OLA_Ensemble_BusinessCase

January 25, 2025

[3]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
```

[4]:
```python
df=pd.read_csv('/content/sample_data/ola_driver_scaler.csv')
df.head()
```

```
/usr/local/lib/python3.11/dist-
packages/google/colab/_dataframe_summarizer.py:88: UserWarning: Could not infer
format, so each element will be parsed individually, falling back to `dateutil`.
To ensure parsing is consistent and as-expected, please specify a format.
  cast_date_col = pd.to_datetime(column, errors="coerce")
```

[4]:
```
   Unnamed: 0    MMM-YY  Driver_ID   Age  Gender City  Education_Level  \
0           0  01/01/19          1  28.0     0.0  C23                2
1           1  02/01/19          1  28.0     0.0  C23                2
2           2  03/01/19          1  28.0     0.0  C23                2
3           3  11/01/20          2  31.0     0.0   C7                2
4           4  12/01/20          2  31.0     0.0   C7                2

   Income Dateofjoining LastWorkingDate  Joining Designation  Grade  \
0   57387      24/12/18             NaN                    1      1
1   57387      24/12/18             NaN                    1      1
2   57387      24/12/18        03/11/19                    1      1
3   67016      11/06/20             NaN                    2      2
4   67016      11/06/20             NaN                    2      2

   Total Business Value  Quarterly Rating
0               2381060                 2
1               -665480                 2
2                     0                 2
3                     0                 1
4                     0                 1
```

[5]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Unnamed: 0            19104 non-null  int64  
 1   MMM-YY                19104 non-null  object 
 2   Driver_ID             19104 non-null  int64  
 3   Age                   19043 non-null  float64
 4   Gender                19052 non-null  float64
 5   City                  19104 non-null  object 
 6   Education_Level       19104 non-null  int64  
 7   Income                19104 non-null  int64  
 8   Dateofjoining         19104 non-null  object 
 9   LastWorkingDate       1616 non-null   object 
 10  Joining Designation   19104 non-null  int64  
 11  Grade                 19104 non-null  int64  
 12  Total Business Value  19104 non-null  int64  
 13  Quarterly Rating      19104 non-null  int64  
dtypes: float64(2), int64(8), object(4)
memory usage: 2.0+ MB
```

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)

Our introduction to Data got following Observations:

There are 19104 rows and 14 columns

Null Values observed in 3 columns

Data type of few columns need correction, converting to date time etc..

Data requires pre-processing before model building. It will be done in upcoming sections

Exploratory Data Analysis

Feature Engineering

Conversion to Required Data types

Checking Null Values

Checking Duplicates

Checking Outliers

[6]:
```
df1=df.copy()
```

[7]:
```
#Remove Unnamed column since we have Driver Id with unique values
df1.drop('Unnamed: 0', axis=1,inplace=True)
```

[8]:
```
df1 = df1.rename(columns={'MMM-YY': 'Reporting_Date'})
```

[9]:
```
# Convert Reporting_Date to datetime, it's in MM-DD-YY format
df1['Reporting_Date'] = pd.to_datetime(df1['Reporting_Date'], format='%m/%d/
 ↪%y',errors='coerce')

# Convert Dateofjoining to datetime, it's in DD-MM-YY format
df1['Dateofjoining'] = pd.to_datetime(df1['Dateofjoining'], format='%d/%m/
 ↪%y',errors='coerce')

# Convert LastWorkingDate to datetime, it's in DD-MM-YY format
df1['LastWorkingDate'] = pd.to_datetime(df1['LastWorkingDate'], format='%d/%m/
 ↪%y',errors='coerce')
```

[10]:
```
# Non-numeric columns
obj_cols = df1.select_dtypes(include='object').columns
obj_cols
```

[10]:
```
Index(['City'], dtype='object')
```

[11]:
```
for _ in obj_cols:
    print()
    print(f'Total Unique Values in {_} column are :- {df1[_].nunique()}')
    print(f'Value counts in {_} column are :-\n {df1[_].value_counts()}')
    print()
    print('-'*120)
```

```
Total Unique Values in City column are :- 29
Value counts in City column are :-
```

```
 City
C20    1008
C29     900
C26     869
C22     809
C27     786
C15     761
C10     744
C12     727
C8      712
C16     709
C28     683
C1      677
C6      660
C5      656
C14     648
C3      637
C24     614
C7      609
C21     603
C25     584
C19     579
C4      578
C13     569
C18     544
C23     538
C9      520
C2      472
C11     468
C17     440
Name: count, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------
```

[12]:
```python
# Numeric columns
num_cols = df1.select_dtypes(include='number').columns
num_cols
```

[12]:
```
Index(['Driver_ID', 'Age', 'Gender', 'Education_Level', 'Income',
       'Joining Designation', 'Grade', 'Total Business Value',
       'Quarterly Rating'],
      dtype='object')
```

[13]:
```python
for _ in num_cols:
    print()
    print(f'Total Unique Values in {_} column are :- {df1[_].nunique()}')
```

```
    print(f'Value counts in {_} column are :-\n {df1[_].
↪value_counts(normalize=True)}')
    print()
    print('-'*120)
```

```
Total Unique Values in Driver_ID column are :- 2381
Value counts in Driver_ID column are :-
 Driver_ID
2110    0.001256
2617    0.001256
1623    0.001256
1642    0.001256
1644    0.001256
          …
1614    0.000052
445     0.000052
2397    0.000052
1619    0.000052
469     0.000052
Name: proportion, Length: 2381, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Age column are :- 36
Value counts in Age column are :-
 Age
36.0    0.067374
33.0    0.065641
34.0    0.064801
30.0    0.060180
32.0    0.060022
35.0    0.059759
31.0    0.056504
29.0    0.053195
37.0    0.045266
38.0    0.044846
39.0    0.041380
28.0    0.040540
27.0    0.039069
40.0    0.036811
41.0    0.034711
26.0    0.029722
42.0    0.025101
25.0    0.023578
44.0    0.021373
```

```
43.0    0.020953
45.0    0.019482
46.0    0.018379
24.0    0.014388
47.0    0.011763
23.0    0.010135
48.0    0.007562
49.0    0.005199
22.0    0.004831
52.0    0.004096
51.0    0.003781
50.0    0.003623
21.0    0.001838
53.0    0.001365
54.0    0.001260
55.0    0.001103
58.0    0.000368
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Gender column are :- 2
Value counts in Gender column are :-
 Gender
0.0    0.581251
1.0    0.418749
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Education_Level column are :- 3
Value counts in Education_Level column are :-
 Education_Level
1    0.359296
2    0.331187
0    0.309516
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Income column are :- 2383
Value counts in Income column are :-
 Income
48747     0.002984
109652    0.001675
```

```
68356     0.001570
42260     0.001466
67490     0.001466
            …
44706     0.000052
72186     0.000052
67162     0.000052
22132     0.000052
35091     0.000052
Name: proportion, Length: 2383, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Joining Designation column are :- 5
Value counts in Joining Designation column are :-
 Joining Designation
1    0.514604
2    0.311715
3    0.149026
4    0.017850
5    0.006805
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Grade column are :- 5
Value counts in Grade column are :-
 Grade
2    0.346891
1    0.272299
3    0.252617
4    0.112228
5    0.015965
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Total Business Value column are :- 10181
Value counts in Total Business Value column are :-
 Total Business Value
0          0.340191
200000     0.015075
250000     0.007747
500000     0.006857
300000     0.005601
```

```
         …
130520    0.000052
275330    0.000052
820160    0.000052
203040    0.000052
448370    0.000052
Name: proportion, Length: 10181, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Quarterly Rating column are :- 4
Value counts in Quarterly Rating column are :-
 Quarterly Rating
1    0.401958
2    0.290672
3    0.203884
4    0.103486
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------
```

[14]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Reporting_Date        19104 non-null  datetime64[ns]
 1   Driver_ID             19104 non-null  int64
 2   Age                   19043 non-null  float64
 3   Gender                19052 non-null  float64
 4   City                  19104 non-null  object
 5   Education_Level       19104 non-null  int64
 6   Income                19104 non-null  int64
 7   Dateofjoining         19104 non-null  datetime64[ns]
 8   LastWorkingDate       1616 non-null   datetime64[ns]
 9   Joining Designation   19104 non-null  int64
 10  Grade                 19104 non-null  int64
 11  Total Business Value  19104 non-null  int64
 12  Quarterly Rating      19104 non-null  int64
dtypes: datetime64[ns](3), float64(2), int64(7), object(1)
memory usage: 1.9+ MB
```

Feature Engineering

Target Variable Creation: Having value 1 if the Last Working Date of the Driver is present else 0

[15]:
```
target =  (df1.groupby('Driver_ID').agg({'LastWorkingDate':
↪'last'})['LastWorkingDate'].isna()).reset_index()
target['LastWorkingDate'].replace({True:0,False:1},inplace=True)
target.rename(columns={'LastWorkingDate':'Target'},inplace=True)
target.head()
```

```
<ipython-input-15-7048b553d20d>:2: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  target['LastWorkingDate'].replace({True:0,False:1},inplace=True)
<ipython-input-15-7048b553d20d>:2: FutureWarning: Downcasting behavior in
`replace` is deprecated and will be removed in a future version. To retain the
old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to
the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  target['LastWorkingDate'].replace({True:0,False:1},inplace=True)
```

[15]:

	Driver_ID	Target
0	1	1
1	2	0
2	4	1
3	5	1
4	6	0

[16]:
```
QR1 =  (df1.groupby('Driver_ID').agg({'Quarterly Rating':'first'})['Quarterly␣
↪Rating']).reset_index()
QR2 = (df1.groupby('Driver_ID').agg({'Quarterly Rating':'last'})['Quarterly␣
↪Rating']).reset_index()
```

[17]:
```
QR1.shape,QR2.shape
```

[17]:
```
((2381, 2), (2381, 2))
```

[18]:
```
QR1.isna().sum(),QR2.isna().sum()
```

[18]:
```
(Driver_ID           0
 Quarterly Rating    0
 dtype: int64,
 Driver_ID           0
```

```
 Quarterly Rating    0
 dtype: int64)
```

[19]:
```
target = target.merge(QR1,on='Driver_ID')
target = target.merge(QR2,on='Driver_ID')
```

[20]:
```
target['Rating_incr']=np.where(target['Quarterly Rating_x'] < target['Quarterly␣
↪Rating_y'], 1,0)
```

[21]:
```
target.head()
```

[21]:

	Driver_ID	Target	Quarterly Rating_x	Quarterly Rating_y	Rating_incr
0	1	1	2	2	0
1	2	0	1	1	0
2	4	1	1	1	0
3	5	1	1	1	0
4	6	0	1	2	1

Income_incr: If the monthly income has increased for any driver then value 1 else 0

[22]:
```
incm1 =  (df1.groupby('Driver_ID').agg({'Income':'first'})['Income']).
↪reset_index()
incm2 = (df1.groupby('Driver_ID').agg({'Income':'last'})['Income']).
↪reset_index()
incm1.shape,incm2.shape
```

[22]:
```
((2381, 2), (2381, 2))
```

[23]:
```
incm1.isna().sum(),incm2.isna().sum()
```

[23]:
```
(Driver_ID    0
 Income       0
 dtype: int64,
 Driver_ID    0
 Income       0
 dtype: int64)
```

[24]:
```
target = target.merge(incm1,on='Driver_ID')
target = target.merge(incm2,on='Driver_ID')
```

[25]:
```
target['Income_incr'] = np.where(target['Income_x'] < target['Income_y'], 1,0)
```

New Features Created

[26]:
```
target2=target[['Driver_ID','Target','Rating_incr','Income_incr']]
target2.head()
```

[26]:

	Driver_ID	Target	Rating_incr	Income_incr
0	1	1	0	0
1	2	0	0	0
2	4	1	0	0
3	5	1	0	0
4	6	0	1	0

Aggregation and Merger of Columns based on Driver_ID

[27]:

```
df2=df1.copy()
```

[28]:

```
functions = {'Reporting_Date':'count',
             'Driver_ID':'first',
             'Age':'max',
             'Gender':'last',
            'City':'last',
             'Education_Level':'last',
             'Dateofjoining':'first',
            'LastWorkingDate':'last',
             'Grade':'last',
             'Total Business Value':'sum',
            'Income':'last',
             'Joining Designation':'last',
             'Quarterly Rating':'last'}
df2 = df2.groupby([df2['Driver_ID']]).aggregate(functions)
df2.rename(columns={'Reporting_Date':'Reportings'},inplace=True)
```

[29]:

```
df2.reset_index(drop=True, inplace=True)
df2 = df2.merge(target2,on='Driver_ID')
df2.columns = df2.columns.str.strip()
df2
```

[29]:

	Reportings	Driver_ID	Age	Gender	City	Education_Level	Dateofjoining	\
0	3	1	28.0	0.0	C23	2	2018-12-24	
1	2	2	31.0	0.0	C7	2	2020-06-11	
2	5	4	43.0	0.0	C13	2	2019-07-12	
3	3	5	29.0	0.0	C9	0	2019-09-01	
4	5	6	31.0	1.0	C11	1	2020-07-31	
…	…	…	…	…	…	…	…	
2376	24	2784	34.0	0.0	C24	0	2015-10-15	
2377	3	2785	34.0	1.0	C9	0	2020-08-28	
2378	9	2786	45.0	0.0	C19	0	2018-07-31	
2379	6	2787	28.0	1.0	C20	2	2018-07-21	
2380	7	2788	30.0	0.0	C27	2	2020-08-06	

	LastWorkingDate	Grade	Total Business Value	Income	\
0	2019-11-03	1	1715580	57387	

```
1             NaT     2              0   67016
2      2020-04-27     2         350000   65603
3      2019-07-03     1         120360   46368
4             NaT     3        1265000   78728
…             …     …              …      …
2376          NaT     3       21748820   82815
2377   2020-10-28     1              0   12105
2378   2019-09-22     2        2815090   35370
2379   2019-06-20     1         977830   69498
2380          NaT     2        2298240   70254

      Joining Designation  Quarterly Rating  Target  Rating_incr  Income_incr
0                       1                 2       1            0            0
1                       2                 1       0            0            0
2                       2                 1       1            0            0
3                       1                 1       1            0            0
4                       3                 2       0            1            0
…                     …                 …     …          …          …
2376                    2                 4       0            1            0
2377                    1                 1       1            0            0
2378                    2                 1       1            0            0
2379                    1                 1       1            0            0
2380                    2                 2       0            1            0

[2381 rows x 16 columns]
```

Finally we got our aggregated dataset with Target variable

There are 2381 rows and 15 columns signifying unique 2381 Driver ids

[30]:
```
df2.isna().sum()
```

[30]:
```
Reportings               0
Driver_ID                0
Age                      0
Gender                   0
City                     0
Education_Level          0
Dateofjoining            0
LastWorkingDate        765
Grade                    0
Total Business Value     0
Income                   0
Joining Designation      0
Quarterly Rating         0
Target                   0
Rating_incr              0
Income_incr              0
```

```
dtype: int64
```

For the null values only in LastWorkingDate column is present for the reason that Driver have not left. We shall be using this feature so keeping it as it is

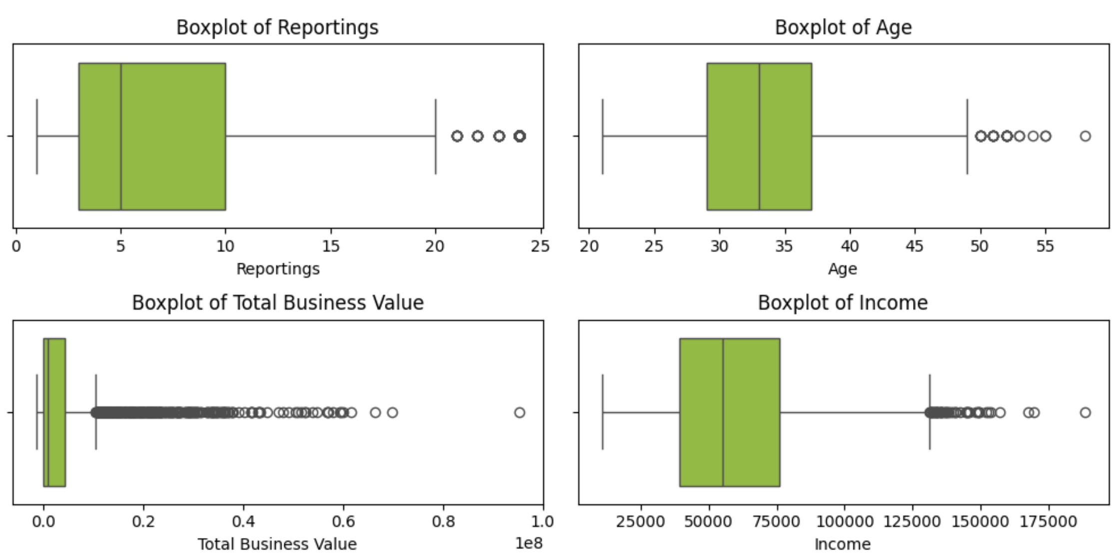
[31]:
```python
df2[df2.duplicated()]
```

[31]:
```
Empty DataFrame
Columns: [Reportings, Driver_ID, Age, Gender, City, Education_Level,
Dateofjoining, LastWorkingDate, Grade, Total Business Value, Income, Joining
Designation, Quarterly Rating, Target, Rating_incr, Income_incr]
Index: []
```

Checking OUTLIERS

[32]:
```python
num_cols=['Reportings','Age','Total Business Value','Income']
```

[33]:
```python
fig = plt.figure(figsize=(10,5))
i=1
for col in num_cols:
  ax = plt.subplot(2,2,i)
  sns.boxplot(x=df2[col],color='yellowgreen')
  plt.title(f'Boxplot of {col}')
  i += 1

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



Highlights:

Data is showing outliers esp. in Total Business Value We have limited dataset and varied values are signifying the range for each Driver, we must keep intact the diversity so that we can make better predictions for any new data which can be of any range.

