In [1]:

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all

import os

import random
#For dealing with tables
import pandas as pd
#For dealing with linear algebra
import numpy as np
#For data visualizs
```

Note: This is just a basic solution to help you understand the flow. Please do re-iterations and make an optimal sol.

In [2]:

```
_data=pd.read_csv('/Users/suraaj/Downloads/ola_driver_scaler_.csv')
```

In [3]:

```
_data.head()
```

Out[3]:

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

In [4]:

```
_data=_data.drop(columns='Unnamed: 0')
```

In [5]:

```
_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 [6]:

```
##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'])
```

In [7]:

```
_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 [8]:

```
_data.isnull().sum()/len(_data)*100
```

Out[8]:

```
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 [9]:

```
_data['Gender'].value_counts()
```

Out[9]:

```
0.0    11074
1.0     7978
Name: Gender, dtype: int64
```

In [10]:

```
_data['Education_Level'].value_counts()
```

Out[10]:

```
1    6864
2    6327
0    5913
Name: Education_Level, dtype: int64
```

KNN Imputation

In [11]:

```
_data_nums=_data.select_dtypes(np.number)
#keeping only the numerical columns
```

In [12]:

```
_data_nums
```

Out[12]:

	Driver_ID	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarte Ratii
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 [13]:

```
_data_nums.isnull().sum()
```

Out[13]:

```
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 [14]:

```
_data_nums.drop(columns='Driver_ID',inplace=True)
columns=_data_nums.columns
```

In [15]:

```
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5, weights='uniform', metric='nan_euclidean',)
imputer.fit(_data_nums)
# transform the dataset
_data_new = imputer.transform(_data_nums)
```

In [16]:

```
_data_new=pd.DataFrame(_data_new)
```

In [17]:

```
_data_new
```

Out[17]:

	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 [18]:

```
_data_new.columns=columns
```

In [19]:

```
_data_new.isnull().sum()
```

Out[19]:

```
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 [20]:

```
remaining_columns=list(set(_data.columns).difference(set(columns)))
```

In [21]:

```
data=pd.concat([_data_new, _data[remaining_columns]],axis=1)
```

In [22]:

```
data.head()
```

Out[22]:

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	Dateofjoini
0	28.0	0.0	2.0	57387.0	1.0	1.0	2381060.0	2.0	2018-12-
1	28.0	0.0	2.0	57387.0	1.0	1.0	-665480.0	2.0	2018-12-
2	28.0	0.0	2.0	57387.0	1.0	1.0	0.0	2.0	2018-12-
3	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	2020-11-
4	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	2020-11-

Checking if the concat is correct or not

In [23]:

```
data[data['Driver_ID']==2788]
```

Out[23]:

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	Dateof
19097	29.0	0.0	2.0	70254.0	2.0	2.0	0.0	1.0	2020
19098	30.0	0.0	2.0	70254.0	2.0	2.0	497690.0	3.0	2020
19099	30.0	0.0	2.0	70254.0	2.0	2.0	740280.0	3.0	2020
19100	30.0	0.0	2.0	70254.0	2.0	2.0	448370.0	3.0	2020
19101	30.0	0.0	2.0	70254.0	2.0	2.0	0.0	2.0	2020
19102	30.0	0.0	2.0	70254.0	2.0	2.0	200420.0	2.0	2020
19103	30.0	0.0	2.0	70254.0	2.0	2.0	411480.0	2.0	2020

In [24]:

```
_data[_data['Driver_ID']==2788]
```

Out[24]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorki
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 [25]:

```
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 [26]:

```
new_train
```

Out[26]:

		Age	Gender	City	Education_Level	Income	Joining Designation	Grade	Dateofjoinir
Driver_ID	**MMM-YY**								
1	**2019-01-01**	28.0	0.0	C23	2.0	57387.0	1.0	1.0	2018-12-:
	2019-02-01	28.0	0.0	C23	2.0	57387.0	1.0	1.0	2018-12-:
	2019-03-01	28.0	0.0	C23	2.0	57387.0	1.0	1.0	2018-12-:
2	**2020-11-01**	31.0	0.0	C7	2.0	67016.0	2.0	2.0	2020-11-(
	2020-12-01	31.0	0.0	C7	2.0	67016.0	2.0	2.0	2020-11-(
...	**...**	...	...	...	...	...	...	...	
2788	**2020-08-01**	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-06-(
	2020-09-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-06-(
	2020-10-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-06-(
	2020-11-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-06-(
	2020-12-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-06-(

19104 rows × 11 columns

In [27]:

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

In [28]:

```
#you can skip the "group by" code if you do sort_values directly
```

In [29]:

```
df.head(10)
```

Out[29]:

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

In [30]:

```
df1=pd.DataFrame()
```

In [31]:

```
df1['Driver_ID']=data['Driver_ID'].unique()
```

In [32]:

```
del _data
```

Aggregation at Driver Level

In [33]:

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

In [34]:

```
df1.head()
```

Out[34]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_Business
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	171
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	35
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	12
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	126

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 [35]:

```
#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'})

#The dataset which has the employee ids and a bollean value which tells if the ratin
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 [ ]:

```
#alternative
```

In [ ]:

```
#np.where(temp_qtrly_rating['Last_Quarterly_Rating'] - temp_qtrly_rating['First_Quar
```

In [36]:

```
df1
```

Out[36]:

