Problem Statement

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

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

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

Demographics (city, age, gender etc.)

Tenure information (joining date, Last Date)

Historical data regarding the performance of the driver (Quarterly rating, Monthly business acquired, grade, Income)

In [3]:
```
import os
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

In [4]:
```
data = pd.read_csv('OLA_Business_Case.csv')
```

In [5]:
```
data.head()
```

Out[5]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining
0	0	01-01-2019	1	28.0	0.0	C23	2	57387	24-12-2018
1	1	02-01-2019	1	28.0	0.0	C23	2	57387	24-12-2018
2	2	03-01-2019	1	28.0	0.0	C23	2	57387	24-12-2018
3	3	11-01-2020	2	31.0	0.0	C7	2	67016	11-06-2020
4	4	12-01-2020	2	31.0	0.0	C7	2	67016	11-06-2020

In [6]:
```
data.shape
```

Out[6]: (19104, 14)

In [7]:
```
data = data.drop(columns = 'Unnamed: 0')
```

In [8]:
```
data
```

Out[8]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWc
0	01-01-2019	1	28.0	0.0	C23	2	57387	24-12-2018	
1	02-01-2019	1	28.0	0.0	C23	2	57387	24-12-2018	
2	03-01-2019	1	28.0	0.0	C23	2	57387	24-12-2018	(
3	11-01-2020	2	31.0	0.0	C7	2	67016	11-06-2020	
4	12-01-2020	2	31.0	0.0	C7	2	67016	11-06-2020	
...	...	...	...	...	...	...	...	...	
19099	08-01-2020	2788	30.0	0.0	C27	2	70254	06-08-2020	
19100	09-01-2020	2788	30.0	0.0	C27	2	70254	06-08-2020	
19101	10-01-2020	2788	30.0	0.0	C27	2	70254	06-08-2020	
19102	11-01-2020	2788	30.0	0.0	C27	2	70254	06-08-2020	
19103	12-01-2020	2788	30.0	0.0	C27	2	70254	06-08-2020	

19104 rows × 13 columns

In [9]:
```
data.info()
```

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

In [10]:
```
##Converting 'MMM-YY' feature to datetime type
data['MMM-YY'] = pd.to_datetime(data['MMM-YY'])
##Converting 'Dateofjoining' feature to datetime type
data['Dateofjoining'] = pd.to_datetime(data['Dateofjoining'])
##Converting 'LastWorkingDate' feature to datetime type
data['LastWorkingDate'] = pd.to_datetime(data['LastWorkingDate'])
```

```
C:\Users\bikim\AppData\Local\Temp\ipykernel_14296\813856409.py:4: UserWarn
ing: Parsing dates in DD/MM/YYYY format when dayfirst=False (the default)
was specified. This may lead to inconsistently parsed dates! Specify a for
mat to ensure consistent parsing.
  data['Dateofjoining'] = pd.to_datetime(data['Dateofjoining'])
C:\Users\bikim\AppData\Local\Temp\ipykernel_14296\813856409.py:6: UserWarn
ing: Parsing dates in DD/MM/YYYY format when dayfirst=False (the default)
was specified. This may lead to inconsistently parsed dates! Specify a for
mat to ensure consistent parsing.
  data['LastWorkingDate'] = pd.to_datetime(data['LastWorkingDate'])
```

In [11]:
```
data.info()
```

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

Step: Imputation of Missing data

In [12]:
```
data.isnull().sum()/len(data)*100
```

Out[12]:
```
MMM-YY                   0.000000
Driver_ID                0.000000
Age                      0.319305
Gender                   0.272194
City                     0.000000
Education_Level          0.000000
Income                   0.000000
Dateofjoining            0.000000
LastWorkingDate         91.541039
Joining Designation      0.000000
Grade                    0.000000
Total Business Value     0.000000
Quarterly Rating         0.000000
dtype: float64
```

In [13]:
```
data['Gender'].value_counts()
```

Out[13]:
```
0.0    11074
1.0     7978
Name: Gender, dtype: int64
```

In [14]:
```
data['Education_Level'].value_counts()
```

Out[14]:
```
1    6864
2    6327
0    5913
Name: Education_Level, dtype: int64
```

KNN Imputation

In [15]:
```
data_nums=data.select_dtypes(np.number)
#keeping only the numerical columns
```

In [16]:
```
data_nums
```

Out[16]:

	Driver_ID	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Qu
0	1	28.0	0.0	2	57387	1	1	2381060	
1	1	28.0	0.0	2	57387	1	1	-665480	
2	1	28.0	0.0	2	57387	1	1	0	
3	2	31.0	0.0	2	67016	2	2	0	
4	2	31.0	0.0	2	67016	2	2	0	
...	...	...	...	...	...	...	...	...	
19099	2788	30.0	0.0	2	70254	2	2	740280	
19100	2788	30.0	0.0	2	70254	2	2	448370	
19101	2788	30.0	0.0	2	70254	2	2	0	
19102	2788	30.0	0.0	2	70254	2	2	200420	
19103	2788	30.0	0.0	2	70254	2	2	411480	

19104 rows × 9 columns

In [17]:
```
data_nums.isnull().sum()
```

Out[17]:
```
Driver_ID                 0
Age                      61
Gender                   52
Education_Level           0
Income                    0
Joining Designation       0
Grade                     0
Total Business Value      0
Quarterly Rating          0
dtype: int64
```

In [18]:
```
data_nums.drop(columns='Driver_ID',inplace=True)
columns=data_nums.columns
```

In [19]:
```
from sklearn.impute import KNNImputer
imputer  = KNNImputer(n_neighbors = 5, weights = 'uniform', metric = 'nan_e
imputer.fit(data_nums)
# transform the dataset
data_new = imputer.transform(data_nums)
```

In [20]:
```
data_new = pd.DataFrame(data_new)
```

In [21]:
```
data_new
```

Out[21]:

	0	1	2	3	4	5	6	7
0	28.0	0.0	2.0	57387.0	1.0	1.0	2381060.0	2.0
1	28.0	0.0	2.0	57387.0	1.0	1.0	-665480.0	2.0
2	28.0	0.0	2.0	57387.0	1.0	1.0	0.0	2.0
3	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0
4	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0
...	...	...	...	...	...	...	...	...
19099	30.0	0.0	2.0	70254.0	2.0	2.0	740280.0	3.0
19100	30.0	0.0	2.0	70254.0	2.0	2.0	448370.0	3.0
19101	30.0	0.0	2.0	70254.0	2.0	2.0	0.0	2.0
19102	30.0	0.0	2.0	70254.0	2.0	2.0	200420.0	2.0
19103	30.0	0.0	2.0	70254.0	2.0	2.0	411480.0	2.0

19104 rows × 8 columns

In [22]:
```
data_new.columns = columns
```

In [23]:
```
data_new.isnull().sum()
```

Out[23]:
```
Age                      0
Gender                   0
Education_Level          0
Income                   0
Joining Designation      0
Grade                    0
Total Business Value     0
Quarterly Rating         0
dtype: int64
```

Getting the remaining columns back

In [24]:
```
remaining_columns = list(set(data.columns).difference(set(columns)))
```

In [25]:
```
data_ = pd.concat([data_new, data[remaining_columns]], axis = 1)
```

In [26]:
```
data_.head()
```

Out[26]:

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	MMM-YY
0	28.0	0.0	2.0	57387.0	1.0	1.0	2381060.0	2.0	2019-01-01
1	28.0	0.0	2.0	57387.0	1.0	1.0	-665480.0	2.0	2019-02-01
2	28.0	0.0	2.0	57387.0	1.0	1.0	0.0	2.0	2019-03-01
3	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	2020-11-01
4	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	2020-12-01

Checking if the concat is correct or not

In [27]:
```
data_[data_['Driver_ID']==2788]
```

Out[27]:

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	MM
19097	29.0	0.0	2.0	70254.0	2.0	2.0	0.0	1.0	20 06
19098	30.0	0.0	2.0	70254.0	2.0	2.0	497690.0	3.0	20 07
19099	30.0	0.0	2.0	70254.0	2.0	2.0	740280.0	3.0	20 08
19100	30.0	0.0	2.0	70254.0	2.0	2.0	448370.0	3.0	20 09
19101	30.0	0.0	2.0	70254.0	2.0	2.0	0.0	2.0	20 10
19102	30.0	0.0	2.0	70254.0	2.0	2.0	200420.0	2.0	20 11
19103	30.0	0.0	2.0	70254.0	2.0	2.0	411480.0	2.0	20 12

In [28]:
```
data[data['Driver_ID']==2788]
```

Out[28]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWc
19097	2020-06-01	2788	29.0	0.0	C27	2	70254	2020-06-08	
19098	2020-07-01	2788	30.0	0.0	C27	2	70254	2020-06-08	
19099	2020-08-01	2788	30.0	0.0	C27	2	70254	2020-06-08	
19100	2020-09-01	2788	30.0	0.0	C27	2	70254	2020-06-08	
19101	2020-10-01	2788	30.0	0.0	C27	2	70254	2020-06-08	
19102	2020-11-01	2788	30.0	0.0	C27	2	70254	2020-06-08	
19103	2020-12-01	2788	30.0	0.0	C27	2	70254	2020-06-08	

In [29]:
```
function_dict = {'Age':'max', 'Gender':'first','City':'first',
 'Education_Level':'last', 'Income':'last',
 'Joining Designation':'last','Grade':'last',
 'Dateofjoining':'last','LastWorkingDate':'last',
 'Total Business Value':'sum','Quarterly Rating':'last'}
new_train = data_.groupby(['Driver_ID','MMM-YY']).aggregate(function_dict)
```

In [30]:
```
new_train
```

Out[30]:

Driver_ID	MMM-YY	Age	Gender	City	Education_Level	Income	Joining Designation	Grade	Dateofjo
1	2019-01-01	28.0	0.0	C23	2.0	57387.0	1.0	1.0	2018-
	2019-02-01	28.0	0.0	C23	2.0	57387.0	1.0	1.0	2018-
	2019-03-01	28.0	0.0	C23	2.0	57387.0	1.0	1.0	2018-
2	2020-11-01	31.0	0.0	C7	2.0	67016.0	2.0	2.0	2020-
	2020-12-01	31.0	0.0	C7	2.0	67016.0	2.0	2.0	2020-
...	...	...	...	...	...	...	...	...	
2788	2020-08-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-
	2020-09-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-
	2020-10-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-
	2020-11-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-
	2020-12-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-

19104 rows × 11 columns

In [31]:
```
#direct sorting can work but you have to use sort_values
df = new_train.sort_index(ascending=[True,True])
```

In [32]:
```
df.head(10)
```

Out[32]:

Driver_ID	MMM-YY	Age	Gender	City	Education_Level	Income	Joining Designation	Grade	Dateofjo
1	2019-01-01	28.0	0.0	C23	2.0	57387.0	1.0	1.0	2018-
	2019-02-01	28.0	0.0	C23	2.0	57387.0	1.0	1.0	2018-
	2019-03-01	28.0	0.0	C23	2.0	57387.0	1.0	1.0	2018-
2	2020-11-01	31.0	0.0	C7	2.0	67016.0	2.0	2.0	2020-
	2020-12-01	31.0	0.0	C7	2.0	67016.0	2.0	2.0	2020-
4	2019-12-01	43.0	0.0	C13	2.0	65603.0	2.0	2.0	2019-
	2020-01-01	43.0	0.0	C13	2.0	65603.0	2.0	2.0	2019-
	2020-02-01	43.0	0.0	C13	2.0	65603.0	2.0	2.0	2019-
	2020-03-01	43.0	0.0	C13	2.0	65603.0	2.0	2.0	2019-
	2020-04-01	43.0	0.0	C13	2.0	65603.0	2.0	2.0	2019-

In [33]:
```
df1 = pd.DataFrame()
```

In [34]:
```
df1['Driver_ID'] = data_['Driver_ID'].unique()
```

In [35]:
```
del data
```

Aggregation at Driver Level

In [36]:
```
df1['Age'] = list(df.groupby('Driver_ID',axis=0).max('MMM-YY')['Age'])
df1['Gender'] = list(df.groupby('Driver_ID').agg({'Gender':'last'})['Gender
df1['City'] = list(df.groupby('Driver_ID').agg({'City':'last'})['City'])
df1['Education'] = list(df.groupby('Driver_ID').agg({'Education_Level':'las
df1['Income'] = list(df.groupby('Driver_ID').agg({'Income':'last'})['Income
df1['Joining_Designation'] = df.groupby('Driver_ID').agg({'Joining Designat
df1['Grade'] = list(df.groupby('Driver_ID').agg({'Grade':'last'})['Grade'])
df1['Total_Business_Value'] = df.groupby('Driver_ID')['Total Business Value
df1['Last_Quarterly_Rating'] = df.groupby('Driver_ID')['Quarterly Rating'].
```

Creating a column which tells if the quarterly rating has increased for that employee for those whose quarterly rating has increased we assign the value 1

In [37]:
```
#Quarterly rating at the beginning
qrf = df.groupby('Driver_ID').agg({'Quarterly Rating':'first'})

