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

import datetime as dt

data=pd.read_csv('ola_driver_scaler.csv')

data.head()
```

```
   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
```

```python
data.shape
```

```
(19104, 14)
```

```python
data.drop('Unnamed: 0', axis=1, inplace=True)

data.columns
```

```
Index(['MMM-YY', 'Driver_ID', 'Age', 'Gender', 'City',
'Education_Level',
       'Income', 'Dateofjoining', 'LastWorkingDate', 'Joining
Designation',
       'Grade', 'Total Business Value', 'Quarterly Rating'],
      dtype='object')
```

```
data['Attrition']=[0 if pd.isna(i) else 1 for i in
data['LastWorkingDate']] #Create the new target column

data['Attrition']. value_counts() #This data is unbalanced

Attrition
0    17488
1     1616
Name: count, dtype: int64

#Given that we know it is important to retain drivers rather than hire
new ones, it is important that we predict the class 1 correctly

data.dtypes

MMM-YY                   object
Driver_ID                 int64
Age                     float64
Gender                  float64
City                     object
Education_Level           int64
Income                    int64
Dateofjoining            object
LastWorkingDate          object
Joining Designation       int64
Grade                     int64
Total Business Value      int64
Quarterly Rating          int64
Attrition                 int64
dtype: object

data.isnull().sum()

MMM-YY                      0
Driver_ID                   0
Age                        61
Gender                     52
City                        0
Education_Level             0
Income                      0
Dateofjoining               0
LastWorkingDate         17488
Joining Designation         0
Grade                       0
Total Business Value        0
Quarterly Rating            0
Attrition                   0
dtype: int64

data['DOJ']=pd.to_datetime(data['Dateofjoining'], format='mixed')
```

```
data['LWD']=pd.to_datetime(data['LastWorkingDate'], format='mixed')

data.shape

(19104, 16)

data.head()

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

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

   Total Business Value  Quarterly Rating  Attrition        DOJ
LWD
0               2381060                 2          0 2018-12-24
NaT
1               -665480                 2          0 2018-12-24
NaT
2                     0                 2          1 2018-12-24 2019-
03-11
3                     0                 1          0 2020-11-06
NaT
4                     0                 1          0 2020-11-06
NaT

data['LWD'].head()

0          NaT
1          NaT
2   2019-03-11
3          NaT
4          NaT
Name: LWD, dtype: datetime64[ns]

data.drop(['Dateofjoining','LastWorkingDate'], axis=1, inplace=True)

data.rename(columns={'DOJ':'Dateofjoining', 'LWD':'LastWorkingDate'},
inplace=True)
```

```
data.groupby('Joining Designation')
['Attrition'].sum()/data.groupby('Joining Designation')
['Attrition'].count()
```

```
Joining Designation
1    0.076493
2    0.094039
3    0.096242
4    0.064516
5    0.061538
Name: Attrition, dtype: float64
```

```
tenure_days=[]
for i in range(data.shape[0]):
    if (data.loc[i,'LastWorkingDate']) is pd.NaT:
        tenure_days.append(pd.to_datetime(dt.date.today())-
data.loc[i,'Dateofjoining'])
    else:
        tenure_days.append(data.loc[i,'LastWorkingDate']-
data.loc[i,'Dateofjoining'])
```

```
tenure_days
```

```
[Timedelta('2341 days 00:00:00'),
 Timedelta('2341 days 00:00:00'),
 Timedelta('77 days 00:00:00'),
 Timedelta('1658 days 00:00:00'),
 Timedelta('1658 days 00:00:00'),
 Timedelta('1993 days 00:00:00'),
 Timedelta('1993 days 00:00:00'),
 Timedelta('1993 days 00:00:00'),
 Timedelta('1993 days 00:00:00'),
 Timedelta('142 days 00:00:00'),
 Timedelta('2325 days 00:00:00'),
 Timedelta('2325 days 00:00:00'),
 Timedelta('57 days 00:00:00'),
 Timedelta('1756 days 00:00:00'),
 Timedelta('1756 days 00:00:00'),
 Timedelta('1756 days 00:00:00'),
 Timedelta('1756 days 00:00:00'),
 Timedelta('1756 days 00:00:00'),
 Timedelta('1706 days 00:00:00'),
 Timedelta('1706 days 00:00:00'),
 Timedelta('57 days 00:00:00'),
 Timedelta('1627 days 00:00:00'),
 Timedelta('2154 days 00:00:00'),
 Timedelta('2154 days 00:00:00'),
 Timedelta('2154 days 00:00:00'),
```

```
 Timedelta('2154 days 00:00:00'),
 Timedelta('2154 days 00:00:00'),
 Timedelta('175 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('3647 days 00:00:00'),
 Timedelta('2008 days 00:00:00'),
 Timedelta('1679 days 00:00:00'),
 Timedelta('1679 days 00:00:00'),
 Timedelta('1679 days 00:00:00'),
 Timedelta('2365 days 00:00:00'),
 Timedelta('84 days 00:00:00'),
 Timedelta('2634 days 00:00:00'),
 Timedelta('2634 days 00:00:00'),
 Timedelta('2634 days 00:00:00'),
 Timedelta('2634 days 00:00:00'),
 Timedelta('2634 days 00:00:00'),
 Timedelta('2634 days 00:00:00'),
 Timedelta('501 days 00:00:00'),
 Timedelta('2325 days 00:00:00'),
 Timedelta('2325 days 00:00:00'),
 Timedelta('2325 days 00:00:00'),
 Timedelta('2325 days 00:00:00'),
 Timedelta('111 days 00:00:00'),
 Timedelta('2036 days 00:00:00'),
 Timedelta('2036 days 00:00:00'),
 Timedelta('2036 days 00:00:00'),
 Timedelta('2036 days 00:00:00'),
 Timedelta('2036 days 00:00:00'),
 Timedelta('128 days 00:00:00'),
```

```
 Timedelta('2567 days 00:00:00'),
 Timedelta('2567 days 00:00:00'),
 Timedelta('2567 days 00:00:00'),
 Timedelta('2567 days 00:00:00'),
 Timedelta('2567 days 00:00:00'),
 Timedelta('2567 days 00:00:00'),
 Timedelta('2567 days 00:00:00'),
 Timedelta('2567 days 00:00:00'),
 Timedelta('2567 days 00:00:00'),
 Timedelta('2567 days 00:00:00'),
 Timedelta('2567 days 00:00:00'),
 Timedelta('2567 days 00:00:00'),
 Timedelta('2567 days 00:00:00'),
 Timedelta('646 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('702 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('520 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
```

```
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2761 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2572 days 00:00:00'),
 Timedelta('2559 days 00:00:00'),
 Timedelta('2559 days 00:00:00'),
 Timedelta('2559 days 00:00:00'),
 Timedelta('2559 days 00:00:00'),
 Timedelta('368 days 00:00:00'),
 Timedelta('2058 days 00:00:00'),
 Timedelta('2058 days 00:00:00'),
 Timedelta('57 days 00:00:00'),
 Timedelta('1783 days 00:00:00'),
 Timedelta('1783 days 00:00:00'),
```

```
 Timedelta('1783 days 00:00:00'),
 Timedelta('1783 days 00:00:00'),
 Timedelta('1783 days 00:00:00'),
 Timedelta('1783 days 00:00:00'),
 Timedelta('1756 days 00:00:00'),
 Timedelta('1756 days 00:00:00'),
 Timedelta('1756 days 00:00:00'),
 Timedelta('106 days 00:00:00'),
 Timedelta('1902 days 00:00:00'),
 Timedelta('1902 days 00:00:00'),
 Timedelta('59 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('2152 days 00:00:00'),
 Timedelta('394 days 00:00:00'),
 Timedelta('1832 days 00:00:00'),
 Timedelta('1832 days 00:00:00'),
 Timedelta('1832 days 00:00:00'),
 Timedelta('1832 days 00:00:00'),
 Timedelta('1832 days 00:00:00'),
 Timedelta('153 days 00:00:00'),
 Timedelta('1994 days 00:00:00'),
 Timedelta('1994 days 00:00:00'),
 Timedelta('1994 days 00:00:00'),
 Timedelta('100 days 00:00:00'),
 Timedelta('1748 days 00:00:00'),
 Timedelta('1748 days 00:00:00'),
 Timedelta('1748 days 00:00:00'),
 Timedelta('1748 days 00:00:00'),
 Timedelta('1748 days 00:00:00'),
 Timedelta('2022 days 00:00:00'),
 Timedelta('2022 days 00:00:00'),
 Timedelta('2022 days 00:00:00'),
 Timedelta('98 days 00:00:00'),
 Timedelta('2149 days 00:00:00'),
 Timedelta('2149 days 00:00:00'),
 Timedelta('2149 days 00:00:00'),
 Timedelta('2149 days 00:00:00'),
 Timedelta('2149 days 00:00:00'),
```

```
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2406 days 00:00:00'),
Timedelta('2406 days 00:00:00'),
Timedelta('2406 days 00:00:00'),
Timedelta('2406 days 00:00:00'),
Timedelta('208 days 00:00:00'),
Timedelta('2505 days 00:00:00'),
Timedelta('222 days 00:00:00'),
Timedelta('1700 days 00:00:00'),
Timedelta('1700 days 00:00:00'),
Timedelta('60 days 00:00:00'),
Timedelta('1819 days 00:00:00'),
Timedelta('1819 days 00:00:00'),
Timedelta('1819 days 00:00:00'),
Timedelta('1819 days 00:00:00'),
Timedelta('1819 days 00:00:00'),
Timedelta('1819 days 00:00:00'),
Timedelta('1819 days 00:00:00'),
Timedelta('2398 days 00:00:00'),
Timedelta('2398 days 00:00:00'),
Timedelta('122 days 00:00:00'),
Timedelta('1686 days 00:00:00'),
Timedelta('1686 days 00:00:00'),
Timedelta('1686 days 00:00:00'),
Timedelta('2554 days 00:00:00'),
Timedelta('2554 days 00:00:00'),
Timedelta('2554 days 00:00:00'),
Timedelta('2554 days 00:00:00'),
Timedelta('2554 days 00:00:00'),
Timedelta('2554 days 00:00:00'),
Timedelta('2554 days 00:00:00'),
Timedelta('2554 days 00:00:00'),
Timedelta('2554 days 00:00:00'),
Timedelta('2554 days 00:00:00'),
Timedelta('2554 days 00:00:00'),
Timedelta('2554 days 00:00:00'),
Timedelta('606 days 00:00:00'),
```

```
 Timedelta('2375 days 00:00:00'),
 Timedelta('92 days 00:00:00'),
 Timedelta('2922 days 00:00:00'),
 Timedelta('2922 days 00:00:00'),
 Timedelta('2922 days 00:00:00'),
 Timedelta('2922 days 00:00:00'),
 Timedelta('2922 days 00:00:00'),
 Timedelta('2922 days 00:00:00'),
 Timedelta('2922 days 00:00:00'),
 Timedelta('2922 days 00:00:00'),
 Timedelta('2922 days 00:00:00'),
 Timedelta('945 days 00:00:00'),
 Timedelta('1789 days 00:00:00'),
 Timedelta('1789 days 00:00:00'),
 Timedelta('1789 days 00:00:00'),
 Timedelta('1789 days 00:00:00'),
 Timedelta('1789 days 00:00:00'),
 Timedelta('1789 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2142 days 00:00:00'),
 Timedelta('2009 days 00:00:00'),
 Timedelta('2009 days 00:00:00'),
 Timedelta('2009 days 00:00:00'),
 Timedelta('2009 days 00:00:00'),
 Timedelta('115 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
```

```
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2498 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('2857 days 00:00:00'),
 Timedelta('1091 days 00:00:00'),
 Timedelta('102 days 00:00:00'),
 Timedelta('2328 days 00:00:00'),
 Timedelta('2328 days 00:00:00'),
 Timedelta('81 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
```

```
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('3169 days 00:00:00'),
 Timedelta('2575 days 00:00:00'),
 Timedelta('2575 days 00:00:00'),
 Timedelta('317 days 00:00:00'),
 Timedelta('1636 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('3309 days 00:00:00'),
 Timedelta('1938 days 00:00:00'),
 Timedelta('1938 days 00:00:00'),
 Timedelta('1938 days 00:00:00'),
 Timedelta('1938 days 00:00:00'),
 Timedelta('140 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
```

```
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('2487 days 00:00:00'),
 Timedelta('686 days 00:00:00'),
 Timedelta('1948 days 00:00:00'),
 Timedelta('1948 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('2330 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
```

```
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('3504 days 00:00:00'),
 Timedelta('2016 days 00:00:00'),
 Timedelta('2016 days 00:00:00'),
 Timedelta('2016 days 00:00:00'),
 Timedelta('2016 days 00:00:00'),
 Timedelta('2016 days 00:00:00'),
 Timedelta('2016 days 00:00:00'),
 Timedelta('2016 days 00:00:00'),
 Timedelta('2016 days 00:00:00'),
 Timedelta('2016 days 00:00:00'),
 Timedelta('2016 days 00:00:00'),
 Timedelta('303 days 00:00:00'),
 Timedelta('113 days 00:00:00'),
 Timedelta('2554 days 00:00:00'),
 Timedelta('266 days 00:00:00'),
 Timedelta('1677 days 00:00:00'),
 Timedelta('1677 days 00:00:00'),
 Timedelta('1677 days 00:00:00'),
 Timedelta('1847 days 00:00:00'),
 Timedelta('1847 days 00:00:00'),
 Timedelta('1847 days 00:00:00'),
 Timedelta('1847 days 00:00:00'),
 Timedelta('1847 days 00:00:00'),
 Timedelta('176 days 00:00:00'),
 Timedelta('1766 days 00:00:00'),
 Timedelta('1766 days 00:00:00'),
 Timedelta('1766 days 00:00:00'),
 Timedelta('1766 days 00:00:00'),
 Timedelta('1766 days 00:00:00'),
 Timedelta('145 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
```

```
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2561 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2440 days 00:00:00'),
 Timedelta('2144 days 00:00:00'),
 Timedelta('31 days 00:00:00'),
 Timedelta('1693 days 00:00:00'),
 Timedelta('1693 days 00:00:00'),
 Timedelta('1693 days 00:00:00'),
```

```
Timedelta('2536 days 00:00:00'),
Timedelta('2536 days 00:00:00'),
Timedelta('2536 days 00:00:00'),
Timedelta('2536 days 00:00:00'),
Timedelta('2536 days 00:00:00'),
Timedelta('2536 days 00:00:00'),
Timedelta('2536 days 00:00:00'),
Timedelta('2536 days 00:00:00'),
Timedelta('2536 days 00:00:00'),
Timedelta('479 days 00:00:00'),
Timedelta('1867 days 00:00:00'),
Timedelta('1867 days 00:00:00'),
Timedelta('1867 days 00:00:00'),
Timedelta('1867 days 00:00:00'),
Timedelta('109 days 00:00:00'),
Timedelta('2093 days 00:00:00'),
Timedelta('2093 days 00:00:00'),
Timedelta('2093 days 00:00:00'),
Timedelta('2093 days 00:00:00'),
Timedelta('2093 days 00:00:00'),
Timedelta('2093 days 00:00:00'),
Timedelta('2093 days 00:00:00'),
Timedelta('2093 days 00:00:00'),
Timedelta('2093 days 00:00:00'),
Timedelta('283 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('2149 days 00:00:00'),
Timedelta('428 days 00:00:00'),
Timedelta('1697 days 00:00:00'),
Timedelta('1697 days 00:00:00'),
Timedelta('1697 days 00:00:00'),
Timedelta('244 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
```

```
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2091 days 00:00:00'),
Timedelta('2693 days 00:00:00'),
Timedelta('2693 days 00:00:00'),
Timedelta('2693 days 00:00:00'),
Timedelta('2693 days 00:00:00'),
Timedelta('2693 days 00:00:00'),
Timedelta('513 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('2184 days 00:00:00'),
Timedelta('426 days 00:00:00'),
Timedelta('2019 days 00:00:00'),
Timedelta('2019 days 00:00:00'),
Timedelta('2019 days 00:00:00'),
Timedelta('2019 days 00:00:00'),
Timedelta('120 days 00:00:00'),
Timedelta('2119 days 00:00:00'),
Timedelta('2119 days 00:00:00'),
Timedelta('2119 days 00:00:00'),
Timedelta('2119 days 00:00:00'),
Timedelta('134 days 00:00:00'),
Timedelta('2504 days 00:00:00'),
Timedelta('204 days 00:00:00'),
Timedelta('1939 days 00:00:00'),
Timedelta('39 days 00:00:00'),
Timedelta('1770 days 00:00:00'),
Timedelta('1770 days 00:00:00'),
Timedelta('1770 days 00:00:00'),
```