Joining_Designation	Grade	Total_Business_Value	Last_Quarterly_Rating	Quarterly_Rating_Increase
1.0	1.0	1715580.0	2.0	
2.0	2.0	0.0	1.0	
2.0	2.0	350000.0	1.0	
1.0	1.0	120360.0	1.0	
3.0	3.0	1265000.0	2.0	
...	...	...	...	.
2.0	3.0	21748820.0	4.0	
1.0	1.0	0.0	1.0	
2.0	2.0	2815090.0	1.0	
1.0	1.0	977830.0	1.0	
2.0	2.0	2298240.0	2.0	

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 [37]:

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

Out[37]:

```
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 [38]:

```
lwr = (df.groupby('Driver_ID').agg({'LastWorkingDate':'last'})['LastWorkingDate'].is

#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 [ ]:

```
#df4[['target']] = np.where(pd.notnull(df4[['LastWorkingDate']]), 1, 0)
```

In [ ]:

```
#Driver_ID_df['LastWorkingDate'].apply(lambda x: 0 if x == None else 1)
```

In [39]:

```
df1
```

Out[39]:

esignation	Grade	Total_Business_Value	Last_Quarterly_Rating	Quarterly_Rating_Increased	Target
1.0	1.0	1715580.0	2.0	0	1
2.0	2.0	0.0	1.0	0	0
2.0	2.0	350000.0	1.0	0	1
1.0	1.0	120360.0	1.0	0	1
3.0	3.0	1265000.0	2.0	1	0
...	...	...	...	...	...
2.0	3.0	21748820.0	4.0	1	0
1.0	1.0	0.0	1.0	0	1
2.0	2.0	2815090.0	1.0	0	1
1.0	1.0	977830.0	1.0	0	1
2.0	2.0	2298240.0	2.0	1	0

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 [40]:

```
#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'})

#The dataset which has the employee ids and a bollean value which tells if the month
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 [41]:

```
df1['Income_Increased'].value_counts()
```

Out[41]:

```
0    2338
1      43
Name: Income_Increased, dtype: int64
```

In [42]:

```
df1.head()
```

Out[42]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_Business
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	171
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	35
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	12
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	126

Statistical Summary

In [44]:

```
df1.describe().T
```

Out[44]:

	count	mean	std	min	25%	50%
Driver_ID	2381.0	1.397559e+03	8.061616e+02	1.0	695.0	1400.0
Age	2381.0	3.377018e+01	5.933265e+00	21.0	30.0	33.0
Gender	2381.0	4.105838e-01	4.914963e-01	0.0	0.0	0.0
Education	2381.0	1.007560e+00	8.162900e-01	0.0	0.0	1.0
Income	2381.0	5.933416e+04	2.838367e+04	10747.0	39104.0	55315.0
Joining_Designation	2381.0	1.820244e+00	8.414334e-01	1.0	1.0	2.0
Grade	2381.0	2.096598e+00	9.415218e-01	1.0	1.0	2.0
Total_Business_Value	2381.0	4.586742e+06	9.127115e+06	-1385530.0	0.0	817680.0
Last_Quarterly_Rating	2381.0	1.427971e+00	8.098389e-01	1.0	1.0	1.0
Quarterly_Rating_Increased	2381.0	1.503570e-01	3.574961e-01	0.0	0.0	0.0
Target	2381.0	6.787064e-01	4.670713e-01	0.0	0.0	1.0
Income_Increased	2381.0	1.805964e-02	1.331951e-01	0.0	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 [45]:

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

Out[45]:

	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 [46]:

```
df1['Target'].value_counts()
```

Out[46]:

```
1    1616
0     765
Name: Target, dtype: int64
```

Out of 2381 drivers, 1616 drivers have left the organization.

In [47]:

```
df1['Target'].value_counts(normalize=True)*100
```

Out[47]:

```
1    67.870643
0    32.129357
Name: Target, dtype: float64
```

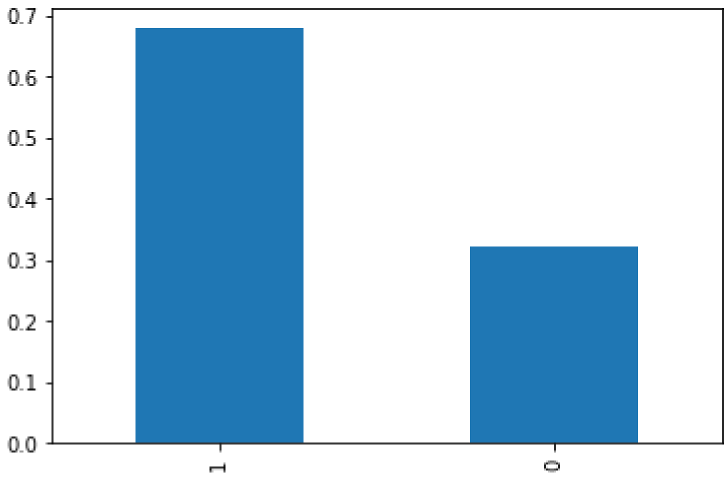
Around 68% driver have left the organization.

In [48]:

```
df1['Target'].value_counts(normalize=True).plot(kind='bar')
```

Out[48]:

<AxesSubplot:>



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

In [49]:

```
#Count of observations in each category
n = ['Gender','City','Education','Joining_Designation','Grade','Last_Quarterly_Ratir

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 [50]:

```
#Proportion of observations in each category
n = ['Gender','City','Education','Joining_Designation','Grade','Last_Quarterly_Ratir

for i in n:
    print(df1[i].value_counts(normalize=True))
    print("-------------------------------------------------------------")
```

