Business Case

OLA - Ensemble Learning

Suman Debnath

Introduction

- Recruiting and retaining drivers is seen by industry watchers as a tough battle for Ola.
- Churn among drivers is high and it's very easy for drivers to stop working for the service on the fly or jump to Uber depending on the rates.

- As the companies get bigger, the high churn could become a bigger problem. To find new drivers, Ola is casting a wide net, including people who don't have cars for jobs. But this acquisition is really costly.
- Losing drivers frequently impacts the morale of the organization and acquiring new drivers is more expensive than retaining existing ones.

- You are working as a data scientist with the Analytics Department of Ola, focused on driver team attrition.
- You are provided with the monthly information for a segment of drivers for 2019 and 2020 and tasked to predict whether a driver will be leaving the company or not based on their attributes like

- Demographics (city, age, gender etc.)

- Tenure information (joining date, Last Date)
- Historical data regarding the performance of the driver (Quarterly rating, Monthly business acquired, grade, Income)

Column Profiling:

- MMMM-YY : Reporting Date (Monthly)
- Driver_ID : Unique id for drivers
- Age : Age of the driver
- Gender : Gender of the driver – Male : 0, Female: 1
- City : City Code of the driver
- Education_Level : Education level – 0 for 10+ ,1 for 12+ ,2 for graduate
- Income : Monthly average Income of the driver
- Date Of Joining : Joining date for the driver
- LastWorkingDate : Last date of working for the driver
- Joining Designation : Designation of the driver at the time of joining
- Grade : Grade of the driver at the time of reporting
- Total Business Value : The total business value acquired by the driver in a month (negative business indicates -cancellation/refund or car EMI adjustments)
- Quarterly Rating : Quarterly rating of the driver: 1,2,3,4,5 (higher is better)

Summary

Data Distribution:

- **Gender**:
  - Male: 1380
  - Female: 956
- **Churn Distribution**:
  - 1 (Churned): 1616 (67.87%)
  - 0 (Not Churned): 765 (32.13%)

Random Forest:

- Train and test score: (0.8697, 0.8679)
- Highest feature importance: Joining year, followed by the number of records available in data, and total business value.
- Recall: 0.866
- Precision: 0.928
- F1-Score: 0.89

Grid Search CV on Random Forest:

- Best parameters: ccp_alpha=0.001, max_depth=10, max_features=7, n_estimators=300
- Best score: 0.8881

Bagging Classifier with Decision Trees:

- 50 Decision Trees, max_depth=7, class_weight="balanced"
- F1 Score: 0.9064
- Precision: 0.9388
- Recall Score: 0.8762
- Accuracy: 0.880

XGBoost Classifier (Grid Search CV):

- Parameters: 'max_depth': 2, 'n_estimators': 100
- Test Scores:
  - Accuracy: 0.87
  - F1 Score: 0.90
  - Recall: 0.923
  - Precision: 0.884
- Highest feature importance: Joining year, followed by the number of records available in data, and total business value.

Gradient Boosting Classifier (GBC):

- Train Score: 0.9144
- Test Score: 0.8910
- Accuracy Score: 0.8910
- ROC-AUC Score (test dataset): 0.9448
- Precision Score (test dataset): 0.9288
- Recall Score (test dataset): 0.9119
- F1 Score (test dataset): 0.9202

Observations

- The probability of churn is higher in cases where the education level is 0 and 1, compared to 2.
- For drivers with a joining designation of 1, the probability of churn is higher.
- When the quarterly rating is 1, the probability of churn is significantly higher.
- A similar pattern is observed for drivers whose quarterly rating has increased throughout their tenure.
- Drivers who joined in 2018 and 2019 have a very high probability of churn compared to those who joined in 2020 or before 2018.

Detailed Analysis

Importing all the `libs`

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import figure

import statsmodels.api as sm
from scipy.stats import norm
from scipy.stats import t

import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

In [2]:
```
ola = pd.read_csv("ola.csv")
```

In [3]:
```
ola.head(5)
```

Out[3]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18
2	2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18
3	3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20
4	4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20

In [4]:
```
df = ola.copy()
```

EDA

In [5]:
```
(df.isna().sum()/len(df))*100
```

Out[5]:
```
Unnamed: 0              0.000000
MMM-YY                  0.000000
Driver_ID               0.000000
Age                     0.319305
Gender                  0.272194
City                    0.000000
Education_Level         0.000000
Income                  0.000000
Dateofjoining           0.000000
LastWorkingDate        91.541039
Joining Designation     0.000000
Grade                   0.000000
Total Business Value    0.000000
Quarterly Rating        0.000000
dtype: float64
```

In [6]:
```
df.head(10)
```

Out[6]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18
2	2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18
3	3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20
4	4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20
5	5	12/01/19	4	43.0	0.0	C13	2	65603	12/07/19
6	6	01/01/20	4	43.0	0.0	C13	2	65603	12/07/19
7	7	02/01/20	4	43.0	0.0	C13	2	65603	12/07/19
8	8	03/01/20	4	43.0	0.0	C13	2	65603	12/07/19
9	9	04/01/20	4	43.0	0.0	C13	2	65603	12/07/19

In [7]:
```
df.shape
```

Out[7]: (19104, 14)

In [8]:
```
# No. of unique drivers
df["Driver_ID"].nunique()
```

Out[8]: 2381

In [9]:
```
df.drop(["Unnamed: 0"],axis = 1 , inplace=True)
```

In [10]:
```
df["Gender"].replace({0.0:"Male",1.0:"Female"},inplace=True)
```

In [11]:
```
df.sample(5)
```

Out[11]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWc
8017	09/01/19	1191	36.0	Male	C7	2	118722	23/06/17	
12407	09/01/20	1850	34.0	Female	C20	2	80779	05/05/20	
6018	01/01/19	896	32.0	Female	C18	2	22680	21/08/18	
17009	03/01/19	2508	37.0	Male	C11	2	64254	18/05/18	
16659	11/01/19	2470	31.0	Male	C27	1	55723	27/02/18	

In [12]:
```
df.groupby('Driver_ID').count()
```

Out[12]:

	MMM-YY	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate
Driver_ID								
1	3	3	3	3	3	3	3	1
2	2	2	2	2	2	2	2	0
4	5	5	5	5	5	5	5	1
5	3	3	3	3	3	3	3	1
6	5	5	5	5	5	5	5	0
...	...	...	...	...	...	...	...	...
2784	24	24	24	24	24	24	24	0
2785	3	3	3	3	3	3	3	1
2786	9	9	9	9	9	9	9	1
2787	6	6	6	6	6	6	6	1
2788	7	7	7	7	7	7	7	0

2381 rows × 12 columns

In [13]:
```
df[df["Driver_ID"]==2784]
```

Out[13]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWo
19055	01/01/19	2784	33.0	Male	C24	0	82815	15/10/15	
19056	02/01/19	2784	33.0	Male	C24	0	82815	15/10/15	
19057	03/01/19	2784	33.0	Male	C24	0	82815	15/10/15	
19058	04/01/19	2784	33.0	Male	C24	0	82815	15/10/15	
19059	05/01/19	2784	33.0	Male	C24	0	82815	15/10/15	
19060	06/01/19	2784	33.0	Male	C24	0	82815	15/10/15	
19061	07/01/19	2784	33.0	Male	C24	0	82815	15/10/15	
19062	08/01/19	2784	33.0	Male	C24	0	82815	15/10/15	
19063	09/01/19	2784	33.0	Male	C24	0	82815	15/10/15	
19064	10/01/19	2784	33.0	Male	C24	0	82815	15/10/15	
19065	11/01/19	2784	33.0	Male	C24	0	82815	15/10/15	
19066	12/01/19	2784	33.0	Male	C24	0	82815	15/10/15	
19067	01/01/20	2784	34.0	Male	C24	0	82815	15/10/15	
19068	02/01/20	2784	34.0	Male	C24	0	82815	15/10/15	
19069	03/01/20	2784	34.0	Male	C24	0	82815	15/10/15	
19070	04/01/20	2784	34.0	Male	C24	0	82815	15/10/15	
19071	05/01/20	2784	34.0	Male	C24	0	82815	15/10/15	
19072	06/01/20	2784	34.0	Male	C24	0	82815	15/10/15	
19073	07/01/20	2784	34.0	Male	C24	0	82815	15/10/15	
19074	08/01/20	2784	34.0	Male	C24	0	82815	15/10/15	
19075	09/01/20	2784	34.0	Male	C24	0	82815	15/10/15	
19076	10/01/20	2784	34.0	Male	C24	0	82815	15/10/15	
19077	11/01/20	2784	34.0	Male	C24	0	82815	15/10/15	
19078	12/01/20	2784	34.0	Male	C24	0	82815	15/10/15	

Restructuring the data by aggregation :

In [14]:

```
agg_df = df.groupby("Driver_ID").aggregate({
                                                'MMM-YY': len,
                                                "Age": max,
                                                "City": 'first',
                                                "Education_Level": max,
                                                "Income": np.mean,
                                                "Dateofjoining":'first',
                                                "Joining Designation":np.mean,
                                                "Grade": np.mean,
                                                "Total Business Value":sum,
```

```
                                         "Quarterly Rating":np.mean
                                     })
```

In [15]:
```
agg_df = agg_df.reset_index()
```

In [16]:
```
final_data = agg_df.rename(columns={
                                   "MMM-YY":"No_of_Records",
                                   "Dateofjoining":"Date_of_joining",
                                   "Joining Designation":"Joining_Designati
                                   "Total Business Value" : "Total_Business_
                                   "Quarterly Rating":"Quarterly_Rating"
                               }
                          )
```

In [17]:
```
final_data
```

Out[17]:

	Driver_ID	No_of_Records	Age	City	Education_Level	Income	Date_of_joining	Joining
0	1	3	28.0	C23	2	57387.0	24/12/18	
1	2	2	31.0	C7	2	67016.0	11/06/20	
2	4	5	43.0	C13	2	65603.0	12/07/19	
3	5	3	29.0	C9	0	46368.0	01/09/19	
4	6	5	31.0	C11	1	78728.0	31/07/20	
...	...	...	...	...	...	...	...	
2376	2784	24	34.0	C24	0	82815.0	15/10/15	
2377	2785	3	34.0	C9	0	12105.0	28/08/20	
2378	2786	9	45.0	C19	0	35370.0	31/07/18	
2379	2787	6	28.0	C20	2	69498.0	21/07/18	
2380	2788	7	30.0	C27	2	70254.0	06/08/20	

