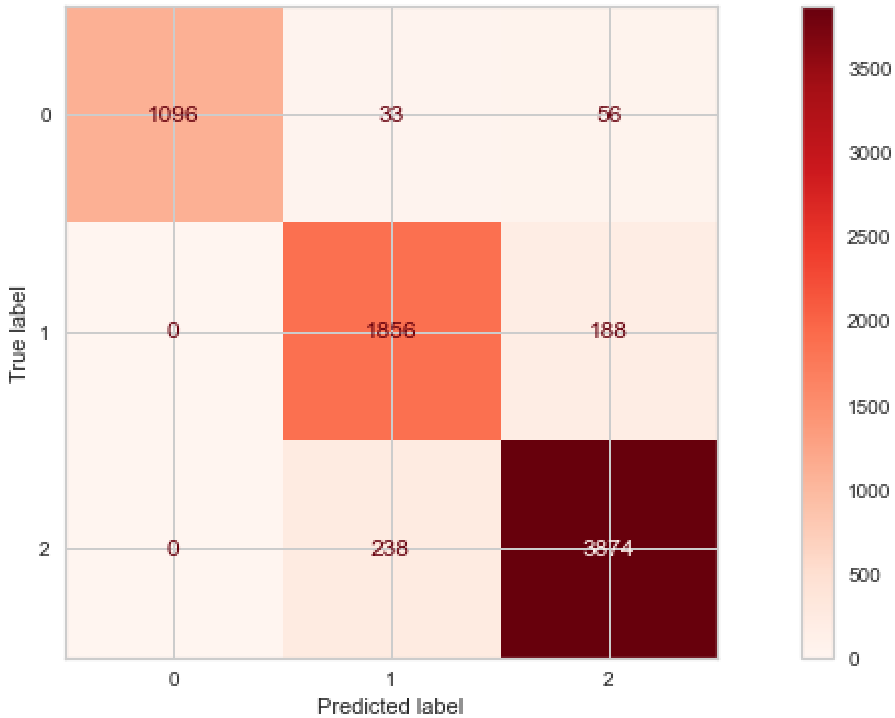
In [ ]:
```
LR_visualizer = ROCAUC(ADA_Model)

LR_visualizer.fit(X_train, y_train)        # Fit the training data to the visu
LR_visualizer.score(X_test, y_test)        # Evaluate the model on the test da
LR_visualizer.show()
```



Out[ ]:
```
<AxesSubplot:title={'center':'ROC Curves for AdaBoostClassifier'}, xlabel='Fa
lse Positive Rate', ylabel='True Positive Rate'>
```

In [ ]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_xtest, cmap='Reds')
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bd5a3a8a
00>
```



In [ ]:
```
LR_mul = multilabel_confusion_matrix(y_test, y_pred_xtest)
LR_mul
```

Out[ ]:
```
array([[[6156,    0],
        [  89, 1096]],

       [[5026,  271],
        [ 188, 1856]],

       [[2985,  244],
        [ 238, 3874]]], dtype=int64)
```

In [ ]:
```
LR_FN = LR_mul[2][1][0]
update_score_card(ADA_Model, LR_FN,'ADA Boosting')
```

GradientBoostingClassifier

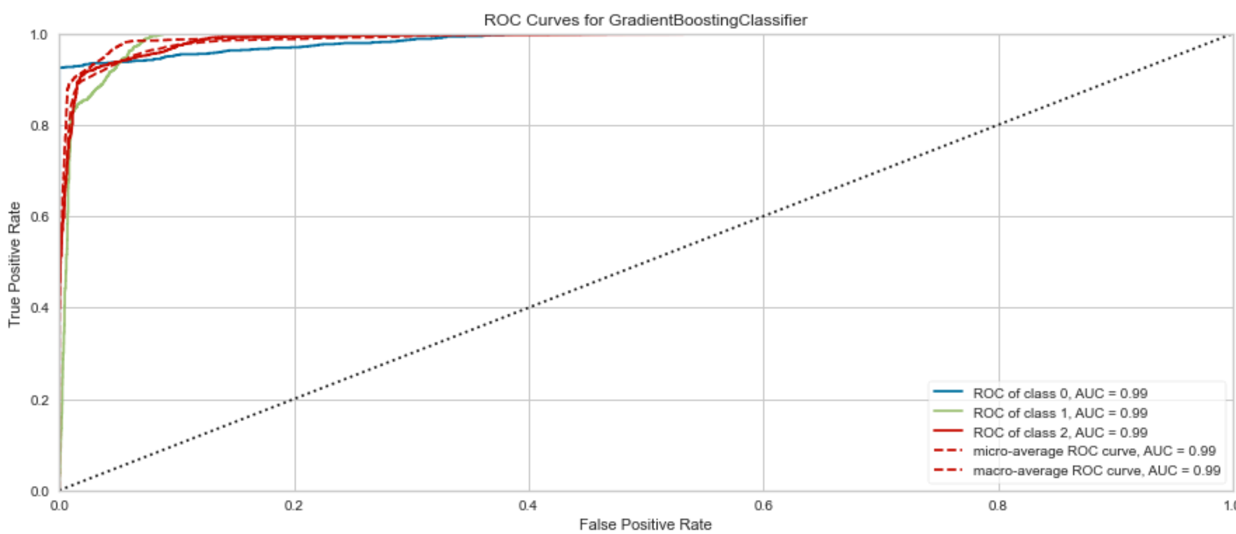
In [ ]:
```
gb=GradientBoostingClassifier()
GB_Model=gb.fit(X_train,y_train)
y_pred_xtest=GB_Model.predict(X_test)
print(classification_report(y_test,y_pred_xtest))
```