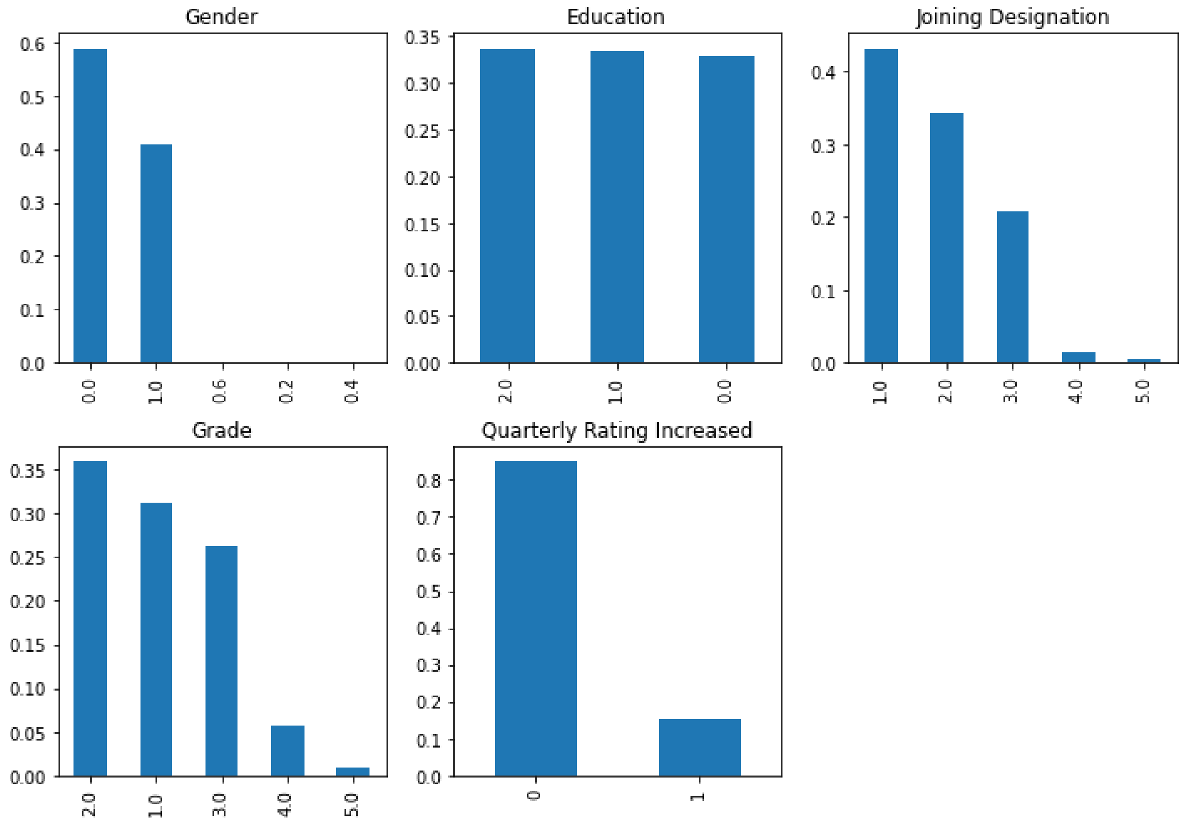
```
0.0    0.587988
1.0    0.409492
0.6    0.001260
0.2    0.000840
0.4    0.000420
Name: Gender, dtype: float64
-------------------------------------------------------------
C20    0.063839
C15    0.042419
C29    0.040319
C26    0.039059
C8     0.037379
C27    0.037379
C10    0.036119
C16    0.035279
C22    0.034439
C3     0.034439
C28    0.034439
C12    0.034019
C5     0.033599
C1     0.033599
C21    0.033179
C14    0.033179
C6     0.032759
C4     0.032339
C7     0.031919
C9     0.031499
C25    0.031079
C23    0.031079
C24    0.030659
C19    0.030239
C2     0.030239
C17    0.029819
C13    0.029819
C18    0.028979
C11    0.026879
Name: City, dtype: float64
-------------------------------------------------------------
2.0    0.336833
1.0    0.333893
0.0    0.329273
Name: Education, dtype: float64
-------------------------------------------------------------
1.0    0.430911
2.0    0.342293
3.0    0.207056
4.0    0.015120
5.0    0.004620
Name: Joining_Designation, dtype: float64
-------------------------------------------------------------
2.0    0.359093
1.0    0.311214
```

```
3.0    0.261655
4.0    0.057959
5.0    0.010080
Name: Grade, dtype: float64
------------------------------------------------------------
1.0    0.732465
2.0    0.152037
3.0    0.070559
4.0    0.044939
Name: Last_Quarterly_Rating, dtype: float64
------------------------------------------------------------
0    0.849643
1    0.150357
Name: Quarterly_Rating_Increased, dtype: float64
------------------------------------------------------------
```

- **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 [51]:

```
er','City','Joining_Designation','Grade','Last_Quarterly_Rating','Quarterly_Rating_In

ts(figsize=(10,7))
t(231)
r'].value_counts(normalize=True).plot.bar(title='Gender')

t(232)
tion'].value_counts(normalize=True).plot.bar(title='Education')

t(233)
ng_Designation'].value_counts(normalize=True).plot.bar(title='Joining Designation')

t(234)
'].value_counts(normalize=True).plot.bar(title='Grade')

t(235)
Quarterly_Rating'].value_counts(normalize=True).plot.bar(title='Last Quarterly Ratin

t(235)
erly_Rating_Increased'].value_counts(normalize=True).plot.bar(title='Quarterly Ratin
layout()
```