2381 rows × 11 columns

In [18]:
```
final_data = pd.merge(left = df.groupby(["Driver_ID"])["LastWorkingDate"].uniq
                      right = final_data,
                      on = "Driver_ID",
                      how="outer"
                     )
```

In [19]:
```
final_data = pd.merge(left = df.groupby(["Driver_ID"])["Gender"].unique().apply
                      right = final_data,
                      on = "Driver_ID",
                      how="outer"
                     )
```

In [20]:
```
data = final_data.copy()
```

In [21]:
```
data["Gender"].value_counts()
```

Out[21]:
```
Gender
Male      1380
Female     956
Name: count, dtype: int64
```

`Target variable creation` : target which tells whether the driver has left the company- driver whose last working day is present will have the value 1

In [22]:
```
pd.Series(np.where(data["LastWorkingDate"].isna(),0,1)).value_counts()
```
Out[22]:
```
1    1616
0     765
Name: count, dtype: int64
```

In [23]:
```
data["Churn"] = data["LastWorkingDate"].fillna(0)
```

In [24]:
```
def apply_0_1(y):
    if y == 0:
        return 0
    if y != 0:
        return 1
```

In [25]:
```
data["Churn"] = data["Churn"].apply(apply_0_1)
```

In [26]:
```
data["Churn"].value_counts()
```
Out[26]:
```
Churn
1    1616
0     765
Name: count, dtype: int64
```

In [27]:
```
data["Churn"].value_counts(normalize=True)*100
```
Out[27]:
```
Churn
1    67.870643
0    32.129357
Name: proportion, dtype: float64
```

- ### class 1 is the driviers who churned . 68%
- ### class 0 is the driviers who have not churned . 32%
- ### Data is imbalanced

Converting date columns into Datatime format :

In [28]:
```
data.head()
```

Out[28]:

	Driver_ID	Gender	LastWorkingDate	No_of_Records	Age	City	Education_Level	Income
0	1	Male	03/11/19	3	28.0	C23	2	57387.0
1	2	Male	NaN	2	31.0	C7	2	67016.0
2	4	Male	27/04/20	5	43.0	C13	2	65603.0
3	5	Male	03/07/19	3	29.0	C9	0	46368.0
4	6	Female	NaN	5	31.0	C11	1	78728.0

In [29]:
```
data["Date_of_joining"] = pd.to_datetime(data["Date_of_joining"])
data["LastWorkingDate"] = pd.to_datetime(data["LastWorkingDate"])
```

In [30]:
```
data["joining_Year"] = data["Date_of_joining"].dt.year
```

In [31]:
```
#  data["joining_month"] = data["Date_of_joining"].dt.month
```

checking for missing values after restructuring :

In [32]:
```
(data.isna().sum()/len(data))*100
```

Out[32]:
```
Driver_ID                0.000000
Gender                   1.889962
LastWorkingDate         32.129357
No_of_Records            0.000000
Age                      0.000000
City                     0.000000
Education_Level          0.000000
Income                   0.000000
Date_of_joining          0.000000
Joining_Designation      0.000000
Grade                    0.000000
Total_Business_Value     0.000000
Quarterly_Rating         0.000000
Churn                    0.000000
joining_Year             0.000000
dtype: float64
```

In [33]:
```
data["Churn"].value_counts(normalize=True)*100
```

Out[33]:
```
Churn
1    67.870643
0    32.129357
Name: proportion, dtype: float64
```

Feature Engineering :

whether the quarterly rating has increased for that driver

- #### for those whose quarterly rating has increased we assign the value 1

In [34]:
```
def app_rating_inc(y):
```

```
    if len(y)>=2:
        for i in range(len(y)):
            if y[-1]>y[-2]:
                return 1
            else:
                return 0
    else:
        return 0
```

In [35]:
```
Quarterly_Rating_increased = df.groupby("Driver_ID")["Quarterly Rating"].unique
```

In [36]:
```
data = pd.merge(left = Quarterly_Rating_increased,
        right = data,
         on = "Driver_ID",
         how="outer"
    )
```

In [37]:
```
# df.groupby("Driver_ID")["Quarterly Rating"].unique().apply(app_rating_inc)
```

In [38]:
```
data["Quarterly_Rating_increased"] = data["Quarterly Rating"]
```

In [39]:
```
data.drop(["Quarterly Rating"],axis=1,inplace=True)
```

whether the monthly income has increased for that driver -

- #### for those whose monthly income has increased we assign the value 1

In [40]:
```
def app_income_inc(y):

    if len(y)>=2:
        for i in range(len(y)):
            if y[-1]>y[-2]:
                return 1
            else:
                return 0
    else:
        return 0
```

In [41]:
```
# df.groupby("Driver_ID")["Income"].unique().apply(app_income_inc).rename("Inc
```

In [42]:
```
data = pd.merge(left = df.groupby("Driver_ID")["Income"].unique().apply(app_in
        right = data,
         on = "Driver_ID",
         how="outer"
    )
```

In [43]:
```
data
```

Out[43]:

	Driver_ID	Increased_Income	Gender	LastWorkingDate	No_of_Records	Age	City	Educ
0	1	0	Male	2019-03-11	3	28.0	C23	
1	2	0	Male	NaT	2	31.0	C7	
2	4	0	Male	2020-04-27	5	43.0	C13	
3	5	0	Male	2019-03-07	3	29.0	C9	
4	6	0	Female	NaT	5	31.0	C11	
...	...	...	...	...	...	...	...	
2376	2784	0	Male	NaT	24	34.0	C24	
2377	2785	0	Female	2020-10-28	3	34.0	C9	
2378	2786	0	Male	2019-09-22	9	45.0	C19	
2379	2787	0	Female	2019-06-20	6	28.0	C20	
2380	2788	0	Male	NaT	7	30.0	C27	

2381 rows × 17 columns

In [44]:
```
Mdata = data.copy()
```

In [45]:
```
Mdata["Gender"].replace({"Male":0,
                         "Female":1},inplace=True)
```

In [46]:
```
Mdata.drop(["Driver_ID"],axis = 1, inplace=True)
```

In [47]:
```
Mdata.isna().sum()
```

Out[47]:
```
Increased_Income              0
Gender                       45
LastWorkingDate             765
No_of_Records                 0
Age                           0
City                          0
Education_Level               0
Income                        0
Date_of_joining               0
Joining_Designation           0
Grade                         0
Total_Business_Value          0
Quarterly_Rating              0
Churn                         0
joining_Year                  0
Quarterly_Rating_increased    0
dtype: int64
```

In [48]:
```
Mdata
```

Out[48]:

	Increased_Income	Gender	LastWorkingDate	No_of_Records	Age	City	Education_Leve
0	0	0.0	2019-03-11	3	28.0	C23	2
1	0	0.0	NaT	2	31.0	C7	2
2	0	0.0	2020-04-27	5	43.0	C13	2
3	0	0.0	2019-03-07	3	29.0	C9	(
4	0	1.0	NaT	5	31.0	C11	'
...	...	...	...	...	...	...	..
2376	0	0.0	NaT	24	34.0	C24	(
2377	0	1.0	2020-10-28	3	34.0	C9	(
2378	0	0.0	2019-09-22	9	45.0	C19	(
2379	0	1.0	2019-06-20	6	28.0	C20	2
2380	0	0.0	NaT	7	30.0	C27	2

2381 rows × 16 columns

In [49]:
```
pd.to_datetime("2021-06-01")
```

Out[49]:
```
Timestamp('2021-06-01 00:00:00')
```

In [50]:
```
Mdata["LastWorkingDate"] = Mdata["LastWorkingDate"].fillna(pd.to_datetime("2021
```

In [51]:
```
(Mdata["LastWorkingDate"] - Mdata["Date_of_joining"])
```

Out[51]:
```
0         77 days
1        207 days
2        142 days
3         57 days
4        305 days
           ...
2376    2056 days
2377      61 days
2378     418 days
2379     334 days
2380     358 days
Length: 2381, dtype: timedelta64[ns]
```

In [52]:
```
Mdata["Driver_tenure_days"] = (Mdata["LastWorkingDate"] - Mdata["Date_of_joini
```

In [53]:
```
Mdata["Driver_tenure_days"] = Mdata["Driver_tenure_days"].dt.days
```

In [54]:
```
Mdata.drop(["LastWorkingDate","Date_of_joining"],inplace=True,axis = 1)
```

In [55]:
```
Mdata.drop(["Driver_tenure_days"],inplace=True,axis = 1)
```

In [56]:
```
Mdata
```

Out[56]:

	Increased_Income	Gender	No_of_Records	Age	City	Education_Level	Income	Joining
0	0	0.0	3	28.0	C23	2	57387.0	
1	0	0.0	2	31.0	C7	2	67016.0	
2	0	0.0	5	43.0	C13	2	65603.0	
3	0	0.0	3	29.0	C9	0	46368.0	
4	0	1.0	5	31.0	C11	1	78728.0	
...	...	...	...	...	...	...	...	
2376	0	0.0	24	34.0	C24	0	82815.0	
2377	0	1.0	3	34.0	C9	0	12105.0	
2378	0	0.0	9	45.0	C19	0	35370.0	
2379	0	1.0	6	28.0	C20	2	69498.0	
2380	0	0.0	7	30.0	C27	2	70254.0	