```
 Timedelta('1770 days 00:00:00'),
 Timedelta('133 days 00:00:00'),
 Timedelta('1692 days 00:00:00'),
 Timedelta('1692 days 00:00:00'),
 Timedelta('1692 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('2107 days 00:00:00'),
 Timedelta('498 days 00:00:00'),
 Timedelta('1885 days 00:00:00'),
 Timedelta('1885 days 00:00:00'),
 Timedelta('1885 days 00:00:00'),
 Timedelta('1885 days 00:00:00'),
 Timedelta('1885 days 00:00:00'),
 Timedelta('127 days 00:00:00'),
 Timedelta('2727 days 00:00:00'),
 Timedelta('2727 days 00:00:00'),
 Timedelta('2727 days 00:00:00'),
 Timedelta('2727 days 00:00:00'),
 Timedelta('516 days 00:00:00'),
 Timedelta('2329 days 00:00:00'),
 Timedelta('2329 days 00:00:00'),
 Timedelta('75 days 00:00:00'),
 Timedelta('2000 days 00:00:00'),
 Timedelta('2000 days 00:00:00'),
 Timedelta('2000 days 00:00:00'),
 Timedelta('2000 days 00:00:00'),
 Timedelta('2000 days 00:00:00'),
 Timedelta('2000 days 00:00:00'),
 Timedelta('2000 days 00:00:00'),
 Timedelta('217 days 00:00:00'),
 Timedelta('1740 days 00:00:00'),
 Timedelta('1740 days 00:00:00'),
 Timedelta('1740 days 00:00:00'),
 Timedelta('1740 days 00:00:00'),
 Timedelta('1740 days 00:00:00'),
```

```
 Timedelta('110 days 00:00:00'),
 Timedelta('3282 days 00:00:00'),
 Timedelta('3282 days 00:00:00'),
 Timedelta('3282 days 00:00:00'),
 Timedelta('3282 days 00:00:00'),
 Timedelta('3282 days 00:00:00'),
 Timedelta('3282 days 00:00:00'),
 Timedelta('3282 days 00:00:00'),
 Timedelta('1162 days 00:00:00'),
 Timedelta('1988 days 00:00:00'),
 Timedelta('1988 days 00:00:00'),
 Timedelta('1988 days 00:00:00'),
 Timedelta('1988 days 00:00:00'),
 Timedelta('1988 days 00:00:00'),
 Timedelta('1988 days 00:00:00'),
 Timedelta('1988 days 00:00:00'),
 Timedelta('1988 days 00:00:00'),
 Timedelta('1988 days 00:00:00'),
 Timedelta('1988 days 00:00:00'),
 Timedelta('1988 days 00:00:00'),
 Timedelta('351 days 00:00:00'),
 Timedelta('1960 days 00:00:00'),
 Timedelta('1960 days 00:00:00'),
 Timedelta('1960 days 00:00:00'),
 Timedelta('1960 days 00:00:00'),
 Timedelta('1960 days 00:00:00'),
 Timedelta('145 days 00:00:00'),
 Timedelta('1864 days 00:00:00'),
 Timedelta('1864 days 00:00:00'),
 Timedelta('1864 days 00:00:00'),
 Timedelta('1864 days 00:00:00'),
 Timedelta('1864 days 00:00:00'),
 Timedelta('1864 days 00:00:00'),
 Timedelta('1864 days 00:00:00'),
 Timedelta('1864 days 00:00:00'),
 Timedelta('1864 days 00:00:00'),
 Timedelta('2072 days 00:00:00'),
 Timedelta('2072 days 00:00:00'),
 Timedelta('2072 days 00:00:00'),
 Timedelta('2072 days 00:00:00'),
 Timedelta('2072 days 00:00:00'),
 Timedelta('151 days 00:00:00'),
 Timedelta('3082 days 00:00:00'),
 Timedelta('3082 days 00:00:00'),
 Timedelta('3082 days 00:00:00'),
 Timedelta('3082 days 00:00:00'),
 Timedelta('3082 days 00:00:00'),
 Timedelta('3082 days 00:00:00'),
 Timedelta('3082 days 00:00:00'),
```

```
 Timedelta('3082 days 00:00:00'),
 Timedelta('1008 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2518 days 00:00:00'),
 Timedelta('2160 days 00:00:00'),
 Timedelta('2160 days 00:00:00'),
 Timedelta('2160 days 00:00:00'),
 Timedelta('2160 days 00:00:00'),
 Timedelta('116 days 00:00:00'),
 Timedelta('1697 days 00:00:00'),
 Timedelta('1697 days 00:00:00'),
 Timedelta('1697 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
```

```
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('3591 days 00:00:00'),
 Timedelta('2361 days 00:00:00'),
 Timedelta('2361 days 00:00:00'),
 Timedelta('2361 days 00:00:00'),
 Timedelta('2361 days 00:00:00'),
 Timedelta('2361 days 00:00:00'),
 Timedelta('2361 days 00:00:00'),
 Timedelta('2361 days 00:00:00'),
 Timedelta('2361 days 00:00:00'),
 Timedelta('293 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('3402 days 00:00:00'),
 Timedelta('1899 days 00:00:00'),
 Timedelta('1899 days 00:00:00'),
 Timedelta('1899 days 00:00:00'),
 Timedelta('1899 days 00:00:00'),
 Timedelta('1899 days 00:00:00'),
 Timedelta('1899 days 00:00:00'),
 Timedelta('189 days 00:00:00'),
```

```
 Timedelta('2014 days 00:00:00'),
 Timedelta('2014 days 00:00:00'),
 Timedelta('2014 days 00:00:00'),
 Timedelta('2014 days 00:00:00'),
 Timedelta('129 days 00:00:00'),
 Timedelta('2063 days 00:00:00'),
 Timedelta('2063 days 00:00:00'),
 Timedelta('2063 days 00:00:00'),
 Timedelta('2063 days 00:00:00'),
 Timedelta('2063 days 00:00:00'),
 Timedelta('2063 days 00:00:00'),
 Timedelta('183 days 00:00:00'),
 Timedelta('1902 days 00:00:00'),
 Timedelta('1902 days 00:00:00'),
 Timedelta('78 days 00:00:00'),
 Timedelta('1749 days 00:00:00'),
 Timedelta('1749 days 00:00:00'),
 Timedelta('1749 days 00:00:00'),
 Timedelta('1749 days 00:00:00'),
 Timedelta('1749 days 00:00:00'),
 Timedelta('2419 days 00:00:00'),
 Timedelta('2419 days 00:00:00'),
 Timedelta('2419 days 00:00:00'),
 Timedelta('2419 days 00:00:00'),
 Timedelta('2419 days 00:00:00'),
 Timedelta('2419 days 00:00:00'),
 Timedelta('2419 days 00:00:00'),
 Timedelta('2419 days 00:00:00'),
 Timedelta('351 days 00:00:00'),
 Timedelta('2489 days 00:00:00'),
 Timedelta('2489 days 00:00:00'),
 Timedelta('2489 days 00:00:00'),
 Timedelta('2489 days 00:00:00'),
 Timedelta('2489 days 00:00:00'),
 Timedelta('2489 days 00:00:00'),
 Timedelta('357 days 00:00:00'),
 Timedelta('2575 days 00:00:00'),
 Timedelta('2575 days 00:00:00'),
 Timedelta('301 days 00:00:00'),
 Timedelta('1776 days 00:00:00'),
 Timedelta('1776 days 00:00:00'),
 Timedelta('1776 days 00:00:00'),
 Timedelta('1776 days 00:00:00'),
 Timedelta('1776 days 00:00:00'),
 Timedelta('1776 days 00:00:00'),
 Timedelta('1952 days 00:00:00'),
 Timedelta('1952 days 00:00:00'),
 Timedelta('1952 days 00:00:00'),
 Timedelta('1952 days 00:00:00'),
```

```
 Timedelta('1952 days 00:00:00'),
 Timedelta('1952 days 00:00:00'),
 Timedelta('1952 days 00:00:00'),
 Timedelta('194 days 00:00:00'),
 Timedelta('4231 days 00:00:00'),
 Timedelta('4231 days 00:00:00'),
 Timedelta('4231 days 00:00:00'),
 Timedelta('4231 days 00:00:00'),
 Timedelta('4231 days 00:00:00'),
 Timedelta('4231 days 00:00:00'),
 Timedelta('4231 days 00:00:00'),
 Timedelta('4231 days 00:00:00'),
 Timedelta('4231 days 00:00:00'),
 Timedelta('4231 days 00:00:00'),
 Timedelta('2212 days 00:00:00'),
 Timedelta('1847 days 00:00:00'),
 Timedelta('1847 days 00:00:00'),
 Timedelta('80 days 00:00:00'),
 Timedelta('2488 days 00:00:00'),
 Timedelta('2488 days 00:00:00'),
 Timedelta('2488 days 00:00:00'),
 Timedelta('2488 days 00:00:00'),
 Timedelta('297 days 00:00:00'),
 Timedelta('1683 days 00:00:00'),
 Timedelta('1683 days 00:00:00'),
 Timedelta('1683 days 00:00:00'),
 Timedelta('1710 days 00:00:00'),
 Timedelta('1710 days 00:00:00'),
 Timedelta('1710 days 00:00:00'),
 Timedelta('1710 days 00:00:00'),
 Timedelta('95 days 00:00:00'),
 Timedelta('1812 days 00:00:00'),
 Timedelta('1812 days 00:00:00'),
 Timedelta('1812 days 00:00:00'),
 Timedelta('1812 days 00:00:00'),
 Timedelta('1812 days 00:00:00'),
 Timedelta('1812 days 00:00:00'),
 Timedelta('1812 days 00:00:00'),
 Timedelta('345 days 00:00:00'),
 Timedelta('2030 days 00:00:00'),
 Timedelta('35 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
```

```
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('3310 days 00:00:00'),
 Timedelta('1777 days 00:00:00'),
 Timedelta('1777 days 00:00:00'),
 Timedelta('1777 days 00:00:00'),
 Timedelta('1777 days 00:00:00'),
 Timedelta('113 days 00:00:00'),
 Timedelta('2538 days 00:00:00'),
 Timedelta('243 days 00:00:00'),
 Timedelta('2578 days 00:00:00'),
 Timedelta('2578 days 00:00:00'),
 Timedelta('302 days 00:00:00'),
 Timedelta('2103 days 00:00:00'),
 Timedelta('2103 days 00:00:00'),
 Timedelta('2103 days 00:00:00'),
 Timedelta('97 days 00:00:00'),
 Timedelta('1925 days 00:00:00'),
 Timedelta('1925 days 00:00:00'),
 Timedelta('1925 days 00:00:00'),
 Timedelta('1925 days 00:00:00'),
 Timedelta('130 days 00:00:00'),
 Timedelta('2133 days 00:00:00'),
 Timedelta('2133 days 00:00:00'),
 Timedelta('2133 days 00:00:00'),
 Timedelta('73 days 00:00:00'),
 Timedelta('1623 days 00:00:00'),
 Timedelta('1861 days 00:00:00'),
 Timedelta('1861 days 00:00:00'),
 Timedelta('1861 days 00:00:00'),
 ...]
```

```
data['tenure in days']=tenure_days
```

```
data['tenure in days']
```

```
0       2341 days
1       2341 days
2         77 days
3       1658 days
4       1658 days
           ...
19099   1809 days
19100   1809 days
19101   1809 days
19102   1809 days
19103   1809 days
Name: tenure in days, Length: 19104, dtype: timedelta64[ns]
```

```
#remove this if the cell above works well
#data['tenure in days']=data['LastWorkingDate']-data['Dateofjoining']
```

```
data.groupby('Attrition')['tenure in days'].mean() #the average tenure
of those who have left is almost 1 year
```

```
Attrition
0   2602 days 17:12:14.492223232
1    357 days 13:46:02.376237624
Name: tenure in days, dtype: timedelta64[ns]
```

```
data[data['Attrition']==1]['tenure in days']
```

```
2        77 days
9       142 days
12       57 days
20       57 days
27      175 days
          ...
19039    19 days
19054    92 days
19081    61 days
19090   418 days
19096   334 days
Name: tenure in days, Length: 1616, dtype: timedelta64[ns]
```

```
data.rename(columns={'MMM-YY':'Reporting_date'}, inplace=True)
```

```
data.dtypes
```

```
Reporting_date                      object
Driver_ID                            int64
Age                                float64
Gender                             float64
City                                object
Education_Level                      int64
```

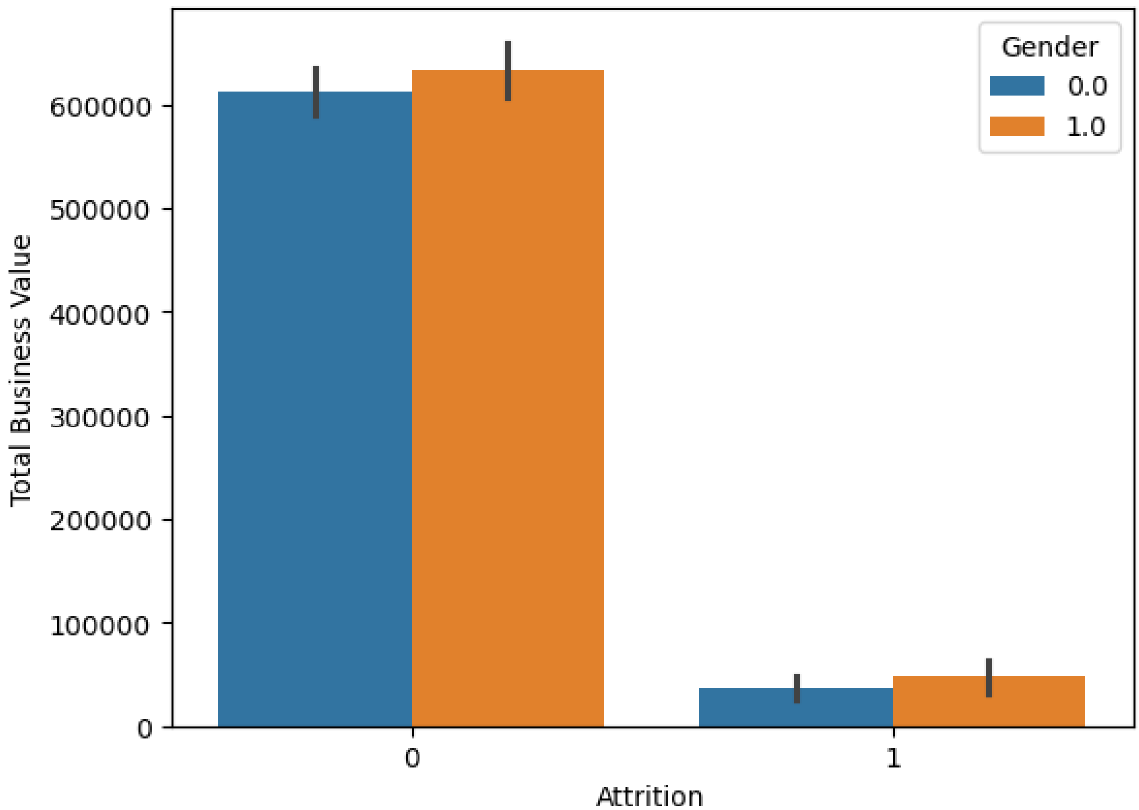
```
Income                                  int64
Joining Designation                     int64
Grade                                   int64
Total Business Value                    int64
Quarterly Rating                        int64
Attrition                               int64
Dateofjoining                  datetime64[ns]
LastWorkingDate                datetime64[ns]
tenure in days                timedelta64[ns]
dtype: object
```