#Quarterly rating at the end
qrl = df.groupby('Driver_ID').agg({'Quarterly Rating':'last'})

qr = (qrl['Quarterly Rating']>qrf['Quarterly Rating']).reset_index()

#the employee ids whose rating has increased
empid = qr[qr['Quarterly Rating']==True]['Driver_ID']

qri = []
for i in df1['Driver_ID']:
    if i in empid.values:
        qri.append(1)
    else:
        qri.append(0)
df1['Quarterly_Rating_Increased'] = qri
```

In [38]:
```
df1
```

Out[38]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_B
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	
1	2	31.0	0.0	C7	2.0	67016.0	2.0	2.0	
2	4	43.0	0.0	C13	2.0	65603.0	2.0	2.0	
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	
...	...	...	...	...	...	...	...	...	
2376	2784	34.0	0.0	C24	0.0	82815.0	2.0	3.0	
2377	2785	34.0	1.0	C9	0.0	12105.0	1.0	1.0	
2378	2786	45.0	0.0	C19	0.0	35370.0	2.0	2.0	
2379	2787	28.0	1.0	C20	2.0	69498.0	1.0	1.0	
2380	2788	30.0	0.0	C27	2.0	70254.0	2.0	2.0	

2381 rows × 11 columns

1. Creating a column called target which tells if the person has left the company
2. Persons who have a last working date will have the value 1
3. The dataset which has the employee ids and specifies if last working date is null and the employee ids who do not have last working date are assigned 0.

In [39]:
```
df.groupby('Driver_ID').agg({'LastWorkingDate':'last'})['LastWorkingDate']
```

Out[39]:
```
Driver_ID
1       2019-03-11
2              NaT
4       2020-04-27
5       2019-03-07
6              NaT
           ...
2784           NaT
2785    2020-10-28
2786    2019-09-22
2787    2019-06-20
2788           NaT
Name: LastWorkingDate, Length: 2381, dtype: datetime64[ns]
```

In [40]:
```
#df1[['target']] = np.where(pd.notnull(df[['LastWorkingDate']]), 1, 0)
#df1['target'] = np.where(pd.notnull(df['LastWorkingDate'].iloc[:len(df1)])
```

In [41]:
```
lwr = (df.groupby('Driver_ID').agg({'LastWorkingDate':'last'})['LastWorking
#The employee ids who do not have last working date
empid = lwr[lwr['LastWorkingDate']==True]['Driver_ID']
target = []
for i in df1['Driver_ID']:
    if i in empid.values:
        target.append(0)
    elif i not in empid.values:
        target.append(1)

df1['target'] = target
```

In [42]:
```
df1
```

Out[42]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_B
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	
1	2	31.0	0.0	C7	2.0	67016.0	2.0	2.0	
2	4	43.0	0.0	C13	2.0	65603.0	2.0	2.0	
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	
...	...	...	...	...	...	...	...	...	
2376	2784	34.0	0.0	C24	0.0	82815.0	2.0	3.0	
2377	2785	34.0	1.0	C9	0.0	12105.0	1.0	1.0	
2378	2786	45.0	0.0	C19	0.0	35370.0	2.0	2.0	
2379	2787	28.0	1.0	C20	2.0	69498.0	1.0	1.0	
2380	2788	30.0	0.0	C27	2.0	70254.0	2.0	2.0	

2381 rows × 12 columns

Creating a column which tells if the monthly income has increased for that employee for those whose monthly income has increased we assign the value 1

In [43]:
```
#Quarterly rating at the beginning
sf = df.groupby('Driver_ID').agg({'Income':'first'})

#Quarterly rating at the end
sl = df.groupby('Driver_ID').agg({'Income':'last'})

s = (sl['Income']>sf['Income']).reset_index()

#the employee ids whose monthly income has increased
empid = s[s['Income']==True]['Driver_ID']