2381 rows × 14 columns

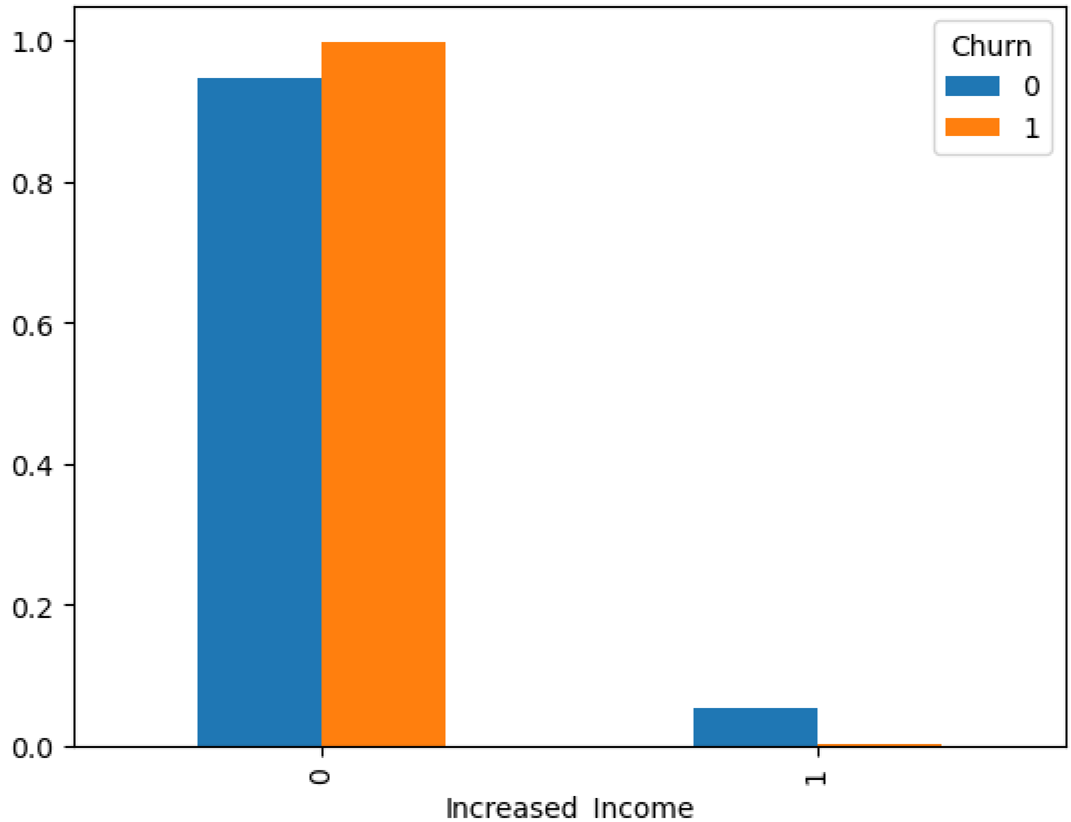
In [57]:
```
Mdata.columns
```

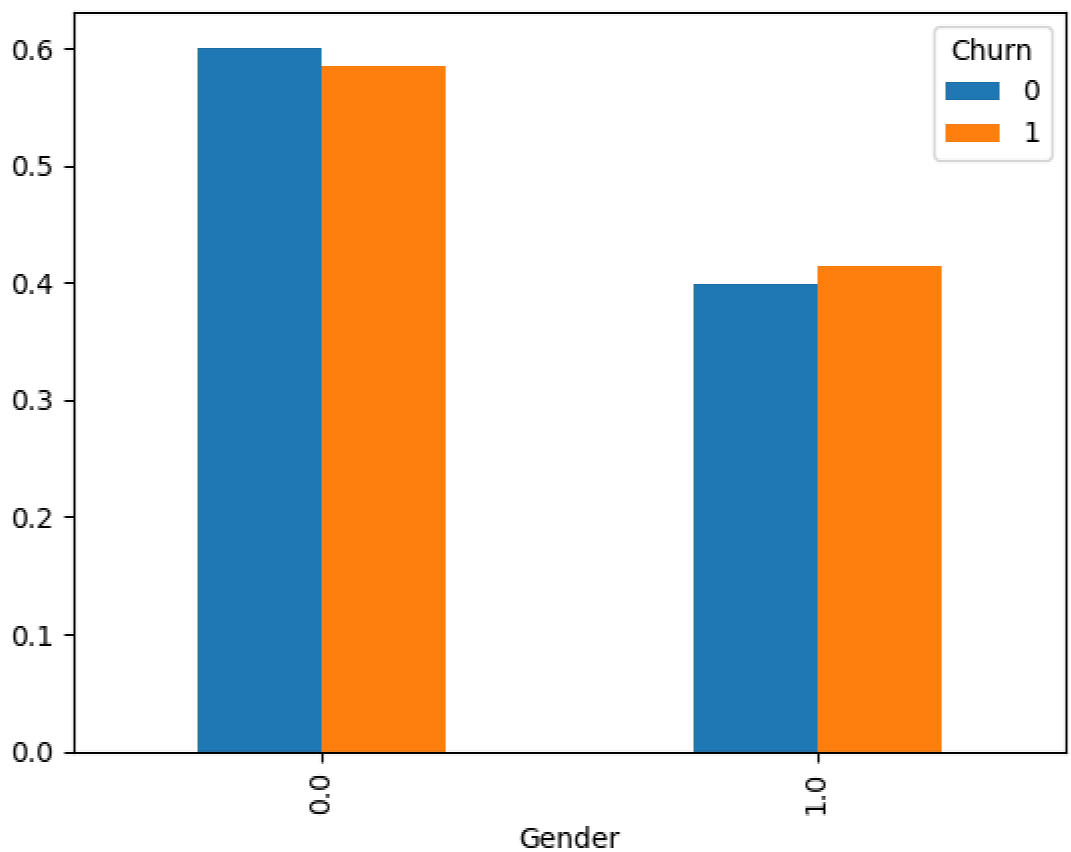
Out[57]:
```
Index(['Increased_Income', 'Gender', 'No_of_Records', 'Age', 'City',
       'Education_Level', 'Income', 'Joining_Designation', 'Grade',
       'Total_Business_Value', 'Quarterly_Rating', 'Churn', 'joining_Year',
       'Quarterly_Rating_increased'],
      dtype='object')
```

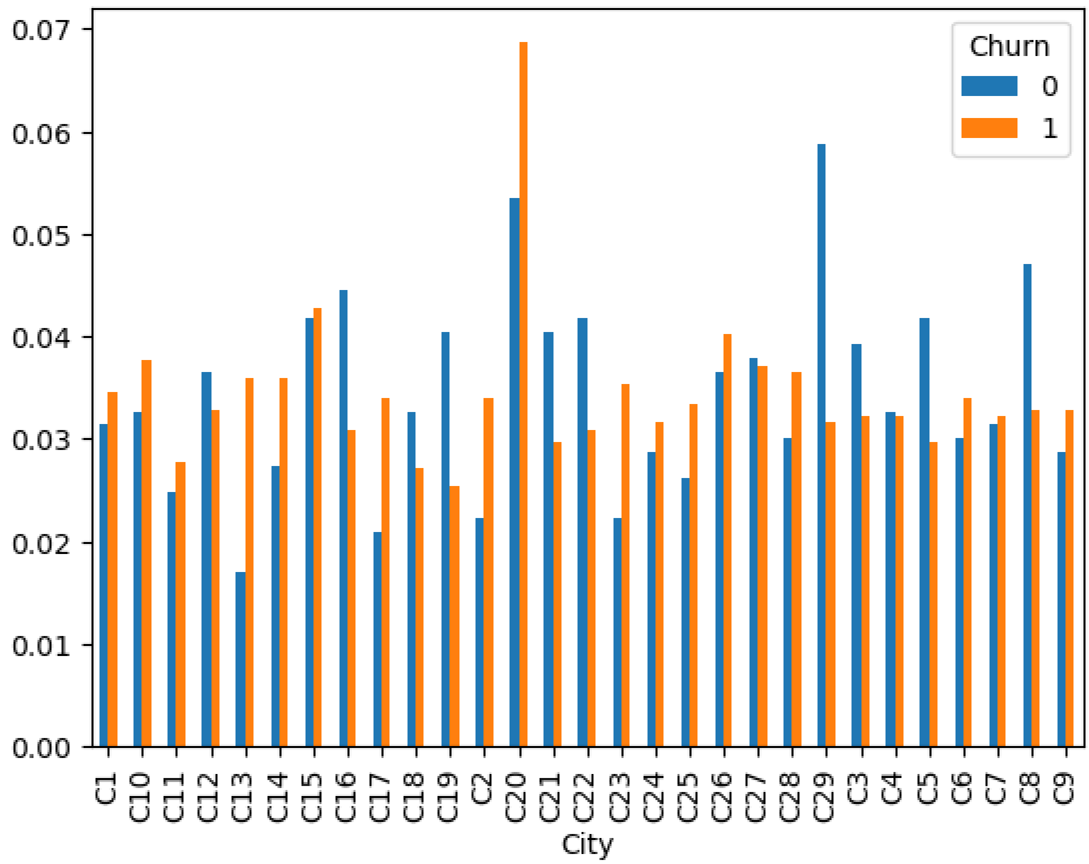
In [58]:
```
Mdata["Grade"] = np.round(Mdata["Grade"])
```

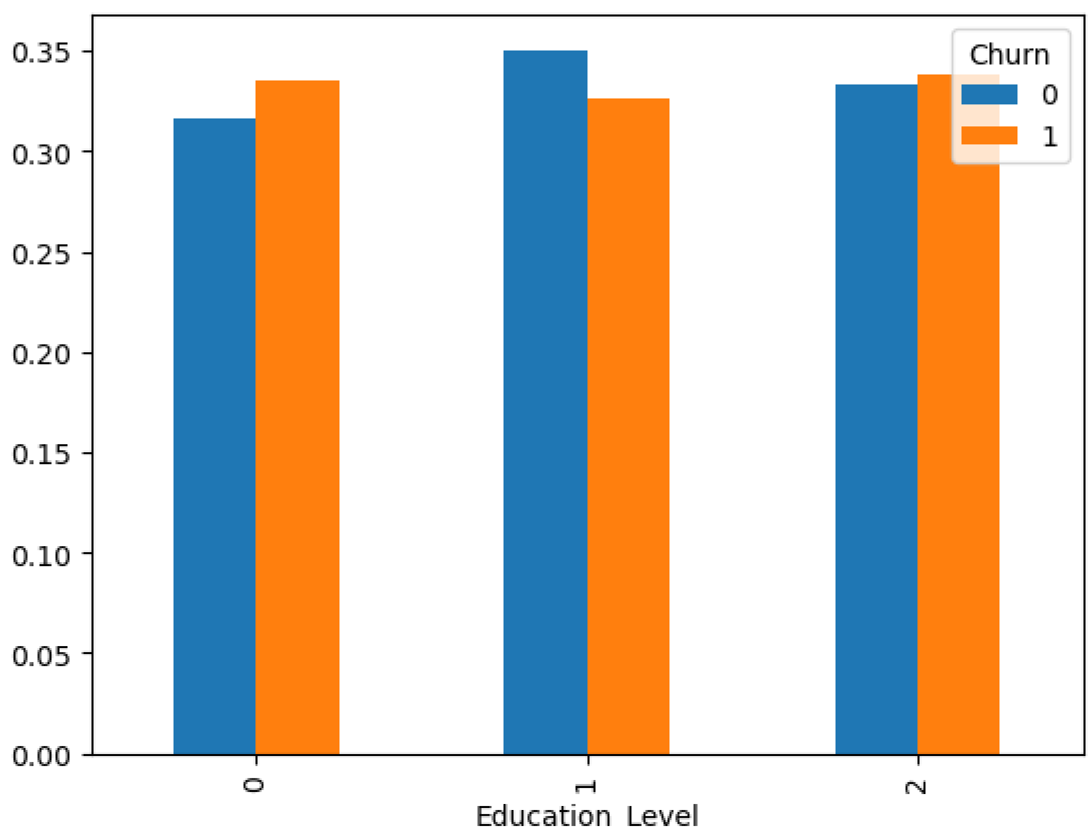
In [59]:
```
Mdata["Quarterly_Rating"]= Mdata["Quarterly_Rating"].round()
```