```
data['Reporting_date']=pd.to_datetime(data['Reporting_date'],
format='mixed')

data.dtypes
```

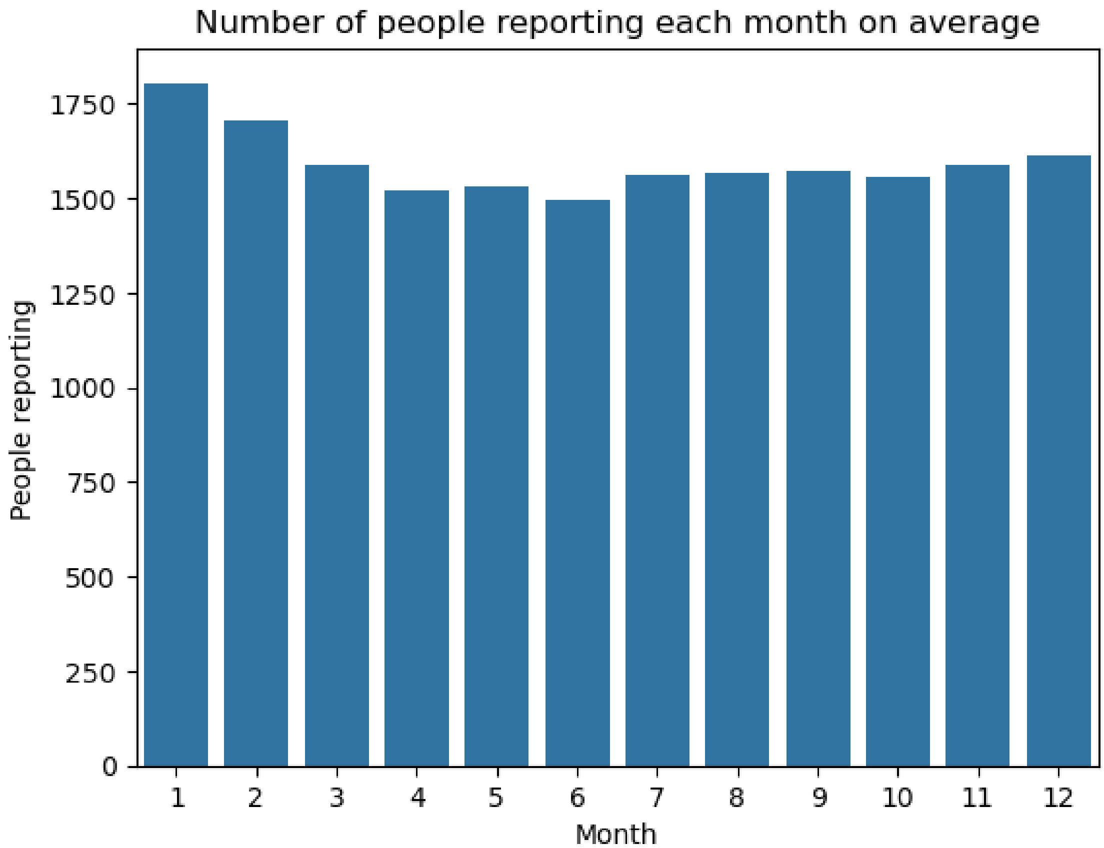
```
Reporting_date                 datetime64[ns]
Driver_ID                               int64
Age                                   float64
Gender                                float64
City                                   object
Education_Level                         int64
Income                                  int64
Joining Designation                     int64
Grade                                   int64
Total Business Value                    int64
Quarterly Rating                        int64
Attrition                               int64
Dateofjoining                  datetime64[ns]
LastWorkingDate                datetime64[ns]
tenure in days                timedelta64[ns]
dtype: object
```

```
#performing knn imputation of on the age and gender columns

import sklearn
from sklearn.impute import KNNImputer, SimpleImputer

knn_imputer=KNNImputer(n_neighbors=5)
sim_imputer=SimpleImputer(strategy='most_frequent')

age=knn_imputer.fit_transform(pd.DataFrame(data['Age'])).reshape(1,-1)
gender=sim_imputer.fit_transform(pd.DataFrame(data['Gender'])).reshape
(1,-1)

age=pd.Series(age[0])

gender=pd.Series(gender[0])

data['Age']=age
```

```
data['Gender']=gender

data.isnull().sum()

Reporting_date               0
Driver_ID                    0
Age                          0
Gender                       0
City                         0
Education_Level              0
Income                       0
Joining Designation          0
Grade                        0
Total Business Value         0
Quarterly Rating             0
Attrition                    0
Dateofjoining                0
LastWorkingDate          17488
tenure in days               0
dtype: int64

data.groupby('Gender')['Attrition'].sum()/data.groupby('Gender')
['Attrition'].count() #Both male and female drivers have equal
attrition rate

Gender
0.0    0.085296
1.0    0.083605
Name: Attrition, dtype: float64

data.groupby(['Gender','Attrition'])['Total Business Value'].mean()

Gender  Attrition
0.0     0            611713.833153
        1             36629.789252
1.0     0            633163.740938
        1             47674.392804
Name: Total Business Value, dtype: float64

sns.barplot(data=data, x='Attrition', y='Total Business Value',
hue='Gender') #We see that women add more business value than men
whether they stay or leave

<Axes: xlabel='Attrition', ylabel='Total Business Value'>
```

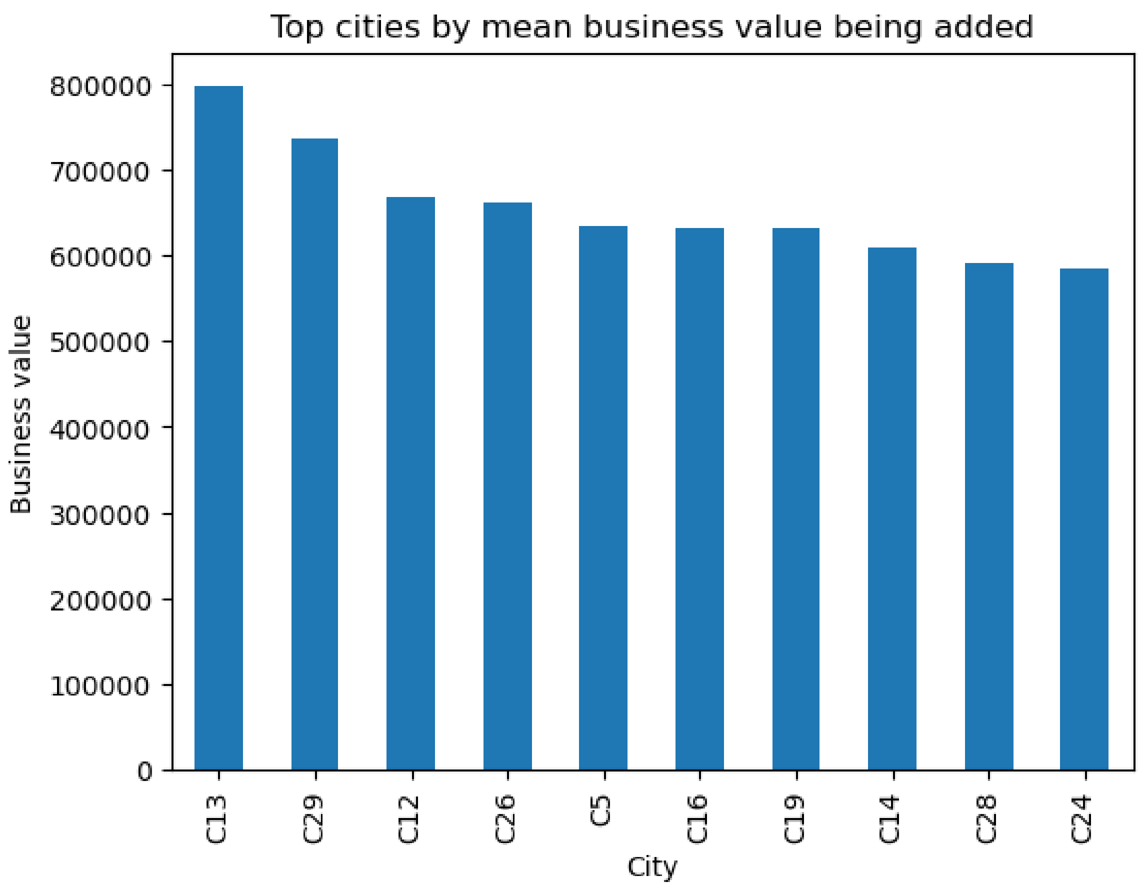



```
sns.countplot(x=data['Reporting_date'].dt.month)
plt.title('Number of people reporting each month on average')
plt.xlabel('Month')
plt.ylabel('People reporting')
plt.show()
```



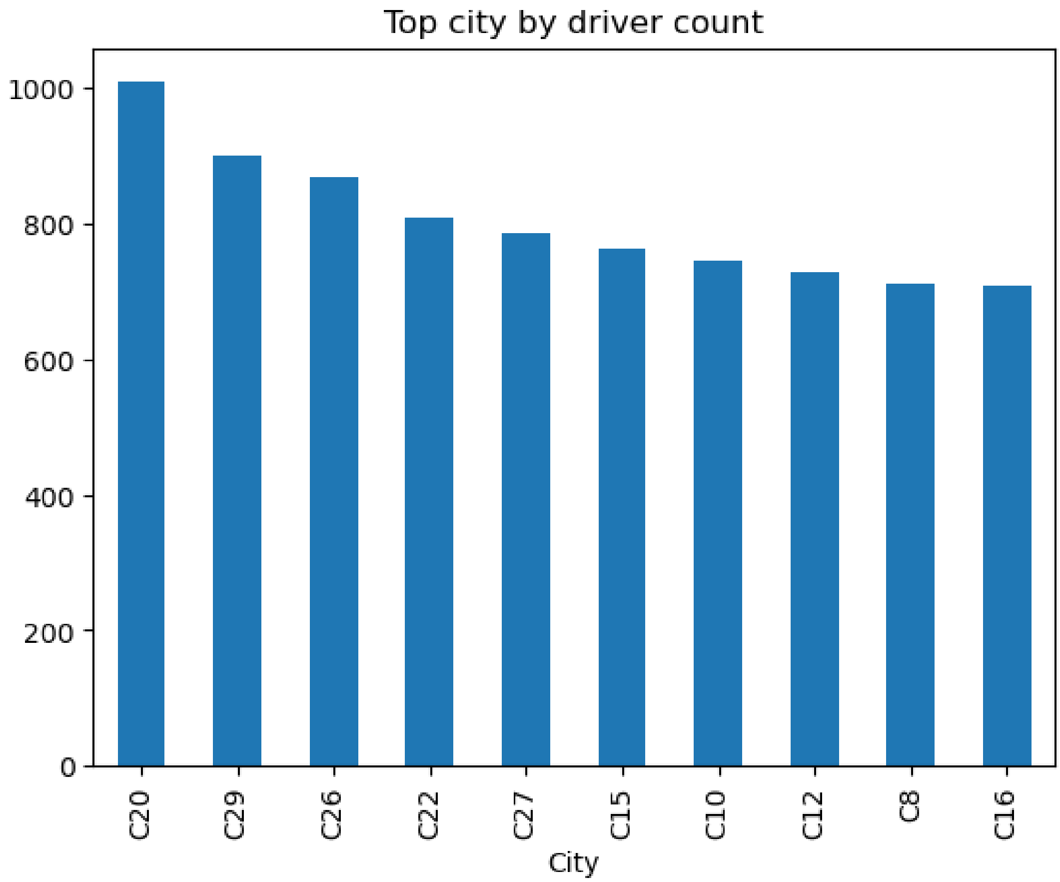

```
data.groupby('City')['Income'].mean().sort_values(ascending=False)
[:10].plot(kind='bar')
plt.title('Top 10 cities by driver income')
plt.ylabel('Income')
plt.show()
```


```
data.groupby('City')['Total Business
Value'].mean().sort_values(ascending=False)[:10].plot(kind='bar')
plt.ylabel('Business value')
plt.title('Top cities by mean business value being added')
plt.show()
```



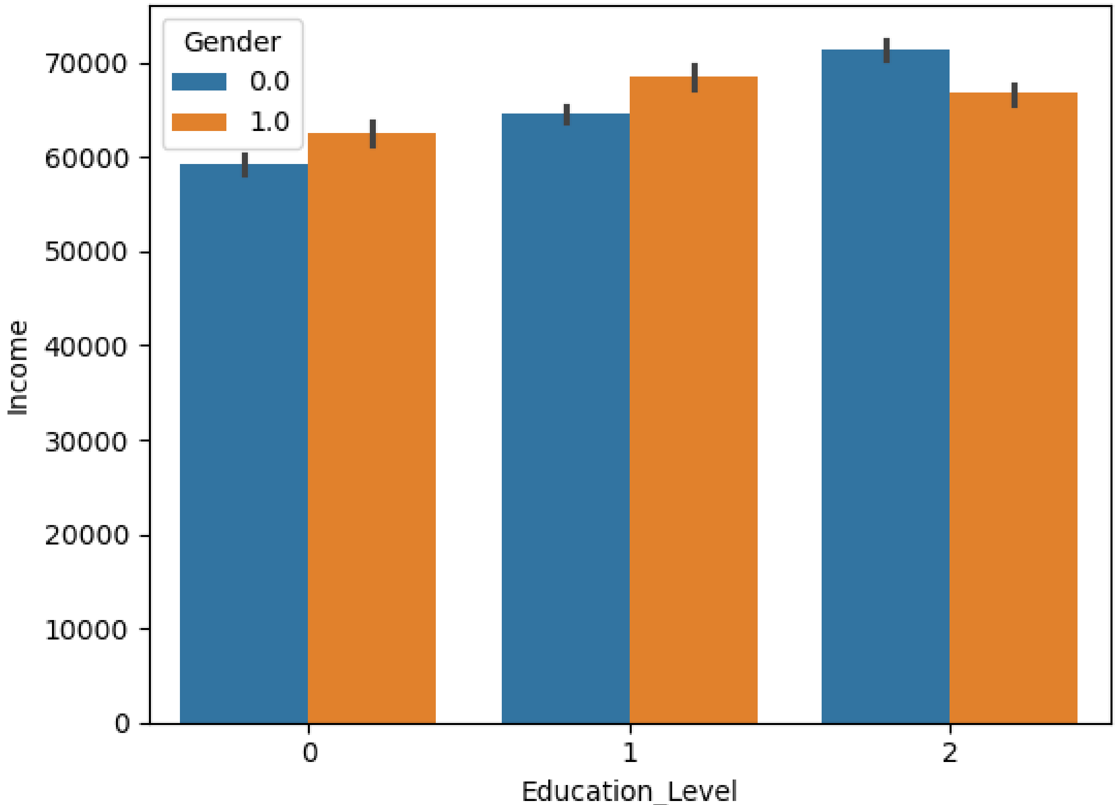

```
data.groupby('City')['Driver_ID'].count().sort_values(ascending=False)
[:10].plot(kind='bar')
plt.title('Top city by driver count')
```

```
Text(0.5, 1.0, 'Top city by driver count')
```

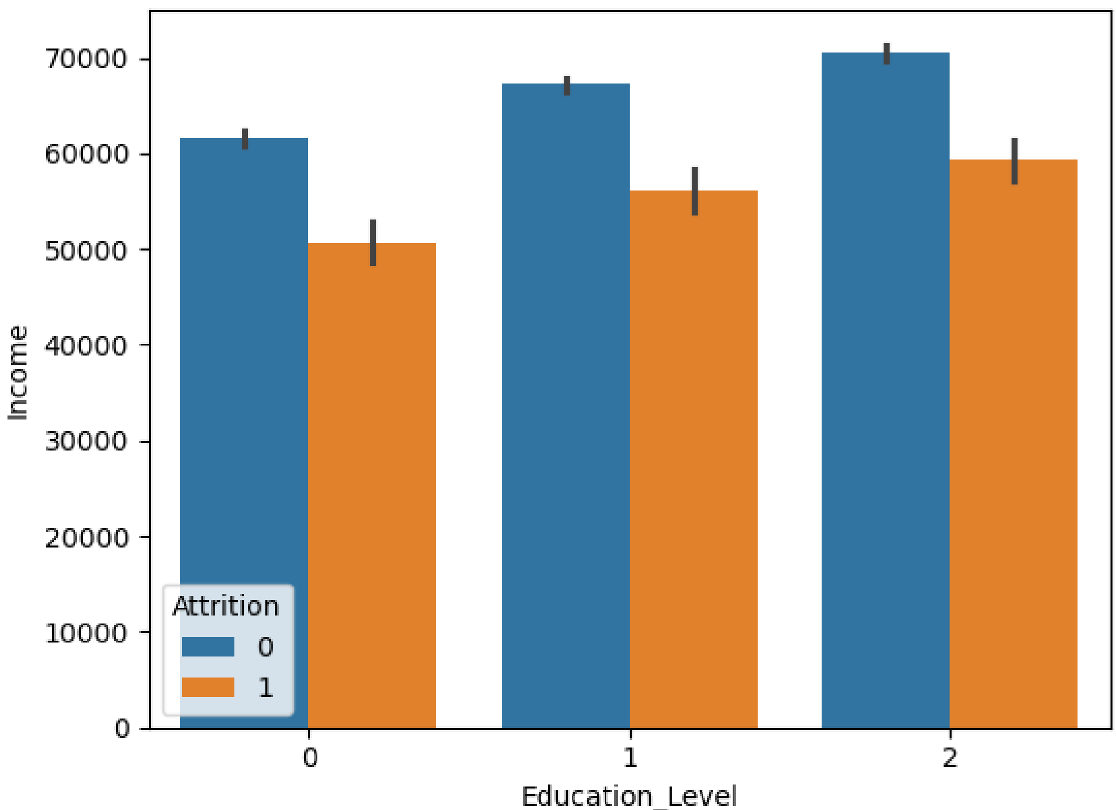



```
#We see that the most attractive cities appearing in each of the 3
graphs are C12, C26, and C29
#So the company should look at putting demand and number of rides to
decide how many drivers to hire in a each city.

