Project Title: OLA - Ensemble Learning

Problem Statement

- 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)

Concepts Tested:

```
Ensemble Learning- Bagging
Ensemble Learning- Boosting
KNN Imputation of Missing Values
Working with an imbalanced dataset
```

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

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

import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

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

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

Out[3]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	La
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()
```

Missing values checK :

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	La
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]:
```
df["Driver_ID"].nunique()   # 2381 drivers data.
```

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)
```

Analysing Structure of Data :

In [11]:
```
df[df["Driver_ID"]==25]
```

Out[11]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorking
114	01/01/19	25	29.0	Male	C24	1	102077	30/10/17	
115	02/01/19	25	29.0	Male	C24	1	102077	30/10/17	
116	03/01/19	25	29.0	Male	C24	1	102077	30/10/17	
117	04/01/19	25	29.0	Male	C24	1	102077	30/10/17	
118	05/01/19	25	29.0	Male	C24	1	102077	30/10/17	
119	06/01/19	25	29.0	Male	C24	1	102077	30/10/17	
120	07/01/19	25	29.0	Male	C24	1	102077	30/10/17	
121	08/01/19	25	29.0	Male	C24	1	102077	30/10/17	
122	09/01/19	25	29.0	Male	C24	1	102077	30/10/17	
123	10/01/19	25	29.0	Male	C24	1	102077	30/10/17	
124	11/01/19	25	30.0	Male	C24	1	102077	30/10/17	
125	12/01/19	25	30.0	Male	C24	1	102077	30/10/17	
126	01/01/20	25	30.0	Male	C24	1	102077	30/10/17	
127	02/01/20	25	30.0	Male	C24	1	102077	30/10/17	
128	03/01/20	25	30.0	Male	C24	1	102077	30/10/17	
129	04/01/20	25	30.0	Male	C24	1	102077	30/10/17	
130	05/01/20	25	30.0	Male	C24	1	102077	30/10/17	
131	06/01/20	25	30.0	Male	C24	1	102077	30/10/17	
132	07/01/20	25	30.0	Male	C24	1	102077	30/10/17	
133	08/01/20	25	30.0	Male	C24	1	102077	30/10/17	
134	09/01/20	25	30.0	Male	C24	1	102077	30/10/17	
135	10/01/20	25	30.0	Male	C24	1	102077	30/10/17	
136	11/01/20	25	31.0	Male	C24	1	102077	30/10/17	
137	12/01/20	25	31.0	Male	C24	1	102077	30/10/17	

Restructuring the data by aggregation :

In [12]:

```
agg_df = df.groupby(["Driver_ID"]).aggregate({'MMM-YY':len,
                                      "Age":max,

                                      "City":np.unique,
                                     "Education_Level":max,
                                      "Income":np.mean,
                                      "Dateofjoining":np.unique,
#                                      "LastWorkingDate":last_value,
                                      "Joining Designation":np.unique,
                                      "Grade": np.mean,
                                     "Total Business Value":sum,
                                      "Quarterly Rating":np.mean
```

```
                                               })
```

In [13]:
```
agg_df = agg_df.reset_index()
```

In [14]:
```
final_data = agg_df.rename(columns={"MMM-YY":"No_of_Records",
                      "Dateofjoining":"Date_of_joining",
                      "Joining Designation":"Joining_Designation",
                      "Total Business Value" : "Total_Business_Value",
                      "Quarterly Rating":"Quarterly_Rating"})
```

In [15]:
```
final_data
```

Out[15]:

	Driver_ID	No_of_Records	Age	City	Education_Level	Income	Date_of_joining	Joining_Desi
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 [16]:
```
final_data = pd.merge(left = df.groupby(["Driver_ID"])["LastWorkingDate"].unique().
        right = final_data,
         on = "Driver_ID",
         how="outer"
    )
```

In [17]:
```
final_data = pd.merge(left = df.groupby(["Driver_ID"])["Gender"].unique().apply(lam
        right = final_data,
         on = "Driver_ID",
         how="outer"
    )
```

In [18]:
```
data = final_data.copy()
```

In [19]:
```
data["Gender"].value_counts()
```

Out[19]:
```
Male      1380
Female     956
Name: Gender, 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 [20]:
```
pd.Series(np.where(data["LastWorkingDate"].isna(),0,1)).value_counts()
```

Out[20]:
```
1    1616
0     765
dtype: int64
```

In [21]:
```
data["Churn"] = data["LastWorkingDate"].fillna(0)
```

In [22]:
```
def apply_0_1(y):
    if y == 0:
        return 0
    if y != 0:
        return 1
```

In [23]:
```
data["Churn"] = data["Churn"].apply(apply_0_1)
```

In [24]:
```
data["Churn"].value_counts()
```

Out[24]:
```
1    1616
0     765
Name: Churn, dtype: int64
```

In [25]:
```
data["Churn"].value_counts(normalize=True)*100
```

Out[25]:
```
1    67.870643
0    32.129357
Name: Churn, dtype: float64
```

- Class 1 is denoted to the drivers who churned . 68%
- Class 0 is denoted to the drivers who have not churned . 32%
- Data is imbalanced.

In [26]:
```
# data["Total_Business_Value"] = data["Total_Business_Value"].replace({0:np.nan})
```

Converting Date Columns into Datatime format :

In [27]:
```
data["Date_of_joining"] = pd.to_datetime(data["Date_of_joining"])
data["LastWorkingDate"] = pd.to_datetime(data["LastWorkingDate"])
```

In [28]:
```
data["joining_Year"] = data["Date_of_joining"].dt.year
```

In [29]:
```
#  data["joining_month"] = data["Date_of_joining"].dt.month
```

Checking for Missing Values after Restructuring :

In [30]:
```
(data.isna().sum()/len(data))*100
```

Out[30]:
```
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 [31]:
```
data["Churn"].value_counts(normalize=True)*100
```

Out[31]:
```
1    67.870643
0    32.129357
Name: Churn, dtype: float64
```

Feature Engineering :

Whether the quarterly rating has increased for driver?

- For those whose quarterly rating has increased, we assign the value 1.