si = []
for i in df1['Driver_ID']:
    if i in empid.values:
        si.append(1)
    else:
        si.append(0)
df1['Income_Increased'] = si
```

In [44]:
```
df1['Income_Increased'].value_counts()
```

Out[44]:
```
0    2338
1      43
Name: Income_Increased, dtype: int64
```

In [45]:
```
df1.head()
```

Out[45]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_Busi
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	
1	2	31.0	0.0	C7	2.0	67016.0	2.0	2.0	
2	4	43.0	0.0	C13	2.0	65603.0	2.0	2.0	
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	

Statistical Summary

In [46]:
```
df1.describe().T
```

Out[46]:

	count	mean	std	min	25%	50
Driver_ID	2381.0	1.397559e+03	8.061616e+02	1.0	695.0	140(
Age	2381.0	3.377018e+01	5.933265e+00	21.0	30.0	33
Gender	2381.0	4.105838e-01	4.914963e-01	0.0	0.0	(
Education	2381.0	1.007560e+00	8.162900e-01	0.0	0.0	1
Income	2381.0	5.933416e+04	2.838367e+04	10747.0	39104.0	5531€
Joining_Designation	2381.0	1.820244e+00	8.414334e-01	1.0	1.0	2
Grade	2381.0	2.096598e+00	9.415218e-01	1.0	1.0	2
Total_Business_Value	2381.0	1.084087e+07	1.462092e+07	-439300.0	750000.0	410172(
Last_Quarterly_Rating	2381.0	1.427971e+00	8.098389e-01	1.0	1.0	1
Quarterly_Rating_Increased	2381.0	1.503570e-01	3.574961e-01	0.0	0.0	(
target	2381.0	6.787064e-01	4.670713e-01	0.0	0.0	1
Income_Increased	2381.0	1.805964e-02	1.331951e-01	0.0	0.0	(

There are 2381 employees in the dataset. The minimum age of the employee in the data is 21 years and the maximum age is 58 years. 75% of the employees have their monthly income less than or equal to 75,986 units. 50% of the mployees have acquired 8,17,680 as the their total business value

In [47]:
```
df1.describe(include=['O'])
```

Out[47]:

	City
count	2381
unique	29
top	C20
freq	152

Most of the drivers in the dataset were male, lived in C20 city and have completed their graduation in education

In [48]:
```
df1['target'].value_counts()
```

Out[48]:
```
1    1616
0     765
Name: target, dtype: int64
```

Out of 2381 drivers, 2164 drivers have left the organization.

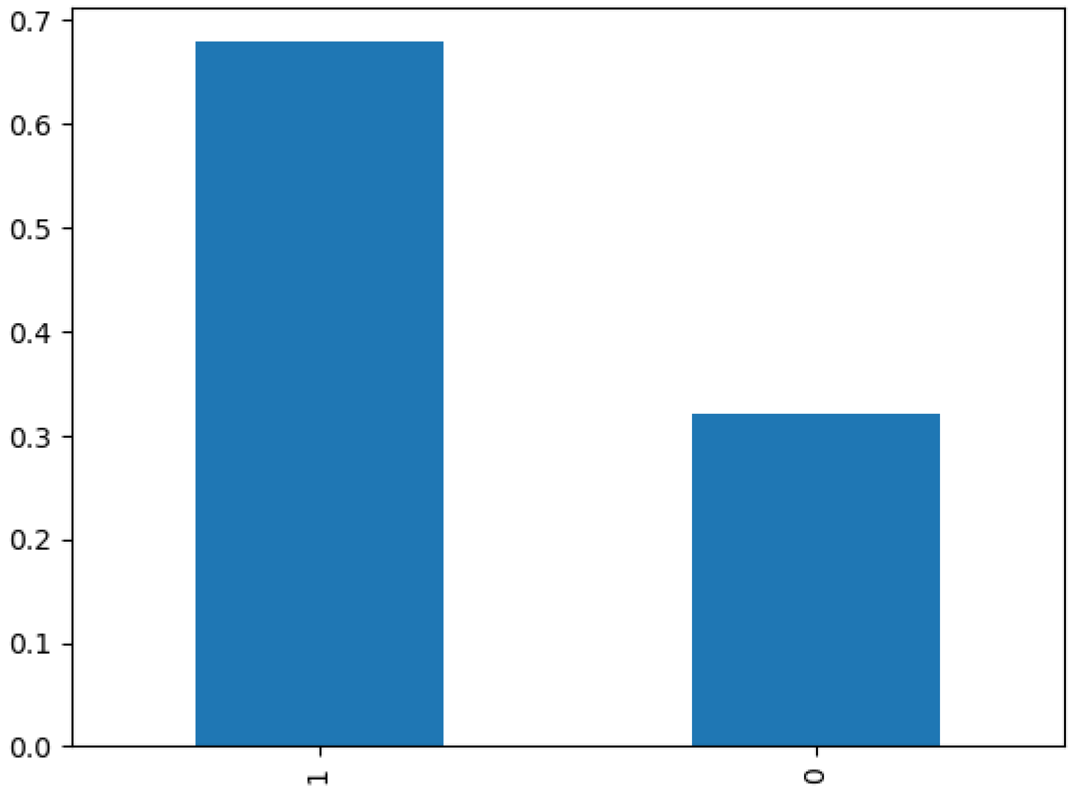
In [49]:
```
df1['target'].value_counts(normalize=True)*100
```

Out[49]:
```
1    67.870643
0    32.129357
Name: target, dtype: float64
```

Around 68% driver have left the organization.

In [50]:
```
df1['target'].value_counts(normalize=True).plot(kind='bar')
```

Out[50]:
```
<Axes: >
```



Categorical Features: Gender, City, Education, Joining_Designation, Designation,Last_Quarterly_Rating, Quarterly_Rating_Increased

In [51]:
```
#Count of observations in each category
n = ['Gender','City','Education','Joining_Designation','Grade','Last_Quarte
for i in n:
    print(df1[i].value_counts())
    print("------------------------------------------------------------")
```

```
0.0    1400
1.0     975
0.6       3
0.2       2
0.4       1
Name: Gender, dtype: int64
------------------------------------------------------------
C20    152
C15    101
C29     96
C26     93
C8      89
C27     89
C10     86
C16     84
C22     82
C3      82
C28     82
C12     81
C5      80
C1      80
C21     79
C14     79
C6      78
C4      77
C7      76
C9      75
C25     74
C23     74
C24     73
C19     72
C2      72
C17     71
C13     71
C18     69
C11     64
Name: City, dtype: int64
------------------------------------------------------------
2.0    802
1.0    795
0.0    784
Name: Education, dtype: int64
------------------------------------------------------------
1.0    1026
2.0     815
3.0     493
4.0      36
5.0      11
Name: Joining_Designation, dtype: int64
------------------------------------------------------------
2.0    855
1.0    741
3.0    623
4.0    138
5.0     24
Name: Grade, dtype: int64
------------------------------------------------------------
1.0    1744
2.0     362
3.0     168
4.0     107
```

```
Name: Last_Quarterly_Rating, dtype: int64
------------------------------------------------------------
0    2023
1     358
Name: Quarterly_Rating_Increased, dtype: int64
------------------------------------------------------------
```

Out of 2381 employees, 1404 employees are of the Male gender and 977 are females.

Out of 2381 employees, 152 employees are from city C20 and 101 from city C15.

Out of 2381 employees, 802 employees have their education as Graduate and 795 have completed their 12.

Out of 2381 employees, 1026 joined with the grade as 1, 815 employees joined with the grade 2.

Out of 2381 employees, 855 employees had their designation as 2 at the time of reporting.

Out of 2381 employees, 1744 employees had their last quarterly rating as 1.

Out of 2381 employees, the quarterly rating has not increased for 2076 employees.

In [52]:

```
#Proportion of observations in each category
n = ['Gender','City','Education','Joining_Designation','Grade','Last_Quarte
for i in n:
    print(df1[i].value_counts(normalize=True))
    print("------------------------------------------------------------")
```