sns.barplot(data=data, x='Education_Level', y='Income', hue='Gender')
```

```
<Axes: xlabel='Education_Level', ylabel='Income'>
```




```
sns.barplot(data=data, x='Education_Level',
y='Income',hue='Attrition') #People who left have lower incomes than
people who stayed

<Axes: xlabel='Education_Level', ylabel='Income'>
```




```
#More educated people get higher ratings in general-and in highest education, the men get more income on average than women

data.columns
```

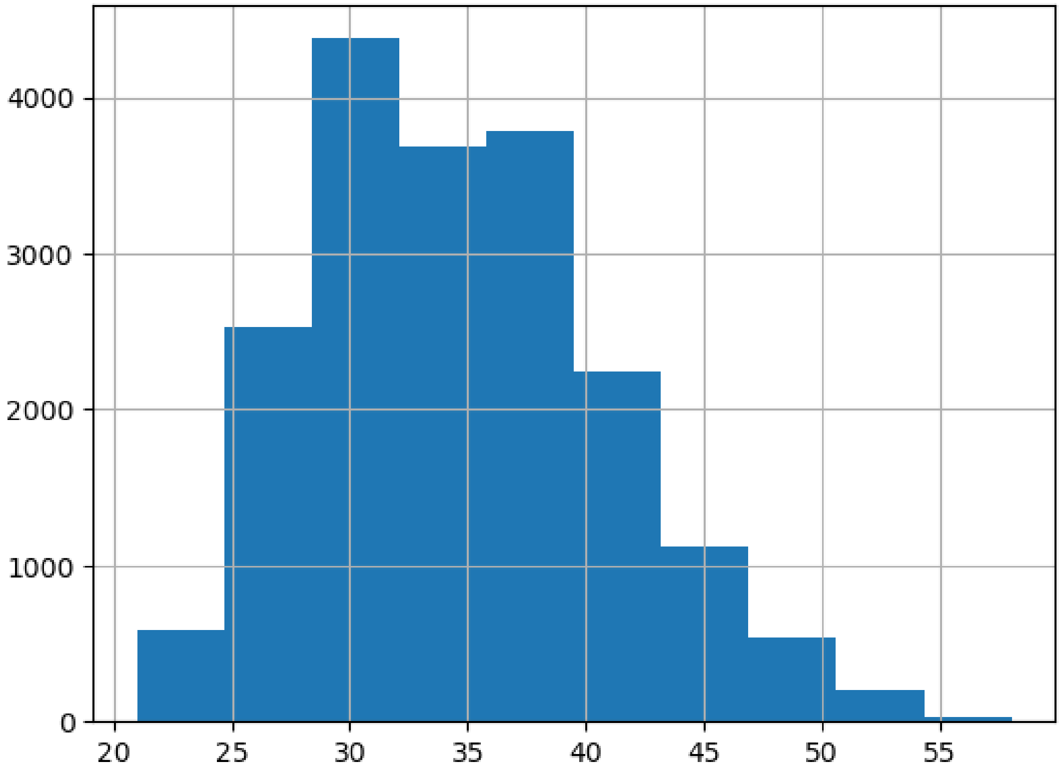
```
Index(['Reporting_date', 'Driver_ID', 'Age', 'Gender', 'City',
       'Education_Level', 'Income', 'Joining Designation', 'Grade',
       'Total Business Value', 'Quarterly Rating', 'Attrition',
       'Dateofjoining', 'LastWorkingDate', 'tenure in days'],
      dtype='object')
```

```
data.groupby('Gender')['Attrition'].value_counts()
```

```
Gender  Attrition
0.0     0            10177
        1              949
1.0     0             7311
        1              667
Name: count, dtype: int64
```

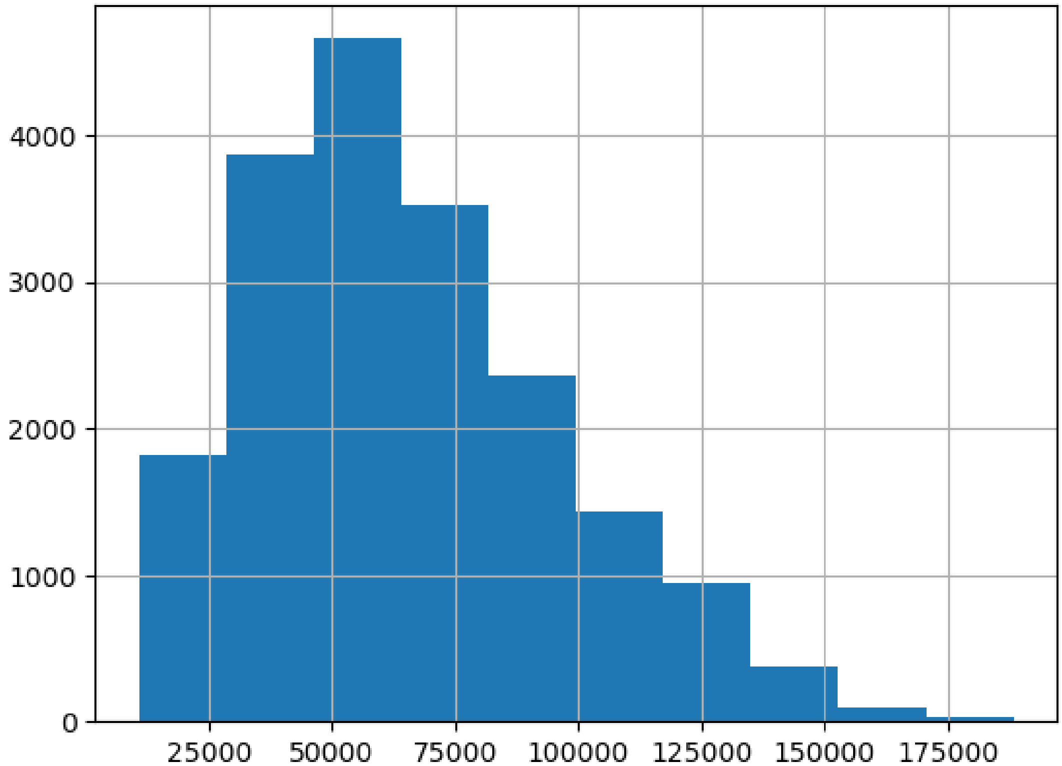
```
data['Age'].hist()
```

```
<Axes: >
```



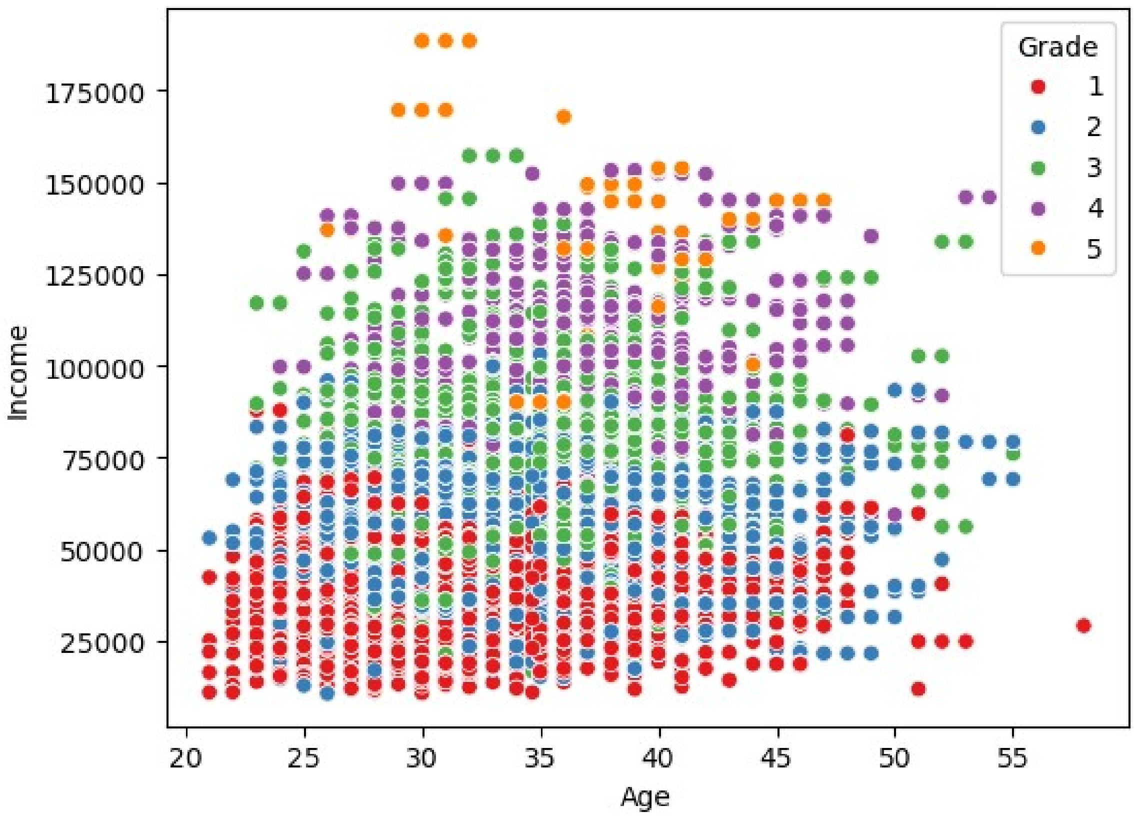

```
data['Income'].hist() #This is a little skewed so I can apply a log transformation, I will leave it as is for now
```

```
<Axes: >
```




```python
sns.scatterplot(data=data, x='Age', y='Income', hue='Grade',
palette='Set1') #We see that age is not correlated with grade, however
it appears income is
```

```
<Axes: xlabel='Age', ylabel='Income'>
```

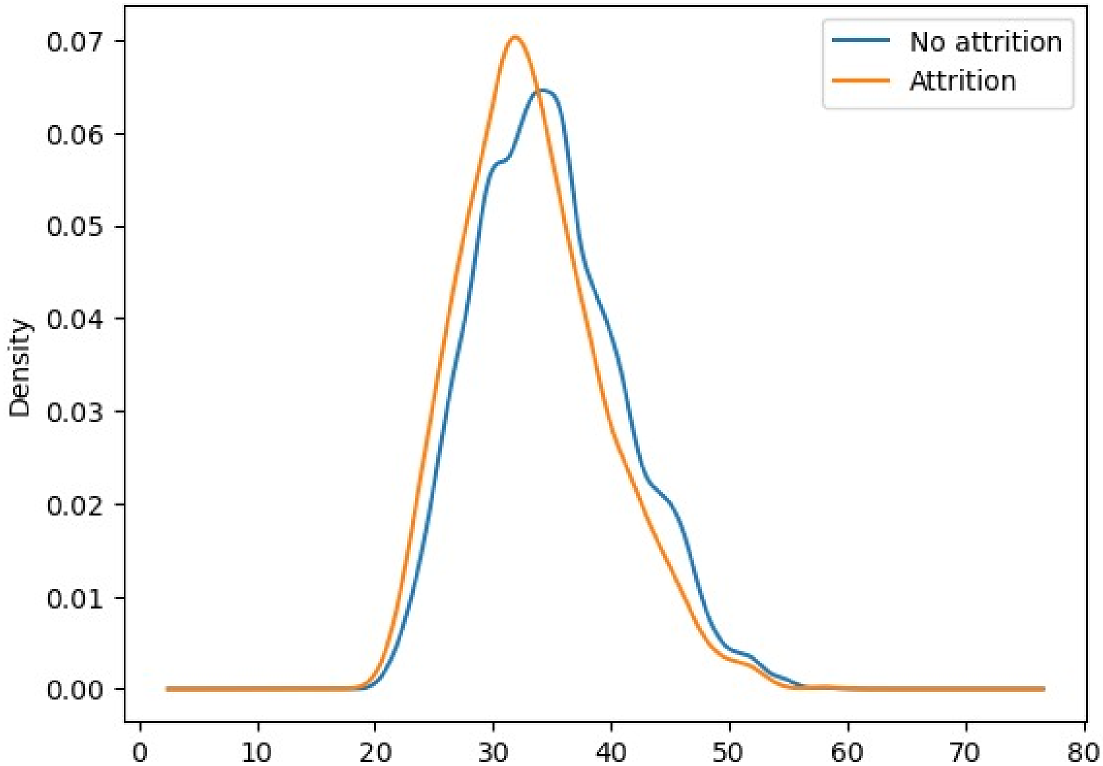



```
data['Attrition'].value_counts()
```

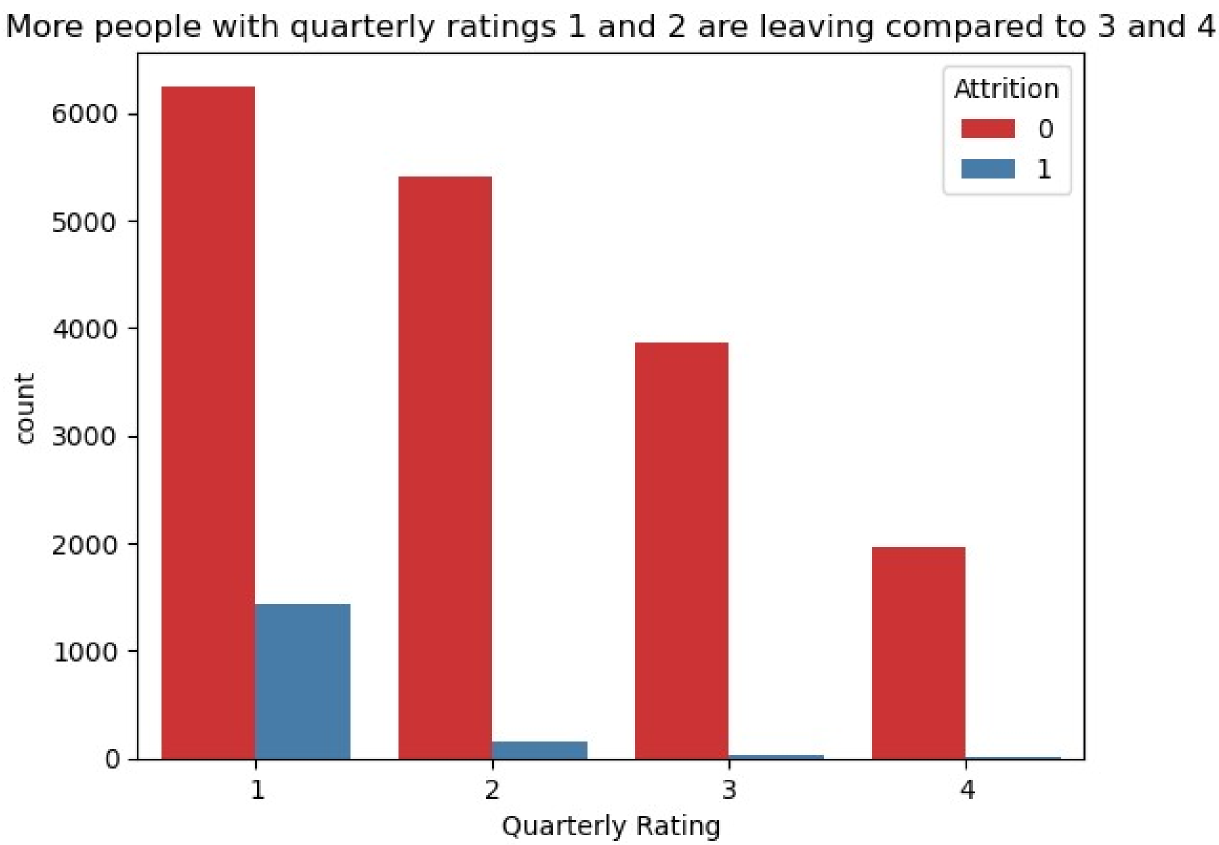
```
Attrition
0    17488
1     1616
Name: count, dtype: int64
```