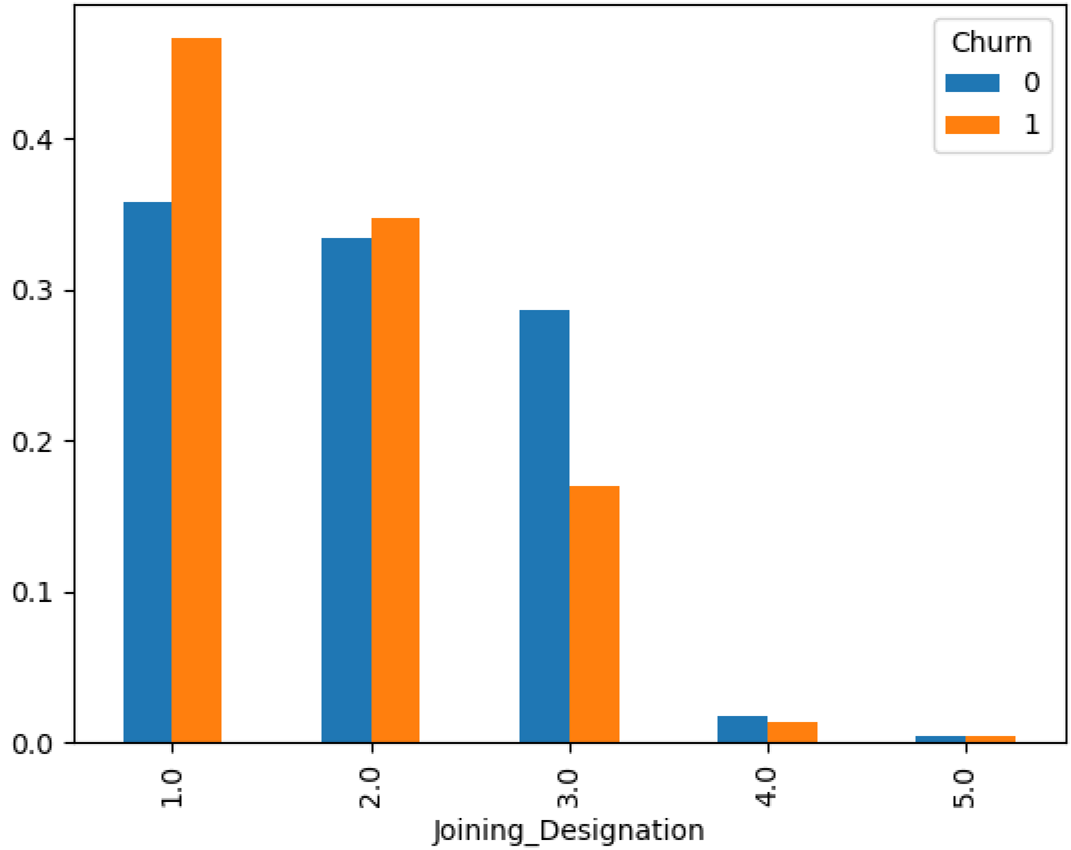
In [60]:
```
categorical_features = ['Increased_Income', 'Gender','City','Education_Level',
                        'Joining_Designation','Grade','Quarterly_Rating','Quarterly_

for col in categorical_features:
    pd.crosstab(index = Mdata[col],
                columns = Mdata["Churn"],
                normalize="columns").plot(kind = "bar")
    plt.show()
```