```
0.0    0.587988
1.0    0.409492
0.6    0.001260
0.2    0.000840
0.4    0.000420
Name: Gender, dtype: float64
------------------------------------------------------------
C20    0.063839
C15    0.042419
C29    0.040319
C26    0.039059
C8     0.037379
C27    0.037379
C10    0.036119
C16    0.035279
C22    0.034439
C3     0.034439
C28    0.034439
C12    0.034019
```

Around 59% employees are of the Male gender.

Around 6.4% employees are from city C20 and 4.2% from city C15.

The proportion of the employees who have completed their Graduate and 12th is approximately same.

Around 43% of the employees joined with the grade 1.

At the time of reporting, 34% of the employees had their grade as 2.

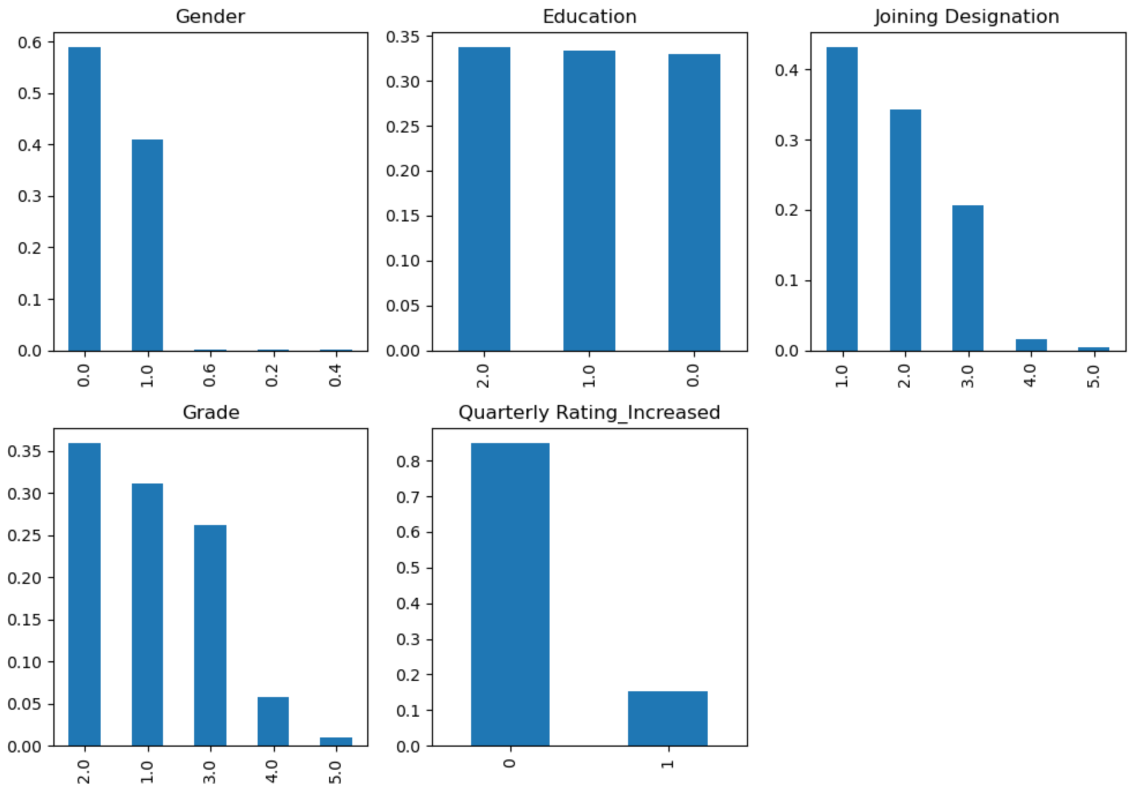
Around 73% of the employees had their last quarterly rating as 1.

The quarterly rating has not increased for around 87% employees.

In [53]:

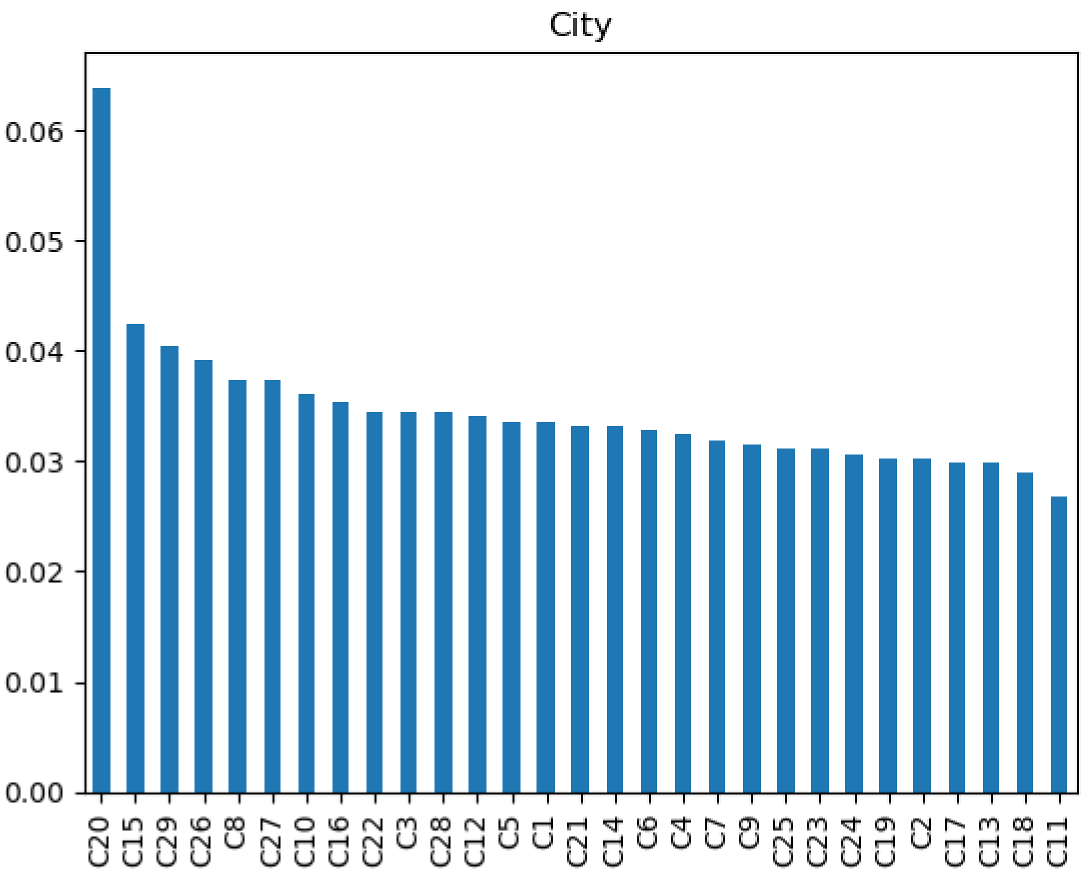
```
n = ['Gender','City','Joining_Designation','Grade','Last_Quarterly_Rating',
plt.subplots(figsize=(10,7))
plt.subplot(231)
df1['Gender'].value_counts(normalize=True).plot.bar(title='Gender')
plt.subplot(232)
df1['Education'].value_counts(normalize=True).plot.bar(title='Education')
plt.subplot(233)
df1['Joining_Designation'].value_counts(normalize=True).plot.bar(title='Joi
plt.subplot(234)
df1['Grade'].value_counts(normalize=True).plot.bar(title='Grade')
plt.subplot(235)
df1['Last_Quarterly_Rating'].value_counts(normalize=True).plot.bar(title='L
plt.subplot(235)
df1['Quarterly_Rating_Increased'].value_counts(normalize=True).plot.bar(tit
plt.tight_layout()
```

```
C:\Users\bikim\AppData\Local\Temp\ipykernel_14296\1433226780.py:3: Matplot
libDeprecationWarning: Auto-removal of overlapping axes is deprecated sinc
e 3.6 and will be removed two minor releases later; explicitly call ax.rem
ove() as needed.
  plt.subplot(231)
```



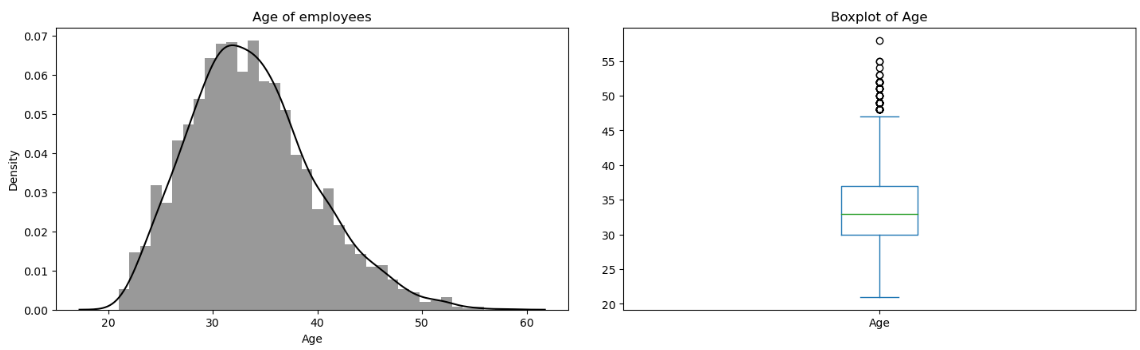
In [54]:
```
df1['City'].value_counts(normalize=True).plot.bar(title='City')
```

Out[54]:
```
<Axes: title={'center': 'City'}>
```



In [55]:
```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.distplot(df1['Age'],color='black')
plt.title("Age of employees")
plt.subplot(122)
df1['Age'].plot.box(title='Boxplot of Age')
plt.tight_layout(pad=3)
```

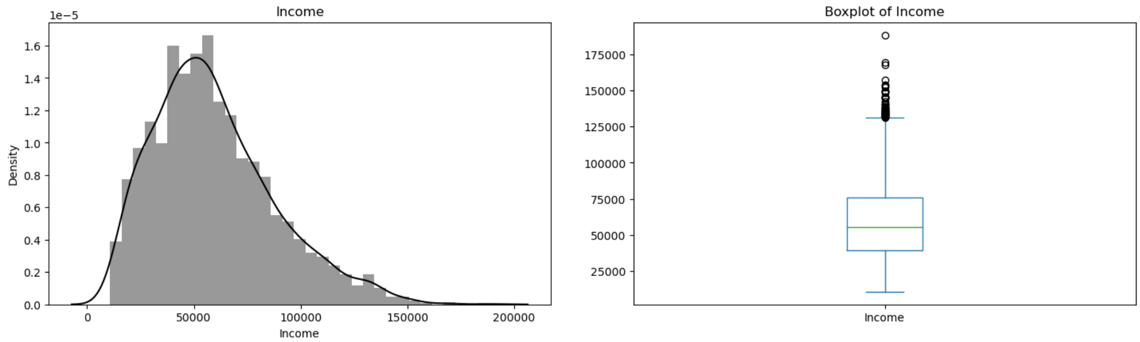
```
C:\Users\bikim\AppData\Local\Temp\ipykernel_14296\2591424612.py:2: Matplot
libDeprecationWarning: Auto-removal of overlapping axes is deprecated sinc
e 3.6 and will be removed two minor releases later; explicitly call ax.rem
ove() as needed.
  plt.subplot(121)
C:\Users\bikim\AppData\Local\Temp\ipykernel_14296\2591424612.py:3: UserWar
ning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://
gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df1['Age'],color='black')
```



There are few outliers in the Age. The distribution is towards the right.

In [56]:
```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.distplot(df1['Income'],color='black')
plt.title("Income")
plt.subplot(122)
df1['Income'].plot.box(title='Boxplot of Income')
plt.tight_layout(pad=3)
```

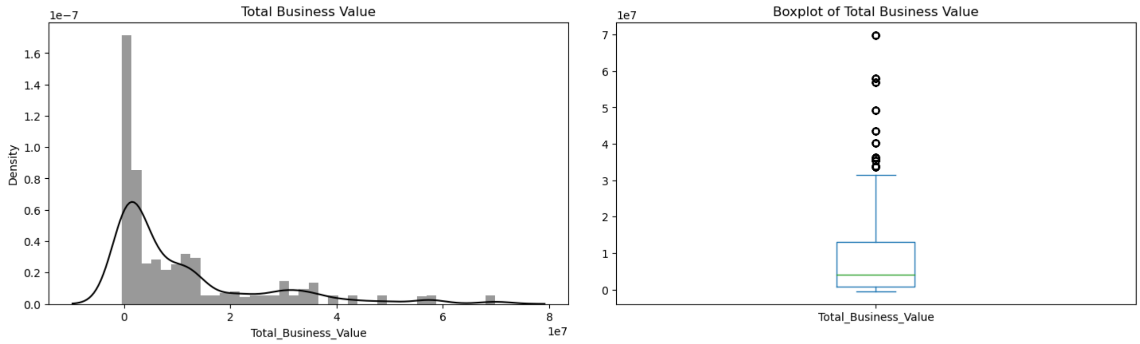
```
C:\Users\bikim\AppData\Local\Temp\ipykernel_14296\736098657.py:2: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.
  plt.subplot(121)
C:\Users\bikim\AppData\Local\Temp\ipykernel_14296\736098657.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df1['Income'],color='black')
```



The distribution of Salary is towards the right and there are outliers for this feature as well.

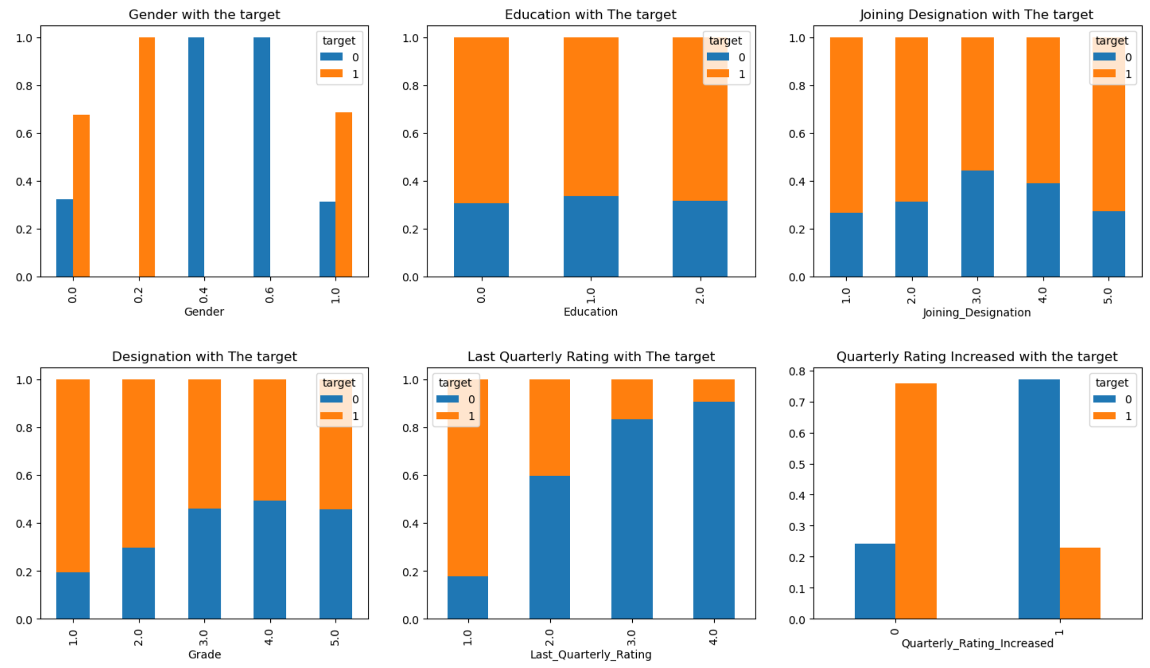
In [57]:
```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.distplot(df1['Total_Business_Value'],color='black')
plt.title("Total Business Value")
plt.subplot(122)
df1['Total_Business_Value'].plot.box(title='Boxplot of Total Business Value
plt.tight_layout(pad=3)
```