```
data.groupby('Attrition')['Age'].plot(kind='kde')
plt.legend(['No attrition','Attrition'])
```

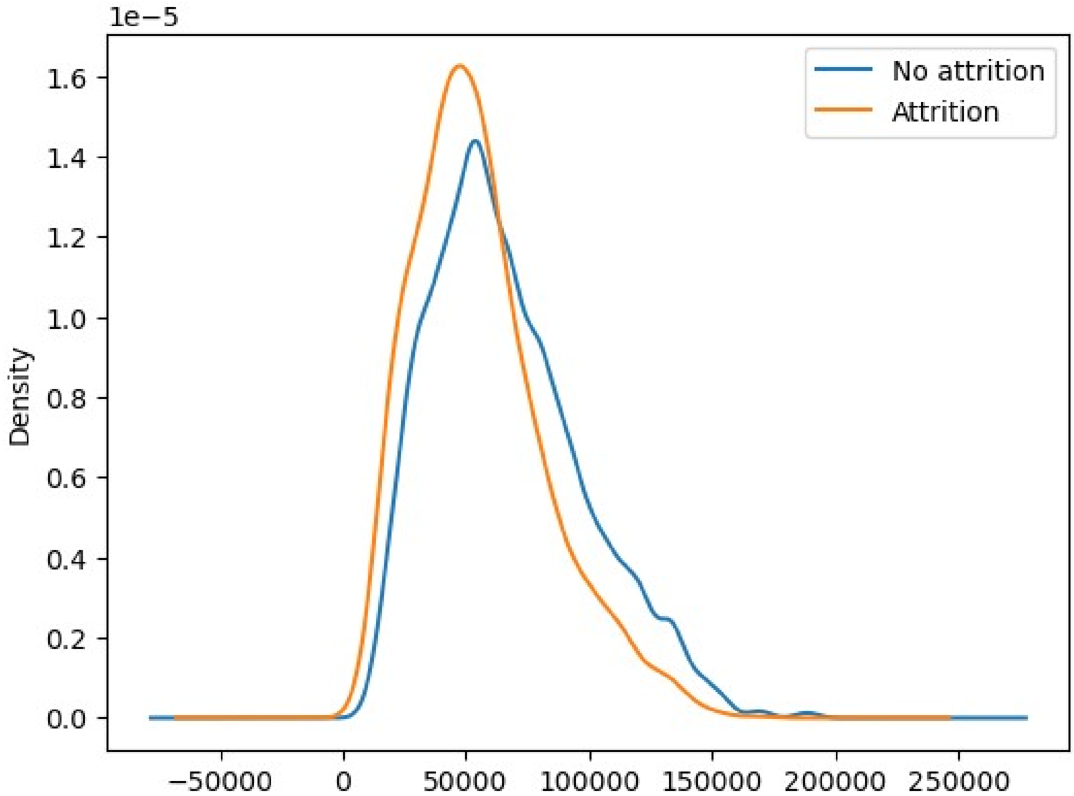
```
<matplotlib.legend.Legend at 0x165969768a0>
```



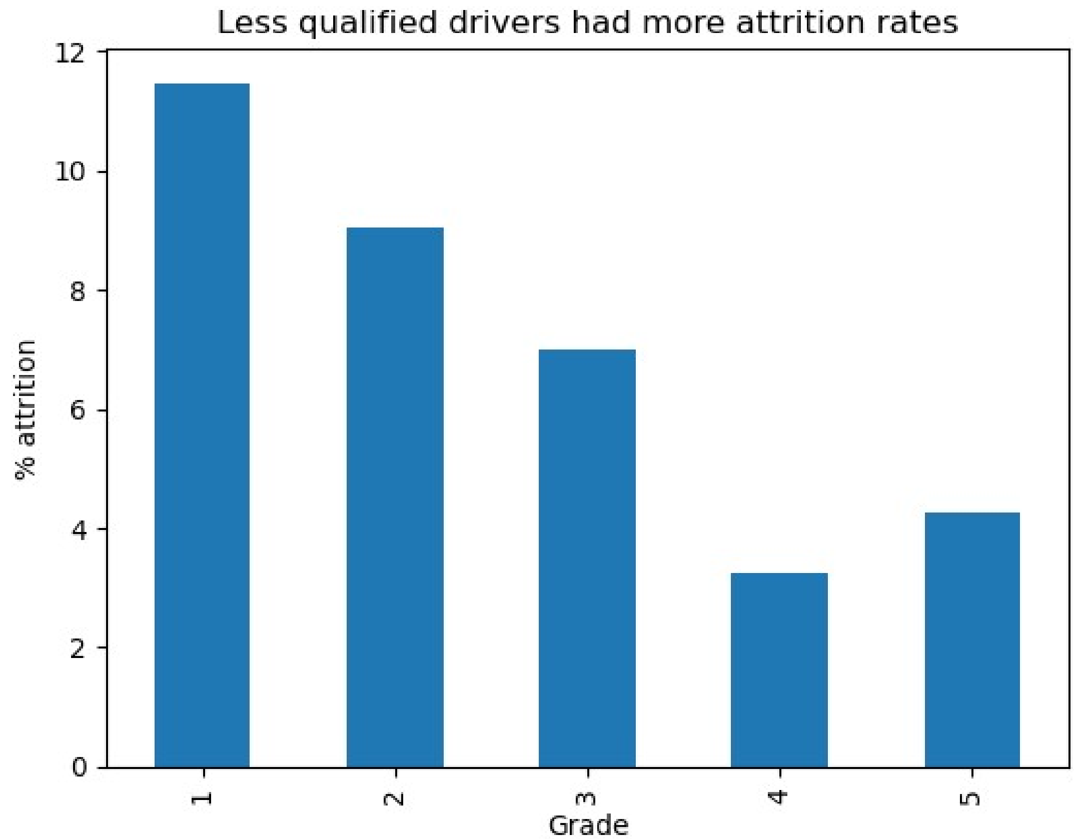

```
sns.countplot(hue='Attrition', x='Quarterly Rating', data=data,
palette='Set1')
plt.title('More people with quarterly ratings 1 and 2 are leaving
compared to 3 and 4')
plt.show()
```



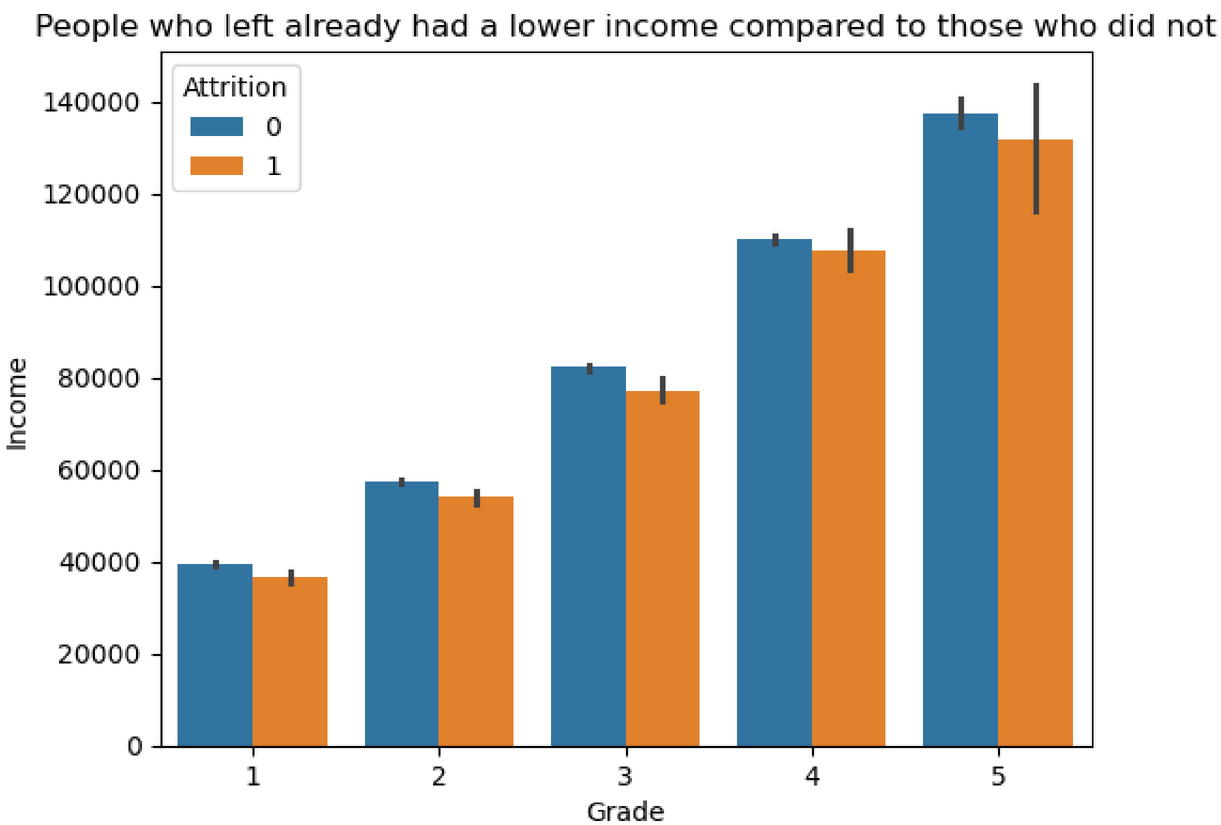

```
sum(data.groupby('Driver_ID')
['Reporting_date'].count().sort_values(ascending=False)==24) #229 of
the drivers reported the maximum duration of 24 days

229

data.groupby('Attrition')['Income'].plot(kind='kde')
plt.legend(['No attrition','Attrition'])
plt.show()
#People who left already had a higher income on average
```



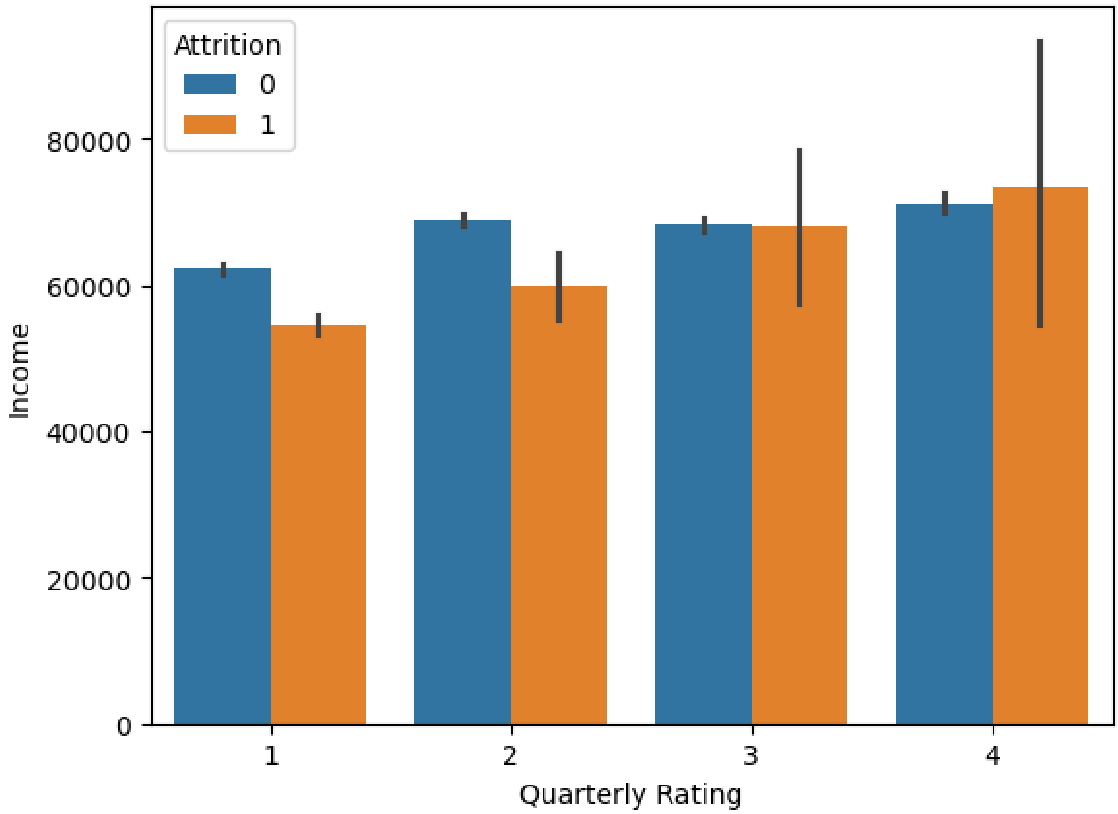

```
(data.groupby('Grade')['Attrition'].sum()*100/data.groupby('Grade')
['Attrition'].count()).plot(kind='bar')
plt.ylabel('% attrition')
plt.title('Less qualified drivers had more attrition rates')
plt.show()
#This is more meaningful than the graph above and shows that attrition
is around 11% for grade 1 and only 4% of grade 5 drivers
```




```
sns.barplot(x='Grade', y='Income', hue='Attrition', data=data)
plt.title('People who left already had a lower income compared to
those who did not')
plt.show()
```




```
sns.barplot(data=data, x='Quarterly Rating', y='Income',
hue='Attrition')

<Axes: xlabel='Quarterly Rating', ylabel='Income'>
```




```
#1. The quarterly rating does not impact the income of the drivers
very significantly, at least among those leaving
#2. For those staying, we see larger error bars, which means the
income cannot be predicted confidently as the quarterly rating
increases
#3. At higher quarterly ratings of 3 or 4, the people who left and the
people who stayed had the same mean income, the only difference is in
the error bars and the confidence of the prediction
#For people in low quarterly ratings, the income of those who left was
lower than those who stayed.
#However, for people with higher quarterly ratings people with similar
or higher income are also leaving
```

```
data['tenure in days']
```

```
0       2341 days
1       2341 days
2         77 days
3       1658 days
```

```
4        1658 days
            ...
19099   1809 days
19100   1809 days
19101   1809 days
19102   1809 days
19103   1809 days
Name: tenure in days, Length: 19104, dtype: timedelta64[ns]
```

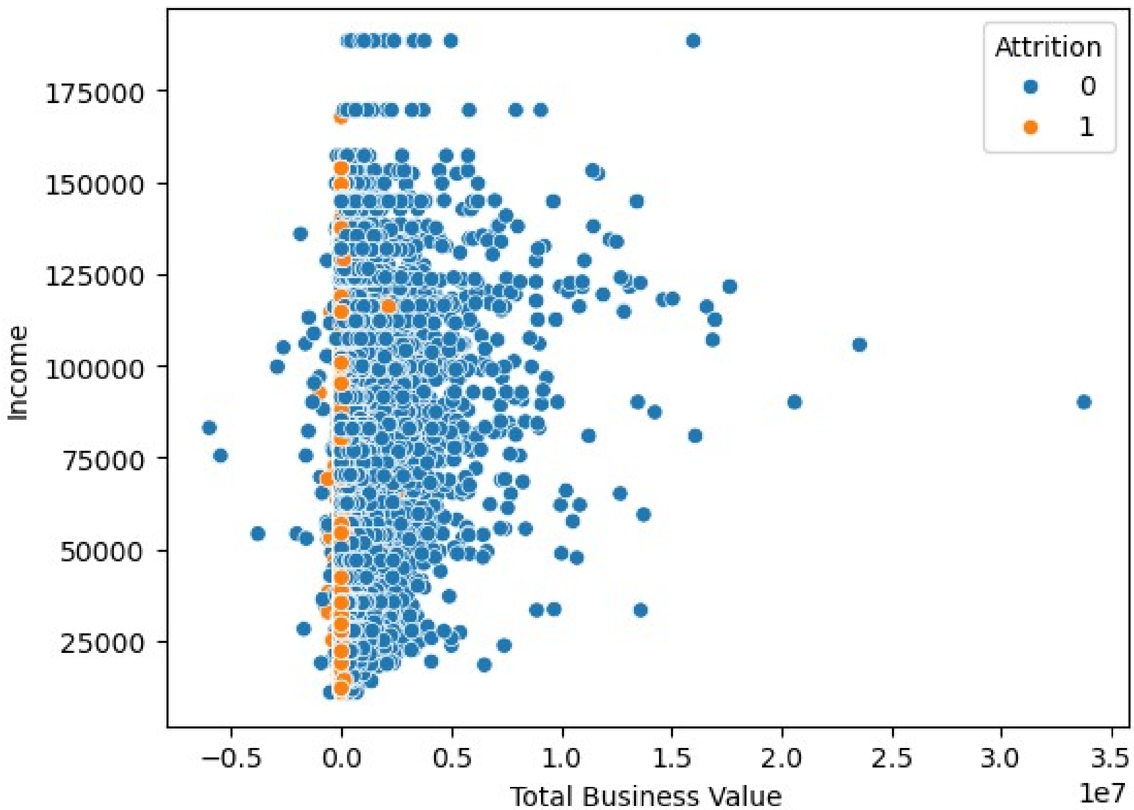
```
data['tenure in days']=data['tenure in days'].apply(str).str.split('
', expand=True)[0]
```

```
data['tenure in days']
```

```
0         2341
1         2341
2           77
3         1658
4         1658
         ...
19099    1809
19100    1809
19101    1809
19102    1809
19103    1809
Name: tenure in days, Length: 19104, dtype: object
```

```
sns.scatterplot(data=data, y='Income', x='Total Business Value',
hue='Attrition')
#Almost all the people who are leaving did not add any business value
```

```
<Axes: xlabel='Total Business Value', ylabel='Income'>
```




```
data.groupby('Attrition')['Total Business Value'].mean() #Those who
left were adding lesser business value

Attrition
0    620681.140782
1     41188.422030
Name: Total Business Value, dtype: float64

#data.groupby('Driver_ID')['Total Business Value'].sum()<=0

#data['Driver_ID'].count()

#(data['Total Business Value']>0).sum()

#data.groupby('Driver_ID')(['Total Business
Value']<=0).sum()/data.groupby('Driver_ID')['Driver_ID'].count()

data.dtypes

Reporting_date          datetime64[ns]
Driver_ID                        int64
Age                            float64
```

```
Gender                                float64
City                                   object
Education_Level                         int64
Income                                  int64
Joining Designation                     int64
Grade                                   int64
Total Business Value                    int64
Quarterly Rating                        int64
Attrition                               int64
Dateofjoining                  datetime64[ns]
LastWorkingDate                datetime64[ns]
tenure in days                         object
dtype: object
```

```
data_copy=data.drop(['Reporting_date',
'Driver_ID','Dateofjoining','LastWorkingDate','tenure in days'],
axis=1)

#I am dropping the tenure as well since it is causing overfitting

data_copy.dtypes
```

```
Age                      float64
Gender                   float64
City                      object
Education_Level            int64
Income                     int64
Joining Designation        int64
Grade                      int64
Total Business Value       int64
Quarterly Rating           int64
Attrition                  int64
dtype: object
```

```
data_copy.City.unique()
```

```
array(['C23', 'C7', 'C13', 'C9', 'C11', 'C2', 'C19', 'C26', 'C20',
'C17',
       'C29', 'C10', 'C24', 'C14', 'C6', 'C28', 'C5', 'C18', 'C27',
'C15',
       'C8', 'C25', 'C21', 'C1', 'C4', 'C3', 'C16', 'C22', 'C12'],
      dtype=object)
```

```
city=pd.get_dummies(data_copy['City'], drop_first=True).astype(int)

city
```

```
       C10  C11  C12  C13  C14  C15  C16  C17  C18  C19  ...  C27  C28
C29  \
0        0    0    0    0    0    0    0    0    0    0  ...    0    0
0
```

```
1          0    0    0    0    0    0    0    0    0    0  ...    0    0
0
2          0    0    0    0    0    0    0    0    0    0  ...    0    0
0
3          0    0    0    0    0    0    0    0    0    0  ...    0    0
0
4          0    0    0    0    0    0    0    0    0    0  ...    0    0
0
...      ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
...
19099      0    0    0    0    0    0    0    0    0    0  ...    1    0
0
19100      0    0    0    0    0    0    0    0    0    0  ...    1    0
0
19101      0    0    0    0    0    0    0    0    0    0  ...    1    0
0
19102      0    0    0    0    0    0    0    0    0    0  ...    1    0
0
19103      0    0    0    0    0    0    0    0    0    0  ...    1    0
0

       C3  C4  C5  C6  C7  C8  C9
0       0   0   0   0   0   0   0
1       0   0   0   0   0   0   0
2       0   0   0   0   0   0   0
3       0   0   0   0   1   0   0
4       0   0   0   0   1   0   0
...    ..  ..  ..  ..  ..  ..  ..
19099   0   0   0   0   0   0   0
19100   0   0   0   0   0   0   0
19101   0   0   0   0   0   0   0
19102   0   0   0   0   0   0   0
19103   0   0   0   0   0   0   0

[19104 rows x 28 columns]

data_copy=pd.concat([data_copy.drop('City', axis=1), city],
axis=1).dropna()



import statsmodels
import statsmodels.api as sm

y=data_copy['Attrition']

x=data_copy.drop('Attrition', axis=1)

x.isnull().sum()
```

```
Age                        0
Gender                     0
Education_Level            0
Income                     0
Joining Designation        0
Grade                      0
Total Business Value       0
Quarterly Rating           0
C10                        0
C11                        0
C12                        0
C13                        0
C14                        0
C15                        0
C16                        0
C17                        0
C18                        0
C19                        0
C2                         0
C20                        0
C21                        0
C22                        0
C23                        0
C24                        0
C25                        0
C26                        0
C27                        0
C28                        0
C29                        0
C3                         0
C4                         0
C5                         0
C6                         0
C7                         0
C8                         0
C9                         0
dtype: int64

import sklearn
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest=train_test_split(x, y, test_size=0.2,
random_state=10)

#xtrain['tenure in days']=xtrain['tenure in days'].astype('int')

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()
```

```
xtrain_num=xtrain[['Age','Income','Total Business Value']]
xtest_num=xtest[['Age', 'Income','Total Business Value']]

xtrain_sc_num=pd.DataFrame(sc.fit_transform(xtrain_num),
columns=['Age','Income','Total Business Value'])
xtest_sc_num=pd.DataFrame(sc.transform(xtest_num),
columns=['Age','Income','Total Business Value'])

xtrain_sc_num.index=xtrain.index
xtest_sc_num.index=xtest.index



xtrain_cat=xtrain.drop(['Age','Income','Total Business Value'],
axis=1)
xtest_cat=xtest.drop(['Age','Income','Total Business Value'], axis=1)



xtrain=pd.concat([xtrain_cat, xtrain_sc_num], axis=1)

xtest=pd.concat([xtest_cat, xtest_sc_num], axis=1)



model_logit=sm.Logit(ytrain, xtrain).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.234459
         Iterations 9
```

```
xtrain.isnull().sum()
```

```
Gender                  0
Education_Level         0
Joining Designation     0
Grade                   0
Quarterly Rating        0
C10                     0
C11                     0
C12                     0
C13                     0
C14                     0
C15                     0
C16                     0
C17                     0
C18                     0
C19                     0
C2                      0
C20                     0
C21                     0
C22                     0
```

```
C23                       0
C24                       0
C25                       0
C26                       0
C27                       0
C28                       0
C29                       0
C3                        0
C4                        0
C5                        0
C6                        0
C7                        0
C8                        0
C9                        0
Age                       0
Income                    0
Total Business Value      0
dtype: int64
```

```
model_logit.summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results

======================================================================
========
Dep. Variable:              Attrition   No. Observations:
15283
Model:                          Logit   Df Residuals:
15247
Method:                           MLE   Df Model:
35
Date:                Thu, 22 May 2025   Pseudo R-squ.:
0.1993
Time:                        16:46:13   Log-Likelihood:
-3583.2
converged:                       True   LL-Null:
-4475.2
Covariance Type:            nonrobust   LLR p-value:
0.000
======================================================================
==================
                           coef    std err          z      P>|z|
[0.025      0.975]
----------------------------------------------------------------------
------------------
Gender                   0.0133      0.062      0.214      0.830
-0.109       0.135
Education_Level         -0.0290      0.038     -0.762      0.446
```

```
-0.103       0.046
Joining Designation      -0.0498      0.061     -0.818      0.413
-0.169       0.070
Grade                    -0.2705      0.064     -4.230      0.000
-0.396      -0.145
Quarterly Rating         -1.4674      0.069    -21.367      0.000
-1.602      -1.333
C10                      -0.3670      0.208     -1.768      0.077
-0.774       0.040
C11                      -0.0880      0.226     -0.390      0.696
-0.530       0.354
C12                      -0.2025      0.203     -0.998      0.318
-0.600       0.195
C13                      -0.0937      0.209     -0.449      0.654
-0.503       0.316
C14                      -0.2617      0.206     -1.273      0.203
-0.665       0.141
C15                      -0.2082      0.192     -1.085      0.278
-0.584       0.168
C16                      -0.4263      0.218     -1.955      0.051
-0.854       0.001
C17                       0.0011      0.210      0.005      0.996
-0.410       0.412
C18                      -0.2960      0.226     -1.308      0.191
-0.739       0.148
C19                      -0.2285      0.225     -1.018      0.309
-0.669       0.211
C2                       -0.0691      0.207     -0.334      0.738
-0.475       0.336
C20                      -0.1539      0.173     -0.891      0.373
-0.493       0.185
C21                      -0.1959      0.215     -0.911      0.362
-0.618       0.226
C22                      -0.4444      0.211     -2.108      0.035
-0.858      -0.031
C23                      -0.1158      0.199     -0.583      0.560
-0.505       0.274
C24                      -0.2013      0.217     -0.928      0.354
-0.627       0.224
C25                      -0.2644      0.207     -1.279      0.201
-0.670       0.141
C26                      -0.3199      0.200     -1.599      0.110
-0.712       0.072
C27                      -0.2372      0.199     -1.189      0.234
-0.628       0.154
C28                      -0.1963      0.203     -0.965      0.334
-0.595       0.202
C29                      -0.3815      0.205     -1.860      0.063
-0.784       0.021
```

```
C3                            -0.3307      0.210     -1.575      0.115
-0.742       0.081
C4                            -0.1993      0.214     -0.931      0.352
-0.619       0.220
C5                            -0.3496      0.208     -1.678      0.093
-0.758       0.059
C6                            -0.0859      0.200     -0.429      0.668
-0.478       0.306
C7                            -0.3379      0.214     -1.576      0.115
-0.758       0.082
C8                            -0.1847      0.208     -0.890      0.374
-0.591       0.222
C9                            -0.1168      0.208     -0.562      0.574
-0.524       0.290
Age                            0.0970      0.034      2.869      0.004
0.031       0.163
Income                         0.0094      0.049      0.193      0.847
-0.086       0.105
Total Business Value    -2.0511      0.174    -11.769      0.000
-2.393      -1.710
======================================================================
==================
"""

#So we see that features such as Gender, Income, Joining designation',
and none of the cities are significant
#Only Age, Grade, Quarterly rating are significant and city C22

#Therefore we should ideally remove all the other features and put
only these

xtest.dtypes

Gender                      float64
Education_Level               int64
Joining Designation           int64
Grade                         int64
Quarterly Rating              int64
C10                           int32
C11                           int32
C12                           int32
C13                           int32
C14                           int32
C15                           int32
C16                           int32
C17                           int32
C18                           int32
C19                           int32
C2                            int32
C20                           int32
```

```
C21                          int32
C22                          int32
C23                          int32
C24                          int32
C25                          int32
C26                          int32
C27                          int32
C28                          int32
C29                          int32
C3                           int32
C4                           int32
C5                           int32
C6                           int32
C7                           int32
C8                           int32
C9                           int32
Age                        float64
Income                     float64
Total Business Value       float64
dtype: object
```

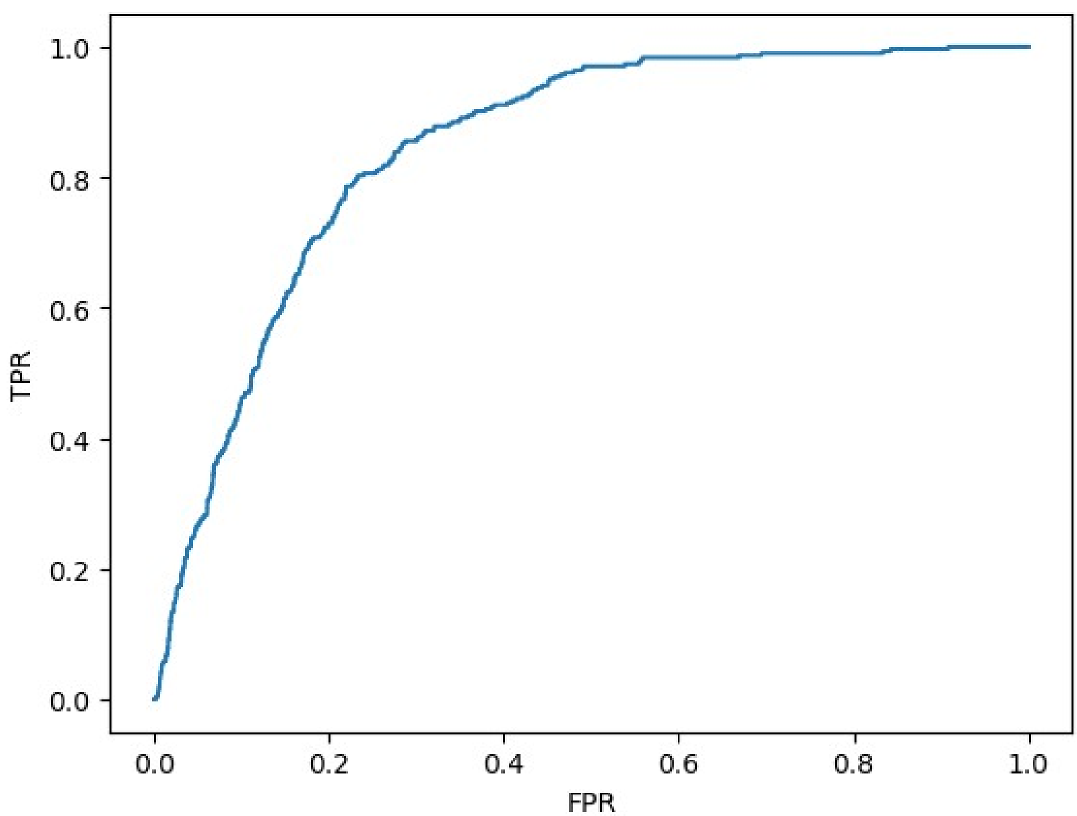
```
#xtest['tenure in days']=xtest['tenure in days'].astype('int')
```

```
ypred=model_logit.predict(xtest)

ypred_bin=[1 if i>=0.5 else 0 for i in ypred]

from sklearn.metrics import classification_report

print(classification_report(ytest, ypred_bin)) #Here we have a perfect recall for the class 0, however, we cannot predict the class 1 accurately at all
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96      3517
           1       0.00      0.00      0.00       304

    accuracy                           0.92      3821
   macro avg       0.46      0.50      0.48      3821
weighted avg       0.85      0.92      0.88      3821
```

```
#trying with a different threshold

ypred_bin_2=[1 if i>=.2 else 0 for i in ypred] #So if we put a threshold of 0.1 we get a better result
```

```
print(classification_report(ytest, ypred_bin_2))

              precision    recall  f1-score   support

           0       0.96      0.86      0.91      3517
           1       0.27      0.58      0.37       304

    accuracy                           0.84      3821
   macro avg       0.61      0.72      0.64      3821
weighted avg       0.90      0.84      0.87      3821


from sklearn.metrics import roc_curve, roc_auc_score

fpr, tpr, thresh=roc_curve(ytest, ypred)

plt.plot(fpr, tpr)
plt.ylabel('TPR')
plt.xlabel('FPR')