In [32]:
```
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 [33]:
```
Quarterly_Rating_increased = df.groupby("Driver_ID")["Quarterly Rating"].unique().a
```

In [34]:
```
data = pd.merge(left = Quarterly_Rating_increased,
        right = data,
         on = "Driver_ID",
         how="outer"
    )
```

In [35]:
```
# df.groupby("Driver_ID")["Quarterly Rating"].unique().apply(app_rating_inc)
```

In [36]:
```
data["Quarterly_Rating_increased"] = data["Quarterly Rating"]
```

In [37]:
```
data.drop(["Quarterly Rating"],axis=1,inplace=True)
```

Whether the monthly income has increased for driver?

- For those whose monthly income has increased, we assign the value 1.

In [38]:
```
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 [39]:
```
# df.groupby("Driver_ID")["Income"].unique().apply(app_income_inc).rename("Increase
```

In [40]:
```
data = pd.merge(left = df.groupby("Driver_ID")["Income"].unique().apply(app_income_
        right = data,
         on = "Driver_ID",
         how="outer"
    )
```

In [41]:
```
data
```

Out[41]:

	Driver_ID	Increased_Income	Gender	LastWorkingDate	No_of_Records	Age	City	Educatio
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 [42]:
```
Mdata = data.copy()
```

In [43]:
```
Mdata["Gender"].replace({"Male":0,
                       "Female":1},inplace=True)
```

In [44]:
```
Mdata.drop(["Driver_ID"],axis = 1, inplace=True)
```

In [45]:
```
Mdata.isna().sum()
```

Out[45]:
```
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 [46]:
```
Mdata
```

Out[46]:

	Increased_Income	Gender	LastWorkingDate	No_of_Records	Age	City	Education_Level	In
0	0	0.0	2019-03-11	3	28.0	C23	2	57
1	0	0.0	NaT	2	31.0	C7	2	67
2	0	0.0	2020-04-27	5	43.0	C13	2	65
3	0	0.0	2019-03-07	3	29.0	C9	0	46
4	0	1.0	NaT	5	31.0	C11	1	78
...	...	...	...	...	...	...	...	
2376	0	0.0	NaT	24	34.0	C24	0	82
2377	0	1.0	2020-10-28	3	34.0	C9	0	12
2378	0	0.0	2019-09-22	9	45.0	C19	0	35
2379	0	1.0	2019-06-20	6	28.0	C20	2	69
2380	0	0.0	NaT	7	30.0	C27	2	70

2381 rows × 16 columns

In [47]:
```
pd.to_datetime("2021-06-01")
```

Out[47]:
```
Timestamp('2021-06-01 00:00:00')
```

In [48]:
```
Mdata["LastWorkingDate"] = Mdata["LastWorkingDate"].fillna(pd.to_datetime("2021-06-
```

In [49]:
```
(Mdata["LastWorkingDate"] - Mdata["Date_of_joining"])
```

Out[49]:
```
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 [50]:
```
Mdata["Driver_tenure_days"] = (Mdata["LastWorkingDate"] - Mdata["Date_of_joining"])
```

In [51]:
```
Mdata["Driver_tenure_days"] = Mdata["Driver_tenure_days"].dt.days
```

In [52]:
```
Mdata.drop(["LastWorkingDate","Date_of_joining"],inplace=True,axis = 1)
```

In [53]:
```
Mdata.drop(["Driver_tenure_days"],inplace=True,axis = 1)
```

In [54]:
```
Mdata
```

Out[54]:

	Increased_Income	Gender	No_of_Records	Age	City	Education_Level	Income	Joining_Des
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 [55]:
```
Mdata.columns
```

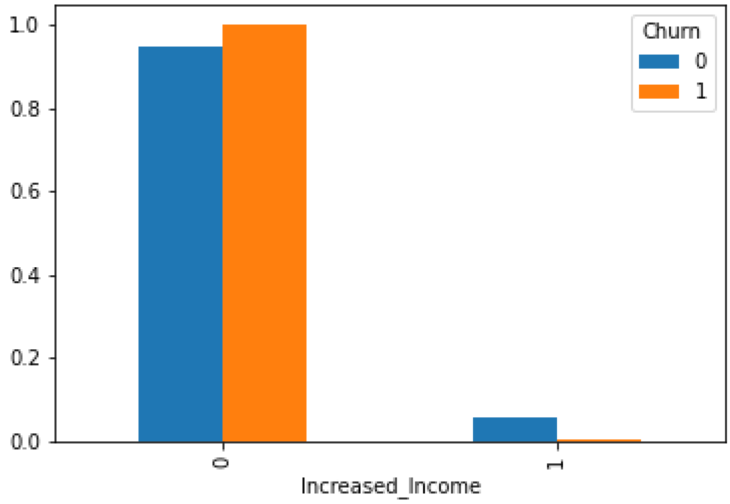
Out[55]:
```
Index(['Increased_Income', 'Gender', 'No_of_Records', 'Age', 'City', 'Education_Le
vel', 'Income', 'Joining_Designation', 'Grade', 'Total_Business_Value', 'Quarterly
_Rating', 'Churn', 'joining_Year', 'Quarterly_Rating_increased'], dtype='object')
```

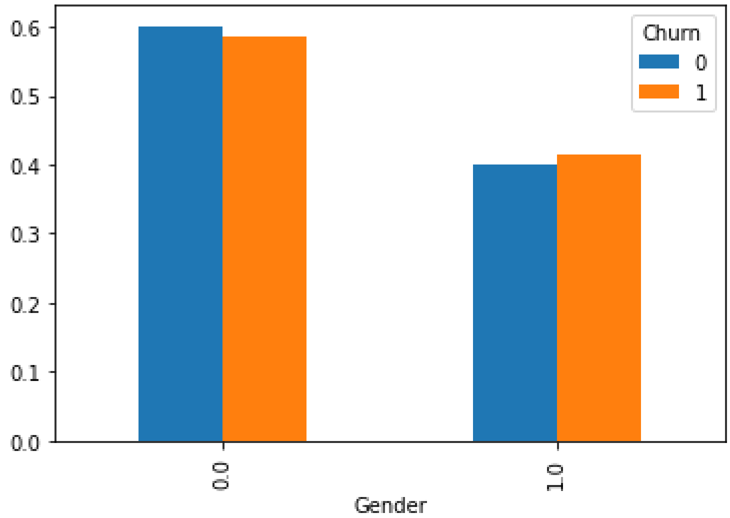
In [56]:
```
Mdata["Grade"] = np.round(Mdata["Grade"])
```