```
C:\Users\bikim\AppData\Local\Temp\ipykernel_14296\4200608847.py:2: Matplot
libDeprecationWarning: Auto-removal of overlapping axes is deprecated sinc
e 3.6 and will be removed two minor releases later; explicitly call ax.rem
ove() as needed.
  plt.subplot(121)
C:\Users\bikim\AppData\Local\Temp\ipykernel_14296\4200608847.py:3: UserWar
ning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://
gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df1['Total_Business_Value'],color='black')
```



The distribution of total business value is towards the right. There are a lot of outliers for the feature Total Business Value.

In [58]:
```
figure,axes=plt.subplots(2,3,figsize=(15,9))

#Gender feature with Target
gender = pd.crosstab(df1['Gender'],df1['target'])
gender.div(gender.sum(1).astype(float),axis=0).plot(kind='bar',stacked=Fals

#Education feature with Target
education = pd.crosstab(df1['Education'],df1['target'])
education.div(education.sum(1).astype(float),axis=0).plot(kind='bar',stacke
 title="Education with The target")

#Joining Designation feature with Target
jde = pd.crosstab(df1['Joining_Designation'],df1['target'])
jde.div(jde.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,ax=ax
 title="Joining Designation with The target")

#Designation feature with Target
desig = pd.crosstab(df1['Grade'],df1['target'])
desig.div(desig.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,a
 title="Designation with The target")

#Last Quarterly Rating feature with Target
lqrate = pd.crosstab(df1['Last_Quarterly_Rating'],df1['target'])
lqrate.div(lqrate.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True
 title="Last Quarterly Rating with The target")

#Quarterly Rating Increased feature with Target
qratei = pd.crosstab(df1['Quarterly_Rating_Increased'],df1['target'])
qratei.div(qratei.sum(1).astype(float),axis=0).plot(kind='bar',stacked=Fals
 title="Quarterly Rating Increased with the target")
plt.tight_layout(pad=3)
```



The proportion of gender and education is more or less the same for both the employees who left the organization and those who did not leave.

The employees who have their grade as 3 or 4 at the time of joining are less likely to leave the organization.

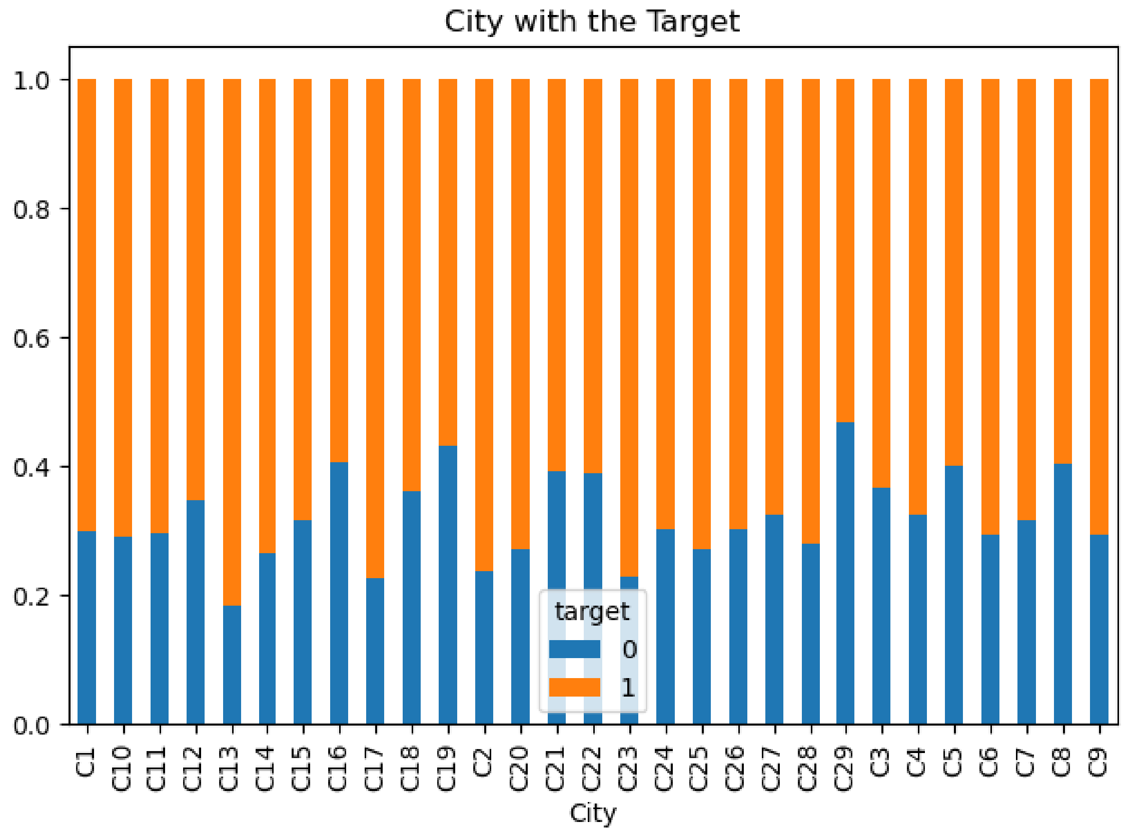
The employees who have their grade as 3 or 4 at the time of reporting are less likely to leave the organization.

The employees who have their last quarterly rating as 3 or 4 at the time of reporting are less likely to leave the organization.

The employees whose quarterly rating has increased are less likely to leave the

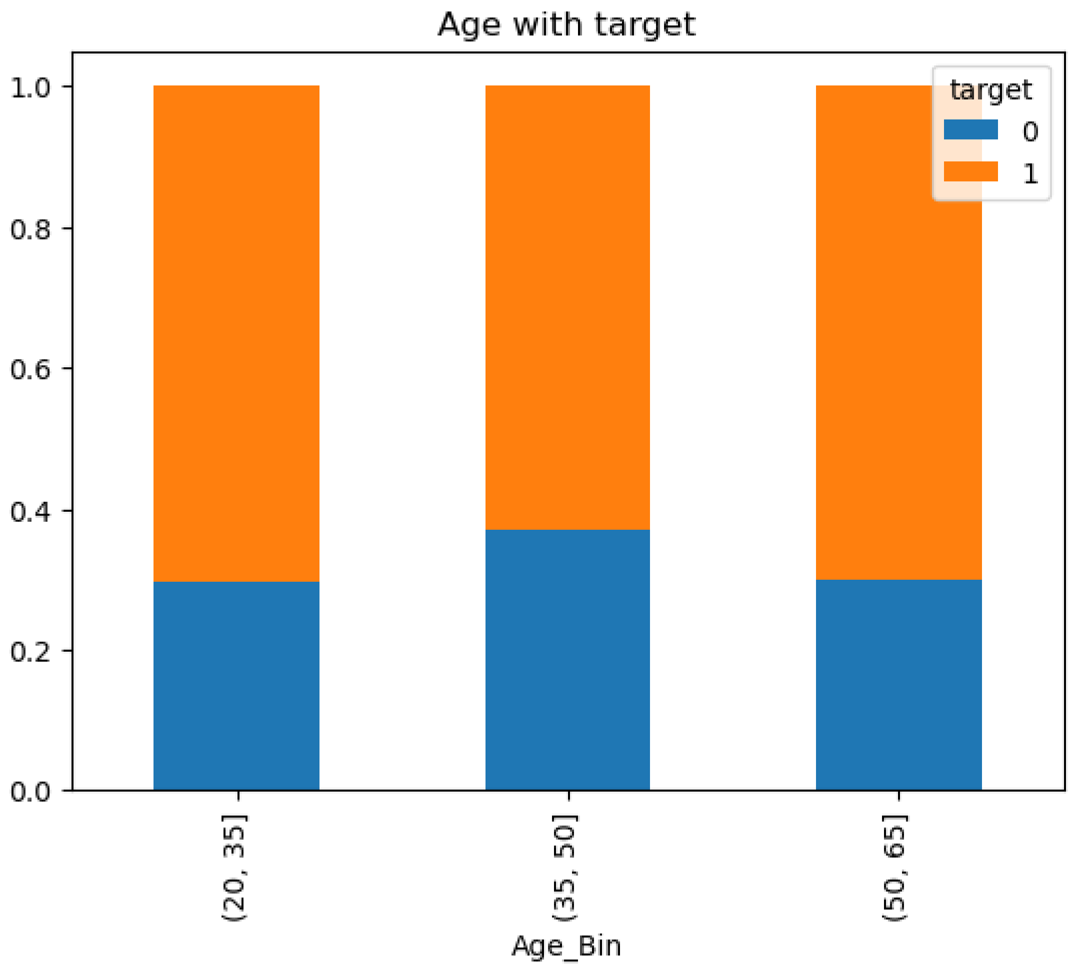
In [59]:
```
#City feature with the target
plt.figure(figsize=(30,7))
city = pd.crosstab(df1['City'],df1['target'])
city.div(city.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,tit
plt.tight_layout()
```