```
              precision    recall  f1-score   support

           0       1.00      0.92      0.96      1185
           1       0.89      0.91      0.90      2044
           2       0.94      0.95      0.95      4112

    accuracy                           0.94      7341
   macro avg       0.94      0.93      0.94      7341
weighted avg       0.94      0.94      0.94      7341
```

In [ ]:
```
print(accuracy_score(y_test,y_pred_xtest))
print(confusion_matrix(y_test,y_pred_xtest))
```

```
0.9367933524043046
[[1096   35   54]
 [   0 1870  174]
 [   0  201 3911]]
```

In [ ]:
```
LR_visualizer = ROCAUC(GB_Model)

LR_visualizer.fit(X_train, y_train)        # Fit the training data to the visu
LR_visualizer.score(X_test, y_test)        # Evaluate the model on the test da
LR_visualizer.show()
```



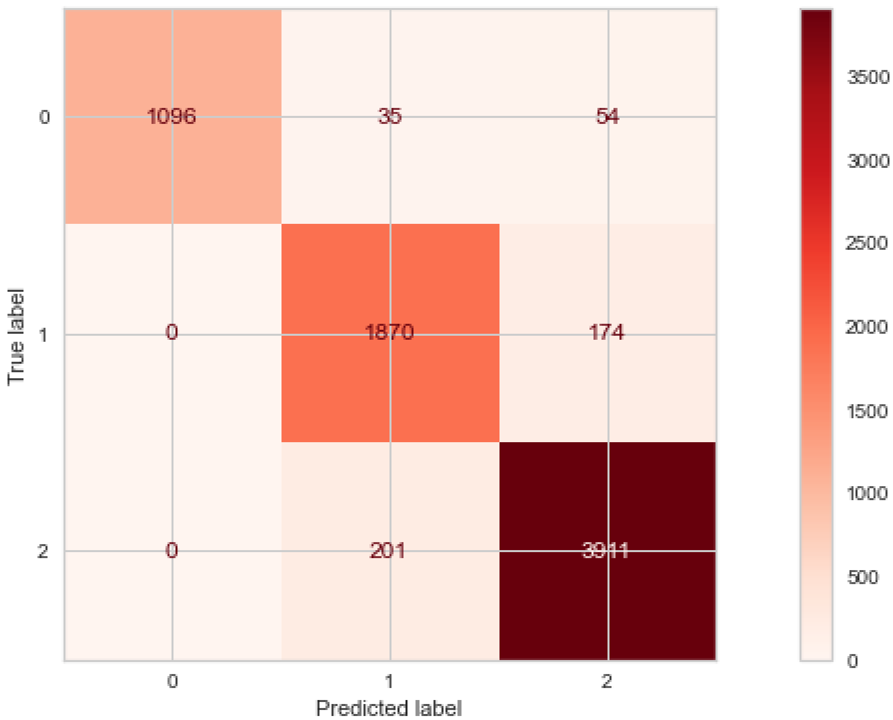
Out[ ]:
```
<AxesSubplot:title={'center':'ROC Curves for GradientBoostingClassifier'}, xl
abel='False Positive Rate', ylabel='True Positive Rate'>
```

In [ ]:
```
LR_mul = multilabel_confusion_matrix(y_test, y_pred_xtest)
LR_mul
```

Out[ ]:
```
array([[[6156,    0],
        [  89, 1096]],