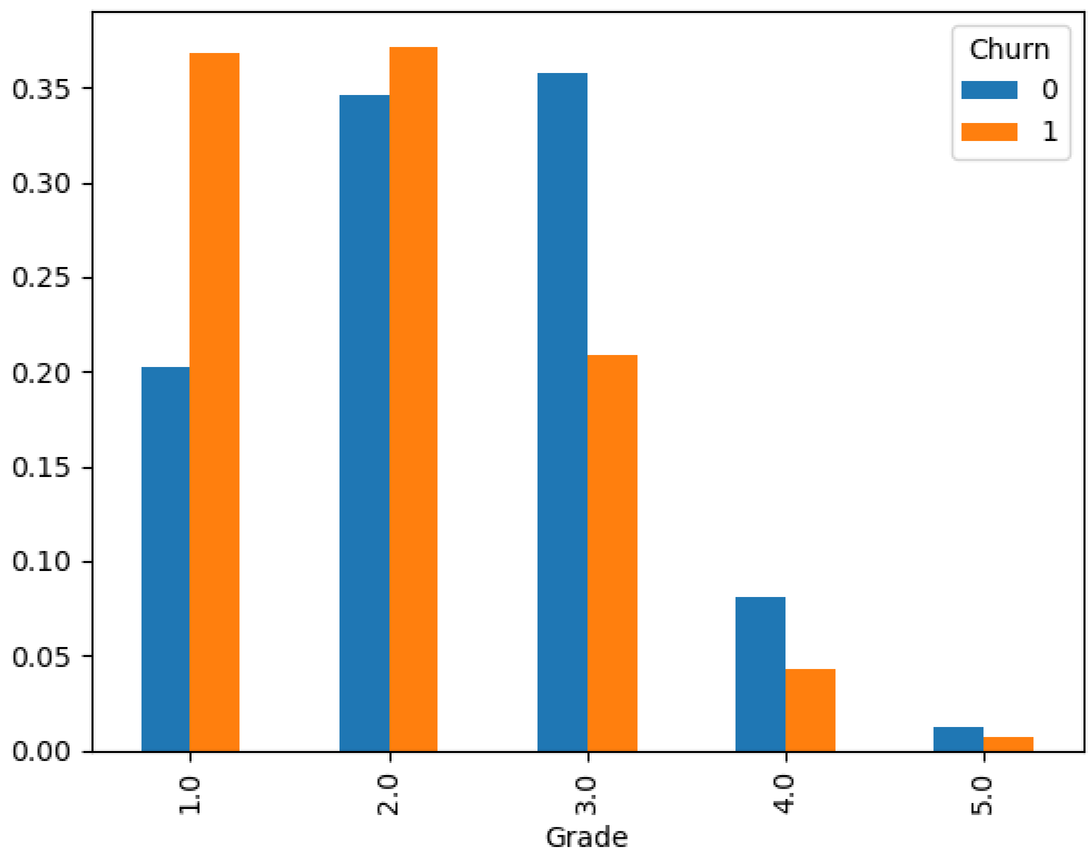

Churn
0
1
1.0
0.8
0.6
0.4
0.2
0.0
0
1
Increased_Income

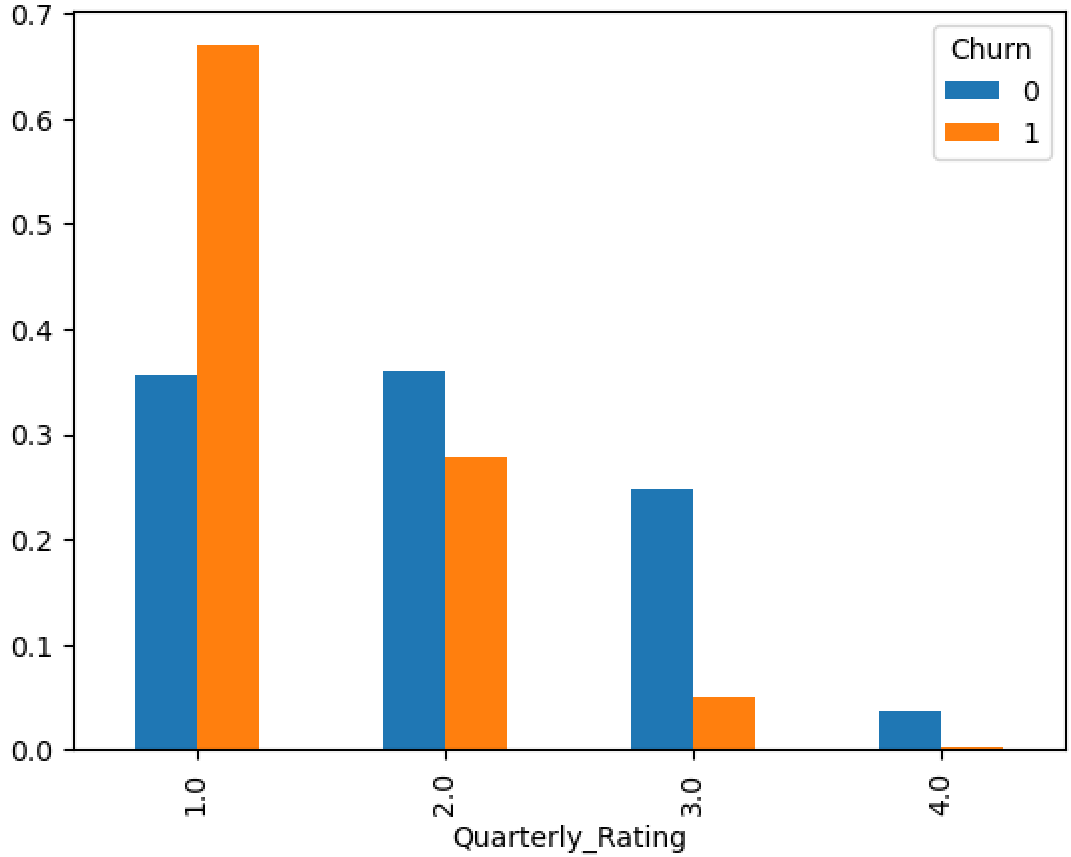

Churn
0
1
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
1.0
Gender

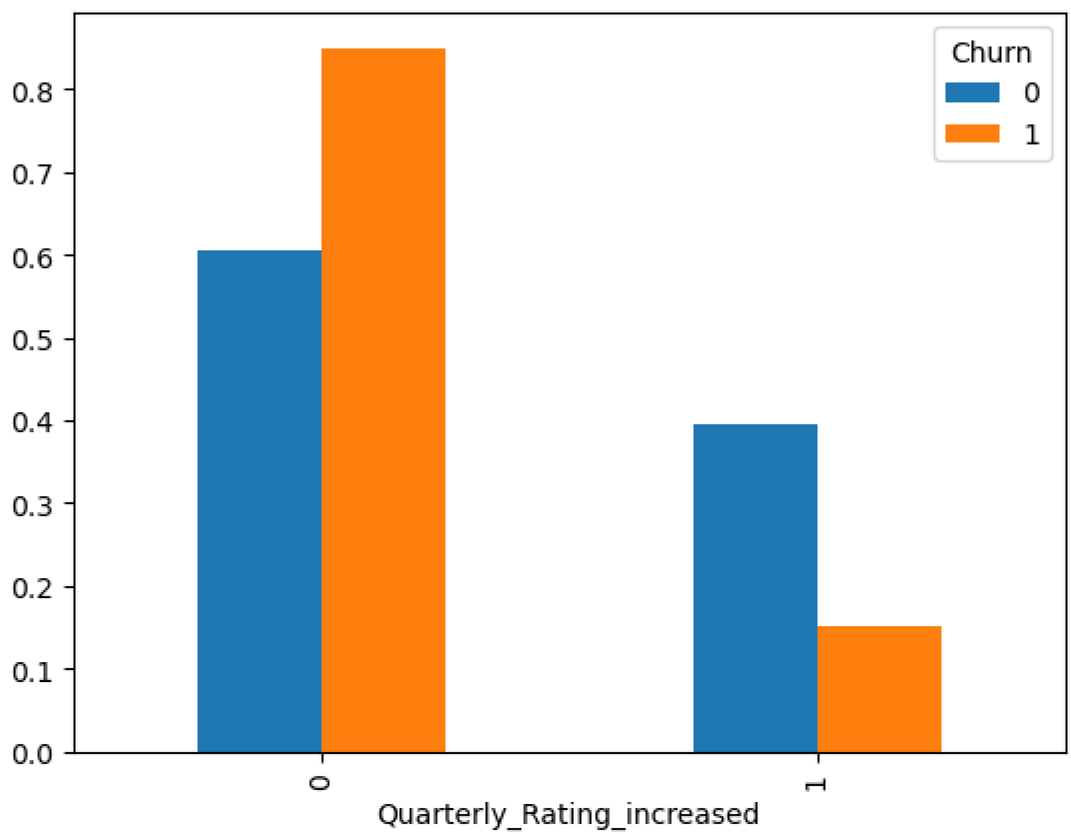

Churn
0
1
City
C1 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C2 C20 C21 C22 C23 C24 C25 C26 C27 C28 C29 C3 C4 C5 C6 C7 C8 C9
0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07

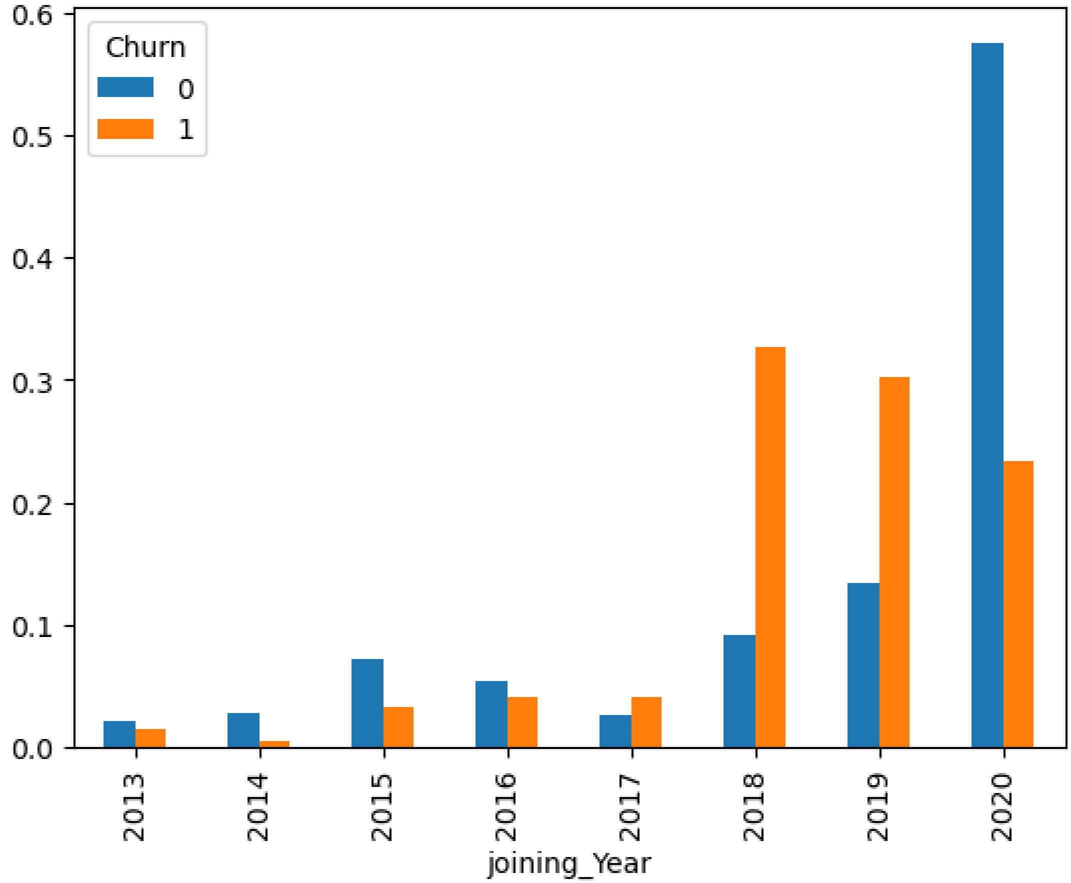

Churn
0
1
Education_Level
0 1 2
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35


Churn
0
1
0.0
0.1
0.2
0.3
0.4
1.0
2.0
3.0
4.0
5.0
Joining_Designation


Churn
0
1
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
1.0
2.0
3.0
4.0
5.0
Grade


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Churn
0
1
1.0
2.0
3.0
4.0
Quarterly_Rating


0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Churn
0
1
0
1
Quarterly_Rating_increased



In [61]:
```
Mdata.isna().sum()
```

Out[61]:
```
Increased_Income              0
Gender                       45
No_of_Records                 0
Age                           0
City                          0
Education_Level               0
Income                        0
Joining_Designation           0
Grade                         0
Total_Business_Value          0
Quarterly_Rating              0
Churn                         0
joining_Year                  0
Quarterly_Rating_increased    0
dtype: int64
```

SimpleImputer

In [62]:
```
from sklearn.impute import SimpleImputer
```

In [63]:
```
imputer = SimpleImputer(strategy='most_frequent')
```

In [64]:
```
Mdata["Gender"] = imputer.fit_transform(X=Mdata["Gender"].values.reshape(-1,1)
```

In [65]:
```
Mdata["Gender"].value_counts(dropna=False)
```

Out[65]:
```
Gender
0.0    1425
1.0     956
Name: count, dtype: int64
```

In [66]:
```
Mdata.isna().sum()
```

Out[66]:
```
Increased_Income              0
Gender                        0
No_of_Records                 0
Age                           0
City                          0
Education_Level               0
Income                        0
Joining_Designation           0
Grade                         0
Total_Business_Value          0
Quarterly_Rating              0
Churn                         0
joining_Year                  0
Quarterly_Rating_increased    0
dtype: int64
```

TargetEncoder

In [67]:
```
from category_encoders import TargetEncoder
TE = TargetEncoder()
```

In [68]:
```
Mdata["City"] = TE.fit_transform(X = Mdata["City"],y = Mdata["Churn"])
```

In [69]:
```
Mdata["joining_Year"] = TE.fit_transform(X = Mdata["joining_Year"],y = Mdata["(
```

```
Warning: No categorical columns found. Calling 'transform' will only return in
put data.
```

In [70]:
```
Mdata
```

Out[70]:

	Increased_Income	Gender	No_of_Records	Age	City	Education_Level	Income	J
0	0	0.0	3	28.0	0.769859	2	57387.0	
1	0	0.0	2	31.0	0.684190	2	67016.0	
2	0	0.0	5	43.0	0.816064	2	65603.0	
3	0	0.0	3	29.0	0.706553	0	46368.0	
4	0	1.0	5	31.0	0.702829	1	78728.0	
...	...	...	...	...	...	...	...	
2376	0	0.0	24	34.0	0.698531	0	82815.0	
2377	0	1.0	3	34.0	0.706553	0	12105.0	
2378	0	0.0	9	45.0	0.570044	0	35370.0	
2379	0	1.0	6	28.0	0.730263	2	69498.0	
2380	0	0.0	7	30.0	0.674162	2	70254.0	

2381 rows × 14 columns

In [71]:

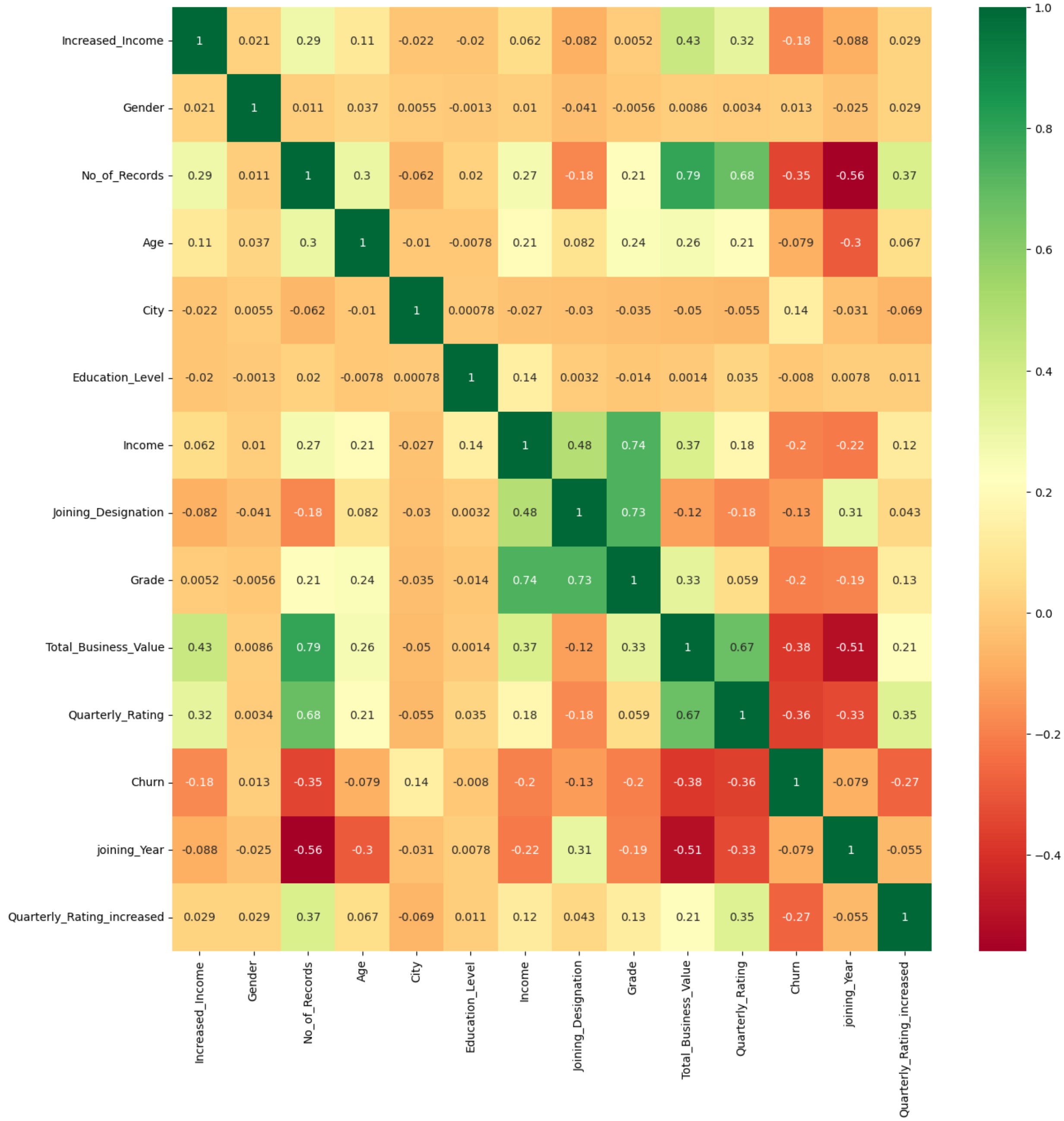
```
# Mdata.drop(["No_of_Records"], axis = 1 , inplace= True)
```