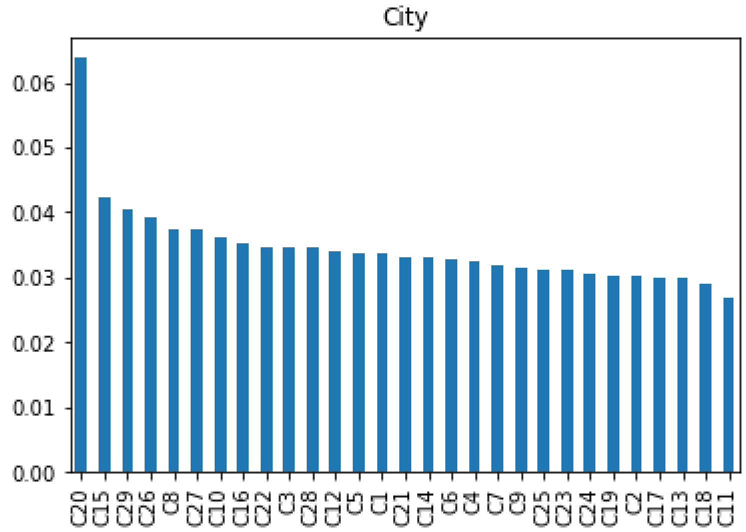


In [52]:

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

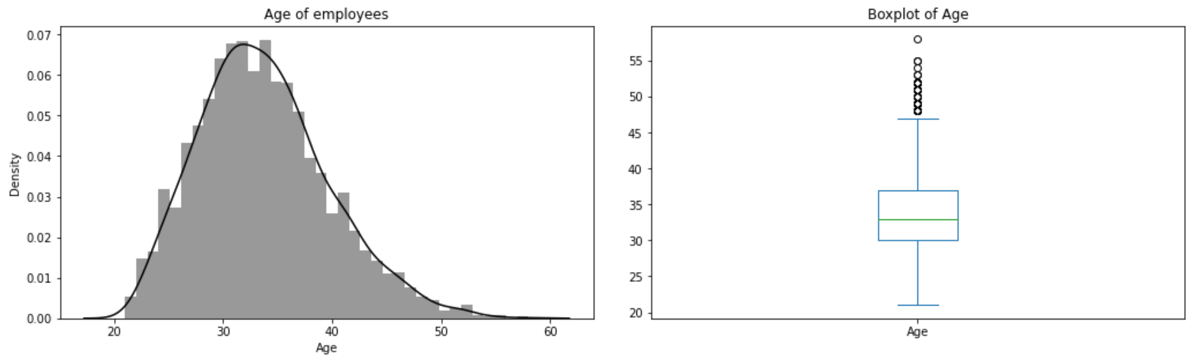
Out[52]:

```
<AxesSubplot:title={'center':'City'}>
```



In [53]:

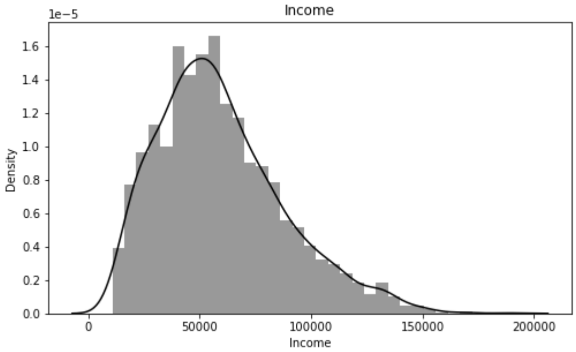
```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.distplot(df1['Age'],color='black')
plt.title("Age of employees")
plt.subplot(122)
df1['Age'].plot.box(title='Boxplot of Age')
plt.tight_layout(pad=3)
```

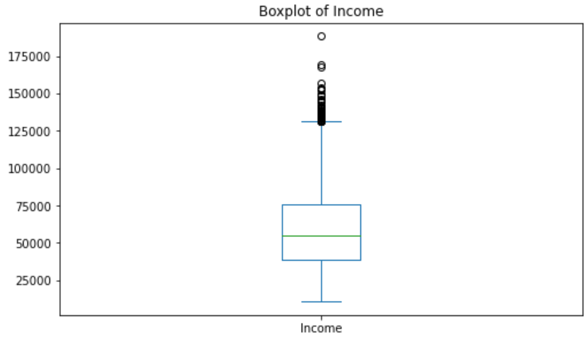


There are few outliers in the Age. The distribution is towards the right.

In [54]:

```
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)
```

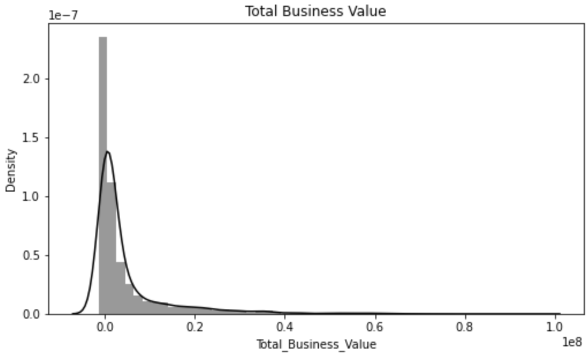


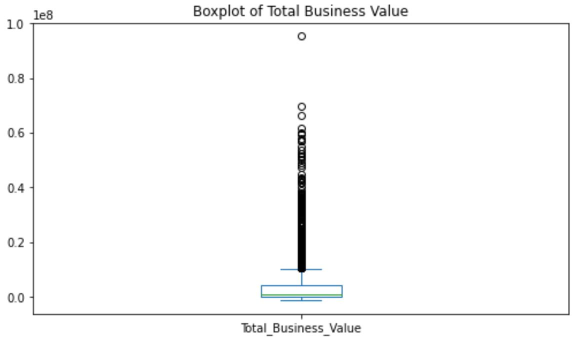


The distribution of Salary is towards the right and there are outliers for this feature as well.