Analysis and Distribution of Variables Statistical Summary

UniVariate Analysis

BiVariate Analysis

Impact of Each Feature on Churn

[34]:
```
df3=df2.copy()
df3.nunique()
```

[34]:
```
Reportings                24
Driver_ID               2381
Age                       36
Gender                     2
City                      29
Education_Level            3
Dateofjoining            869
LastWorkingDate          493
Grade                      5
Total Business Value    1629
Income                  2339
Joining Designation        5
Quarterly Rating           4
Target                     2
Rating_incr                2
Income_incr                2
dtype: int64
```

[35]:
```
columns_to_convert=['Reportings','Gender','City','Education_Level','Grade','Joining␣
↪Designation','Quarterly Rating','Rating_incr','Income_incr','Target']

df3[columns_to_convert] = df3[columns_to_convert].apply(lambda x: x.
↪astype('category'))
df3.describe(include='all').T
```

[35]:
```
                       count unique  top    freq  \
Reportings            2381.0   24.0  5.0   309.0
Driver_ID             2381.0    NaN  NaN     NaN
Age                   2381.0    NaN  NaN     NaN
Gender                2381.0    2.0  0.0  1404.0
City                    2381     29  C20     152
Education_Level       2381.0    3.0  2.0   802.0
Dateofjoining           2381    NaN  NaN     NaN
LastWorkingDate         1616    NaN  NaN     NaN
```

```
Grade                  2381.0    5.0  2.0   855.0
Total Business Value   2381.0    NaN  NaN     NaN
Income                 2381.0    NaN  NaN     NaN
Joining Designation    2381.0    5.0  1.0  1026.0
Quarterly Rating       2381.0    4.0  1.0  1744.0
Target                 2381.0    2.0  1.0  1616.0
Rating_incr            2381.0    2.0  0.0  2023.0
Income_incr            2381.0    2.0  0.0  2338.0

                                               mean                  min  \
Reportings                                      NaN                  NaN
Driver_ID                               1397.559009                  1.0
Age                                       33.663167                 21.0
Gender                                          NaN                  NaN
City                                            NaN                  NaN
Education_Level                                 NaN                  NaN
Dateofjoining         2019-01-27 12:58:58.009239808  2013-01-04 00:00:00
LastWorkingDate       2019-12-26 23:22:34.455445760  2018-12-31 00:00:00
Grade                                           NaN                  NaN
Total Business Value                 4586741.822764           -1385530.0
Income                                 59334.157077              10747.0
Joining Designation                             NaN                  NaN
Quarterly Rating                                NaN                  NaN
Target                                          NaN                  NaN
Rating_incr                                     NaN                  NaN
Income_incr                                     NaN                  NaN

                                      25%                  50%  \
Reportings                            NaN                  NaN
Driver_ID                           695.0               1400.0
Age                                  29.0                 33.0
Gender                                NaN                  NaN
City                                  NaN                  NaN
Education_Level                       NaN                  NaN
Dateofjoining         2018-06-26 00:00:00  2019-06-23 00:00:00
LastWorkingDate       2019-06-10 00:00:00  2019-12-20 12:00:00
Grade                                 NaN                  NaN
Total Business Value                  0.0             817680.0
Income                            39104.0              55315.0
Joining Designation                   NaN                  NaN
Quarterly Rating                      NaN                  NaN
Target                                NaN                  NaN
Rating_incr                           NaN                  NaN
Income_incr                           NaN                  NaN

                                      75%                  max            std
Reportings                            NaN                  NaN            NaN
```

```
Driver_ID                             2100.0                2788.0      806.161628
Age                                     37.0                  58.0        5.983375
Gender                                   NaN                   NaN             NaN
City                                     NaN                   NaN             NaN
Education_Level                          NaN                   NaN             NaN
Dateofjoining          2020-04-14 00:00:00   2020-12-28 00:00:00             NaN
LastWorkingDate        2020-07-14 00:00:00   2020-12-28 00:00:00             NaN
Grade                                    NaN                   NaN             NaN
Total Business Value               4173650.0            95331060.0  9127115.313446
Income                               75986.0              188418.0    28383.666384
Joining Designation                      NaN                   NaN             NaN
Quarterly Rating                         NaN                   NaN             NaN
Target                                   NaN                   NaN             NaN
Rating_incr                              NaN                   NaN             NaN
Income_incr                              NaN                   NaN             NaN
```

Observations:

Five number of reportings are having highest frequency

Males are higher in ratio than females among Drivers

C20 is the city with maximum drivers

Maximum Drivers have Grade 2

Maximum number of Drivers have Quarterly Rating as 1

[36]:
```python
num_cols=['Reportings','Age','Total Business Value','Income']
#Considering a few integer datatype columns as categorical since they have got␣
↪limited unique values and categorical in nature for EDA purpose
cat_cols=['Gender','City','Education_Level','Grade','Joining␣
↪Designation','Quarterly Rating','Rating_incr','Income_incr','Target']
```

Categorical Features

[37]:
```python
for _ in cat_cols:
    print()
    print(f'Total Unique Values in {_} column are :- {df2[_].nunique()}')
    print(f'Value counts in {_} column are :-\n {df2[_].
↪value_counts(normalize=True)}')
    print()
    print('-'*120)
```

```
Total Unique Values in Gender column are :- 2
Value counts in Gender column are :-
 Gender
0.0    0.589668
1.0    0.410332
```

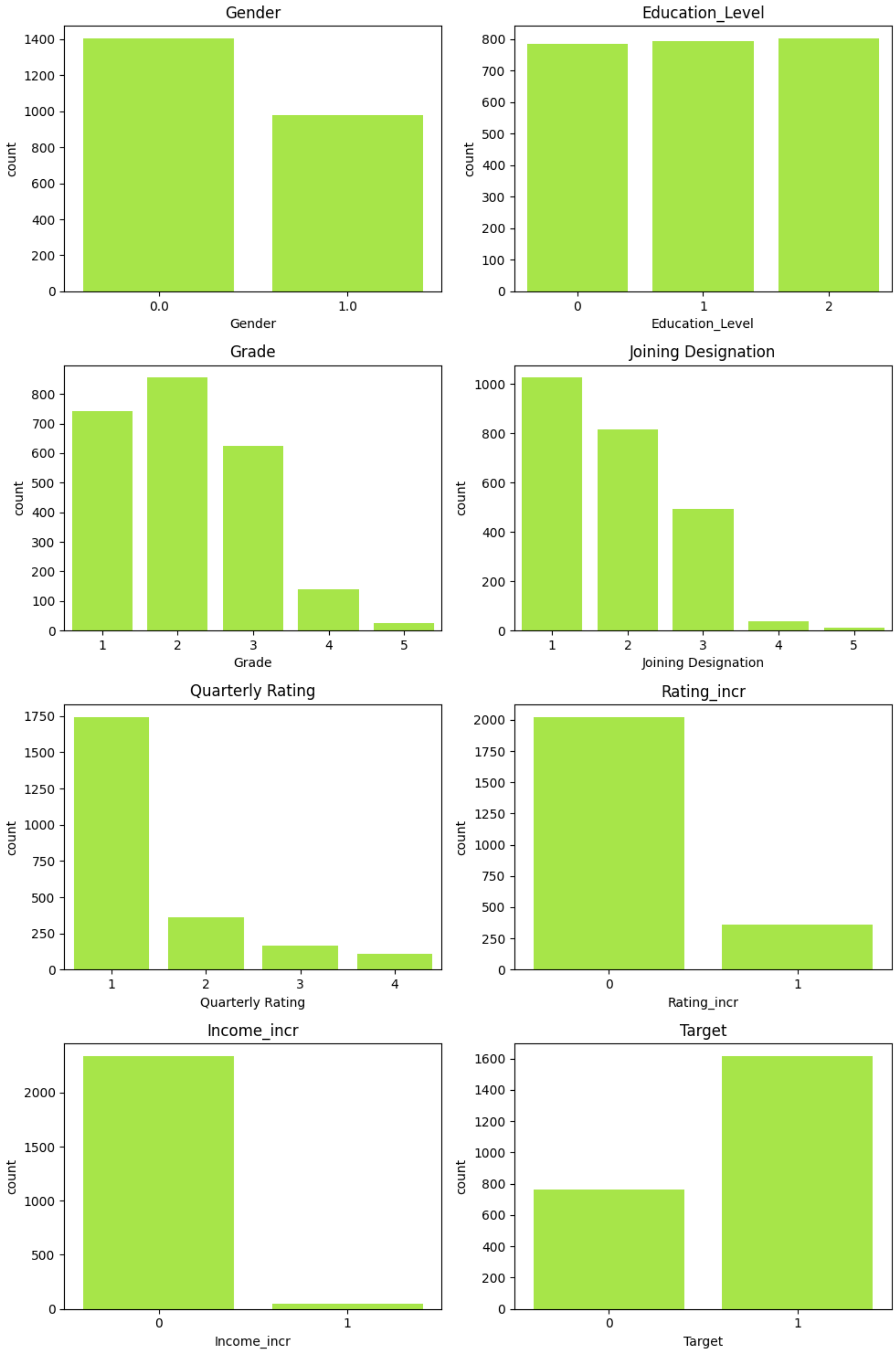
```
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in City column are :- 29
Value counts in City column are :-
 City
C20    0.063839
C15    0.042419
C29    0.040319
C26    0.039059
C8     0.037379
C27    0.037379
C10    0.036119
C16    0.035279
C22    0.034439
C3     0.034439
C28    0.034439
C12    0.034019
C5     0.033599
C1     0.033599
C21    0.033179
C14    0.033179
C6     0.032759
C4     0.032339
C7     0.031919
C9     0.031499
C25    0.031079
C23    0.031079
C24    0.030659
C19    0.030239
C2     0.030239
C17    0.029819
C13    0.029819
C18    0.028979
C11    0.026879
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Education_Level column are :- 3
Value counts in Education_Level column are :-
 Education_Level
2    0.336833
1    0.333893
0    0.329273
```

```
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Grade column are :- 5
Value counts in Grade column are :-
 Grade
2    0.359093
1    0.311214
3    0.261655
4    0.057959
5    0.010080
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Joining Designation column are :- 5
Value counts in Joining Designation column are :-
 Joining Designation
1    0.430911
2    0.342293
3    0.207056
4    0.015120
5    0.004620
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Quarterly Rating column are :- 4
Value counts in Quarterly Rating column are :-
 Quarterly Rating
1    0.732465
2    0.152037
3    0.070559
4    0.044939
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Rating_incr column are :- 2
Value counts in Rating_incr column are :-
 Rating_incr
0    0.849643
1    0.150357
```

```
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Income_incr column are :- 2
Value counts in Income_incr column are :-
 Income_incr
0    0.98194
1    0.01806
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in Target column are :- 2
Value counts in Target column are :-
 Target
1    0.678706
0    0.321294
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------
```

