In [194…

```python
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn import preprocessing


import warnings
warnings.filterwarnings('ignore')
```

In [195…

```python
df = pd.read_csv('ola_driver_scaler.csv')
```

In [196…

```python
df.shape
#shape of data
```

Out[196… (19104, 14)

In [197…

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

Out[197…

	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
5	5	12/01/19	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	0
6	6	01/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	0
7	7	02/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	0
8	8	03/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	350000
9	9	04/01/20	4	43.0	0.0	C13	2	65603	12/07/19	27/04/20	2	2	0

In [198…

```python
#delete Unnamed
df = df.drop(df.columns[[0]], axis=1)
```

In [199…

```python
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  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 [200…

```python
#Presuming the input date format to be DD MM YY
df['MMM-YY'] =  pd.to_datetime(df['MMM-YY'], infer_datetime_format = True)
df['LastWorkingDate'] =  pd.to_datetime(df['LastWorkingDate'], infer_datetime_format = True)
df['Dateofjoining'] =  pd.to_datetime(df['Dateofjoining'], infer_datetime_format = True)
```

In [201…

```python
#Outout format YYYY MM DD
```

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

Out[201…

	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
5	2019-12-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	0	1
6	2020-01-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	0	1
7	2020-02-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	0	1
8	2020-03-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	350000	1
9	2020-04-01	4	43.0	0.0	C13	2	65603	2019-12-07	2020-04-27	2	2	0	1

In [202…

```
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 [203…

```
df.nunique()
```

Out[203…

```
MMM-YY                     24
Driver_ID                2381
Age                        36
Gender                      2
City                       29
Education_Level             3
Income                   2383
Dateofjoining             869
LastWorkingDate           493
Joining Designation         5
Grade                       5
Total Business Value    10181
Quarterly Rating            4
dtype: int64
```

In [204…

```
##conversion of categorical attributes to 'category'
#City is a categorical variable that is not coverted to numeric data type we can do one hot encoding on it.
#Target encoding or Ordinal Encoding is not possible for it.
```

In [205…

```
df['MMM-YY'].value_counts()
```

Out[205…
```
2019-01-01    1022
2019-02-01     944
2019-03-01     870
2020-12-01     819
2020-10-01     818
2020-08-01     812
2020-09-01     809
2020-07-01     806
2020-11-01     805
2019-12-01     795
2019-04-01     794
2020-01-01     782
2019-11-01     781
2020-06-01     770
2020-05-01     766
2019-05-01     764
2019-09-01     762
2020-02-01     761
2019-07-01     757
2019-08-01     754
2019-10-01     739
2020-04-01     729
2019-06-01     726
2020-03-01     719
Name: MMM-YY, dtype: int64
```

In [206…
```
df['Age'].value_counts()
```

Out[206…
```
36.0    1283
33.0    1250
34.0    1234
30.0    1146
32.0    1143
35.0    1138
31.0    1076
29.0    1013
37.0     862
38.0     854
39.0     788
28.0     772
27.0     744
40.0     701
41.0     661
26.0     566
42.0     478
25.0     449
44.0     407
43.0     399
45.0     371
46.0     350
24.0     274
47.0     224
23.0     193
48.0     144
49.0      99
22.0      92
52.0      78
51.0      72
50.0      69
21.0      35
53.0      26
54.0      24
55.0      21
58.0       7
Name: Age, dtype: int64
```

In [207…
```
df['Gender'].value_counts()
```

Out[207…
```
0.0    11074
1.0     7978
Name: Gender, dtype: int64
```

In [208…
```
df['City'].value_counts()
```

Out[208…
```
C20    1008
C29     900
C26     869
C22     809
C27     786
```

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

In [209…
```
df['Education_Level'].value_counts()
```

Out[209…
```
1    6864
2    6327
0    5913
Name: Education_Level, dtype: int64
```

In [210…
```
df['Joining Designation'].value_counts()
```

Out[210…
```
1    9831
2    5955
3    2847
4     341
5     130
Name: Joining Designation, dtype: int64
```

In [211…
```
df['Grade'].value_counts()
```

Out[211…
```
2    6627
1    5202
3    4826
4    2144
5     305
Name: Grade, dtype: int64
```

In [212…
```
df['Quarterly Rating'].value_counts()
```

Out[212…
```
1    7679
2    5553
3    3895
4    1977
Name: Quarterly Rating, dtype: int64
```

In [213…
```
#Missing value detection
df.isna().sum()
```

Out[213…
```
MMM-YY                      0
Driver_ID                   0
Age                        61
Gender                     52
City                        0
Education_Level             0
Income                      0
Dateofjoining               0
LastWorkingDate         17488
Joining Designation         0
```

```
Grade                      0
Total Business Value       0
Quarterly Rating           0
dtype: int64
```

In [214…

```
#Age, Gender have some missing values. LastWorkingDate has plenty of missing values.
```

In [215…

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

Out[215…

	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
5	2019-12-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	0	1
6	2020-01-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	0	1
7	2020-02-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	0	1
8	2020-03-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	350000	1
9	2020-04-01	4	43.0	0.0	C13	2	65603	2019-12-07	2020-04-27	2	2	0	1

In [216…

```
#Univariate analysis and bivaraite analsis is done after aggregation and feature engineering of the data.
#Otherwse we end up with false / erronous graphical analysis
```

In [217…

```
#Feature Engineering
```

In [218…

```
df.head(50)
```

Out[218…

	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
5	2019-12-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	0	1
6	2020-01-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	0	1
7	2020-02-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	0	1
8	2020-03-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	350000	1
9	2020-04-01	4	43.0	0.0	C13	2	65603	2019-12-07	2020-04-27	2	2	0	1
10	2019-01-01	5	29.0	0.0	C9	0	46368	2019-01-09	NaT	1	1	0	1
	2019-												

11	02-01	5	29.0	0.0	C9	0	46368	2019-01-09	NaT	1	1	120360	1
12	2019-03-01	5	29.0	0.0	C9	0	46368	2019-01-09	2019-03-07	1	1	0	1
13	2020-08-01	6	31.0	1.0	C11	1	78728	2020-07-31	NaT	3	3	0	1
14	2020-09-01	6	31.0	1.0	C11	1	78728	2020-07-31	NaT	3	3	0	1
15	2020-10-01	6	31.0	1.0	C11	1	78728	2020-07-31	NaT	3	3	0	2
16	2020-11-01	6	31.0	1.0	C11	1	78728	2020-07-31	NaT	3	3	1265000	2
17	2020-12-01	6	31.0	1.0	C11	1	78728	2020-07-31	NaT	3	3	0	2
18	2020-09-01	8	34.0	0.0	C2	0	70656	2020-09-19	NaT	3	3	0	1
19	2020-10-01	8	34.0	0.0	C2	0	70656	2020-09-19	NaT	3	3	0	1
20	2020-11-01	8	34.0	0.0	C2	0	70656	2020-09-19	2020-11-15	3	3	0	1
21	2020-12-01	11	28.0	1.0	C19	2	42172	2020-12-07	NaT	1	1	0	1
22	2019-07-01	12	35.0	0.0	C23	2	28116	2019-06-29	NaT	1	1	500000	4
23	2019-08-01	12	35.0	0.0	C23	2	28116	2019-06-29	NaT	1	1	1707180	4
24	2019-09-01	12	35.0	0.0	C23	2	28116	2019-06-29	NaT	1	1	400000	4
25	2019-10-01	12	35.0	0.0	C23	2	28116	2019-06-29	NaT	1	1	0	1
26	2019-11-01	12	35.0	0.0	C23	2	28116	2019-06-29	NaT	1	1	0	1
27	2019-12-01	12	35.0	0.0	C23	2	28116	2019-06-29	2019-12-21	1	1	0	1
28	2019-01-01	13	29.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	250000	1
29	2019-02-01	13	29.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	1719680	1
30	2019-03-01	13	29.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	545240	1
31	2019-04-01	13	29.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	250000	2
32	2019-05-01	13	29.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	895510	2
33	2019-06-01	13	29.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	0	2
34	2019-07-01	13	29.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	350650	2
35	2019-08-01	13	29.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	708360	2
36	2019-09-01	13	29.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	1190290	2
37	2019-10-01	13	29.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	0	1
38	2019-11-01	13	30.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	200000	1
39	2019-12-01	13	30.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	300000	1
40	2020-01-01	13	30.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	151110	1
41	2020-02-01	13	30.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	200000	1
42	2020-03-01	13	30.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	1593590	1
43	2020-04-01	13	30.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	0	1
44	2020-05-01	13	30.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	400000	1
45	2020-06-01	13	30.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	258610	1
46	2020-	13	30.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	150000	1

	07-01												
47	2020-08-01	13	30.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	500000	1
48	2020-09-01	13	30.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	200000	1
49	2020-10-01	13	30.0	0.0	C19	2	119227	2015-05-28	NaT	1	4	350000	1

In [219…

```
#We cannot do first aggreagation and the nfeature engineering. For quarterly ratings if we first aggregate
#and then try to calculate from the aggregated data for which driver the quarterly rating has increases then
#that would be IMPOSSIBLE from the aggreagatd data !
```

In [220…

```
# 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

#As we can see for driver ID 13 quarterly rating increases but then also deacreases. We make the target variable
# as 1 if the rating increases and either the driver maintains it or it increases further. The rating has to be
# MONOTONICALLY INCREASING


def ratincr(y):

    res = 0
    #max rating till now
    mr = y[0]
    for i in range(1, len(y)):
        if y[i] > mr:
            res = 1
            mr = y[i]
        elif y[i] < mr :
            res = 0
            return res
        else:
            continue

    return res


df['QR Increase'] = df.groupby('Driver_ID')['Quarterly Rating'].transform(lambda x: ratincr(x.values))
```

In [221…

```
df['QR Increase'].value_counts()
```

Out[221…

```
0    17394
1     1710
Name: QR Increase, dtype: int64
```

In [222…

```
# 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

#We use the income column for this. We set value 1 for the drivers whose income has increased STRICTLY
#MONOTONICALLY. If income increases then decreases we don't consider it. Also if income reamains the same
#we give value 0.


df['Income Increase'] = df.groupby('Driver_ID')['Income'].transform(lambda x: ratincr(x.values))
```

In [223…

```
df['Income Increase'].value_counts()
```

Out[223…

```
0    18128
1      976
Name: Income Increase, dtype: int64
```

In [224…

```
# 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

df['Left_Ola'] =  df.groupby('Driver_ID')['LastWorkingDate'].transform(lambda x: int(pd.notna (x.values[-1])))
```

In [225…

```
df['Left_Ola'].value_counts()
```

Out[225…

```
1    10359
0     8745
```

```
Name: Left_Ola, dtype: int64
```

In [226…

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

Out[226…

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterly Rating	In
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	
5	2019-12-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	0	1	
6	2020-01-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	0	1	
7	2020-02-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	0	1	
8	2020-03-01	4	43.0	0.0	C13	2	65603	2019-12-07	NaT	2	2	350000	1	
9	2020-04-01	4	43.0	0.0	C13	2	65603	2019-12-07	2020-04-27	2	2	0	1	

In [227…

```python
#Now we can aggreagate the data
create_segment_dict = {

    'MMM-YY' : 'first',
    'Age' : 'mean',
    'Gender' : 'first',
    'City' : 'first',
    'Education_Level' : 'first',
    'Income' : 'mean',
    'Dateofjoining' : 'first',
    'LastWorkingDate' : 'last',
    'Joining Designation' : 'first',
    'Grade' : 'first',
    'Total Business Value' : 'sum',
    'Quarterly Rating': 'mean',
    'QR Increase' : 'first',
    'Income Increase' : 'first',
    'Left_Ola' : 'first'

}
```

In [228…

```python
seg = df.groupby('Driver_ID').agg(create_segment_dict).reset_index()
```

In [229…

```python
seg.head(20)
```

Out[229…

	Driver_ID	MMM-YY	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value
0	1	2019-01-01	28.000000	0.0	C23	2	57387.000000	2018-12-24	2019-03-11	1	1	1715580
1	2	2020-11-01	31.000000	0.0	C7	2	67016.000000	2020-11-06	NaT	2	2	0
2	4	2019-12-01	43.000000	0.0	C13	2	65603.000000	2019-12-07	2020-04-27	2	2	350000
3	5	2019-01-01	29.000000	0.0	C9	0	46368.000000	2019-01-09	2019-03-07	1	1	120360
4	6	2020-08-01	31.000000	1.0	C11	1	78728.000000	2020-07-31	NaT	3	3	1265000
5	8	2020-09-01	34.000000	0.0	C2	0	70656.000000	2020-09-19	2020-11-15	3	3	0
6	11	2020-12-01	28.000000	1.0	C19	2	42172.000000	2020-12-07	NaT	1	1	0

7	12	2019-07-01	35.000000	0.0	C23	2	28116.000000	2019-06-29	2019-12-21	1	1	2607180
8	13	2019-01-01	29.608696	0.0	C19	2	119227.000000	2015-05-28	2020-11-25	1	4	10213040
9	14	2020-10-01	39.000000	1.0	C26	0	19734.000000	2020-10-16	NaT	3	3	0
10	16	2019-01-01	30.000000	1.0	C23	0	52963.000000	2018-11-30	2019-02-22	2	2	346800
11	17	2019-01-01	42.142857	0.0	C20	2	51099.000000	2018-03-06	2019-07-20	1	1	1017640
12	18	2019-01-01	27.000000	1.0	C17	1	31631.000000	2019-01-09	2019-04-30	1	1	0
13	20	2019-10-01	26.000000	1.0	C19	0	40342.000000	2019-10-25	2020-03-01	3	3	0
14	21	2019-01-01	33.285714	1.0	C29	1	22755.000000	2018-05-12	2020-02-17	1	1	6962550
15	22	2019-01-01	40.400000	0.0	C10	2	31224.000000	2018-05-25	2020-04-26	1	1	7539490
16	24	2019-01-01	30.888889	0.0	C24	2	76308.000000	2018-05-25	2019-10-27	1	2	4101720
17	25	2019-01-01	29.666667	0.0	C24	1	102077.000000	2017-10-30	NaT	1	3	36351110
18	26	2019-01-01	41.833333	0.0	C14	2	126132.333333	2018-05-07	NaT	1	3	69867900
19	29	2019-01-01	30.000000	0.0	C13	2	30312.000000	2018-05-20	2019-05-23	1	1	1273170

In [230…

```
seg.isna().sum()
#Only last working date has missing values now.
#Age and Gender were also missing but once we aggregate data they are not missing.

#This means that for every driver we had the age or gender present in at least one column and when we did
#group by we found it
```

Out[230…

```
Driver_ID                 0
MMM-YY                    0
Age                       0
Gender                    0
City                      0
Education_Level           0
Income                    0
Dateofjoining             0
LastWorkingDate         765
Joining Designation       0
Grade                     0
Total Business Value      0
Quarterly Rating          0
QR Increase               0
Income Increase           0
Left_Ola                  0
dtype: int64
```

In [231…

```
#Missing value treatment
#Last working date is missing in 765 out of 19104 of the rows ie 4% of the the drivers are still
#working for the company and have not left it. We capture this information in the target variable. So
#in order to fill the missing values we cannot do KNN imputation bcoz it needs to have some dependency on
#other features where we are imputing.

#For LastWorkingDate we fill the null values with an arbitraRy date or current date becasue the drivers
#are still working  with the company. I choose 6 months post the last date provided to us i.e. 01/06/2021
```

In [232…

```
date_miss =  pd.to_datetime('01-06-2021', format='%d-%m-%Y')
```

In [233…

```
seg['LastWorkingDate'] =  seg['LastWorkingDate'].fillna(date_miss)
```

In [234…

```
#6 months from the time period provided we set the max date and use it to fill the missing value
np.max(seg['LastWorkingDate'])
```

Out[234…

```
Timestamp('2021-06-01 00:00:00')
```

In [235…
```python
seg['Join Year'] = seg['Dateofjoining'].transform(lambda x : x.year)
seg['Join Month'] = seg['Dateofjoining'].transform(lambda x : x.month)
```

In [236…
```python
#Duration is in days
seg['Duration'] = seg['LastWorkingDate'] - seg['Dateofjoining']
```

In [237…
```python
seg['Duration'] = seg['Duration'].apply(lambda x : x.days)
```

In [238…
```python
seg['Duration']
```

Out[238…
```
0         77
1        207
2        142
3         57
4        305
        ...
2376    2056
2377      61
2378     418
2379     334
2380     358
Name: Duration, Length: 2381, dtype: int64
```

In [239…
```python
seg.head()
```

Out[239…

	Driver_ID	MMM-YY	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterly Rating	In
0	1	2019-01-01	28.0	0.0	C23	2	57387.0	2018-12-24	2019-03-11	1	1	1715580	2.0	
1	2	2020-11-01	31.0	0.0	C7	2	67016.0	2020-11-06	2021-06-01	2	2	0	1.0	
2	4	2019-12-01	43.0	0.0	C13	2	65603.0	2019-12-07	2020-04-27	2	2	350000	1.0	
3	5	2019-01-01	29.0	0.0	C9	0	46368.0	2019-01-09	2019-03-07	1	1	120360	1.0	
4	6	2020-08-01	31.0	1.0	C11	1	78728.0	2020-07-31	2021-06-01	3	3	1265000	1.6	

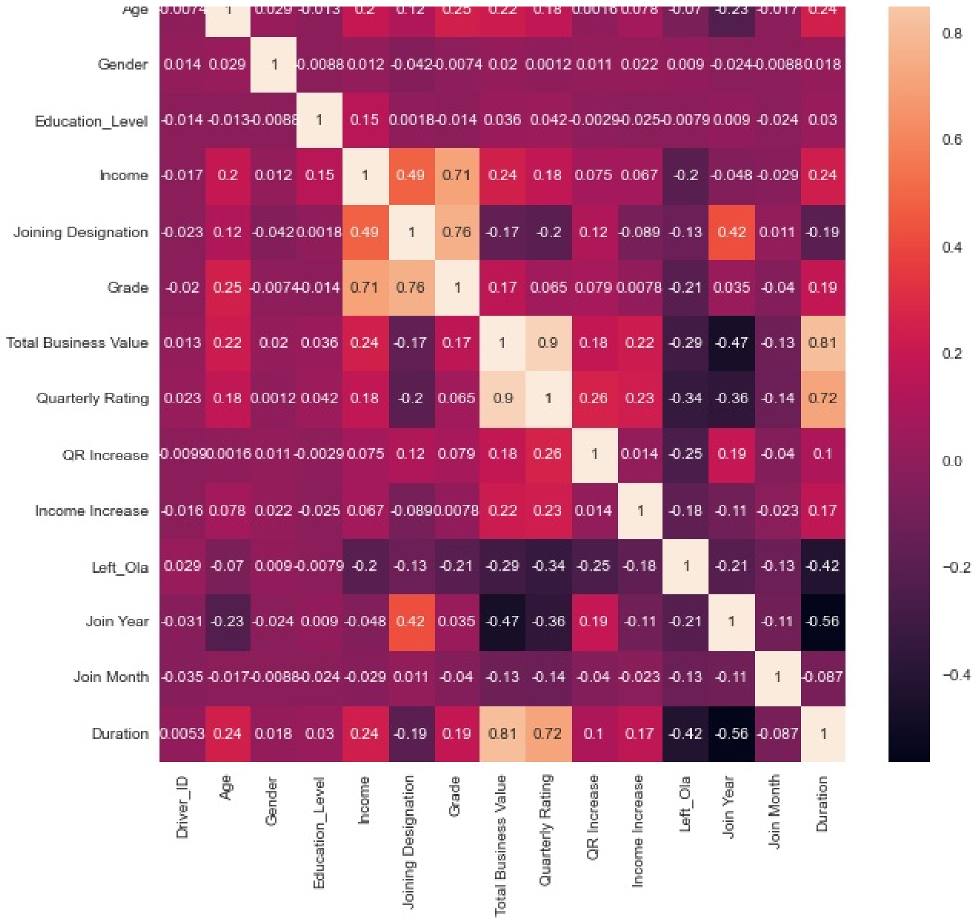
In [240…
```python
seg.describe()
```

Out[240…

	Driver_ID	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	QR Increa
count	2381.000000	2381.000000	2381.000000	2381.00000	2381.000000	2381.000000	2381.000000	2.381000e+03	2381.000000	2381.000
mean	1397.559009	33.369298	0.410332	1.00756	59232.460484	1.820244	2.078538	4.586742e+06	1.566304	0.087
std	806.161628	5.890567	0.491997	0.81629	28298.214012	0.841433	0.931321	9.127115e+06	0.719652	0.282
min	1.000000	21.000000	0.000000	0.00000	10747.000000	1.000000	1.000000	-1.385530e+06	1.000000	0.000
25%	695.000000	29.000000	0.000000	0.00000	39104.000000	1.000000	1.000000	0.000000e+00	1.000000	0.000
50%	1400.000000	33.000000	0.000000	1.00000	55285.000000	2.000000	2.000000	8.176800e+05	1.000000	0.000
75%	2100.000000	37.000000	1.000000	2.00000	75835.000000	2.000000	3.000000	4.173650e+06	2.000000	0.000
max	2788.000000	58.000000	1.000000	2.00000	188418.000000	5.000000	5.000000	9.533106e+07	4.000000	1.000

In [241…
```python
fig, ax = plt.subplots(figsize=(10, 10))
Var_Corr = seg.corr(method = 'spearman')
sns.heatmap(Var_Corr, xticklabels=Var_Corr.columns, yticklabels=Var_Corr.columns, annot=True, ax=ax)

#The new variables added do not show any good correlation with the existing variables

#Duration is highly correalated with quarterly rating and total business value.
```

Out[241…
```
<AxesSubplot:>
```



In [242…

```
seg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Driver_ID             2381 non-null   int64
 1   MMM-YY                2381 non-null   datetime64[ns]
 2   Age                   2381 non-null   float64
 3   Gender                2381 non-null   float64
 4   City                  2381 non-null   object
 5   Education_Level       2381 non-null   int64
 6   Income                2381 non-null   float64
 7   Dateofjoining         2381 non-null   datetime64[ns]
 8   LastWorkingDate       2381 non-null   datetime64[ns]
 9   Joining Designation   2381 non-null   int64
 10  Grade                 2381 non-null   int64
 11  Total Business Value  2381 non-null   int64
 12  Quarterly Rating      2381 non-null   float64
 13  QR Increase           2381 non-null   int64
 14  Income Increase       2381 non-null   int64
 15  Left_Ola              2381 non-null   int64
 16  Join Year             2381 non-null   int64
 17  Join Month            2381 non-null   int64
 18  Duration              2381 non-null   int64
dtypes: datetime64[ns](3), float64(4), int64(11), object(1)
memory usage: 353.6+ KB
```

In [243…

```
seg.nunique()
```

Out[243…

```
Driver_ID               2381
MMM-YY                    24
Age                      663
Gender                     2
City                      29
Education_Level            3
Income                  2339
Dateofjoining            869
LastWorkingDate          494
Joining Designation        5
Grade                      5
Total Business Value    1629
```

```
Quarterly Rating        163
QR Increase               2
Income Increase           2
Left_Ola                  2
Join Year                 8
Join Month               12
Duration                928
dtype: int64
```

In [244…

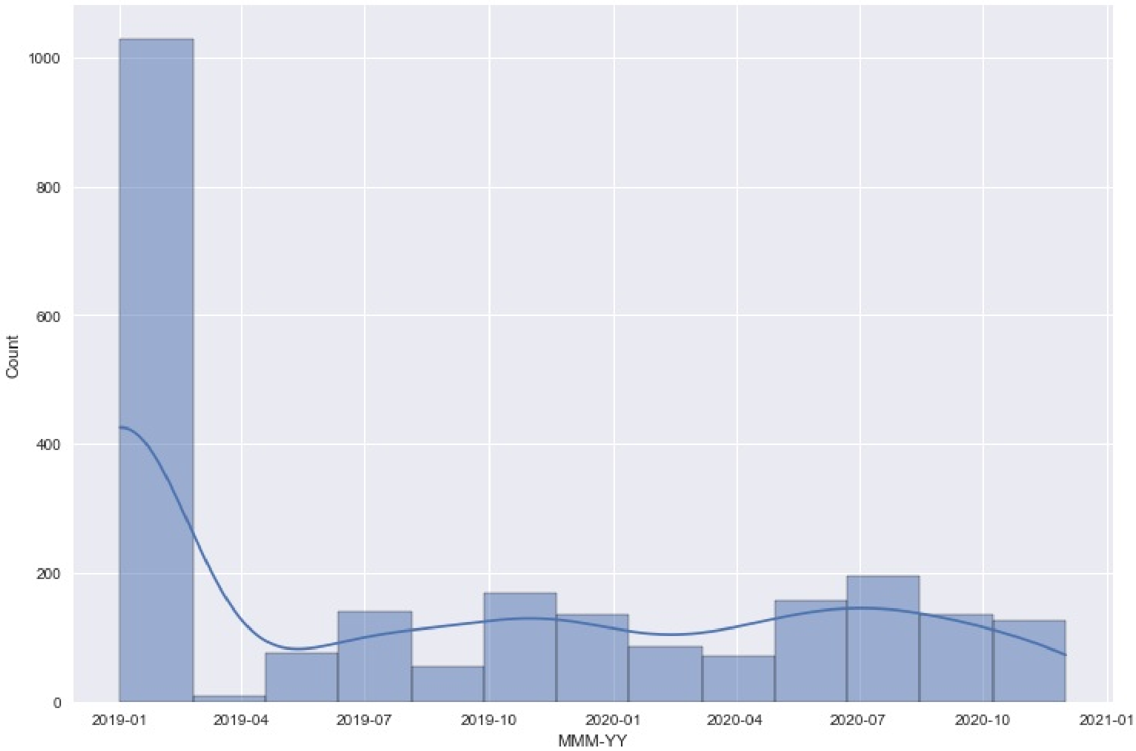
```python
#Univariate and Bivariate anaylysis on the aggregated data

#Discrete variables are :  Age, Dateofjoining , LastWorkingDate
#Continous variables are : Income, TotalBusinessValue, Quarterly Rating
```

In [245…

```python
#MMM-YY
sns.displot( x = 'MMM-YY', data = seg, kde = True, height = 7, aspect = 1.5)
```

Out[245…

```
<seaborn.axisgrid.FacetGrid at 0x1c3013e1d30>
```

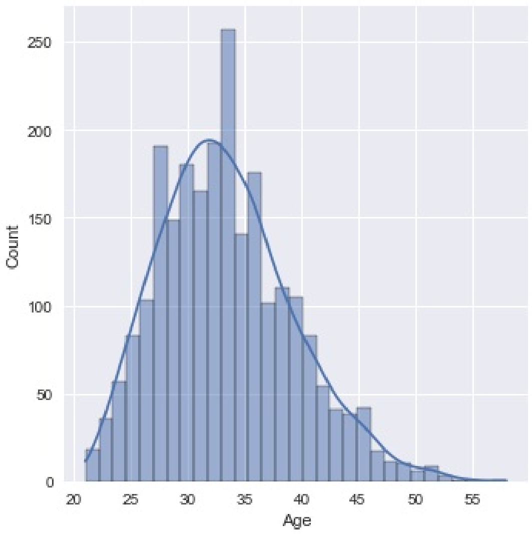


In [246…

```python
sns.displot( x = 'Age', data = seg, kde = True)
```

Out[246…

```
<seaborn.axisgrid.FacetGrid at 0x1c37eaf0790>
```

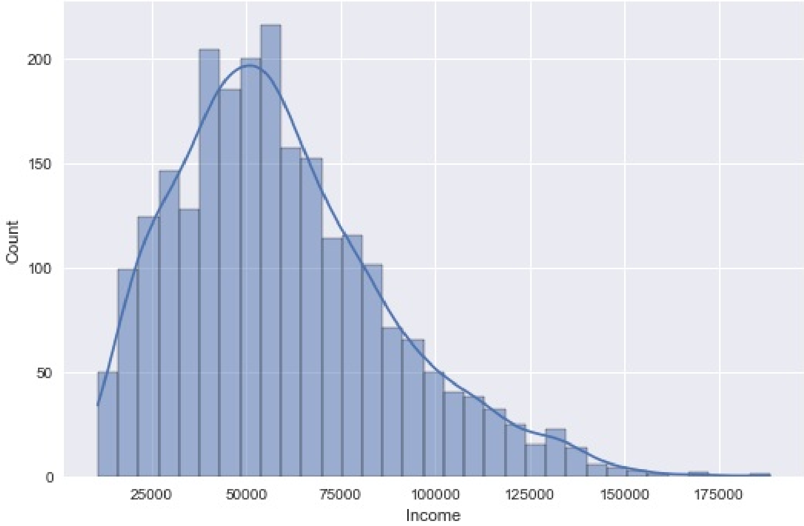


In [247…

```
sns.displot( x = 'Income', data = seg, kde = True, aspect = 1.5)
```

Out[247…

```
<seaborn.axisgrid.FacetGrid at 0x1c37ea75e80>
```

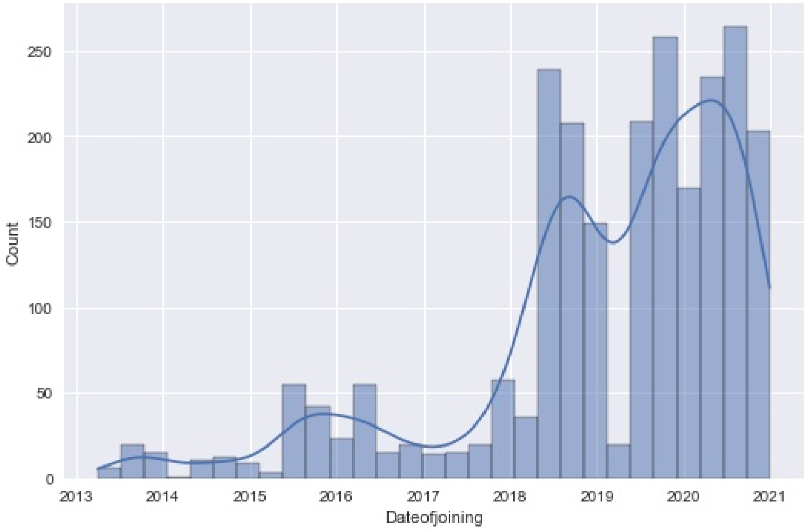


In [248…

```
sns.displot( x = 'Dateofjoining', data = seg, kde = True, aspect = 1.5)
```

Out[248…

```
<seaborn.axisgrid.FacetGrid at 0x1c37e59b670>
```

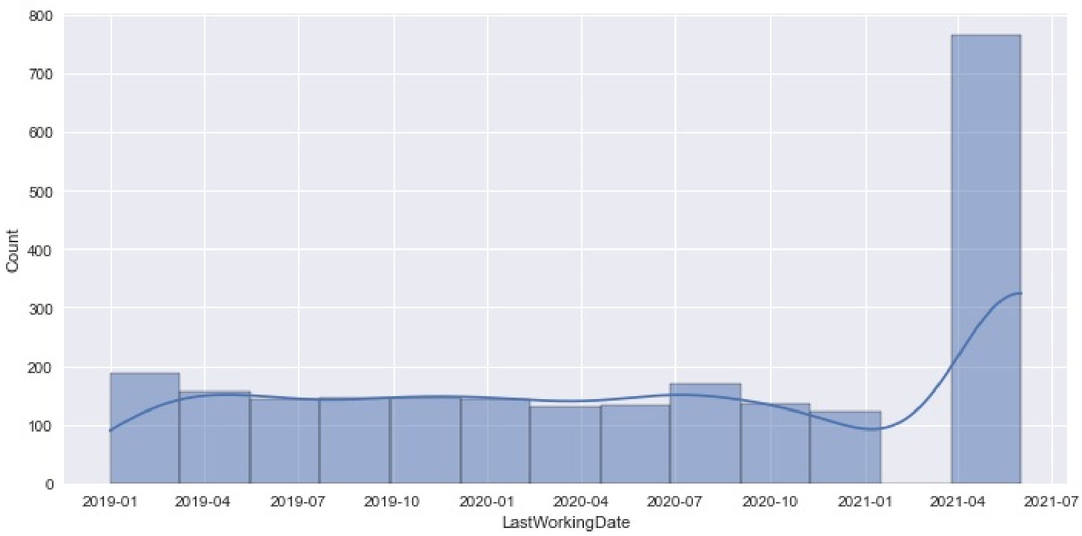


In [249…

```
sns.displot( x = 'LastWorkingDate', data = seg, kde = True, aspect = 2)
```

Out[249…

```
<seaborn.axisgrid.FacetGrid at 0x1c37eaf1520>
```

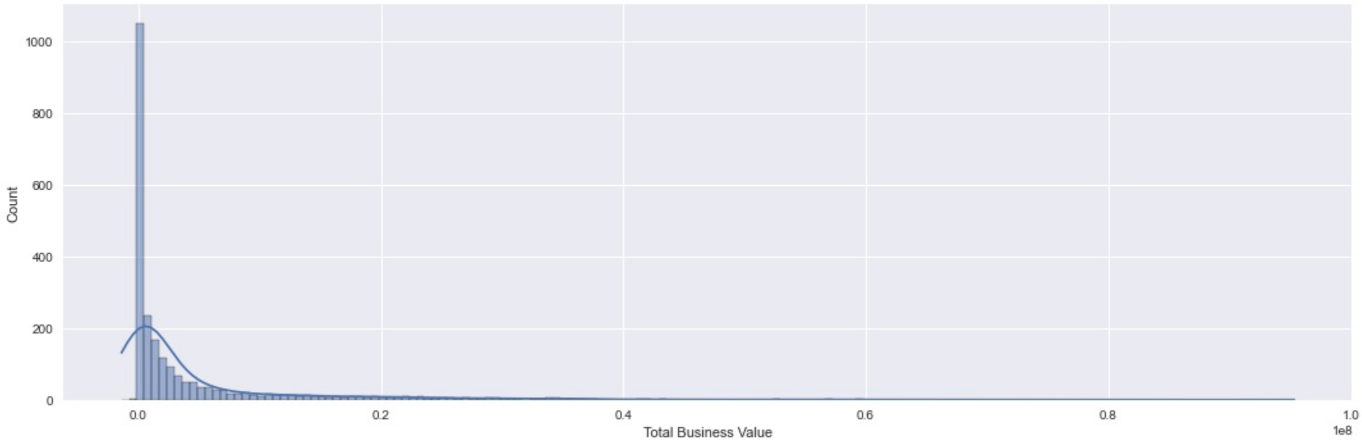


In [250…

```python
sns.displot( x = 'Total Business Value', data = seg, kde = True, aspect = 3)
```

Out[250…

```
<seaborn.axisgrid.FacetGrid at 0x1c37eae0c40>
```

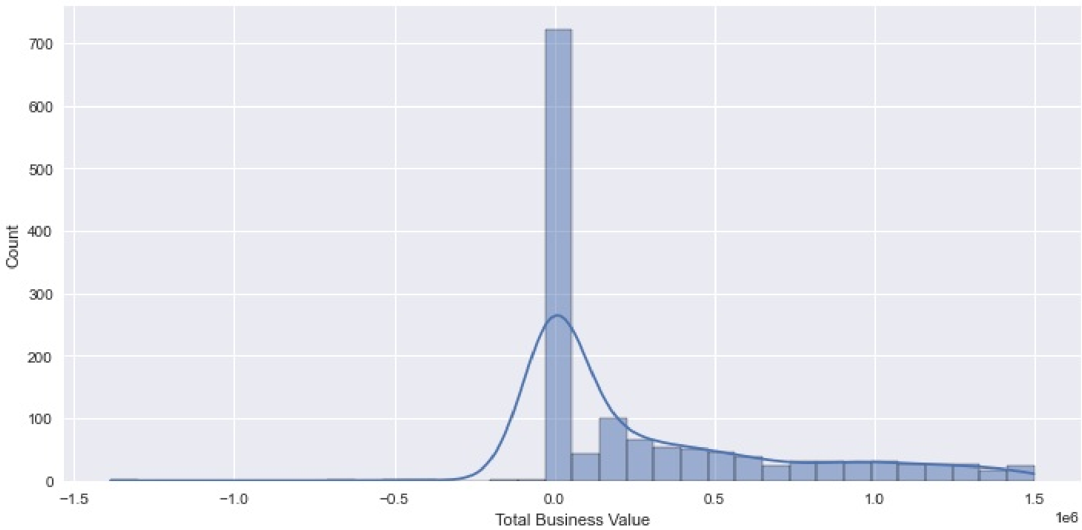


In [251…

```python
sns.displot( seg['Total Business Value'][seg['Total Business Value'] < 1500000], kde = True, height = 5, aspect=
```

Out[251…

```
<seaborn.axisgrid.FacetGrid at 0x1c37e9b4d00>
```



In [252…

```python
#For Total business value we would need to do some outlier treatment
from matplotlib import pyplot
fig, ax = pyplot.subplots(figsize=(15,8))
sns.boxplot(x=seg['Total Business Value'], ax=ax)
```

Out[252…

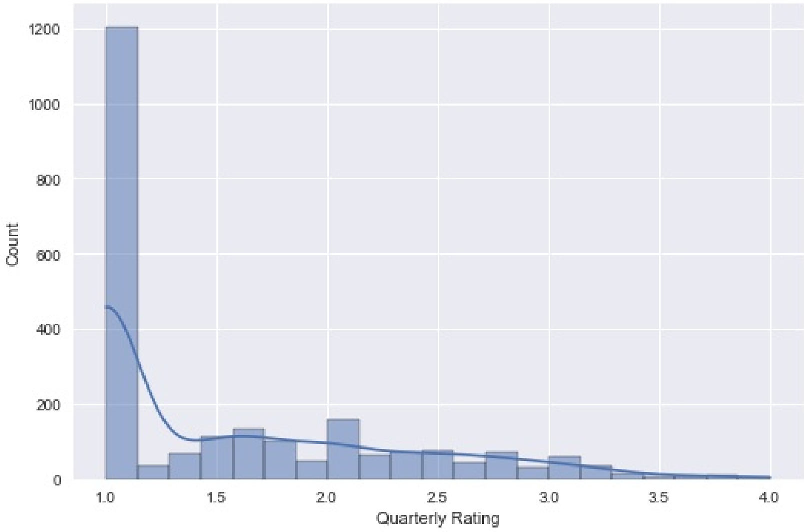
```
<AxesSubplot:xlabel='Total Business Value'>
```

In [253…

```
sns.displot( x = 'Quarterly Rating', data = seg, kde = True, aspect = 1.5)
```

Out[253…

<seaborn.axisgrid.FacetGrid at 0x1c37e907280>

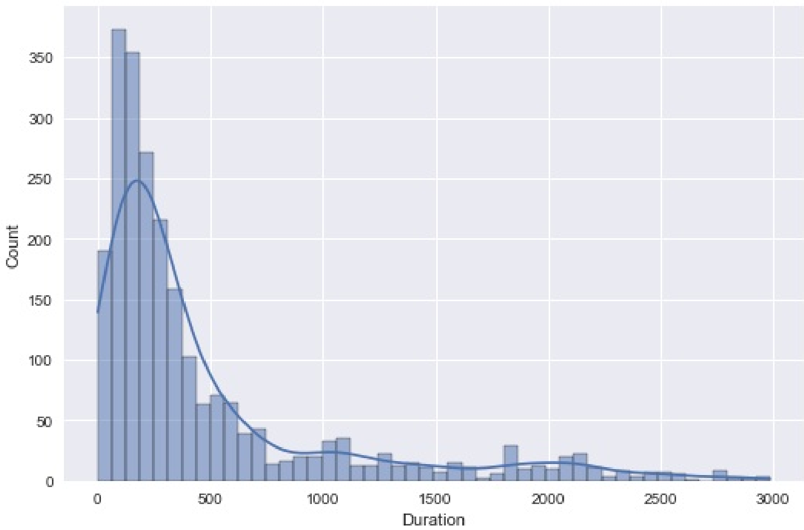


In [254…

```
sns.displot( x = 'Duration', data = seg, kde = True, aspect = 1.5)
```

Out[254…

<seaborn.axisgrid.FacetGrid at 0x1c30131a7f0>



In [255…

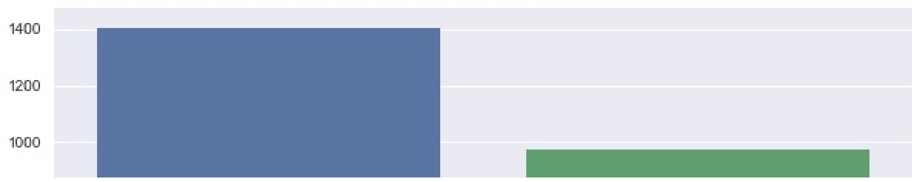
```
#Barplots / Countplots of all categorical variables
```

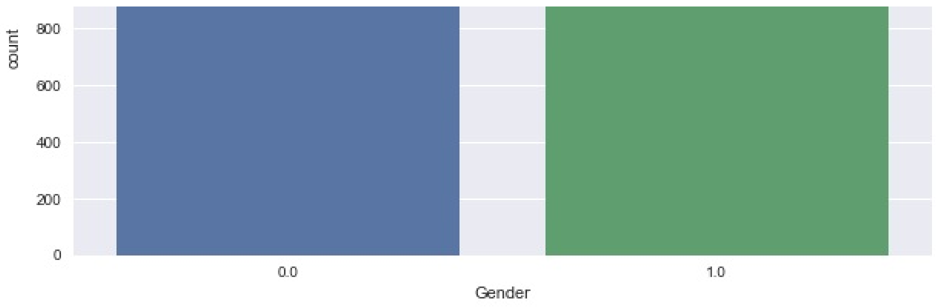
In [256…

```
#Gender
fig, ax = plt.subplots(figsize=(10, 5))
sns.countplot(x= 'Gender', data = seg, ax= ax)
```

Out[256…

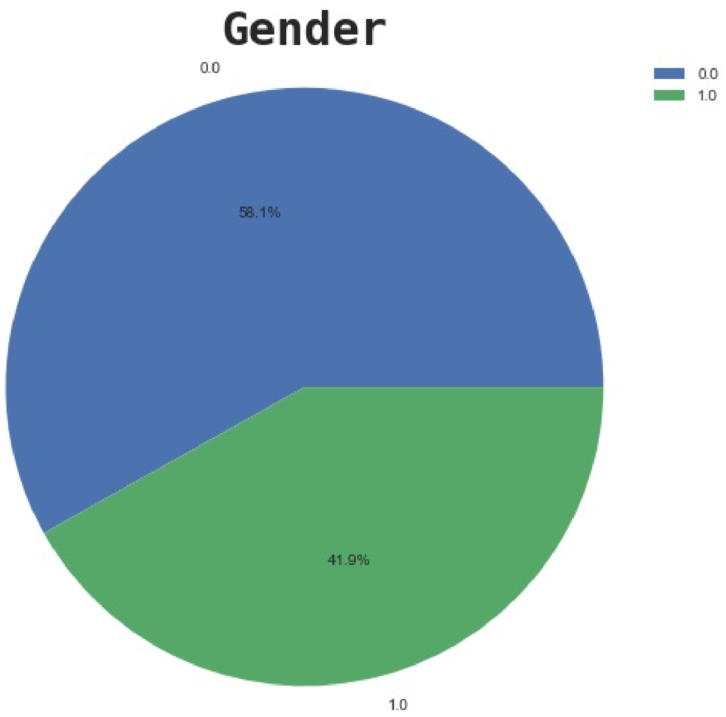
<AxesSubplot:xlabel='Gender', ylabel='count'>





In [257…

```
ssn = df['Gender'].value_counts()
plt.style.use('seaborn')
plt.figure(figsize = (10, 8))
plt.pie(ssn.values, labels = ssn.index, autopct = '%1.1f%%')
plt.title('Gender', fontdict = {'fontname' : 'Monospace','fontsize' : 30, 'fontweight' : 'bold'})
plt.legend()
plt.axis('equal')
plt.show()
```

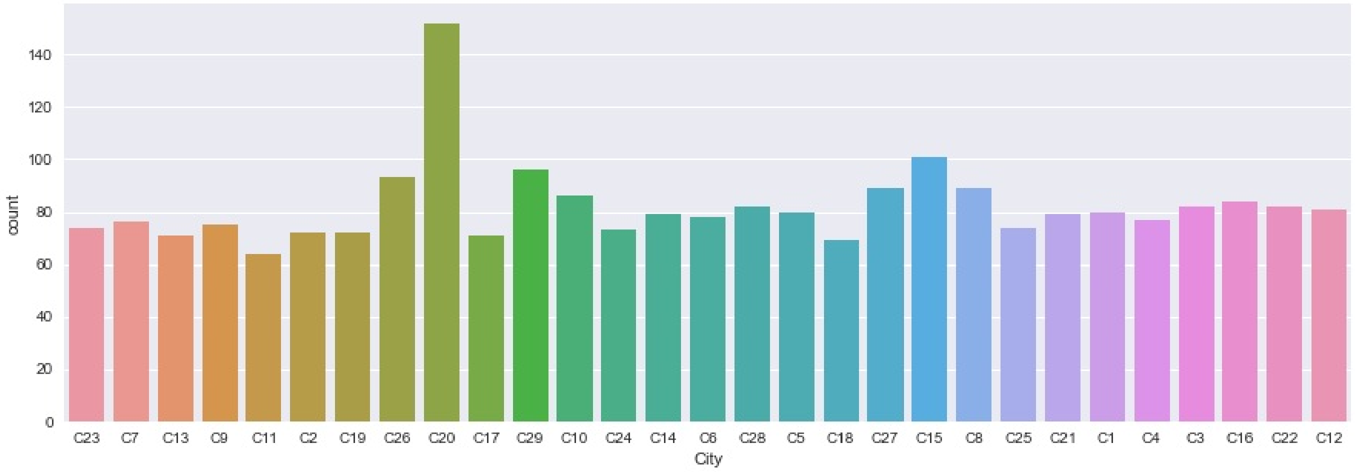


In [258…

```
#City
fig, ax = plt.subplots(figsize=(15, 5))
sns.countplot(x= 'City', data = seg, ax= ax)
```

Out[258…

```
<AxesSubplot:xlabel='City', ylabel='count'>
```

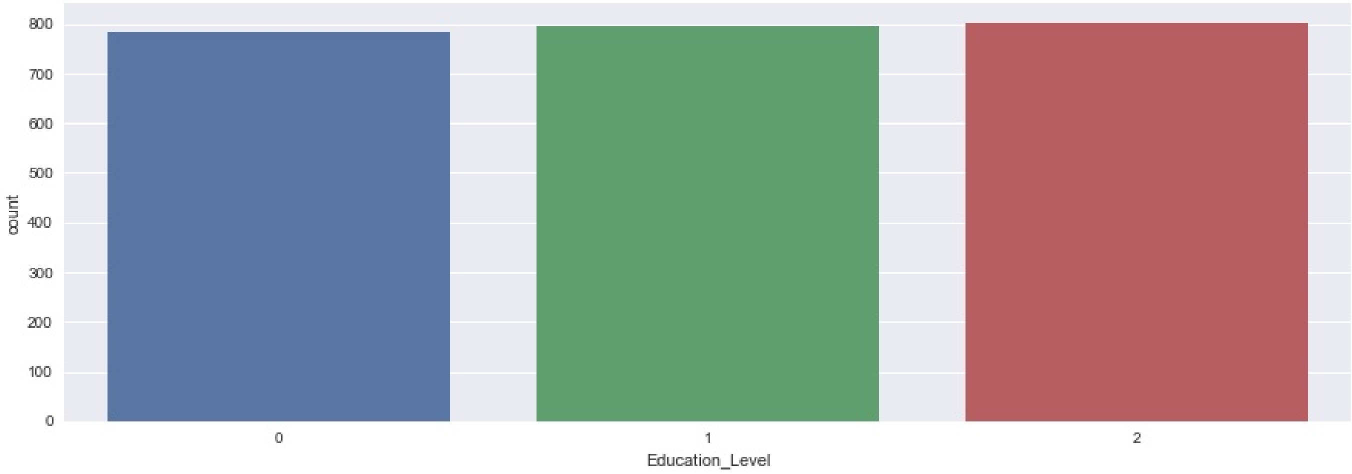


In [259…

```
#Education level
fig, ax = plt.subplots(figsize=(15, 5))
sns.countplot(x= 'Education_Level', data = seg, ax= ax)
```

Out[259…

```
<AxesSubplot:xlabel='Education_Level', ylabel='count'>
```

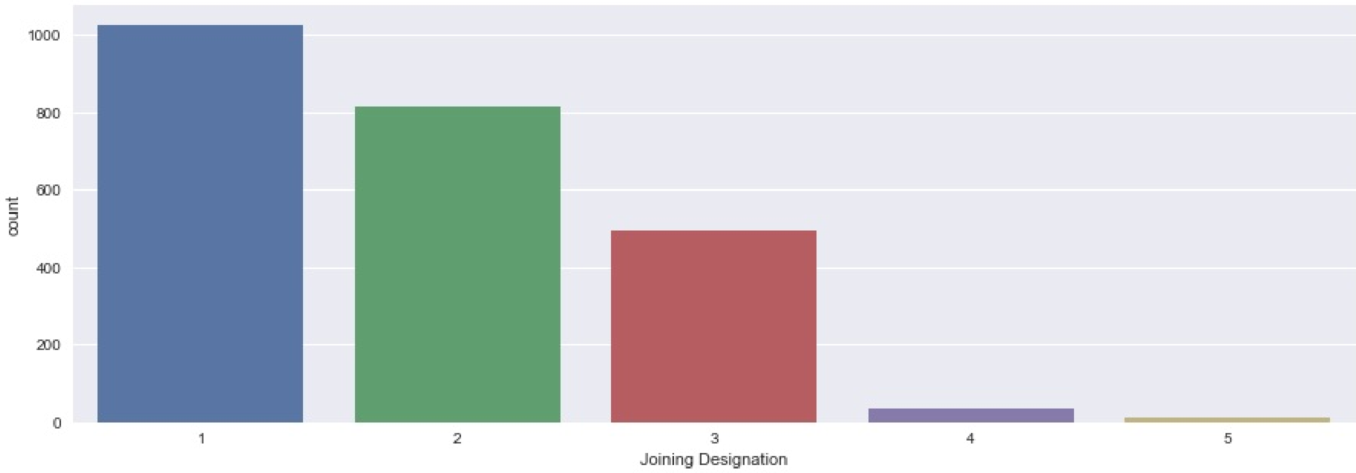


In [260…

```
#Joining Designation
fig, ax = plt.subplots(figsize=(15, 5))
sns.countplot(x= 'Joining Designation', data = seg, ax= ax)
```

Out[260…

```
<AxesSubplot:xlabel='Joining Designation', ylabel='count'>
```

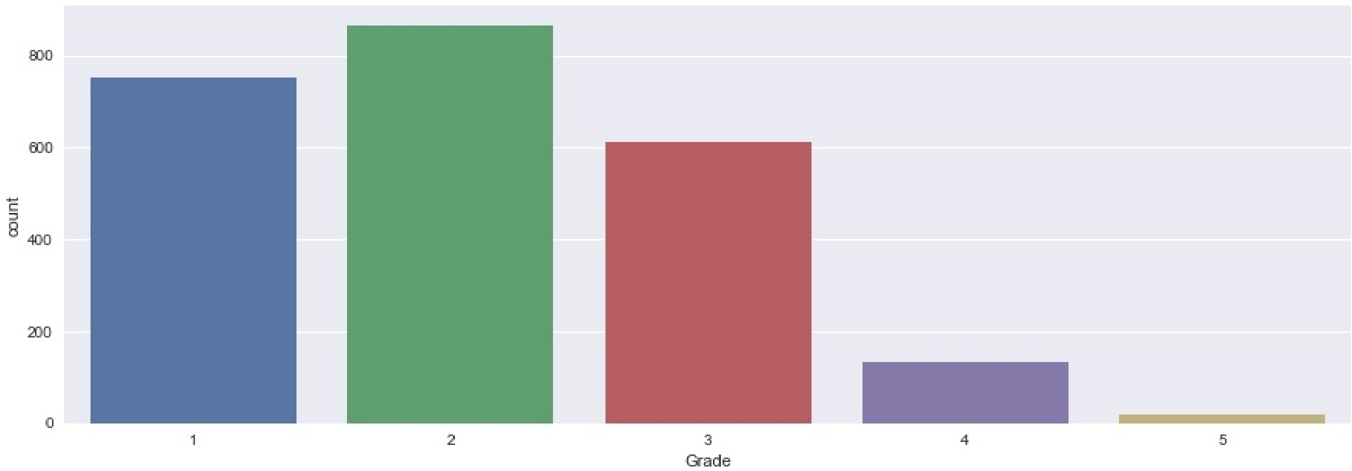


In [261…

```
#Grade
fig, ax = plt.subplots(figsize=(15, 5))
sns.countplot(x= 'Grade', data = seg, ax= ax)
```

Out[261…

```
<AxesSubplot:xlabel='Grade', ylabel='count'>
```

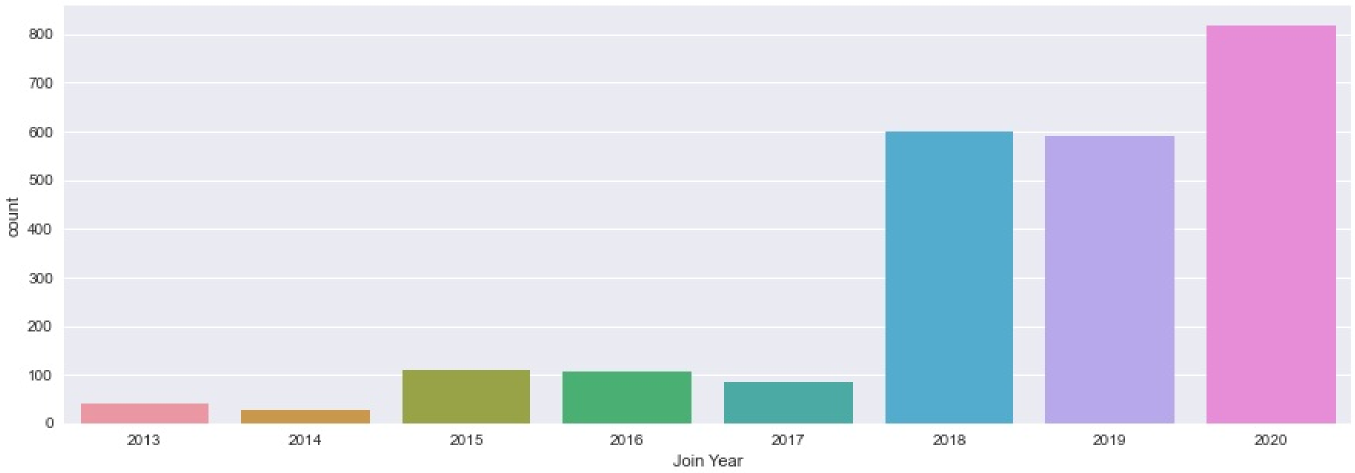


In [262…

```
fig, ax = plt.subplots(figsize=(15, 5))
sns.countplot(x= 'Join Year', data = seg, ax= ax)
```

Out[262…

<AxesSubplot:xlabel='Join Year', ylabel='count'>

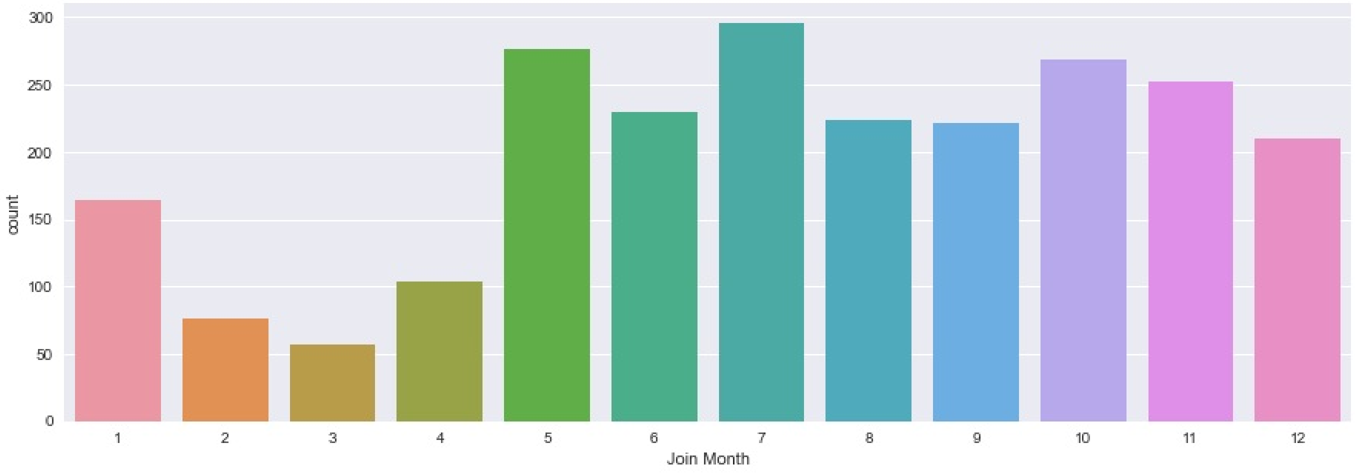


In [263…

```
fig, ax = plt.subplots(figsize=(15, 5))
sns.countplot(x= 'Join Month', data = seg, ax= ax)
```

Out[263…

<AxesSubplot:xlabel='Join Month', ylabel='count'>

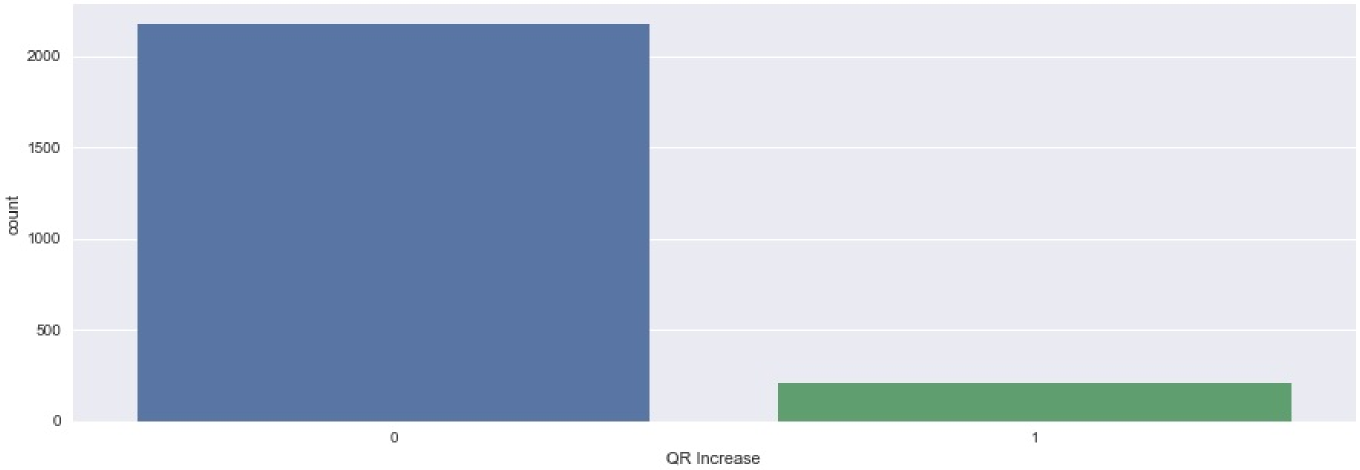


In [264…

```
fig, ax = plt.subplots(figsize=(15, 5))
sns.countplot(x= 'QR Increase', data = seg, ax= ax)
```

Out[264…

<AxesSubplot:xlabel='QR Increase', ylabel='count'>



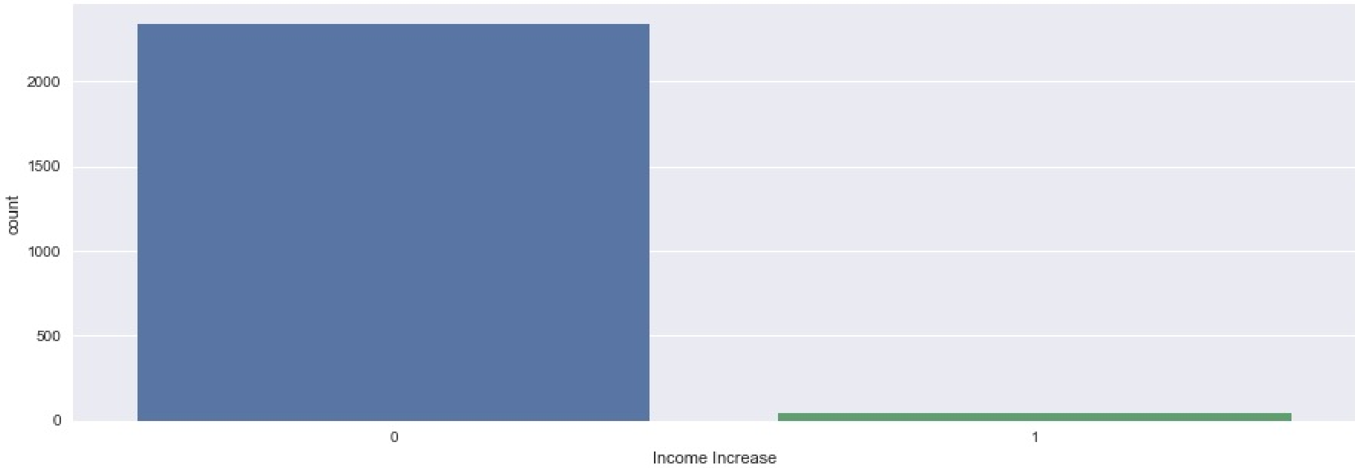
In [265…

```
fig, ax = plt.subplots(figsize=(15, 5))
```

```
fig, ax = plt.subplots(figsize=(15, 5))
sns.countplot(x= 'Income Increase', data = seg, ax= ax)
```

Out[265…

<AxesSubplot:xlabel='Income Increase', ylabel='count'>

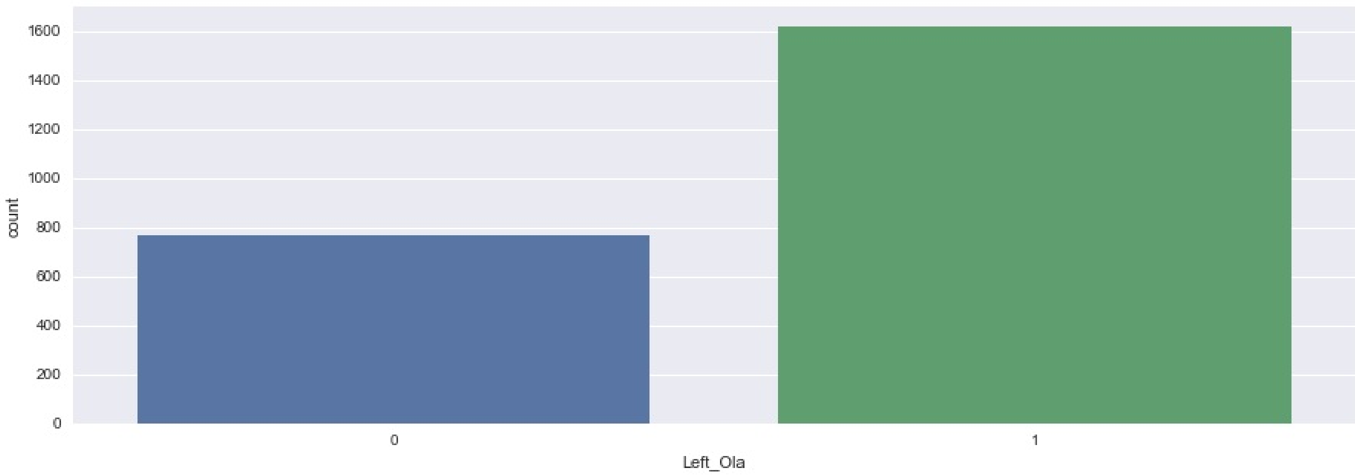


In [266…

```
fig, ax = plt.subplots(figsize=(15, 5))
sns.countplot(x= 'Left_Ola', data = seg, ax= ax)
```

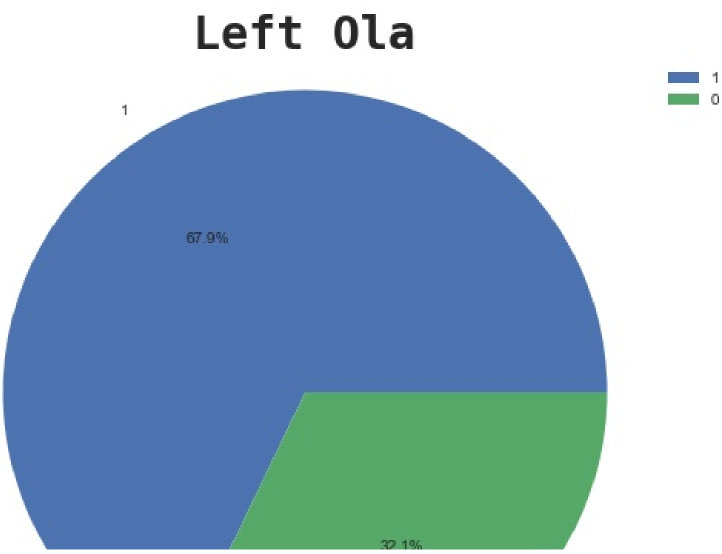
Out[266…

<AxesSubplot:xlabel='Left_Ola', ylabel='count'>



In [267…

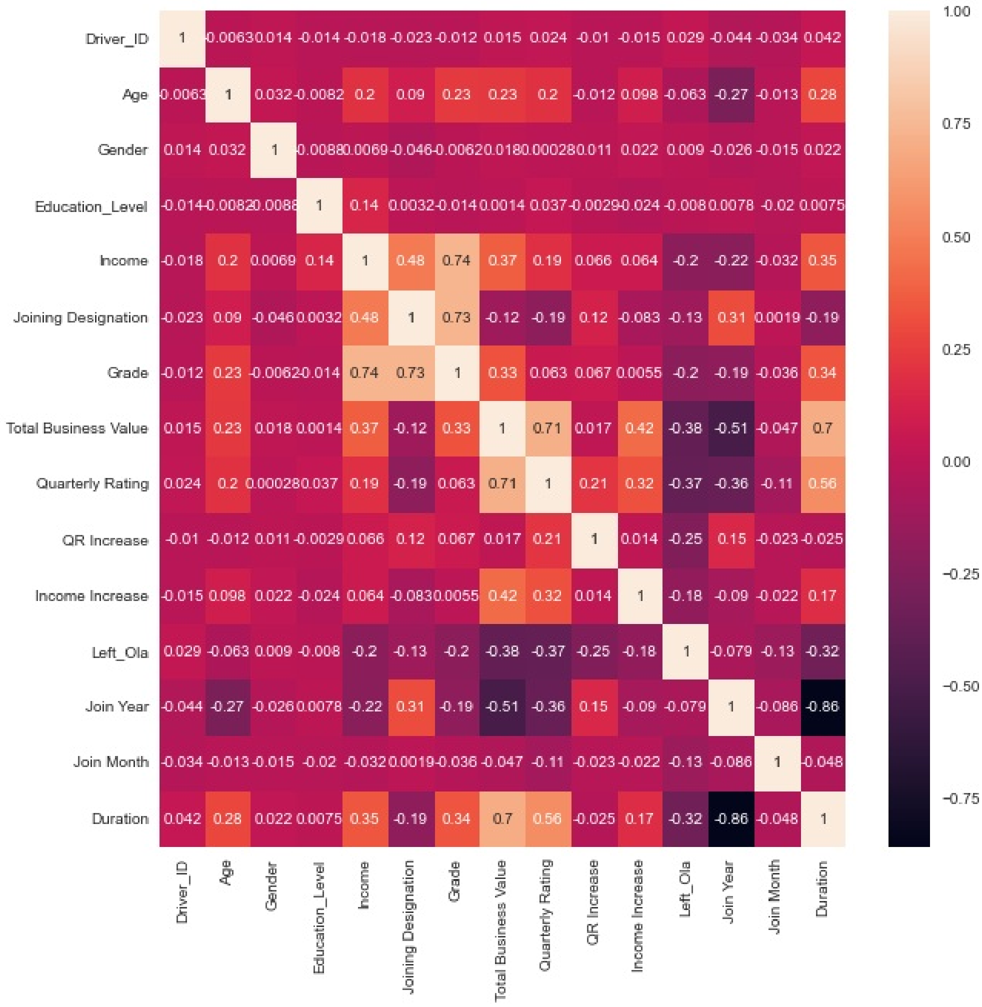
```
ssn = seg['Left_Ola'].value_counts()
plt.style.use('seaborn')
plt.figure(figsize = (10, 8))
plt.pie(ssn.values, labels = ssn.index, autopct = '%1.1f%%')
plt.title('Left Ola', fontdict = {'fontname' : 'Monospace','fontsize' : 30, 'fontweight' : 'bold'})
plt.legend()
plt.axis('equal')
plt.show()
```



In [268…

```python
#Bivariate analysis

#Heatmap

#Since variables aren't normally distributed we do not consider Pearson correlation
fig, ax = plt.subplots(figsize=(10, 10))
Var_Corr = seg.corr(method = 'pearson')
sns.heatmap(Var_Corr, xticklabels=Var_Corr.columns, yticklabels=Var_Corr.columns, annot=True, ax=ax)
```

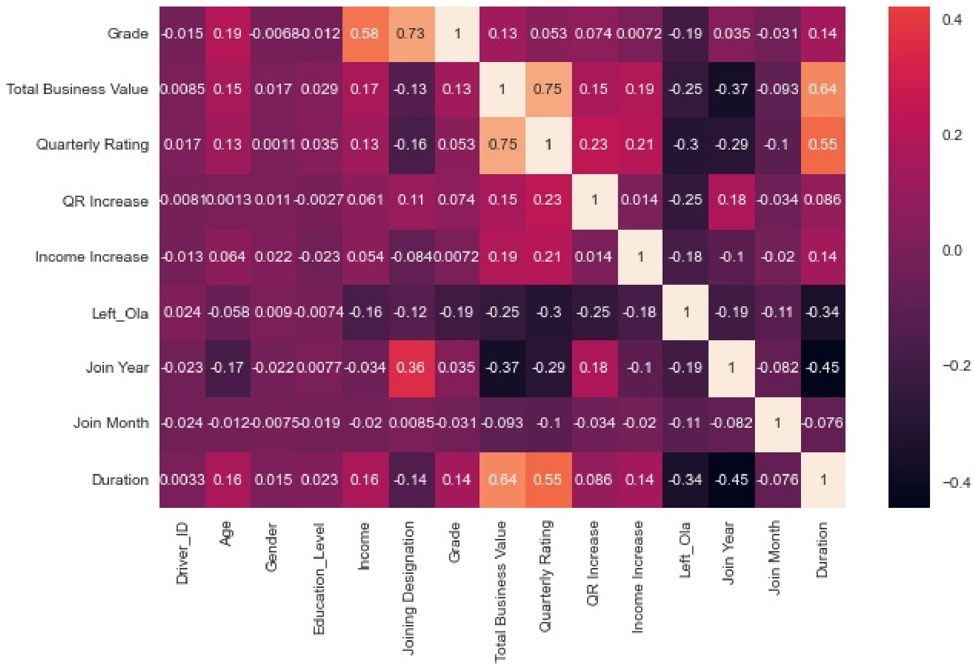
Out[268… <AxesSubplot:>



In [269…

```python
fig, ax = plt.subplots(figsize=(10, 10))
Var_Corr = seg.corr(method = 'kendall')
sns.heatmap(Var_Corr, xticklabels=Var_Corr.columns, yticklabels=Var_Corr.columns, annot=True, ax=ax)
```

Out[269… <AxesSubplot:>

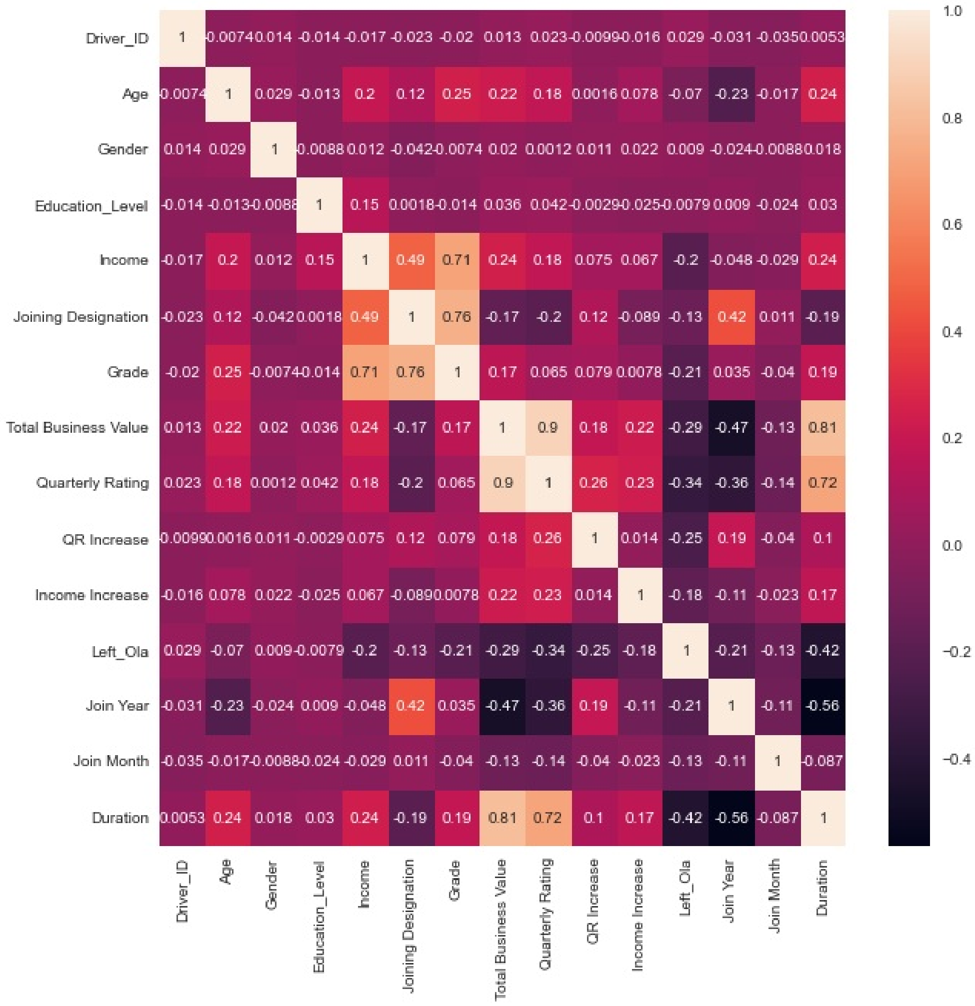


In [270…

```
fig, ax = plt.subplots(figsize=(10, 10))
Var_Corr = seg.corr(method = 'spearman')
sns.heatmap(Var_Corr, xticklabels=Var_Corr.columns, yticklabels=Var_Corr.columns, annot=True, ax=ax)
```

Out[270…

```
<AxesSubplot:>
```



In [271…

```
seg['Join Year'].value_counts()
```

Out[271…

```
2020    818
2018    599
2019    591
2015    109
```

```
2016    108
2017     86
2013     41
2014     29
Name: Join Year, dtype: int64
```

In [272…

```
seg['Join Month'].value_counts()
```

Out[272…

```
7     296
5     276
10    269
11    253
6     230
8     224
9     222
12    210
1     164
4     104
2      76
3      57
Name: Join Month, dtype: int64
```

Insights based on EDA

Univarite analysis done for continous variables :

1. Age :Most of the drivers are in the age group of 25-40
2. Earning per month :Most of the drivers earn between 25,000 to 80,000 a month.
3. Date of joining : Post 2018 Ola enrolled most of the drivers on the platform.
4. Last Wokring Date : we have in the range 2018-12-31 to 2020-12-28. Looking at the graph we can conclude that most months the number of drivers that leave the company is the at a constant rate.
5. Total business value : Has a big range. 50% of the values lie between 0 and 6.99 lacs. However some values are negative as well.
6. Gender : 0 gender has is over-represented
7. City : Almost all cities are euqally present with C26, C20 and C29 having slightly more counts than others. We cna say drivers are evenly distributed across cities
8. Education : drivers are evenly distrobuted across 0 , 1 and 2
9. Joining Designation : Mst drivers have designation 1. 50% drivers have designation either 1 or 2
10. Grade : 1,2 and 3 are the most common ones. 50% drivers of the drivers have either 1, 2 or 3
11. Mean Quarterly Rating : Most of the drivers have quartelry rating of 1.

Bivariate analysis :

1. Quarterly Rating and Total business value are very strongly correlated.
2. Quarterly Rating and duration strongly correlated.
3. Grade and Income are strongly correalated
4. Grade and joining desgination are strongly correlated.
5. Total Business value and duration are strongly correlated.

In [273…

```
seg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Driver_ID             2381 non-null   int64
 1   MMM-YY                2381 non-null   datetime64[ns]
 2   Age                   2381 non-null   float64
 3   Gender                2381 non-null   float64
 4   City                  2381 non-null   object
 5   Education_Level       2381 non-null   int64
 6   Income                2381 non-null   float64
 7   Dateofjoining         2381 non-null   datetime64[ns]
 8   LastWorkingDate       2381 non-null   datetime64[ns]
 9   Joining Designation   2381 non-null   int64
 10  Grade                 2381 non-null   int64
 11  Total Business Value  2381 non-null   int64
 12  Quarterly Rating      2381 non-null   float64
 13  QR Increase           2381 non-null   int64
 14  Income Increase       2381 non-null   int64
 15  Left_Ola              2381 non-null   int64
 16  Join Year             2381 non-null   int64
```

```
 17  Join Month            2381 non-null   int64
 18  Duration              2381 non-null   int64
dtypes: datetime64[ns](3), float64(4), int64(11), object(1)
memory usage: 353.6+ KB
```

In [274…

```python
seg.describe()
```

Out[274…

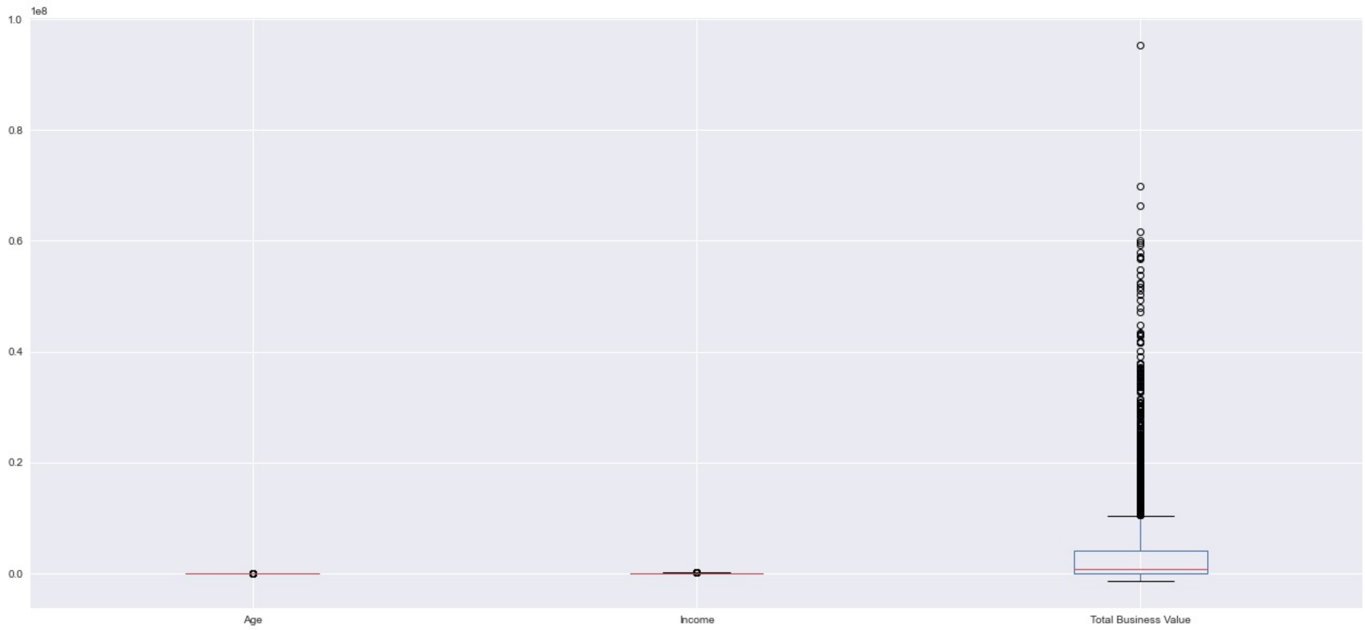
	Driver_ID	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	QR Increa
count	2381.000000	2381.000000	2381.000000	2381.00000	2381.000000	2381.000000	2381.000000	2.381000e+03	2381.000000	2381.000
mean	1397.559009	33.369298	0.410332	1.00756	59232.460484	1.820244	2.078538	4.586742e+06	1.566304	0.0873
std	806.161628	5.890567	0.491997	0.81629	28298.214012	0.841433	0.931321	9.127115e+06	0.719652	0.282
min	1.000000	21.000000	0.000000	0.00000	10747.000000	1.000000	1.000000	-1.385530e+06	1.000000	0.000
25%	695.000000	29.000000	0.000000	0.00000	39104.000000	1.000000	1.000000	0.000000e+00	1.000000	0.000
50%	1400.000000	33.000000	0.000000	1.00000	55285.000000	2.000000	2.000000	8.176800e+05	1.000000	0.000
75%	2100.000000	37.000000	1.000000	2.00000	75835.000000	2.000000	3.000000	4.173650e+06	2.000000	0.000
max	2788.000000	58.000000	1.000000	2.00000	188418.000000	5.000000	5.000000	9.533106e+07	4.000000	1.000

In [275…

```python
#Before splitting we need to do outlier treatment

#Variables we need to check for outlier detection
num_cols = ['Age', 'Income', 'Total Business Value']

seg[num_cols].boxplot(figsize=(22,10))
```

Out[275…

<AxesSubplot:>



In [276…

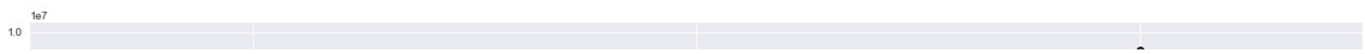
```python
Q1 = seg[num_cols].quantile(0.25)
Q3 = seg[num_cols].quantile(0.75)

IQR = Q3 - Q1

for i in num_cols:
    Q1ss = seg[i].quantile(0.25)
    Q3ss = seg[i].quantile(0.75)
    IQRss = Q3ss - Q1ss
    seg = seg[ ~ ((seg[i] < (Q1ss -  1.5 * IQRss)) | (seg[i] > (Q3ss + 1.5 * IQRss))) ]
```

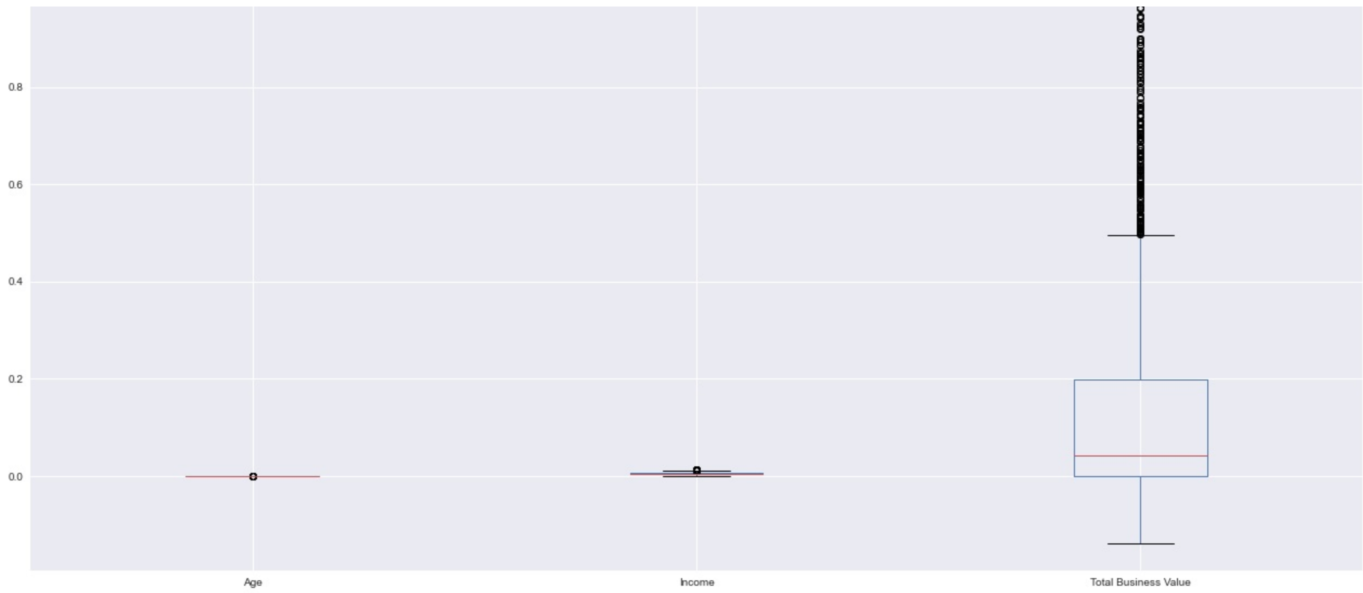
In [277…

```python
seg[num_cols].boxplot(figsize=(22,10))
```

Out[277…

<AxesSubplot:>





In [278…
```
#Join Year and Join Months are not useful and do not show any correaltion as well with other
# variables. I delete them
X = seg.drop(['Driver_ID','MMM-YY','Dateofjoining','LastWorkingDate','Left_Ola','Join Year', 'Join Month'], axis=
y = seg['Left_Ola']
```

In [279…
```
#Encoding, Outlier detection and treatment remaining

from sklearn.model_selection import train_test_split

#Train imputer on train and then apply on cv and test
X_tr_cv, X_test, y_tr_cv, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_tr_cv, y_tr_cv, test_size=0.25, random_state=42)
```

In [280…
```
seg['City'].nunique()
```

Out[280… 29

In [281…
```
#We cannot use One hot encoding on city as the number of uniq cities is 29 and we will get 29 new columns
#and this is not optimal

#Encoding of city variable
#We use target encoding on it. Fit on train data and use on train, test and validation
```

In [282…
```
from category_encoders import TargetEncoder

# Data is already trained on the tarining dataset
# This function is used to do the transformation on the training, testing and validationd dataset
def target_encode(x,var_list, encoder):

    for i in range(0, len(var_list)):
        x[var_list[i]] = encoder.transform(x[var_list[i]])
```

In [283…
```
encoder = TargetEncoder()
```

In [284…
```
encoder.fit(X_train['City'], y_train)
```

Out[284… TargetEncoder(cols=['City'])

In [285…
```
target_encode(X_train, ['City'], encoder)
target_encode(X_val, ['City'], encoder)
target_encode(X_test, ['City'], encoder)
```

In [286…
```
#Bagging
```

In [287…
```
y_train.value_counts()
```

```
y_train.value_counts()
```

Out[287…

```
1    893
0    305
Name: Left_Ola, dtype: int64
```

In [288…

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
```

In [289…

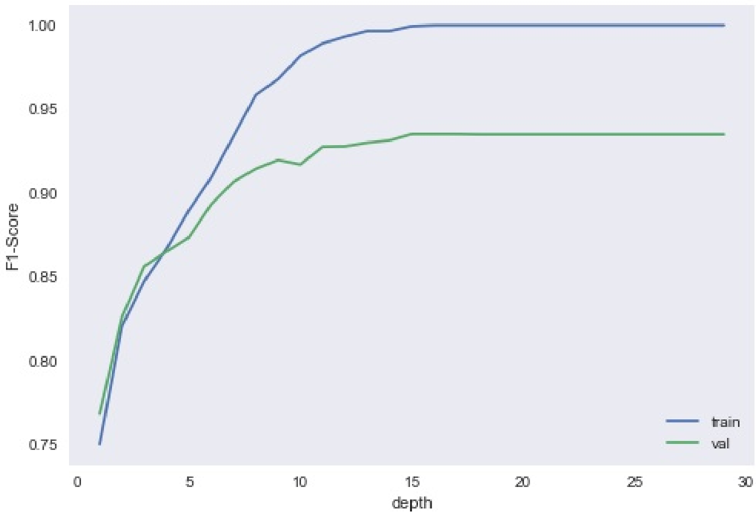
```python
train_scores = []
val_scores = []
l=1
u=30
d=1
w=0.8
num_learners=200
row_sampling_rate = 0.75

#We can try class weights as balanced : inversely proportional to frequency or also
#as 0: 0.8 and 1:0.2 . Both are same
```

In [290…

```python
for depth in np.arange(l,u,d):
    clf = RandomForestClassifier(max_depth=depth, max_samples=row_sampling_rate, n_estimators = num_learners,
                                 class_weight = 'balanced', random_state = 0)
    clf.fit(X_train, y_train)
    train_y_pred = clf.predict(X_train)
    val_y_pred = clf.predict(X_val)
    train_score = f1_score(y_train, train_y_pred)
    val_score = f1_score(y_val, val_y_pred)
    train_scores.append(train_score)
    val_scores.append(val_score)
```

In [291…

```python
import matplotlib.pyplot as plt
plt.figure()
plt.plot(list(np.arange(l,u,d)), train_scores, label="train")
plt.plot(list(np.arange(l,u,d)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("depth")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```



In [292…

```python
#depth of 7 seems best. After that the curve is flattening for both train and val data
```

In [293…

```python
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score

best_idx = np.argmax(val_scores)
l_best = 7 #l+d*best_idx
clf = RandomForestClassifier(max_depth=l_best, max_samples=row_sampling_rate, n_estimators = num_learners,
                             class_weight = 'balanced', random_state = 0)
clf.fit(X_train, y_train)

y_pred_val = clf.predict(X_val)
val_score = f1_score(y_val, y_pred_val)
```

```
#This is the accuracy
test_score = clf.score(X_test, y_test)
print("Accuracy: ", test_score)

y_pred_test = clf.predict(X_test)
test_score= f1_score(y_test, y_pred_test)
print("f1 score :", test_score)


confusion_matrix(y_test, y_pred_test)
```

```
Accuracy:  0.885
f1 score : 0.9201388888888888
```

Out[293…

```
array([[ 89,  13],
       [ 33, 265]], dtype=int64)
```

In [294…

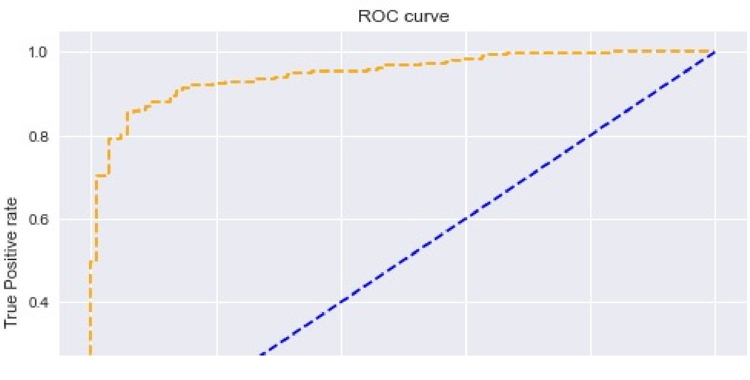
```
#Classification report
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_test))
```

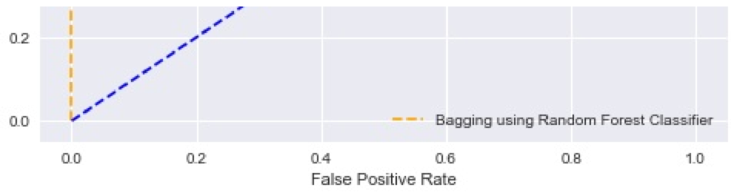
```
              precision    recall  f1-score   support

           0       0.73      0.87      0.79       102
           1       0.95      0.89      0.92       298

    accuracy                           0.89       400
   macro avg       0.84      0.88      0.86       400
weighted avg       0.90      0.89      0.89       400
```

In [302…

```
#ROC AUC Curve

from sklearn.metrics import roc_curve


pred_prob = clf.predict_proba(X_test)

fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)

from sklearn.metrics import roc_auc_score

auc_score = roc_auc_score(y_test, pred_prob[:,1])
print(auc_score)

plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Bagging using Random Forest Classifier')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```

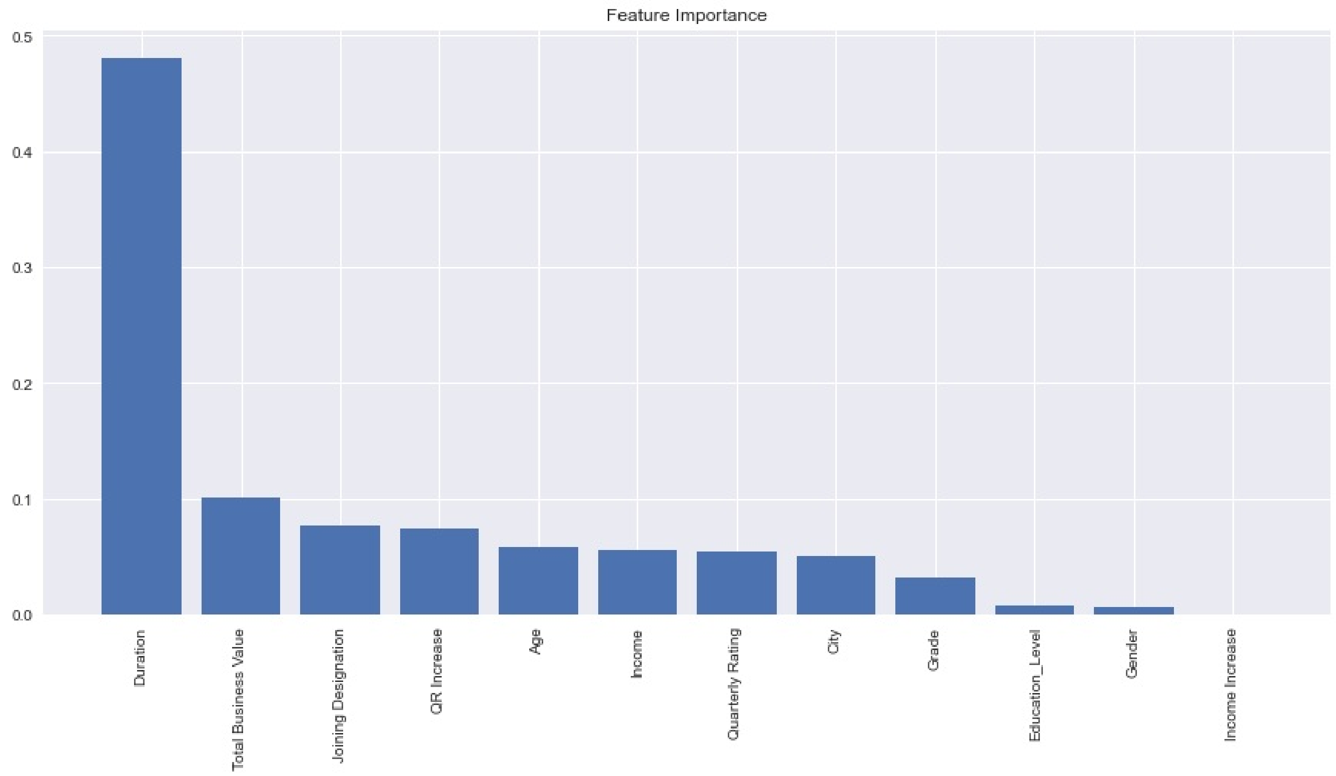
```
0.947986577181208
```





In [296…

```
# Feature importance
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1] # Sort feature importances in descending or
names = [X_train.columns[i] for i in indices] # Rearrange feature names so they mat
plt.figure(figsize=(15, 7)) # Create plot
plt.title("Feature Importance") # Create plot title
plt.bar(range(X_train.shape[1]), importances[indices]) # Add bars
plt.xticks(range(X_train.shape[1]), names, rotation=90) # Add feature names as x-ax
plt.show() # Show plot
```



In [154…

```
#Boosting algorithm

X_train1, X_test1, Y_train1, Y_test1 = train_test_split(X, y, test_size = 0.2, shuffle = True)
print(f"Sizes of the sets created are:\nTraining set:{X_train.shape[0]}\nTest set:{X_test.shape[0]}")
```

```
Sizes of the sets created are:
Training set:1422
Test set:356
```

In [155…

```
#Train encoder on training data
encoder1 = TargetEncoder()
encoder1.fit(X_train1['City'], Y_train1)
```

Out[155…

```
TargetEncoder(cols=['City'])
```

In [156…

```
# Transform both the Training and Test Data
target_encode(X_train1, ['City'], encoder1)
target_encode(X_test1, ['City'], encoder1)
```

In [177…

```
# Xgboost
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
```

```python
from sklearn.model_selection import StratifiedKFold

import datetime as dt

params = {
        'learning_rate': [0.1, 0.5, 0.8],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [5, 6, 7, 8, 9, 10]
        }
xgb = XGBClassifier(n_estimators=100, objective='binary:hinge', silent=True)
```

In [178…

```python
folds = 3

skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 1001)

random_search = RandomizedSearchCV(xgb, param_distributions=params, n_iter=10, scoring='accuracy', n_jobs=4, cv=s


start = dt.datetime.now()
random_search.fit(X_train, Y_train)
end = dt.datetime.now()
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[12:57:37] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.6.0/src/learner.cc:627:
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language 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 any such cases.
```

In [179…

```python
print('\n Best hyperparameters:')
print(random_search.best_params_)
```

```
 Best hyperparameters:
{'subsample': 0.8, 'max_depth': 6, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
```

In [180…

```python
best_xgb = XGBClassifier(n_estimators=100, objective='binary:hinge', subsample=0.8, max_depth=6, learning_rate=0.
best_xgb.fit(X_train1, Y_train1)
```

```
[12:57:37] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.6.0/src/learner.cc:627:
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language 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 any such cases.
```

Out[180…

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1.0,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.1, max_bin=256, 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=100,
              n_jobs=0, num_parallel_tree=1, objective='binary:hinge',
              predictor='auto', random_state=0, reg_alpha=0, ...)
```

In [181…

```python
print(f"Time taken for training : {end - start}\nTraining accuracy:{best_xgb.score(X_train1, Y_train1)}\nTest Acc
```

```
Time taken for training : 0:00:00.685444
Training accuracy:1.0
Test Accuracy: 0.901685393258427
```

In [185…

```python
#Classification report

Y_pred_test1 =  best_xgb.predict(X_test1)
print(classification_report(Y_test1, Y_pred_test1))
```

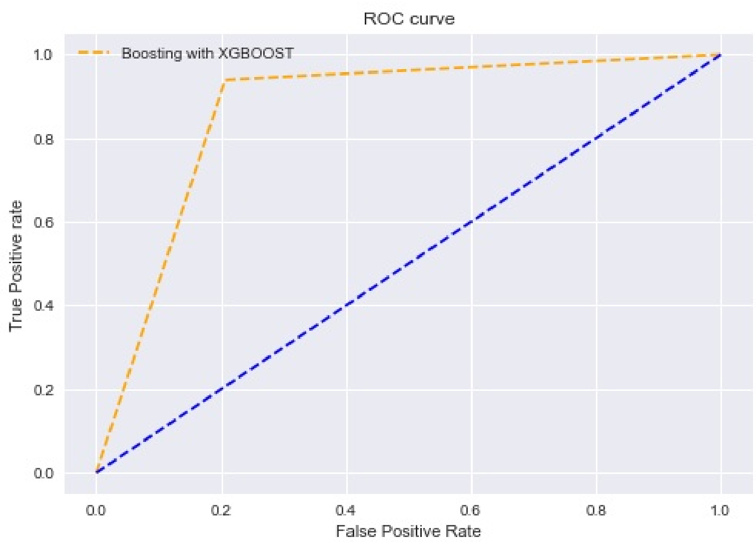
```
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

           0       0.82      0.79      0.81        92
           1       0.93      0.94      0.93       264

    accuracy                           0.90       356
   macro avg       0.87      0.87      0.87       356
weighted avg       0.90      0.90      0.90       356
```

In [301…

```python
#ROC AUC Curve

from sklearn.metrics import roc_curve


pred_prob = best_xgb.predict_proba(X_test1)

fpr1, tpr1, thresh1 = roc_curve(Y_test1, pred_prob[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(Y_test1))]
p_fpr, p_tpr, _ = roc_curve(Y_test1, random_probs, pos_label=1)

from sklearn.metrics import roc_auc_score

auc_score = roc_auc_score(Y_test1, pred_prob[:,1])
print(auc_score)

plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Boosting with XGBOOST')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```

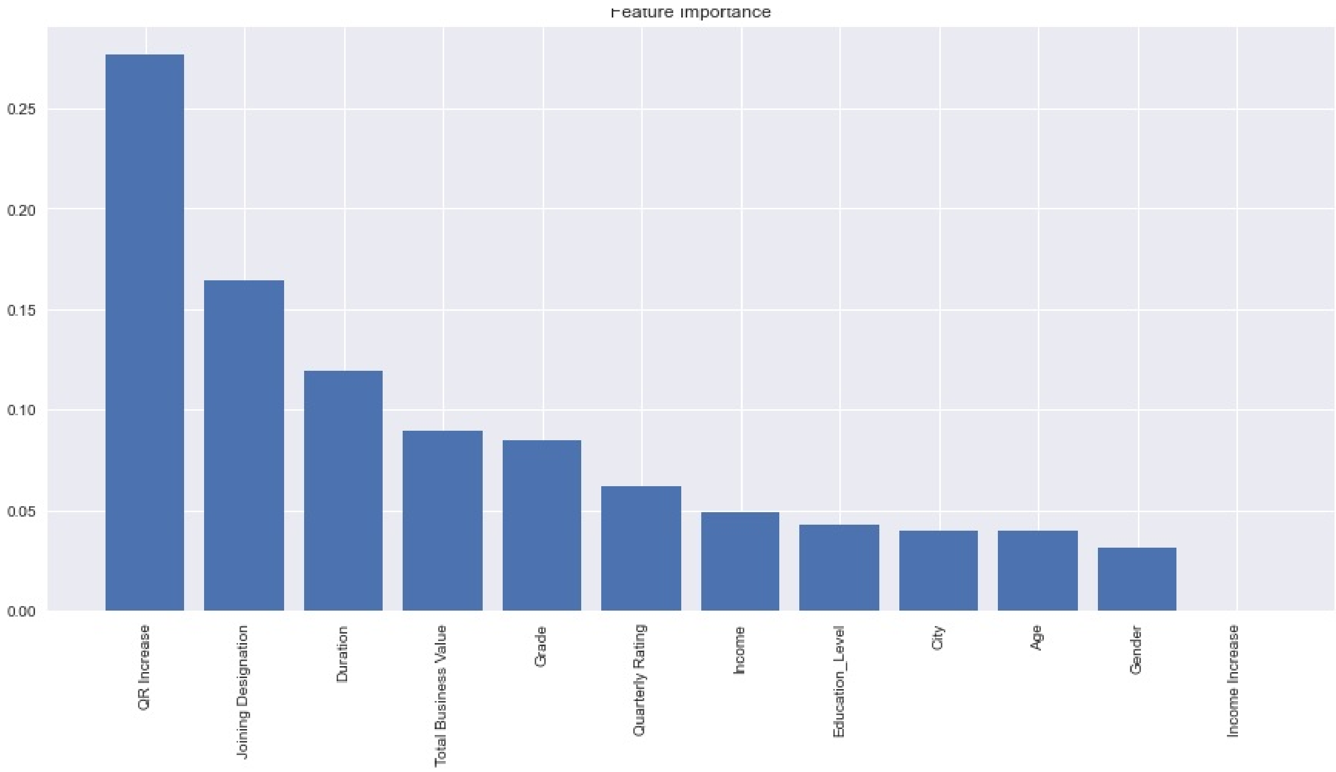
```
0.8664361001317523
```



In [182…

```python
print(best_xgb.feature_importances_)

importances = best_xgb.feature_importances_
indices = np.argsort(importances)[::-1] # Sort feature importances in descending or
names = [X_train1.columns[i] for i in indices] # Rearrange feature names so they mat
plt.figure(figsize=(15, 7)) # Create plot
plt.title("Feature Importance") # Create plot title
plt.bar(range(X_train1.shape[1]), importances[indices]) # Add bars
plt.xticks(range(X_train1.shape[1]), names, rotation=90) # Add feature names as x-ax
plt.show() # Show plot
```

```
[0.03951734 0.03174563 0.03968253 0.04295236 0.04927465 0.16447556
 0.08465444 0.08952111 0.0618978  0.27709773 0.         0.11918075]
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js