In [55]:

```
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)
```

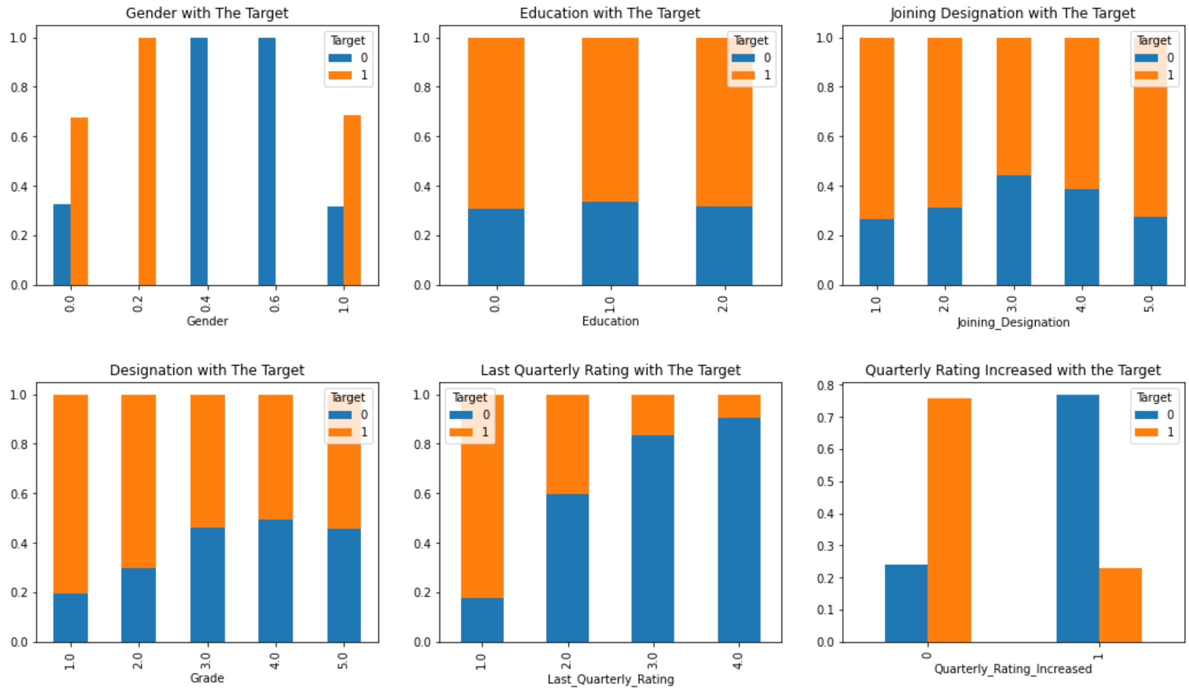




The distribution of total business value is towards the right. There are a lot of outliers for the feature Total Business Value.

In [56]:

```
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=False,ax=axes

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

#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=axes[0,2],
                                                     title="Joining Designation wit

#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,ax=axes[1,
                                                     title="Designation with The Targ

#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,ax=axes[
                                                     title="Last Quarterly Rating w

#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=False,ax=axes
                                                     title="Quarterly Rating Inc

plt.tight_layout(pad=3)
```

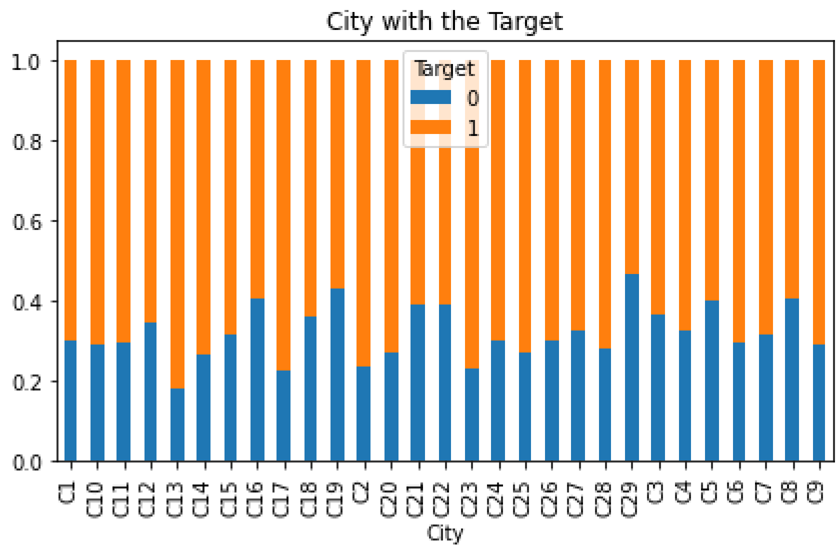


- **The proportion of gender and education is more or less the same for both the employees who left the organization and those who did not leave.**
- **The employees who have their grade as 3 or 4 at the time of joining are less likely to leave the organization.**
- **The employees who have their grade 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 organization.**

In [57]:

```
#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,title="City
plt.tight_layout()
```

```
<Figure size 2160x504 with 0 Axes>
```