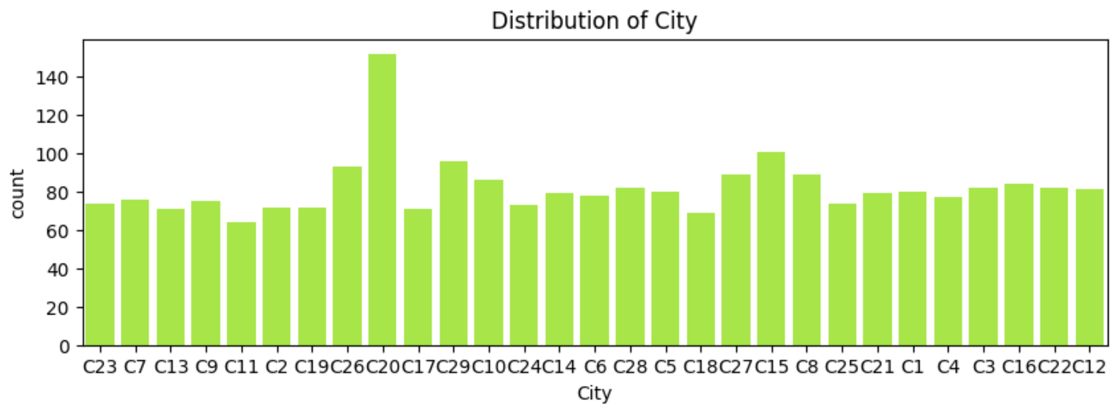
[38]:

```python
newcat_cols=['Gender','Education_Level','Grade','Joining␣
↪Designation','Quarterly Rating','Rating_incr','Income_incr','Target']
plt.figure(figsize=(10,15))
i=1
for col in newcat_cols:
  ax=plt.subplot(4,2,i)
  sns.countplot(x=df2[col],color='greenyellow')
  plt.title(f'{col}')
  i += 1

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

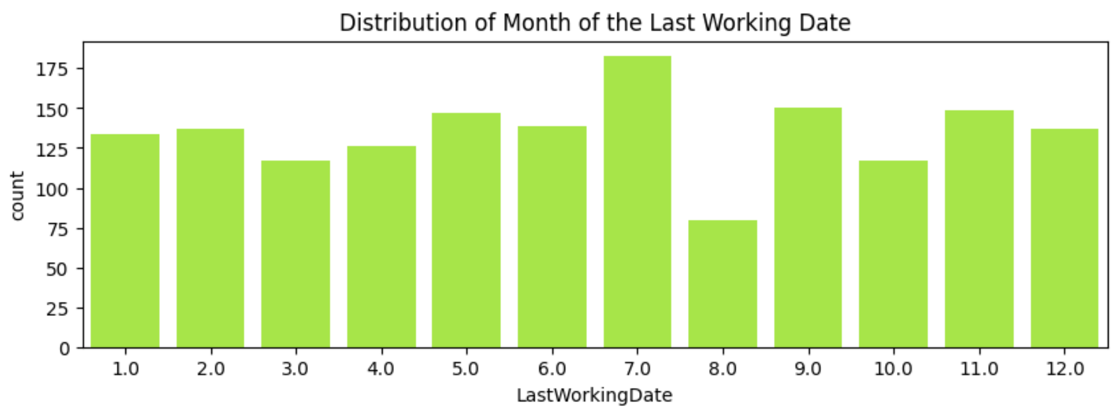

Gender
count
Gender
0.0
1.0
Education_Level
count
Education_Level
Grade
count
Grade
Joining Designation
count
Joining Designation
Quarterly Rating
count
Quarterly Rating
Rating_incr
count
Rating_incr
Income_incr
count
Income_incr
Target
count
Target

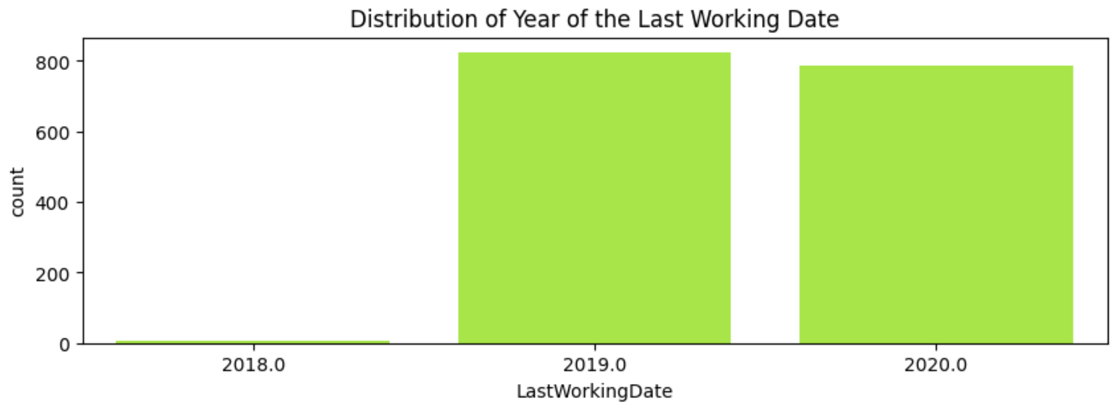
[39]:
```
plt.figure(figsize=(10,3))
sns.countplot(x=df2['City'],color='greenyellow')
plt.title('Distribution of City')
```

[39]: Text(0.5, 1.0, 'Distribution of City')

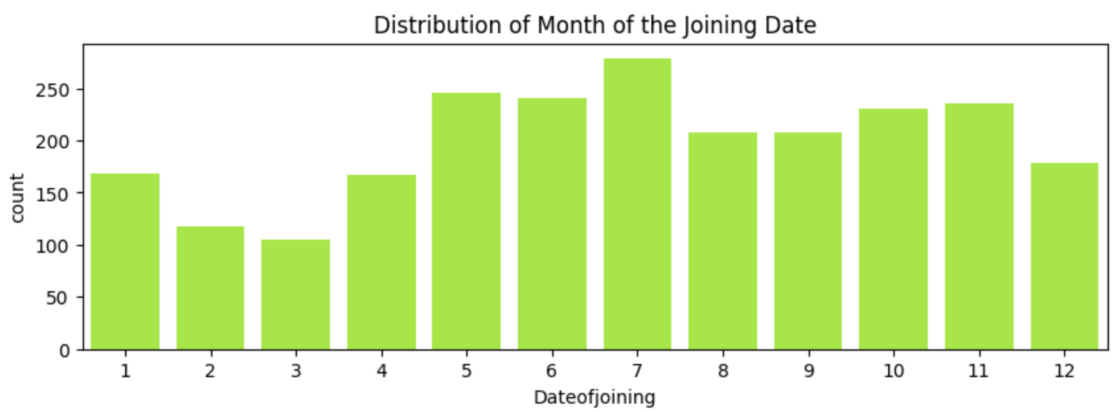


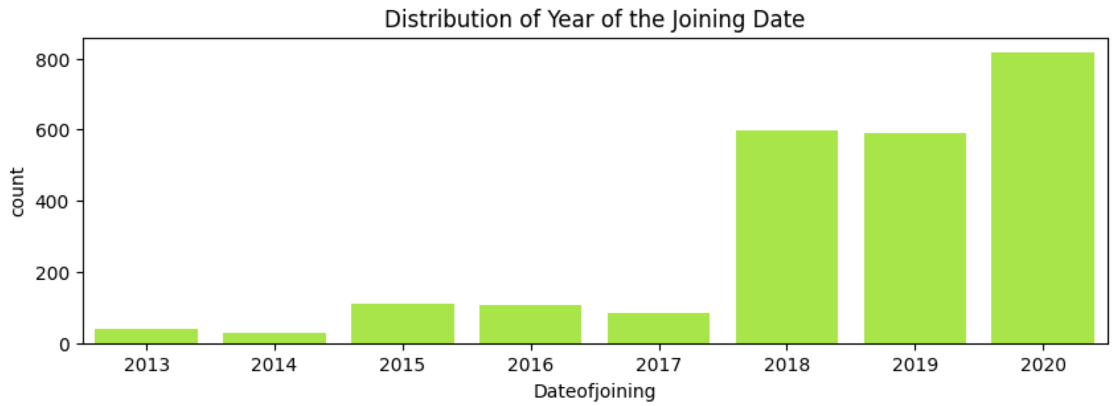
[40]:
```
plt.figure(figsize=(10,3))
sns.countplot(x=df2['LastWorkingDate'].dt.month,color='greenyellow')
plt.title('Distribution of Month of the Last Working Date')

plt.figure(figsize=(10,3))
sns.countplot(x=df2['LastWorkingDate'].dt.year,color='greenyellow')
plt.title('Distribution of Year of the Last Working Date')

plt.show()
```



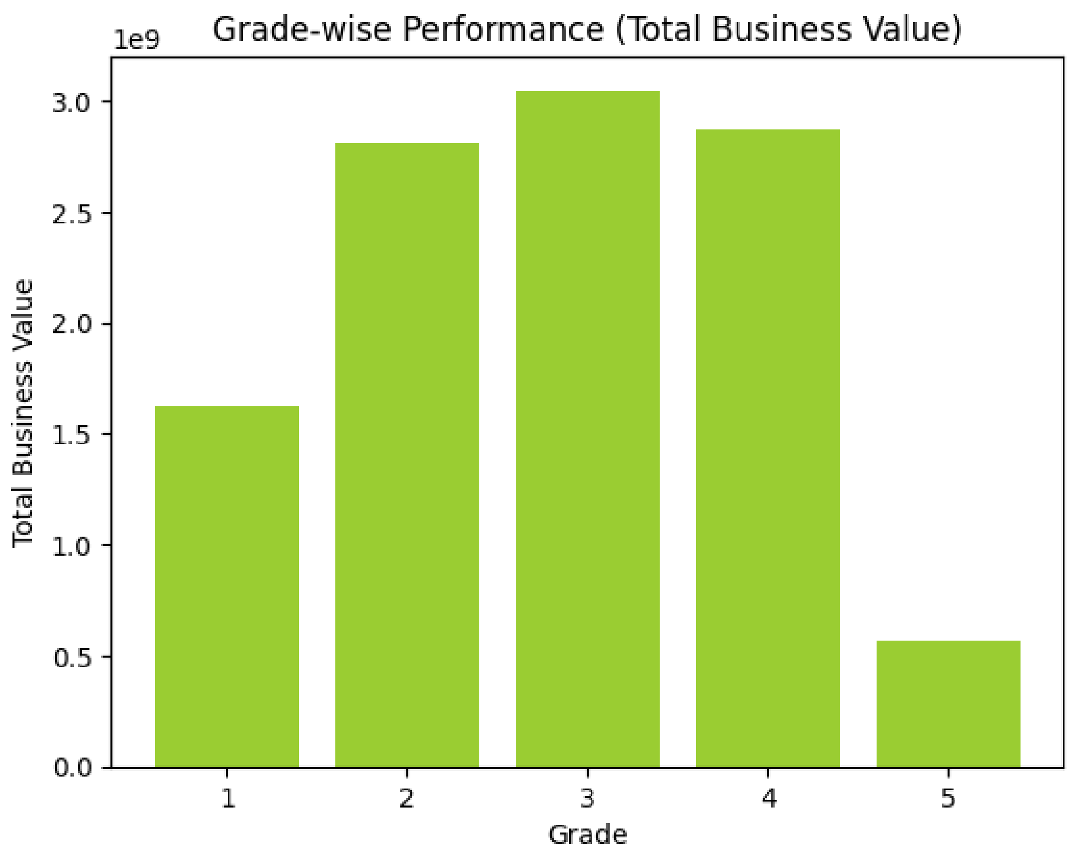


[41]:
```
plt.figure(figsize=(10,3))
sns.countplot(x=df2['Dateofjoining'].dt.month,color='greenyellow')
plt.title('Distribution of Month of the Joining Date')

plt.figure(figsize=(10,3))
sns.countplot(x=df2['Dateofjoining'].dt.year,color='greenyellow')
plt.title('Distribution of Year of the Joining Date')

plt.show()
```





[42]:
```
# Group data by grade and calculate total business value
grade_wise_value = df2.groupby('Grade')['Total Business Value'].sum()

# Create the plot
plt.bar(grade_wise_value.index, grade_wise_value.values,color='yellowgreen')
plt.xlabel('Grade')
plt.ylabel('Total Business Value')
plt.title('Grade-wise Performance (Total Business Value)')
plt.show()
```



City with Most Improvement in Quarterly Rating over the past year

[43]:
```
df4=df1.copy()
```

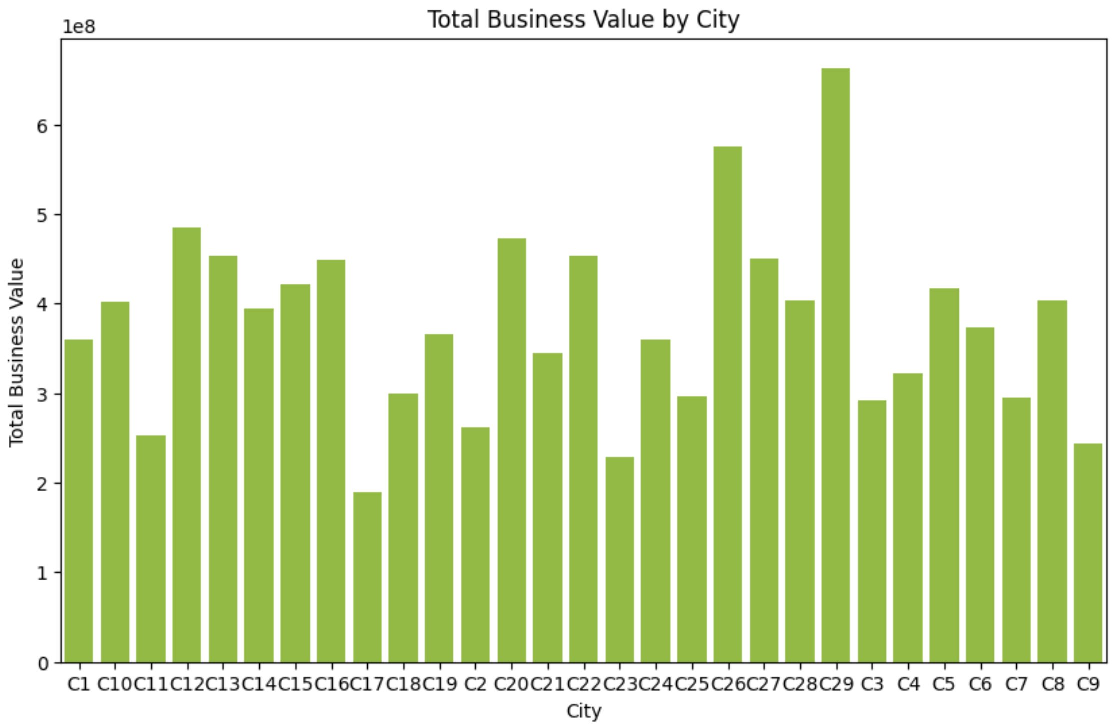
[44]:
```
df4['Reporting_Date'] = pd.to_datetime(df4['Reporting_Date'])

# Use the last date from the dataset as the reference date
last_date = df4['Reporting_Date'].max()
one_year_ago = last_date - pd.DateOffset(years=1)

# Filter data for the past year
df_past_year = df4[df4['Reporting_Date'] >= one_year_ago]

# Check if the DataFrame after filtering is empty
if df_past_year.empty:
    raise ValueError("No data available for the past year. Please check the␣
 ↪date range or the data.")

# Group by city and calculate the change in Quarterly Rating
rating_change = df_past_year.groupby('City').agg(
```

```
    start_rating=('Quarterly Rating', 'first'),
    end_rating=('Quarterly Rating', 'last')
).reset_index()

# Calculate the improvement (change) in Quarterly Rating
rating_change['rating_improvement'] = rating_change['end_rating'] -␣
 ↪rating_change['start_rating']
```

[45]:
```
if rating_change.empty or rating_change['rating_improvement'].isnull().all():
    raise ValueError("No improvements found. Please check the data.")

# Find the city with the greatest improvement
most_improved_city = rating_change.loc[rating_change['rating_improvement'].
 ↪idxmax(), 'City']

print(f'The city with the most improvement in Quarterly Rating over the past␣
 ↪year is: {most_improved_city}')
```

```
The city with the most improvement in Quarterly Rating over the past year is:
C22
```

[46]:
```
# Aggregate total business value by city
city_tbv = df4.groupby('City')['Total Business Value'].sum().reset_index()

# Plot the total business value for each city
plt.figure(figsize=(10, 6))
sns.barplot(data=city_tbv, x='City', y='Total Business Value',␣
 ↪color='yellowgreen')

# Add title and labels
plt.title('Total Business Value by City')
plt.xlabel('City')
plt.ylabel('Total Business Value')
plt.show()
```



Impact of Time of the Year on Quarterly Rating

[47]:
```
df4['Month'] = df4['Reporting_Date'].dt.month
df4['Quarter'] = df4['Reporting_Date'].dt.quarter

# Aggregate Quarterly Ratings by month and quarter
ratings_by_month = df4.groupby('Month')['Quarterly Rating'].mean()
ratings_by_quarter = df4.groupby('Quarter')['Quarterly Rating'].mean()

# Plotting
plt.figure(figsize=(10, 6))

plt.subplot(2, 1, 1)
plt.plot(ratings_by_month, marker='o',color='yellowgreen')
plt.title('Average Quarterly Rating by Month')
plt.xlabel('Month')
plt.ylabel('Average Quarterly Rating')
plt.xticks(range(1, 13))

plt.subplot(2, 1, 2)
plt.plot(ratings_by_quarter, marker='o',color='yellowgreen')
plt.title('Average Quarterly Rating by Quarter')
plt.xlabel('Quarter')
plt.ylabel('Average Quarterly Rating')
```

```
plt.xticks(range(1, 5))

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

Observations:

68% of the Drivers have been churned

Hardly 2% of the Drivers got Increment in Income

15% of the Drivers got Increase in Rating

73% had their last Quarter Rating as 1 followed by 15% having 2

Joining Designation is highest for 1 with 43% followed by 2 with 34%

Grade at the time of Reporting is highest for Grade 2 with 36% followed by Grade 1 with 31%

Distribution of Education Level for all 3 levels is almost same with 33%

C20 is the city with highest number of drivers followed C15

Males are higher in numbers with 59% and Females at 41%

Most of the Drivers had their last working date in the month of July and year 2019

Most of the Drivers joined in the month of July and year 2020

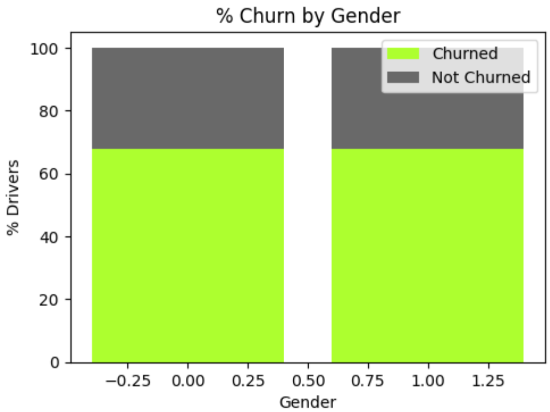
Drivers with Grade 3 have highest business value followed by Grade 4 and 2 The city with the most improvement in Quarterly Rating over the past year is C22