       [[5061,  236],
        [ 174, 1870]],

       [[3001,  228],
        [ 201, 3911]]], dtype=int64)
```

In [ ]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_xtest, cmap='Reds')
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bd70ea8e
50>
```



In [ ]:
```
LR_FN = LR_mul[2][1][0]
update_score_card(GB_Model, LR_FN,'Gradient Boosting')
```

In [ ]:
```
score_card.sort_values(by=['f1-score'],ascending=False)
```

Out[ ]:

	Model	Precision Score	Recall Score	False Negatives	Kappa Score	f1-score
4	XGB	0.943575	0.931852	216	0.891763	0.937088
7	Gradient Boosting	0.944284	0.930295	201	0.891061	0.936731
5	LGBM	0.943361	0.930054	208	0.889948	0.936146
2	Random Forest	0.940595	0.924845	211	0.882204	0.932166
6	ADA Boosting	0.937779	0.925013	238	0.879255	0.93079
3	Extra Tree	0.926692	0.908714	256	0.852936	0.917194
1	Decision Tree	0.902946	0.904135	290	0.843953	0.903478
0	Logistic Regression	0.671991	0.633901	618	0.464028	0.64833

In [ ]:

```
GB_Model.feature_importances_
```

Out[ ]:

```
array([1.73550201e-04, 5.54701251e-04, 1.44876412e-01, 3.87204143e-01,
       2.15691768e-05, 2.67363105e-05, 2.15027256e-01, 6.64687808e-06,
       5.08928017e-05, 2.11453484e-05, 2.42982402e-05, 4.76905085e-06,
       9.81825029e-06, 2.12489040e-05, 1.95503585e-05, 2.51957262e-01])
```

Tuning Parameters

from sklearn.model_selection import GridSearchCV params=[{'criterion':["gini","entropy","log_loss"], 'n_estimators':[100,200,500,1000], 'min_samples_split':[2,4,6,8], 'max_depth':[2,4,6,8]}] rf=RandomForestClassifier() grid=GridSearchCV(estimator=rf,cv=5,param_grid=params) grid.fit(X,y) grid.best_params_

In [ ]:

In [ ]:

```
rf=RandomForestClassifier(criterion='gini',max_depth=8,n_estimators=500,min_sa
RF_Model_ad=rf.fit(X_train,y_train)
y_pred_xtest=RF_Model_ad.predict(X_test)
print(classification_report(y_test,y_pred_xtest))
```

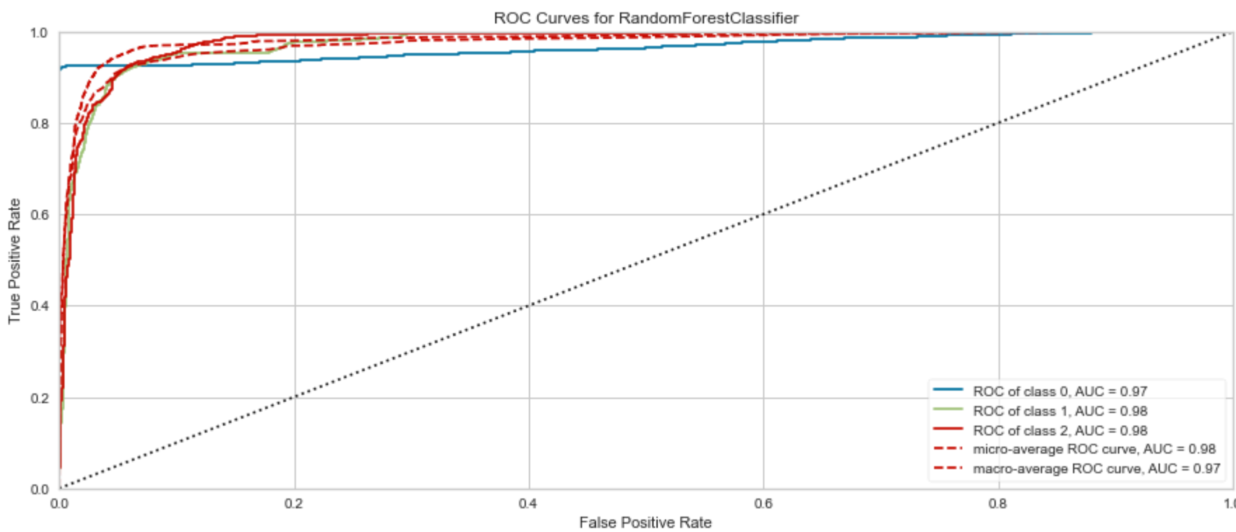
```
              precision    recall  f1-score   support

           0       1.00      0.90      0.95      1185
           1       0.88      0.90      0.89      2044
           2       0.93      0.95      0.94      4112

    accuracy                           0.93      7341
   macro avg       0.94      0.92      0.93      7341
weighted avg       0.93      0.93      0.93      7341
```