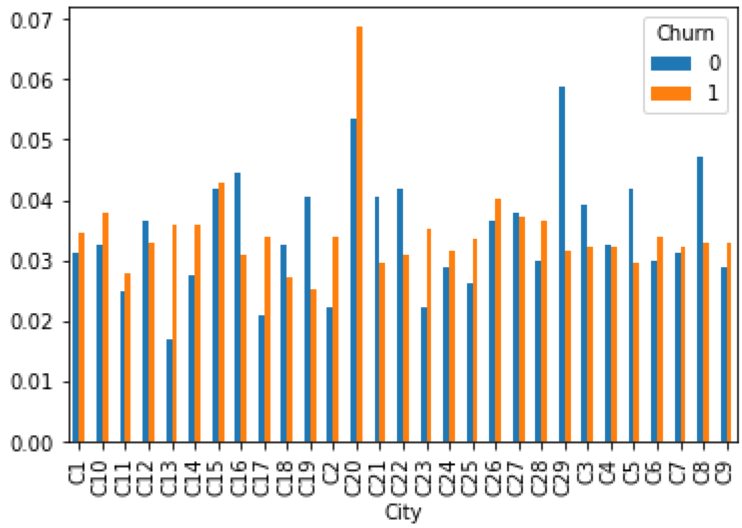
In [57]:
```
Mdata["Quarterly_Rating"]= Mdata["Quarterly_Rating"].round()
```

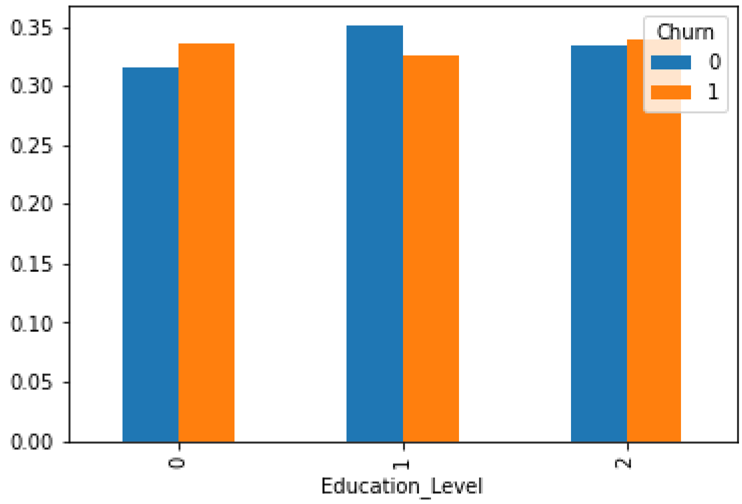
In [58]:
```
categorical_features = ['Increased_Income', 'Gender','City','Education_Level',
                       'Joining_Designation','Grade','Quarterly_Rating','Quarterly_Rati
```

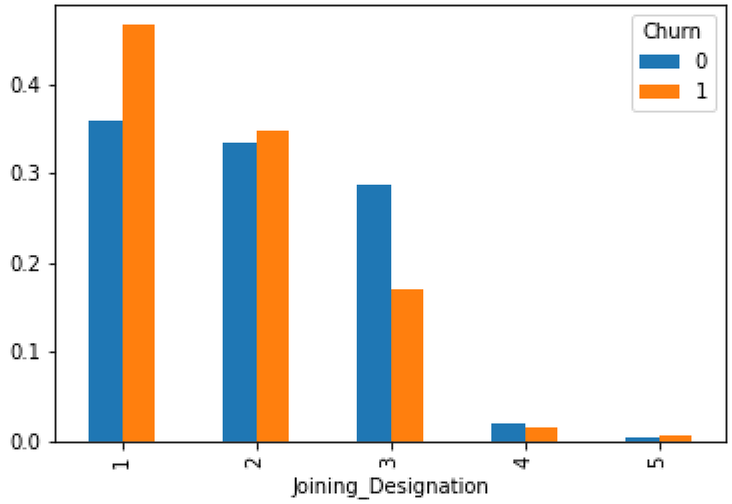
```
for col in categorical_features:
    pd.crosstab(index = Mdata[col],
                columns = Mdata["Churn"],
                normalize="columns").plot(kind = "bar")
    plt.show()
```