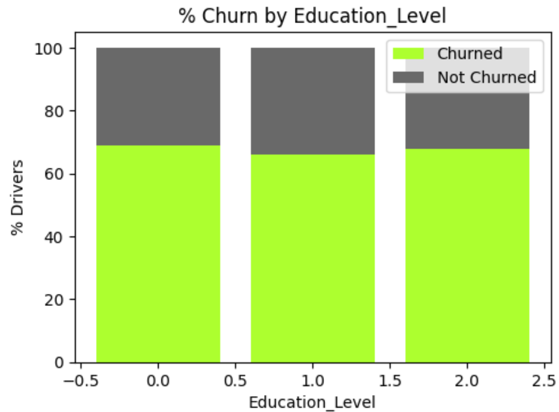
Total Business Value of Drivers is highest in C29 followed by C26 Average Quarterly Rating is found to be highest in 3rd Quarter and the same is found highest in the month of March

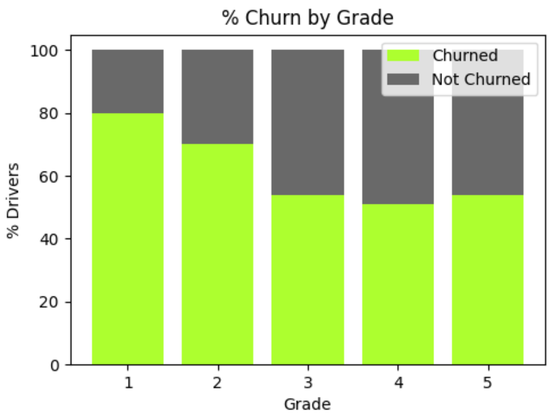
Impact of each feature churn

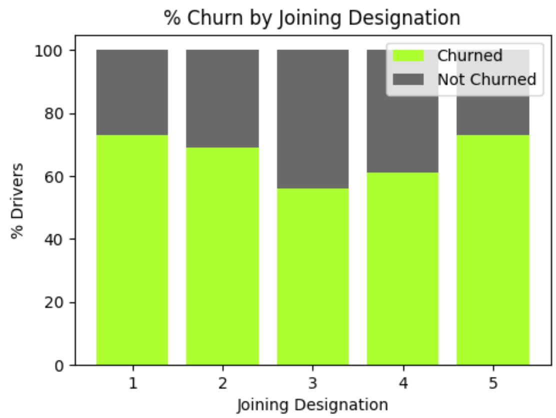
[48]:
```
newcat1_cols=['Gender','Education_Level','Grade','Joining␣
↪Designation','Quarterly Rating','Rating_incr','Income_incr']
```

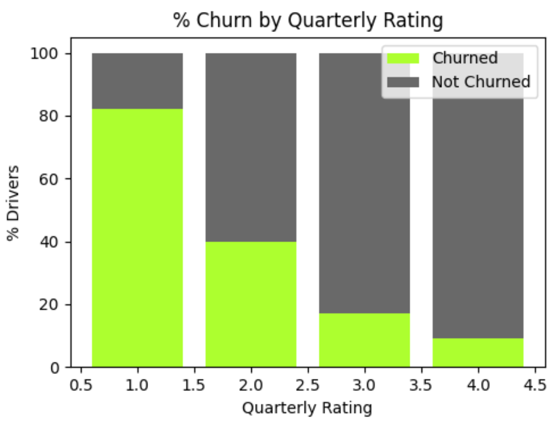
[49]:
```
plt.figure(figsize=(10,15))
i=1
for col in newcat1_cols:
    ax = plt.subplot(4, 2, i)

    data = df2.pivot_table(index=col, columns='Target', aggfunc='size')

    # Convert counts to percentages
    data = data.div(data.sum(axis=1), axis=0).multiply(100).round()
    data.reset_index(inplace=True)

    # Plotting the bars
    plt.bar(data[col], data[1], color='greenyellow',label='Churned')
    plt.bar(data[col], data[0], color='dimgrey', bottom=data[1],label='Not␣
↪Churned')

    # Labeling and titles
    plt.xlabel(f'{col}')
    plt.ylabel('% Drivers')
    plt.title(f'% Churn by {col}')
    plt.legend(['Churned', 'Not Churned'])

    i += 1

# Adjust layout and display the plot
plt.tight_layout()
plt.show()
```

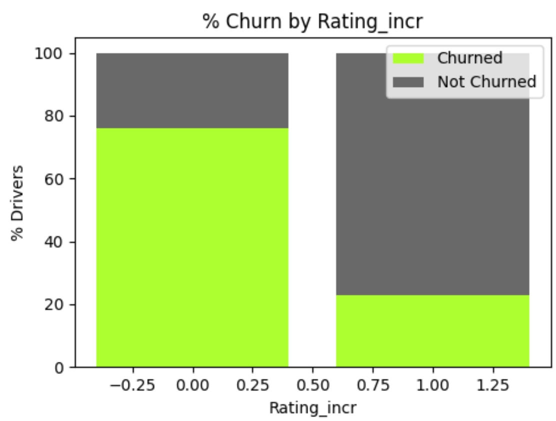

% Churn by Gender
Churned
Not Churned
% Drivers
Gender

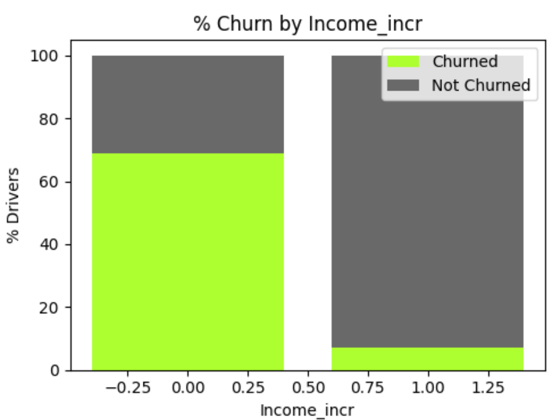

% Churn by Education_Level
Churned
Not Churned
% Drivers
Education_Level


% Churn by Grade
Churned
Not Churned
% Drivers
Grade


% Churn by Joining Designation
Churned
Not Churned
% Drivers
Joining Designation

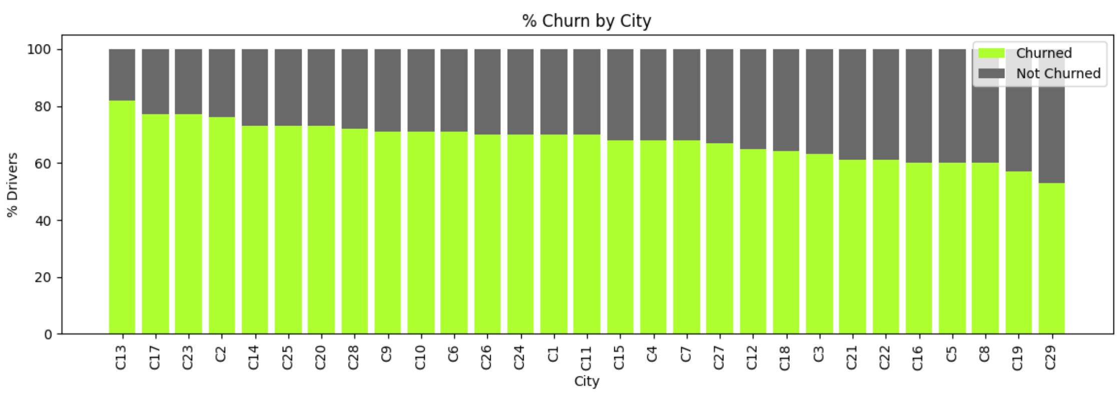

% Churn by Quarterly Rating
Churned
Not Churned
% Drivers
Quarterly Rating


% Churn by Rating_incr
Churned
Not Churned
% Drivers
Rating_incr


% Churn by Income_incr
Churned
Not Churned
% Drivers
Income_incr

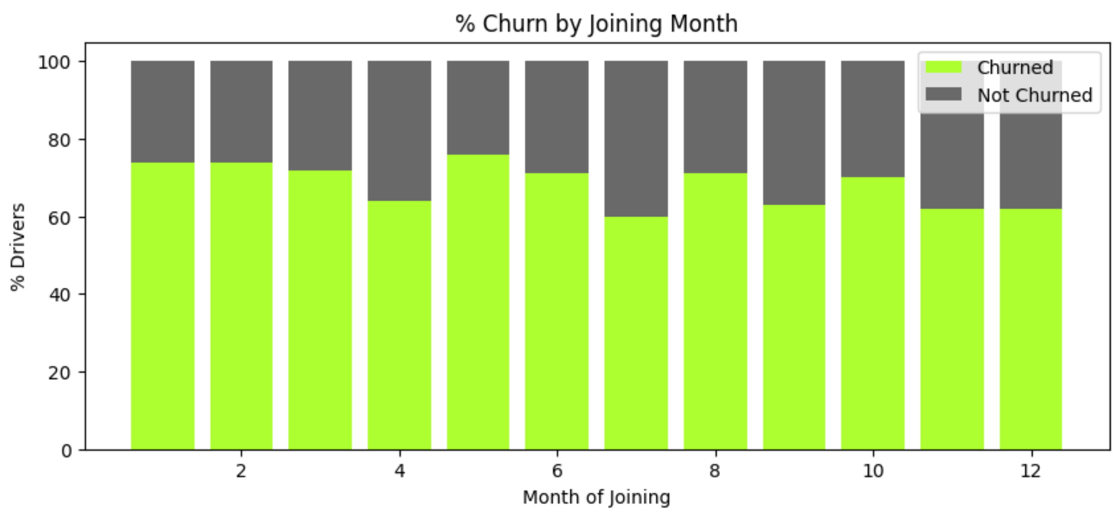
[50]:

```
city = df2.pivot_table(index='City', columns='Target', aggfunc='size')
city = city.div(city.sum(axis=1), axis=0).multiply(100).round()
city.reset_index(inplace=True)
city = city.sort_values(by=1, ascending=False)

plt.figure(figsize=(14,4))
plt.bar(city['City'], city[1], color='greenyellow')
plt.bar(city['City'], city[0], color='dimgrey', bottom=city[1])
# Labeling and titles
plt.xlabel('City')
plt.ylabel('% Drivers')
plt.title(f'% Churn by City')
plt.legend(['Churned', 'Not Churned'])
plt.xticks(rotation=90)
plt.show()
```

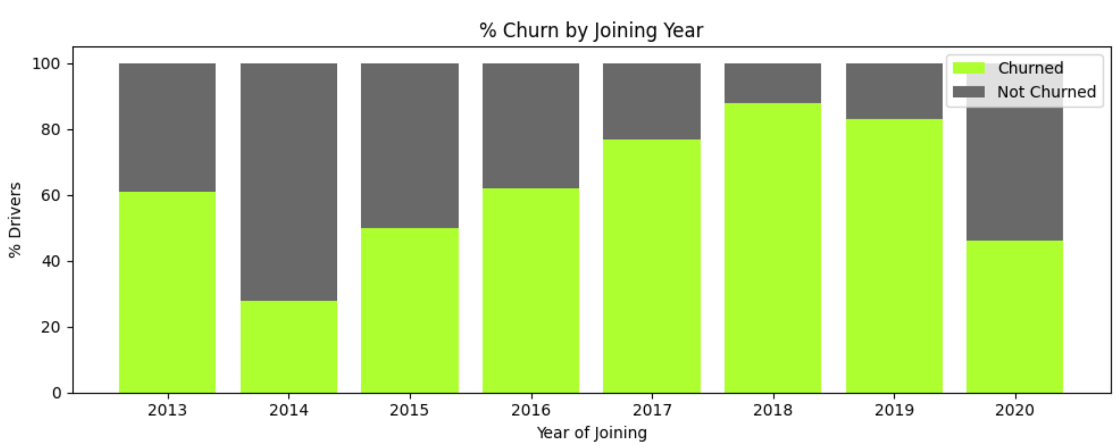


[51]:

```
m = df2.pivot_table(index=df2['Dateofjoining'].dt.month, columns='Target',␣
↪aggfunc='size')
m = m.div(m.sum(axis=1), axis=0).multiply(100).round()
m.reset_index(inplace=True)
plt.figure(figsize=(10,4))
plt.bar(m['Dateofjoining'], m[1], color='greenyellow')
plt.bar(m['Dateofjoining'], m[0], color='dimgrey', bottom=m[1])
# Labeling and titles
plt.xlabel('Month of Joining')
plt.ylabel('% Drivers')
plt.title(f'% Churn by Joining Month')
plt.legend(['Churned', 'Not Churned'])
plt.show()
```



[52]:
```
y = df2.pivot_table(index=df2['Dateofjoining'].dt.year, columns='Target',␣
↪aggfunc='size')
    y = y.div(y.sum(axis=1), axis=0).multiply(100).round()
    y.reset_index(inplace=True)

    plt.figure(figsize=(10,4))
    plt.bar(y['Dateofjoining'], y[1], color='greenyellow')
    plt.bar(y['Dateofjoining'], y[0], color='dimgrey', bottom=y[1])

    # Labeling and titles
    plt.xlabel('Year of Joining')
    plt.ylabel('% Drivers')
    plt.title(f'% Churn by Joining Year')
    plt.legend(['Churned', 'Not Churned'])
    plt.tight_layout()
    plt.show()
```



Observations:

There is no effect of Gender and Education Level on Churn

80% of the Drivers with Grade 1 got churned followed by Grade 2 with almost 70% churn

Drivers with Joining Designation 1 and 5 got churned the most with almost 75%

80% of the Drivers with Quarterly Rating 1 left the company followed by 40% of QR2 and almost 18% of QR3

Almost 77% of the Drivers who did not get any increase in Rating left the company

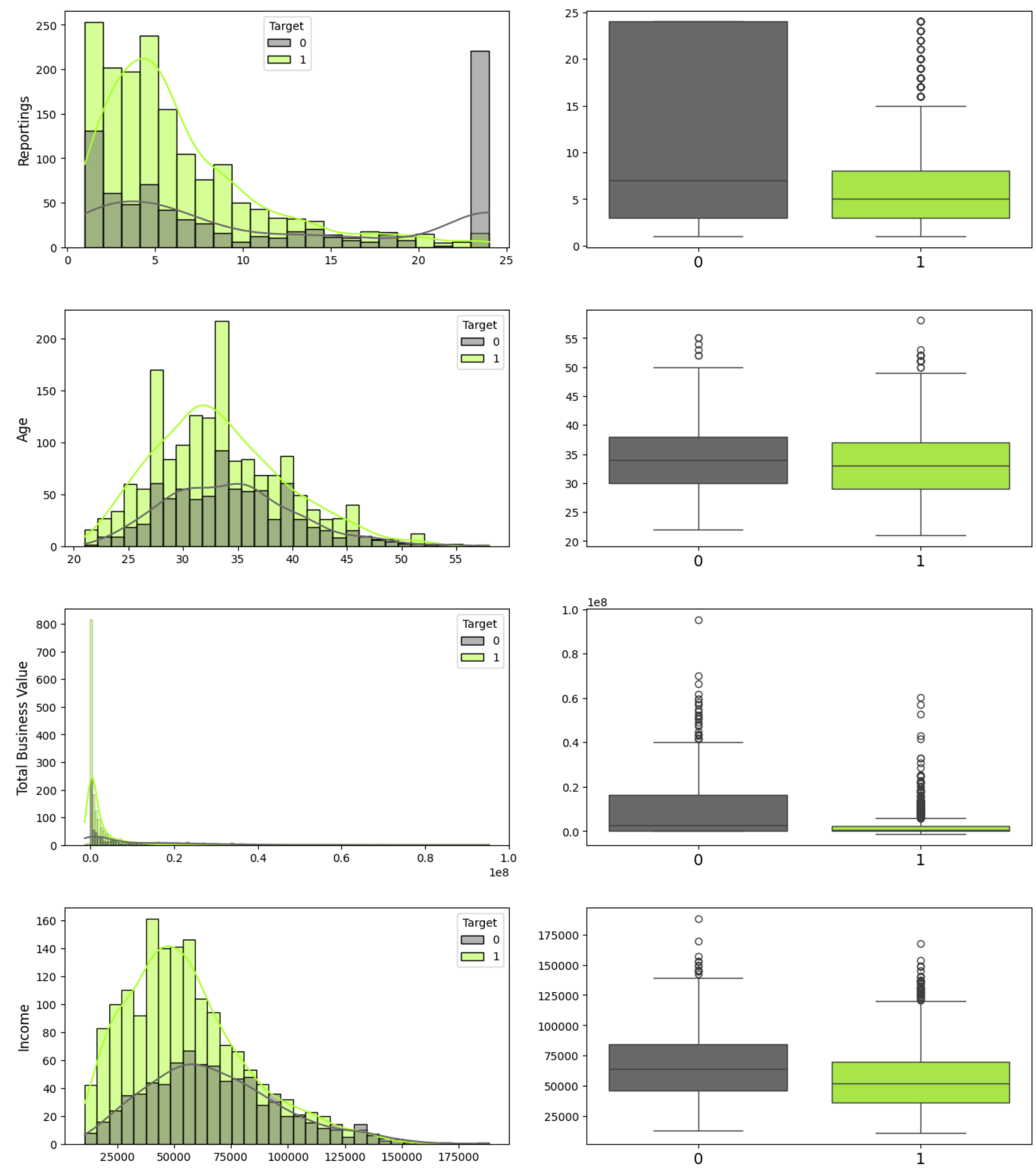
70% of the Drivers who did not get any increment in income left the company

80% of the Drivers from City C13 left the company closely followed by C17 and C23

There is no significant observation on churn w.r.t joining month 90% of the Drivers who joined in the year 2018 left the company followed by 2019 and 2017

[53]:
```
import warnings
import matplotlib.colors as mcolors
import seaborn as sns
```

[54]:
```
warnings.simplefilter(action='ignore', category=FutureWarning)
fig, ax = plt.subplots(4,2,figsize=(13,15))
i=0
color_dict = {0: 'dimgrey', 1: 'greenyellow'}
for col in num_cols:
    sns.boxplot(data=df2, y=col, x='Target', ax=ax[i,1],
               palette=('dimgrey','greenyellow'))
    sns.histplot(data=df2, x=col, hue='Target', ax=ax[i, 0], legend=True,
                palette=color_dict, kde=True, fill=True)
    ax[i,0].set_ylabel(col, fontsize=12)
    ax[i,0].set_xlabel(' ')
    ax[i,1].set_xlabel(' ')
    ax[i,1].set_ylabel(' ')
    ax[i,1].xaxis.set_tick_params(labelsize=14)
    i += 1

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



Observations:

Number of Reportings and Age are relatively lesser for Drivers who left

Most of the Drivers getting churned belong to age between 25-35. Distribution is close to normal

Income is less for the Drivers who left. Distribution is slightly right skewed

Total Business Value is lesser for Drivers who left. Distribution is right skewed

Relationship among feature

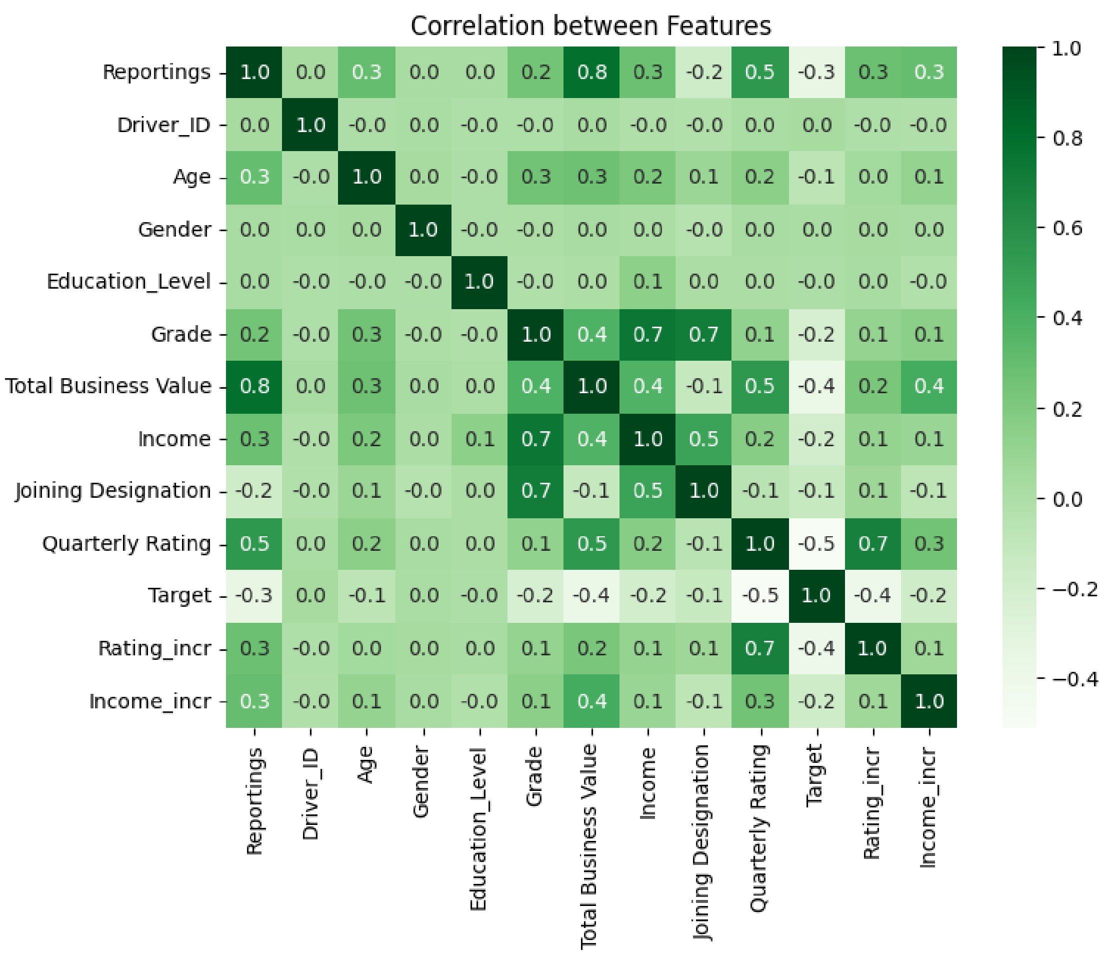
Correlation

OLS Regression Analysis

Hypothesis Testing

[55]:
```
numerical_df2 = df2.select_dtypes(include=['int64', 'float64'])
```

[56]:
```
#Correlation among features
plt.figure(figsize=(8,6))
sns.heatmap(numerical_df2.corr(), annot=True, fmt=".1f",cmap='Greens')
plt.title('Correlation between Features')
plt.show()
```



Highlights:

Reportings is highly positively correlated to Total Business Value

Quarterly Rating and Rating_incr are highly correlated for obvious reasons

Grade is highly positively correlated to Income and Joining Designations

We can consider to drop few of these features basis above observations.

However, multicollinearity can arise due to the combined influence of multiple features, not just pairs.

Setting a single threshold for correlation coefficients to identify features for removal can be arbitrary and might not reflect the true impact on the model.Finally we can conclude this with Feature Importance

Impact of Significant drop in Quarterly Rating over Total Business Value in subsequent period

[57]:
```
import statsmodels.api as sm
# Define a significant drop in Quarterly Rating
significant_drop_threshold = 2  # Example: A drop of 2 or more points

# Calculate the difference in Quarterly Rating between consecutive quarters
df4['Rating_Drop'] = df4.groupby('Driver_ID')['Quarterly Rating'].diff()

# Identify periods with significant drops
df4['Significant_Drop'] = df4['Rating_Drop'] <= -significant_drop_threshold

# Shift Total Business Value to get the subsequent period's value
df4['Subsequent_Business_Value'] = df4.groupby('Driver_ID')['Total Business␣
↪Value'].shift(-1)

# Filter rows with significant drops
significant_drops = df4[df4['Significant_Drop']]

# Prepare data for regression analysis
regression_data = significant_drops[['Rating_Drop',␣
↪'Subsequent_Business_Value']].dropna()

# Add a constant to the independent variable (required for statsmodels)
regression_data = sm.add_constant(regression_data)

# Fit the regression model
model = sm.OLS(regression_data['Subsequent_Business_Value'],␣
↪regression_data[['const', 'Rating_Drop']])
results = model.fit()

# Display the regression results
print(results.summary())

# Interpretation of results
if results.pvalues['Rating_Drop'] < 0.05:
    print("There is a significant impact of rating drops on the subsequent␣
↪period's business value.")
else:
```

```
    print("There is no significant impact of rating drops on the subsequent␣
↪period's business value.")
```

```
                                OLS Regression Results
================================================================================
=====
Dep. Variable:     Subsequent_Business_Value   R-squared:
0.034
Model:                                   OLS   Adj. R-squared:
0.031
Method:                        Least Squares   F-statistic:
10.04
Date:                       Sat, 25 Jan 2025   Prob (F-statistic):
0.00170
Time:                               18:28:23   Log-Likelihood:
-4078.8
No. Observations:                        284   AIC:
8162.
Df Residuals:                            282   BIC:
8169.
Df Model:                                  1
Covariance Type:                   nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        7.282e+05   1.49e+05      4.881      0.000    4.35e+05    1.02e+06
Rating_Drop  2.159e+05   6.81e+04      3.169      0.002    8.18e+04     3.5e+05
==============================================================================
Omnibus:                      152.943   Durbin-Watson:                   1.779
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1165.326
Skew:                           2.072   Prob(JB):                    8.97e-254
Kurtosis:                      12.017   Cond. No.                         15.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
There is a significant impact of rating drops on the subsequent period's
business value.
```

Which all Features have impact on Quarterly Rating

[58]:
```
df5=df4.copy()
numerical_df = df5.select_dtypes(include=['int64', 'float64'])

# Remove non-relevant columns
exclude_columns = ['Reporting_Date', 'Dateofjoining', 'LastWorkingDate']
numerical_df = numerical_df.drop(columns=exclude_columns, errors='ignore')
```

```python
# Drop rows with missing values
numerical_df.dropna(inplace=True)

# Separate the target variable and features
X = numerical_df.drop('Quarterly Rating', axis=1)
y = numerical_df['Quarterly Rating']

# Add a constant to the feature matrix (required for statsmodels)
X = sm.add_constant(X)

# Fit the regression model using statsmodels
model = sm.OLS(y, X).fit()

# Print the summary of the regression model
print(model.summary())

# Extract p-values from the model summary
p_values = model.pvalues

# Filter features with p-value less than 0.05
significant_features = p_values[p_values < 0.05].index.tolist()

# Remove the constant term if it's included in the significant features
if 'const' in significant_features:
    significant_features.remove('const')

print("Significant numerical features impacting Quarterly Rating:")
for feature in significant_features:
    print(feature)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Quarterly Rating   R-squared:                       0.410
Model:                            OLS   Adj. R-squared:                  0.409
Method:                 Least Squares   F-statistic:                     1002.
Date:                Sat, 25 Jan 2025   Prob (F-statistic):               0.00
Time:                        18:28:23   Log-Likelihood:                -16709.
No. Observations:               14441   AIC:                         3.344e+04
Df Residuals:                   14430   BIC:                         3.352e+04
Df Model:                          10
Covariance Type:            nonrobust
================================================================================
=============
                                coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
```

```
-------------
const                          1.7415      0.042     41.820      0.000
1.660       1.823
Driver_ID                   1.332e-05   7.92e-06      1.683      0.092
-2.2e-06    2.88e-05
Age                            0.0158      0.001     15.034      0.000
0.014       0.018
Gender                        -0.0269      0.013     -2.068      0.039
-0.052      -0.001
Education_Level                0.0141      0.008      1.693      0.090
-0.002       0.030
Income                      4.075e-06   3.43e-07     11.869      0.000
3.4e-06    4.75e-06
Joining Designation           -0.2161      0.009    -22.865      0.000
-0.235      -0.198
Grade                         -0.1829      0.011    -16.104      0.000
-0.205      -0.161
Total Business Value        3.089e-07   5.88e-09     52.519      0.000
2.97e-07     3.2e-07
Rating_Drop                    0.4291      0.011     37.405      0.000
0.407       0.452
Subsequent_Business_Value  2.412e-07   5.56e-09     43.375      0.000
2.3e-07    2.52e-07
==============================================================================
Omnibus:                      241.154   Durbin-Watson:                   0.614
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              373.595
Skew:                           0.174   Prob(JB):                     7.50e-82
Kurtosis:                       3.707   Cond. No.                     1.05e+07
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.05e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
Significant numerical features impacting Quarterly Rating:
Age
Gender
Income
Joining Designation
Grade
Total Business Value
Rating_Drop
Subsequent_Business_Value
```

Hypothesis Testing

```
[59]: from scipy.stats import chi2_contingency
newcat2_cols=['Reportings','Gender','City','Education_Level','Grade','Joining␣
↪Designation','Quarterly Rating','Rating_incr','Income_incr']
for col in newcat2_cols:
    chi2, p, dof, expected = chi2_contingency(pd.crosstab(df3[col],␣
↪df3['Target']))
    if p > 0.05:
        print('>>>>>>> Independent feature - Not Significant:',col,' >> p value:
↪',p)
```

```
>>>>>>> Independent feature - Not Significant: Gender  >> p value:
0.6943902798506425
>>>>>>> Independent feature - Not Significant: Education_Level  >> p value:
0.46643939521309963
```

Based on Hypothesis Testing and as observed in our Graphical Impact analysis of Churn on Gender and Education_Level, we found same observation that these features are not significant for determining Churn.

However, we shall not remove these features now as this might miss complex non-linear relationships or interactions between multiple features that could be crucial for the model. Which we shall learn in Feature Importance

Data Preparation for Modeling Encoding

Scaling

Train Test Split

Class Imbalance- SMOTE

```
[60]: df_prep=df2.drop(columns=['Driver_ID','LastWorkingDate'],axis=1)
df_prep['Month']=df_prep['Dateofjoining'].dt.month
df_prep['Year']=df_prep['Dateofjoining'].dt.year
df_prep.drop('Dateofjoining',axis=1,inplace=True)
```

```
[61]: df_encoded = pd.get_dummies(df_prep,'City', drop_first=True)*1
df_encoded.head()
```

```
[61]:    Reportings   Age  Gender  Education_Level  Grade  Total Business Value  \
0           3  28.0     0.0                2      1               1715580
1           2  31.0     0.0                2      2                     0
2           5  43.0     0.0                2      2                350000
3           3  29.0     0.0                0      1                120360
4           5  31.0     1.0                1      3               1265000

   Income  Joining Designation  Quarterly Rating  Target  …  City_C27  \
0   57387                    1                 2       1  …         0
1   67016                    2                 1       0  …         0
2   65603                    2                 1       1  …         0
```

```
3   46368                        1                  1       1  …          0
4   78728                        3                  2       0  …          0

   City_C28  City_C29  City_C3  City_C4  City_C5  City_C6  City_C7  City_C8  \
0         0         0        0        0        0        0        0        0
1         0         0        0        0        0        0        1        0
2         0         0        0        0        0        0        0        0
3         0         0        0        0        0        0        0        0
4         0         0        0        0        0        0        0        0

   City_C9
0        0
1        0
2        0
3        1
4        0

[5 rows x 42 columns]
```

[62]:
```python
df_encoded.shape
```

[62]:
```
(2381, 42)
```

Train Test Split

[63]:
```python
from sklearn.model_selection import train_test_split
#Prepare X and y dataset i.e. independent and dependent datasets

X = df_encoded.drop(['Target'], axis=1)
y = df_encoded['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)
```

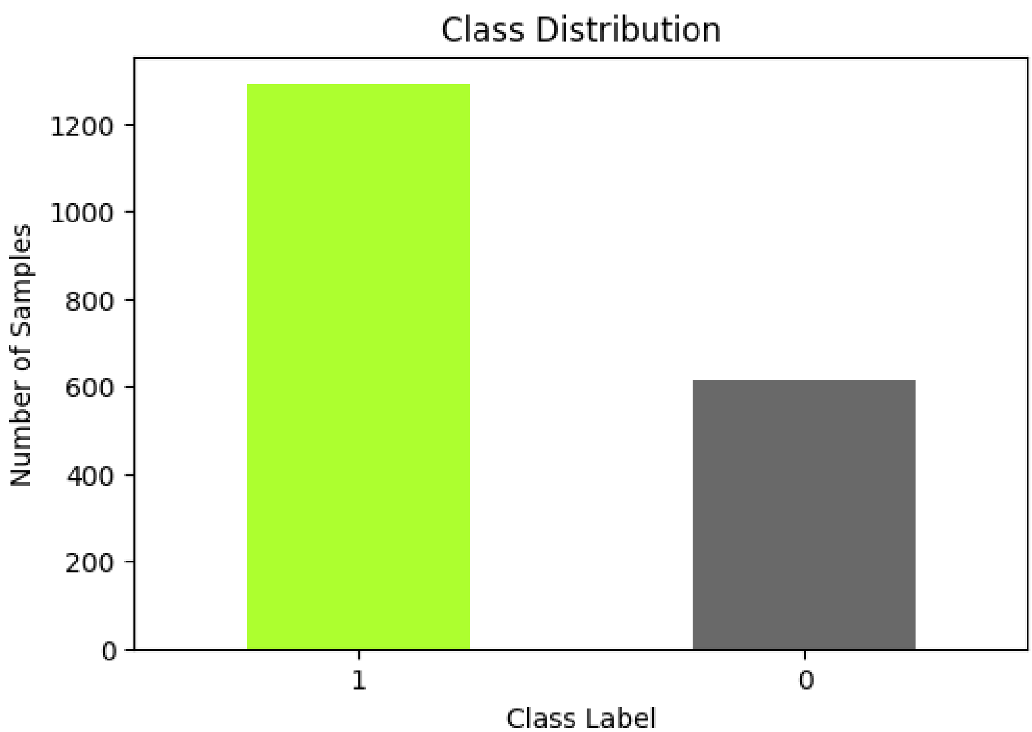
Scaling

[64]:
```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X.columns)
```

Check class Imbalance

[65]:
```python
# Count class frequencies
class_counts = y_train.value_counts()

# Create a bar chart
```

```
plt.figure(figsize=(6, 4))
class_counts.plot(kind='bar', color=['greenyellow', 'dimgrey'])
plt.xlabel('Class Label')
plt.ylabel('Number of Samples')
plt.title('Class Distribution')
plt.xticks(rotation=0)  # Rotate x-axis labels for better readability
plt.show()

# Print class ratio (optional)
print(f"Class Ratio (Majority / Minority): {class_counts.iloc[0] / class_counts.
↪iloc[1]:.2f}")
```



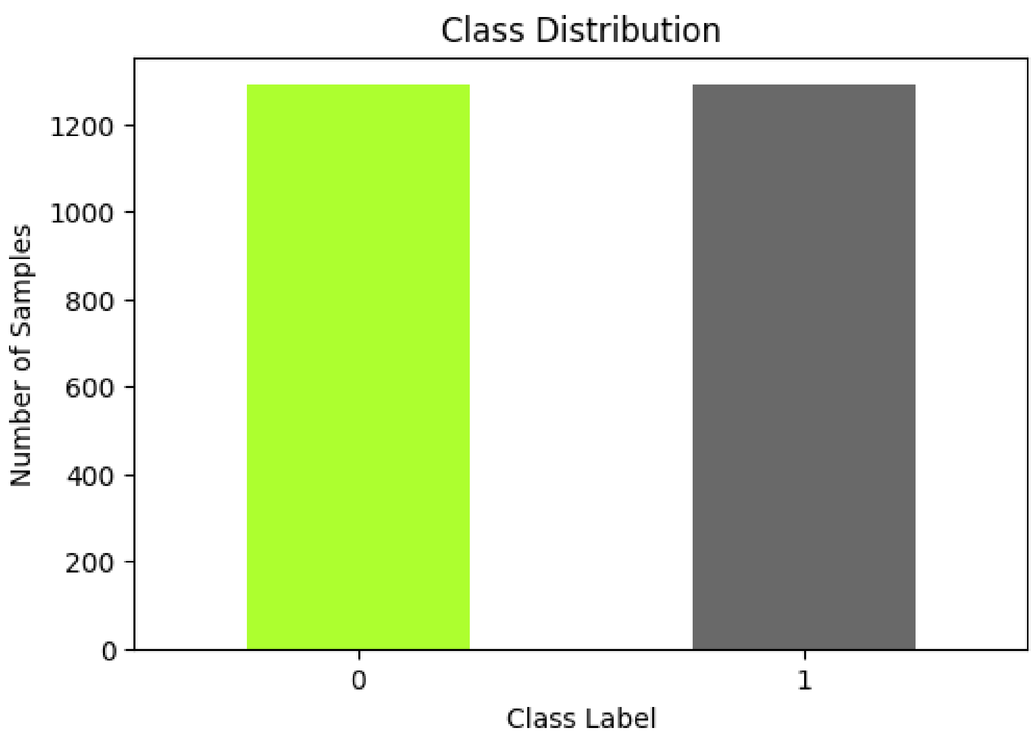

```
Class Ratio (Majority / Minority): 2.10
```

SMOTE

Synthetic Minority Over-sampling Technique) is often used to handle imbalanced datasets, especially when the target variable has significantly fewer instances of one class compared to the other. If our binary classification problem has an imbalanced target variable, applying SMOTE can help improve model performance by generating synthetic samples of the minority class.

[66]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train_scaled, y_train)
```

```python
# Count class frequencies
class_counts = y_train_res.value_counts()
# Create a bar chart
plt.figure(figsize=(6, 4))
class_counts.plot(kind='bar', color=['greenyellow', 'dimgrey'])
plt.xlabel('Class Label')
plt.ylabel('Number of Samples')
plt.title('Class Distribution')
plt.xticks(rotation=0)  # Rotate x-axis labels for better readability
plt.show()
# Print class ratio (optional)
print(f"Class Ratio (Majority / Minority): {class_counts.iloc[0] / class_counts.
 ↪iloc[1]:.2f}")
```




```
Class Ratio (Majority / Minority): 1.00
```

Ensemble Learning: Bagging (Random Forest Classifier) Hyperparameter Tuning using Gridsearch CV

Model Score / Accuracy Measurement

Confusion Matrix

Feature Importance

ROC Curve & AUC

Precision Recall Curve

[67]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
import time
```

[68]:
```
params = {"max_depth":[7,10,15],
             "n_estimators":[100,200,300,400],
             "max_features":[4,7,10],
             "ccp_alpha":[0.0005,0.00075,0.001]}
```

[69]:
```
grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42),␣
 ↪param_grid=params, cv=5, n_jobs=-1, verbose=2)

# Measure the time taken to fit the model
start_time = time.time()
grid_search.fit(X_train_res, y_train_res)
end_time = time.time()

print("Best parameters found by GridSearchCV:", grid_search.best_params_)
print(f"Total training time: {end_time - start_time:.2f} seconds")
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best parameters found by GridSearchCV: {'ccp_alpha': 0.0005, 'max_depth': 15,
'max_features': 7, 'n_estimators': 100}
Total training time: 485.73 seconds
```

[70]:
```
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, ConfusionMatrixDisplay
# Retrieve the best model (estimator)
best_model = grid_search.best_estimator_

# Make predictions on the test set
y_train_pred = best_model.predict(X_train_res)
y_test_pred = best_model.predict(X_test_scaled)

# Evaluate the model
# Accuracy
train_accuracy = accuracy_score(y_train_res, y_train_pred)
print(f"Training Accuracy: {train_accuracy:.2f}")

test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Test Accuracy: {test_accuracy:.2f}")
```

```
Training Accuracy: 0.98
Test Accuracy: 0.89
```

[71]:
```
grid_search.best_score_
```

[71]: 0.9150666064574395

Observations:

Training Accuracy: 0.98: This denotes that during the training phase, the Random Forest model achieved an accuracy of 98% on the training data. This high training accuracy suggests that the model was able to fit the training data quite well.

Test Accuracy: 0.89: After training, when the model was evaluated on unseen or test data, it achieved an accuracy of 89%. This accuracy represents how well the model generalizes to new, unseen data. An accuracy of 89% suggests that the model performs well on the test data, although it's slightly lower than the training accuracy, which is expected.

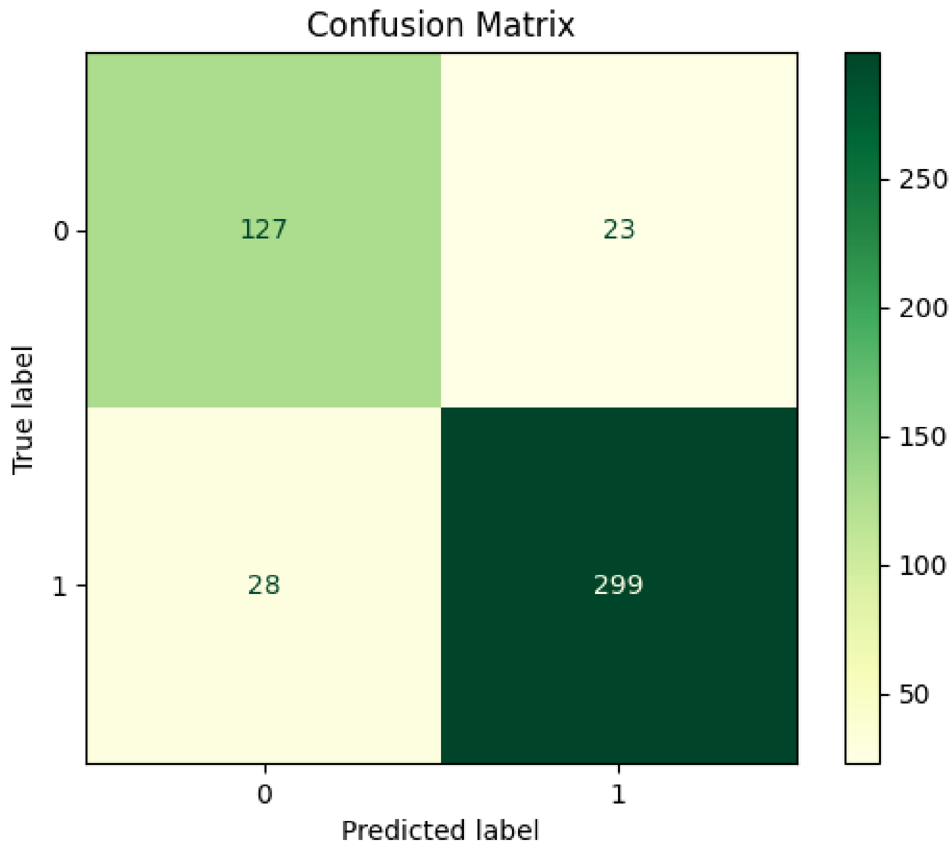
Model Best Score is 0.915: This score likely refers to the best cross-validated score achieved during the hyperparameter tuning process. The score of 0.915 suggests that the model achieved a high performance metric (such as accuracy, F1-score, etc.) during cross-validation with the best set of hyperparameters found by GridSearchCV.

Confusion Matrix / Classification Report

[72]:
```
conf_matrix = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[127  23]
 [ 28 299]]
```

[73]:
```
disp = ConfusionMatrixDisplay(conf_matrix)
cmap = plt.cm.YlGn
disp.plot(cmap=cmap)
plt.title('Confusion Matrix')
plt.show()
print(classification_report(y_test, y_test_pred))
```



	precision	recall	f1-score	support
0	0.82	0.85	0.83	150
1	0.93	0.91	0.92	327
accuracy			0.89	477
macro avg	0.87	0.88	0.88	477
weighted avg	0.89	0.89	0.89	477

Observations:

Precision: Precision is the ratio of correctly predicted positive observations to the total predicted positives. For class 0, the precision is 0.82, and for class 1, it is 0.93. This means that when the model predicts class 0, it is correct 82% of the time, and when it predicts class 1, it is correct 93% of the time.

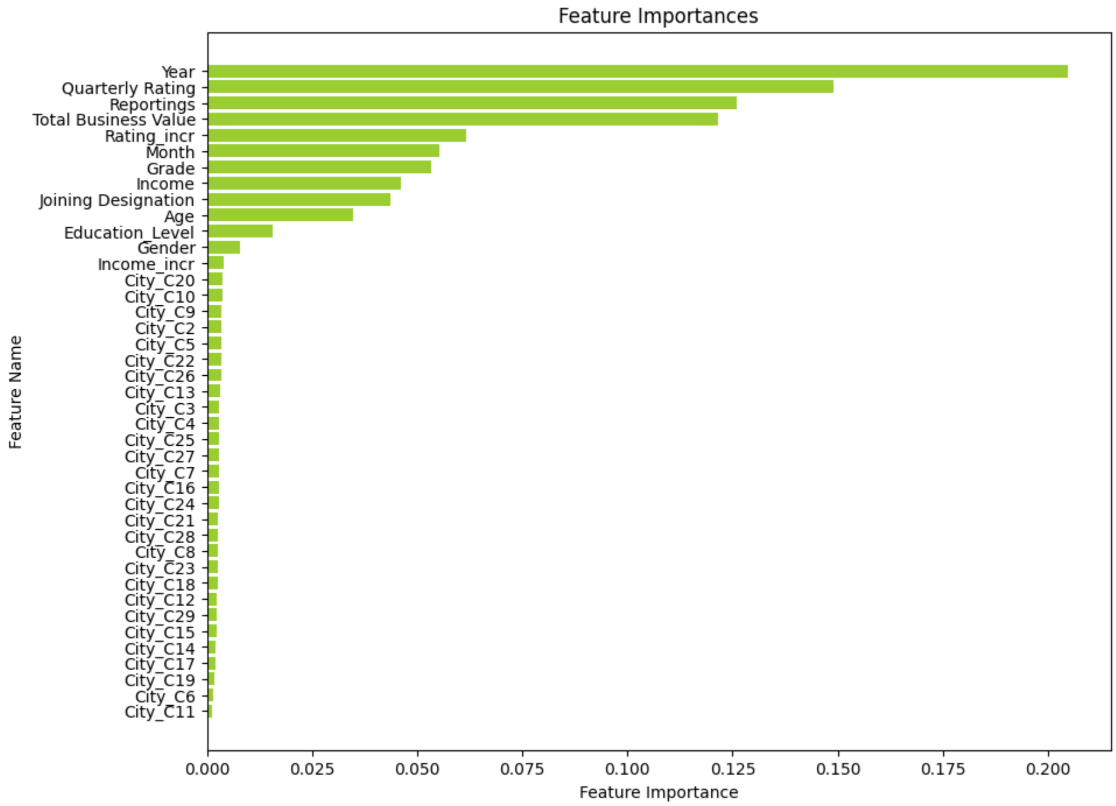
Recall (Sensitivity): Recall is the ratio of correctly predicted positive observations to the all observations in actual class. For class 0, the recall is 0.85, and for class 1, it is 0.91. This implies that the model is able to capture 85% of the actual class 0 instances and 91% of the actual class 1 instances.

F1-score: F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall. For class 0, the F1-score is 0.83, and for class 1, it is 0.92. The weighted average of these scores is 0.89.

Support: Support is the number of actual occurrences of the class in the specified dataset. For class 0, the support is 150, and for class 1, it is 327.

Accuracy: Accuracy is the ratio of correctly predicted observations to the total observations. In this case, the overall accuracy of the model on the test data is 0.89, meaning it correctly predicts the class for 89% of the samples.

[74]:
```
feature_importances = best_model.feature_importances_

# Assuming X_train_res is your training data
# Assuming column_names is a list containing the names of your features
# You may obtain column_names from your DataFrame if you used one initially

# Create a dictionary to store feature names and their importances
feature_importance_dict = dict(zip(X_train_res.columns, feature_importances))

# Sort the dictionary by importance values in descending order
sorted_feature_importance = sorted(feature_importance_dict.items(), key=lambda␣
↪x: x[1], reverse=True)

# Extract feature names and importances
sorted_feature_names = [x[0] for x in sorted_feature_importance]
sorted_importances = [x[1] for x in sorted_feature_importance]

# Plot feature importances
plt.figure(figsize=(10, 8))
plt.barh(sorted_feature_names, sorted_importances,color='yellowgreen')
plt.xlabel('Feature Importance')
plt.ylabel('Feature Name')
plt.title('Feature Importances')
plt.gca().invert_yaxis()  # Invert y-axis to show highest importance at the top
plt.show()
```



Feature Importance in case of RandomForestClassifier:

Year is the most important feature in determining Churn followed by Quarterly Ratings, Reportings and Business Values Least important is City, Income increment followed by Education Level and Age. Our initial EDA too inferred that Age and Education Level are not significant in determining Churn

ROC Curve & AUC

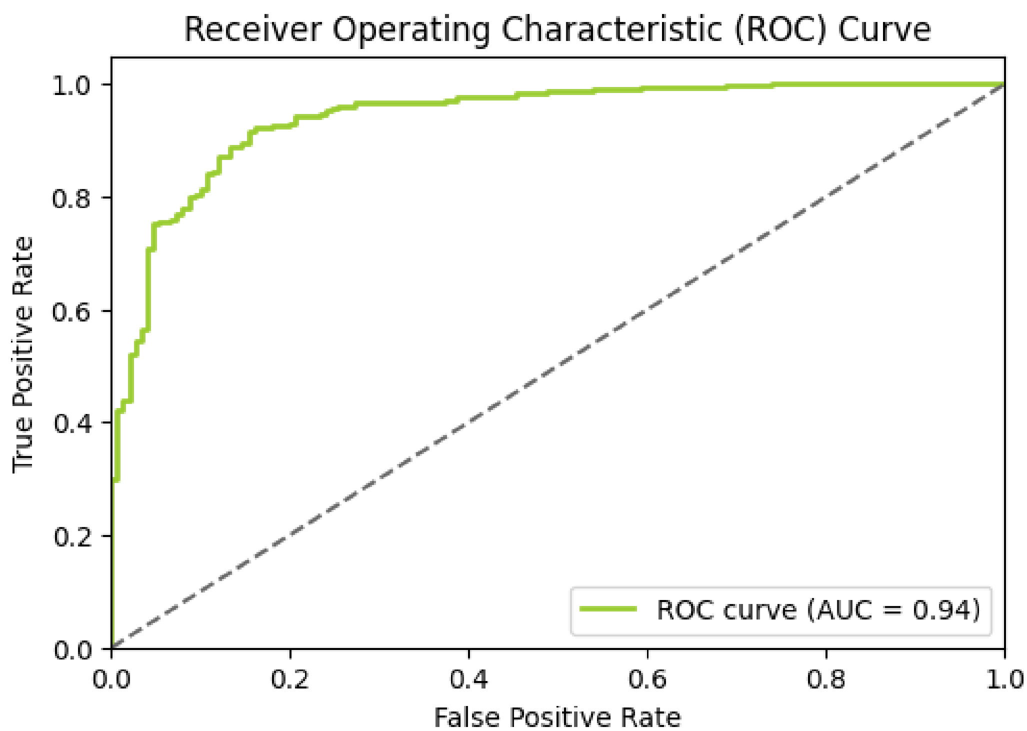
The Receiver Operating Characteristic (ROC) curve is a graphical representation of the performance of a binary classification model. It helps evaluate and compare different models by illustrating the trade-off between the true positive rate (TPR) and false positive rate (FPR) at various classification thresholds.

The area under the ROC curve (AUC) is a commonly used metric to quantify the overall performance of a classifier.

A perfect classifier would have an AUC of 1, while a random classifier would have an AUC of 0.5. The higher the AUC value, the better the classifier's performance in distinguishing between positive and negative instances.

[75]:
```
from sklearn.metrics import roc_curve, roc_auc_score
```

```
# Make predictions on the test set
y_pred_proba = best_model.predict_proba(X_test_scaled)[:, 1]

# Compute ROC curve and ROC-AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Plot ROC curve
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='yellowgreen', lw=2, label='ROC curve (AUC = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='dimgrey', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



Performance Interpretation:

An AUC of 0.94 means that there is a 94% chance that the model will correctly distinguish between a randomly chosen positive instance and a randomly chosen negative instance. High Discrimination

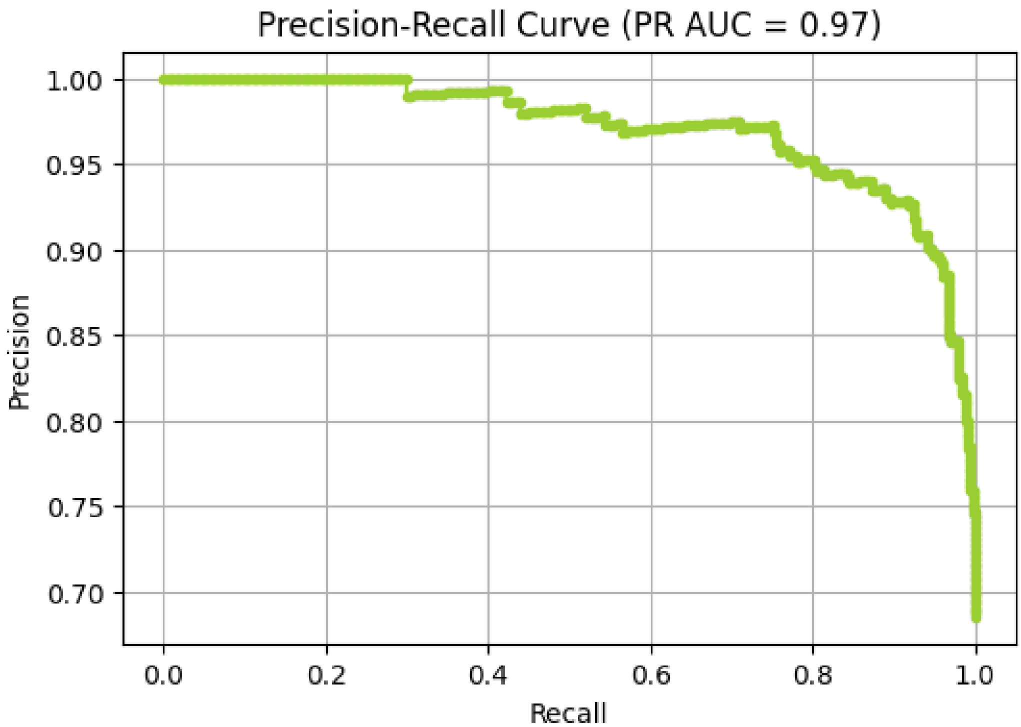
Ability: The model has a strong ability to discriminate between the positive and negative classes. Practical Implications:

Model Reliability: An AUC of 0.94 suggests that the model is very reliable for making predictions and has a low likelihood of making incorrect classifications. Threshold Selection: The high AUC indicates that the model will perform well across a range of threshold settings, providing flexibility in choosing a threshold that balances sensitivity and specificity according to specific requirements.

Precision Recall Curve

The Precision-Recall (PR) curve is another graphical representation commonly used to evaluate the performance of a binary classification model. It provides insights into the trade-off between precision and recall at various classification thresholds.

[76]:
```
from sklearn.metrics import precision_recall_curve,auc
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_proba)
pr_auc = auc(recall, precision)

# Plot the precision-recall curve
plt.figure(figsize=(6, 4))
plt.plot(recall, precision, marker='.',color='yellowgreen')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (PR AUC = {:.2f})'.format(pr_auc))
plt.grid(True)

# Annotate the PR AUC value on the plot
#plt.text(0.7, 0.2, 'PR AUC = {:.2f}'.format(pr_auc), fontsize=12)

plt.show()
```



High PR AUC:

A PR AUC of 0.97 is very high, indicating that the model has both high precision and high recall across different thresholds. This means the model is very good at identifying positive instances without producing many false positives. Model Performance:

High Precision: The model makes very few false positive errors, meaning that most of the positive predictions are correct.

High Recall: The model successfully identifies a large proportion of actual positive instances, missing very few.

Context of Imbalanced Datasets:

PR AUC is particularly informative when dealing with imbalanced datasets. In such scenarios, traditional metrics like accuracy can be misleading because they may be dominated by the majority class.

The PR AUC provides a clearer picture of how well the model is performing with respect to the minority class (often the more important class in imbalanced datasets).

Ensemble Learning: Boosting (LightGBM)

Hyperparameter Tuning using GridsearchCV

Model Score / Accuracy Measurement

Confusion Matrix

Feature Importance

ROC Curve & AUC

Precision Recall Curve

[78]:
```
import lightgbm as lgb
import sys
import os
import time
from sklearn.model_selection import GridSearchCV

# Dummy file to suppress warnings
class DummyFile(object):
    def write(self, x):
        pass

# Redirect stderr to dummy file
sys.stderr = DummyFile()

# Suppress LightGBM and other warnings
os.environ['PYTHONWARNINGS'] = 'ignore'

# Reset stderr
sys.stderr = sys.__stderr__

# Reset PYTHONWARNINGS
del os.environ['PYTHONWARNINGS']

# Configure LightGBM to suppress warnings
model = lgb.LGBMClassifier(silent=True, verbose=-1)

# Define the grid of parameters to search
gridParams = {
    'learning_rate': [0.1, 0.3, 0.5],
    'boosting_type': ['gbdt'],
    'objective': ['binary'],
    'max_depth': [5, 6, 7, 8],
    'colsample_bytree': [0.5, 0.7],
    'subsample': [0.5, 0.7]
}

# Setup GridSearchCV
grid = GridSearchCV(estimator=model, param_grid=gridParams, cv=3,␣
↪scoring='neg_log_loss', verbose=0)  # Updated verbose

# Perform the grid search
start_time = time.time()
```

```python
grid.fit(X_train_res, y_train_res)
end_time = time.time()

# Print the best parameters found
print("Best parameters found: ", grid.best_params_)
# Best score
print("Best log loss: ", -grid.best_score_)
print(f"Total training time: {end_time - start_time:.2f} seconds")
```

```
Best parameters found:  {'boosting_type': 'gbdt', 'colsample_bytree': 0.5,
'learning_rate': 0.1, 'max_depth': 5, 'objective': 'binary', 'subsample': 0.5}
Best log loss:  0.23786119504245762
Total training time: 15.56 seconds
```

[79]:
```python
# Retrieve the best model (estimator)
best_model = grid.best_estimator_

# Make predictions on the test set
y_train_pred = best_model.predict(X_train_res)
y_test_pred = best_model.predict(X_test_scaled)

# Evaluate the model
# Accuracy
train_accuracy = accuracy_score(y_train_res, y_train_pred)
print(f"Training Accuracy: {train_accuracy:.2f}")

test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Test Accuracy: {test_accuracy:.2f}")
```

```
Training Accuracy: 0.96
Test Accuracy: 0.90
```

Observations:

A log loss of 0.238 means that, on average, the model's predicted probabilities are close to the actual outcomes. It indicates that the model's probability predictions are relatively accurate.

A training accuracy of 0.96 means that the model correctly predicts the class labels for 96% of the samples in the training dataset. It suggests that the model has learned the patterns present in the training data relatively well.

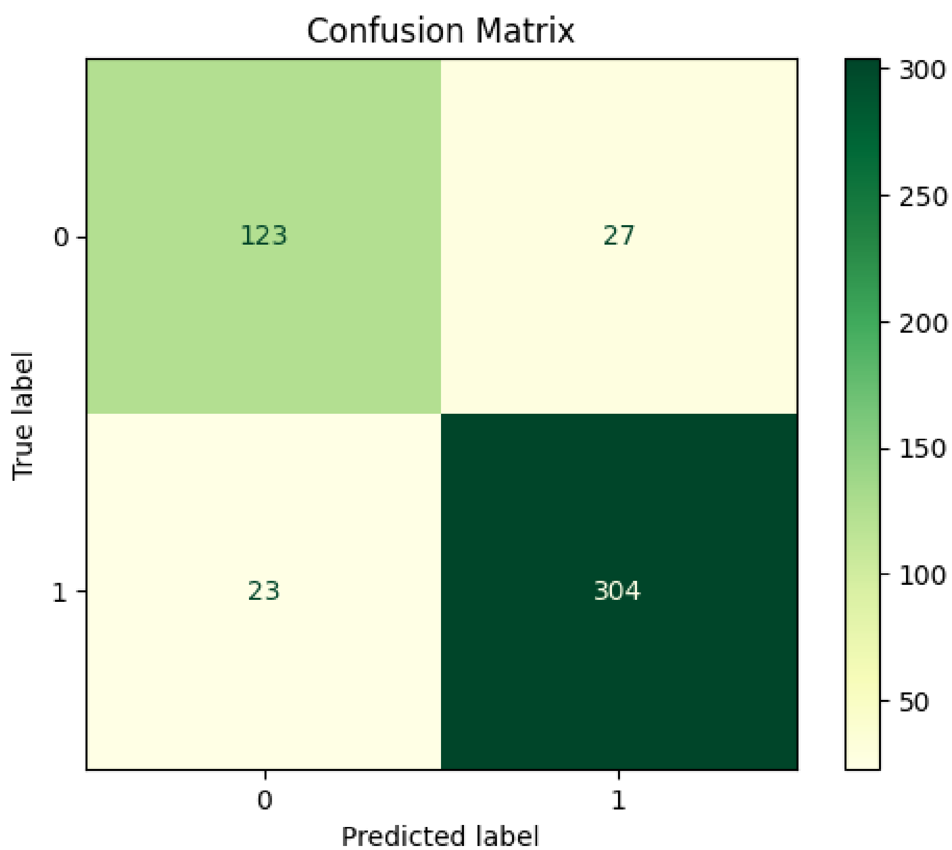
A test accuracy of 0.90 means that the model correctly predicts the class labels for 90% of the samples in the test dataset. It suggests that the model performs well on unseen data, indicating good generalization ability.

Confusion matrix/Classification Report

[80]:
```python
conf_matrix = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[123  27]
 [ 23 304]]
```

[81]:
```
disp = ConfusionMatrixDisplay(conf_matrix)
cmap = plt.cm.YlGn
disp.plot(cmap=cmap)
plt.title('Confusion Matrix')
plt.show()
```



[82]:
```
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.82      0.83       150
           1       0.92      0.93      0.92       327

    accuracy                           0.90       477
   macro avg       0.88      0.87      0.88       477
weighted avg       0.89      0.90      0.89       477
```

Precision: Precision is the ratio of correctly predicted positive observations to the total predicted positives. For class 0, the precision is 0.84, and for class 1, it is 0.92. This means that when the model predicts class 0, it is correct 84% of the time, and when it predicts class 1, it is correct 92% of the time.