In [72]:

```
plt.figure(figsize=(15, 15))
sns.heatmap(Mdata.corr(),annot=True, cmap="RdYlGn", annot_kws={"size":10})
```

Out[72]:

```
<Axes: >
```



In [73]:
```
X = Mdata.drop(["Churn"],axis = 1)
y = Mdata["Churn"]
```

KNNImputer

In [74]:
```
import numpy as np
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=5)
```

In [75]:
```
X = pd.DataFrame(imputer.fit_transform(X),columns=X.columns)
```

In [76]:
```
X
```

Out[76]:

	Increased_Income	Gender	No_of_Records	Age	City	Education_Level	Income	Jo
0	0.0	0.0	3.0	28.0	0.769859	2.0	57387.0	
1	0.0	0.0	2.0	31.0	0.684190	2.0	67016.0	
2	0.0	0.0	5.0	43.0	0.816064	2.0	65603.0	
3	0.0	0.0	3.0	29.0	0.706553	0.0	46368.0	
4	0.0	1.0	5.0	31.0	0.702829	1.0	78728.0	
...	...	...	...	...	...	...	...	
2376	0.0	0.0	24.0	34.0	0.698531	0.0	82815.0	
2377	0.0	1.0	3.0	34.0	0.706553	0.0	12105.0	
2378	0.0	0.0	9.0	45.0	0.570044	0.0	35370.0	
2379	0.0	1.0	6.0	28.0	0.730263	2.0	69498.0	
2380	0.0	0.0	7.0	30.0	0.674162	2.0	70254.0	

2381 rows × 13 columns

In [77]:

```
X.describe()
```

Out[77]:

	Increased_Income	Gender	No_of_Records	Age	City	Education_L
count	2381.000000	2381.000000	2381.00000	2381.000000	2381.000000	2381.00
mean	0.018480	0.401512	8.02352	33.663167	0.678662	1.0C
std	0.134706	0.490307	6.78359	5.983375	0.065356	0.81
min	0.000000	0.000000	1.00000	21.000000	0.531324	0.00
25%	0.000000	0.000000	3.00000	29.000000	0.634237	0.00
50%	0.000000	0.000000	5.00000	33.000000	0.698531	1.00
75%	0.000000	1.000000	10.00000	37.000000	0.719430	2.00
max	1.000000	1.000000	24.00000	58.000000	0.816064	2.00

train_test_split

In [78]:

```
from sklearn.model_selection import train_test_split

X_train , X_test, y_train ,y_test = train_test_split(X,y,
                                                     random_state=5,
                                                     test_size=0.2)
```

In [79]:

```
y.value_counts()
```

Out[79]:

```
Churn
1    1616
0     765
Name: count, dtype: int64
```

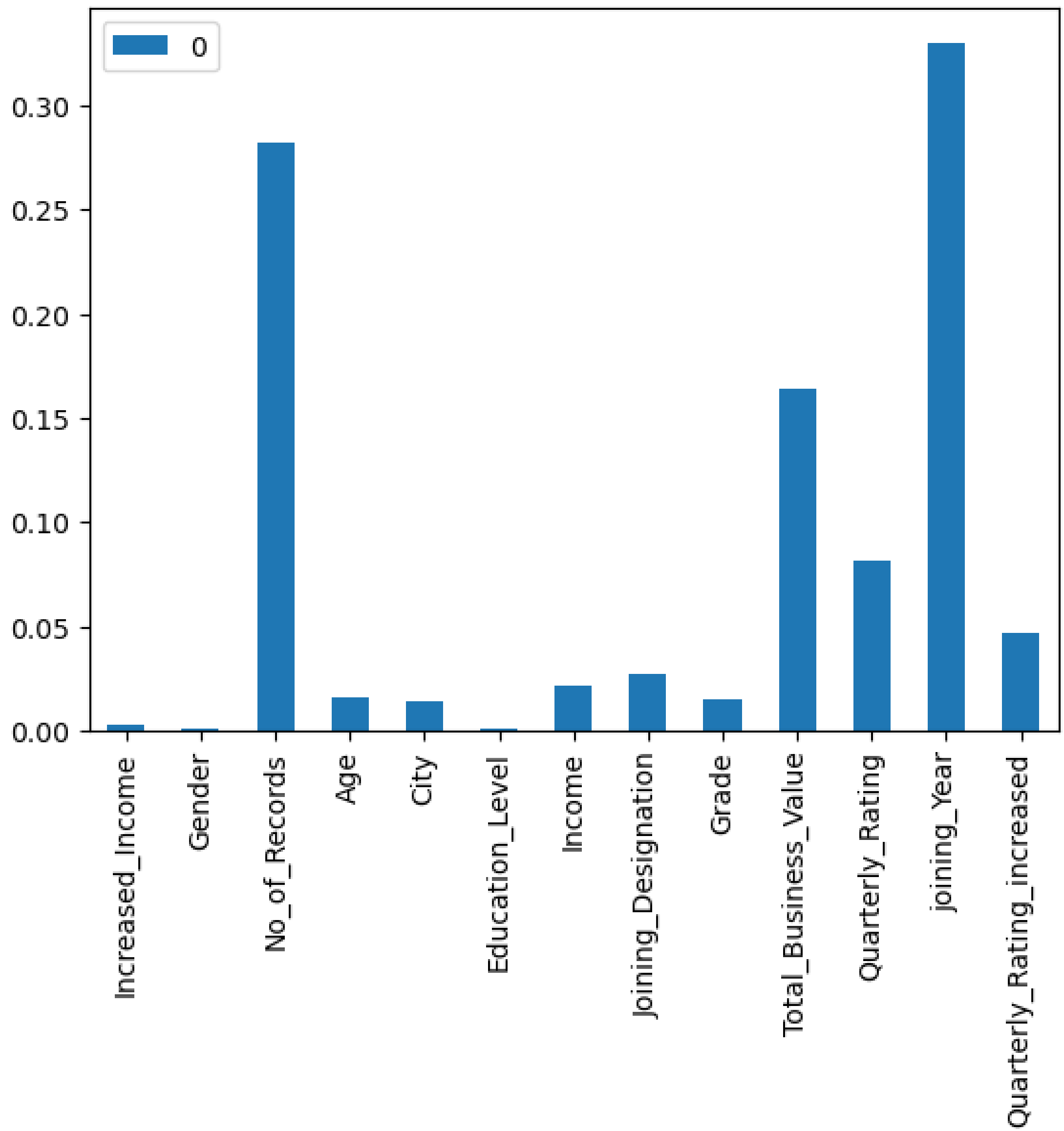
In [80]:
```
765 + 1616
```
Out[80]:
```
2381
```

StandardScaler

In [81]:
```
from sklearn.preprocessing import StandardScaler
```
In [82]:
```
scaler = StandardScaler()
```
In [83]:
```
scaler.fit(X_train)
```
Out[83]:
```
▾ StandardScaler
StandardScaler()
```
In [84]:
```
X_train = scaler.transform(X_train)
X_test =  scaler.transform(X_test)
```
In [ ]:

RandomForestClassifier

In [85]:
```
from sklearn.ensemble import RandomForestClassifier
```
In [86]:
```
RF = RandomForestClassifier(n_estimators=100,
    criterion='entropy',
    max_depth=10,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features='sqrt',
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    bootstrap=True,
    oob_score=False,
    n_jobs=None,
    random_state=None,
    verbose=0,
    warm_start=False,
    class_weight="balanced",
    ccp_alpha=0.0085,
    max_samples=None,)
```
In [87]:
```
RF.fit(X_train,y_train)
```
Out[87]:
```
▾ RandomForestClassifier
RandomForestClassifier(ccp_alpha=0.0085, class_weight='balanced',
                       criterion='entropy', max_depth=10)
```

In [88]:
```
RF.score(X_train,y_train),RF.score(X_test,y_test)
```
Out[88]:
```
(0.8571428571428571, 0.8616352201257862)
```
In [89]:
```
RF.feature_importances_
```
Out[89]:
```
array([0.00257681, 0.00046543, 0.28223717, 0.01553743, 0.01363866,
       0.00098849, 0.02104094, 0.02750293, 0.01454233, 0.16364764,
       0.08109249, 0.32994611, 0.04678358])
```
In [90]:
```
X.columns
```
Out[90]:
```
Index(['Increased_Income', 'Gender', 'No_of_Records', 'Age', 'City',
       'Education_Level', 'Income', 'Joining_Designation', 'Grade',
       'Total_Business_Value', 'Quarterly_Rating', 'joining_Year',
       'Quarterly_Rating_increased'],
      dtype='object')
```
In [91]:
```
pd.DataFrame(data=RF.feature_importances_,
             index=X.columns).plot(kind="bar")
```
Out[91]:
```
<Axes: >
```


In [92]:
```
from sklearn.metrics import f1_score , precision_score, recall_score,confusion_
```

In [93]:
```
confusion_matrix(y_test,RF.predict(X_test) )
```

Out[93]:
```
array([[151,  11],
       [ 55, 260]])
```

In [94]:
```
confusion_matrix(y_train,RF.predict(X_train) )
```

Out[94]:
```
array([[ 562,   41],
       [ 231, 1070]])
```

In [95]:
```
f1_score(y_test,RF.predict(X_test)),f1_score(y_train,RF.predict(X_train))
```

Out[95]:
```
(0.887372013651877, 0.8872305140961857)
```

In [96]:
```
precision_score(y_test,RF.predict(X_test)),precision_score(y_train,RF.predict(X
```

Out[96]:
```
(0.959409594095941, 0.963096309630963)
```

In [97]:
```
recall_score(y_test,RF.predict(X_test)),recall_score(y_train,RF.predict(X_trai
```

Out[97]:
```
(0.8253968253968254, 0.8224442736356649)
```

GridSearchCV - on RandomForestClassifier

In [98]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

parameters = {"max_depth":[7,10,15],
             "n_estimators":[100,200,300,400],
             "max_features":[4,7,10],
             "ccp_alpha":[0.0005,0.00075,0.001]}

RFC = RandomForestClassifier()
grid_search = GridSearchCV(
    estimator = RFC,
    param_grid = parameters,
    scoring = "accuracy",
    n_jobs = -1,
    refit=True,                   # need not to train again after grid search
    cv=3,
    pre_dispatch='2*n_jobs',
    return_train_score=False)
```

In [99]:
```
grid_search.fit(X_train,y_train.values.ravel())
```

Out[99]:
```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [100…
```
grid_search.best_estimator_
```

Out[100]:

```
▾ RandomForestClassifier
RandomForestClassifier(ccp_alpha=0.00075, max_depth=15, max_features=
7,
                       n_estimators=300)
```

In [101…

```
grid_search.best_score_
```

Out[101]:

```
0.8881409539895841
```

In [102…

```
grid_search.best_params_
```

Out[102]:

```
{'ccp_alpha': 0.00075, 'max_depth': 15, 'max_features': 7, 'n_estimators': 30
0}
```

In [ ]:

In [103…

```
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(n_estimators=100,
    criterion='entropy',
    max_depth=7,
    min_samples_split=2,
    min_samples_leaf=1,

    class_weight="balanced",
    ccp_alpha=0.0001,
    max_samples=None)
```

In [104…

```
RF.fit(X_train , y_train)
```

Out[104]:

```
▾ RandomForestClassifier
RandomForestClassifier(ccp_alpha=0.0001, class_weight='balanced',
                       criterion='entropy', max_depth=7)
```

In [105…

```
RF.score(X_train,y_train),RF.score(X_test,y_test)
```

Out[105]:

```
(0.8991596638655462, 0.8867924528301887)
```

In [106…

```
y_test_pred = RF.predict(X_test)
y_train_pred = RF.predict(X_train)
```

In [107…

```
f1_score(y_test,y_test_pred),f1_score(y_train,y_train_pred)
```

Out[107]:

```
(0.9114754098360656, 0.923076923076923)
```

In [108…

```
precision_score(y_test,y_test_pred),precision_score(y_train,y_train_pred)
```

Out[108]:

```
(0.9423728813559322, 0.9640167364016736)
```

In [109…

```
recall_score(y_test,y_test_pred),recall_score(y_train,y_train_pred)
```

Out[109]:

```
(0.8825396825396825, 0.8854727132974635)
```

BaggingClassifier

In [110…
```
from sklearn.tree import DecisionTreeClassifier
```

In [111…
```
from sklearn.ensemble import BaggingClassifier
```

In [112…
```
bagging_classifier_model = BaggingClassifier(base_estimator=  DecisionTreeClas

                                            n_estimators=50,
                                            max_samples=1.0,
                                            max_features=1.0,
                                            bootstrap=True,
                                            bootstrap_features=False,
                                            oob_score=False,
                                            warm_start=False,
                                            n_jobs=None,
                                            random_state=None,
                                            verbose=0,)
```

In [113…
```
bagging_classifier_model.fit(X_train,y_train)
```

Out[113]:
```
▸ BaggingClassifier
▸ base_estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [114…
```
from sklearn.metrics import f1_score , precision_score, recall_score,confusion_
```

In [115…
```
y_test_pred = bagging_classifier_model.predict(X_test)
y_train_pred = bagging_classifier_model.predict(X_train)
```

In [116…
```
confusion_matrix(y_test,y_test_pred)
```

Out[116]:
```
array([[142,  20],
       [ 45, 270]])
```

In [117…
```
confusion_matrix(y_train,y_train_pred)
```

Out[117]:
```
array([[ 558,   45],
       [ 128, 1173]])
```

In [118…
```
f1_score(y_test,y_test_pred),f1_score(y_train,y_train_pred)
```

Out[118]:
```
(0.8925619834710743, 0.9313219531560143)
```

In [119…
```
precision_score(y_test,y_test_pred),precision_score(y_train,y_train_pred)
```

Out[119]:
```
(0.9310344827586207, 0.9630541871921182)
```

In [120…
```
recall_score(y_test,y_test_pred),recall_score(y_train,y_train_pred)
```

Out[120]:
```
(0.8571428571428571, 0.9016141429669485)
```

In [121…
```
bagging_classifier_model.score(X_test,y_test)
```
Out[121]: 0.8637316561844863

In [122…
```
bagging_classifier_model.score(X_train,y_train)
```
Out[122]: 0.9091386554621849

In [123…
```
# !pip install xgboost
```

In [124…
```
from xgboost import XGBClassifier
```

In [125…
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

parameters = {"max_depth":[2,4,6,10],
             "n_estimators":[100,200,300,400]    }


grid_search = GridSearchCV(
    estimator = XGBClassifier(),
    param_grid = parameters,
    scoring = "accuracy",
    n_jobs = -1,
    refit=True,                   # need not to train again after grid search
    cv=3,
    pre_dispatch='2*n_jobs',
    return_train_score=False)


grid_search.fit(X_train,y_train.values.ravel())

grid_search.best_estimator_

grid_search.best_score_

grid_search.best_params_
```
Out[125]: {'max_depth': 2, 'n_estimators': 100}

In [126…
```
xgb = XGBClassifier(n_estimators=100,
                    max_depth = 2)
xgb.fit(X_train, y_train)
```

Out[126]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_roun
ds=None,
              enable_categorical=False, eval_metric=None, feature_typ
es=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_b
in=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=2, max_leaves=None,
```

In [127…

```
y_test_pred = xgb.predict(X_test)
y_train_pred = xgb.predict(X_train)
```

In [128…

```
confusion_matrix(y_test,y_test_pred)
```

Out[128]:

```
array([[124,  38],
       [ 27, 288]])
```

In [129…

```
confusion_matrix(y_train,y_train_pred)
```

Out[129]:

```
array([[ 518,   85],
       [  74, 1227]])
```

In [130…

```
xgb.score(X_train,y_train),xgb.score(X_test,y_test)
```

Out[130]:

```
(0.9164915966386554, 0.8637316561844863)
```

In [131…

```
f1_score(y_test,y_test_pred),f1_score(y_train,y_train_pred)
```

Out[131]:

```
(0.8985959438377534, 0.939150401836969)
```

In [132…

```
recall_score(y_test,y_test_pred),recall_score(y_train,y_train_pred)
```

Out[132]:

```
(0.9142857142857143, 0.9431206764027671)
```

In [133…

```
precision_score(y_test,y_test_pred),precision_score(y_train,y_train_pred)
```

Out[133]:

```
(0.8834355828220859, 0.9352134146341463)
```

In [134…

```
xgb.feature_importances_
```

Out[134]:

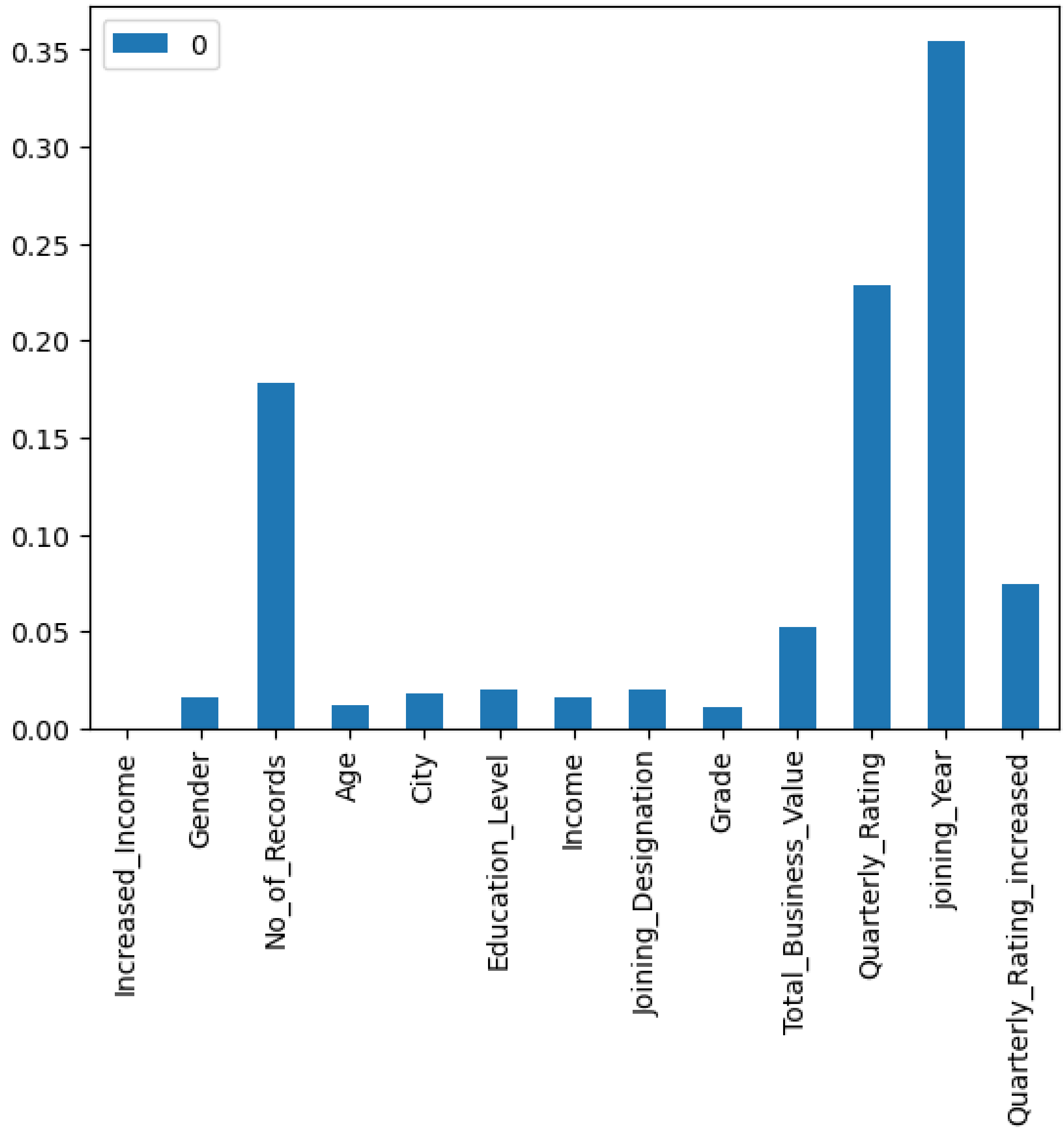
```
array([0.        , 0.01611706, 0.17770752, 0.01213652, 0.0177371 ,
       0.01956165, 0.01636491, 0.02013278, 0.01096871, 0.05184174,
       0.22887574, 0.35447577, 0.07408047], dtype=float32)
```

In [135…

```
pd.DataFrame(data=xgb.feature_importances_,
             index=X.columns).plot(kind="bar")
```

Out[135]:

```
<Axes: >
```



GradientBoostingClassifier

In [136…

```
def GradientBoostingClassifier(X, y):
    from sklearn.ensemble import  GradientBoostingClassifier
    from sklearn.metrics import f1_score, accuracy_score , roc_auc_score,auc,r
    X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        test_size=0.2,
                                                        random_state=1)

    lr = GradientBoostingClassifier()
    scaler = StandardScaler()
    scaler.fit(X_train)
    X_train = scaler.transform(X_train)
    X_test = scaler.transform(X_test)

    lr.fit(X_train, y_train)
    y_pred = lr.predict(X_test)
    prob = lr.predict_proba(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print('Train Score : ', lr.score(X_train, y_train), '\n')
```

```
    print('Test Score : ', lr.score(X_test, y_test), '\n')
    print('Accuracy Score : ', accuracy_score(y_test, y_pred), '\n')
    print(cm, "---> confusion Matrix ", '\n')
    print("ROC-AUC score  test dataset:  ", roc_auc_score(y_test, prob[:, 1]),
    print("precision score  test dataset:  ", precision_score(y_test, y_pred),
    print("Recall score  test dataset:  ", recall_score(y_test, y_pred), '\n')
    print("f1 score  test dataset :  ", f1_score(y_test, y_pred), '\n')
    return (prob[:,1], y_test)
```

In [ ]:

In [ ]:

In [137…

```
probs , y_test = GradientBoostingClassifier(X,y)
```

```
Train Score :  0.914390756302521

Test Score :  0.8909853249475891

Accuracy Score :  0.8909853249475891

[[125  23]
 [ 29 300]] ---> confusion Matrix

ROC-AUC score  test dataset:   0.9449088145896656

precision score  test dataset:   0.9287925696594427

Recall score  test dataset:   0.9118541033434651

f1 score  test dataset :   0.9202453987730062
```

In [ ]:

In [ ]:

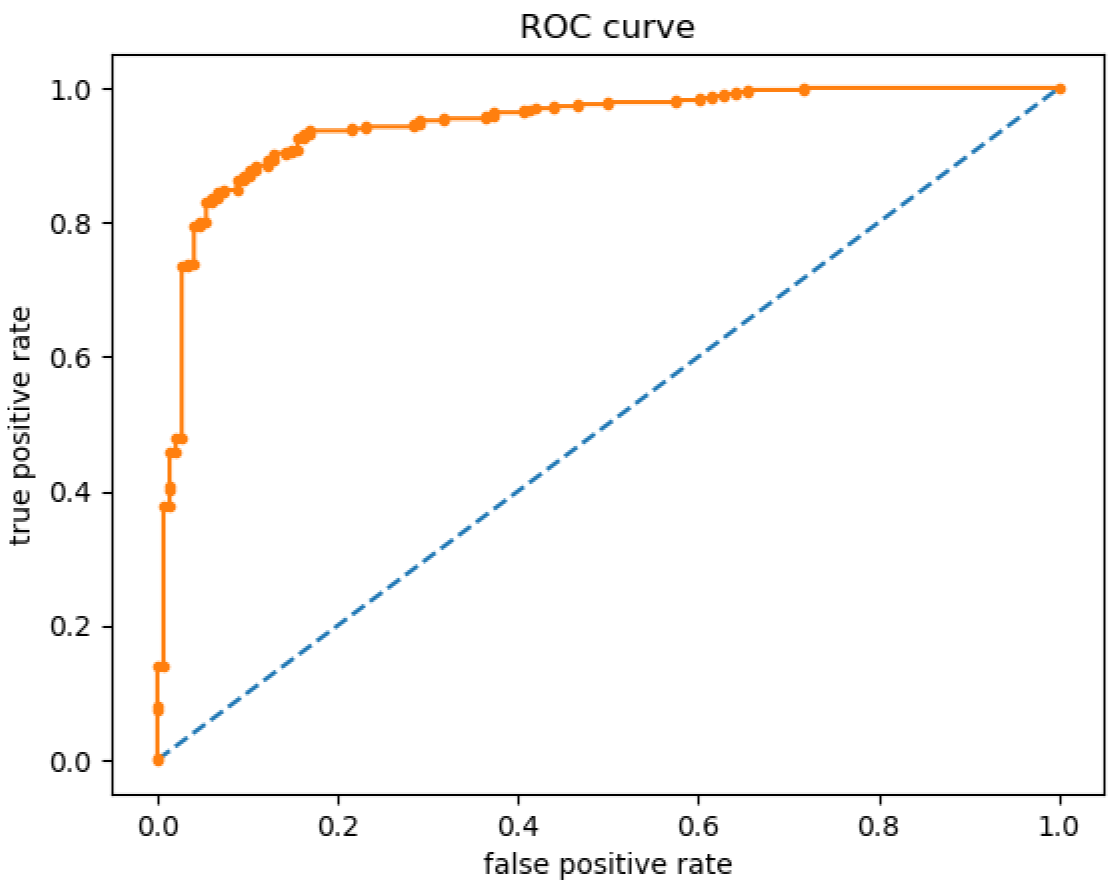
In [138…

```
def plot_pre_curve(y_test,probs):
    from sklearn.metrics import precision_recall_curve
    precision, recall, thresholds = precision_recall_curve(y_test, probs)
    plt.plot([0, 1], [0.5, 0.5], linestyle='--')
    # plot the precision-recall curve for the model
    plt.plot(recall, precision, marker='.')
    plt.title("Precision Recall curve")
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    # show the plot
    plt.show()

def plot_roc(y_test,prob):
    from sklearn.metrics import roc_curve
    fpr, tpr, thresholds = roc_curve(y_test, probs)
    # plot no skill
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr, tpr, marker='.')
    plt.title("ROC curve")
    plt.xlabel('false positive rate')
```

```
    plt.ylabel('true positive rate')
    # show the plot
    plt.show()
```

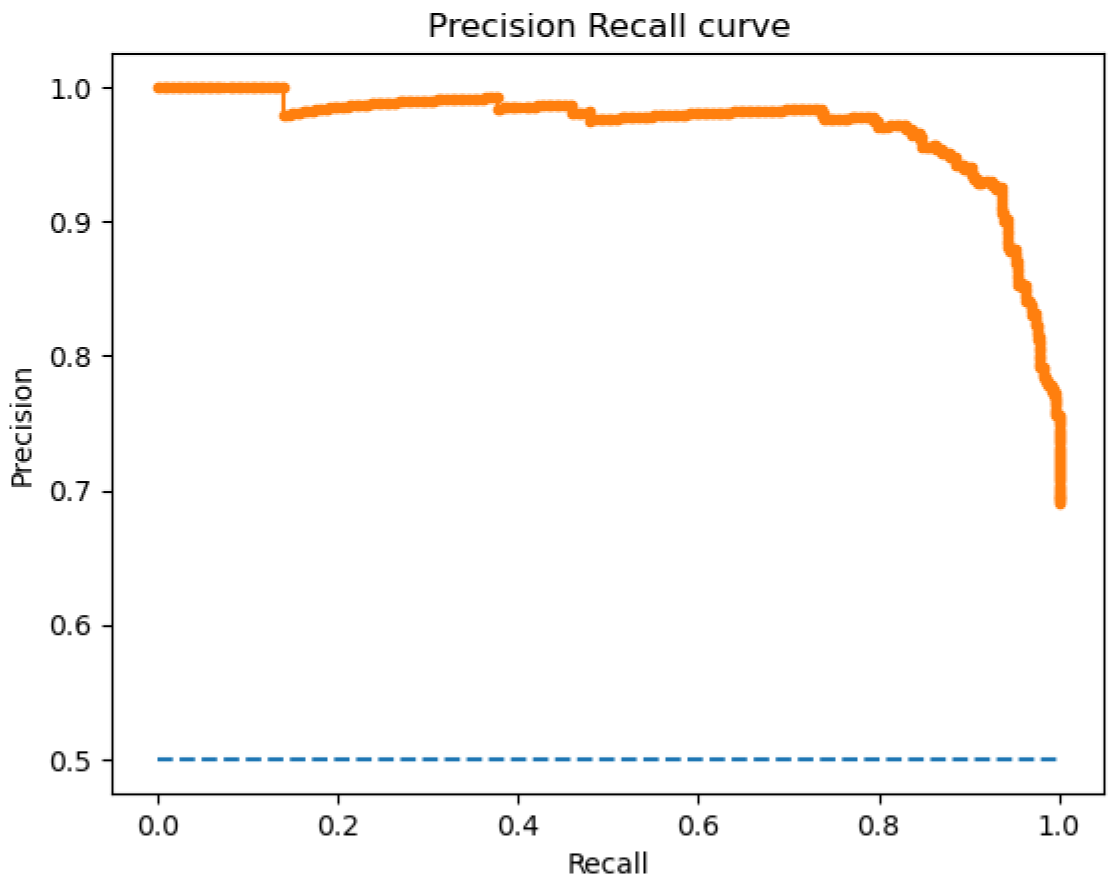
In [139…

```
plot_roc(y_test , probs)
```



In [140…

```
plot_pre_curve(y_test , probs)
```


Precision Recall curve
1.0
0.9
0.8
0.7
0.6
0.5
Precision
0.0
0.2
0.4
0.6
0.8
1.0
Recall

```
In [ ]:
```

```
In [ ]:
```