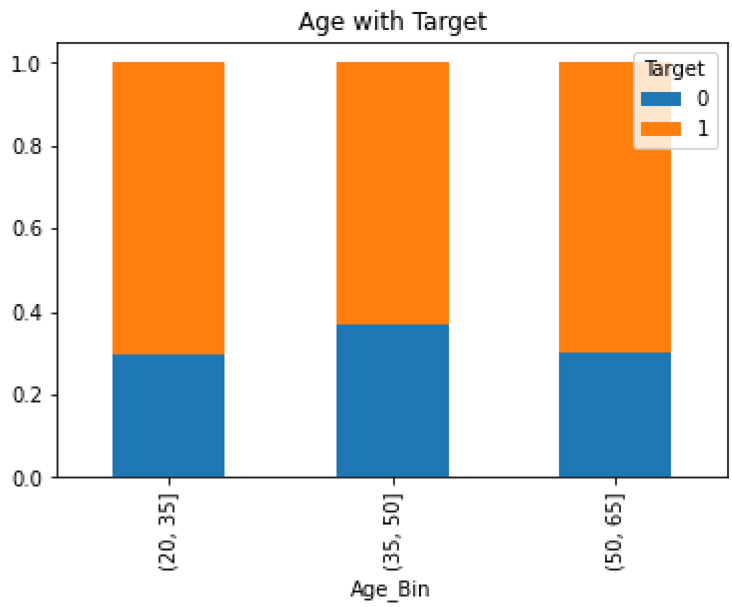


In [58]:

```
#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,title="A
```

Out[58]:

```
<AxesSubplot:title={'center':'Age with Target'}, xlabel='Age_Bin'>
```



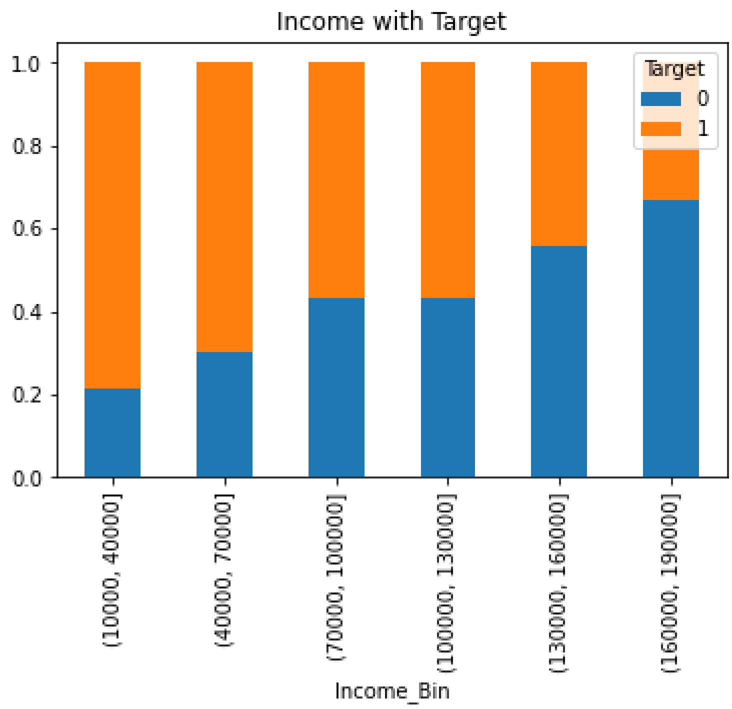
The employees whose age is in the 20-35 or 50-65 groups are less likely to leave the organization.

In [59]:

```
#Binning the Income into categories
df1['Income_Bin'] = pd.cut(df1['Income'],bins=[10000, 40000, 70000, 100000, 130000,

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

Out[59]:

```
<AxesSubplot:title={'center':'Income with Target'}, xlabel='Income_Bi
n'>
```



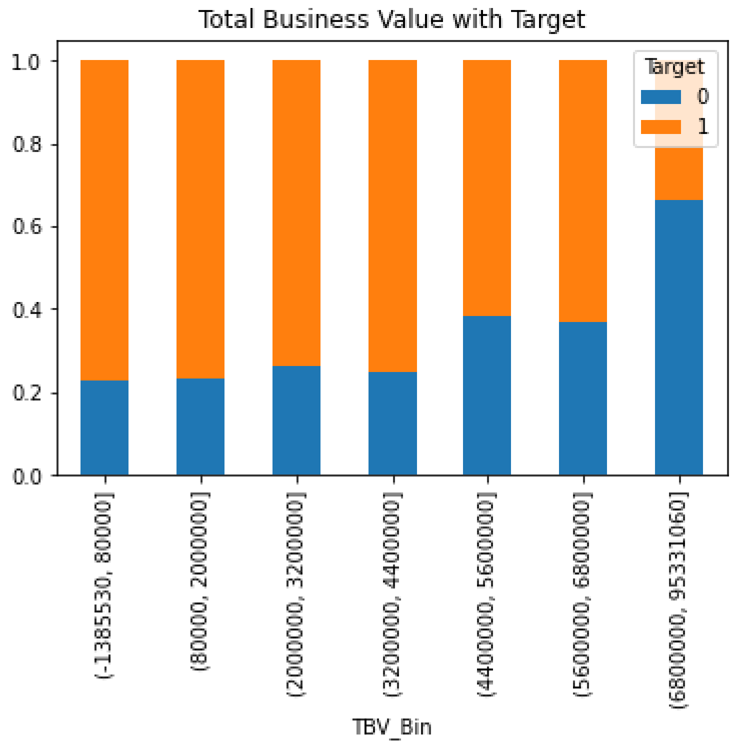
The employees whose monthly income is in 1,60,000-1,90,000 or 1,30,000-1,60,000 are less likely to leave the organization.

In [60]:

```
#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,title="T
```

Out[60]:

```
<AxesSubplot:title={'center':'Total Business Value with Target'}, xlab
el='TBV_Bin'>
```



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

In [61]:

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

In [62]:

```
df1.head()
```

Out[62]:

Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_Business_Valu
1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	1715580.
2	31.0	0.0	C7	2.0	67016.0	2.0	2.0	0.
4	43.0	0.0	C13	2.0	65603.0	2.0	2.0	350000.
5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	120360.
6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	1265000.

Step:One Hot Encoding

Converting categorical variables to numeric values so that Machine Learning models can be applied.

Alternatively, we can do "Target" Imputation

In [63]:

```

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 [64]:

```
df1
```

Out[64]:

Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_Business_Val
1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	171558(
2	31.0	0.0	C7	2.0	67016.0	2.0	2.0	(
4	43.0	0.0	C13	2.0	65603.0	2.0	2.0	35000(
5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	12036(
6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	126500(
...	...	...	...	...	...	...	...	
2784	34.0	0.0	C24	0.0	82815.0	2.0	3.0	2174882(
2785	34.0	1.0	C9	0.0	12105.0	1.0	1.0	(
2786	45.0	0.0	C19	0.0	35370.0	2.0	2.0	281509(
2787	28.0	1.0	C20	2.0	69498.0	1.0	1.0	97783(
2788	30.0	0.0	C27	2.0	70254.0	2.0	2.0	229824(

'ows × 42 columns

In [65]:

```
#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 [66]:

```
X=pd.DataFrame(X)
```

In [67]:

```
X.columns=X_cols
```

In [68]:

```
X
```

Out[68]:

	Age	Gender	Education	Income	Joining_Designation	Grade	Total_Business_Value
0	0.189189	0.0	1.0	0.262508	0.00	0.00	0.032064
1	0.270270	0.0	1.0	0.316703	0.25	0.25	0.014326
2	0.594595	0.0	1.0	0.308750	0.25	0.25	0.017944
3	0.216216	0.0	0.0	0.200489	0.00	0.00	0.015570
4	0.270270	1.0	0.5	0.382623	0.50	0.50	0.027405
...	...	...	...	...	...	...	...
2376	0.351351	0.0	0.0	0.405626	0.25	0.50	0.239197
2377	0.351351	1.0	0.0	0.007643	0.00	0.00	0.014326
2378	0.648649	0.0	0.0	0.138588	0.25	0.25	0.043432
2379	0.189189	1.0	1.0	0.330673	0.00	0.00	0.024436
2380	0.243243	0.0	1.0	0.334928	0.25	0.25	0.038088

2381 rows × 39 columns

In [69]:

```
#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, random_s
```

In [70]:

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[70]:

```
((1904, 39), (477, 39), (1904,), (477,))
```

Random Forest with class weights

In [71]:

```
from sklearn.utils import class_weight
```

In [72]:

```
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': 4, 'n_estimators': 50}
The score is : 0.8565554147554608
              precision    recall  f1-score   support

           0       0.70      0.57      0.63       148
           1       0.82      0.89      0.86       329

    accuracy                           0.79       477
   macro avg       0.76      0.73      0.74       477
weighted avg       0.79      0.79      0.79       477

[[ 85  63]
 [ 36 293]]
```

- 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': 4, 'n_estimators': 100}
The score is : 0.8590413497245795
              precision    recall  f1-score   support

           0       0.69      0.58      0.63       148
           1       0.82      0.88      0.85       329

    accuracy                           0.79       477
   macro avg       0.76      0.73      0.74       477
weighted avg       0.78      0.79      0.78       477

[[ 86  62]
 [ 38 291]]
```

- 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).

No Need for now ## Step-6:Balancing the dataset using SMOTE

Since the Dataset is imbalance and is biased towards target=1, so we will use SMOTE to balance the dataset

In [74]:

```
#print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
#print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train == 0)))

# import SMOTE module from imblearn library
# pip install imblearn (if you don't have imblearn in your system)
#from imblearn.over_sampling import SMOTE
#sm = SMOTE(random_state = 7)
#X_train, y_train = sm.fit_resample(X_train, y_train.ravel())

#print('After OverSampling, the shape of train_X: {}'.format(X_train.shape))
#print('After OverSampling, the shape of train_y: {} \n'.format(y_train.shape))

#print("After OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
#print("After OverSampling, counts of label '0': {}".format(sum(y_train == 0)))
```

Random Forest Classifier

In [75]:

```
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)
```

```
Best parameters are : {'max_depth': 4, 'n_estimators': 200}
The score is : 0.8580705129381029
```

In [76]:

```
pred = c.predict(X_test)
print(classification_report(y_test,pred))
print(confusion_matrix(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.70      0.57      0.63       148
           1       0.82      0.89      0.86       329

    accuracy                           0.79       477
   macro avg       0.76      0.73      0.74       477
weighted avg       0.79      0.79      0.79       477

[[ 85  63]
 [ 36 293]]
```

- The Random Forest method out of all predicted 0 the measure of correctly predicted is 72%, and for 1 it is 82%(Precision).
- The Random Forest method out of all actual 0 the measure of correctly predicted is 57%, and for 1 it is 90%(Recall).

XGBoost Classifier

In [79]:

```
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)
```

```
[22:20:43] WARNING: /Users/runner/work/xgboost/xgboost/python-package/
build/temp.macosx-10.9-x86_64-3.7/xgboost/src/learner.cc:627:
Parameters: { "class_weight" } might not be used.

  This could be a false alarm, with some parameters getting used by la
nguage bindings but
  then being mistakenly passed down to XGBoost core, or some parameter
actually being used
  but getting flagged wrongly here. Please open an issue if you find a
ny such cases.


              precision    recall  f1-score   support

           0       0.67      0.56      0.61       148
           1       0.82      0.88      0.85       329

    accuracy                           0.78       477
   macro avg       0.75      0.72      0.73       477
weighted avg       0.77      0.78      0.77       477

[[ 83  65]
 [ 40 289]]
```

- The XGBoost method out of all predicted 0 the measure of correctly predicted is 60%, and for 1 it is 82% (Precision).
- The XGBoost method out of all actual 0 the measure of correctly predicted is 59%, and for 1 it is 82% (Recall).