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




```
print(roc_auc_score(ytest, ypred)) #So with logit itself the model
gives an 82% score
```

```
0.8442382301004144
```

```python
#Now I am trying with fewer features

x=x[['Age','Grade','Quarterly Rating']]

xtrain, xtest, ytrain, ytest=train_test_split(x, y, test_size=0.2,
random_state=10)

model_logit=sm.Logit(ytrain, xtrain).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.241661
         Iterations 8
```

```python
model_logit.summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results

==============================================================================
========
Dep. Variable:              Attrition   No. Observations:
15283
Model:                          Logit   Df Residuals:
15280
Method:                           MLE   Df Model:
2
Date:                Thu, 22 May 2025   Pseudo R-squ.:
0.1747
Time:                        16:46:14   Log-Likelihood:
-3693.3
converged:                       True   LL-Null:
-4475.2
Covariance Type:            nonrobust   LLR p-value:
0.000
==============================================================================
==============
                       coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
--------------
Age                  0.0268      0.003      8.659      0.000
0.021       0.033
Grade               -0.3080      0.033     -9.355      0.000      -
0.373      -0.243
Quarterly Rating    -1.7667      0.065    -27.083      0.000      -
1.895      -1.639
```

```
==============================================================================
==============
"""

from sklearn.metrics import recall_score, accuracy_score,
classification_report

ypred_logit=model_logit.predict(xtest)

ypred_logit_bin=[1 if i>=0.2 else 0 for i in ypred_logit]

print(classification_report(ytest, ypred_logit_bin))

              precision    recall  f1-score   support

           0       0.94      0.91      0.92      3517
           1       0.25      0.36      0.30       304

    accuracy                           0.86      3821
   macro avg       0.60      0.63      0.61      3821
weighted avg       0.89      0.86      0.87      3821


from sklearn.tree import DecisionTreeClassifier

model_tree=DecisionTreeClassifier(random_state=10)

model_tree.fit(xtrain, ytrain)

DecisionTreeClassifier(random_state=10)

ypred_tree=model_tree.predict(xtest)

print(classification_report(ytest, ypred_tree))

              precision    recall  f1-score   support

           0       0.92      1.00      0.96      3517
           1       0.00      0.00      0.00       304

    accuracy                           0.92      3821
   macro avg       0.46      0.50      0.48      3821
weighted avg       0.85      0.92      0.88      3821


ypred_train=model_tree.predict(xtrain)

print(classification_report(ytrain, ypred_train))
#There is no difference between training and test results, so there is
no need for regularization or bagging

              precision    recall  f1-score   support
```

```
              0       0.91      1.00      0.96     13971
              1       0.70      0.01      0.02      1312

       accuracy                           0.91     15283
      macro avg       0.81      0.51      0.49     15283
   weighted avg       0.90      0.91      0.88     15283
```

```python
#Hence trying to adjust max depth

model_tree_1=DecisionTreeClassifier(max_depth=5) #Trying with a small depth, and that appears to make the class 1 recall completely bad

model_tree_1.fit(xtrain, ytrain)
```

```
DecisionTreeClassifier(max_depth=5)
```

```python
ypred_tree_1=model_tree_1.predict(xtest)
print(classification_report(ytest, ypred_tree_1))
```

```
              precision    recall  f1-score   support

              0       0.92      1.00      0.96      3517
              1       0.00      0.00      0.00       304

       accuracy                           0.92      3821
      macro avg       0.46      0.50      0.48      3821
   weighted avg       0.85      0.92      0.88      3821
```

```python
model_tree_2=DecisionTreeClassifier(max_depth=30) #increasing the depth to see if this gives a better recall for class 1

model_tree_2.fit(xtrain, ytrain)
```

```
DecisionTreeClassifier(max_depth=30)
```

```python
ypred_tree_2=model_tree_2.predict(xtest)

print(classification_report(ytest, ypred_tree_2)) #it still shows very poor results for class 1
```

```
              precision    recall  f1-score   support

              0       0.92      1.00      0.96      3517
              1       0.00      0.00      0.00       304

       accuracy                           0.92      3821
      macro avg       0.46      0.50      0.48      3821
   weighted avg       0.85      0.92      0.88      3821
```

```
#This shows the model needs a very high depth to start classifying the results of class 1 correctly

#Hence trying GradientBoosting

from sklearn.ensemble import GradientBoostingClassifier

model_gb=GradientBoostingClassifier(random_state=10)

model_gb.fit(xtrain, ytrain)
```

```
GradientBoostingClassifier(random_state=10)
```

```
ypred_gb_0=model_gb.predict(xtest)

print(classification_report(ytest, ypred_gb_0)) #Gradient Boosting does not improve it directly, we see we are still getting only 1% recall of class 1
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96      3517
           1       0.00      0.00      0.00       304

    accuracy                           0.92      3821
   macro avg       0.46      0.50      0.48      3821
weighted avg       0.85      0.92      0.88      3821
```

```
ypred_gb_train=model_gb.predict(xtrain)

print(classification_report(ytrain, ypred_gb_train))
#So doing gradient boosting is not helping here, it is underfitting because the values of the 1 class are not being pedicted corectly
```

```
              precision    recall  f1-score   support

           0       0.91      1.00      0.96     13971
           1       0.67      0.01      0.02      1312

    accuracy                           0.91     15283
   macro avg       0.79      0.50      0.49     15283
weighted avg       0.89      0.91      0.87     15283
```

```
#Hence trying to increase the samples using SMOTE

import imblearn

from imblearn.over_sampling import SMOTE

smote=SMOTE()
```

```
xtrain_sm, ytrain_sm=smote.fit_resample(xtrain, ytrain)

model_logit_2=sm.Logit(ytrain_sm, xtrain_sm).fit() #trying logit with
SMOTE
```

```
Optimization terminated successfully.
         Current function value: 0.512970
         Iterations 7
```

```
model_logit_2.summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results

======================================================================
========
Dep. Variable:              Attrition   No. Observations:
27942
Model:                          Logit   Df Residuals:
27939
Method:                           MLE   Df Model:
2
Date:                Thu, 22 May 2025   Pseudo R-squ.:
0.2599
Time:                        16:46:16   Log-Likelihood:
-14333.
converged:                       True   LL-Null:
-19368.
Covariance Type:            nonrobust   LLR p-value:
0.000
======================================================================
==============
                       coef    std err          z      P>|z|
[0.025      0.975]
----------------------------------------------------------------------
--------------
Age                  0.1002      0.002     64.003      0.000
0.097       0.103
Grade               -0.2930      0.016    -18.716      0.000      -
0.324      -0.262
Quarterly Rating    -2.0062      0.029    -68.346      0.000      -
2.064      -1.949
======================================================================
==============
"""
```

```
ypred_logit_2=model_logit_2.predict(xtest)

ypred_bin=[1 if i>=0.5 else 0 for i in ypred_logit_2]
```

```
print(classification_report(ytest, ypred_bin)) #so using smote with logit clearly helps
```

```
              precision    recall  f1-score   support

           0       0.98      0.66      0.79      3517
           1       0.18      0.86      0.29       304

    accuracy                           0.67      3821
   macro avg       0.58      0.76      0.54      3821
weighted avg       0.92      0.67      0.75      3821
```

```
ypred_bin2=[1 if i>=0.1 else 0 for i in ypred_logit_2] #by reducing the threshold, the overall accuracy has dipped

#This gives me a clue that I should try stacking models and put a logisitc regression at the end and lower the threshold

print(classification_report(ytest, ypred_bin2)) #This now gives the highest recall for class 1, this is the overall best result
```

```
              precision    recall  f1-score   support

           0       1.00      0.32      0.48      3517
           1       0.11      0.99      0.20       304

    accuracy                           0.37      3821
   macro avg       0.55      0.65      0.34      3821
weighted avg       0.93      0.37      0.46      3821
```

```
model_tree=DecisionTreeClassifier(random_state=10)

model_tree.fit(xtrain_sm, ytrain_sm)
```

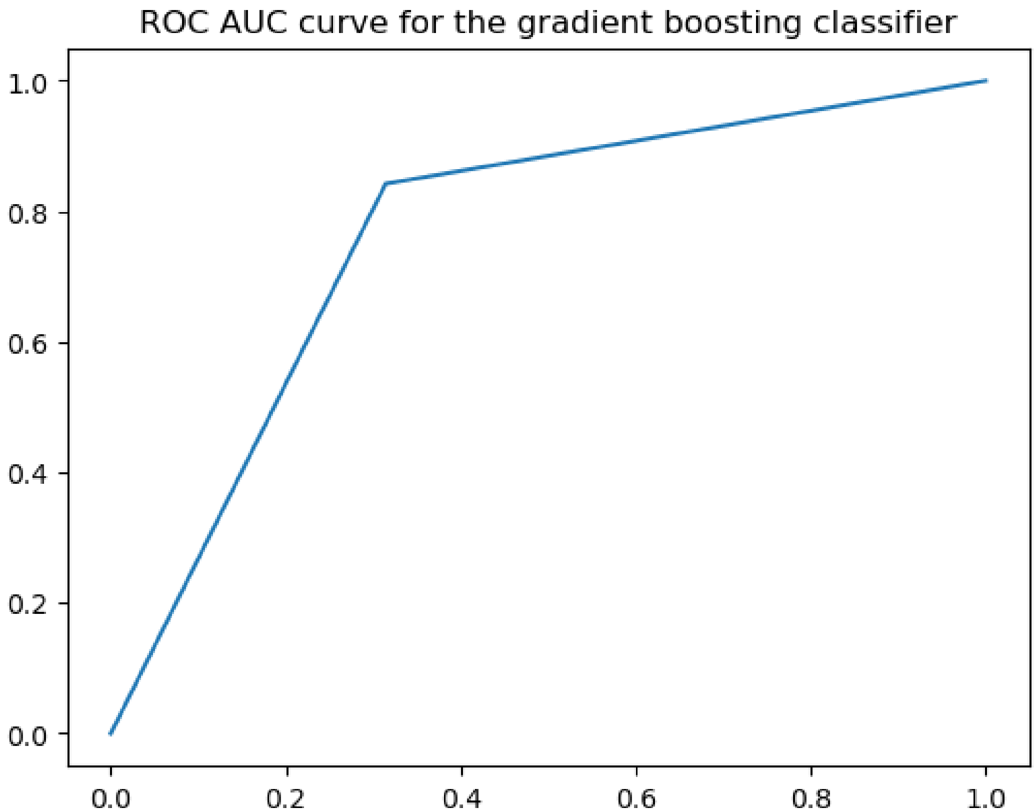
```
DecisionTreeClassifier(random_state=10)
```

```
ypred_tree_sm=model_tree.predict(xtest)

print(classification_report(ytest, ypred_tree_sm))
```

```
              precision    recall  f1-score   support

           0       0.98      0.68      0.80      3517
           1       0.18      0.84      0.30       304

    accuracy                           0.69      3821
   macro avg       0.58      0.76      0.55      3821
weighted avg       0.92      0.69      0.76      3821
```

```
model_gb=GradientBoostingClassifier(random_state=10)

model_gb.fit(xtrain_sm,ytrain_sm)

GradientBoostingClassifier(random_state=10)

ypred_gb2=model_gb.predict(xtest)

print(classification_report(ytest, ypred_gb2))
#using a combination of smote with gradient boosting classifier has
yielded the best results so far

              precision    recall  f1-score   support

           0       0.98      0.68      0.80      3517
           1       0.18      0.85      0.30       304

    accuracy                           0.69      3821
   macro avg       0.58      0.76      0.55      3821
weighted avg       0.92      0.69      0.76      3821


#Can I increase it further and ensure that the recall of both classes
is coming out well

#So I am trying to decrease the number of estimators maybe because
with more estimators in parallel, it is not giving a great accuracy

model_gb3=GradientBoostingClassifier(n_estimators=20, random_state=20)

model_gb3.fit(xtrain_sm, ytrain_sm)

GradientBoostingClassifier(n_estimators=20, random_state=20)

ypred_gb3=model_gb3.predict(xtest)

print(classification_report(ytest, ypred_gb3))

              precision    recall  f1-score   support

           0       0.98      0.65      0.78      3517
           1       0.18      0.88      0.30       304

    accuracy                           0.67      3821
   macro avg       0.58      0.77      0.54      3821
weighted avg       0.92      0.67      0.75      3821


#I then tried reducing learning rate to see if I can increase the
recall of the class 1 further without sacrificing overall accuracy too
much
```

```
model_gb4=GradientBoostingClassifier(n_estimators=10,
learning_rate=.05, random_state=10)

model_gb4.fit(xtrain_sm, ytrain_sm)

GradientBoostingClassifier(learning_rate=0.05, n_estimators=10,
random_state=10)

ypred_gb4=model_gb4.predict(xtest)

print(classification_report(ytest, ypred_gb4))

              precision    recall  f1-score   support

           0       0.98      0.69      0.81      3517
           1       0.19      0.84      0.31       304

    accuracy                           0.70      3821
   macro avg       0.58      0.76      0.56      3821
weighted avg       0.92      0.70      0.77      3821


#then i tried a max depth of 3 and it worked better, reducing depth
below that is not improving anything

model_gb5=GradientBoostingClassifier(n_estimators=10,
learning_rate=0.05, max_depth=3,random_state=10)

model_gb5.fit(xtrain_sm, ytrain_sm)

GradientBoostingClassifier(learning_rate=0.05, n_estimators=10,
random_state=10)

ypred_gb5=model_gb5.predict(xtest)

print(classification_report(ytest, ypred_gb5))

              precision    recall  f1-score   support

           0       0.98      0.69      0.81      3517
           1       0.19      0.84      0.31       304

    accuracy                           0.70      3821
   macro avg       0.58      0.76      0.56      3821
weighted avg       0.92      0.70      0.77      3821


fpr, tpr, thresh=roc_curve(ytest, ypred_gb5)

plt.plot(fpr, tpr)
plt.title('ROC AUC curve for the gradient boosting classifier')

Text(0.5, 1.0, 'ROC AUC curve for the gradient boosting classifier')
```




```
print(roc_auc_score(ytest, ypred_gb5))

0.7639585172769854

#i understand that there is class imbalance, and to address that, one
of the ways is to use stratified kfold sampling
#So I will use stratified kfold sampling on top of the smote data to
see if it imporves results




from sklearn.model_selection import StratifiedKFold

skf=StratifiedKFold(n_splits=10,shuffle=True, random_state=10)

from sklearn.metrics import recall_score, accuracy_score

model_gb5=GradientBoostingClassifier(n_estimators=20,
learning_rate=0.05, max_depth=3,random_state=10)

model_recall_score=[]
model_acc_score=[]
for train_index, test_index in skf.split(x,y):
    xtrain_fold, xtest_fold=x.iloc[train_index], x.iloc[test_index]
```

```
    ytrain_fold, ytest_fold=y.iloc[train_index], y.iloc[test_index]
    xtrain_sm_fold, ytrain_sm_fold=smote.fit_resample(xtrain_fold,
ytrain_fold)
    model_gb5.fit(xtrain_sm_fold, ytrain_sm_fold)
    ypred_gb5=model_gb5.predict(xtest_fold)
    model_recall_score.append(recall_score(ytest_fold, ypred_gb5))
    model_acc_score.append(accuracy_score(ytest_fold, ypred_gb5))

model_recall_score

[0.8580246913580247,
 0.8333333333333334,
 0.8518518518518519,
 0.8333333333333334,
 0.8260869565217391,
 0.84472049689441,
 0.8881987577639752,
 0.8571428571428571,
 0.8209876543209876,
 0.845679012345679]

np.mean(model_recall_score) #this is still not giving great recall
even though for some row sample sets it is crossing 85%

0.8459358944866191

import xgboost

from xgboost import XGBClassifier

model_xgb=XGBClassifier(random_state=10) #The xgboost classifier does
not provide a good recall here

model_recall_score_xgb=[]
for train_index, test_index in skf.split(x,y):
    xtrain_fold, xtest_fold=x.iloc[train_index], x.iloc[test_index]
    ytrain_fold, ytest_fold=y.iloc[train_index], y.iloc[test_index]
    xtrain_sm_fold, ytrain_sm_fold=smote.fit_resample(xtrain_fold,
ytrain_fold)
    model_xgb.fit(xtrain_sm_fold, ytrain_sm_fold)
    ypred_xgb_kf=model_xgb.predict(xtest_fold)
    model_recall_score_xgb.append(recall_score(ytest_fold,
ypred_xgb_kf))

model_recall_score_xgb

[0.8580246913580247,
 0.8641975308641975,
 0.8395061728395061,
 0.8148148148148148,
 0.8385093167701864,
```

```
 0.8509316770186336,
 0.8757763975155279,
 0.8385093167701864,
 0.8148148148148148,
 0.845679012345679]
```

```
from sklearn.ensemble import RandomForestClassifier
model_RF=RandomForestClassifier(max_depth=3, n_estimators=20)

model_recall_score_rf=[]
for train_index, test_index in skf.split(x,y):
    xtrain_fold, xtest_fold=x.iloc[train_index], x.iloc[test_index]
    ytrain_fold, ytest_fold=y.iloc[train_index], y.iloc[test_index]
    xtrain_sm_fold, ytrain_sm_fold=smote.fit_resample(xtrain_fold,
ytrain_fold)
    model_RF.fit(xtrain_sm_fold, ytrain_sm_fold)
    ypred_rf_kf=model_RF.predict(xtest_fold)
    model_recall_score_rf.append(recall_score(ytest_fold,
ypred_rf_kf))

model_recall_score_rf
```

```
[0.8580246913580247,
 0.8333333333333334,
 0.8518518518518519,
 0.8333333333333334,
 0.8260869565217391,
 0.84472049689441,
 0.8819875776397516,
 0.84472049689441,
 0.8024691358024691,
 0.845679012345679]
```

```
model_RF.fit(xtrain_sm, ytrain_sm)
```

```
RandomForestClassifier(max_depth=3, n_estimators=20)
```

```
ypred_rf=model_RF.predict(xtest)

print(classification_report(ytest, ypred_rf)) #this works because we
have only 3 features
```

```
              precision    recall  f1-score   support

           0       0.98      0.69      0.81      3517
           1       0.19      0.84      0.31       304

    accuracy                           0.70      3821
```

```
   macro avg       0.58      0.76      0.56      3821
weighted avg       0.92      0.70      0.77      3821
```

```
#pip install xgboost
```

```
from xgboost import XGBClassifier

model_xgb=XGBClassifier()

model_xgb.fit(xtrain, ytrain)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None,
early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None,
feature_types=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None,
max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None,
max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None,
n_estimators=None,
              n_jobs=None, num_parallel_tree=None, ...)
```

```
ypred_xgb_1=model_xgb.predict(xtest)

print(classification_report(ytest, ypred_xgb_1))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96      3517
           1       0.00      0.00      0.00       304

    accuracy                           0.92      3821
   macro avg       0.46      0.50      0.48      3821
weighted avg       0.85      0.92      0.88      3821
```

```
model_xgb2=XGBClassifier()

model_xgb2.fit(xtrain_sm, ytrain_sm)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None,
```

```
early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None,
feature_types=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None,
max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None,
max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None,
n_estimators=None,
              n_jobs=None, num_parallel_tree=None, ...)
```

```
ypred_xgb2=model_xgb2.predict(xtest)

print(classification_report(ytest, ypred_xgb2)) #We see that xgboost
does not address my recall problem with class 1, it only increases
overall accuracy
```

```
              precision    recall  f1-score   support

           0       0.98      0.67      0.80      3517
           1       0.18      0.85      0.30       304

    accuracy                           0.69      3821
   macro avg       0.58      0.76      0.55      3821
weighted avg       0.92      0.69      0.76      3821
```

```
from sklearn.ensemble import AdaBoostClassifier

model_adb=AdaBoostClassifier()

model_adb.fit(xtrain_sm, ytrain_sm)
```

```
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(

AdaBoostClassifier()
```

```
ypred_adb=model_adb.predict(xtest)

print(classification_report(ytest, ypred_adb))
```

```
              precision    recall  f1-score   support
```

```
              0       0.98      0.69      0.81      3517
              1       0.19      0.84      0.31       304

       accuracy                           0.70      3821
      macro avg       0.58      0.76      0.56      3821
   weighted avg       0.92      0.70      0.77      3821
```

```
model_adb=AdaBoostClassifier(n_estimators=10, learning_rate=.05,
random_state=10)
```

```
model_adb.fit(xtrain_sm, ytrain_sm)
```

```
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
```

```
AdaBoostClassifier(learning_rate=0.05, n_estimators=10,
random_state=10)
```

```
ypred_adb_2=model_adb.predict(xtest)
```

```
print(classification_report(ytest, ypred_adb_2))
```

```
              precision    recall  f1-score   support

              0       0.99      0.64      0.78      3517
              1       0.18      0.89      0.30       304

       accuracy                           0.66      3821
      macro avg       0.58      0.77      0.54      3821
   weighted avg       0.92      0.66      0.74      3821
```

```
model_recall_score_adb=[]
model_acc_score=[]
for train_index, test_index in skf.split(x,y):
    xtrain_fold, xtest_fold=x.iloc[train_index], x.iloc[test_index]
    ytrain_fold, ytest_fold=y.iloc[train_index], y.iloc[test_index]
    xtrain_sm_fold, ytrain_sm_fold=smote.fit_resample(xtrain_fold,
ytrain_fold)
    model_adb.fit(xtrain_sm_fold, ytrain_sm_fold)
    ypred_adb=model_adb.predict(xtest_fold)
    model_recall_score.append(recall_score(ytest_fold, ypred_adb))
    model_acc_score.append(accuracy_score(ytest_fold, ypred_adb))
```

```
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
```

```
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

```python
np.mean(model_recall_score) #Adaboost classifier recall score increased from 42% to 60% when I trained it using CV
```

```
0.8660474656851468

#Therefore the idea to increase the accuracy and recall is to use
RepeatedKFold

#I am putting a training a GBM model using RepeatedKFold below

from sklearn.model_selection import RepeatedKFold

rkf=RepeatedKFold(n_splits=5,n_repeats=10,random_state=10)

model_recall_score=[]
model_acc_score=[]
for train_index, test_index in rkf.split(x,y):
    xtrain_fold_rf,
xtest_fold_rf=x.iloc[train_index],x.iloc[test_index]
    ytrain_fold_rf, ytest_fold_rf=y.iloc[train_index],
y.iloc[test_index]
    xtrain_sm_rf, ytrain_sm_rf=smote.fit_resample(xtrain_fold_rf,
ytrain_fold_rf)
    model_gb5.fit(xtrain_sm_rf, ytrain_sm_rf)
    ypred_sm_rf=model_gb5.predict(xtest_fold_rf)
    model_recall_score.append(recall_score(ytest_fold_rf,
ypred_sm_rf))

np.mean(model_recall_score) #This is still not giving a better result
than just using a GBM model

0.8479543828019207



from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import StackingClassifier

stk_clf=StackingClassifier(estimators=[('ADB',model_adb),
('GBM',model_gb5)], final_estimator=LogisticRegression())

stk_clf

StackingClassifier(estimators=[('ADB',
                                AdaBoostClassifier(learning_rate=0.05,
                                                   n_estimators=10,
                                                   random_state=10)),
                               ('GBM',

GradientBoostingClassifier(learning_rate=0.05,

n_estimators=20,

random_state=10))],
                   final_estimator=LogisticRegression())
```

```
stk_clf.fit(xtrain_sm, ytrain_sm)
```

```
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
```

```
StackingClassifier(estimators=[('ADB',
                                AdaBoostClassifier(learning_rate=0.05,
                                                   n_estimators=10,
                                                   random_state=10)),
                               ('GBM',

GradientBoostingClassifier(learning_rate=0.05,

n_estimators=20,

random_state=10))],
                   final_estimator=LogisticRegression())
```

```
ypred_stk=stk_clf.predict(xtest)
```

```
print(classification_report(ytest, ypred_stk))
```

```
              precision    recall  f1-score   support

           0       0.98      0.69      0.81      3517
           1       0.19      0.84      0.31       304

    accuracy                           0.70      3821
   macro avg       0.58      0.76      0.56      3821
weighted avg       0.92      0.70      0.77      3821
```

```python
ypred_stk_proba=stk_clf.predict_proba(xtest)

ypred_stk_proba=ypred_stk_proba[:,1] #take only probability of 1 class

ypred_stk_bin=[1 if i>=0.1 else 0 for i in ypred_stk_proba]

print(classification_report(ytest, ypred_stk_bin)) #this is the second
best result
```

```
              precision    recall  f1-score   support

           0       0.99      0.46      0.63      3517
           1       0.13      0.95      0.23       304

    accuracy                           0.50      3821
   macro avg       0.56      0.71      0.43      3821
weighted avg       0.92      0.50      0.60      3821
```

```python
pip install LightGBM
```

```
Requirement already satisfied: LightGBM in e:\python\lib\site-packages
(4.6.0)
Requirement already satisfied: numpy>=1.17.0 in e:\python\lib\site-
packages (from LightGBM) (1.26.4)
Requirement already satisfied: scipy in e:\python\lib\site-packages
(from LightGBM) (1.13.1)
Note: you may need to restart the kernel to use updated packages.
```

```python
import lightgbm as lgb

model_lgb=lgb.LGBMClassifier(n_estimators=10, learning_rate=.05)

model_lgb.fit(xtrain_sm, ytrain_sm)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Number of positive: 13971, number of negative: 13971
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000904 seconds.
You can set `force_col_wise=true` to remove the overhead.
```

```
[LightGBM] [Info] Total Bins 208
[LightGBM] [Info] Number of data points in the train set: 27942,
number of used features: 3
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 ->
initscore=0.000000
```

```
LGBMClassifier(learning_rate=0.05, n_estimators=10)
```

```
ypred_lgb=model_lgb.predict(xtest)
```

```
print(classification_report(ytest, ypred_lgb))
```

```
              precision    recall  f1-score   support

           0       0.98      0.68      0.80      3517
           1       0.18      0.85      0.30       304

    accuracy                           0.69      3821
   macro avg       0.58      0.76      0.55      3821
weighted avg       0.92      0.69      0.76      3821
```

```
model_stk_2=StackingClassifier(estimators=[('lgb', model_lgb),
('ADB',model_adb), ('GBM', model_gb5)],
final_estimator=LogisticRegression())
```

```
model_stk_2.fit(xtrain_sm, ytrain_sm)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Number of positive: 13971, number of negative: 13971
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000163 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 208
[LightGBM] [Info] Number of data points in the train set: 27942,
number of used features: 3
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 ->
initscore=0.000000
```

```
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Number of positive: 11177, number of negative: 11176
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.011635 seconds.
```

```
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 218
[LightGBM] [Info] Number of data points in the train set: 22353,
number of used features: 3
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500022 ->
initscore=0.000089
[LightGBM] [Info] Start training from score 0.000089
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Number of positive: 11176, number of negative: 11177
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000248 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 212
[LightGBM] [Info] Number of data points in the train set: 22353,
number of used features: 3
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499978 ->
initscore=-0.000089
[LightGBM] [Info] Start training from score -0.000089
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Number of positive: 11177, number of negative: 11177
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000242 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 212
[LightGBM] [Info] Number of data points in the train set: 22354,
number of used features: 3
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 ->
initscore=0.000000
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Number of positive: 11177, number of negative: 11177
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000245 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 205
[LightGBM] [Info] Number of data points in the train set: 22354,
number of used features: 3
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 ->
initscore=0.000000
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Number of positive: 11177, number of negative: 11177
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000205 seconds.
```

```
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 207
[LightGBM] [Info] Number of data points in the train set: 22354,
number of used features: 3
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 ->
initscore=0.000000

E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(

StackingClassifier(estimators=[('lgb',
                                LGBMClassifier(learning_rate=0.05,
                                               n_estimators=10)),
                               ('ADB',
                                AdaBoostClassifier(learning_rate=0.05,
                                                   n_estimators=10,
                                                   random_state=10)),
                               ('GBM',

GradientBoostingClassifier(learning_rate=0.05,

n_estimators=20,

random_state=10))],
                   final_estimator=LogisticRegression())
```

```python
ypred_stk_2=model_stk_2.predict(xtest)

print(classification_report(ytest, ypred_stk_2))
```

```
              precision    recall  f1-score   support

           0       0.98      0.69      0.81      3517
           1       0.19      0.84      0.31       304

    accuracy                           0.70      3821
   macro avg       0.58      0.76      0.56      3821
weighted avg       0.92      0.70      0.77      3821
```

```python
ypred_probla_stk_2=model_stk_2.predict_proba(xtest)[:,1]

ypred_probla_stk_2
```

```
array([0.06812235, 0.72639533, 0.09733343, ..., 0.67249155,
0.06733472,
       0.05501743])
```

```python
ypred_bin_stk_2=[1 if i>=.1 else 0 for i in ypred_probla_stk_2]

print(classification_report(ytest, ypred_bin_stk_2)) #so this gives me
a slightly better overall accuracy
```

```
              precision    recall  f1-score   support

           0       0.99      0.53      0.69      3517
           1       0.14      0.92      0.25       304

    accuracy                           0.56      3821
   macro avg       0.57      0.73      0.47      3821
weighted avg       0.92      0.56      0.65      3821
```

```
pip install catboost
```

```
Requirement already satisfied: catboost in e:\python\lib\site-packages
(1.2.8)
Requirement already satisfied: graphviz in e:\python\lib\site-packages
(from catboost) (0.20.3)
Requirement already satisfied: matplotlib in e:\python\lib\site-
packages (from catboost) (3.9.2)
Requirement already satisfied: numpy<3.0,>=1.16.0 in e:\python\lib\
site-packages (from catboost) (1.26.4)
Requirement already satisfied: pandas>=0.24 in e:\python\lib\site-
packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in e:\python\lib\site-packages
```

```
(from catboost) (1.13.1)
Requirement already satisfied: plotly in e:\python\lib\site-packages
(from catboost) (5.24.1)
Requirement already satisfied: six in e:\python\lib\site-packages
(from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in e:\python\
lib\site-packages (from pandas>=0.24->catboost) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in e:\python\lib\site-
packages (from pandas>=0.24->catboost) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in e:\python\lib\site-
packages (from pandas>=0.24->catboost) (2023.3)
Requirement already satisfied: contourpy>=1.0.1 in e:\python\lib\site-
packages (from matplotlib->catboost) (1.2.0)
Requirement already satisfied: cycler>=0.10 in e:\python\lib\site-
packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in e:\python\lib\
site-packages (from matplotlib->catboost) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in e:\python\lib\
site-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: packaging>=20.0 in e:\python\lib\site-
packages (from matplotlib->catboost) (24.1)
Requirement already satisfied: pillow>=8 in e:\python\lib\site-
packages (from matplotlib->catboost) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in e:\python\lib\site-
packages (from matplotlib->catboost) (3.1.2)
Requirement already satisfied: tenacity>=6.2.0 in e:\python\lib\site-
packages (from plotly->catboost) (8.2.3)
Note: you may need to restart the kernel to use updated packages.
```

```python
import catboost

from catboost import CatBoostClassifier, Pool

model_catboost=CatBoostClassifier(learning_rate=0.01, depth=3,
iterations=2) #this works well when I simplify it by reducing the
depth and complexity

model_catboost.fit(xtrain_sm, ytrain_sm)
```

```
0:	learn: 0.6895776	total: 4.05ms	remaining: 4.05ms
1:	learn: 0.6859223	total: 8.77ms	remaining: 0us
```

```
<catboost.core.CatBoostClassifier at 0x1659b738ad0>
```

```python
ypred_cat=model_catboost.predict(xtest)

print(classification_report(ytest, ypred_cat))
```

```
              precision    recall  f1-score   support

           0       0.99      0.64      0.78      3517
```

```
           1       0.18      0.89      0.30       304

    accuracy                           0.66      3821
   macro avg       0.58      0.77      0.54      3821
weighted avg       0.92      0.66      0.74      3821
```

```
model_stk_3=StackingClassifier(estimators=[('catboost',model_catboost)
,('ADB',model_adb),('GBM',model_gb5)],
final_estimator=LogisticRegression())
```

```
model_stk_3.fit(xtrain_sm, ytrain_sm)
```

```
0:	learn: 0.6895776	total: 4.76ms	remaining: 4.76ms
1:	learn: 0.6859223	total: 9.39ms	remaining: 0us
```

```
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
```

```
0:	learn: 0.6894442	total: 4.89ms	remaining: 4.89ms
1:	learn: 0.6856558	total: 9.95ms	remaining: 0us
0:	learn: 0.6896024	total: 5.25ms	remaining: 5.25ms
1:	learn: 0.6859776	total: 10.1ms	remaining: 0us
0:	learn: 0.6896491	total: 4.66ms	remaining: 4.66ms
1:	learn: 0.6860893	total: 10.3ms	remaining: 0us
0:	learn: 0.6895973	total: 7.5ms	remaining: 7.5ms
1:	learn: 0.6859621	total: 16.3ms	remaining: 0us
0:	learn: 0.6895842	total: 4.9ms	remaining: 4.9ms
1:	learn: 0.6859574	total: 9.64ms	remaining: 0us
```

```
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
```

```
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(
E:\python\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and
will be removed in 1.6. Use the SAMME algorithm to circumvent this
warning.
  warnings.warn(

StackingClassifier(estimators=[('catboost',
                                <catboost.core.CatBoostClassifier
object at 0x000001659B738AD0>),
                               ('ADB',
                                AdaBoostClassifier(learning_rate=0.05,
                                                   n_estimators=10,
                                                   random_state=10)),
                               ('GBM',

GradientBoostingClassifier(learning_rate=0.05,

n_estimators=20,

random_state=10))],
                   final_estimator=LogisticRegression())

ypred_stk_3=model_stk_3.predict(xtest)

print(classification_report(ytest, ypred_stk_3))

              precision    recall  f1-score   support

           0       0.98      0.69      0.81      3517
           1       0.19      0.84      0.31       304

    accuracy                           0.70      3821
   macro avg       0.58      0.76      0.56      3821
weighted avg       0.92      0.70      0.77      3821


ypred_proba_stk_3=model_stk_3.predict_proba(xtest)[:,1]

ypred_bin_stk_3=[1 if i>=0.1 else 0 for i in ypred_proba_stk_3]

print(classification_report(ytest, ypred_bin_stk_3)) #this still gives
the same result as using just 2 algorithms

              precision    recall  f1-score   support

           0       0.99      0.46      0.63      3517
           1       0.13      0.95      0.23       304
```

```
    accuracy                           0.50      3821
   macro avg       0.56      0.71      0.43      3821
weighted avg       0.92      0.50      0.60      3821


# conclusions
#Those who are leaving have a lower income, lower quarterly ratings,
and are adding lower business value than those who are not
# The cities where more drivers are reporting are different from the
cities where the mean income is highest.
#   There is an opportunity for OLA to focus only on these 3 cities
because business value from each driver, income for each driver, as
well as driver availability are all high

# Only 3 features are significant in determining driver attrition :
Age, Grade, and Quarterly Rating
# Keeping only these 3 features produces better results as keeping all
the features


#1. Our data is imbalanced, and the use of SMOTE works well
#2. Bagging techniques like RandomForest as well as models with strong
regularization features such as xgboost are not working well in our
case.
#3. Boosting through Gradient Boosting or AdaBoost or LightGBM works
well in predicting the recall of the target class-drivers who are
leaving
#4. Increasing the complexity of the model or the number of estimators
is not working out here
#our model works well when there are very few layers in the tree, when
we use very few estimators, and when we lower the threshold in the
binary classification model
#5. In our case, it is more important to predict whether a driver will
leave, because it is more expensive to hire new drivers than retain
them.
#The side effect of that is low precision, whenever in doubt, we end
uo predicting that a person will leave, so only 10-15% of our
predictions are true that way
#Which means that our cost of incentives to existing drivers will go
up in general as the cost of hiring new drivers will come down.

#However the best result is coming when we take a logistic regression
model and use smote and then lower the threshold to 0.1 or lower
```