In [ ]:

```
print(accuracy_score(y_test,y_pred_xtest))
print(confusion_matrix(y_test,y_pred_xtest))
```

```
0.9276665304454434
[[1072   33   80]
 [   0 1841  203]
 [   0  215 3897]]
```

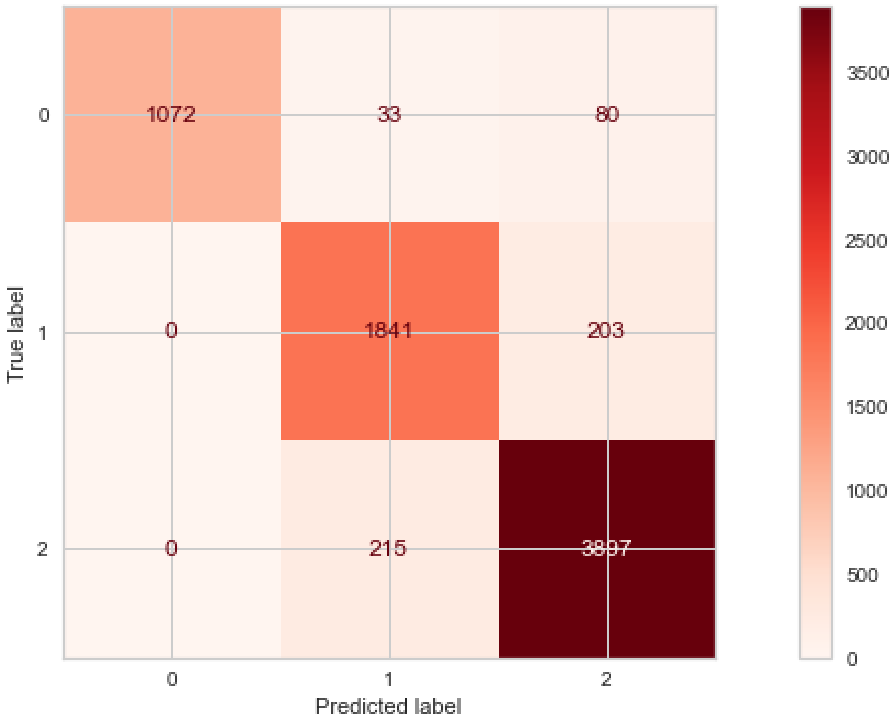
In [ ]:
```
LR_visualizer = ROCAUC(RF_Model_ad)

LR_visualizer.fit(X_train, y_train)        # Fit the training data to the visu
LR_visualizer.score(X_test, y_test)        # Evaluate the model on the test da
LR_visualizer.show()
```



Out[ ]:
```
<AxesSubplot:title={'center':'ROC Curves for RandomForestClassifier'}, xlabe
l='False Positive Rate', ylabel='True Positive Rate'>
```

In [ ]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_xtest, cmap='Reds')
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bd6a9db5
e0>
```



In [ ]:
```
LR_mul = multilabel_confusion_matrix(y_test, y_pred_xtest)
LR_mul
```

Out[ ]:
```
array([[[6156,    0],
        [ 113, 1072]],

       [[5049,  248],
        [ 203, 1841]],

       [[2946,  283],
        [ 215, 3897]]], dtype=int64)
```

In [ ]:
```
LR_FN = LR_mul[2][1][0]
update_score_card(RF_Model_ad, LR_FN,'Random Forest Tuned')
```

In [ ]:

params = {"n_estimators" : [90,100,110,120,130,140,150,200,250], 'learning_rate' : [1.0,0.1,0.01,0.001,0.0001]} clf = AdaBoostClassifier() Grid = GridSearchCV(clf,param_grid=params,cv=5) Grid.fit(X_train, y_train) Grid.best_params_

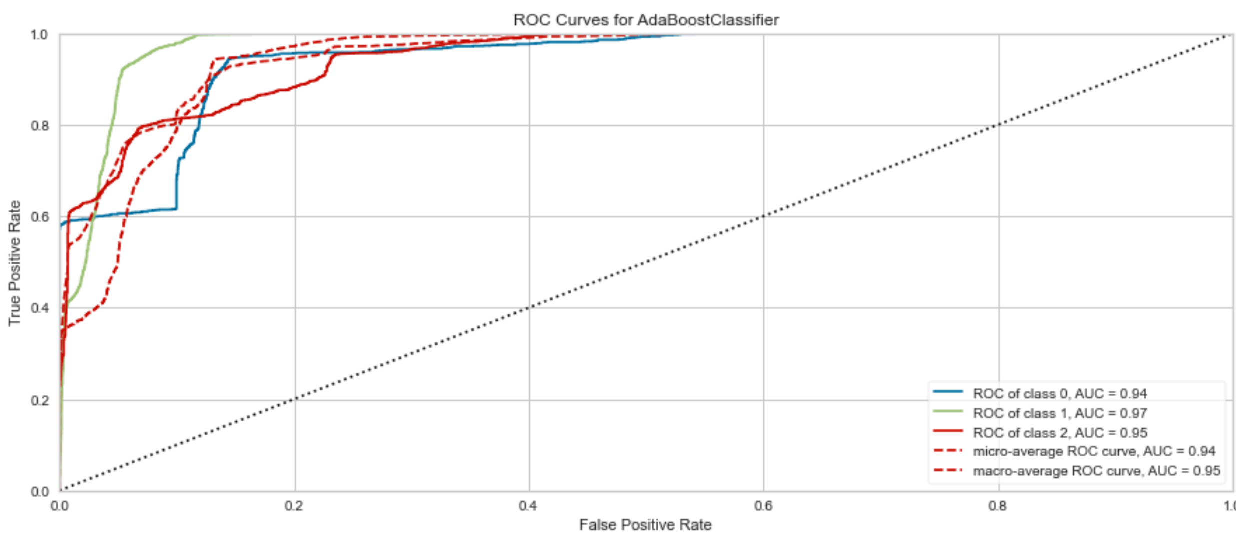
In [ ]:
```
ada=AdaBoostClassifier(learning_rate=1.0,n_estimators=110)
ADA_Model_ad=ada.fit(X_train,y_train)
y_pred_xtest=ADA_Model_ad.predict(X_test)
print(classification_report(y_test,y_pred_xtest))
```

```
              precision    recall  f1-score   support

           0       1.00      0.92      0.96      1185
           1       0.90      0.86      0.88      2044
           2       0.92      0.96      0.94      4112

    accuracy                           0.93      7341
   macro avg       0.94      0.92      0.93      7341
weighted avg       0.93      0.93      0.93      7341
```

In [ ]:
```
print(accuracy_score(y_test,y_pred_xtest))
print(confusion_matrix(y_test,y_pred_xtest))
```

```
0.9291649639013758
[[1096   37   52]
 [   1 1766  277]
 [   2  151 3959]]
```

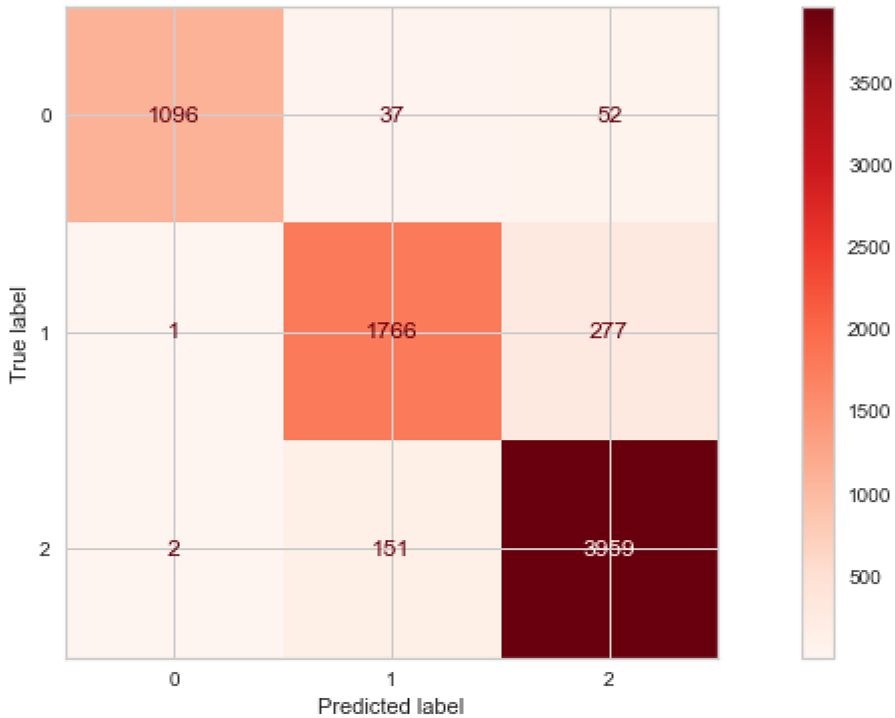
In [ ]:
```
LR_visualizer = ROCAUC(ADA_Model_ad)

LR_visualizer.fit(X_train, y_train)        # Fit the training data to the visu
LR_visualizer.score(X_test, y_test)        # Evaluate the model on the test da
LR_visualizer.show()
```



Out[ ]:
```
<AxesSubplot:title={'center':'ROC Curves for AdaBoostClassifier'}, xlabel='Fa
lse Positive Rate', ylabel='True Positive Rate'>
```

In [ ]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred_xtest, cmap='Reds')
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bd70ea81
00>
```



In [ ]:
```
LR_mul = multilabel_confusion_matrix(y_test, y_pred_xtest)
LR_mul
```

Out[ ]:
```
array([[[6153,    3],
        [  89, 1096]],

       [[5109,  188],
        [ 278, 1766]],

       [[2900,  329],
        [ 153, 3959]]], dtype=int64)
```

In [ ]:
```
LR_FN = LR_mul[2][1][0]
update_score_card(ADA_Model_ad, LR_FN,'ADA Boosting Tuned')
```

In [ ]:
```
score_card
```

Out[ ]:

	Model	Precision Score	Recall Score	False Negatives	Kappa Score	f1-score
0	Logistic Regression	0.671991	0.633901	618	0.464028	0.64833
1	Decision Tree	0.902946	0.904135	290	0.843953	0.903478
2	Random Forest	0.940595	0.924845	211	0.882204	0.932166
3	Extra Tree	0.926692	0.908714	256	0.852936	0.917194
4	XGB	0.943575	0.931852	216	0.891763	0.937088
5	LGBM	0.943361	0.930054	208	0.889948	0.936146
6	ADA Boosting	0.937779	0.925013	238	0.879255	0.93079
7	Gradient Boosting	0.944284	0.930295	201	0.891061	0.936731
8	Random Forest Tuned	0.93786	0.91768	215	0.874909	0.926918
9	ADA Boosting Tuned	0.941444	0.917226	153	0.876708	0.928594

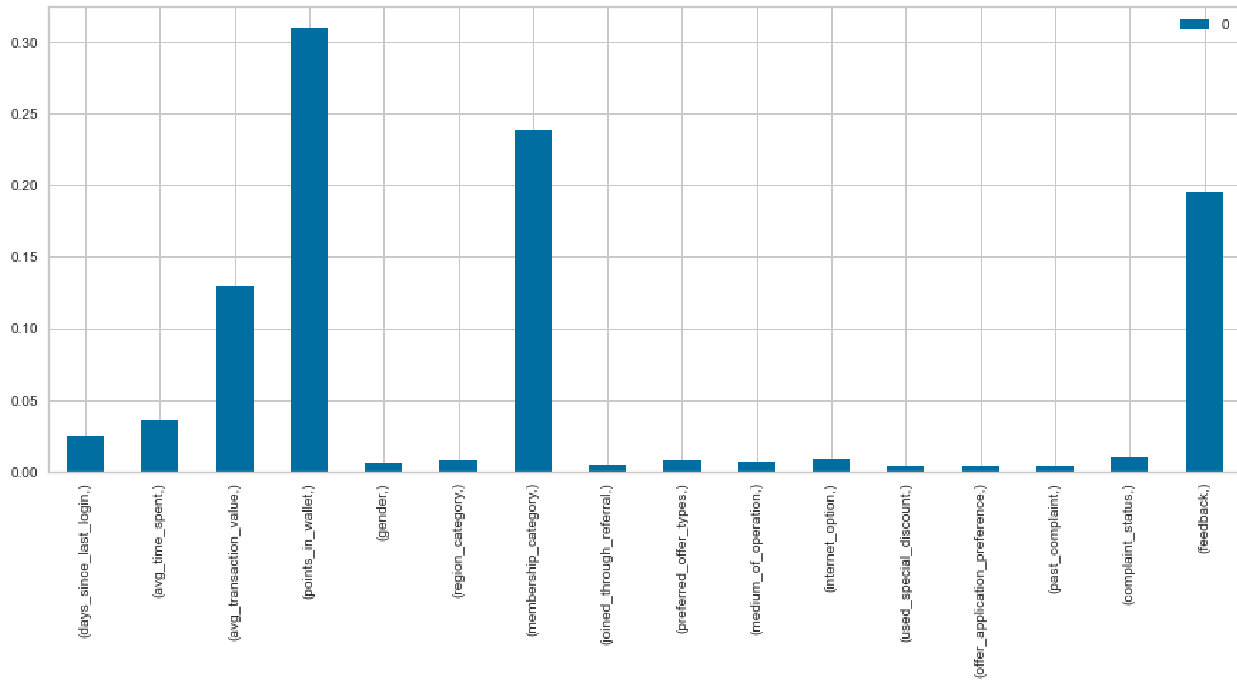
Summary: From the above models, we could see that the Gradient Boosting over other models is holding a very good Precision score, Recall Score, Kappa score, f1-score and very less number of False Negative values for the churn risk class of 2. Since, our business involves False Negative values to be costly for our class 2 churn segment, We feel it is important to lower the False Negative of class 2 segement. Since, Gradient Boosting is having ideal scores, it is practically impossible to have those scores, Hence we choose Random forest as our final model.

In [ ]:

```
pd.DataFrame(RF_Model.feature_importances_,index = [X.columns]).plot(kind='bar
```

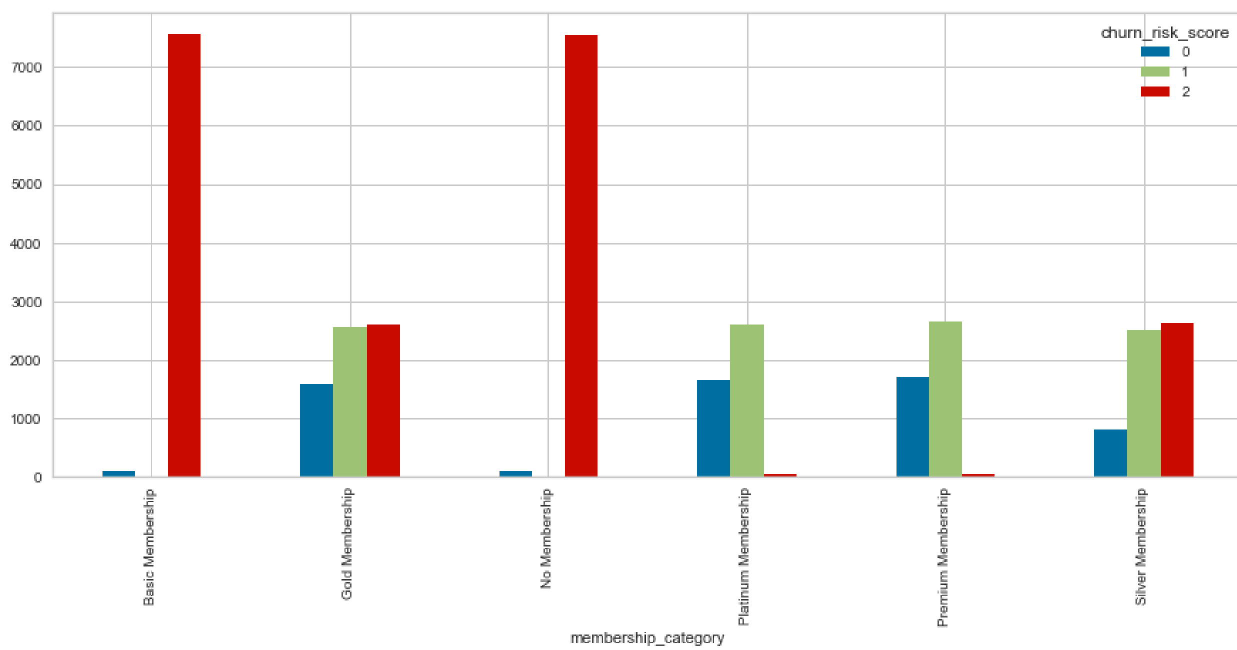
Out[ ]:

```
<AxesSubplot:>
```



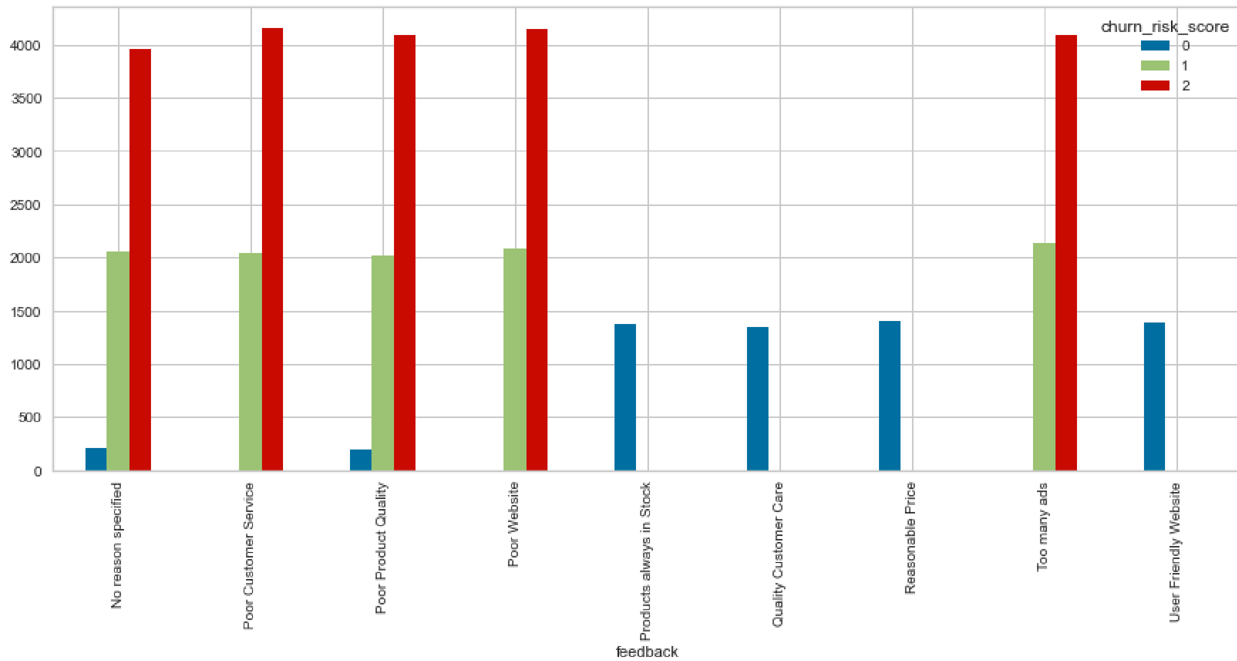
In [ ]:
```
pd.crosstab(df.membership_category,df.churn_risk_score).plot(kind='bar')
```

Out[ ]: <AxesSubplot:xlabel='membership_category'>



In [ ]:
```
pd.crosstab(df.feedback,df.churn_risk_score).plot(kind='bar')
```

Out[ ]: <AxesSubplot:xlabel='feedback'>



Business Recommendations

Based on EDA observations and model predictions, the following suggestions have been made:

- Most of the feedbacks are Not Specified,poor customer service,poor website and poor product quality which is affecting our model prediction. The Web development team can make the feedback windows as a compulsory one.
- Quality customer care should be provided. They should address all the concerns.
- Customer with Basic membership and No membership are tend to get churn risk rate 3
- We should try to upsell the memberships. Converting memberships to next level will provide the customers with extra benefits.
- We have to provide offers and bonus for our existing customers. This will improve our revenue in a long run.

In [ ]:

In [ ]:

In [ ]: