Objective :

- Ola, 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

In [26]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```python
ola = pd.read_csv(r"C:\Users\Acer\Downloads\ola_driver_scaler.csv")
```

Column Profiling:

1. MMMM-YY : Reporting Date (Monthly)
2. Driver_ID : Unique id for drivers
3. Age : Age of the driver
4. Gender : Gender of the driver – Male : 0, Female: 1
5. City : City Code of the driver
6. Education_Level : Education level – 0 for 10+ ,1 for 12+ ,2 for graduate
7. Income : Monthly average Income of the driver
8. Date Of Joining : Joining date for the driver
9. LastWorkingDate : Last date of working for the driver
10. Joining Designation : Designation of the driver at the time of joining
11. Grade : Grade of the driver at the time of reporting
12. Total Business Value : The total business value acquired by the driver in a month (negative business indicates cancellation/refund or car EMI adjustments)
13. Quarterly Rating : Quarterly rating of the driver: 1,2,3,4,5 (higher is better)

In [4]:
```
ola
```

Out[4]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	2381060
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	-665480
2	2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18	03/11/19	1	1	0
3	3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2	0
4	4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
19099	19099	08/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2	740280
19100	19100	09/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2	448370
19101	19101	10/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2	0
19102	19102	11/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2	200420
19103	19103	12/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2	411480

19104 rows × 14 columns

Objective :

Ola, 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

Observation 1:

- Here the Target variable is not shown explicitly
- from the observation the last working day can be made the Target column
- We need to convert the nan values to 0 and others to 1 maybe , need to converm later

Problems :

- here the data is not given in driver id terms , its a monthly data which is not useful for our analysis.
- will have to figure ways to make this to driver id terms for analysis

1. Import the dataset and do usual exploratory analysis steps like checking the structure

```
In [5]: df = ola.copy()
```

```
In [9]: df.shape
```

```
Out[9]: (19104, 14)
```

Observation 2

- in the description there are only 14 columns but in the dataset there is 15 :
- unnamed column can be dropped

```
In [17]: df.drop(columns=['Unnamed: 0'],inplace = True)
```

In [22]:
```
df.describe(include= 'all')
```

Out[22]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation
count	19104	19104.000000	19043.000000	19052.000000	19104	19104.000000	19104.000000	19104	1616	19104.000000
unique	24	NaN	NaN	NaN	29	NaN	NaN	869	493	NaN
top	2019-01-01 00:00:00	NaN	NaN	NaN	C20	NaN	NaN	2015-07-23 00:00:00	2020-07-29 00:00:00	NaN
freq	1022	NaN	NaN	NaN	1008	NaN	NaN	192	70	NaN
first	2019-01-01 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	2013-04-01 00:00:00	2018-12-31 00:00:00	NaN
last	2020-12-01 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	2020-12-28 00:00:00	2020-12-28 00:00:00	NaN
mean	NaN	1415.591133	34.668435	0.418749	NaN	1.021671	65652.025126	NaN	NaN	1.690536
std	NaN	810.705321	6.257912	0.493367	NaN	0.800167	30914.515344	NaN	NaN	0.836984
min	NaN	1.000000	21.000000	0.000000	NaN	0.000000	10747.000000	NaN	NaN	1.000000
25%	NaN	710.000000	30.000000	0.000000	NaN	0.000000	42383.000000	NaN	NaN	1.000000
50%	NaN	1417.000000	34.000000	0.000000	NaN	1.000000	60087.000000	NaN	NaN	1.000000
75%	NaN	2137.000000	39.000000	1.000000	NaN	2.000000	83969.000000	NaN	NaN	2.000000
max	NaN	2788.000000	58.000000	1.000000	NaN	2.000000	188418.000000	NaN	NaN	5.000000

In [19]:
```
df.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
```

observation 3

- will have to convert MMM-YY to date time format
- will have to convert Dateofjoining to date time format
- last working date should also be converted to the same

2. Convert date-like features to their respective data type

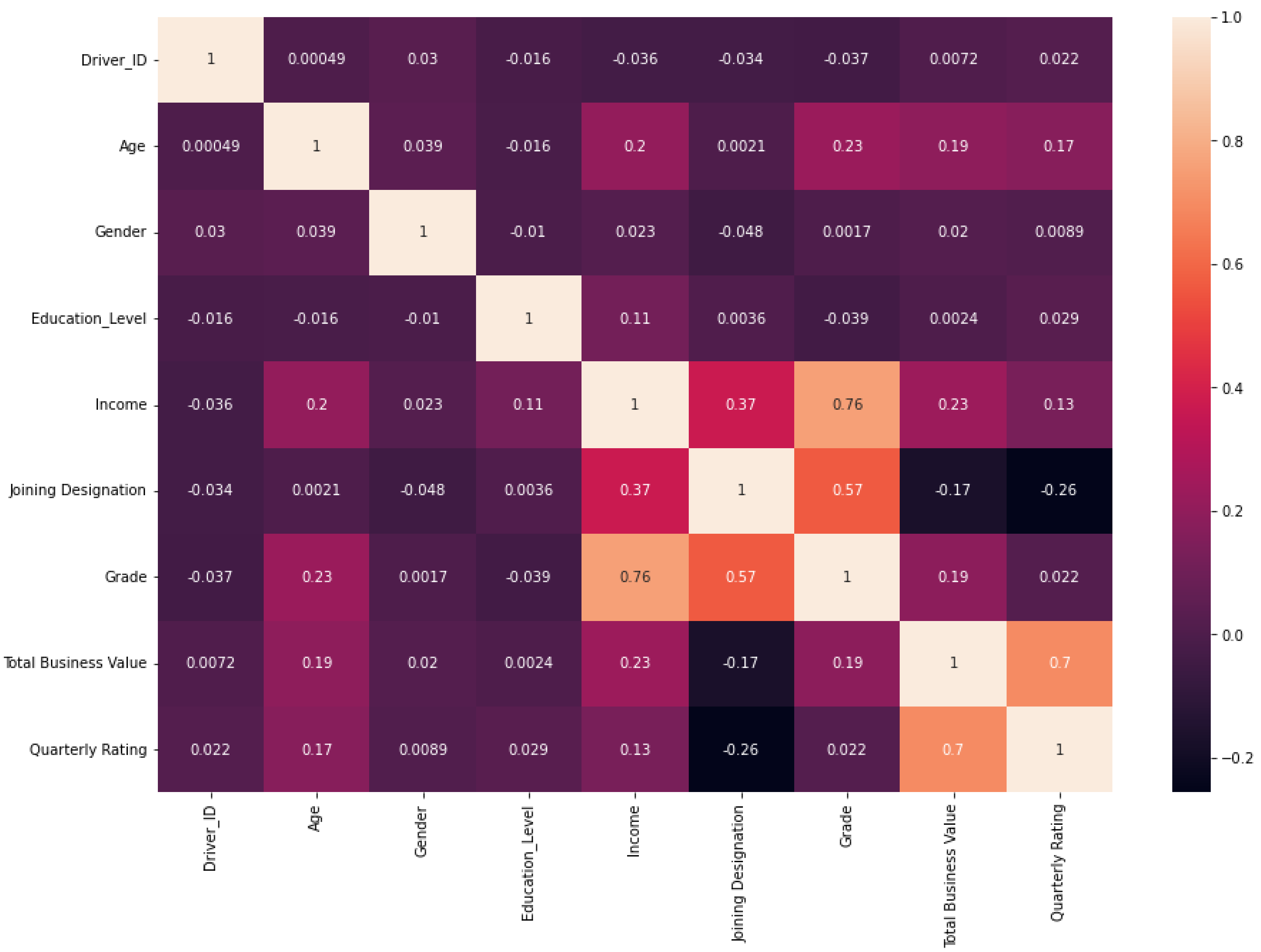
In [15]:
```
df['MMM-YY'] = pd.to_datetime(df['MMM-YY'])
df['Dateofjoining'] = pd.to_datetime(df['Dateofjoining'])
df['LastWorkingDate'] = pd.to_datetime(df['LastWorkingDate'])
```

In [20]:
```
df.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
```

In [27]:
```
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(method= 'spearman'), annot = True)
```

Out[27]:
```
<AxesSubplot:>
```



	Driver_ID	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating
Driver_ID	1	0.00049	0.03	-0.016	-0.036	-0.034	-0.037	0.0072	0.022
Age	0.00049	1	0.039	-0.016	0.2	0.0021	0.23	0.19	0.17
Gender	0.03	0.039	1	-0.01	0.023	-0.048	0.0017	0.02	0.0089
Education_Level	-0.016	-0.016	-0.01	1	0.11	0.0036	-0.039	0.0024	0.029
Income	-0.036	0.2	0.023	0.11	1	0.37	0.76	0.23	0.13
Joining Designation	-0.034	0.0021	-0.048	0.0036	0.37	1	0.57	-0.17	-0.26
Grade	-0.037	0.23	0.0017	-0.039	0.76	0.57	1	0.19	0.022
Total Business Value	0.0072	0.19	0.02	0.0024	0.23	-0.17	0.19	1	0.7
Quarterly Rating	0.022	0.17	0.0089	0.029	0.13	-0.26	0.022	0.7	1

Observation 4:

- there is no point in doing a heatmap now as the data is not on id level and there needs to be alot of work done

3. Check for missing values and Prepare data for KNN Imputation

You may consider only numerical features for this purpose

In [30]:
```
df.isnull().sum()/len(df)*100
```

Out[30]:
```
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
```

Observation 5:

- There are 31% missing data in the age column
- There are 27% missing data in the gender column
- There are 91% missing data in the lastworkingday column but this is the target variable and can be dealt with seperatly

In [33]:
```
# since its asked to take only numeric features for imputation , we are taking just that
# np.number is a new fuction for it
numericdf = df.select_dtypes(include=np.number)
```

In [34]:
```
numericdf
```

Out[34]:

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

19104 rows × 9 columns

In [37]:
```
numericdf.isnull().sum()/len(numericdf)*100
```

Out[37]:
```
Driver_ID              0.000000
Age                    0.319305
Gender                 0.272194
Education_Level        0.000000
Income                 0.000000
Joining Designation    0.000000
Grade                  0.000000
Total Business Value   0.000000
Quarterly Rating       0.000000
dtype: float64
```

KNN

In [38]:
```
from sklearn.impute import KNNImputer
```

- since the data needs to be aggregated on the basis of driver id , we dont need to impute the Driver id making it float and reconverting it back to int again. so we drop it

In [47]:
```
numericdf.drop(columns=['Driver_ID'],inplace = True)
numericdf
```

Out[47]:

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

19104 rows × 8 columns

In [48]:
```
imputer = KNNImputer(n_neighbors=5 , weights= 'uniform' , metric= 'nan_euclidean')
_numericdf = imputer.fit_transform(numericdf)
new_numericdf = pd.DataFrame(_numericdf)
```

In [50]:
```
# we got the output without the names of column
new_numericdf
```

Out[50]:

	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 [51]:
```
col_names = numericdf.columns
new_numericdf.columns = col_names
new_numericdf
```

Out[51]:

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating
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 [52]:
```
new_numericdf.isnull().sum()/len(numericdf)*100
```

Out[52]:
```
Age                     0.0
Gender                  0.0
Education_Level         0.0
Income                  0.0
Joining Designation     0.0
Grade                   0.0
Total Business Value    0.0
Quarterly Rating        0.0
dtype: float64
```

Observation 6:

- Now we have to stitch back the dataset back to original form as the imupation has been done
- for that we can use the join / merge or concat for this

In [61]:
```
dff
```

Out[61]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterly Rating
0	2019-01-01	1	28.0	0.0	C23	2	57387	2018-12-24	NaT	1	1	2381060	2
1	2019-02-01	1	28.0	0.0	C23	2	57387	2018-12-24	NaT	1	1	-665480	2
2	2019-03-01	1	28.0	0.0	C23	2	57387	2018-12-24	2019-03-11	1	1	0	2
3	2020-11-01	2	31.0	0.0	C7	2	67016	2020-11-06	NaT	2	2	0	1
4	2020-12-01	2	31.0	0.0	C7	2	67016	2020-11-06	NaT	2	2	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
19099	2020-08-01	2788	30.0	0.0	C27	2	70254	2020-06-08	NaT	2	2	740280	3
19100	2020-09-01	2788	30.0	0.0	C27	2	70254	2020-06-08	NaT	2	2	448370	3
19101	2020-10-01	2788	30.0	0.0	C27	2	70254	2020-06-08	NaT	2	2	0	2
19102	2020-11-01	2788	30.0	0.0	C27	2	70254	2020-06-08	NaT	2	2	200420	2
19103	2020-12-01	2788	30.0	0.0	C27	2	70254	2020-06-08	NaT	2	2	411480	2

19104 rows × 13 columns

In [57]:
```
new_numericdf.columns
```

Out[57]:
```
Index(['Age', 'Gender', 'Education_Level', 'Income', 'Joining Designation',
       'Grade', 'Total Business Value', 'Quarterly Rating'],
      dtype='object')
```

In [78]:
```
df.drop(columns=['Gender', 'Education_Level', 'Income', 'Joining Designation',
        'Grade', 'Total Business Value', 'Quarterly Rating'], axis=1, inplace=True)
```

In [64]:
```
new_numericdf
```

Out[64]:

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating
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 [79]:
```
df['Gender'] = new_numericdf['Gender']
df['Education_Level'] = new_numericdf['Education_Level']
df['Income'] = new_numericdf['Income']
df['Joining Designation'] = new_numericdf['Joining Designation']
df['Grade'] = new_numericdf['Grade']
df['Total Business Value'] = new_numericdf['Total Business Value']
df['Quarterly Rating'] = new_numericdf['Quarterly Rating']
```

In [80]:
```
df
```

Out[80]:

	MMM-YY	Driver_ID	Age	City	Dateofjoining	LastWorkingDate	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating
0	2019-01-01	1	28.0	C23	2018-12-24	NaT	0.0	2.0	57387.0	1.0	1.0	2381060.0	2.0
1	2019-02-01	1	28.0	C23	2018-12-24	NaT	0.0	2.0	57387.0	1.0	1.0	-665480.0	2.0
2	2019-03-01	1	28.0	C23	2018-12-24	2019-03-11	0.0	2.0	57387.0	1.0	1.0	0.0	2.0
3	2020-11-01	2	31.0	C7	2020-11-06	NaT	0.0	2.0	67016.0	2.0	2.0	0.0	1.0
4	2020-12-01	2	31.0	C7	2020-11-06	NaT	0.0	2.0	67016.0	2.0	2.0	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
19099	2020-08-01	2788	30.0	C27	2020-06-08	NaT	0.0	2.0	70254.0	2.0	2.0	740280.0	3.0
19100	2020-09-01	2788	30.0	C27	2020-06-08	NaT	0.0	2.0	70254.0	2.0	2.0	448370.0	3.0
19101	2020-10-01	2788	30.0	C27	2020-06-08	NaT	0.0	2.0	70254.0	2.0	2.0	0.0	2.0
19102	2020-11-01	2788	30.0	C27	2020-06-08	NaT	0.0	2.0	70254.0	2.0	2.0	200420.0	2.0
19103	2020-12-01	2788	30.0	C27	2020-06-08	NaT	0.0	2.0	70254.0	2.0	2.0	411480.0	2.0

19104 rows × 13 columns

Re_indexing to old format for ease of understanding

In [84]:
```
df = df.reindex(columns=['Driver_ID', 'MMM-YY', 'Age' , 'Gender' , 'City','Education_Level','Income','Joining Desig
      'LastWorkingDate',  'Total Business Value','Quarterly Rating'])
```

In [85]:
```
df
```

Out[85]:

	Driver_ID	MMM-YY	Age	Gender	City	Education_Level	Income	Joining Designation	Grade	Dateofjoining	LastWorkingDate	Total Business Value	Quarterly Rating
0	1	2019-01-01	28.0	0.0	C23	2.0	57387.0	1.0	1.0	2018-12-24	NaT	2381060.0	2.0
1	1	2019-02-01	28.0	0.0	C23	2.0	57387.0	1.0	1.0	2018-12-24	NaT	-665480.0	2.0
2	1	2019-03-01	28.0	0.0	C23	2.0	57387.0	1.0	1.0	2018-12-24	2019-03-11	0.0	2.0
3	2	2020-11-01	31.0	0.0	C7	2.0	67016.0	2.0	2.0	2020-11-06	NaT	0.0	1.0
4	2	2020-12-01	31.0	0.0	C7	2.0	67016.0	2.0	2.0	2020-11-06	NaT	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
19099	2788	2020-08-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-06-08	NaT	740280.0	3.0
19100	2788	2020-09-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-06-08	NaT	448370.0	3.0
19101	2788	2020-10-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-06-08	NaT	0.0	2.0
19102	2788	2020-11-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-06-08	NaT	200420.0	2.0
19103	2788	2020-12-01	30.0	0.0	C27	2.0	70254.0	2.0	2.0	2020-06-08	NaT	411480.0	2.0

19104 rows × 13 columns

4. Aggregate data in order to remove multiple occurrences of same driver data (We did something similar in Delhivery business Case)

You can start from storing unique Driver IDs in an empty dataframe and then bring all the features at same level (Groupby Driver ID)

- Observation 7:
  - Here the data can be aggregated on the basis of driver ID and month which leaves us with the question of what to do with the other columns
  - for each column we can take different strategies like :

```
- Age : max
- MMM-YY : last
- Gender : first
- Education_level : last
- Income : last
- Joining Designation : last
- Grade : last
- Dateofjoining : last
- last working day : last
- Total business value : sum
- Quarterly Rating : last
```

In [122]:
```
dff = df.pivot_table(index= ['Driver_ID' , 'City' , 'Education_Level'],

                    values= [ 'MMM-YY','Age', 'Gender', 'Income','Dateofjoining','LastWorkingDate', 'Joining Designa
                             'Grade', 'Total Business Value'],

                    aggfunc= {'Age' : 'max' ,'MMM-YY' : 'last', 'Gender' : 'first' ,
                              'Income' : 'last','Joining Designation' : 'last' ,'Grade' : 'last' ,
                              'Dateofjoining' : 'last','LastWorkingDate' : 'last',
                              'Total Business Value' : 'sum' })

dff.reset_index(inplace= True)
```

In [123]:
```
dff
```

Out[123]:

	Driver_ID	City	Education_Level	Age	Dateofjoining	Gender	Grade	Income	Joining Designation	LastWorkingDate	MMM-YY	Total Business Value
0	1	C23	2.0	28.0	2018-12-24	0.0	1.0	57387.0	1.0	2019-03-11	2019-03-01	1715580.0
1	2	C7	2.0	31.0	2020-11-06	0.0	2.0	67016.0	2.0	NaT	2020-12-01	0.0
2	4	C13	2.0	43.0	2019-12-07	0.0	2.0	65603.0	2.0	2020-04-27	2020-04-01	350000.0
3	5	C9	0.0	29.0	2019-01-09	0.0	1.0	46368.0	1.0	2019-03-07	2019-03-01	120360.0
4	6	C11	1.0	31.0	2020-07-31	1.0	3.0	78728.0	3.0	NaT	2020-12-01	1265000.0
...	...	...	...	...	...	...	...	...	...	...	...	...
2376	2784	C24	0.0	34.0	2015-10-15	0.0	3.0	82815.0	2.0	NaT	2020-12-01	21748820.0
2377	2785	C9	0.0	34.0	2020-08-28	1.0	1.0	12105.0	1.0	2020-10-28	2020-10-01	0.0
2378	2786	C19	0.0	45.0	2018-07-31	0.0	2.0	35370.0	2.0	2019-09-22	2019-09-01	2815090.0
2379	2787	C20	2.0	28.0	2018-07-21	1.0	1.0	69498.0	1.0	2019-06-20	2019-06-01	977830.0
2380	2788	C27	2.0	30.0	2020-06-08	0.0	2.0	70254.0	2.0	NaT	2020-12-01	2298240.0

2381 rows × 12 columns

Observation 7:

- this doesnot comtain the Quartely rating which has to be feature engineed and fitted later.
- the data are segregated and stiched to a driver level doing all the aggregation works mentioned above

5. Feature Engineering Steps:

Create a column which tells whether the quarterly rating has increased for that driver - for those whose quarterly rating has increased we assign the value 1

Target variable creation: Create a column called target which tells whether the driver has left the company- driver whose last working day is present will have the value 1

Create a column which tells whether the monthly income has increased for that driver - for those whose monthly income has increased we assign the value 1

5.1 Create a column which tells whether the quarterly rating has increased for that driver - for those whose quarterly rating has increased we assign the value 1

In [131]:
```
df['Quarterly Rating'].value_counts()
```

Out[131]:
```
1.0    7679
2.0    5553
3.0    3895
4.0    1977
Name: Quarterly Rating, dtype: int64
```

In [133]:
```
a = df.groupby('Driver_ID').agg({'Quarterly Rating' : 'first'})
a
```

Out[133]:

```
          Quarterly Rating
Driver_ID
        1              2.0
        2              1.0
        4              1.0
        5              1.0
        6              1.0
      ...              ...
     2784              3.0
     2785              1.0
     2786              2.0
     2787              2.0
     2788              1.0

2381 rows × 1 columns
```

In [135]:
```
b = df.groupby('Driver_ID').agg({'Quarterly Rating' : 'last'})
b
```

Out[135]:

Driver_ID	Quarterly Rating
1	2.0
2	1.0
4	1.0
5	1.0
6	2.0
...	...
2784	4.0
2785	1.0
2786	1.0
2787	1.0
2788	2.0

2381 rows × 1 columns

Observastion -

- we notice that our aggregated new dataset has 2381 rows and this quarterly also contails 2381 for both first and last
- so assign a value for those first and last are different ie if last > first = 1

In [138]:
```
c = (b['Quarterly Rating'] > a['Quarterly Rating']).reset_index()
```

Out[138]:

	Driver_ID	Quarterly Rating
0	1	False
1	2	False
2	4	False
3	5	False
4	6	True
...	...	...
2376	2784	True
2377	2785	False
2378	2786	False
2379	2787	False
2380	2788	True

2381 rows × 2 columns

In [143]:
```
# converting boolean into 1 and 0
c["Quarterly Rating"] = c["Quarterly Rating"].astype(int)
```

In [144]:
```
c
```

Out[144]:

	Driver_ID	Quarterly Rating
0	1	0
1	2	0
2	4	0
3	5	0
4	6	1
...	...	...
2376	2784	1
2377	2785	0
2378	2786	0
2379	2787	0
2380	2788	1

2381 rows × 2 columns

In [147]:
```
d = c["Quarterly Rating"].values
d
```

Out[147]:
```
array([0, 0, 0, ..., 0, 0, 1])
```

In [149]:
```
# instead of concating or joining we can do this.
dff['Increased_Quarterly_Rating'] = d
```

In [152]:
```
dff
```

Out[152]:
```
      Driver_ID City Education_Level  Age Dateofjoining Gender Grade  Income     Joining LastWorkingDate  MMM-      Total Increased_Q
                                                                             Designation                   YY   Business
                                                                                                                  Value
   0          1  C23             2.0 28.0    2018-12-24    0.0   1.0 57387.0         1.0      2019-03-11 2019-   1715580.0
                                                                                                           03-01
   1          2   C7             2.0 31.0    2020-11-06    0.0   2.0 67016.0         2.0             NaT 2020-         0.0
                                                                                                           12-01
   2          4  C13             2.0 43.0    2019-12-07    0.0   2.0 65603.0         2.0      2020-04-27 2020-    350000.0
                                                                                                           04-01
   3          5   C9             0.0 29.0    2019-01-09    0.0   1.0 46368.0         1.0      2019-03-07 2019-    120360.0
                                                                                                           03-01
   4          6  C11             1.0 31.0    2020-07-31    1.0   3.0 78728.0         3.0             NaT 2020-   1265000.0
                                                                                                           12-01
 ...        ...  ...             ...  ...           ...    ...   ...     ...         ...             ...   ...         ...
2376       2784  C24             0.0 34.0    2015-10-15    0.0   3.0 82815.0         2.0             NaT 2020-  21748820.0
                                                                                                           12-01
2377       2785   C9             0.0 34.0    2020-08-28    1.0   1.0 12105.0         1.0      2020-10-28 2020-         0.0
                                                                                                           10-01
2378       2786  C19             0.0 45.0    2018-07-31    0.0   2.0 35370.0         2.0      2019-09-22 2019-   2815090.0
                                                                                                           09-01
2379       2787  C20             2.0 28.0    2018-07-21    1.0   1.0 69498.0         1.0      2019-06-20 2019-    977830.0
                                                                                                           06-01
2380       2788  C27             2.0 30.0    2020-06-08    0.0   2.0 70254.0         2.0             NaT 2020-   2298240.0
                                                                                                           12-01

2381 rows × 13 columns
```

In [ ]:
```

```

In [ ]:
```

```

5.2 Target variable creation: Create a column called target which tells whether the driver has left the company- driver whose last working day is present will have the value 1

In [169]:
```
d = dff.copy()
```

In [186]:
```
d.groupby('Driver_ID').agg({'LastWorkingDate' : 'last'})['LastWorkingDate'].isna().reset_index().replace({False: 1, Tr

```

Out[186]:

	Driver_ID	LastWorkingDate
0	1	1
1	2	0
2	4	1
3	5	1
4	6	0
...	...	...
2376	2784	0
2377	2785	1
2378	2786	1
2379	2787	1
2380	2788	0

2381 rows × 2 columns

In [190]:
```
ddd = dd['LastWorkingDate'].values
```

Out[190]:
```
2381
```

In [191]:
```
dff['Target'] = ddd
```

In [200]:
```
# Droppped the last working day column now that we have feature engineered from it
dff.drop(['LastWorkingDate'],axis = 1, inplace = True)
```

In [198]:
```
dff
```

Out[198]:

	Driver_ID	City	Education_Level	Age	Dateofjoining	Gender	Grade	Income	Joining Designation	MMM-YY	Total Business Value	Increased_Quarterly_Rating	Ta
0	1	C23	2.0	28.0	2018-12-24	0.0	1.0	57387.0	1.0	2019-03-01	1715580.0	0	
1	2	C7	2.0	31.0	2020-11-06	0.0	2.0	67016.0	2.0	2020-12-01	0.0	0	
2	4	C13	2.0	43.0	2019-12-07	0.0	2.0	65603.0	2.0	2020-04-01	350000.0	0	
3	5	C9	0.0	29.0	2019-01-09	0.0	1.0	46368.0	1.0	2019-03-01	120360.0	0	
4	6	C11	1.0	31.0	2020-07-31	1.0	3.0	78728.0	3.0	2020-12-01	1265000.0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2376	2784	C24	0.0	34.0	2015-10-15	0.0	3.0	82815.0	2.0	2020-12-01	21748820.0	1	
2377	2785	C9	0.0	34.0	2020-08-28	1.0	1.0	12105.0	1.0	2020-10-01	0.0	0	
2378	2786	C19	0.0	45.0	2018-07-31	0.0	2.0	35370.0	2.0	2019-09-01	2815090.0	0	
2379	2787	C20	2.0	28.0	2018-07-21	1.0	1.0	69498.0	1.0	2019-06-01	977830.0	0	
2380	2788	C27	2.0	30.0	2020-06-08	0.0	2.0	70254.0	2.0	2020-12-01	2298240.0	1	

2381 rows × 13 columns

5.3 Create a column which tells whether the monthly income has increased for that driver - for those whose monthly income has increased we assign the value 1

In [193]:
```
dff
```

Out[193]:

	Driver_ID	City	Education_Level	Age	Dateofjoining	Gender	Grade	Income	Joining Designation	LastWorkingDate	MMM-YY	Total Business Value	Increasec
0	1	C23	2.0	28.0	2018-12-24	0.0	1.0	57387.0	1.0	2019-03-11	2019-03-01	1715580.0	
1	2	C7	2.0	31.0	2020-11-06	0.0	2.0	67016.0	2.0	NaT	2020-12-01	0.0	
2	4	C13	2.0	43.0	2019-12-07	0.0	2.0	65603.0	2.0	2020-04-27	2020-04-01	350000.0	
3	5	C9	0.0	29.0	2019-01-09	0.0	1.0	46368.0	1.0	2019-03-07	2019-03-01	120360.0	
4	6	C11	1.0	31.0	2020-07-31	1.0	3.0	78728.0	3.0	NaT	2020-12-01	1265000.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2376	2784	C24	0.0	34.0	2015-10-15	0.0	3.0	82815.0	2.0	NaT	2020-12-01	21748820.0	
2377	2785	C9	0.0	34.0	2020-08-28	1.0	1.0	12105.0	1.0	2020-10-28	2020-10-01	0.0	
2378	2786	C19	0.0	45.0	2018-07-31	0.0	2.0	35370.0	2.0	2019-09-22	2019-09-01	2815090.0	
2379	2787	C20	2.0	28.0	2018-07-21	1.0	1.0	69498.0	1.0	2019-06-20	2019-06-01	977830.0	
2380	2788	C27	2.0	30.0	2020-06-08	0.0	2.0	70254.0	2.0	NaT	2020-12-01	2298240.0	

2381 rows × 14 columns

In [224]:
```
a = df.groupby('Driver_ID').agg({'Income' : 'first'})
a

```

Out[224]:
```
          Income
Driver_ID
        1  57387.0
        2  67016.0
        4  65603.0
        5  46368.0
        6  78728.0
      ...      ...
     2784  82815.0
     2785  12105.0
     2786  35370.0
     2787  69498.0
     2788  70254.0

2381 rows × 1 columns
```

In [225]:
```python
b = df.groupby('Driver_ID').agg({'Income' : 'last'})
b
```

Out[225]:

	Income
Driver_ID	
1	57387.0
2	67016.0
4	65603.0
5	46368.0
6	78728.0
...	...
2784	82815.0
2785	12105.0
2786	35370.0
2787	69498.0
2788	70254.0

2381 rows × 1 columns

In [228]:
```
c = (b['Income'] > a['Income']).reset_index()
c
```

Out[228]:

	Driver_ID	Income
0	1	False
1	2	False
2	4	False
3	5	False
4	6	False
...	...	...
2376	2784	False
2377	2785	False
2378	2786	False
2379	2787	False
2380	2788	False

2381 rows × 2 columns

In [214]:
```
# converting boolean into 1 and 0
c["Income"] = c["Income"].astype(int)
c["Income"].value_counts()
```

Out[214]:
```
0    2338
1      43
Name: Income, dtype: int64
```

In [218]:
```
d = c["Income"].values
d
```

Out[218]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [219]:
```
# instead of concating or joining we can do this.
dff['Increased_Income'] = d
```

In [236]:
```
dff
```

Out[236]:

	Driver_ID	City	Education_Level	Age	Dateofjoining	Gender	Grade	Income	Joining Designation	MMM-YY	Total Business Value	Increased_Quarterly_Rating	Ta
0	1	C23	2.0	28.0	2018-12-24	0.0	1.0	57387.0	1.0	2019-03-01	1715580.0	0	
1	2	C7	2.0	31.0	2020-11-06	0.0	2.0	67016.0	2.0	2020-12-01	0.0	0	
2	4	C13	2.0	43.0	2019-12-07	0.0	2.0	65603.0	2.0	2020-04-01	350000.0	0	
3	5	C9	0.0	29.0	2019-01-09	0.0	1.0	46368.0	1.0	2019-03-01	120360.0	0	
4	6	C11	1.0	31.0	2020-07-31	1.0	3.0	78728.0	3.0	2020-12-01	1265000.0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2376	2784	C24	0.0	34.0	2015-10-15	0.0	3.0	82815.0	2.0	2020-12-01	21748820.0	1	
2377	2785	C9	0.0	34.0	2020-08-28	1.0	1.0	12105.0	1.0	2020-10-01	0.0	0	
2378	2786	C19	0.0	45.0	2018-07-31	0.0	2.0	35370.0	2.0	2019-09-01	2815090.0	0	
2379	2787	C20	2.0	28.0	2018-07-21	1.0	1.0	69498.0	1.0	2019-06-01	977830.0	0	
2380	2788	C27	2.0	30.0	2020-06-08	0.0	2.0	70254.0	2.0	2020-12-01	2298240.0	1	

2381 rows × 14 columns

6.Statistical summary of the derived dataset

6.Statistical summary of the derived dataset

In [243]:
```python
import seaborn as sns
```

In [238]:
```python
dff.describe()
```

Out[238]:

	Driver_ID	Education_Level	Age	Gender	Grade	Income	Joining Designation	Total Business Value	Increased_Quarterly_Rating
count	2381.000000	2381.00000	2381.000000	2381.000000	2381.000000	2381.000000	2381.000000	2.381000e+03	2381.000000
mean	1397.559009	1.00756	33.663167	0.411172	2.096598	59334.157077	1.820244	4.586742e+06	0.150357
std	806.161628	0.81629	5.983375	0.491740	0.941522	28383.666384	0.841433	9.127115e+06	0.357496
min	1.000000	0.00000	21.000000	0.000000	1.000000	10747.000000	1.000000	-1.385530e+06	0.000000
25%	695.000000	0.00000	29.000000	0.000000	1.000000	39104.000000	1.000000	0.000000e+00	0.000000
50%	1400.000000	1.00000	33.000000	0.000000	2.000000	55315.000000	2.000000	8.176800e+05	0.000000
75%	2100.000000	2.00000	37.000000	1.000000	3.000000	75986.000000	2.000000	4.173650e+06	0.000000
max	2788.000000	2.00000	58.000000	1.000000	5.000000	188418.000000	5.000000	9.533106e+07	1.000000

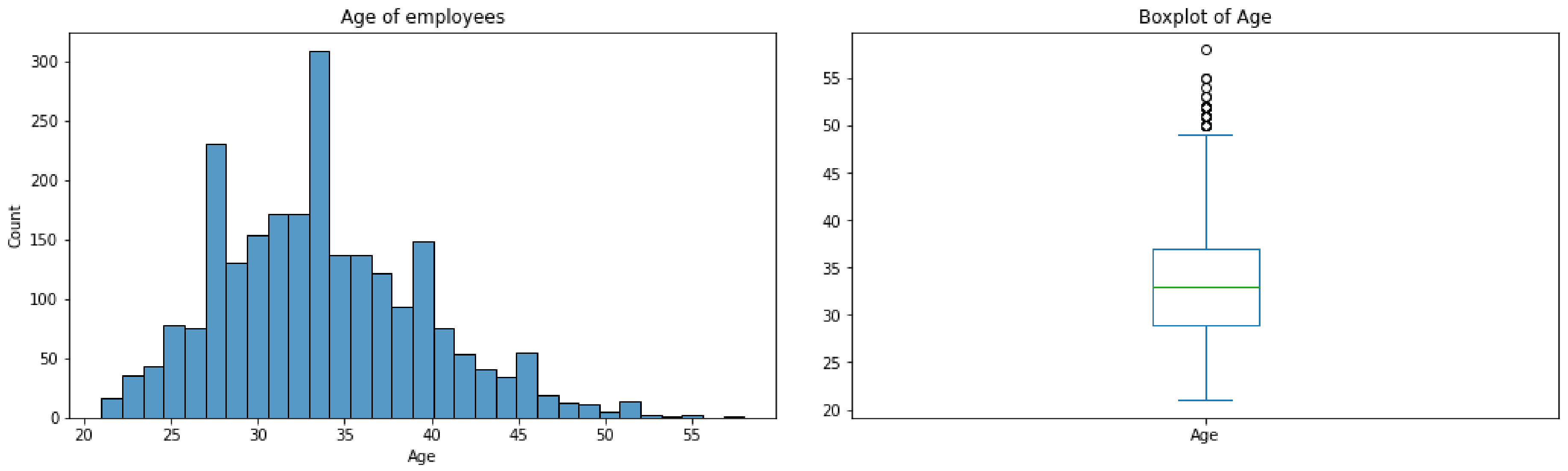
Observation 9 :

- Age : the maximum age is 58 and the minimum age is 21
- Income : Mean income - 59,334 and max income is 1,88,418 with 75% people making less than 75,986
- Total business value of 50% people are around 8,17,680

Contineous

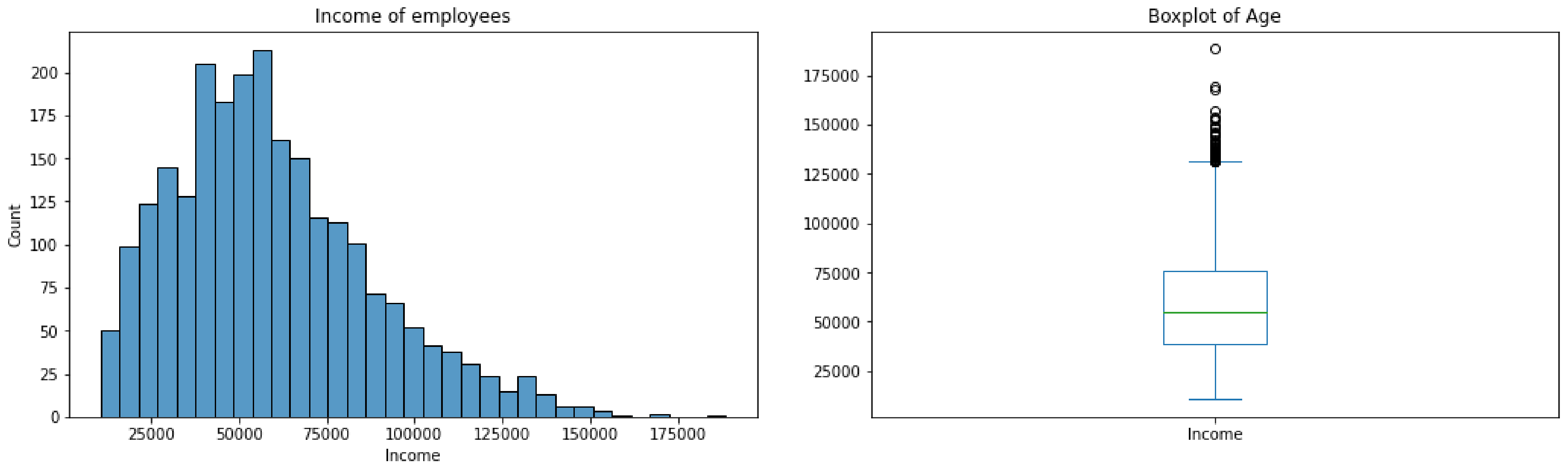
In [274]:
```python
continuous = ['Age', 'Income', 'Total_Business_Value']
```

In [264]:
```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.histplot(dff['Age'])
plt.title("Age of employees")
plt.subplot(122)
dff['Age'].plot.box(title='Boxplot of Age')
plt.tight_layout(pad=3)
```



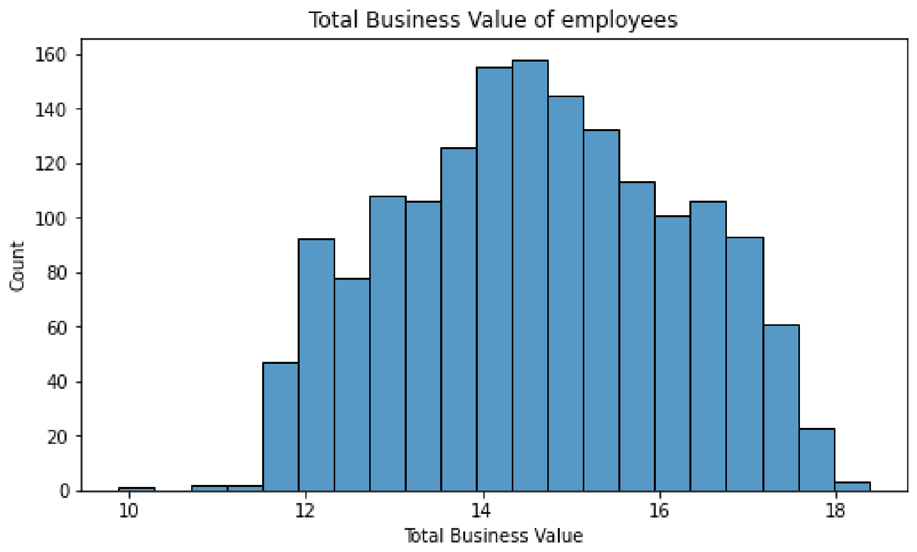
In [ ]:
```
 
```

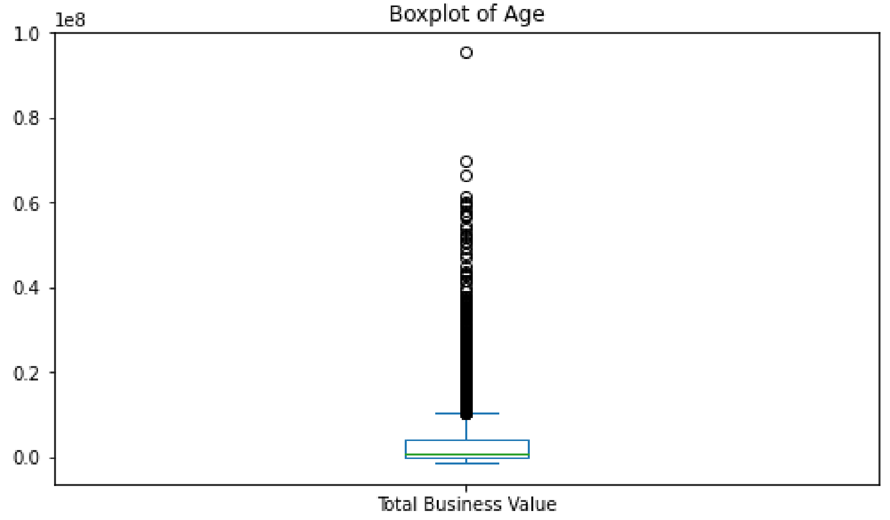
In [265]:
```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.histplot(dff['Income'])
plt.title("Income of employees")
plt.subplot(122)
dff['Income'].plot.box(title='Boxplot of Age')
plt.tight_layout(pad=3)
```



In [ ]:
```

```

In [271]:
```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.histplot(np.log(dff['Total Business Value']))
plt.title("Total Business Value of employees")
plt.subplot(122)
dff['Total Business Value'].plot.box(title='Boxplot of Age')
plt.tight_layout(pad=3)
```





Observation 10 :

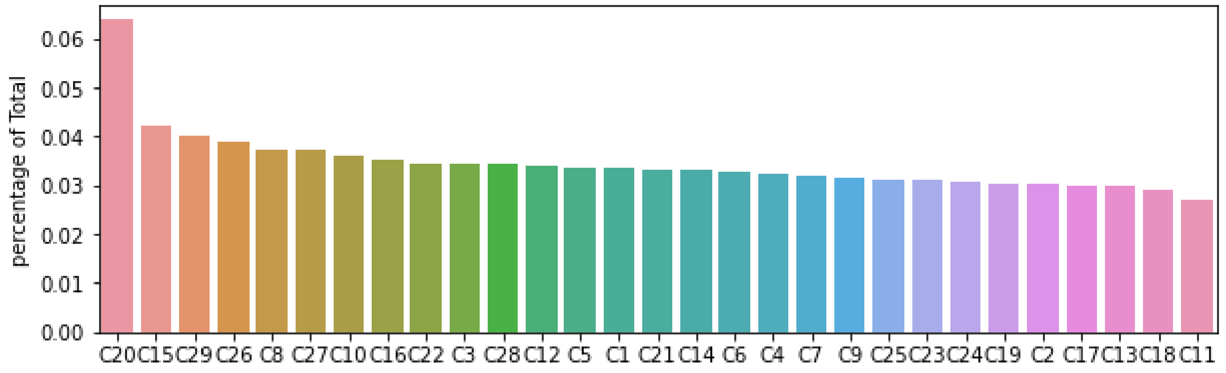
- The contineous columns like 'Age', 'Income', 'Total_Business_Value' has got outliers
- For the Age the distribution looks almost normal with some outliers
- For Income the distribution is not normal, its to the right and it has got some outliers as well.

Categorical Features

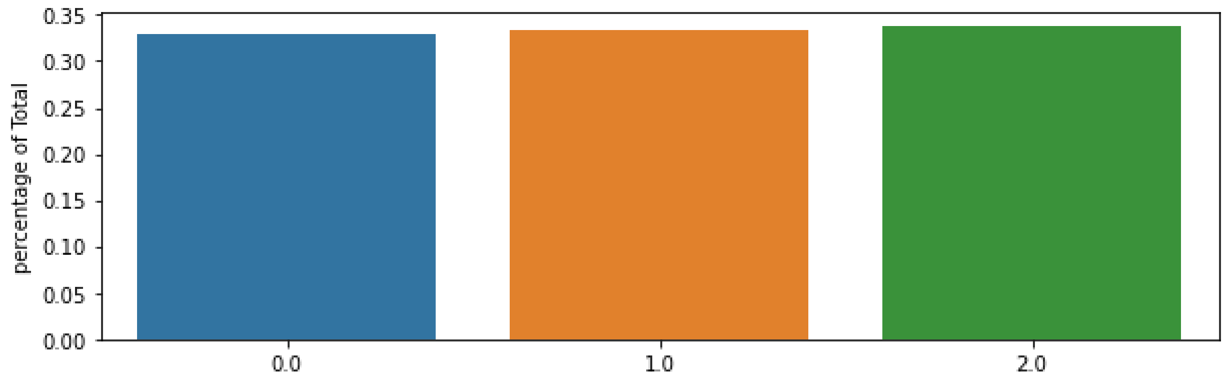
In [294]:
```
categorical = ['City', 'Education_Level', 'Gender', 'Grade', 'Joining Designation', 'Increased_Quarterly_Rating', 'T
```

In [296]:
```python
def barplot_columns(data, categorical):
    for col in categorical:
        print(f'Plotting the : {col}')
        print('_'*50)
        plt.figure(figsize=(10,3))
        x = dff[col].value_counts(normalize = True).index
        y = dff[col].value_counts(normalize = True).values
        sns.barplot(x=x, y=y)
        plt.xticks(rotation=0)
        plt.ylabel('percentage of Total')
        plt.show()
barplot_columns(df, categorical)
```

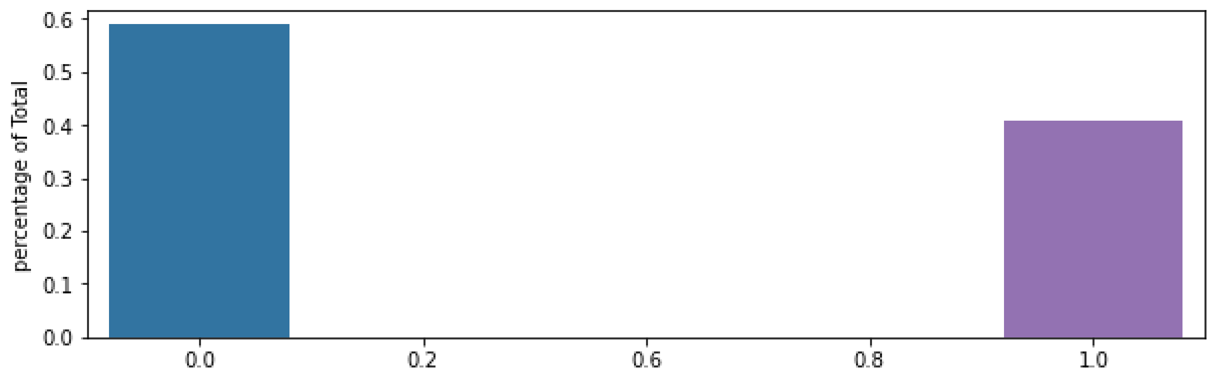
```
Plotting the : City
__________________________________________________
```



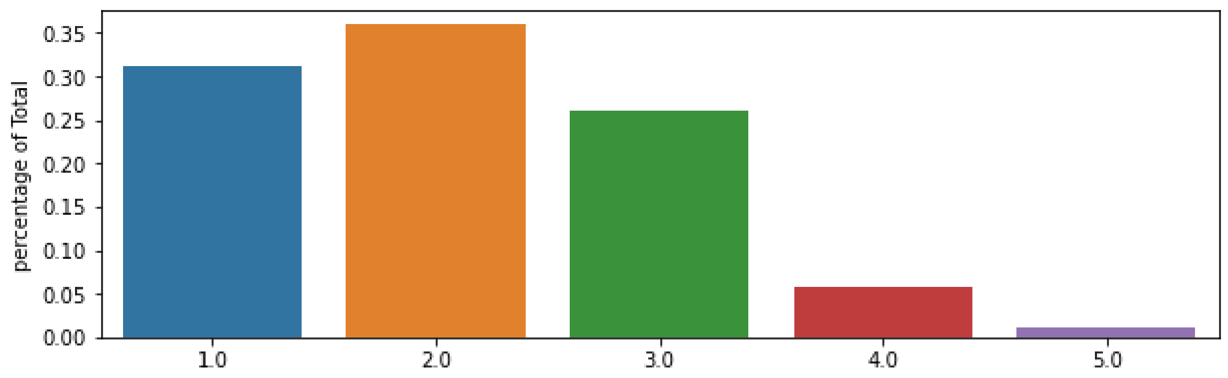
```
Plotting the : Education_Level
__________________________________________________
```



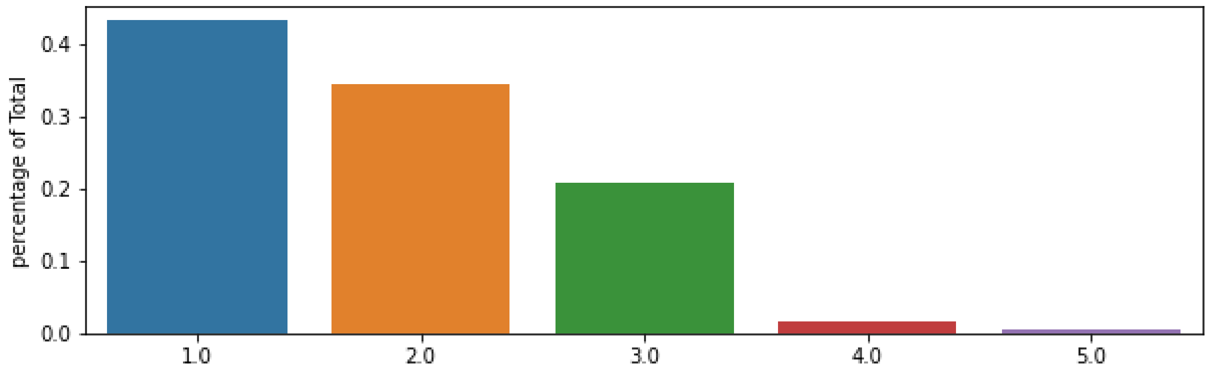
```
Plotting the : Gender
____________________________________________________
```



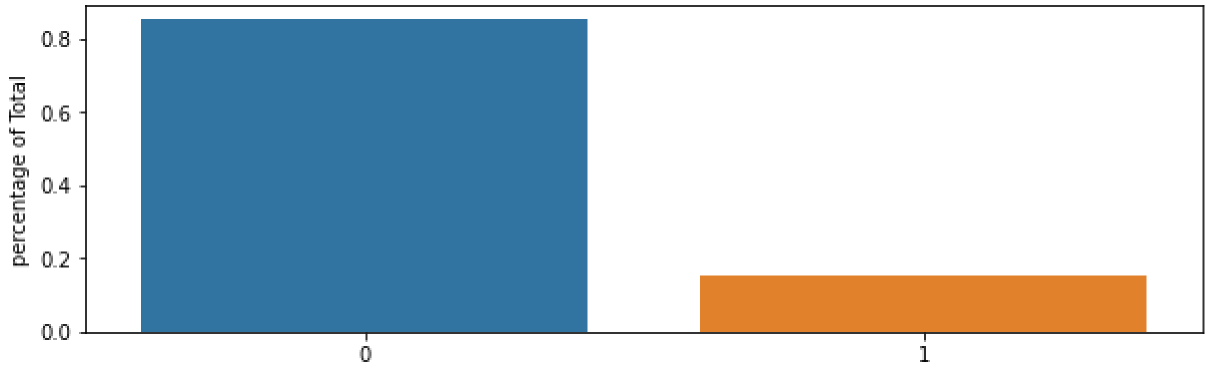
```
Plotting the : Grade
____________________________________________________
```



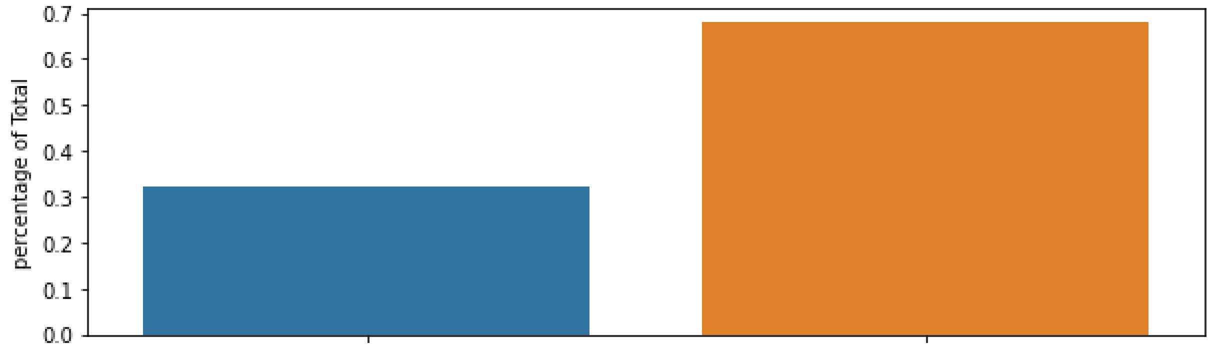
```
Plotting the : Joining Designation
_________________________________________________
```



```
Plotting the : Increased_Quarterly_Rating
_________________________________________________
```



```
Plotting the : Target
_________________________________________________
```



7. Check correlation among independent variables and how they interact with each other

In [327]:
```
dff
```

Out[327]:

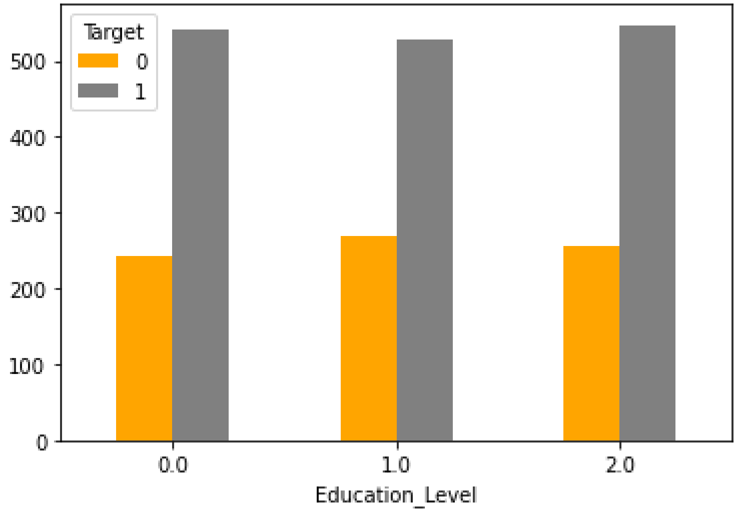
_ID	City	Education_Level	Age	Dateofjoining	Gender	Grade	Income	Joining Designation	MMM-YY	Total Business Value	Increased_Quarterly_Rating	Target	Increase
1	C23	2.0	28.0	2018-12-24	0.0	1.0	57387.0	1.0	2019-03-01	1715580.0	0	1	
2	C7	2.0	31.0	2020-11-06	0.0	2.0	67016.0	2.0	2020-12-01	0.0	0	0	
4	C13	2.0	43.0	2019-12-07	0.0	2.0	65603.0	2.0	2020-04-01	350000.0	0	1	
5	C9	0.0	29.0	2019-01-09	0.0	1.0	46368.0	1.0	2019-03-01	120360.0	0	1	
6	C11	1.0	31.0	2020-07-31	1.0	3.0	78728.0	3.0	2020-12-01	1265000.0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
784	C24	0.0	34.0	2015-10-15	0.0	3.0	82815.0	2.0	2020-12-01	21748820.0	1	0	
785	C9	0.0	34.0	2020-08-28	1.0	1.0	12105.0	1.0	2020-10-01	0.0	0	1	
786	C19	0.0	45.0	2018-07-31	0.0	2.0	35370.0	2.0	2019-09-01	2815090.0	0	1	
787	C20	2.0	28.0	2018-07-21	1.0	1.0	69498.0	1.0	2019-06-01	977830.0	0	1	
788	C27	2.0	30.0	2020-06-08	0.0	2.0	70254.0	2.0	2020-12-01	2298240.0	1	0	

14 columns

In [331]:
```
edu_level = pd.crosstab(dff['Education_Level'],dff['Target'])
print(edu_level)
edu_level.plot.bar(rot=0 , color = ['orange','grey'])
```

```
Target             0    1
Education_Level          
0.0              242  542
1.0              268  527
2.0              255  547
```

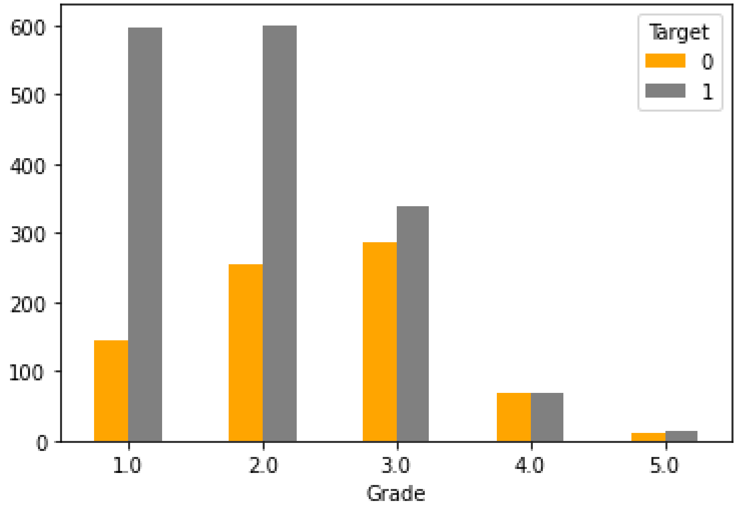
Out[331]:
```
<AxesSubplot:xlabel='Education_Level'>
```



In [333]:
```
grade = pd.crosstab(dff['Grade'],dff['Target'])
print(grade)
grade.plot.bar(rot=0 , color = ['orange','grey'])
```

```
Target    0    1
Grade
1.0     145  596
2.0     255  600
3.0     286  337
4.0      68   70
5.0      11   13
```

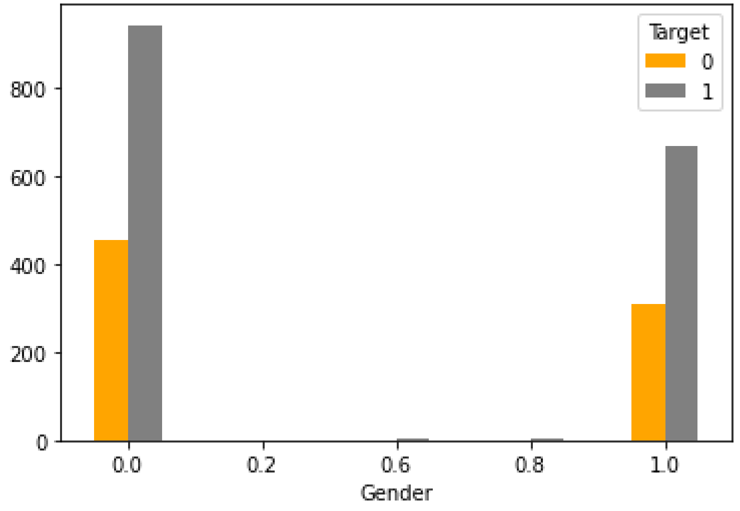
Out[333]:
```
<AxesSubplot:xlabel='Grade'>
```



In [326]:
```
gender = pd.crosstab(dff['Gender'],dff['Target'])
print(gender)
gender.plot.bar(rot=0 , color = ['orange','grey'])
```

```
Target    0    1
Gender
0.0     456  944
0.2       0    1
0.6       0    2
0.8       0    2
1.0     309  667
```

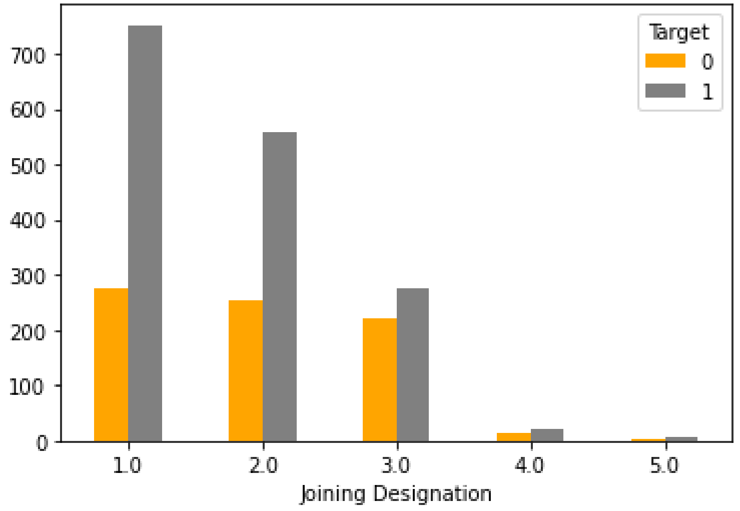
Out[326]:
```
<AxesSubplot:xlabel='Gender'>
```



In [336]:
```
JoiningD = pd.crosstab(dff['Joining Designation'],dff['Target'])
print(JoiningD)
JoiningD.plot.bar(rot=0 , color = ['orange','grey'])
```

```
Target                0    1
Joining Designation
1.0                 274  752
2.0                 255  560
3.0                 219  274
4.0                  14   22
5.0                   3    8
```

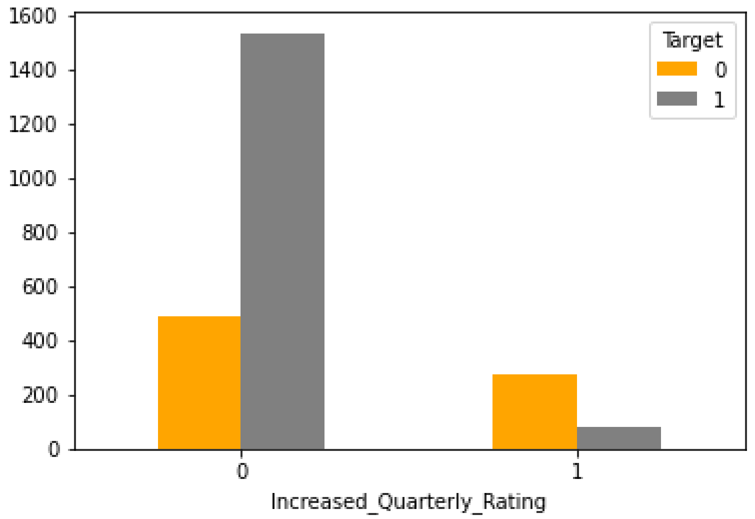
Out[336]:
```
<AxesSubplot:xlabel='Joining Designation'>
```



In [340]:
```python
Increased_Q = pd.crosstab(dff['Increased_Quarterly_Rating'],dff['Target'])
print(Increased_Q)
Increased_Q.plot.bar(rot=0 , color = ['orange','grey'])


```

```
Target                        0     1
Increased_Quarterly_Rating
0                           489  1534
1                           276    82
```

Out[340]:
```
<AxesSubplot:xlabel='Increased_Quarterly_Rating'>
```



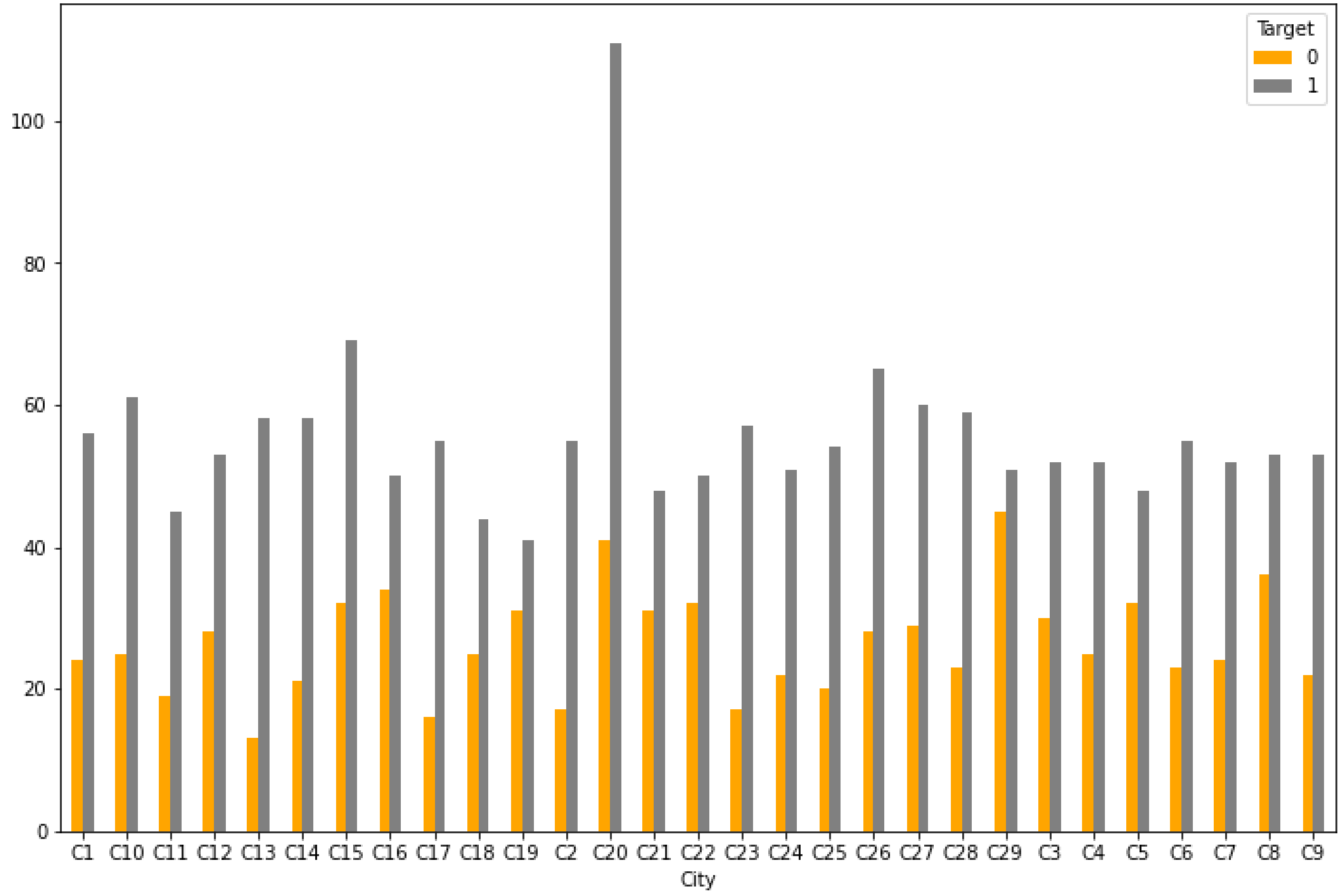
Observation 12 :

- Education Level : Education level of drivers is uniformly distributed. All three categories have almost equal number of drivers.
- 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.
- Grade : 5 unique Grades, Grade 2 has highest & Grade 5 has lowest number of drivers
- Joining Designation :5 unique categories present. JD-1 has highest count & JD-5 has lowest count.
- The employees who have their grade as 3 or 4 at the time of joiningDesignation are less likely to leave the organization.

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

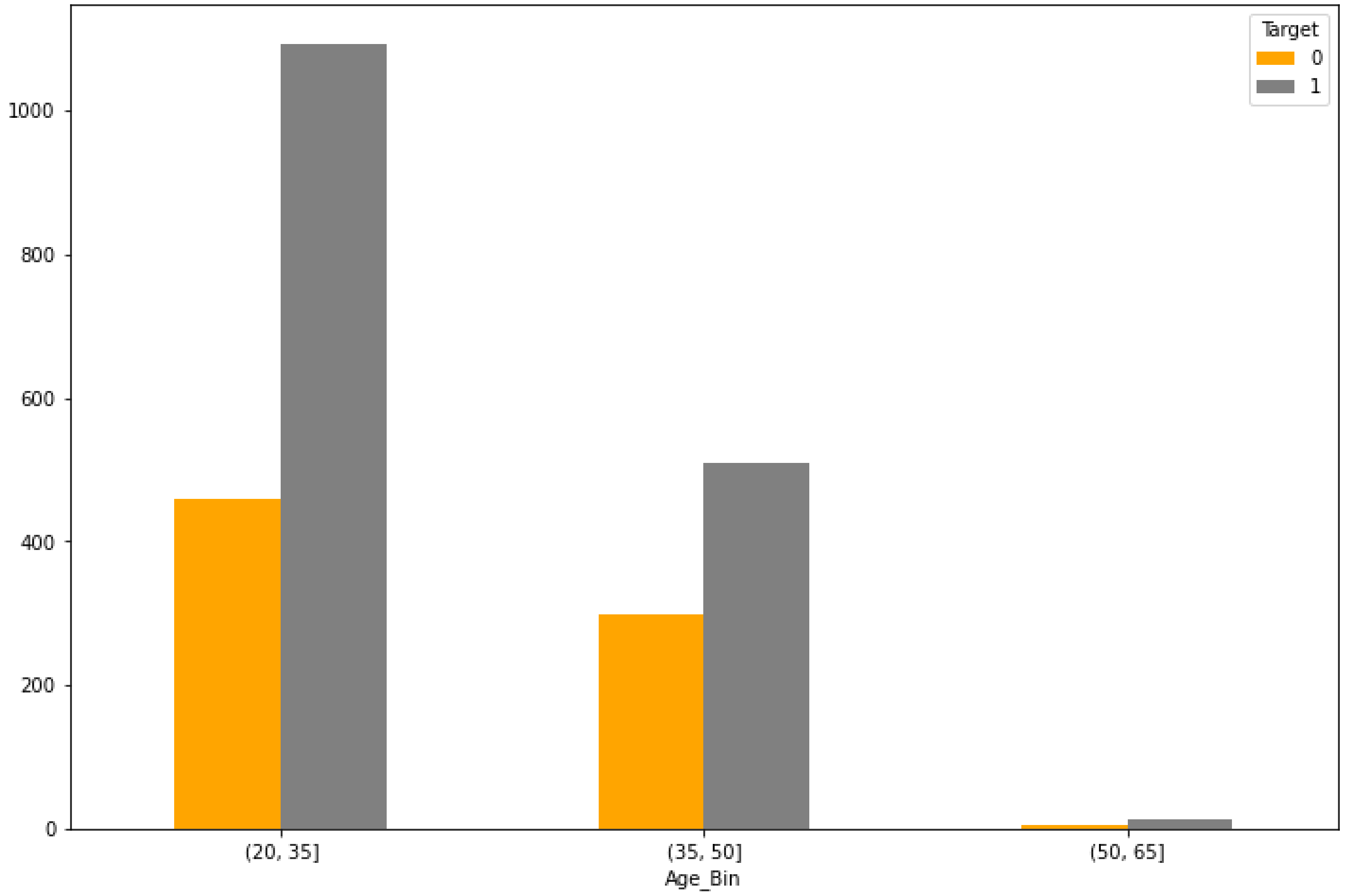
In [382]:
```
plt.rcParams["figure.figsize"] = (12, 8)
city = pd.crosstab(dff['City'],dff['Target'])
city.plot.bar(rot=0 , color = ['orange','grey'])

```

Out[382]:
```
<AxesSubplot:xlabel='City'>
```



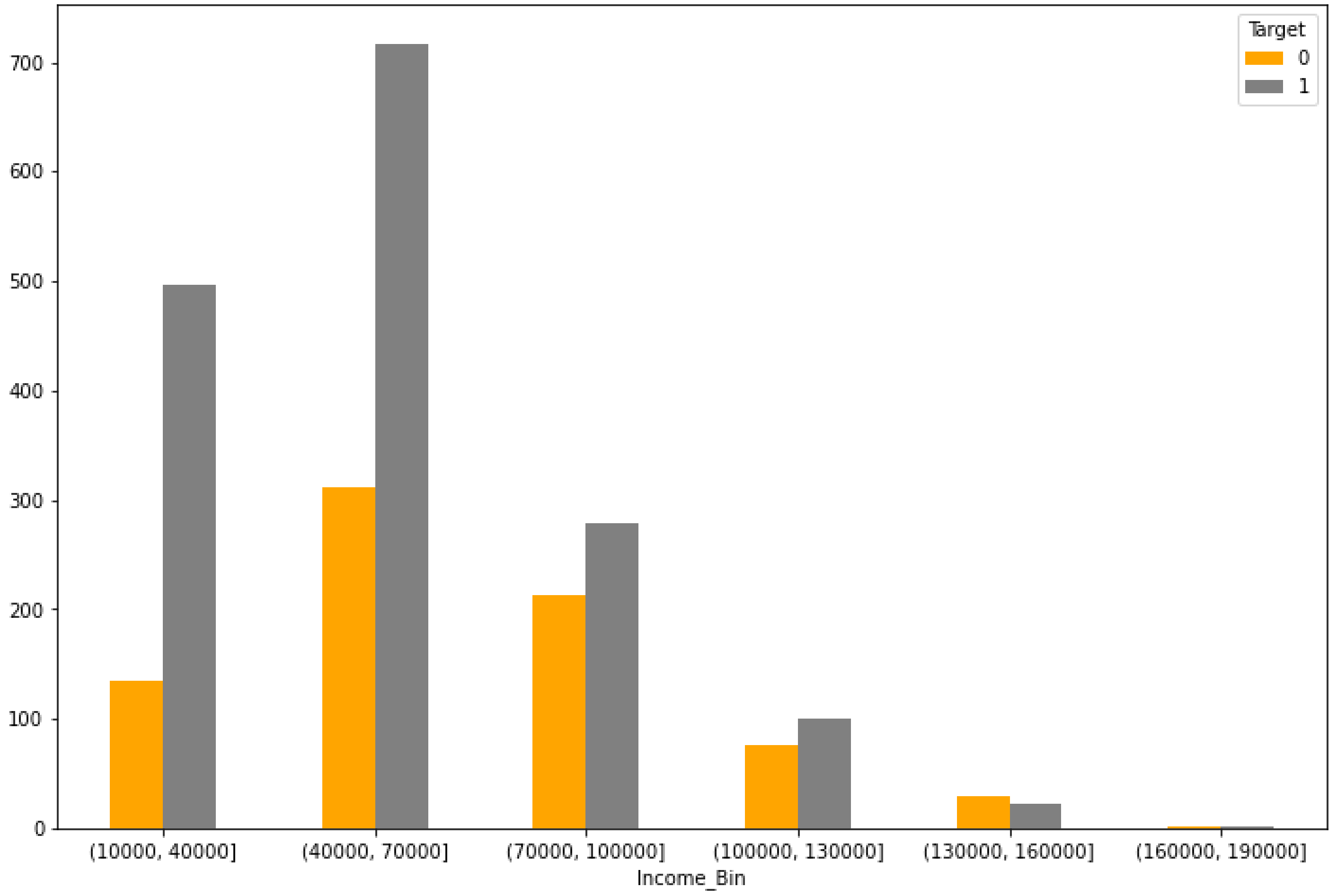
In [384]:
```
plt.rcParams["figure.figsize"] = (12, 8)
dff['Age_Bin'] = pd.cut(dff['Age'],bins=[20,35,50,65])
city = pd.crosstab(dff['Age_Bin'],dff['Target'])
city.plot.bar(rot=0 , color = ['orange','grey'])
```

Out[384]:
```
<AxesSubplot:xlabel='Age_Bin'>
```

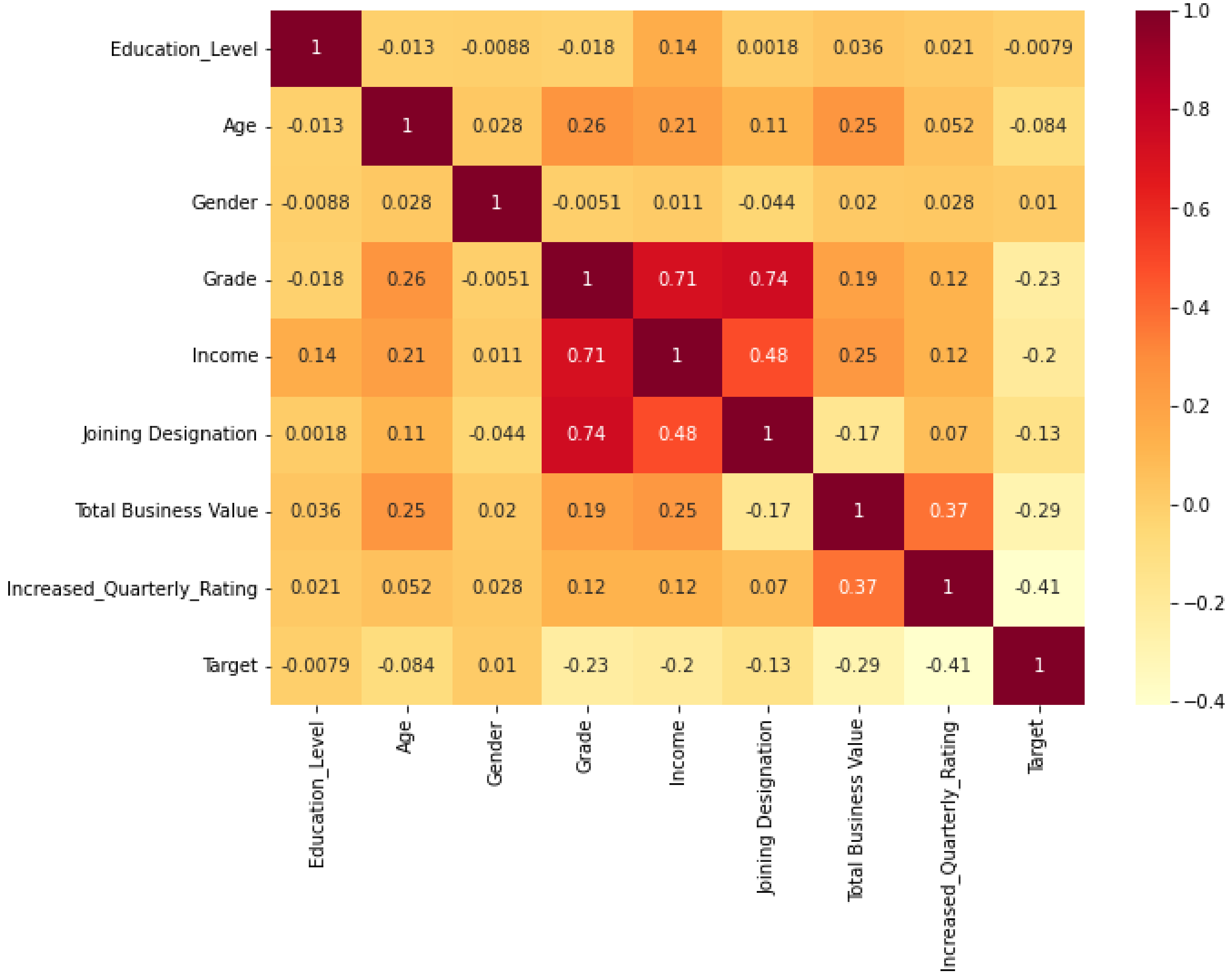


In [386]:
```
plt.rcParams["figure.figsize"] = (12, 8)
dff['Income_Bin'] = pd.cut(dff['Income'],bins=[10000, 40000, 70000, 100000, 130000, 160000, 190000 ])
city = pd.crosstab(dff['Income_Bin'],dff['Target'])
city.plot.bar(rot=0 , color = ['orange','grey'])
```

Out[386]:
```
<AxesSubplot:xlabel='Income_Bin'>
```



In [396]:
```
matrix_col = ['Education_Level', 'Age', 'Gender', 'Grade', 'Income', 'Joining Designation', 'Total Business Value',
'Increased_Quarterly_Rating','Target']
corr = dff[matrix_col].corr(method='spearman')
plt.figure(figsize=(10,7))
sns.heatmap(corr, annot = True,  cmap = 'YlOrRd')
plt.show()
```



Observation 12 :

- For the Age bins we can see that - lesser age with less income has more chance of leaving
- for the age bin of same lesss with large income has a more probability to stay
- There is a great correlation between Income and Grade ,indicating drivers with higher grades have higher monthly income.
- There is a significant correlation between Joining Designation and Grade, This indicates as the Grade of Driver increases, joining designation also increases.

In [390]:
```
dff.drop(['Age_Bin','Income_Bin'],axis=1,inplace=True)
```

In [391]:
```
dff.head()
```

Out[391]:

	Driver_ID	City	Education_Level	Age	Dateofjoining	Gender	Grade	Income	Joining Designation	MMM-YY	Total Business Value	Increased_Quarterly_Rating	Target
0	1	C23	2.0	28.0	2018-12-24	0.0	1.0	57387.0	1.0	2019-03-01	1715580.0	0	1
1	2	C7	2.0	31.0	2020-11-06	0.0	2.0	67016.0	2.0	2020-12-01	0.0	0	0
2	4	C13	2.0	43.0	2019-12-07	0.0	2.0	65603.0	2.0	2020-04-01	350000.0	0	1
3	5	C9	0.0	29.0	2019-01-09	0.0	1.0	46368.0	1.0	2019-03-01	120360.0	0	1
4	6	C11	1.0	31.0	2020-07-31	1.0	3.0	78728.0	3.0	2020-12-01	1265000.0	1	0

8 . One hot encoding of the categorical variable

In [397]:
```
dff.select_dtypes(include=['object']).columns.tolist()
```

Out[397]:
```
['City']
```

Changed name back to df

In [400]:
```
df = pd.get_dummies(dff, columns = ['City'])
```

In [ ]:
```
df.columns
```

In [418]:
```
df.drop(['MMM-YY'],axis=1,inplace=True)
```

9. Standardization of training data

- we only want to do scaling on numerical columns and thus we exclude the colums - City , Driver ID , Target and DateOfJoining

In [405]:
```
X = df.drop(['Dateofjoining'],axis=1,inplace=True)
```

In [415]:
```
X = df
```

In [419]:
```
Xcols=X.columns
Xcols
```

Out[419]:
```
Index(['Education_Level', 'Age', 'Gender', 'Grade', 'Income',
       'Joining Designation', 'Total Business Value',
       'Increased_Quarterly_Rating', 'Increased_Income', 'City_C1', 'City_C10',
       'City_C11', 'City_C12', 'City_C13', 'City_C14', 'City_C15', 'City_C16',
       'City_C17', 'City_C18', 'City_C19', 'City_C2', 'City_C20', 'City_C21',
       'City_C22', 'City_C23', 'City_C24', 'City_C25', 'City_C26', 'City_C27',
       'City_C28', 'City_C29', 'City_C3', 'City_C4', 'City_C5', 'City_C6',
       'City_C7', 'City_C8', 'City_C9'],
      dtype='object')
```

In [420]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [421]:
```
X = scaler.fit_transform(X)
```

In [426]:
```
X = pd.DataFrame(X)
X.columns = Xcols
X
```

Out[426]:

	Education_Level	Age	Gender	Grade	Income	Joining Designation	Total Business Value	Increased_Quarterly_Rating	Increased_Income	City_C1	...	City_
0	1.0	0.189189	0.0	0.00	0.262508	0.00	0.032064	0.0	0.0	0.0	...	
1	1.0	0.270270	0.0	0.25	0.316703	0.25	0.014326	0.0	0.0	0.0	...	
2	1.0	0.594595	0.0	0.25	0.308750	0.25	0.017944	0.0	0.0	0.0	...	
3	0.0	0.216216	0.0	0.00	0.200489	0.00	0.015570	0.0	0.0	0.0	...	
4	0.5	0.270270	1.0	0.50	0.382623	0.50	0.027405	1.0	0.0	0.0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
2376	0.0	0.351351	0.0	0.50	0.405626	0.25	0.239197	1.0	0.0	0.0	...	
2377	0.0	0.351351	1.0	0.00	0.007643	0.00	0.014326	0.0	0.0	0.0	...	
2378	0.0	0.648649	0.0	0.25	0.138588	0.25	0.043432	0.0	0.0	0.0	...	
2379	1.0	0.189189	1.0	0.00	0.330673	0.00	0.024436	0.0	0.0	0.0	...	
2380	1.0	0.243243	0.0	0.25	0.334928	0.25	0.038088	1.0	0.0	0.0	...	

2381 rows × 38 columns

10. Model building

In [427]:
```
from sklearn.model_selection import train_test_split,GridSearchCV
```

In [430]:
```
y = X['Target']
X = X.drop(['Target'], axis = 1)
X_train , X_test , y_train, y_test = train_test_split(X , y, test_size=0.2 , random_state= 23)
```

In [431]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[431]:
```
((1904, 38), (477, 38), (1904,), (477,))
```

BAGGING

Random Forest with class weights

since Random forest can work with Imbalanced data we are trying this out now

In [433]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils import class_weight
```

In [447]:
```
rfmodel = RandomForestClassifier(class_weight = 'balanced')
```

In [450]:
```
param_grid = {'max_depth':np.arange(2,20,5), 'n_estimators':[5,10,50,100,500,1000] , 'criterion' : ['gini', 'entropy

gs_rfmodel = GridSearchCV(rfmodel , param_grid , cv = 3 , scoring = 'roc_auc')
gs_rfmodel.fit(X_train,y_train)
```

Out[450]:
```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [451]:
```
print(f'We can get roc_auc score of {np.round(gs_rfmodel.best_score_, 4)} using {gs_rfmodel.best_params_}')
```

```
We can get roc_auc score of 0.8145 using {'criterion': 'entropy', 'max_depth': 12, 'n_estimators': 1000}
```

In [ ]:
```python

```

In [439]:
```python
rfmodel = RandomForestClassifier(class_weight = 'balanced_subsample')
```

In [440]:
```python
param_grid = {'max_depth':np.arange(2,20,5), 'n_estimators':[5,10,50,100,500,1000] , 'criterion' : ['gini', 'entropy

gs_rfmodel = GridSearchCV(rfmodel , param_grid , cv = 3 , scoring = 'roc_auc')
gs_rfmodel.fit(X_train,y_train)
```

Out[440]:
```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [441]:
```python
print(f'We can get roc_auc score of {np.round(gs_rfmodel.best_score_, 4)} using {gs_rfmodel.best_params_}')
```

```
We can get roc_auc score of 0.8151 using {'criterion': 'entropy', 'max_depth': 7, 'n_estimators': 50}
```

In [ ]:
```python

```

11. Class Imbalance Treatment

In [443]:

```python
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 = 23)
X_train_sm, y_train_sm = sm.fit_resample(X_train, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train_sm.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_sm.shape))

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

```
Before OverSampling, counts of label '1': 1278
Before OverSampling, counts of label '0': 626

After OverSampling, the shape of train_X: (2556, 38)
After OverSampling, the shape of train_y: (2556,)

After OverSampling, counts of label '1': 1278
After OverSampling, counts of label '0': 1278
```

In [ ]:

```python

```

In [444]:

```python
rfmodel = RandomForestClassifier(class_weight = 'balanced')
```

In [445]:
```
param_grid = {'max_depth':np.arange(2,20,5), 'n_estimators':[5,10,50,100,500,1000] , 'criterion' : ['gini', 'entropy

gs_rfmodel = GridSearchCV(rfmodel , param_grid , cv = 3 , scoring = 'roc_auc')
gs_rfmodel.fit(X_train_sm,y_train_sm)
```

Out[445]:
```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [446]:
```
print(f'We can get roc_auc score of {np.round(gs_rfmodel.best_score_, 4)} using {gs_rfmodel.best_params_}')
```

```
We can get roc_auc score of 0.895 using {'criterion': 'gini', 'max_depth': 17, 'n_estimators': 1000}
```

In [ ]:
```

```

Random Forest with best Hyperparameters

In [461]:
```
rfc = RandomForestClassifier(bootstrap=True,
                                criterion = 'gini',
                                max_depth=17,
                                n_estimators=1000)
```

In [462]:
```
rfc.fit(X_train_sm, y_train_sm)
```

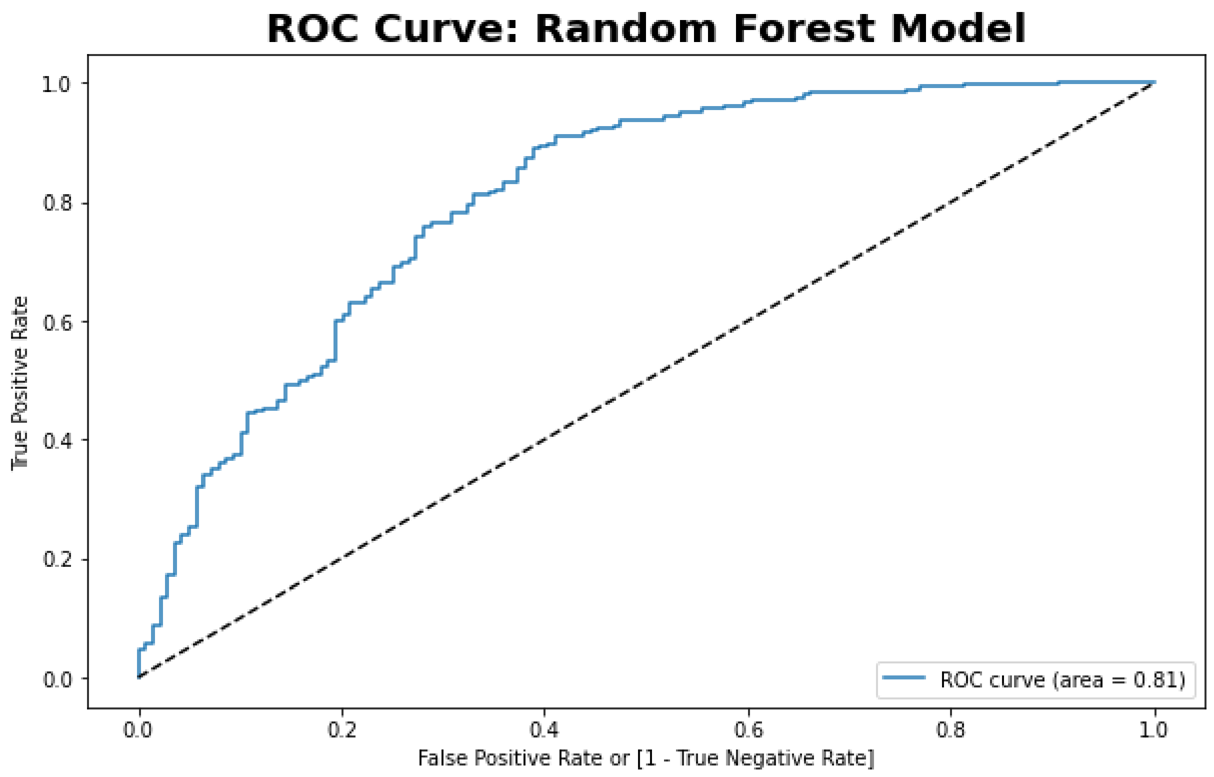
Out[462]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=17, n_estimators=1000)
```

In [463]:
```
y_pred_rf = rfc.predict(X_test)
```

In [464]:

```
#ROC_AUC Curve
from sklearn import metrics
y_pred_rf_proba = rfc.predict_proba(X_test)
fpr, tpr, thr = metrics.roc_curve(y_test , y_pred_rf_proba[:,1])
auc_score = metrics.roc_auc_score( y_test, y_pred_rf_proba[:,1] )
```

In [465]:
```
plt.figure(figsize=(10,6))
plt.plot(fpr,tpr, label='ROC curve (area = %0.2f)' % auc_score )
plt.plot([0, 1], [0, 1], 'k--')
plt.title('ROC Curve: Random Forest Model', fontsize = 20, fontweight = 'bold')
plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()
print('-'*70)
print(f'ROC_AUC score = \t {np.round(auc_score,5)}')
print('-'*70)
```



```
----------------------------------------------------------------------
ROC_AUC score = 	 0.80639
----------------------------------------------------------------------
```

In [466]:
```python
#Important Metrics of Model
print('-'*70)
print('Important Metrics of Random Forest Model')
print('-'*70)
print(f'ROC_AUC score = \t {np.round(auc_score,5)}')
print(f'Accuracy of Model : \t {np.round(metrics.accuracy_score(y_test,y_pred_rf),5)}')
print(f'f1_score of Model : \t {np.round(metrics.f1_score(y_test,y_pred_rf),5)}')
print(f'Precision of Model : \t {np.round(metrics.precision_score(y_test,y_pred_rf),5)}')
print(f'Recall of Model : \t {np.round(metrics.recall_score(y_test,y_pred_rf),5)}')
print('-'*70)
```

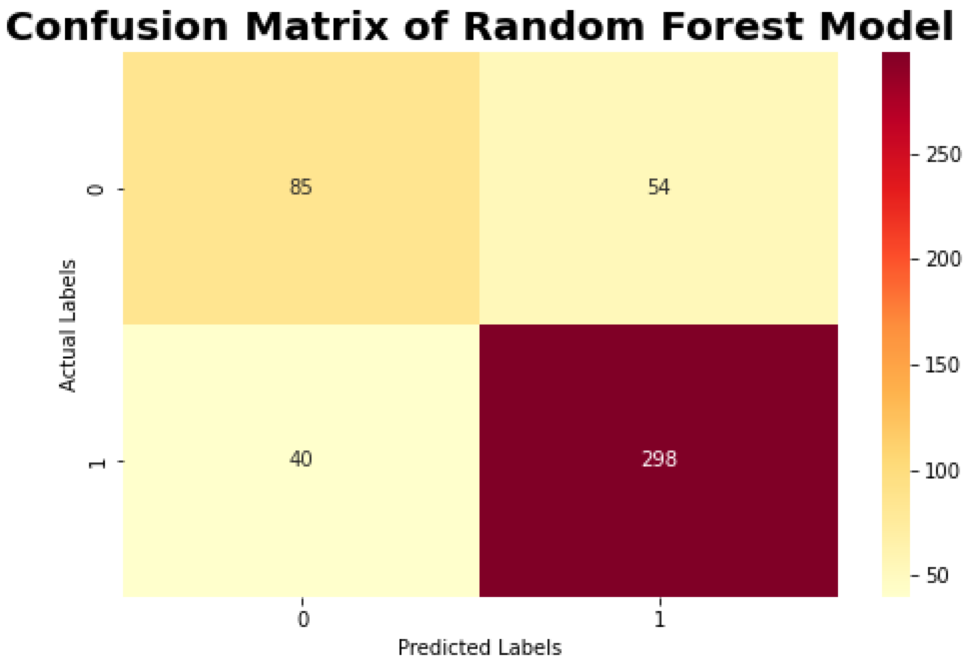
```
----------------------------------------------------------------------
Important Metrics of Random Forest Model
----------------------------------------------------------------------
ROC_AUC score =          0.80639
Accuracy of Model :      0.80294
f1_score of Model :      0.86377
Precision of Model :     0.84659
Recall of Model :        0.88166
----------------------------------------------------------------------
```

In [467]:

```python
# Classification Report
from sklearn.metrics import classification_report,confusion_matrix
print('-'*70)
print('Classification Report: Random Forest')
print('-'*70)
print(classification_report(y_test,y_pred_rf))
print('-'*70)
```

```
----------------------------------------------------------------------
Classification Report: Random Forest
----------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.68      0.61      0.64       139
           1       0.85      0.88      0.86       338

    accuracy                           0.80       477
   macro avg       0.76      0.75      0.75       477
weighted avg       0.80      0.80      0.80       477

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

In [468]:
```python
conf_matrix_rf = confusion_matrix(y_test,y_pred_rf)
plt.figure(figsize=(8,5))
sns.heatmap(conf_matrix_rf, annot = True, cmap = 'YlOrRd', fmt="1.0f")
plt.title('Confusion Matrix of Random Forest Model',fontsize = 20, fontweight = 'bold' )
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()
```



In [471]:

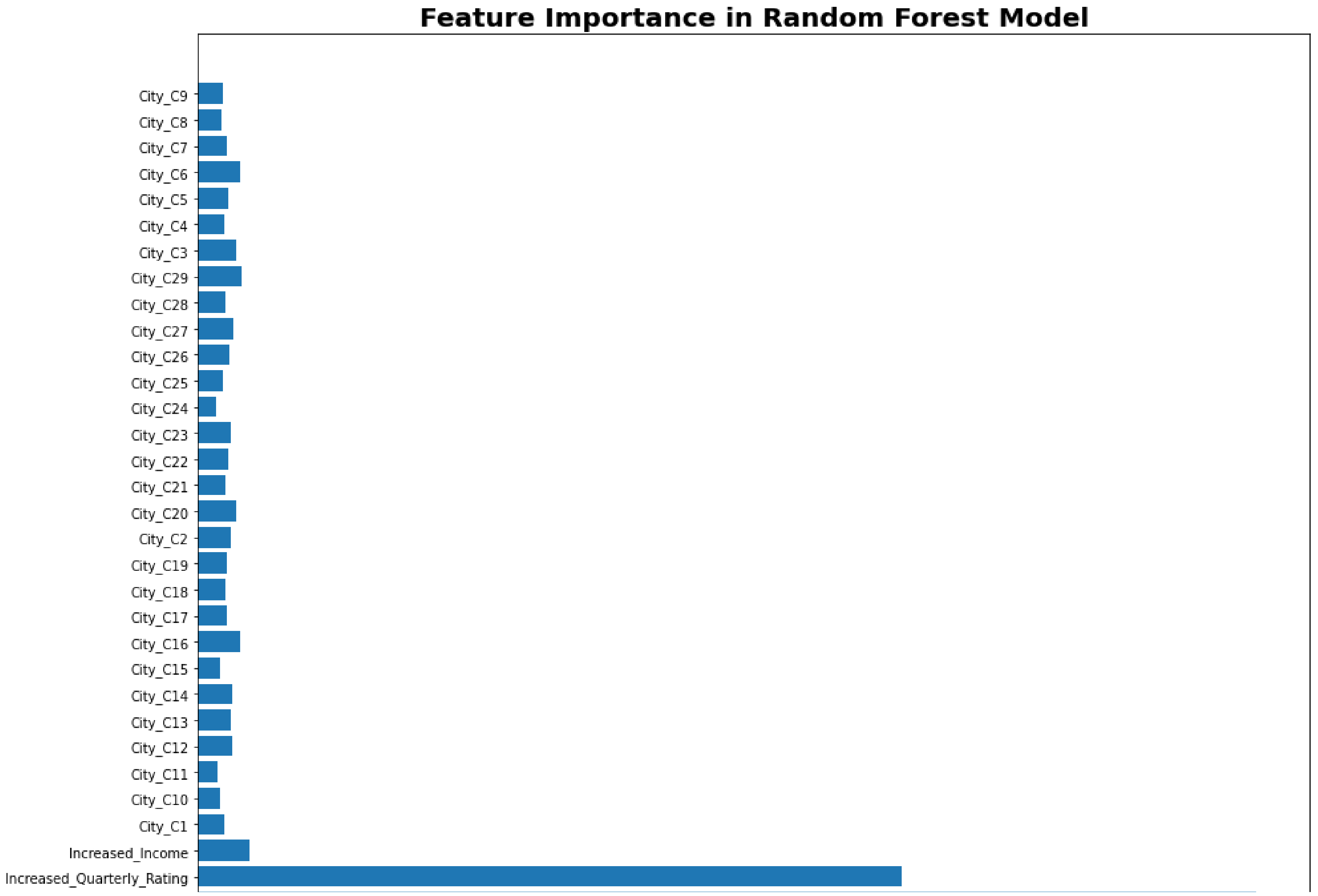
```
df
```

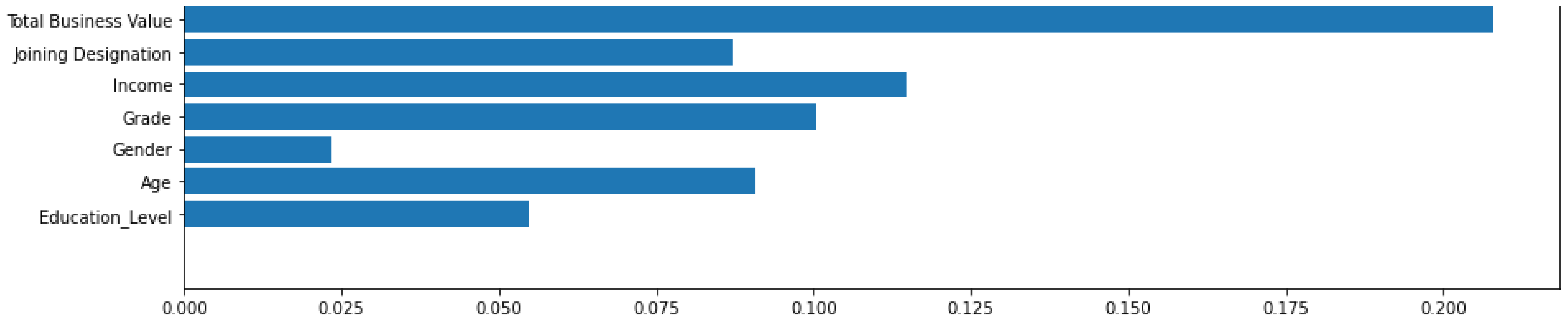
Out[471]:

	Education_Level	Age	Gender	Grade	Income	Joining Designation	Total Business Value	Increased_Quarterly_Rating	Increased_Income	City_C1	...	City_C27
0	2.0	28.0	0.0	1.0	57387.0	1.0	1715580.0	0	0	0	...	0
1	2.0	31.0	0.0	2.0	67016.0	2.0	0.0	0	0	0	...	0
2	2.0	43.0	0.0	2.0	65603.0	2.0	350000.0	0	0	0	...	0
3	0.0	29.0	0.0	1.0	46368.0	1.0	120360.0	0	0	0	...	0
4	1.0	31.0	1.0	3.0	78728.0	3.0	1265000.0	1	0	0	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
2376	0.0	34.0	0.0	3.0	82815.0	2.0	21748820.0	1	0	0	...	0
2377	0.0	34.0	1.0	1.0	12105.0	1.0	0.0	0	0	0	...	0
2378	0.0	45.0	0.0	2.0	35370.0	2.0	2815090.0	0	0	0	...	0
2379	2.0	28.0	1.0	1.0	69498.0	1.0	977830.0	0	0	0	...	0
2380	2.0	30.0	0.0	2.0	70254.0	2.0	2298240.0	1	0	0	...	1

2381 rows × 38 columns

In [541]:
```
plt.figure(figsize=(15,15))
plt.barh(df.columns[:], rfc.feature_importances_)
plt.title('Feature Importance in Random Forest Model',fontsize = 20, fontweight = 'bold' )
plt.show()
```





In [ ]:

```python

```

BOOSTING

Model-02 LightGBM BOOSTING : Hyperparameter tuning using GridSearch_CV

In [491]:

```python
# !pip install lightgbm
```

In [494]:

```python
import lightgbm as lgb
from lightgbm import LGBMClassifier
```

In [495]:

```python
lgbmodel = LGBMClassifier()
```

In [497]:
```
param_grid_lgb = {
    'lgb__learning_rate': [0.005,0.01,0.05,0.1],
    'lgb__n_estimators': [40,60,80,100],
    'lgb__num_leaves': np.arange(10,20,2),
    'lgb__num_iterations': [ 100, 500, 1000, 2000] }


gs_lgbmodel = GridSearchCV(lgbmodel , param_grid_lgb , cv = 3 , n_jobs=-1, verbose=1, scoring = 'roc_auc')
gs_lgbmodel.fit(X_train_sm,y_train_sm)
```

```
Fitting 3 folds for each of 320 candidates, totalling 960 fits
[LightGBM] [Warning] Unknown parameter: lgb__num_iterations
[LightGBM] [Warning] Unknown parameter: lgb__learning_rate
[LightGBM] [Warning] Unknown parameter: lgb__n_estimators
[LightGBM] [Warning] Unknown parameter: lgb__num_leaves
```

Out[497]:
```
▸ GridSearchCV
▸ estimator: LGBMClassifier
    ▸ LGBMClassifier
```

In [498]:
```
print(f'We can get roc_auc score of {np.round(gs_lgbmodel.best_score_, 4)} using {gs_lgbmodel.best_params_}')
```

```
We can get roc_auc score of 0.8744 using {'lgb__learning_rate': 0.005, 'lgb__n_estimators': 40, 'lgb__num_iterations':
100, 'lgb__num_leaves': 10}
```

In [ ]:
```

```

Model with the best hyperparameters

In [499]:
```
lgb = lgb.LGBMClassifier(learning_rate = 0.005 ,n_estimators = 40, num_iterations=100 , num_leaves = 10)
```

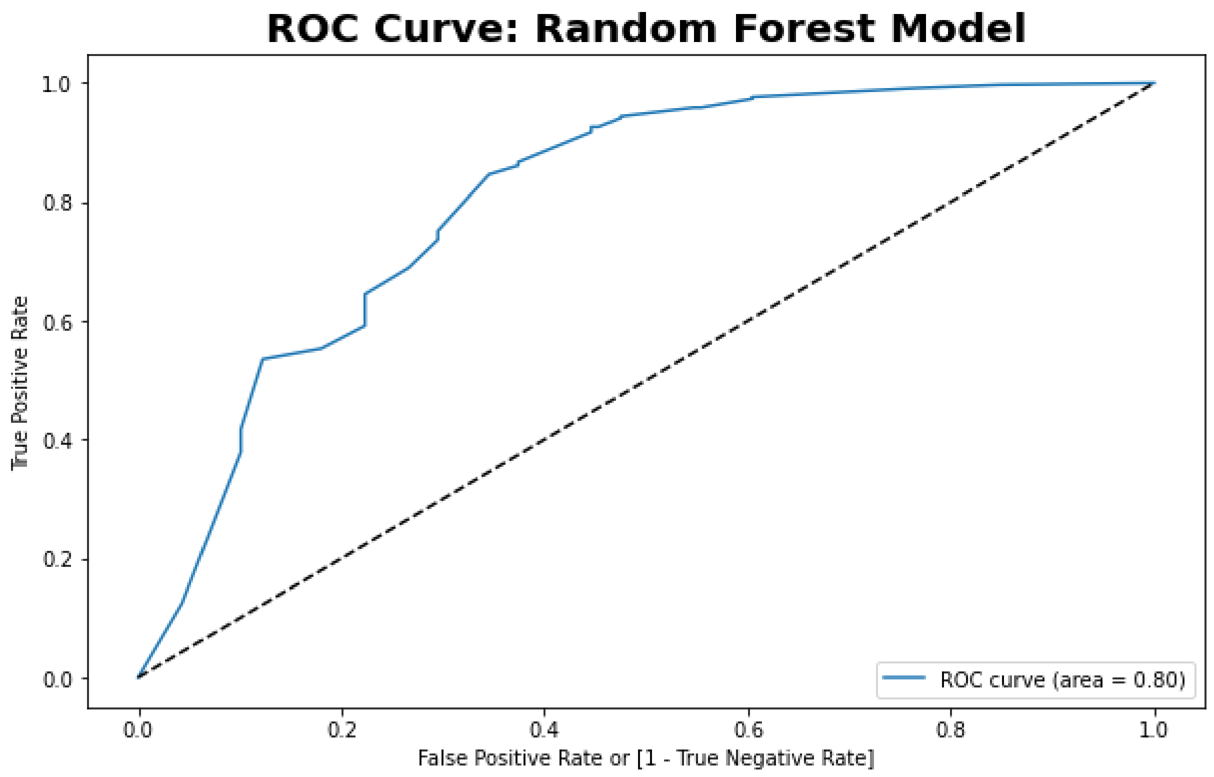
In [501]:
```
lgb.fit(X_train_sm, y_train_sm)
```

Out[501]:
```
▾                              LGBMClassifier
LGBMClassifier(learning_rate=0.005, n_estimators=40, num_iterations=100,
               num_leaves=10)
```

In [502]:
```
y_pred_lgb = lgb.predict(X_test)
```

In [503]:
```
#ROC_AUC Curve
from sklearn import metrics
y_pred_lgb_proba = lgb.predict_proba(X_test)
fpr, tpr, thr = metrics.roc_curve(y_test , y_pred_lgb_proba[:,1])
auc_score = metrics.roc_auc_score( y_test, y_pred_lgb_proba[:,1] )
```

In [504]:
```python
plt.figure(figsize=(10,6))
plt.plot(fpr,tpr, label='ROC curve (area = %0.2f)' % auc_score )
plt.plot([0, 1], [0, 1], 'k--')
plt.title('ROC Curve: Random Forest Model', fontsize = 20, fontweight = 'bold')
plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()
print('-'*70)
print(f'ROC_AUC score = \t {np.round(auc_score,5)}')
print('-'*70)
```



```
----------------------------------------------------------------------
ROC_AUC score =          0.80491
----------------------------------------------------------------------
```

In [506]:

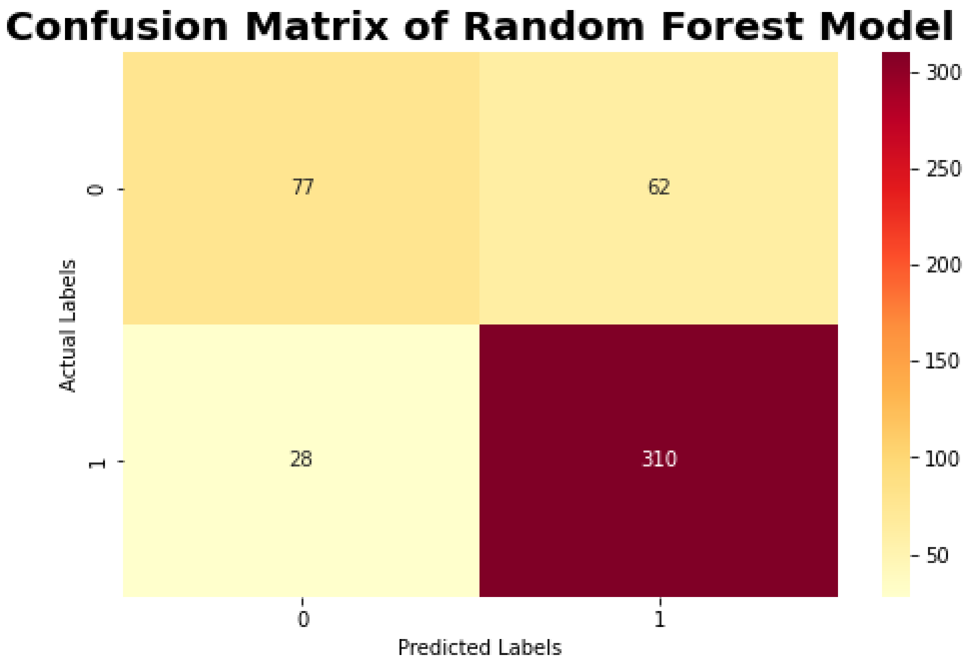
```python
#Important Metrics of Model
print('-'*70)
print('Important Metrics of LightGBM Model')
print('-'*70)
print(f'ROC_AUC score = \t {np.round(auc_score,5)}')
print(f'Accuracy of Model : \t {np.round(metrics.accuracy_score(y_test,y_pred_lgb),5)}')
print(f'f1_score of Model : \t {np.round(metrics.f1_score(y_test,y_pred_lgb),5)}')
print(f'Precision of Model : \t {np.round(metrics.precision_score(y_test,y_pred_lgb),5)}')
print(f'Recall of Model : \t {np.round(metrics.recall_score(y_test,y_pred_lgb),5)}')
print('-'*70)
```

```
----------------------------------------------------------------------
Important Metrics of LightGBM Model
----------------------------------------------------------------------
ROC_AUC score =          0.80491
Accuracy of Model :      0.81132
f1_score of Model :      0.87324
Precision of Model :     0.83333
Recall of Model :        0.91716
----------------------------------------------------------------------
```

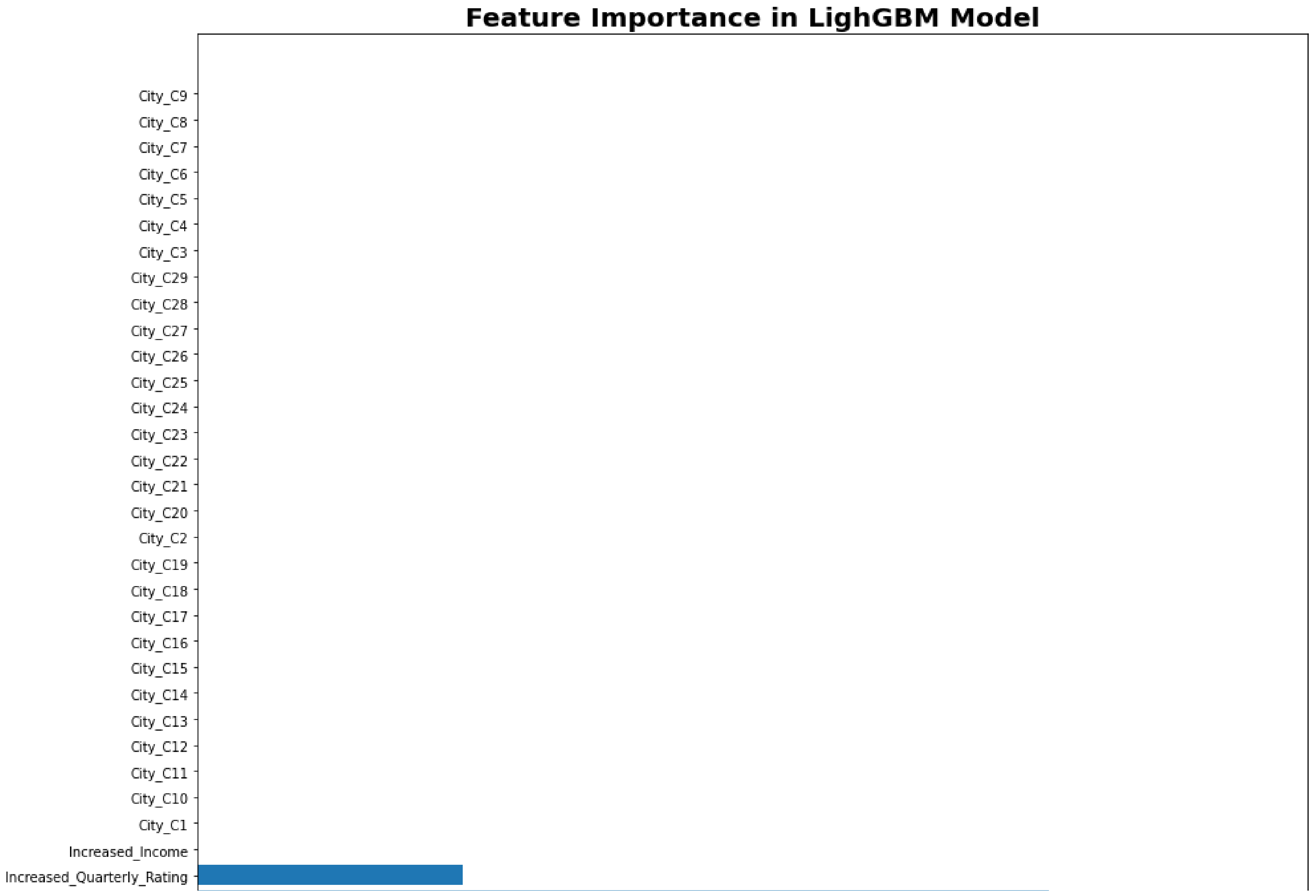
In [507]:
```python
# Classification Report
from sklearn.metrics import classification_report,confusion_matrix
print('-'*70)
print('Classification Report: Random Forest')
print('-'*70)
print(classification_report(y_test,y_pred_lgb))
print('-'*70)
```

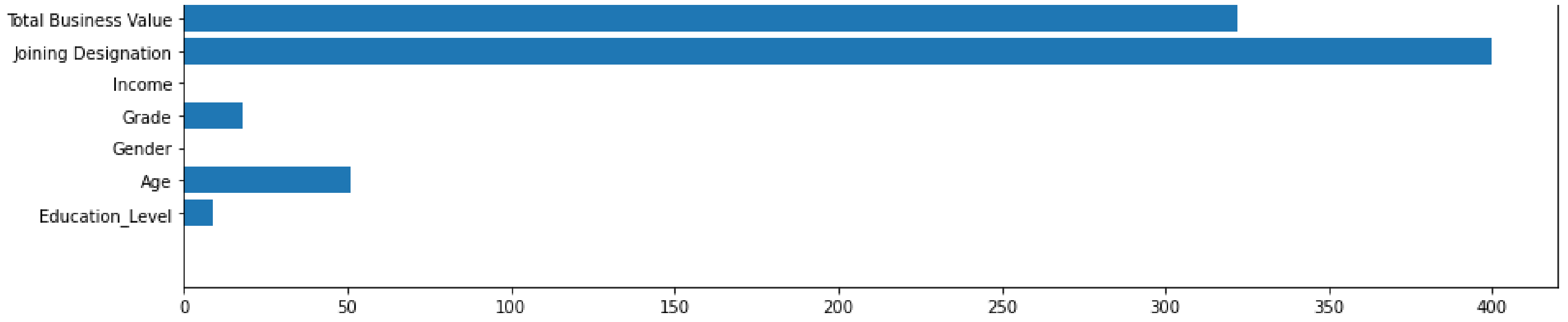
```
----------------------------------------------------------------------
Classification Report: Random Forest
----------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.73      0.55      0.63       139
           1       0.83      0.92      0.87       338

    accuracy                           0.81       477
   macro avg       0.78      0.74      0.75       477
weighted avg       0.80      0.81      0.80       477

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

In [508]:
```
conf_matrix_rf = confusion_matrix(y_test,y_pred_lgb)
plt.figure(figsize=(8,5))
sns.heatmap(conf_matrix_rf, annot = True, cmap = 'YlOrRd', fmt="1.0f")
plt.title('Confusion Matrix of Random Forest Model',fontsize = 20, fontweight = 'bold' )
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()
```



In [512]:
```
plt.figure(figsize=(15,15))
plt.barh(df.columns[:], lgb.feature_importances_)
plt.title('Feature Importance in LighGBM Model',fontsize = 20, fontweight = 'bold' )
plt.show()
```





In [ ]:
```
lgbmodel = LGBMClassifier()
```

XGBoost Classifier

In [515]:
```
# !pip install xgboost
```

In [517]:
```
import xgboost as xgb
```

In [518]:
```
xgbmodel = xgb.XGBClassifier(class_weight ='balanced')
```

In [522]:
```
param_grid_xgb = {
    'xgb__learning_rate': [0.005,0.01,0.05,0.1],
    'xgb__n_estimators': [40,60,80,100],
    'xgb__num_leaves': np.arange(10,20,2),
    'xgb__num_iterations': [ 100, 500, 1000, 2000] }


gs_xgbmodel = GridSearchCV(xgbmodel , param_grid_xgb , cv = 3 , n_jobs=-1, verbose=1, scoring = 'roc_auc')
gs_xgbmodel.fit(X_train_sm,y_train_sm)
```

```
Fitting 3 folds for each of 320 candidates, totalling 960 fits
[00:13:58] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/learner.cc:767:
Parameters: { "class_weight", "xgb__learning_rate", "xgb__n_estimators", "xgb__num_iterations", "xgb__num_leaves" } are
not used.
```

Out[522]:
```
▸ GridSearchCV
▸ estimator: XGBClassifier
    ▸ XGBClassifier
```

In [523]:
```
print(f'We can get roc_auc score of {np.round(gs_xgbmodel.best_score_, 4)} using {gs_xgbmodel.best_params_}')
```

```
We can get roc_auc score of 0.8745 using {'xgb__learning_rate': 0.005, 'xgb__n_estimators': 40, 'xgb__num_iterations':
100, 'xgb__num_leaves': 10}
```

Model with the best hyperparameters

In [528]:
```
xgb = xgb.XGBClassifier(learning_rate = 0.005 ,n_estimators = 40, num_iterations=100 , num_leaves = 10)
```

In [529]:
```
xgb.fit(X_train_sm, y_train_sm)
```

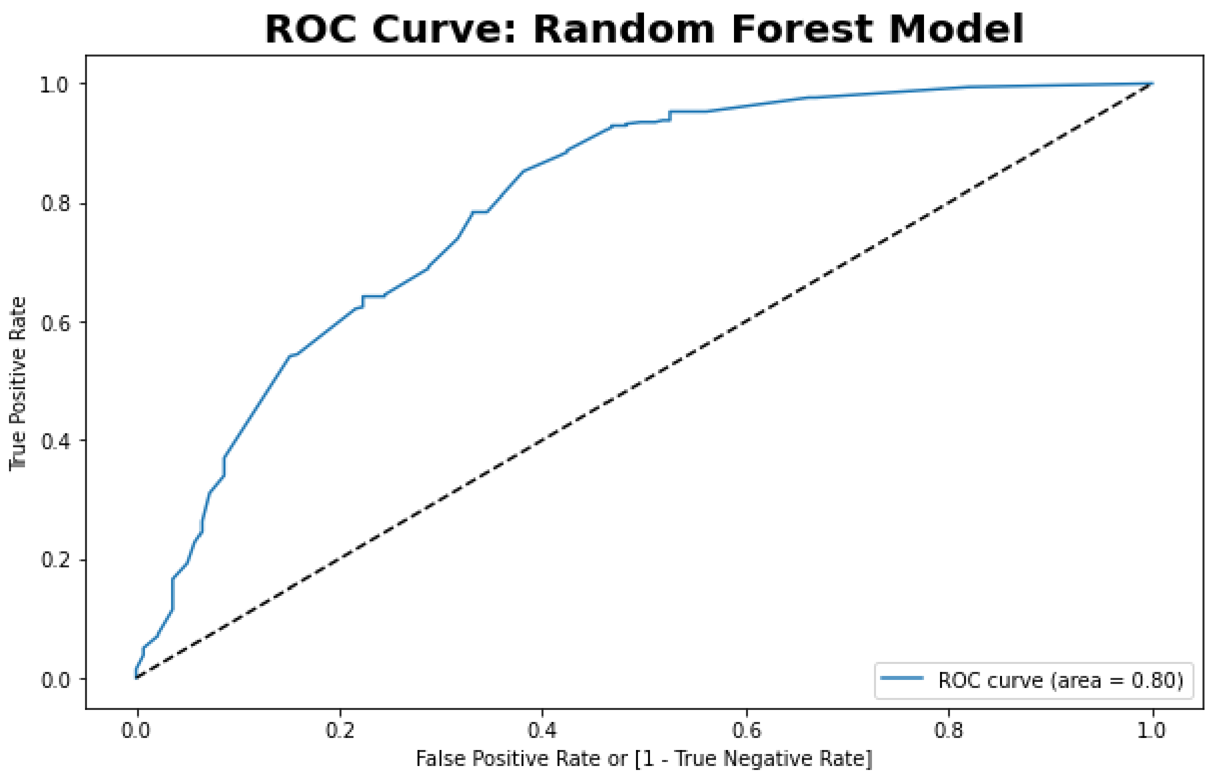
```
[00:31:59] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/learner.cc:767:
Parameters: { "num_iterations", "num_leaves" } are not used.
```

Out[529]:
```
XGBClassifier

XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.005, max_bin=256,
              max_cat_threshold=64, max_cat_to_onehot=4, max_delta_step=0,
              max_depth=6, max_leaves=0, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=40, n_jobs=0,
              num_iterations=100, num_leaves=10, num_parallel_tree=1, ...)
```

In [530]:
```
y_pred_xgb = xgb.predict(X_test)
```

In [531]:
```
#ROC_AUC Curve
from sklearn import metrics
y_pred_xgb_proba = xgb.predict_proba(X_test)
fpr, tpr, thr = metrics.roc_curve(y_test , y_pred_xgb_proba[:,1])
auc_score = metrics.roc_auc_score( y_test, y_pred_xgb_proba[:,1] )
```

In [532]:
```
plt.figure(figsize=(10,6))
plt.plot(fpr,tpr, label = 'ROC curve (area = %0.2f)' % auc_score )
plt.plot([0, 1], [0, 1], 'k--')
plt.title('ROC Curve: Random Forest Model', fontsize = 20, fontweight = 'bold')
plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
plt.ylabel('True Positive Rate')
plt.legend(loc = "lower right" )
plt.show()
print('-'*70)
print(f'ROC_AUC score = \t {np.round(auc_score,5)}')
print('-'*70)
```



```
----------------------------------------------------------------------
ROC_AUC score =          0.79825
----------------------------------------------------------------------
```

In [535]:

```python
#Important Metrics of Model
print('-'*70)
print('Important Metrics of XGBoost Model')
print('-'*70)
print(f'ROC_AUC score = \t {np.round(auc_score,5)}')
print(f'Accuracy of Model : \t {np.round(metrics.accuracy_score(y_test,y_pred_xgb),5)}')
print(f'f1_score of Model : \t {np.round(metrics.f1_score(y_test,y_pred_xgb),5)}')
print(f'Precision of Model : \t {np.round(metrics.precision_score(y_test,y_pred_xgb),5)}')
print(f'Recall of Model : \t {np.round(metrics.recall_score(y_test,y_pred_xgb),5)}')
print('-'*70)
```

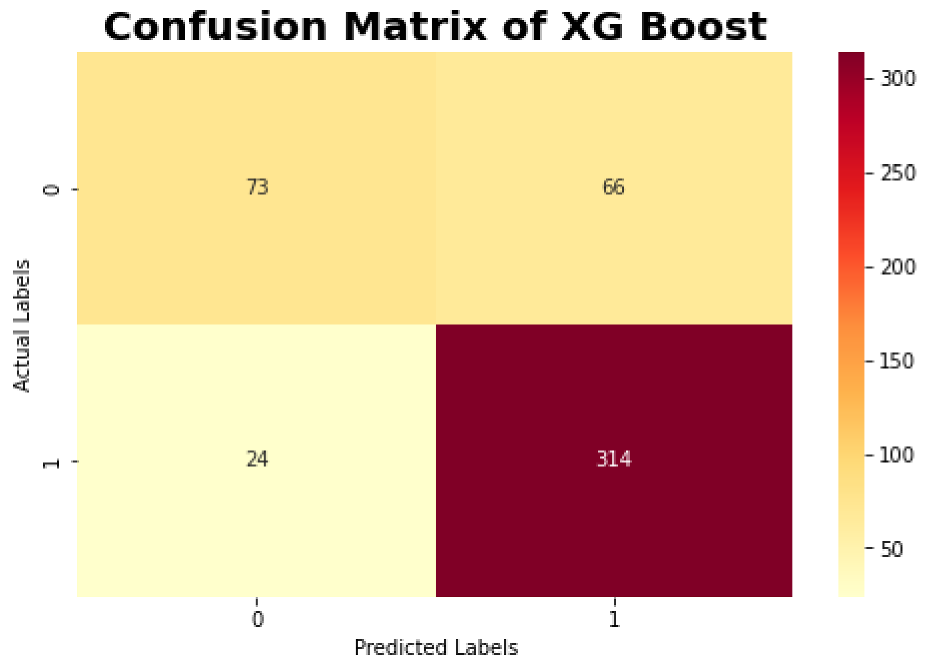
```
----------------------------------------------------------------------
Important Metrics of XGBoost Model
----------------------------------------------------------------------
ROC_AUC score =          0.79825
Accuracy of Model :      0.81132
f1_score of Model :      0.87465
Precision of Model :     0.82632
Recall of Model :        0.92899
----------------------------------------------------------------------
```

In [536]:

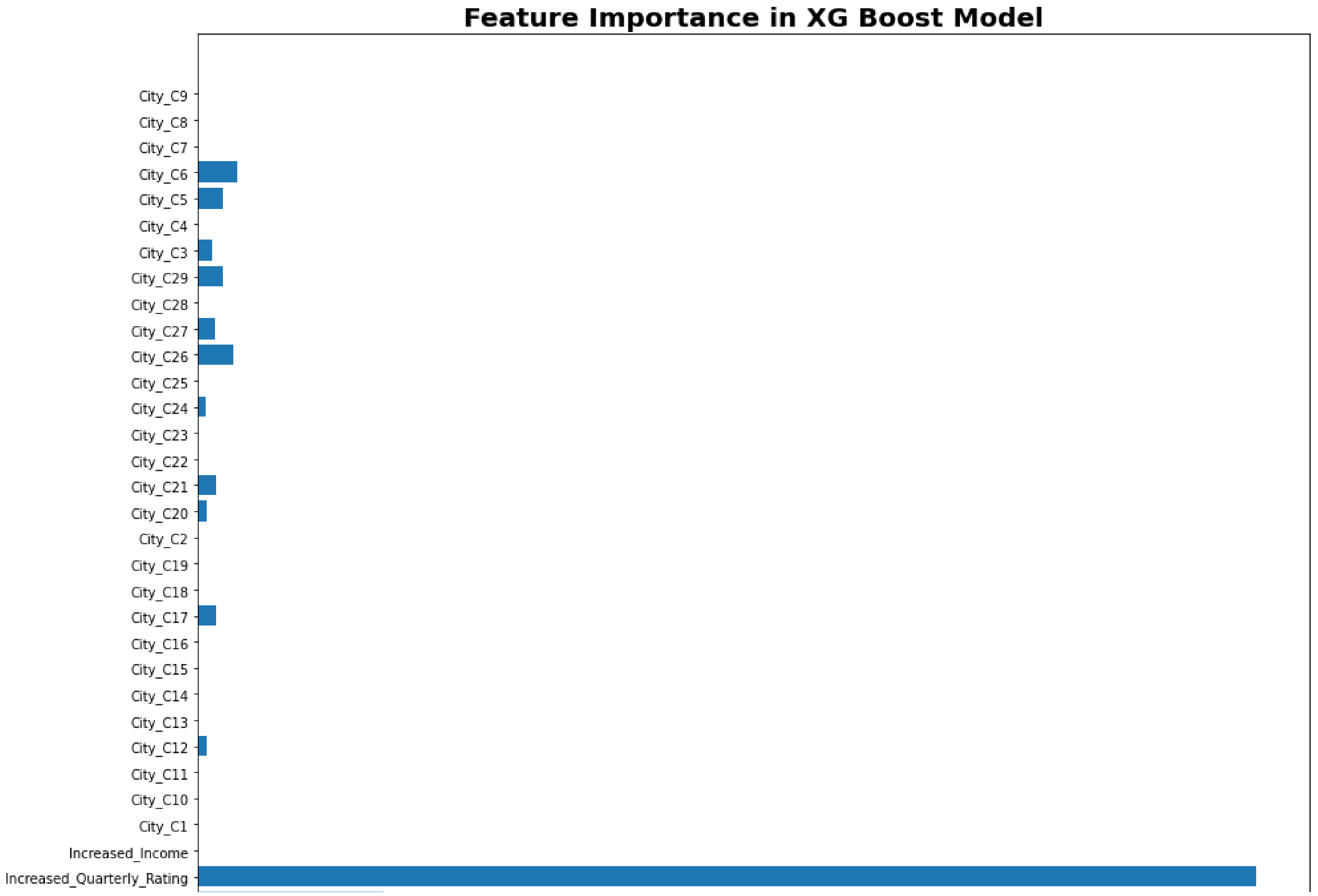
```python
# Classification Report
from sklearn.metrics import classification_report,confusion_matrix
print('-'*70)
print('Classification Report: XG Boost')
print('-'*70)
print(classification_report(y_test,y_pred_xgb))
print('-'*70)
```

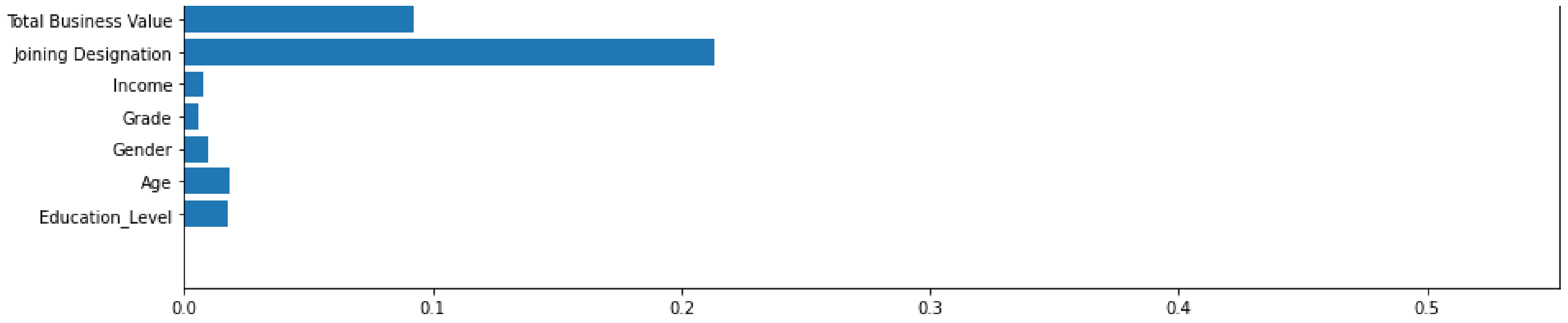
```
----------------------------------------------------------------------
Classification Report: XG Boost
----------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.75      0.53      0.62       139
           1       0.83      0.93      0.87       338

    accuracy                           0.81       477
   macro avg       0.79      0.73      0.75       477
weighted avg       0.80      0.81      0.80       477

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

In [537]:
```
conf_matrix_rf = confusion_matrix(y_test,y_pred_xgb)
plt.figure(figsize=(8,5))
sns.heatmap(conf_matrix_rf, annot = True, cmap = 'YlOrRd', fmt="1.0f")
plt.title('Confusion Matrix of XG Boost',fontsize = 20, fontweight = 'bold' )
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()
```



In [538]:
```
plt.figure(figsize=(15,15))
plt.barh(df.columns[:], xgb.feature_importances_)
plt.title('Feature Importance in XG Boost Model',fontsize = 20, fontweight = 'bold' )
plt.show()
```





Actionable Insights & Recommendations

Observation 1:

```
- Here the Target variable is not shown explicitly
- from the observation the last working day can be made the Target column
- We need to convert the nan values to 0 and others to 1 maybe , need to converm later
```

Observation 2

```
- in the description there are only 14 columns but in the dataset there is 15 :
- unnamed column can be dropped
```

Observation 3

```
- will have to convert MMM-YY to date time format
- will have to convert Dateofjoining to date time format
- last working date should also be converted to the same
```

Observation 4:

```
- there is no point in doing a heatmap now as the data is not on id level and there needs to be alot of work don
e
```

Observation 5:

```
- There are 31% missing data in the age column
- There are 27% missing data in the gender column
- There are 91% missing data in the lastworkingday column but this is the target variable and can be dealt with
 seperatly
```

Observation 6:

```
- Now we have to stitch back the dataset back to original form as the imupation has been done
- for that we can use the join / merge or concat for this
```

Observation 9 :

```
- Age : the maximum age is 58 and the minimum age is 21
- Income : Mean income - 59,334 and max income is 1,88,418 with 75% people making less than 75,986
- Total business value of 50% people are around 8,17,680
```

Observation 10 :

```
- The contineous columns like 'Age', 'Income', 'Total_Business_Value' has got outliers
- For the Age the distribution looks almost normal with some outliers
- For Income the distribution is not normal, its to the right and it has got some outliers as well.
```

Observation 12 :

```
- Education Level : Education level of drivers is uniformly distributed. All three categories have almost equal
 number of drivers.
- The proportion of gender and education is more or less the same for both the employees who left the organizati
on and those who did not leave.
- Grade : 5 unique Grades, Grade 2 has highest & Grade 5 has lowest number of drivers
- Joining Designation :5 unique categories present. JD-1 has highest count & JD-5 has lowest count.
- The employees who have their grade as 3 or 4 at the time of joiningDesignation are less likely to leave the or
ganization.
- The employees whose quarterly rating has increased are less likely to leave the organization.
```

Observation 13 :

```
- For the Age bins we can see that - lesser age with less income has more chance of leaving
- for the age bin of same lesss with large income has a more probability to stay
- There is a great correlation between Income and Grade ,indicating drivers with higher grades have higher month
ly income.
- There is a significant correlation between Joining Designation and Grade, This indicates as the Grade of Drive
r increases, joining designation also increases.
```

Model-01 Random Forest : Hyperparameter tuning using GridSearch_CV

- Model was created using Bagging Technique : Random Forest. Hyperparameter tunning was done using GridSearchCV. Best model was selected based on highest ROC_AUC score.

Important Metrics of Random Forest Model

```
ROC_AUC score =      0.80639
Accuracy of Model :      0.80294
f1_score of Model :      0.86377
Precision of Model :      0.84659
Recall of Model :      0.88166
```

Feature Importance of Random Forest: =>

```
- Total Business Value has highest importance followed by City.
```

Model-02 LightGBM BOOSTING : Hyperparameter tuning using GridSearch_CV

- Model was created using Boosting Technique : LightGBM. Hyperparameter tunning was done using GridSearchCV. Best model was selected based on highest ROC_AUC score.

Important Metrics of LightGBM Model

```
ROC_AUC score =      0.80491
Accuracy of Model :      0.81132
f1_score of Model :      0.87324
Precision of Model :      0.83333
Recall of Model :      0.91716
```

Feature Importance of LightGBM Model: =>

```
- Joining Designation has highest importance followed by City.
```

Model-03 XG BOOSTING : Hyperparameter tuning using GridSearch_CV

- Model was created using Boosting Technique : XGBM. Hyperparameter tunning was done using GridSearchCV. Best model was selected based on highest ROC_AUC score.

Important Metrics of XGBoost Model

```
ROC_AUC score =      0.79825
Accuracy of Model :      0.81132
f1_score of Model :      0.87465
Precision of Model :      0.82632
Recall of Model :      0.92899
```

Feature Importance of XGBoost Model: =>

```
- Increased Quarterly Rating has highest importance followed by City.
```

Recommendations

- Action points for Business based on Analysis:
- Churn rate of Drivers is very high (i.e. around 70%). This is not a healthy situation for business.
- Based on both model created, it was concluded that City & Quarterly Rating has significant impact on Churning of Drivers.
- Company should take extra care of Cities where Churn rate is high.
- Company should take extra efforts in terms of training of compensation to improve the Quarterly Rating of Drivers.
- those drivers tends to stay whose incomes and quaterly ratings are increased, thus Ola needs to increase income and quaterly rating to retain drivers who are working with them for long time.

In [ ]:

```

```