Decision Tree Classifier

In [80]:

```
# 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       0.55      0.54      0.54       148
           1       0.79      0.80      0.80       329

    accuracy                           0.72       477
   macro avg       0.67      0.67      0.67       477
weighted avg       0.72      0.72      0.72       477

[[ 80  68]
 [ 66 263]]
```

- The Decision Tree method out of all predicted 0 the measure of correctly predicted is 47%, and for 1 it is 79%(Precision).
- The Decision Tree method out of all actual 0 the measure of correctly predicted is 57%, and for 1 it is 71% (Recall).

Step-8: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 [81]:

```
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': 4, 'n_estimators': 200}
The score is : 0.8580705129381029
```

In [82]:

```
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.estimators_], axis
elapsed_time = time.time() - start_time

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

```
Elapsed time to compute the importances: 0.026 seconds
```

In [91]:

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

Out[91]:

	0	1
7	Last_Quarterly_Rating	0.087235
6	Total_Business_Value	0.056945
8	Quarterly_Rating_Increased	0.055344
3	Income	0.023209
5	Grade	0.019051
0	Age	0.016031
4	Joining_Designation	0.015579
2	Education	0.010235
9	Income_Increased	0.006744
1	Gender	0.006630
17	City_C16	0.005214
13	City_C12	0.004325
20	City_C19	0.004219
31	City_C29	0.004196
28	City_C26	0.004119
27	City_C25	0.004016
32	City_C3	0.003944
24	City_C22	0.003869
22	City_C20	0.003856
36	City_C7	0.003767
26	City_C24	0.003763
21	City_C2	0.003742
35	City_C6	0.003665
34	City_C5	0.003618
23	City_C21	0.003609
15	City_C14	0.003595
11	City_C10	0.003594
29	City_C27	0.003528
25	City_C23	0.003495
18	City_C17	0.003450
12	City_C11	0.003374
33	City_C4	0.003363
37	City_C8	0.003355

	0	1
10	City_C1	0.003335
38	City_C9	0.003205
14	City_C13	0.003154
30	City_C28	0.002813
16	City_C15	0.002771
19	City_C18	0.002526

In [88]:

```
pd.DataFrame(zip(X_train.columns,importances)).sort_values(by=[1], ascending=False)
```

Out[88]:

	0	1
6	Total_Business_Value	0.180056
3	Income	0.145222
7	Last_Quarterly_Rating	0.142459
0	Age	0.110757
8	Quarterly_Rating_Increased	0.060083
4	Joining_Designation	0.041801
2	Education	0.040969
5	Grade	0.036618
1	Gender	0.025986
17	City_C16	0.010301
24	City_C22	0.009637
32	City_C3	0.008895
22	City_C20	0.008872
13	City_C12	0.008812
20	City_C19	0.008748
23	City_C21	0.008735
28	City_C26	0.008626
34	City_C5	0.007886
31	City_C29	0.007859
27	City_C25	0.007729
36	City_C7	0.007692
33	City_C4	0.007358
35	City_C6	0.007179
10	City_C1	0.007137
26	City_C24	0.006790
18	City_C17	0.006757
37	City_C8	0.006625
29	City_C27	0.006586
21	City_C2	0.006494
11	City_C10	0.006402
16	City_C15	0.006390
25	City_C23	0.006328
14	City_C13	0.006056

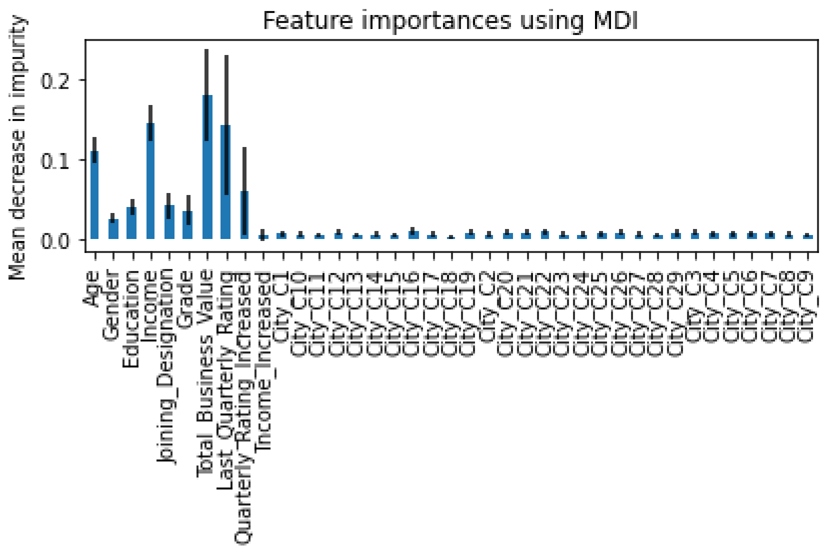
	0	1
15	City_C14	0.006042
12	City_C11	0.005969
38	City_C9	0.005621
30	City_C28	0.005319
9	Income_Increased	0.005152
19	City_C18	0.004053

In [ ]:

```
#Bonus
```

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 [ ]:

```
#limitations of impurity-based feature importances: Fix it
https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```