```
<Figure size 3000x700 with 0 Axes>
```



In [60]:
```
#Binning the Age into categories
df1['Age_Bin'] = pd.cut(df1['Age'],bins=[20,35,50,65])

#Age feature with Target
agebin = pd.crosstab(df1['Age_Bin'],df1['target'])
agebin.div(agebin.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True
```

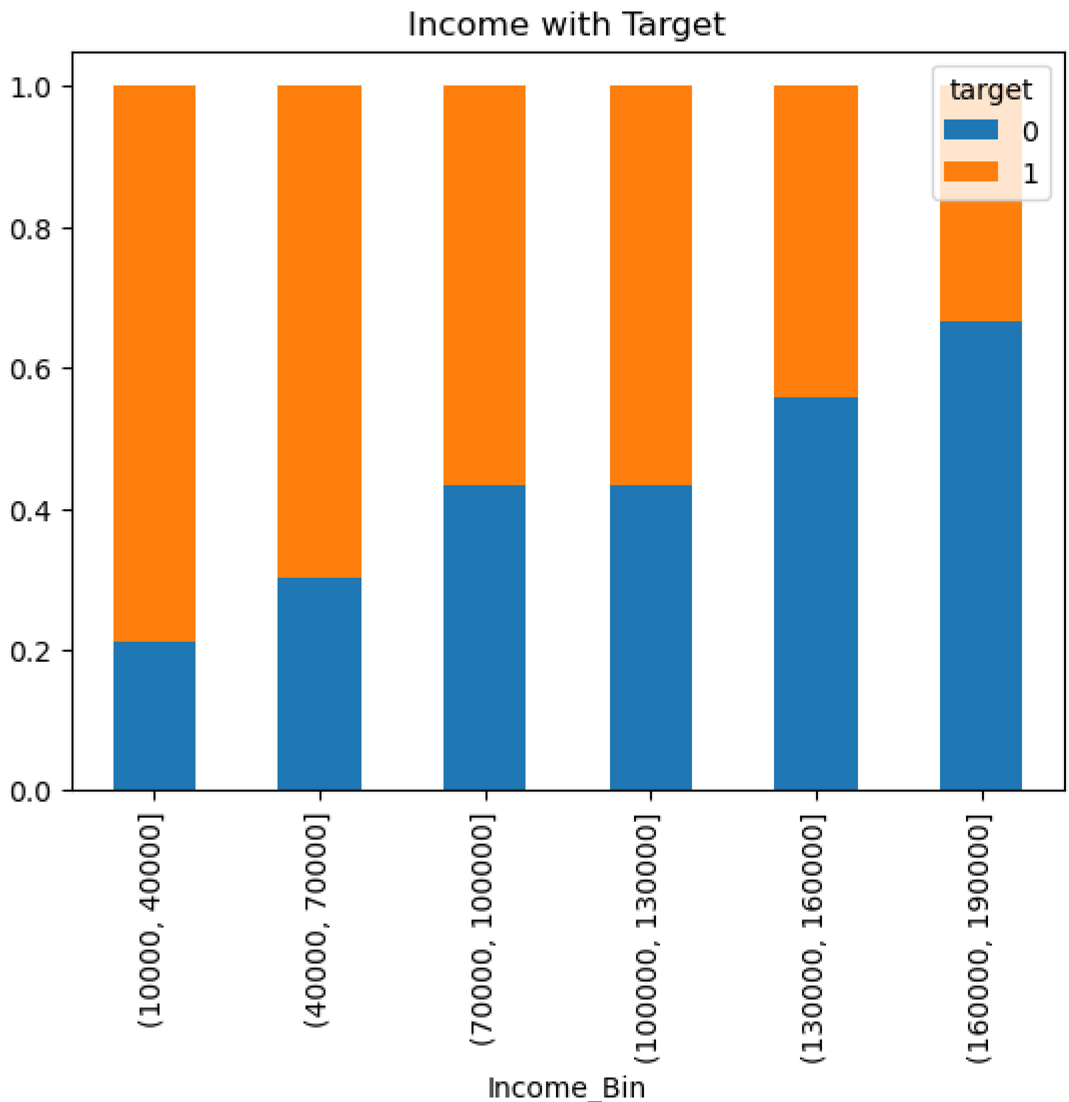
Out[60]:
```
<Axes: title={'center': 'Age with target'}, xlabel='Age_Bin'>
```



The employees whose age is in the 20-35 or 50-65 groups are more likely to leave the organization.

In [61]:
```
# Binninhg the Income into categories
df1['Income_Bin'] = pd.cut(df1['Income'],bins=[10000, 40000, 70000, 100000,

# Salary feature with Target
Salarybin = pd.crosstab(df1['Income_Bin'],df1['target'])
Salarybin.div(Salarybin.sum(1).astype(float),axis=0).plot(kind='bar',stacke
```

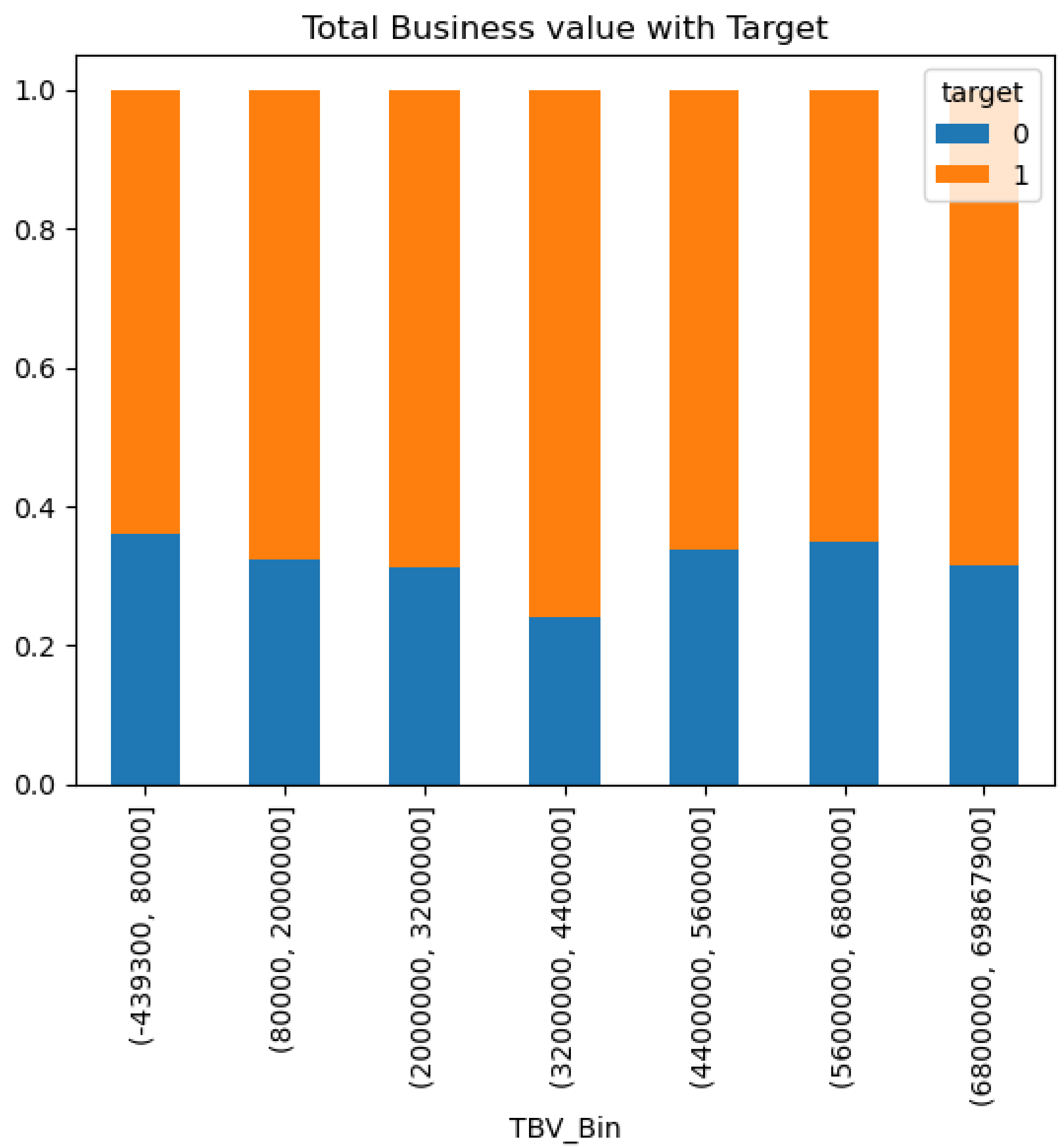
Out[61]:
```
<Axes: title={'center': 'Income with Target'}, xlabel='Income_Bin'>
```



Drivers whose monthly income 160000 - 190000 are less likely to leave the organization

In [62]:
```
#Defining the bins and groups
m1 = round(df1['Total_Business_Value'].min())
m2 = round(df1['Total_Business_Value'].max())
bins = [m1, 80000 , 2000000 , 3200000, 4400000, 5600000, 6800000, m2]