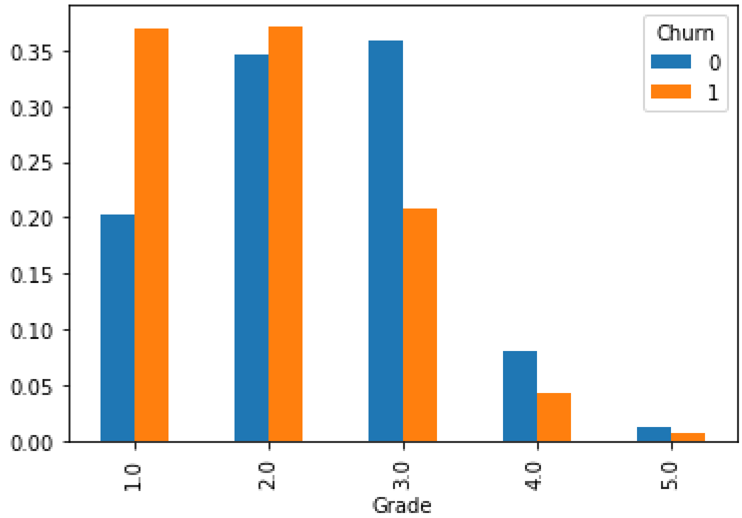


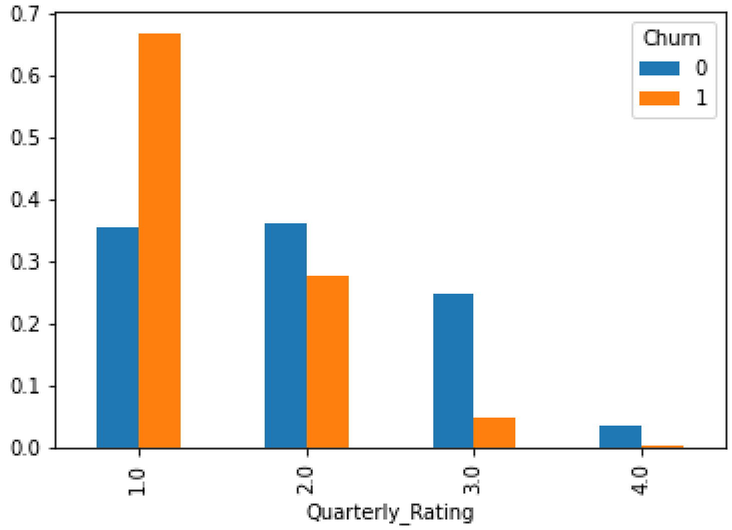


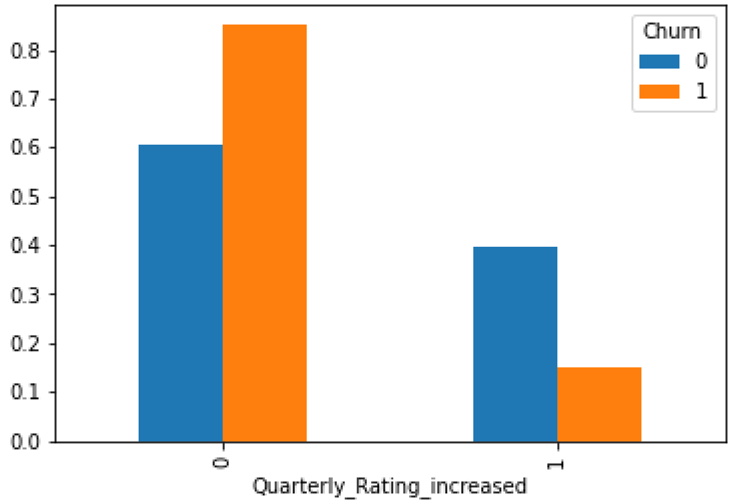


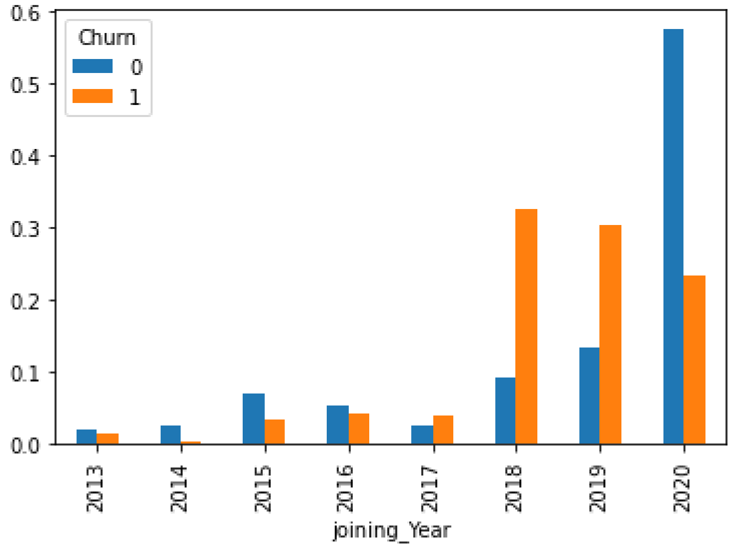

Churn
0
1
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
Education_Level


Churn
0
1
0.4
0.3
0.2
0.1
0.0
1
2
3
4
5
Joining_Designation


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







In [59]:
```
Mdata.isna().sum()
```

Out[59]:
```
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 [60]:
```
from sklearn.impute import SimpleImputer
```

In [61]:
```
imputer = SimpleImputer(strategy='most_frequent')
```

In [62]:
```
Mdata["Gender"] = imputer.fit_transform(X=Mdata["Gender"].values.reshape(-1,1),y=Mc
```

In [63]:
```
Mdata["Gender"].value_counts(dropna=False)
```

Out[63]:
```
0.0    1425
1.0     956
Name: Gender, dtype: int64
```

In [64]:
```
Mdata.isna().sum()
```

Out[64]:
```
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 [65]:
```
from category_encoders import TargetEncoder
TE = TargetEncoder()
```

In [66]:
```
Mdata["City"] = TE.fit_transform(X = Mdata["City"],y = Mdata["Churn"])
```

In [67]:
```
Mdata["joining_Year"] = TE.fit_transform(X = Mdata["joining_Year"],y = Mdata["Churr
```

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

In [68]:
```
Mdata
```

Out[68]:

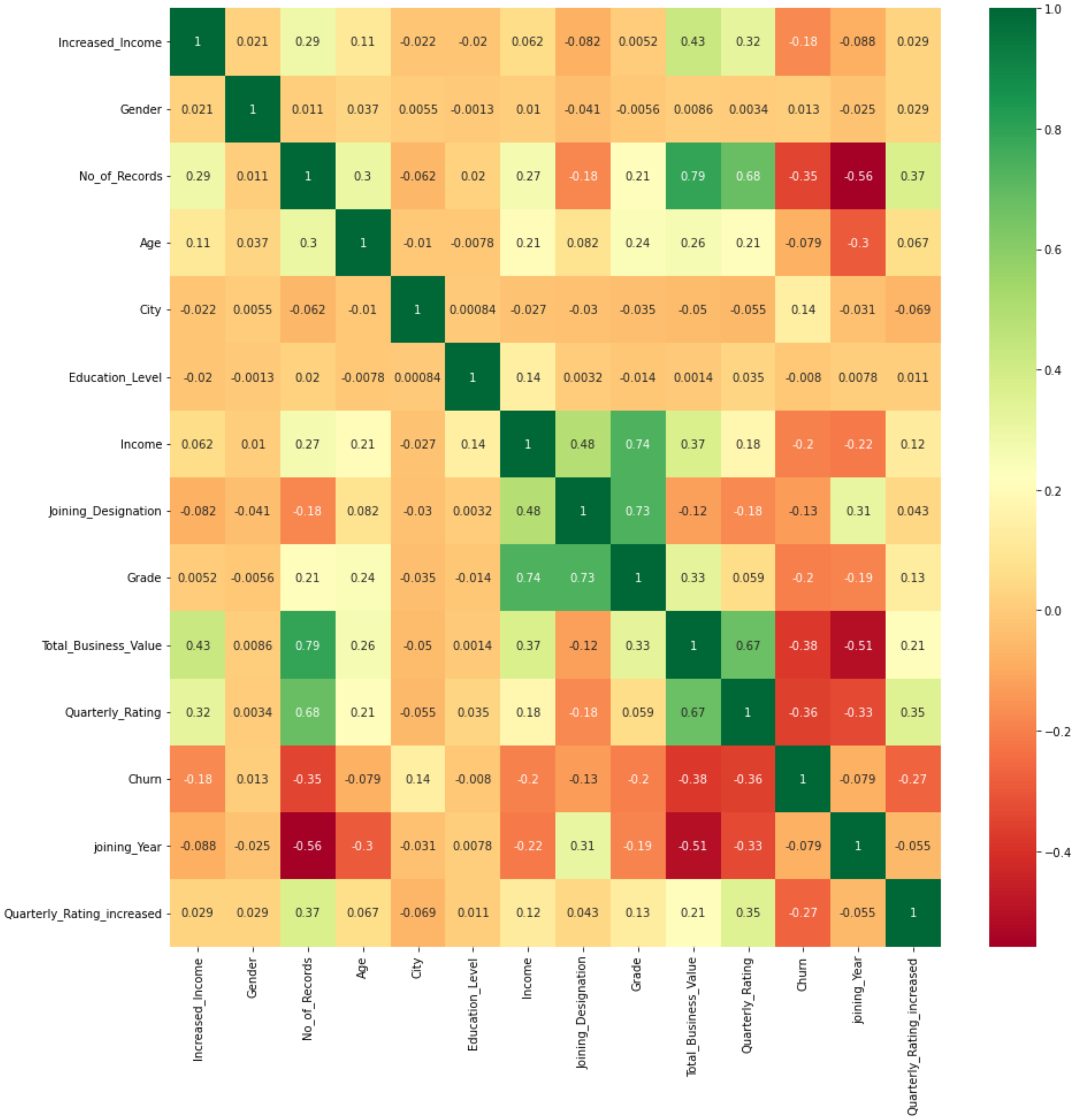
	Increased_Income	Gender	No_of_Records	Age	City	Education_Level	Income	Joining
0	0	0.0	3	28.0	0.770270	2	57387.0	
1	0	0.0	2	31.0	0.684211	2	67016.0	
2	0	0.0	5	43.0	0.816901	2	65603.0	
3	0	0.0	3	29.0	0.706667	0	46368.0	
4	0	1.0	5	31.0	0.703125	1	78728.0	
...	...	...	...	...	...	...	...	
2376	0	0.0	24	34.0	0.698630	0	82815.0	
2377	0	1.0	3	34.0	0.706667	0	12105.0	
2378	0	0.0	9	45.0	0.569444	0	35370.0	
2379	0	1.0	6	28.0	0.730263	2	69498.0	
2380	0	0.0	7	30.0	0.674157	2	70254.0	

2381 rows × 14 columns

In [69]:
```
# Mdata.drop(["No_of_Records"], axis = 1 , inplace= True)
```

In [70]:
```
plt.figure(figsize=(15, 15))
sns.heatmap(Mdata.corr(),annot=True, cmap="RdYlGn", annot_kws={"size":10})
```

Out[70]: <AxesSubplot:>



sns.heatmap(Mdata.corr())

In [71]:
```
X = Mdata.drop(["Churn"],axis = 1)
y = Mdata["Churn"]
```

KNNImputer

In [72]:
```
import numpy as np
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=5)
```

In [73]:
```
X = pd.DataFrame(imputer.fit_transform(X),columns=X.columns)
```

In [74]:
```
X
```

Out[74]:

	Increased_Income	Gender	No_of_Records	Age	City	Education_Level	Income	Joining
0	0.0	0.0	3.0	28.0	0.770270	2.0	57387.0	
1	0.0	0.0	2.0	31.0	0.684211	2.0	67016.0	
2	0.0	0.0	5.0	43.0	0.816901	2.0	65603.0	
3	0.0	0.0	3.0	29.0	0.706667	0.0	46368.0	
4	0.0	1.0	5.0	31.0	0.703125	1.0	78728.0	
...	...	...	...	...	...	...	...	
2376	0.0	0.0	24.0	34.0	0.698630	0.0	82815.0	
2377	0.0	1.0	3.0	34.0	0.706667	0.0	12105.0	
2378	0.0	0.0	9.0	45.0	0.569444	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	0.674157	2.0	70254.0	

2381 rows × 13 columns

In [75]:
```
X.describe()
```

Out[75]:

	Increased_Income	Gender	No_of_Records	Age	City	Education_Level
count	2381.000000	2381.000000	2381.00000	2381.000000	2381.000000	2381.00000
mean	0.018480	0.401512	8.02352	33.663167	0.678706	1.00756
std	0.134706	0.490307	6.78359	5.983375	0.065565	0.81629
min	0.000000	0.000000	1.00000	21.000000	0.531250	0.00000
25%	0.000000	0.000000	3.00000	29.000000	0.634146	0.00000
50%	0.000000	0.000000	5.00000	33.000000	0.698630	1.00000
75%	0.000000	1.000000	10.00000	37.000000	0.719512	2.00000
max	1.000000	1.000000	24.00000	58.000000	0.816901	2.00000

train_test_split

In [76]:
```
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 [77]:
```
y.value_counts()
```

Out[77]:
```
1    1616
0     765
Name: Churn, dtype: int64
```

In [78]:
```
765 + 1616
```

Out[78]:
```
2381
```

StandardScaler

In [79]:
```
from sklearn.preprocessing import StandardScaler
```

In [80]:
```
scaler = StandardScaler()
```

In [81]:
```
scaler.fit(X_train)
```

Out[81]:
```
▾ StandardScaler
StandardScaler()
```

In [82]:
```
X_train = scaler.transform(X_train)
X_test =  scaler.transform(X_test)
```

RandomForestClassifier

In [83]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [84]:
```
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 [85]:
```
RF.fit(X_train,y_train)
```

Out[85]:
```
▾                          RandomForestClassifier
RandomForestClassifier(ccp_alpha=0.0085, class_weight='balanced',
                       criterion='entropy', max_depth=10)
```

In [86]:
```
RF.score(X_train,y_train),RF.score(X_test,y_test)
```

Out[86]:
```
(0.8697478991596639, 0.8679245283018868)
```

In [87]:
```
RF.feature_importances_
```

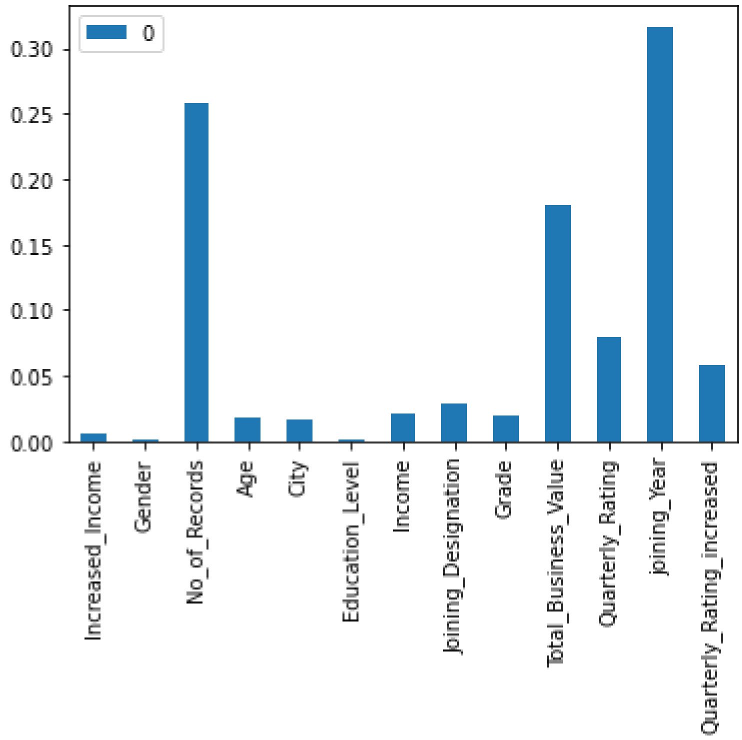
Out[87]:
```
array([0.00590403, 0.00050725, 0.25754642, 0.01764032, 0.0158143 ,
       0.00143737, 0.02139929, 0.02819439, 0.01867883, 0.17940811,
       0.07943974, 0.31669506, 0.05733489])
```

In [88]:
```
X.columns
```

Out[88]:
```
Index(['Increased_Income', 'Gender', 'No_of_Records', 'Age', 'City', 'Education_Le
vel', 'Income', 'Joining_Designation', 'Grade', 'Total_Business_Value', 'Quarterly
_Rating', 'joining_Year', 'Quarterly_Rating_increased'], dtype='object')
```

In [89]:
```
pd.DataFrame(data=RF.feature_importances_,
             index=X.columns).plot(kind="bar")
```

Out[89]:
```
<AxesSubplot:>
```



In [90]:
```
from sklearn.metrics import f1_score , precision_score, recall_score,confusion_matr
```

In [91]:
```
confusion_matrix(y_test,RF.predict(X_test) )
```

Out[91]:
```
array([[141,  21],
       [ 42, 273]], dtype=int64)
```

In [92]:
```
confusion_matrix(y_train,RF.predict(X_train) )
```

Out[92]:
```
array([[ 537,   66],
       [ 182, 1119]], dtype=int64)
```

In [93]:
```
f1_score(y_test,RF.predict(X_test)),f1_score(y_train,RF.predict(X_train))
```

Out[93]:
```
(0.896551724137931, 0.9002413515687852)
```

In [94]:
```
precision_score(y_test,RF.predict(X_test)),precision_score(y_train,RF.predict(X_tra
```

Out[94]:
```
(0.9285714285714286, 0.9443037974683545)
```

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

Out[95]:
```
(0.8666666666666667, 0.8601076095311299)
```

GridSearchCV - on RandomForestClassifier

In [96]:
```
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 [97]:
```
grid_search.fit(X_train,y_train.values.ravel())
```

Out[97]:

```
GridSearchCV
estimator: RandomForestClassifier
    RandomForestClassifier
```

In [98]:
```
grid_search.best_estimator_
```

Out[98]:

```
RandomForestClassifier
RandomForestClassifier(ccp_alpha=0.001, max_depth=10, max_features=7,
                       n_estimators=300)
```

In [99]:
```
grid_search.best_score_
```

Out[99]:
```
0.8881417819617973
```

In [100…
```
grid_search.best_params_
```

Out[100]:
```
{'ccp_alpha': 0.001, 'max_depth': 10, 'max_features': 7, 'n_estimators': 300}
```

In [101…
```
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 [102…
```
RF.fit(X_train , y_train)
```

Out[102]:
```
▾ RandomForestClassifier
RandomForestClassifier(ccp_alpha=0.0001, class_weight='balanced',
                       criterion='entropy', max_depth=7)
```

In [103…
```
RF.score(X_train,y_train),RF.score(X_test,y_test)
```

Out[103]:
```
(0.9028361344537815, 0.8825995807127882)
```

In [104…
```
y_test_pred = RF.predict(X_test)
y_train_pred = RF.predict(X_train)
```

In [105…
```
f1_score(y_test,y_test_pred),f1_score(y_train,y_train_pred)
```

Out[105]:
```
(0.9093851132686084, 0.9264998013508144)
```

In [106…
```
precision_score(y_test,y_test_pred),precision_score(y_train,y_train_pred)
```

Out[106]:
```
(0.9273927392739274, 0.9588815789473685)
```

In [107…
```
recall_score(y_test,y_test_pred),recall_score(y_train,y_train_pred)
```

Out[107]:
```
(0.8920634920634921, 0.8962336664104535)
```

BaggingClassifier

In [108…
```
from sklearn.tree import DecisionTreeClassifier
```

In [109…
```
from sklearn.ensemble import BaggingClassifier
```

In [110…
```
bagging_classifier_model = BaggingClassifier(base_estimator=  DecisionTreeClassifie

                                             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 [111…
```
bagging_classifier_model.fit(X_train,y_train)
```

Out[111]:
```
▸ BaggingClassifier
▸ base_estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

In [112…
```
from sklearn.metrics import f1_score , precision_score, recall_score,confusion_matr
```

In [113…
```
y_test_pred = bagging_classifier_model.predict(X_test)
y_train_pred = bagging_classifier_model.predict(X_train)
```

In [114…
```
confusion_matrix(y_test,y_test_pred)
```
Out[114]:
```
array([[144,  18],
       [ 39, 276]], dtype=int64)
```
In [115…
```
confusion_matrix(y_train,y_train_pred)
```
Out[115]:
```
array([[ 558,   45],
       [ 116, 1185]], dtype=int64)
```
In [116…
```
f1_score(y_test,y_test_pred),f1_score(y_train,y_train_pred)
```
Out[116]:
```
(0.9064039408866995, 0.9363887791386803)
```
In [117…
```
precision_score(y_test,y_test_pred),precision_score(y_train,y_train_pred)
```
Out[117]:
```
(0.9387755102040817, 0.9634146341463414)
```
In [118…
```
recall_score(y_test,y_test_pred),recall_score(y_train,y_train_pred)
```
Out[118]:
```
(0.8761904761904762, 0.9108378170637971)
```
In [119…
```
bagging_classifier_model.score(X_test,y_test)
```
Out[119]:
```
0.8805031446540881
```
In [120…
```
bagging_classifier_model.score(X_train,y_train)
```
Out[120]:
```
0.9154411764705882
```
In [121…
```
# !pip install xgboost
```
In [122…
```
from xgboost import XGBClassifier
```
In [123…
```
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[123]:
```
{'max_depth': 2, 'n_estimators': 100}
```

In [124…
```
xgb = XGBClassifier(n_estimators=100,
                   max_depth = 2)
xgb.fit(X_train, y_train)
```

Out[124]:

▾ XGBClassifier

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree
=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_bin=256, max_cat_threshold=64, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=2, max_leaves=0, min_child_we
ight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
```

In [125…
```
y_test_pred = xgb.predict(X_test)
y_train_pred = xgb.predict(X_train)
```

In [126…
```
confusion_matrix(y_test,y_test_pred)
```

Out[126]:
```
array([[124,  38],
       [ 24, 291]], dtype=int64)
```

In [127…
```
confusion_matrix(y_train,y_train_pred)
```

Out[127]:
```
array([[ 515,   88],
       [  76, 1225]], dtype=int64)
```

In [128…
```
xgb.score(X_train,y_train),xgb.score(X_test,y_test)
```

Out[128]:
```
(0.9138655462184874, 0.870020964360587)
```

In [129…
```
f1_score(y_test,y_test_pred),f1_score(y_train,y_train_pred)
```

Out[129]:
```
(0.9037267080745341, 0.9372609028309103)
```

In [130…
```
recall_score(y_test,y_test_pred),recall_score(y_train,y_train_pred)
```

Out[130]:
```
(0.9238095238095239, 0.9415833973866257)
```

In [131…
```
precision_score(y_test,y_test_pred),precision_score(y_train,y_train_pred)
```

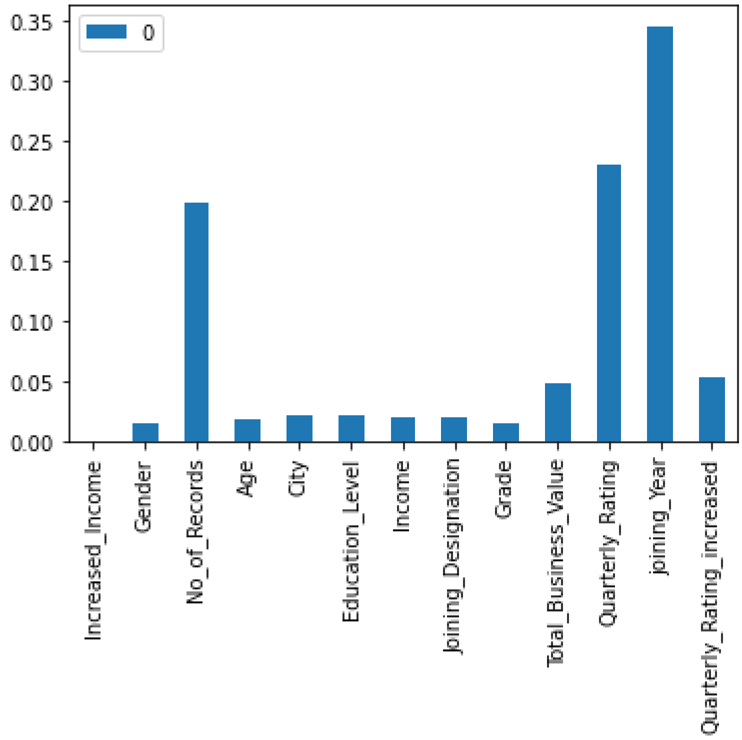
Out[131]:
```
(0.8844984802431611, 0.9329779131759329)
```

In [132…
```
xgb.feature_importances_
```

Out[132]:
```
array([0.        , 0.01420613, 0.19747032, 0.01697209, 0.02113413,
       0.02173466, 0.01887255, 0.01899261, 0.01514235, 0.04826141,
       0.22931552, 0.3451485 , 0.05274975], dtype=float32)
```

In [133…
```
pd.DataFrame(data=xgb.feature_importances_,
             index=X.columns).plot(kind="bar")
```

Out[133]:
```
<AxesSubplot:>
```



GradientBoostingClassifier

In [134…

```
def GradientBoostingClassifier(X, y):
    from sklearn.ensemble import  GradientBoostingClassifier
    from sklearn.metrics import f1_score, accuracy_score , roc_auc_score,auc,recall
    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]),'\n')
    print("precision score  test dataset:  ", precision_score(y_test, y_pred),'\n')
    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 [135…

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

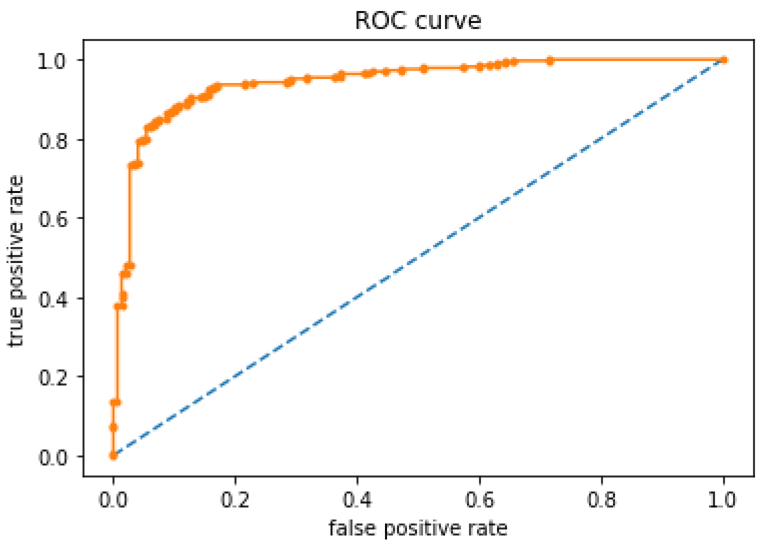
```
Train Score :  0.914390756302521

Test Score :  0.8909853249475891

Accuracy Score :  0.8909853249475891

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

ROC-AUC score  test dataset:   0.9447855910621867

precision score  test dataset:   0.9287925696594427

Recall score  test dataset:   0.9118541033434651

f1 score  test dataset :   0.9202453987730062
```

In [136…

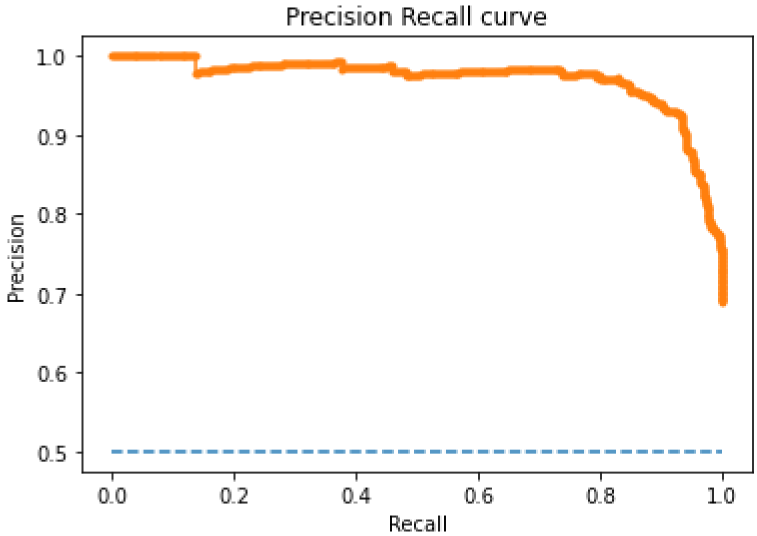
```python
def plot_pre_curve(y_test,probs):
    from sklearn.metrics import precision_recall_curve
    precision, recall, thresholds = precision_recall_curve(y_test, probs)
    plt.plot([0, 1], [0.5, 0.5], linestyle='--')
    # plot the precision-recall curve for the model
    plt.plot(recall, precision, 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 [137…

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



In [138...
```
plot_pre_curve(y_test , probs)
```



Inferences :

from data distribution: Male 1380 Female 956

Churn : distribution: 1 1616 (67.870%) 0 765 (32.12%)

- Probability of Churn is higher in case of education level 0 and 1 than 2.
- in case of joining destination 1, probability of churn is higher.

- in case of quarterly rating is 1, probability of churn is significantly higher.
- also same pattern is observed in case of when driver's quarterly rating has increased through out tenure.

- due to some reason , for drivers who joined in 2018 and 2019 , probability of churn is very high compare to 2020 and before 2018.

Random Forest :

- train and test score : (0.8697478991596639, 0.8679245283018868)
- feature importance : highest is : joining year , followed by No of records available in data, and total business value.
- recall : 0.866
- precision: 0.928
- f1-score : 0.89

on Grid Search CV : RF :

- best params : ccp_alpha=0.001, max_depth=10, max_features=7,n_estimators=300
- Gridsearch RF best score : 0.8881417819617973

Bagging Classfier : wwith Decision Tree :

- with 50 DTs. when max_depth=7, class_weight="balanced"
- f1 score : 0.9064039408866995

- precision : 0.9387755102040817
- recall_score : 0.8761904761904762
- accuracy: 0.880

XGBoost Classifier: (Grid SEARCH CV :) 'max_depth': 2, 'n_estimators': 100

- test Scores :
- Accuracy : 0.87
- f1 score : 0.90
- recall : 0.923
- precision : 0.884

- feature importance : highest is : joining year , followed by No of records available in data, and total business value.

GradientBoostingClassifier : GBDC:

- Train Score : 0.914390756302521
- Test Score : 0.8909853249475891
- Accuracy Score : 0.8909853249475891
- ROC-AUC score test dataset: 0.9447855910621867
- precision score test dataset: 0.9287925696594427
- Recall score test dataset: 0.9118541033434651
- f1 score test dataset : 0.9202453987730062