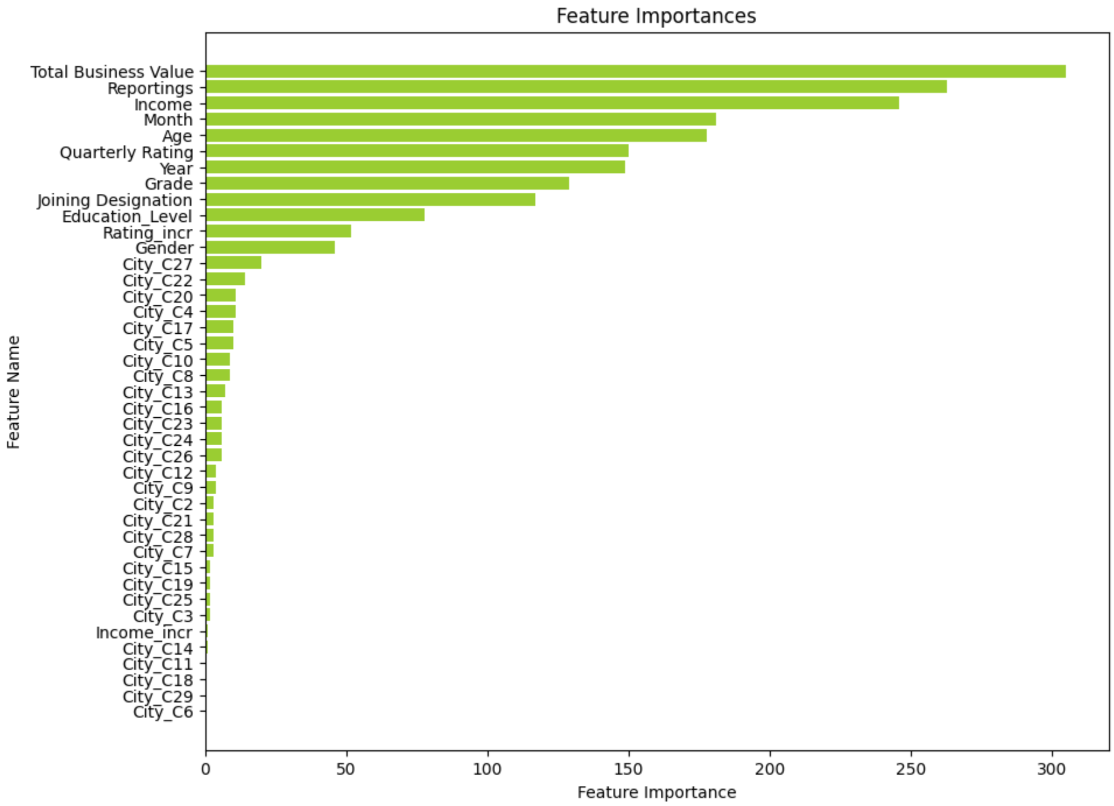
Recall (Sensitivity): Recall is the ratio of correctly predicted positive observations to the all observations in actual class. For class 0, the recall is 0.82, and for class 1, it is 0.93. This implies that the model is able to capture 82% of the actual class 0 instances and 93% of the actual class 1 instances.

F1-score: F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall. For class 0, the F1-score is 0.83, and for class 1, it is 0.92. The weighted average of these scores is 0.90

Support: Support is the number of actual occurrences of the class in the specified dataset. For class 0, the support is 150, and for class 1, it is 327.

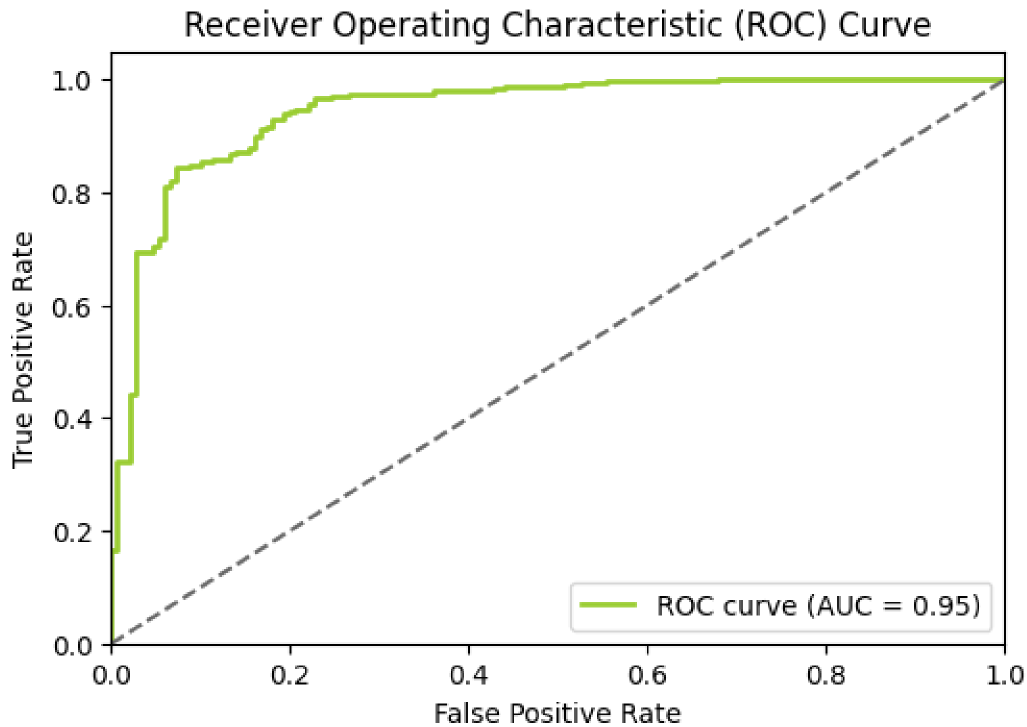
Accuracy: Accuracy is the ratio of correctly predicted observations to the total observations. In this case, the overall accuracy of the model on the test data is 0.90, meaning it correctly predicts the class for 90% of the samples.

[83]:
```
feature_importances = best_model.feature_importances_

# Assuming X_train_res is your training data
# Assuming column_names is a list containing the names of your features
# You may obtain column_names from your DataFrame if you used one initially

# Create a dictionary to store feature names and their importances
feature_importance_dict = dict(zip(X_train_res.columns, feature_importances))

# Sort the dictionary by importance values in descending order
sorted_feature_importance = sorted(feature_importance_dict.items(), key=lambda␣
 ↪x: x[1], reverse=True)

# Extract feature names and importances
sorted_feature_names = [x[0] for x in sorted_feature_importance]
sorted_importances = [x[1] for x in sorted_feature_importance]

# Plot feature importances
plt.figure(figsize=(10, 8))
plt.barh(sorted_feature_names, sorted_importances,color='yellowgreen')
plt.xlabel('Feature Importance')
plt.ylabel('Feature Name')
plt.title('Feature Importances')
plt.gca().invert_yaxis()  # Invert y-axis to show highest importance at the top
plt.show()
```



Roc curve AOC

[84]:
```
warnings.filterwarnings("ignore")
# Make predictions on the test set
y_pred_proba = best_model.predict_proba(X_test_scaled)[:, 1]

# Compute ROC curve and ROC-AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Plot ROC curve
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='yellowgreen', lw=2, label='ROC curve (AUC = %0.2f)' %
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='dimgrey', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

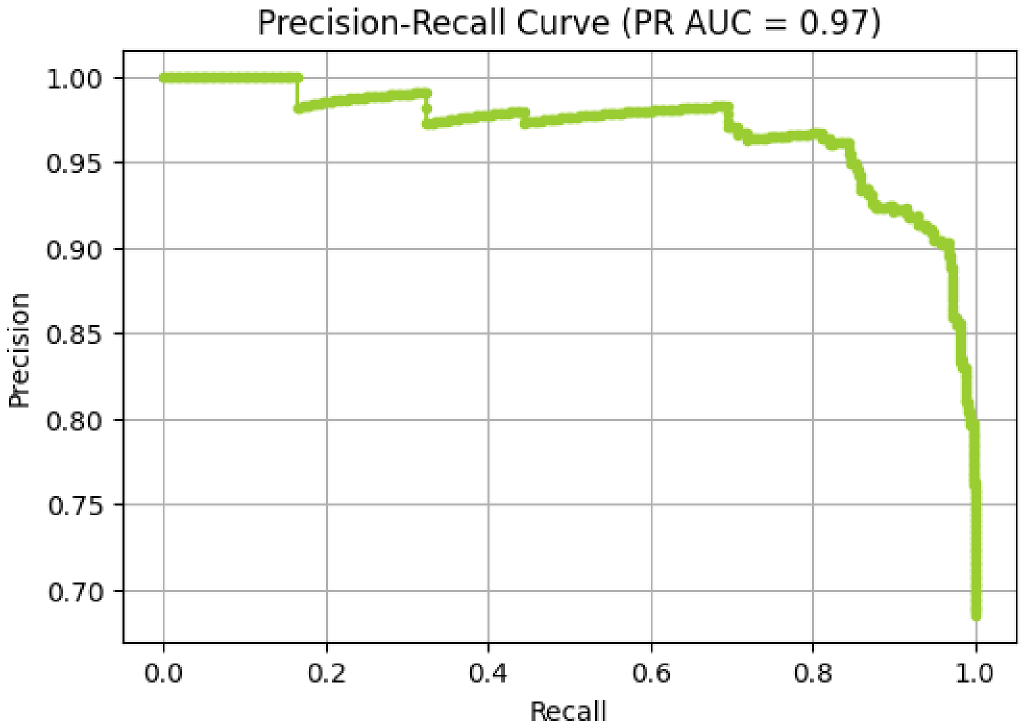


Observations:

An AUC of 0.95 means that the binary classification model has excellent discrimination ability, with high true positive rates and low false positive rates across different thresholds. It suggests that the model performs well in distinguishing between positive and negative samples, making it highly reliable for classification tasks.

Precision Recall Curve

[85]:
```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_proba)
pr_auc = auc(recall, precision)
# Plot the precision-recall curve
plt.figure(figsize=(6, 4))
plt.plot(recall, precision, marker='.',color='yellowgreen')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (PR AUC = {:.2f})'.format(pr_auc))
plt.grid(True)

plt.show()
```



Observations:

A PR AUC of 0.97 suggests that the binary classification model performs exceptionally well in terms of both precision and recall.

It indicates that the model achieves very high precision (the proportion of true positive predictions among all positive predictions) and recall (the proportion of true positive predictions among all actual positive samples) across different thresholds.

Such PR AUC value implies that the model makes very few false positive and false negative predictions, making it highly reliable for classification tasks, especially in scenarios where both precision and recall are crucial.

Business Insights General Observations * Five number of reportings are having the highest frequency. * Males are higher in ratio than females among Drivers (59% males, 41% females). * C20 is the city with the maximum number of drivers, followed by C15. * Maximum Drivers have Grade 2. * Maximum number of Drivers have Quarterly Rating as 1. * 68% of the Drivers have been churned. * Hardly 2% of the Drivers got an Increment in Income. * 15% of the Drivers got an Increase in Rating. * 73% had their last Quarter Rating as 1, followed by 15% having 2. * Joining Designation is highest for 1 (43%), followed by 2 (34%). * Grade at the time of Reporting is highest for Grade 2 (36%), followed by Grade 1 (31%). * Distribution of Education Level for all 3 levels is almost the same (33% each).

Business and Quarterly Trends * Most of the Drivers had their last working date in July 2019. * Most of the Drivers joined in the month of July 2020. * Drivers with Grade 3 have the highest business value, followed by Grade 4 and Grade 2. * The city with the most improvement in

Quarterly Rating over the past year is C22. * Total Business Value of Drivers is highest in C29, followed by C26. * Average Quarterly Rating is found to be highest in the 3rd Quarter, particularly in March.

Churn Insights

- 80% of Drivers with Grade 1 got churned, followed by Grade 2 (70%).
- Drivers with Joining Designation 1 and 5 got churned the most (~75%).
- 80% of Drivers with Quarterly Rating 1 left the company, followed by 40% of QR2 and 18% of QR3.
- 77% of Drivers who did not get any increase in Rating left the company.
- 70% of Drivers who did not get any increment in Income left the company.
- 80% of Drivers from City C13 left the company, closely followed by C17 and C23.
- There is no significant observation on churn with respect to joining month.
- 90% of Drivers who joined in 2018 left the company, followed by 2019 and 2017.

Demographics and Churn Analysis

- Number of Reportings and Age are relatively lower for Drivers who left.
- Most Drivers who got churned belong to the age range 25-35 (close to normal distribution).
- Income is lower for Drivers who left (slightly right-skewed distribution).
- Total Business Value is lower for Drivers who left (right-skewed distribution).

Correlations and Statistical Insights

- Reportings are highly positively correlated to Total Business Value.
- Quarterly Rating and Rating_incr are highly correlated (as expected).
- Grade is highly positively correlated with Income and Joining Designations.
- Drop in Quarterly Rating has a statistically significant impact on the subsequent period's Business Value.
- The OLS summary indicates that Age, Gender, Income, Joining Designation, Grade, and Total Business Value impact the Quarterly Rating.
- A Driver's Total Business Value and Churn Rate are affected by the City they operate in.

Machine Learning Models and Results Ensemble ML Bagging (Random Forest Classifier) * F1-score (harmonic mean of precision and recall): * For Class 0: 0.83 * For Class 1: 0.92 * Weighted average: 0.89 Ensemble ML Boosting (Light GBM) * F1-score (harmonic mean of precision and recall): * For Class 0: 0.83 * For Class 1: 0.92 * Weighted average: 0.90

Recommendations

1-Training and Development

Driver Training Programs:

Target Audience: Drivers with Grade 2 and those in high-churn categories.

Content: Improve driving skills, customer service, and adherence to safety protocols.

Objective: Enhance performance and reduce churn rates.

2-Incentive Schemes

Performance-based Incentives:

Top Performers: Reward drivers with high business value and low churn rates.

Incentives: Financial bonuses, recognition programs, and career progression opportunities.

Churn Reduction: Special bonuses for drivers maintaining high quarterly ratings and consistent performance.

Focus: Encourage retention, especially for drivers in high-churn cities.

3-Recruitment Strategies

Targeted Recruitment: Cities with Growth Potential: Prioritize cities like C22 for recruitment drives.

Strategy: Highlight benefits and career growth opportunities in recruitment campaigns.

Demographics:

Age Group: Focus on drivers aged 25-35, who have shown high performance potential. Gender Balance: Maintain a balanced recruitment strategy to address gender representation disparities.

4-Operational Improvements

City-Specific Strategies:

High Business Value Cities: Enhance support and resources in cities like C29 and C26.

Initiatives: Provide better infrastructure, more support staff, and improved working conditions.

Churn Management in High-Risk Cities: Implement special programs in cities with high churn rates (e.g., C13, C17, C23).

Approach: Conduct exit interviews to understand reasons for churn and address them proactively.

5-Continuous Monitoring and Feedback

Feedback Mechanisms:

Driver Surveys: Regularly collect feedback from drivers about their experiences, challenges, and suggestions.

Frequency: Quarterly surveys and feedback sessions.

Customer Feedback: Gather customer reviews and ratings to identify areas for improvement.

Integration: Use feedback to refine training programs and operational strategies.

6-Data-Driven Decision Making

Utilize Model Insights:

Feature Analysis: Focus on key features identified by models, such as Total Business Value, Quarterly Ratings, and Reportings.

Actionable Insights: Develop policies and programs based on these critical features.

Periodic Model Reviews: Regularly update and validate models to ensure their relevance and accuracy.

Adaptation: Adjust strategies based on updated insights and emerging trends.

7-Strategic Partnerships

Collaboration with Educational Institutions:

Training Programs: Partner with driving schools and educational institutions to provide advanced training to drivers.

Objective: Enhance skills and knowledge, leading to better performance and customer satisfaction.

8-Compensation and Benefits

Income Increment and Benefits:

Performance-Linked Pay: Implement a compensation structure that rewards performance improvements and loyalty.

Objective: Reduce churn by providing financial stability and growth opportunities.

Additional Benefits: Offer health insurance, retirement benefits, and other perks to make the job more attractive.

9-Operational Efficiency

Process Improvements:

Churn Analysis: Regularly analyze churn data to identify patterns and implement targeted interventions.

Proactive Measures: Develop early warning systems to identify drivers at risk of churn and address their concerns promptly.

Conclusion:

Implementing these recommendations will help Ola improve driver retention, enhance performance, and ultimately provide a better service to its customers. The focus should be on targeted training, strategic recruitment, performance-based incentives, and continuous monitoring to adapt to changing dynamics in the transportation industry.