#Binning the Total Business Value into categories
df1['TBV_Bin'] = pd.cut(df1['Total_Business_Value'],bins)

#Total Business Value feature with Target
tbvbin = pd.crosstab(df1['TBV_Bin'],df1['target'])
tbvbin.div(tbvbin.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True
```

Out[62]:
```
<Axes: title={'center': 'Total Business value with Target'}, xlabel='TBV_B
in'>
```



The employees who have acquired total business value greater than 68,00,000 are less likely to leave the organiztion.

In [63]:
```
#Dropping the bins columns
df1.drop(['Age_Bin','Income_Bin','TBV_Bin'],axis=1,inplace=True)
```

In [64]:
```
df1.head()
```

Out[64]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_Busi
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	
1	2	31.0	0.0	C7	2.0	67016.0	2.0	2.0	
2	4	43.0	0.0	C13	2.0	65603.0	2.0	2.0	
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	

Step:One Hot Encoding

Alternatively, we can do "Target" Imputation

In [65]:
```
df1 = pd.concat([df1,pd.get_dummies(df1['City'],prefix='City')],axis=1)
```

Step-5:Scaling the data (Only done on training set)

Normalising the Dataset. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.

Dropping the encoded and scaled columns

In [66]:
```
df1
```

Out[66]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_B
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	
1	2	31.0	0.0	C7	2.0	67016.0	2.0	2.0	
2	4	43.0	0.0	C13	2.0	65603.0	2.0	2.0	
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	
...	...	...	...	...	...	...	...	...	
2376	2784	34.0	0.0	C24	0.0	82815.0	2.0	3.0	
2377	2785	34.0	1.0	C9	0.0	12105.0	1.0	1.0	
2378	2786	45.0	0.0	C19	0.0	35370.0	2.0	2.0	
2379	2787	28.0	1.0	C20	2.0	69498.0	1.0	1.0	
2380	2788	30.0	0.0	C27	2.0	70254.0	2.0	2.0	

2381 rows × 42 columns

In [67]:
```
#Feature Variables
X = df1.drop(['Driver_ID','target','City'],axis=1)
X_cols=X.columns
# MinMaxScaler
scaler = MinMaxScaler()
#Mathematically learning the distribution
X=scaler.fit_transform(X)
```

In [68]:
```
X=pd.DataFrame(X)
X
```

Out[68]:

	0	1	2	3	4	5	6	7	8	9	...	29	30	31
0	0.189189	0.0	1.0	0.262508	0.00	0.00	0.030649	0.333333	0.0	0.0	...	0.0	0.0	0.0
1	0.270270	0.0	1.0	0.316703	0.25	0.25	0.030649	0.000000	0.0	0.0	...	0.0	0.0	0.0
2	0.594595	0.0	1.0	0.308750	0.25	0.25	0.030649	0.000000	0.0	0.0	...	0.0	0.0	0.0
3	0.216216	0.0	0.0	0.200489	0.00	0.00	0.006248	0.000000	0.0	0.0	...	0.0	0.0	0.0
4	0.270270	1.0	0.5	0.382623	0.50	0.50	0.006248	0.333333	1.0	0.0	...	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2376	0.351351	0.0	0.0	0.405626	0.25	0.50	0.445311	1.000000	1.0	0.0	...	0.0	0.0	0.0
2377	0.351351	1.0	0.0	0.007643	0.00	0.00	0.445311	0.000000	0.0	0.0	...	0.0	0.0	0.0
2378	0.648649	0.0	0.0	0.138588	0.25	0.25	0.445311	0.000000	0.0	0.0	...	0.0	0.0	0.0
2379	0.189189	1.0	1.0	0.330673	0.00	0.00	0.445311	0.000000	0.0	0.0	...	0.0	0.0	0.0
2380	0.243243	0.0	1.0	0.334928	0.25	0.25	0.445311	0.333333	1.0	0.0	...	1.0	0.0	0.0

2381 rows × 39 columns

In [69]:
```
X.columns=X_cols
X
```

Out[69]:

	Age	Gender	Education	Income	Joining_Designation	Grade	Total_Business_Va
0	0.189189	0.0	1.0	0.262508	0.00	0.00	0.030(
1	0.270270	0.0	1.0	0.316703	0.25	0.25	0.030(
2	0.594595	0.0	1.0	0.308750	0.25	0.25	0.030(
3	0.216216	0.0	0.0	0.200489	0.00	0.00	0.006:
4	0.270270	1.0	0.5	0.382623	0.50	0.50	0.006:
...	...	...	...	...	...	...	
2376	0.351351	0.0	0.0	0.405626	0.25	0.50	0.445:
2377	0.351351	1.0	0.0	0.007643	0.00	0.00	0.445:
2378	0.648649	0.0	0.0	0.138588	0.25	0.25	0.445:
2379	0.189189	1.0	1.0	0.330673	0.00	0.00	0.445:
2380	0.243243	0.0	1.0	0.334928	0.25	0.25	0.445:

2381 rows × 39 columns

In [70]:
```
#Target Variable
y = df1['target']
# split into 80:20 ration
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,
```

In [71]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[71]:
```
((1904, 39), (477, 39), (1904,), (477,))
```

Random Forest with class weights

In [72]:
```
from sklearn.utils import class_weight
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

param = {'max_depth':[2,3,4], 'n_estimators':[50,100,150,200]}
random_forest = RandomForestClassifier(class_weight ='balanced')

c = GridSearchCV(random_forest,param,cv=3,scoring='f1')
c.fit(X_train,y_train)

def display(results):
    print(f'Best parameters are : {results.best_params_}')
    print(f'The score is : {results.best_score_}')

display(c)
y_pred = c.predict(X_test)

print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Best parameters are : {'max_depth': 2, 'n_estimators': 100}
The score is : 0.8489699955535354
              precision    recall  f1-score   support

           0       0.69      0.56      0.62       148
           1       0.82      0.89      0.85       329

    accuracy                           0.79       477
   macro avg       0.75      0.72      0.74       477
weighted avg       0.78      0.79      0.78       477

[[ 83  65]
 [ 37 292]]
```

The Random Forest With Class Weighting method out of all predicted 0 the measure of correctly predicted is 70%, and for 1 it is 82%(Precision).

The Random Forest With Class Weighting method out of all actual 0 the measure of correctly predicted is 58%, and for 1 it is 89%(Recall).

In [73]:
```
param = {'max_depth':[2,3,4], 'n_estimators':[50,100,150,200]}

random_forest = RandomForestClassifier(class_weight ='balanced_subsample')

c = GridSearchCV(random_forest,param,cv=3,scoring='f1')
c.fit(X_train,y_train)

def display(results):
    print(f'Best parameters are : {results.best_params_}')
    print(f'The score is : {results.best_score_}')
display(c)
y_pred = c.predict(X_test)

print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Best parameters are : {'max_depth': 2, 'n_estimators': 50}
The score is : 0.8494942984897184
              precision    recall  f1-score   support

           0       0.70      0.55      0.62       148
           1       0.81      0.90      0.85       329

    accuracy                           0.79       477
   macro avg       0.76      0.72      0.73       477
weighted avg       0.78      0.79      0.78       477

[[ 81  67]
 [ 34 295]]
```

The Random Forest With Bootstrap Class Weighting method out of all predicted 0 the measure of correctly predicted is 69%, and for 1 it is 82%(Precision).

The Random Forest With Bootstrap Class Weighting method out of all actual 0 the measure of correctly predicted is 58%, and for 1 it is 88%(Recall).

XGBoost Classifier

In [76]:
```
!pip install xgboost
import xgboost as xgb
my_model = xgb.XGBClassifier(class_weight ='balanced')
#
my_model.fit(X_train, y_train)

# Predicting the Test set results
y_pred = my_model.predict(X_test)

print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Collecting xgboost
  Downloading xgboost-2.0.3-py3-none-win_amd64.whl (99.8 MB)
     ---------------------------------------- 99.8/99.8 MB 4.1 MB/s eta 0:
00:00
Requirement already satisfied: numpy in d:\anaconda\lib\site-packages (fro
m xgboost) (1.23.5)
Requirement already satisfied: scipy in d:\anaconda\lib\site-packages (fro
m xgboost) (1.10.0)
Installing collected packages: xgboost
Successfully installed xgboost-2.0.3

D:\Anaconda\lib\site-packages\xgboost\core.py:160: UserWarning: [14:18:16]
WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group
-i-0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\learner.cc:742:
Parameters: { "class_weight" } are not used.

  warnings.warn(smsg, UserWarning)
              precision    recall  f1-score   support

           0       0.63      0.55      0.59       148
           1       0.81      0.86      0.83       329

    accuracy                           0.76       477
   macro avg       0.72      0.70      0.71       477
weighted avg       0.75      0.76      0.76       477

[[ 81  67]
 [ 47 282]]
```

The XGBoost method out of all predicted 0 the measure of correctly predicted is 63%, and for 1 it is 82%(Precision).

The XGBoost method out of all actual 0 the measure of correctly predicted is 55%, and for 1 it is 86%(Recall)

Decision Tree Classifier

In [78]:
```
from sklearn.tree import DecisionTreeClassifier

# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
              precision    recall  f1-score   support

           0       0.51      0.53      0.52       148
           1       0.78      0.78      0.78       329

    accuracy                           0.70       477
   macro avg       0.65      0.65      0.65       477
weighted avg       0.70      0.70      0.70       477

[[ 78  70]
 [ 74 255]]
```

The Decision Tree method out of all predicted 0 the measure of correctly predicted is 51%, and for 1 it is 79% (Precision).

The Decision Tree method out of all actual 0 the measure of correctly predicted is 53%, and for 1 it is 78%(Recall)

Result Analysis

We observe that we are not getting very high recall on target 0 which may be due to small unbalanced dataset.

Higher precision means that an algorithm returns more relevant results than irrelevant ones, and high recall means that an algorithm returns most of the relevant results (whether or not irrelevant ones are also returned).

Feature Importance for the best model so far in Random Forest Model

In [79]:
```
param = {'max_depth':[2,3,4], 'n_estimators':[50,100,150,200]}

random_forest = RandomForestClassifier(class_weight ='balanced')

random_forest.fit(X_train,y_train)

def display(results):
 print(f'Best parameters are : {results.best_params_}')
 print(f'The score is : {results.best_score_}')
display(c)
```

```
Best parameters are : {'max_depth': 2, 'n_estimators': 50}
The score is : 0.8494942984897184
```

In [80]:
```
import time
import numpy as np

start_time = time.time()
importances = random_forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in random_forest.estimator
elapsed_time = time.time() - start_time

print(f"Elapsed time to compute the importances: {elapsed_time:.3f} seconds
```

```
Elapsed time to compute the importances: 0.032 seconds
```

In [81]:
```
pd.DataFrame(zip(X_train.columns,std)).sort_values(by=[1], ascending=False)
```

Out[81]:

	0	1
7	Last_Quarterly_Rating	0.067320
8	Quarterly_Rating_Increased	0.051143
3	Income	0.026069
5	Grade	0.018832
0	Age	0.017993
6	Total_Business_Value	0.016880
4	Joining_Designation	0.011673
2	Education	0.008932
9	Income_Increased	0.007719
1	Gender	0.007596
17	City_C16	0.005026
13	City_C12	0.004372
31	City_C29	0.004231
24	City_C22	0.004193
23	City_C21	0.004181
27	City_C25	0.004092
36	City_C7	0.003974
37	City_C8	0.003908
34	City_C5	0.003888
35	City_C6	0.003861
32	City_C3	0.003856
20	City_C19	0.003809
22	City_C20	0.003760
28	City_C26	0.003614
21	City_C2	0.003507
25	City_C23	0.003503
33	City_C4	0.003449
29	City_C27	0.003407
26	City_C24	0.003389
16	City_C15	0.003357
18	City_C17	0.003356
10	City_C1	0.003322
38	City_C9	0.003306
11	City_C10	0.003190
14	City_C13	0.003146
15	City_C14	0.003034
12	City_C11	0.002968
19	City_C18	0.002919

	0	1
30	City_C28	0.002632

In [82]:
```
pd.DataFrame(zip(X_train.columns,importances)).sort_values(by=[1], ascendin
```

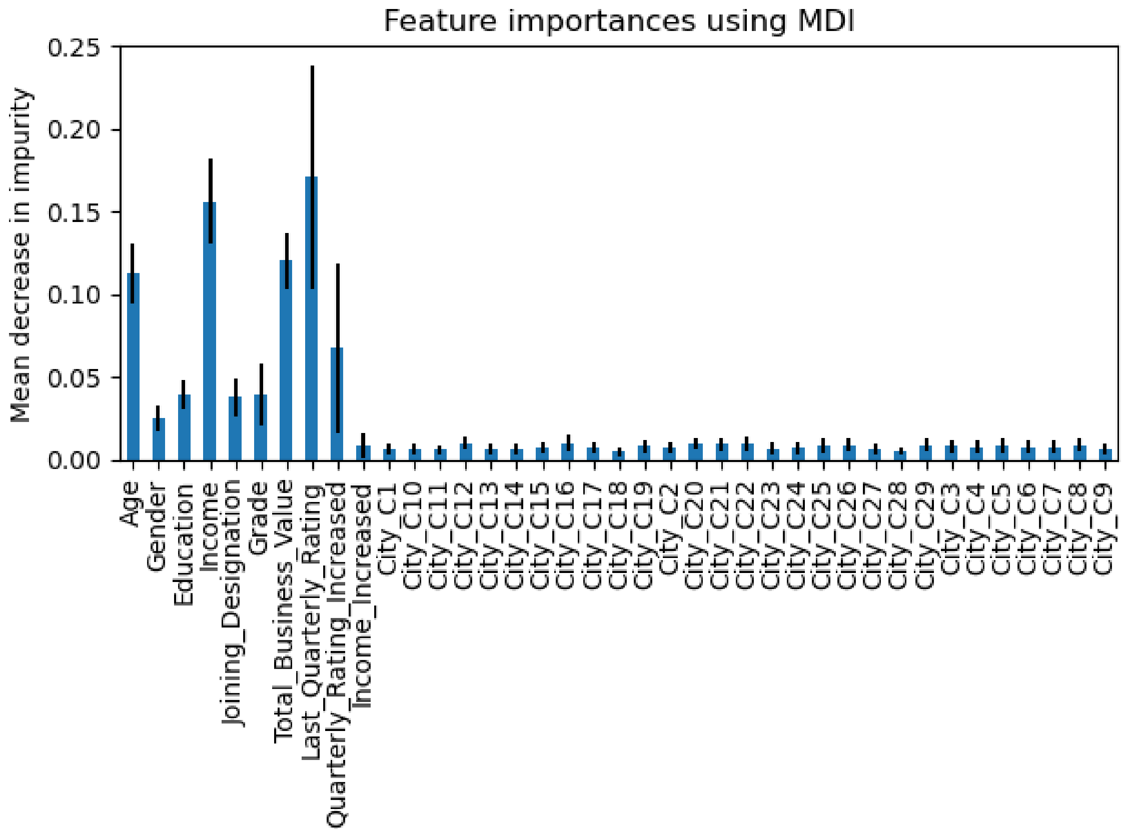
Out[82]:

	0	1
7	Last_Quarterly_Rating	0.170938
3	Income	0.156337
6	Total_Business_Value	0.120360
0	Age	0.112565
8	Quarterly_Rating_Increased	0.067817
5	Grade	0.039594
2	Education	0.039381
4	Joining_Designation	0.038014
1	Gender	0.024681
13	City_C12	0.010185
17	City_C16	0.010145
22	City_C20	0.009680
24	City_C22	0.009419
23	City_C21	0.009243
31	City_C29	0.009052
28	City_C26	0.008956
37	City_C8	0.008762
27	City_C25	0.008583
34	City_C5	0.008521
9	Income_Increased	0.008454
20	City_C19	0.008039
32	City_C3	0.008011
33	City_C4	0.007946
36	City_C7	0.007811
21	City_C2	0.007724
35	City_C6	0.007596
18	City_C17	0.007030
26	City_C24	0.006993
16	City_C15	0.006967
25	City_C23	0.006880
38	City_C9	0.006791
29	City_C27	0.006718
14	City_C13	0.006457
15	City_C14	0.006136
10	City_C1	0.006127
11	City_C10	0.006040
12	City_C11	0.005811
30	City_C28	0.005276

	0	1
19	City_C18	0.004961

In [83]:

```
import pandas as pd
forest_importances = pd.Series(importances, index=X_train.columns)
fig, ax = plt.subplots()
forest_importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```



In [ ]: