Business Case : OLA - Ensemble Learning

Submitted by : Aditya Vyas

Section 1 : Understanding the data

```
# importing required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm

# reading the dataset
df = pd.read_csv('D:/Learning/Scalar -
DataScience/Datasets/ola_driver_scaler.csv')

# getting a high level view
df.head()
```

```
   Unnamed: 0    MMM-YY  Driver_ID   Age  Gender City  Education_Level
\
0           0  01/01/19          1  28.0     0.0  C23                2

1           1  02/01/19          1  28.0     0.0  C23                2

2           2  03/01/19          1  28.0     0.0  C23                2

3           3  11/01/20          2  31.0     0.0   C7                2

4           4  12/01/20          2  31.0     0.0   C7                2


   Income Dateofjoining LastWorkingDate  Joining Designation  Grade  \
0   57387      24/12/18             NaN                   1      1
1   57387      24/12/18             NaN                   1      1
2   57387      24/12/18        03/11/19                   1      1
3   67016      11/06/20             NaN                   2      2
4   67016      11/06/20             NaN                   2      2

   Total Business Value  Quarterly Rating
0               2381060                 2
1               -665480                 2
2                     0                 2
```

```
3                     0                  1
4                     0                  1
```

```
# understanding datatype and null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Unnamed: 0             19104 non-null  int64  
 1   MMM-YY                 19104 non-null  object 
 2   Driver_ID              19104 non-null  int64  
 3   Age                    19043 non-null  float64
 4   Gender                 19052 non-null  float64
 5   City                   19104 non-null  object 
 6   Education_Level        19104 non-null  int64  
 7   Income                 19104 non-null  int64  
 8   Dateofjoining          19104 non-null  object 
 9   LastWorkingDate        1616 non-null   object 
 10  Joining Designation    19104 non-null  int64  
 11  Grade                  19104 non-null  int64  
 12  Total Business Value   19104 non-null  int64  
 13  Quarterly Rating       19104 non-null  int64  
dtypes: float64(2), int64(8), object(4)
memory usage: 2.0+ MB
```

```
df.shape
```

```
(19104, 14)
```

```
# checking NULL values
df.isnull().sum()
```

```
Unnamed: 0                  0
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
```

```
# getting deeper view to understand the features better
df.head(20)
```

```
    Unnamed: 0    MMM-YY  Driver_ID   Age  Gender City
Education_Level  \
0            0  01/01/19          1  28.0     0.0  C23
2
1            1  02/01/19          1  28.0     0.0  C23
2
2            2  03/01/19          1  28.0     0.0  C23
2
3            3  11/01/20          2  31.0     0.0   C7
2
4            4  12/01/20          2  31.0     0.0   C7
2
5            5  12/01/19          4  43.0     0.0  C13
2
6            6  01/01/20          4  43.0     0.0  C13
2
7            7  02/01/20          4  43.0     0.0  C13
2
8            8  03/01/20          4  43.0     0.0  C13
2
9            9  04/01/20          4  43.0     0.0  C13
2
10          10  01/01/19          5  29.0     0.0   C9
0
11          11  02/01/19          5  29.0     0.0   C9
0
12          12  03/01/19          5  29.0     0.0   C9
0
13          13  08/01/20          6  31.0     1.0  C11
1
14          14  09/01/20          6  31.0     1.0  C11
1
15          15  10/01/20          6  31.0     1.0  C11
1
16          16  11/01/20          6  31.0     1.0  C11
1
17          17  12/01/20          6  31.0     1.0  C11
1
18          18  09/01/20          8  34.0     0.0   C2
0
19          19  10/01/20          8  34.0     0.0   C2
0

    Income Dateofjoining LastWorkingDate  Joining Designation
Grade  \
0    57387      24/12/18             NaN                   1      1
```

1	57387	24/12/18	NaN	1	1
2	57387	24/12/18	03/11/19	1	1
3	67016	11/06/20	NaN	2	2
4	67016	11/06/20	NaN	2	2
5	65603	12/07/19	NaN	2	2
6	65603	12/07/19	NaN	2	2
7	65603	12/07/19	NaN	2	2
8	65603	12/07/19	NaN	2	2
9	65603	12/07/19	27/04/20	2	2
10	46368	01/09/19	NaN	1	1
11	46368	01/09/19	NaN	1	1
12	46368	01/09/19	03/07/19	1	1
13	78728	31/07/20	NaN	3	3
14	78728	31/07/20	NaN	3	3
15	78728	31/07/20	NaN	3	3
16	78728	31/07/20	NaN	3	3
17	78728	31/07/20	NaN	3	3
18	70656	19/09/20	NaN	3	3
19	70656	19/09/20	NaN	3	3

	Total Business Value	Quarterly Rating
0	2381060	2
1	-665480	2
2	0	2
3	0	1
4	0	1
5	0	1
6	0	1
7	0	1
8	350000	1
9	0	1
10	0	1

```
11                  120360                  1
12                       0                  1
13                       0                  1
14                       0                  1
15                       0                  2
16                 1265000                  2
17                       0                  2
18                       0                  1
19                       0                  1
```

```
# chekcing stats
df.describe(include = 'all').T
```

```
                        count unique       top  freq           mean  \
Unnamed: 0            19104.0    NaN       NaN   NaN         9551.5
MMM-YY                  19104     24  01/01/19  1022            NaN
Driver_ID             19104.0    NaN       NaN   NaN    1415.591133
Age                   19043.0    NaN       NaN   NaN      34.668435
Gender                19052.0    NaN       NaN   NaN       0.418749
City                    19104     29       C20  1008            NaN
Education_Level       19104.0    NaN       NaN   NaN       1.021671
Income                19104.0    NaN       NaN   NaN   65652.025126
Dateofjoining           19104    869  23/07/15   192            NaN
LastWorkingDate          1616    493  29/07/20    70            NaN
Joining Designation   19104.0    NaN       NaN   NaN       1.690536
Grade                 19104.0    NaN       NaN   NaN        2.25267
Total Business Value  19104.0    NaN       NaN   NaN  571662.074958
Quarterly Rating      19104.0    NaN       NaN   NaN       2.008899

                                  std        min      25%       50%
75%  \
Unnamed: 0                5514.994107        0.0  4775.75    9551.5
14327.25
MMM-YY                         NaN        NaN      NaN       NaN
NaN
Driver_ID                 810.705321        1.0    710.0    1417.0
2137.0
Age                         6.257912       21.0     30.0      34.0
39.0
Gender                      0.493367        0.0      0.0       0.0
1.0
City                           NaN        NaN      NaN       NaN
NaN
Education_Level             0.800167        0.0      0.0       1.0
2.0
Income                  30914.515344    10747.0  42383.0   60087.0
83969.0
Dateofjoining                  NaN        NaN      NaN       NaN
NaN
LastWorkingDate                NaN        NaN      NaN       NaN
```

```
NaN
Joining Designation         0.836984        1.0      1.0       1.0
2.0
Grade                       1.026512        1.0      1.0       2.0
3.0
Total Business Value  1128312.218461 -6000000.0      0.0  250000.0
699700.0
Quarterly Rating            1.009832        1.0      1.0       2.0
3.0
```

```
                              max
Unnamed: 0                19103.0
MMM-YY                        NaN
Driver_ID                  2788.0
Age                          58.0
Gender                        1.0
City                          NaN
Education_Level               2.0
Income                   188418.0
Dateofjoining                 NaN
LastWorkingDate               NaN
Joining Designation           5.0
Grade                         5.0
Total Business Value  33747720.0
Quarterly Rating              4.0
```

Observation Set 1 :

1. The shape of the dataset is 19104 rows and 14 columns
2. The first columns is unnamed and appears to be index data, we will drop it before getting into the next section.
3. There are a few Null values, which we will look at in deatil later.
4. The NULL values in LastWorkingDay is alarmingly high, it appears that this is because for active drivers this value logically will be NULL. We will use this later to create out Target column.
5. There are a few 'date' columns, will have dig deeper to extract meaningful information from these later.
6. There are multiple rows for each Driver_ID, which means we can aggregate data to arrive at a more meaningful table or data format.
7. There are a few columns (Gender, Education_Level etc) which currently are in Int/Float, but should be in categorical form. We will keep this in mind during pre-processing.
8. There seems to be no readily available Target column, we will create that using the given data

Section 2 : Data Preparation

We understood in the previous section that the data has multiple enteries for each driver and there are several rows where the LWD is mentioned. Meaning for those drivers who have not quit the LWD values are NULL. We are going to group the dataset on Driver_ID while choosing the first and last values for each column accordingly. We will also take both first and last values for a few columns becuase we would want to see if there is any change in those over the tenure of the driver.

```
# Task : dropping the first column
df_1 = df.iloc[:,1:]

# defining keys to aggregate data at Drver_ID level

# since there are a few colums () where we are interested to know
about the change in the value,
# we will define 2 keys to capture first anf last values

dri_agg_key_first = {
    'MMM-YY' : 'count',
    'Gender' : 'first',
    'City' : 'first',
    'Education_Level' : 'first',
    'Income' : 'first',
    'Dateofjoining' :'first',
    'Joining Designation' : 'first',
    'Grade' : 'first',
    'Total Business Value' : 'sum',
    'Quarterly Rating' : 'first'}

dri_agg_key_last = {
    'Age' : 'last',
    'Income' : 'last',
    'LastWorkingDate' : 'last',
    'Grade' : 'last',
    'Quarterly Rating' : 'last'}

# getting grouped data with first key in a temporary dataframe
temp_df_1 =
df_1.groupby('Driver_ID').agg(dri_agg_key_first).reset_index().rename(
columns = {'MMM-YY' : 'No of times reported',

'Income' : 'Income_first',

'Grade' : 'Grade_first',

'Quarterly Rating' : 'QR_first'})
```

```
# getting grouped data with second key in a temporary dataframe
temp_df_2 =
df_1.groupby('Driver_ID').agg(dri_agg_key_last).reset_index().rename(c
olumns = {'Income' : 'Income_last',

'Grade' : 'Grade_last',

'Quarterly Rating' : 'QR_last'})

# concatinating both the temporary dataframes
df_2 = pd.concat([temp_df_1, temp_df_2.drop('Driver_ID', axis = 1)],
axis = 1)

# creating additional columns to get if there is any difference in
select columns
df_2['Grade_diff'] = df_2['Grade_last'] - df_2['Grade_first']
df_2['QR_diff'] = df_2['QR_last'] - df_2['QR_first']
df_2['Income_diff'] = df_2['Income_last'] - df_2['Income_first']

# defining a function to map values into a new feature
def map_values(x):
    if x < 0:
        return 'Negative'
    if x > 0:
        return 'Positive'
    else :
        return 'No Change'

# creating new columns to capture change (positive or negative) in
Grade, QR and Income
df_2['Grade_change'] = df_2['Grade_diff'].apply(map_values)
df_2['QR_change'] = df_2['QR_diff'].apply(map_values)
df_2['Income_change'] = df_2['Income_diff'].apply(map_values)

# converting date columns in the required format
df_2['DoJ'] = pd.to_datetime(df_2['Dateofjoining'], format =
'%d/%m/%y')
df_2['LWD'] = pd.to_datetime(df_2['LastWorkingDate'], format =
'%d/%m/%y')

# defining function to create Target column values
def map_target(x):
    if x.isnull():
        return 1
    else :
        return 0

# creating the Target column
df_2['Target'] = df_2['LastWorkingDate'].isnull().astype(int)
```

```
# extracting further information form the datetime columns
df_2['DoJ_dayname'] = df_2['DoJ'].dt.day_name()
df_2['DoJ_day'] = df_2['DoJ'].dt.day
df_2['DoJ_month'] = df_2['DoJ'].dt.month
df_2['DoJ_year'] = df_2['DoJ'].dt.year
#df_2['DoJ_week'] = df_2['DoJ'].dt.isocalendar().week

df_2['LWD_dayname'] = df_2['LWD'].dt.day_name()
df_2['LWD_day'] = df_2['LWD'].dt.day
df_2['LWD_month'] = df_2['LWD'].dt.month
df_2['LWD_year'] = df_2['LWD'].dt.year
#df_2['LWD_week'] = df_2['LWD'].dt.isocalendar().week

# dropping the original datetome columns
df_2.drop(columns = ['Dateofjoining','LastWorkingDate','DoJ','LWD'],
axis = 1, inplace = True)

# for all the NULL values in LastWorkingDay features, we will insert
'Stil Active'
df_2 = df_2.fillna('Still Active')

# as a result of the above command, we have a few features with mix
values (int and str). This must be treated.
col_with_mix_values = ['LWD_day','LWD_month','LWD_year']

for col in col_with_mix_values:
    df_2[col] = df_2[col].apply(lambda x : '{}'.format(x))
```

Encoding criteria to be implemented in the later section

Columns for ordinal encoding:

1. No of time reported
2. Joining Designation
3. Grade
4. Quarterly Rating
5. dayname

'City' has 29 unique values. Ordinal encoding will not make sense. One-hot encoding will result in many new columns. Frequency encoding can be considered. We will finally try to compare the performance after one-hot encoding and frequency encoding

For Grade, QR and Inc we will use One Hot encoding

Section 3 : EDA

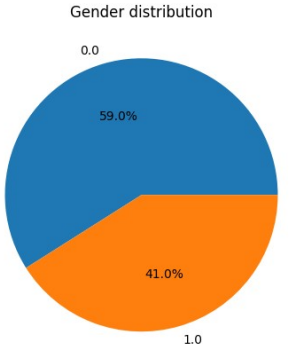
Now that we have the data in the desired format with all necessary features created, we will move ahead with visualizing and analysing the patterns the data holds.

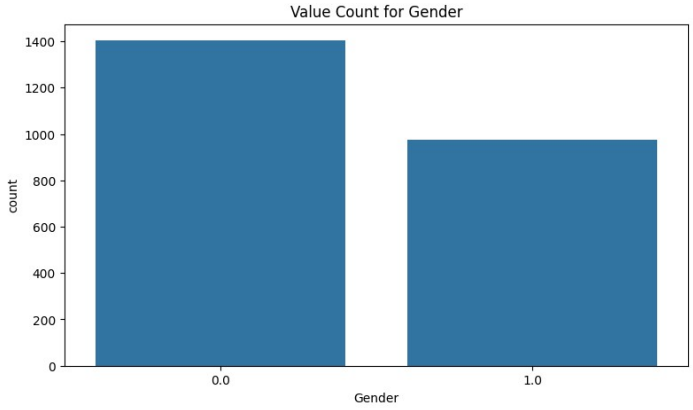
```
# exploring feature 1 - Driver ID
df_2['Driver_ID'].nunique()
```

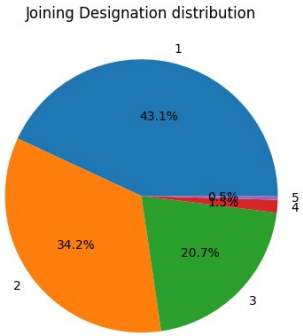
```
2381
```

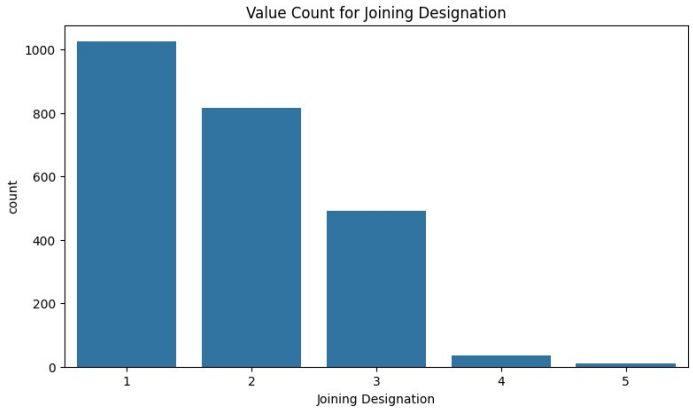
```
# defining a function for analyzing all categorical/object type
features
def plot_1(c):
    fig, axs = plt.subplots(1,2,figsize = (20,5))
    axs[0].pie(df_2[c].value_counts(), labels =
df_2[c].value_counts().index, autopct = '%1.1f%%')
    axs[0].set_title('{} distribution'.format(c))

    sns.countplot(data = df_2, x = c)
    axs[1].set_title('Value Count for {}'.format(c))
    # Value counts
    #value_counts = df_2[c].value_counts()
    #value_counts_percentage = value_counts / value_counts.sum() * 100
    #axs[1].bar(value_counts_percentage.index,
value_counts_percentage)
    #axs[1].set_xticks(value_counts_percentage.index)
    #axs[1].set_xticklabels(value_counts_percentage.index)
    #axs[1].set_title('Value Count Percentage')
    #for i, v in enumerate(value_counts_percentage):
    #    axs[1].text(i, v + 1, f"{v:.2f}%", ha='center')



# defining a function to check distribution of numerical features
def check_distribution(data):
    fig = plt.figure(figsize = [15,10])

    gs = fig.add_gridspec(2,2)
    ax1 = fig.add_subplot(gs[0, 0])
    ax2 = fig.add_subplot(gs[0, 1])
    ax3 = fig.add_subplot(gs[1, :])


    sns.boxplot(x = data.iloc[:,0], ax = ax3)
    sns.histplot(data.iloc[:,0], ax = ax2, kde = True)
    sm.qqplot(data.iloc[:,0], ax = ax1, line = '45', fit = True)

    ax1.title.set_text('qqplot for {}'.format(data.columns[0]))
    ax2.title.set_text('histplot distribution for
{}'.format(data.columns[0]))
    ax3.title.set_text('Boxplot for {}'.format(data.columns[0]))

    sns.despine()
    fig.tight_layout()
    plt.show()
```

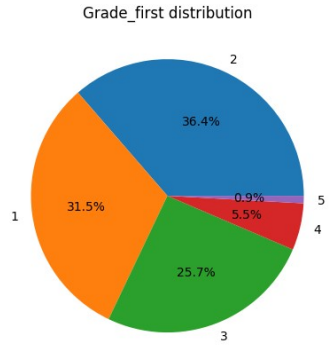
```
    return

# creating a list of all categorical/object type
cat_col = ['Gender', 'Joining
Designation','Grade_first','Education_Level','City','Grade_change','QR
_change', 'Income_change', 'DoJ_dayname', 'LWD_dayname','LWD_month']

num_col = ['No of times reported', 'Income_first', 'Total Business
Value']

#running the earlier defined function inside the FOR loop for quick
output
for c in cat_col:
    plot_1(c)
```

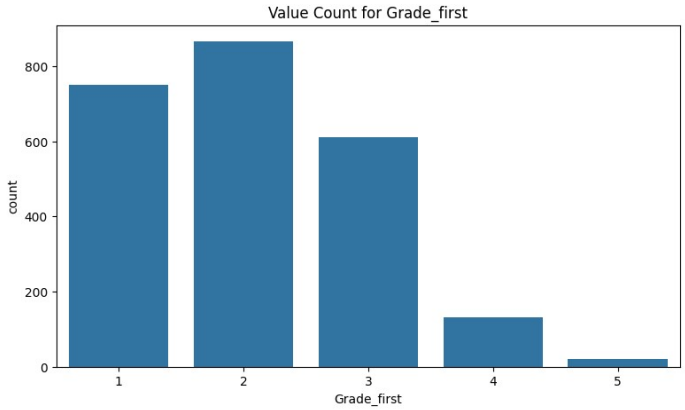


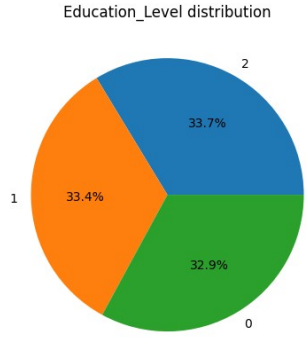


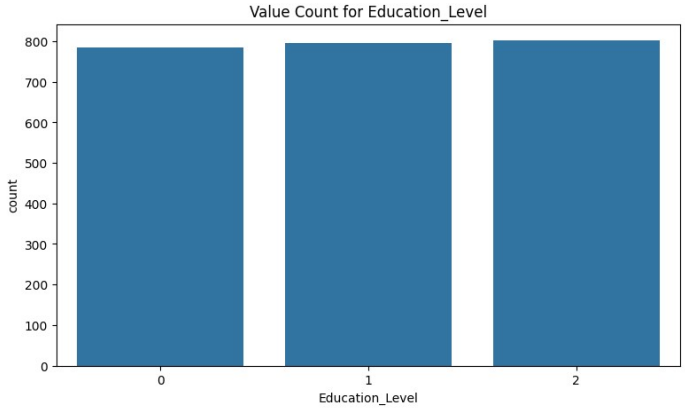


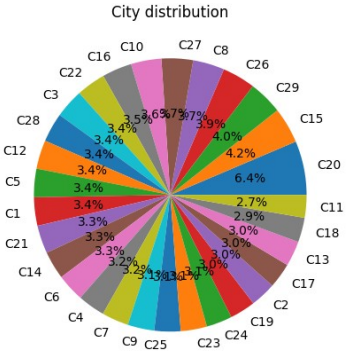


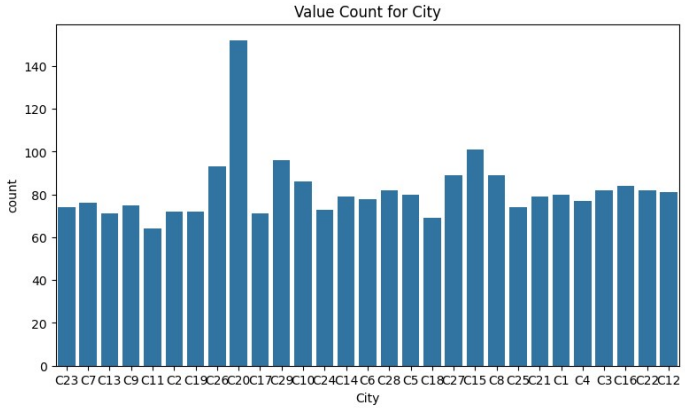

Grade_first distribution
2
36.4%
1
31.5%
0.9%
5
5.5%
4
25.7%
3

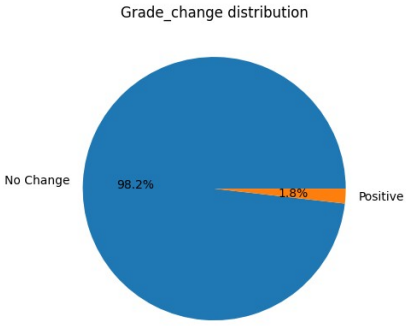

Value Count for Grade_first
800
600
400
200
0
count
1
2
3
4
5
Grade_first

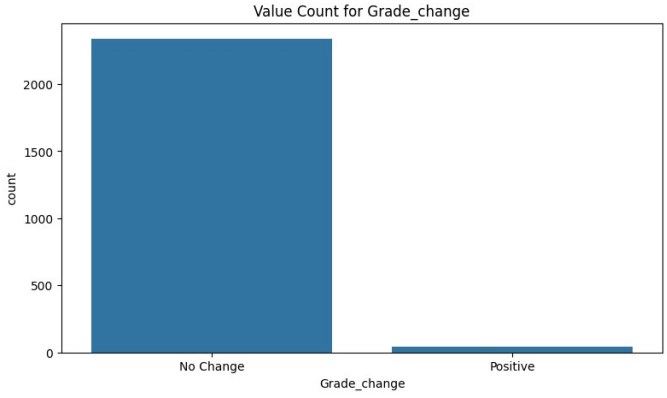

Education_Level distribution
2
33.7%
1
33.4%
32.9%
0

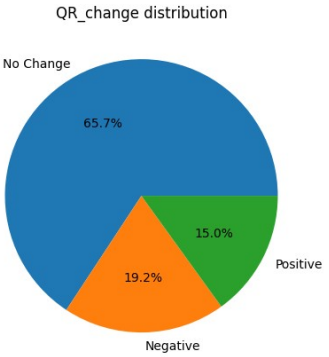

Value Count for Education_Level
800
700
600
500
400
300
200
100
0
count
0
1
2
Education_Level

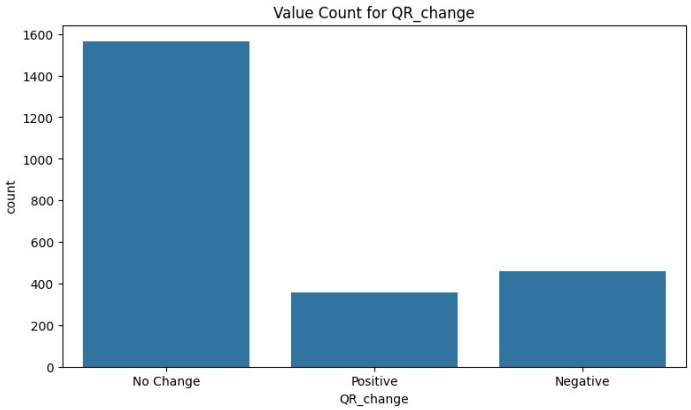

City distribution
C16
C10
C27
C8
C22
C26
C3
C29
C28
C15
C12
4.2%
C20
C5
6.4%
C1
2.7%
C11
C21
2.9%
C18
C14
3.0%
C13
C6
C17
C4
C2
C7
C9
C25
C23
C24
C19

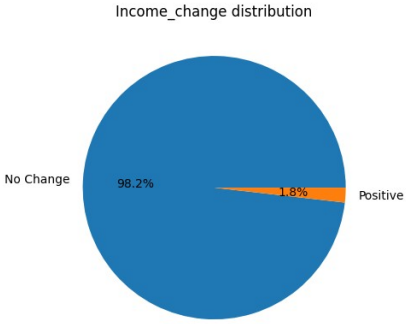

Value Count for City
140
120
100
80
60
40
20
0
count
C23 C7 C13 C9 C11 C2 C19 C26 C20 C17 C29 C10 C24 C14 C6 C28 C5 C18 C27 C15 C8 C25 C21 C1 C4 C3 C16 C22 C12
City

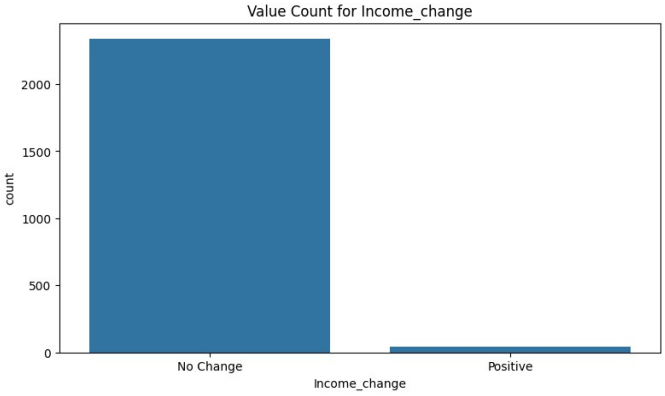

Grade_change distribution
No Change
98.2%
1.8%
Positive

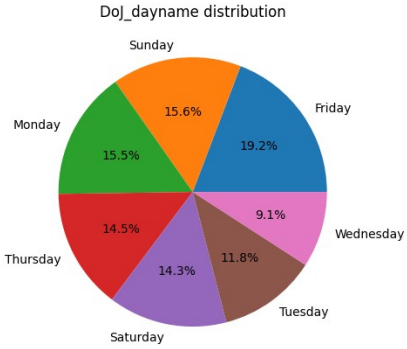

Value Count for Grade_change
2000
1500
1000
500
0
count
No Change
Positive
Grade_change

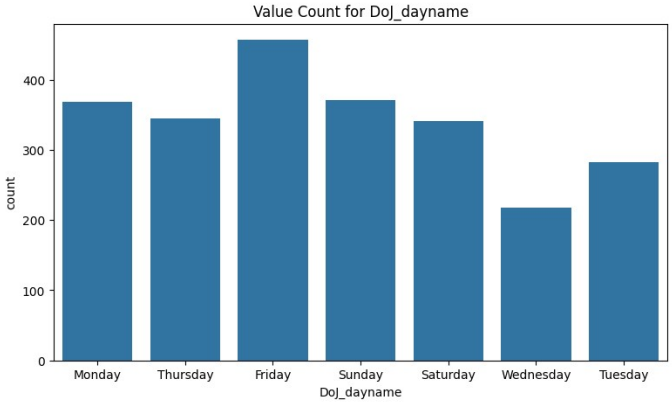

QR_change distribution
No Change
65.7%
15.0%
Positive
19.2%
Negative

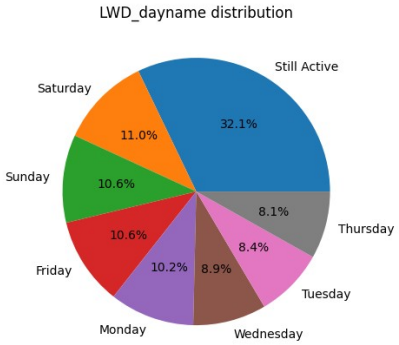

Value Count for QR_change
count
1600
1400
1200
1000
800
600
400
200
0
No Change
Positive
Negative
QR_change

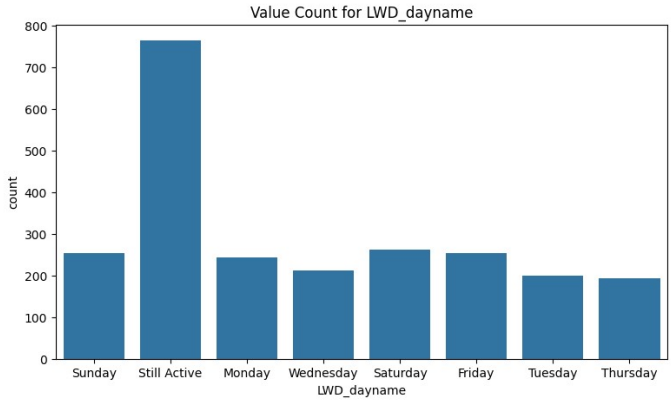

Income_change distribution
No Change
98.2%
1.8%
Positive

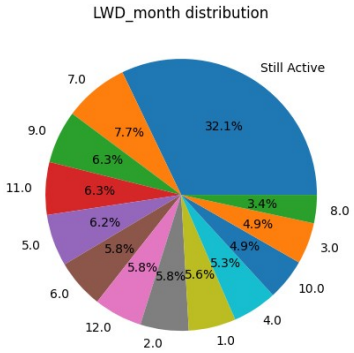

Value Count for Income_change
count
2000
1500
1000
500
0
No Change
Positive
Income_change

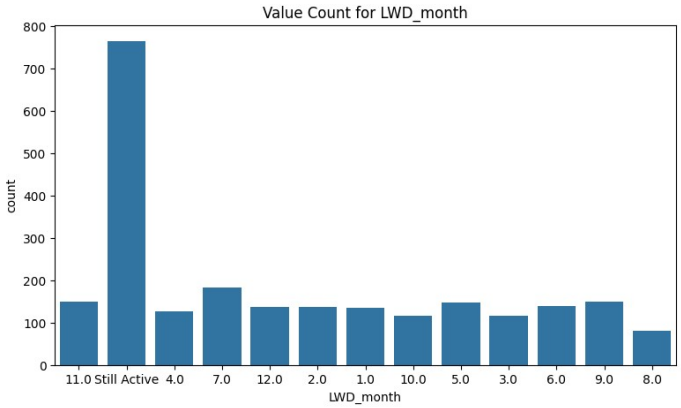

DoJ_dayname distribution
Sunday
15.6%
Friday
19.2%
Monday
15.5%
9.1%
Wednesday
14.5%
Thursday
14.3%
11.8%
Tuesday
Saturday


Value Count for DoJ_dayname
count
400
300
200
100
0
Monday
Thursday
Friday
Sunday
Saturday
Wednesday
Tuesday
DoJ_dayname

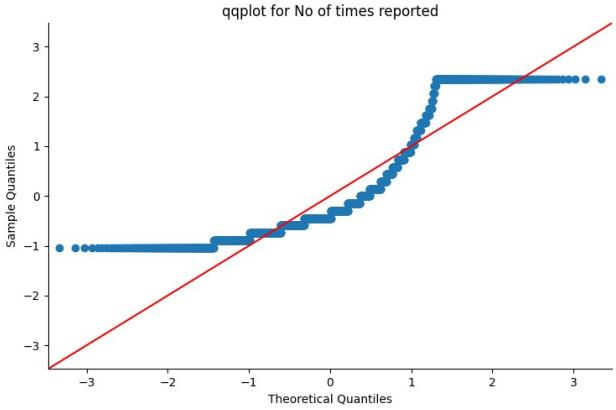

LWD_dayname distribution
Still Active
Saturday
32.1%
11.0%
Sunday
10.6%
8.1%
Thursday
10.6%
8.4%
Friday
10.2%
8.9%
Tuesday
Monday
Wednesday

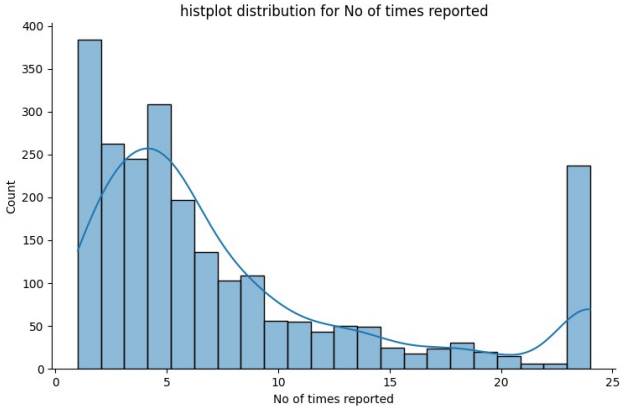

Value Count for LWD_dayname
count
800
700
600
500
400
300
200
100
0
Sunday
Still Active
Monday
Wednesday
Saturday
Friday
Tuesday
Thursday
LWD_dayname

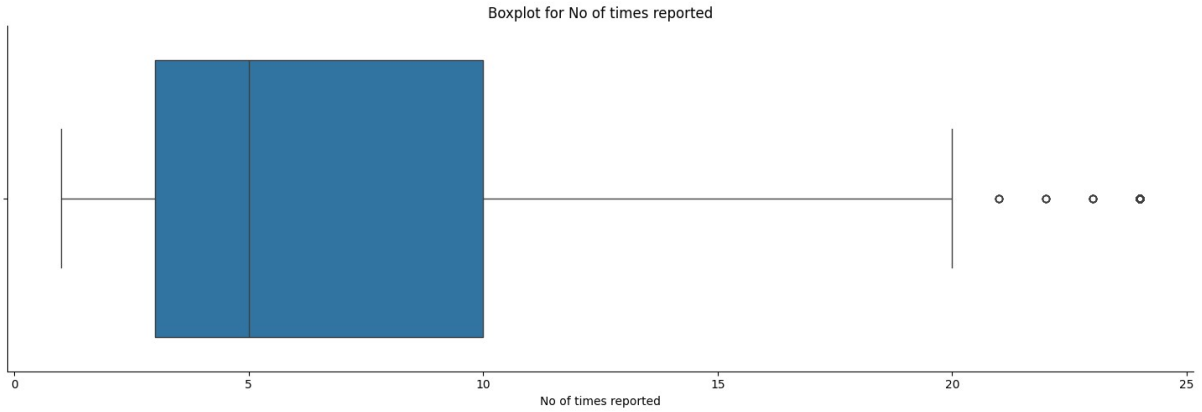


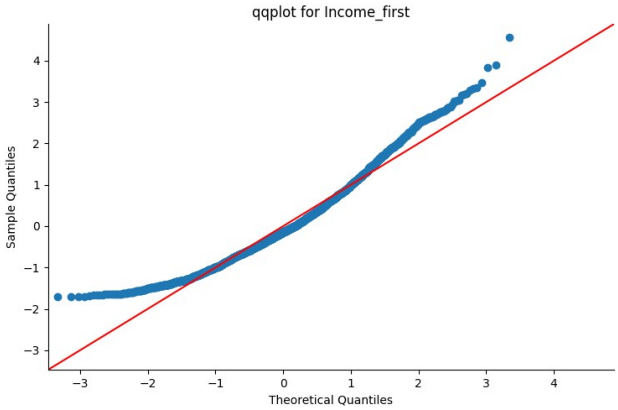


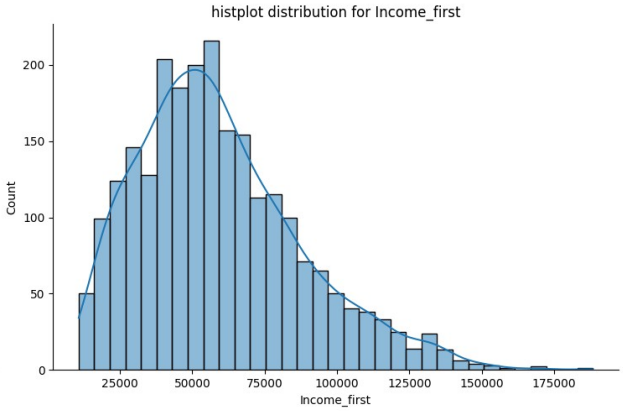

```
for c in num_col:
    check_distribution(df_2[[c]])
```

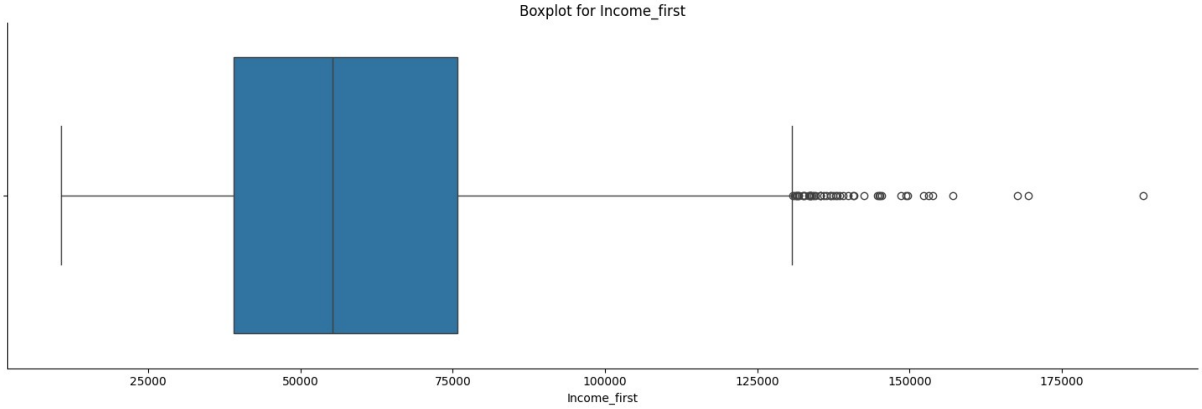


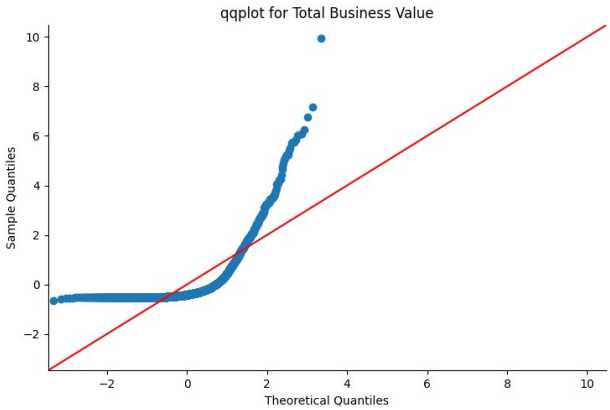


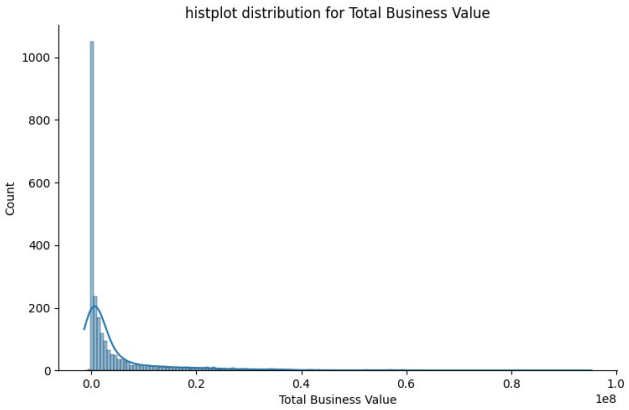


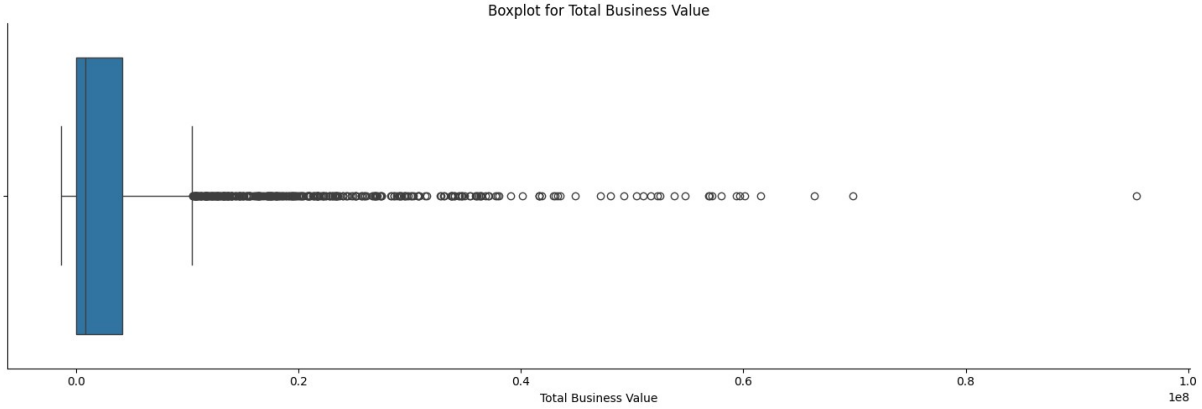

qqplot for Income_first
Sample Quantiles
Theoretical Quantiles


histplot distribution for Income_first
Count
Income_first

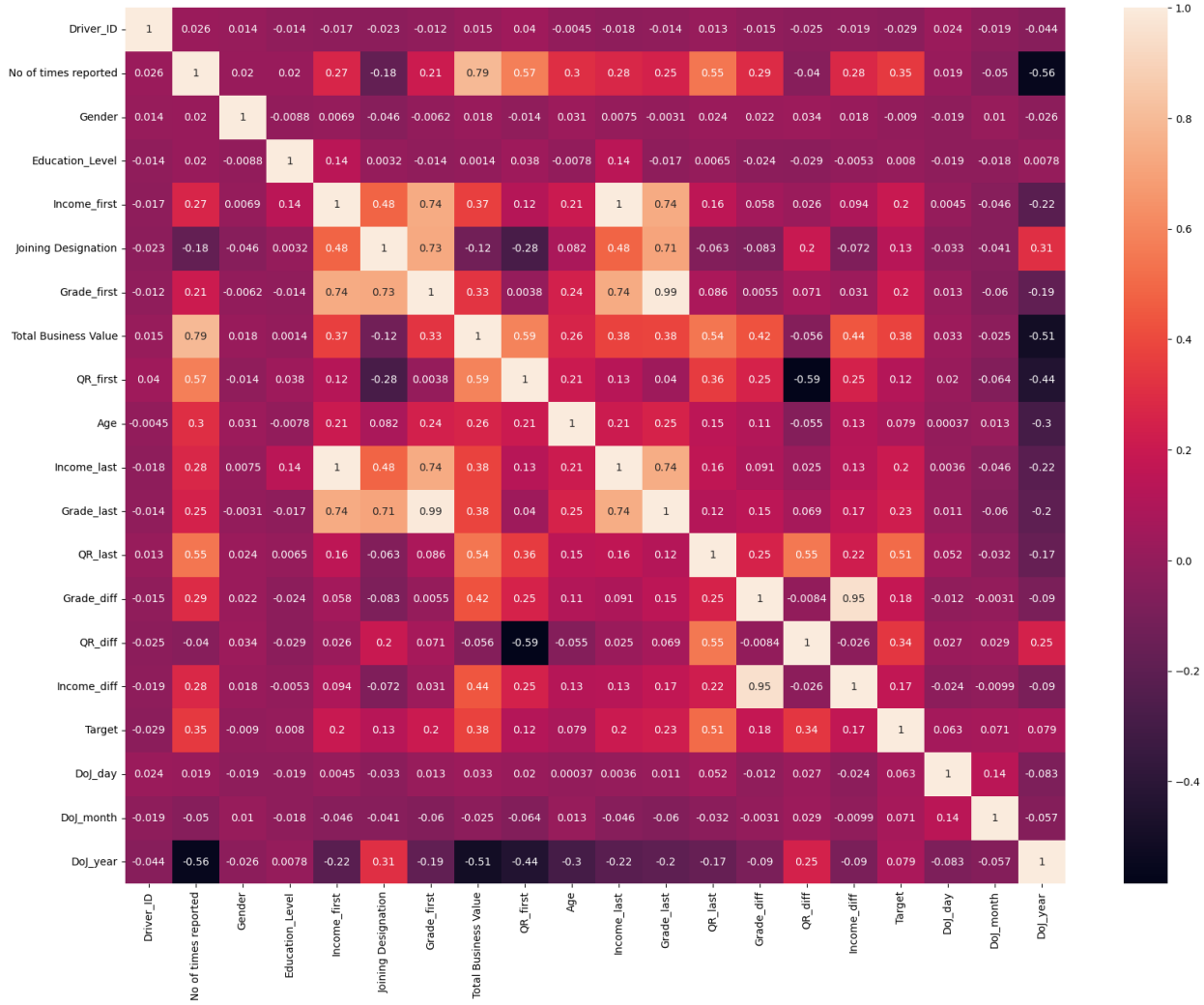

Boxplot for Income_first
Income_first


qqplot for Total Business Value
Sample Quantiles
Theoretical Quantiles


histplot distribution for Total Business Value
Count
Total Business Value
1e8

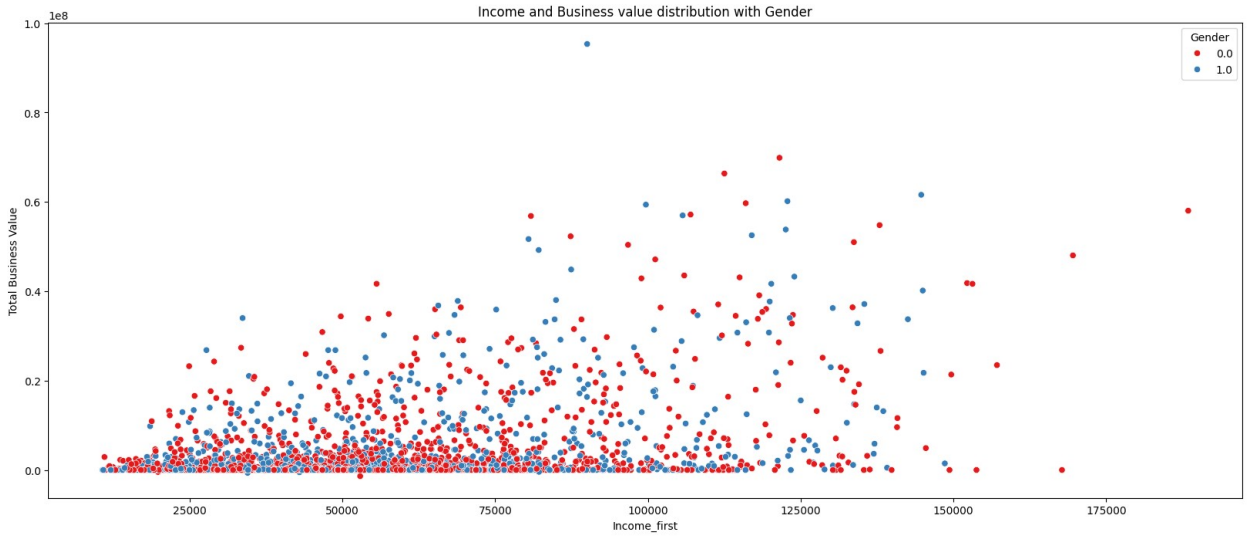

Boxplot for Total Business Value
Total Business Value
1e8

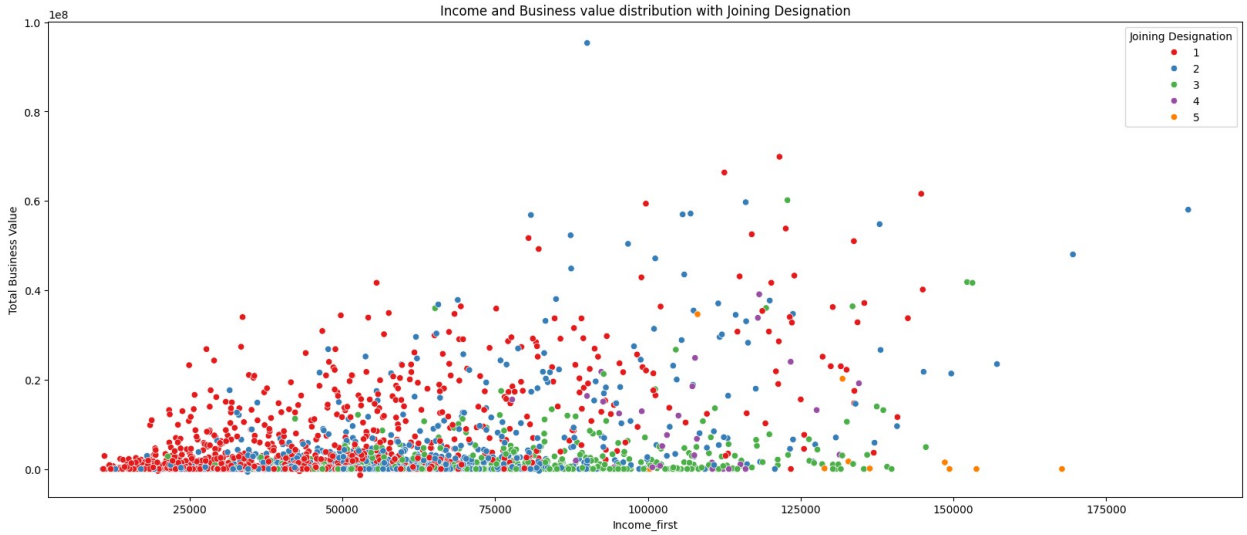
```
plt.figure(figsize = (20,15))
sns.heatmap(df_2.drop(columns = ['City','Grade_change','QR_change',
'Income_change',
                         'DoJ_dayname',
'LWD_dayname','LWD_month','LWD_day','LWD_year'],axis = 1).corr(),
annot = True)
plt.show()
```

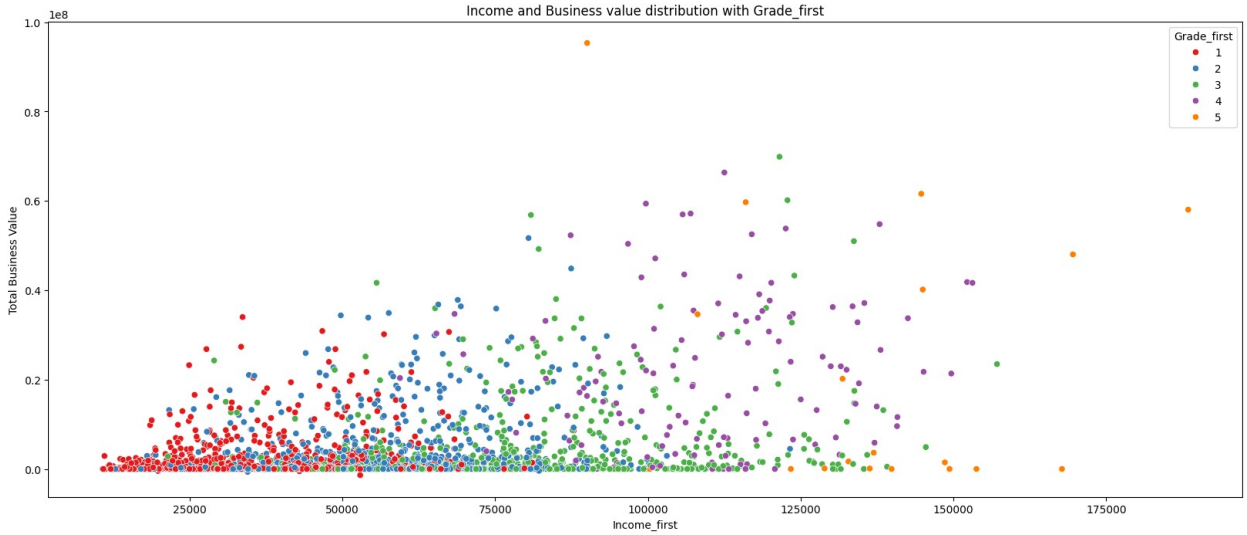


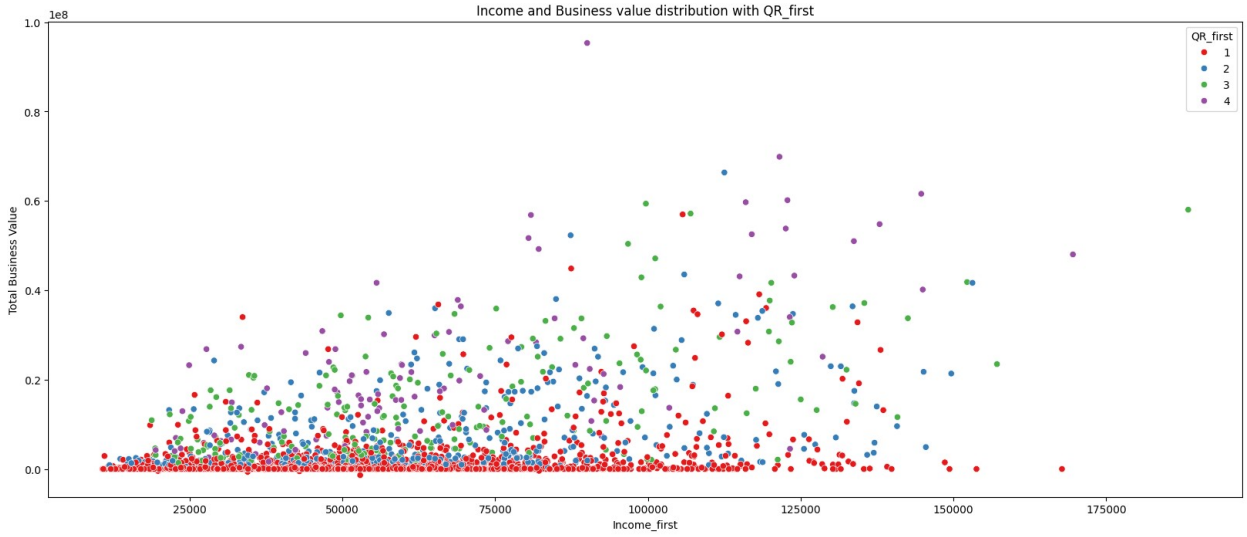

```
# creating alist of columns against which we want a scatter plot
between income and total business value
col_list_2 = ['Gender', 'Education_Level', 'Joining
Designation','Grade_first','QR_first',
              'Grade_change','QR_change','Income_change',

'DoJ_dayname','DoJ_month','DoJ_year','LWD_dayname','LWD_month','LWD_ye
ar',
             'Target']
```

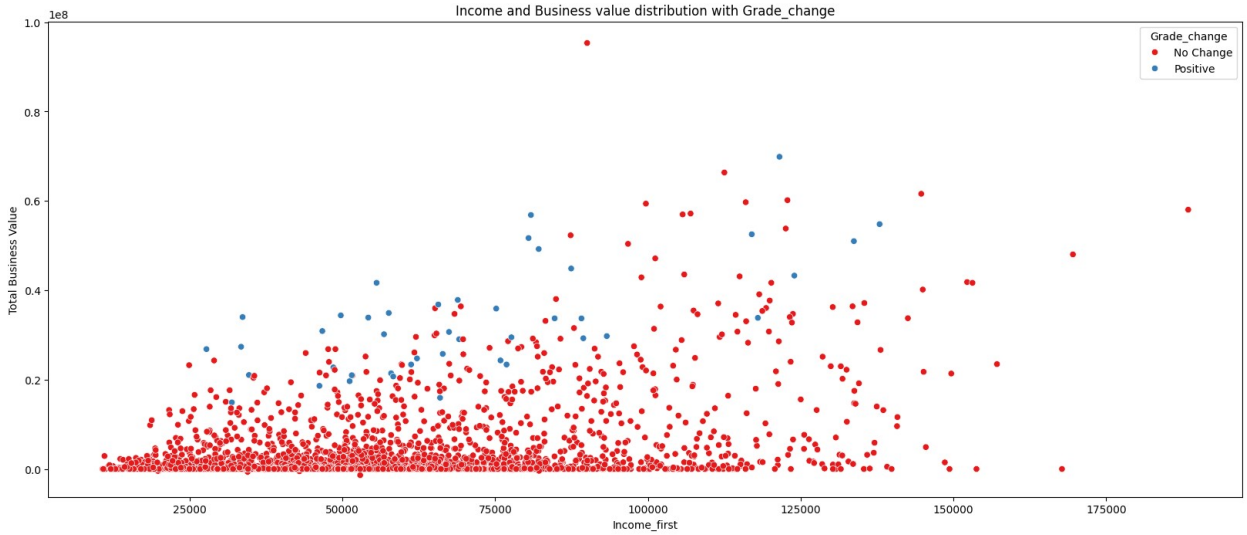
```
for c in (col_list_2):
    plt.figure(figsize = (20,8))
    sns.scatterplot(data = df_2, x = 'Income_first', y = 'Total
Business Value', hue = c, palette = 'Set1')
    plt.title('Income and Business value distribution with
{}'.format(c))
plt.show()
```

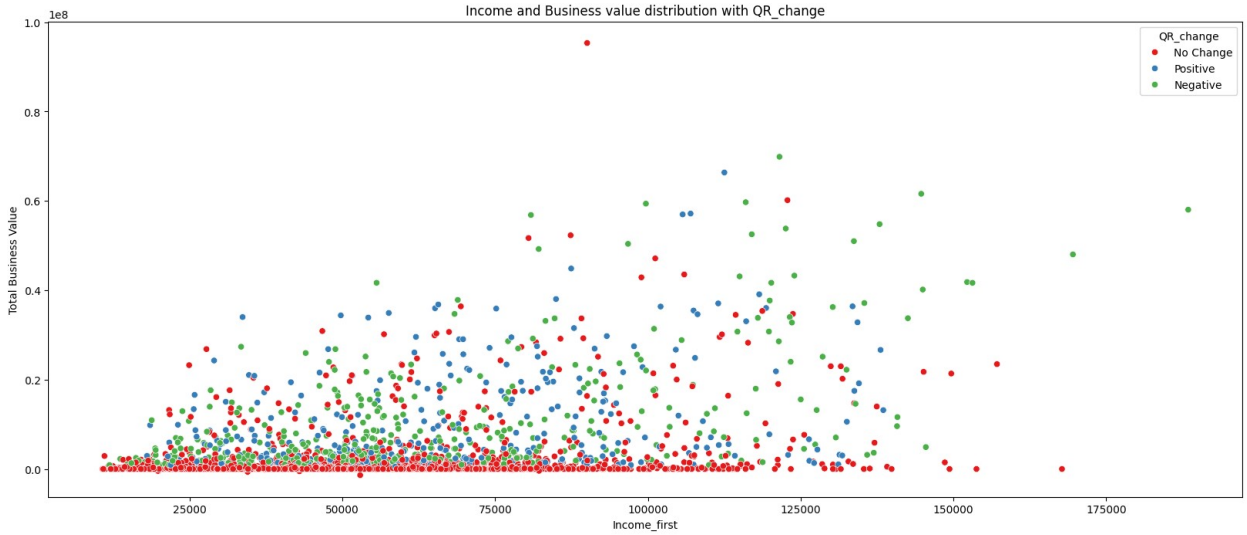


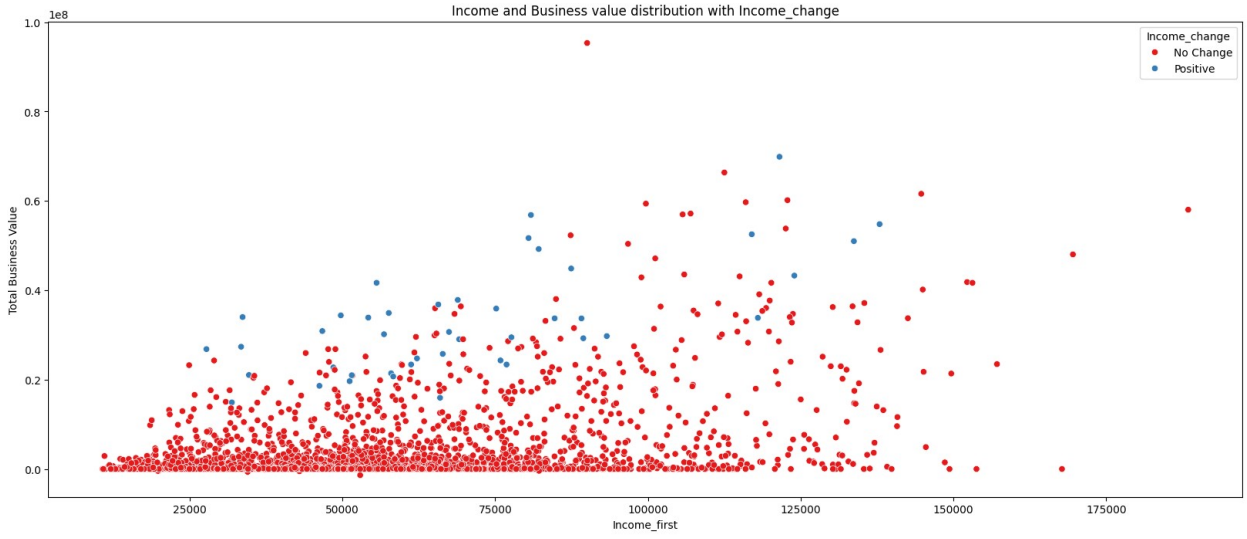

Income and Business value distribution with Joining Designation
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Total Business Value
25000
50000
75000
100000
125000
150000
175000
Income_first
Joining Designation
1
2
3
4
5

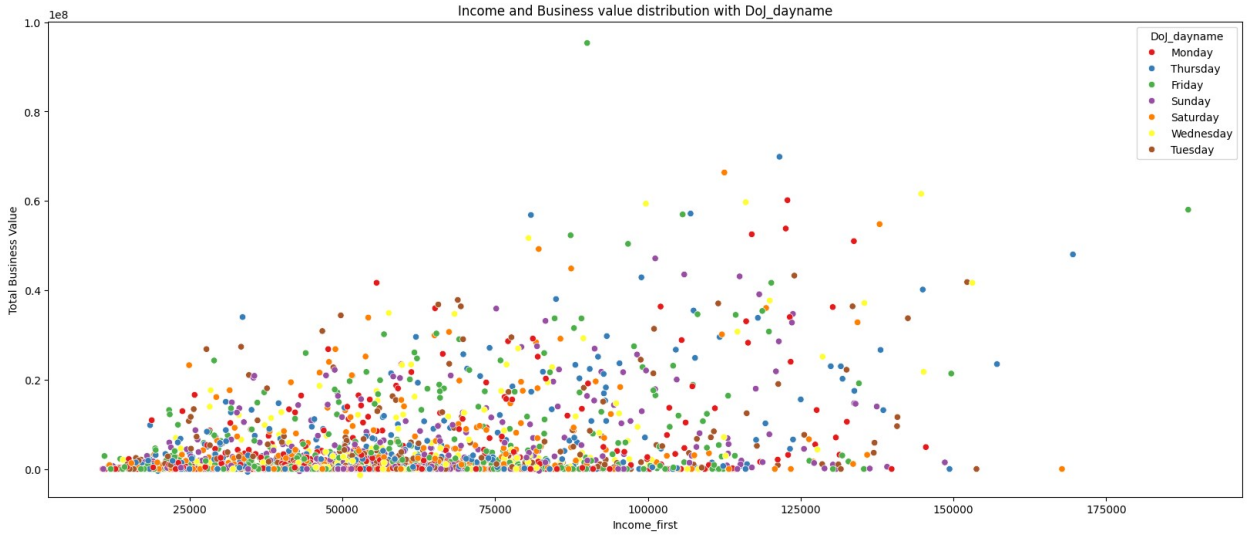

Income and Business value distribution with Grade_first
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Total Business Value
25000
50000
75000
100000
125000
150000
175000
Income_first
Grade_first
1
2
3
4
5

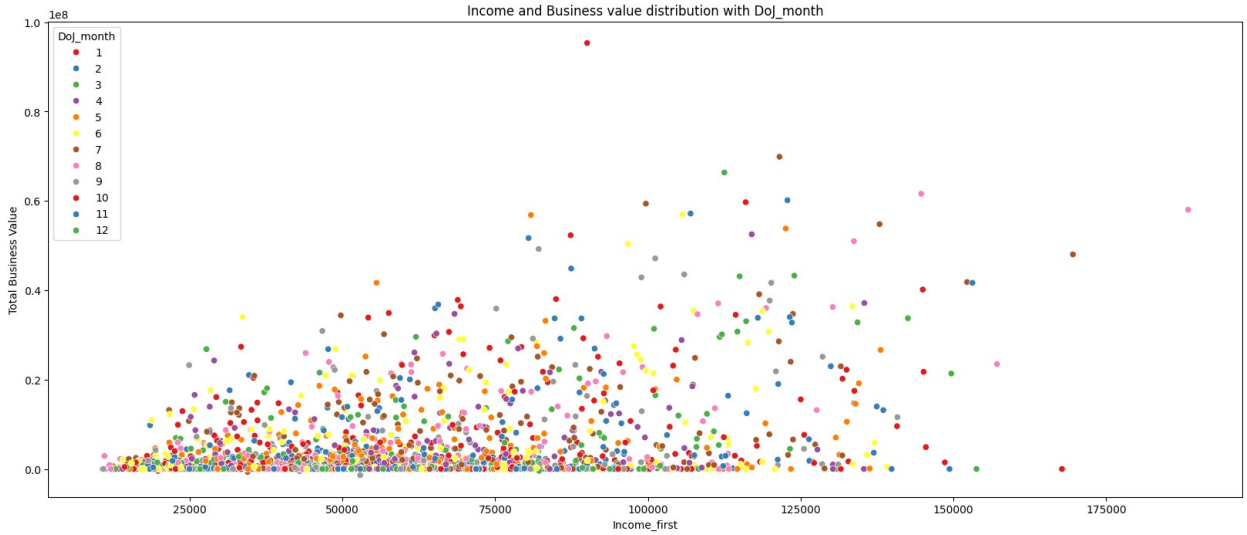

Income and Business value distribution with QR_first
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Total Business Value
25000
50000
75000
100000
125000
150000
175000
Income_first
QR_first
1
2
3
4


Income and Business value distribution with Grade_change
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Total Business Value
25000
50000
75000
100000
125000
150000
175000
Income_first
Grade_change
No Change
Positive

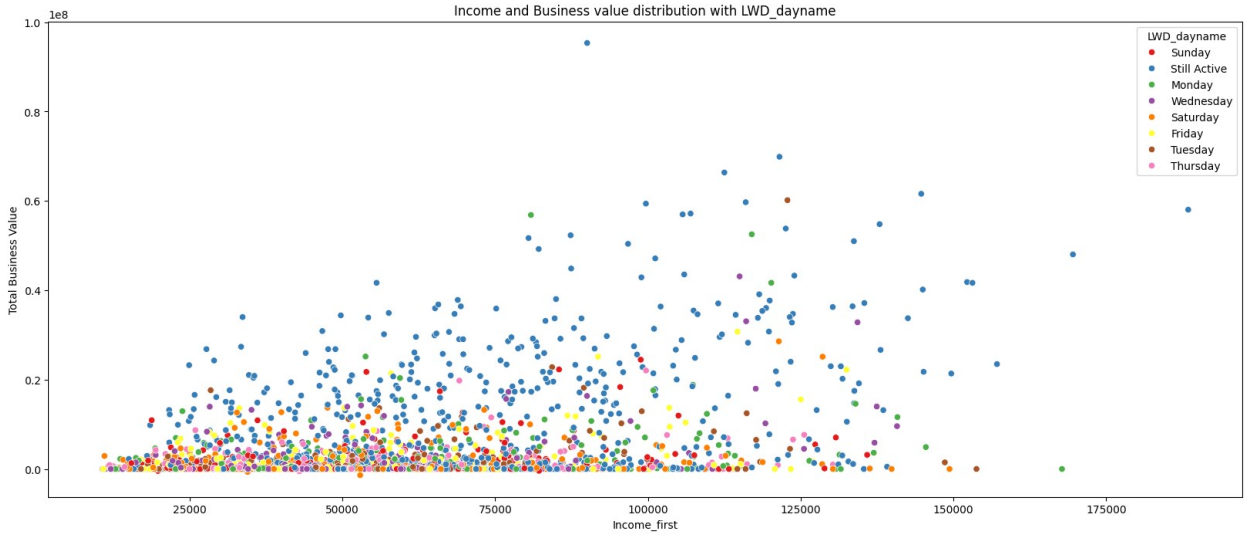

Income and Business value distribution with QR_change
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Total Business Value
25000
50000
75000
100000
125000
150000
175000
Income_first
QR_change
No Change
Positive
Negative

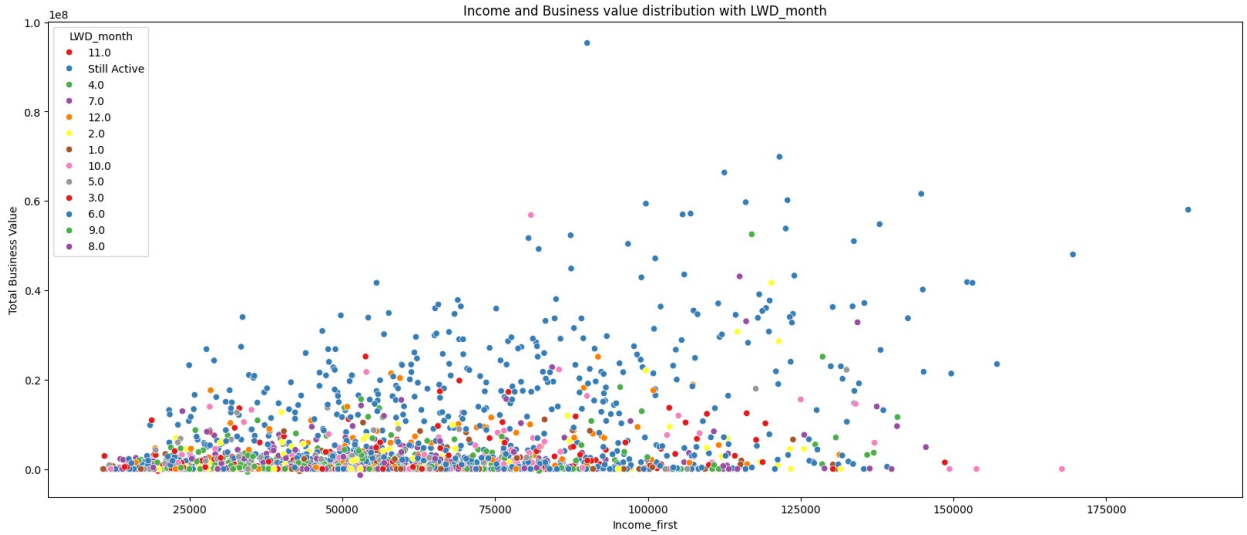

Income and Business value distribution with Income_change
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Total Business Value
25000
50000
75000
100000
125000
150000
175000
Income_first
Income_change
No Change
Positive

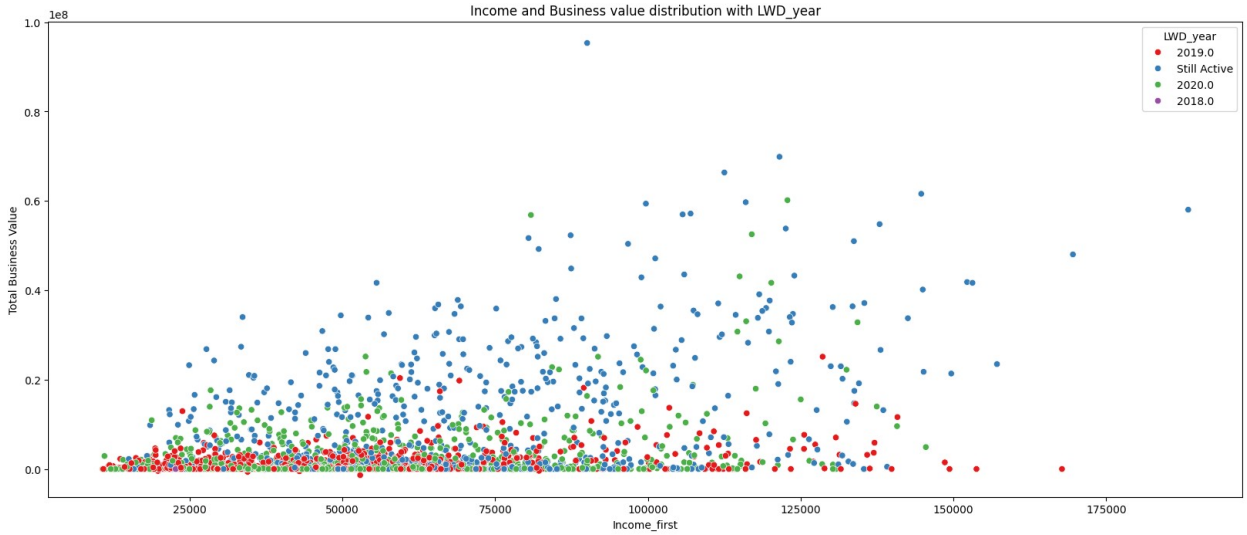

Income and Business value distribution with DoJ_dayname
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Total Business Value
25000
50000
75000
100000
125000
150000
175000
Income_first
DoJ_dayname
Monday
Thursday
Friday
Sunday
Saturday
Wednesday
Tuesday

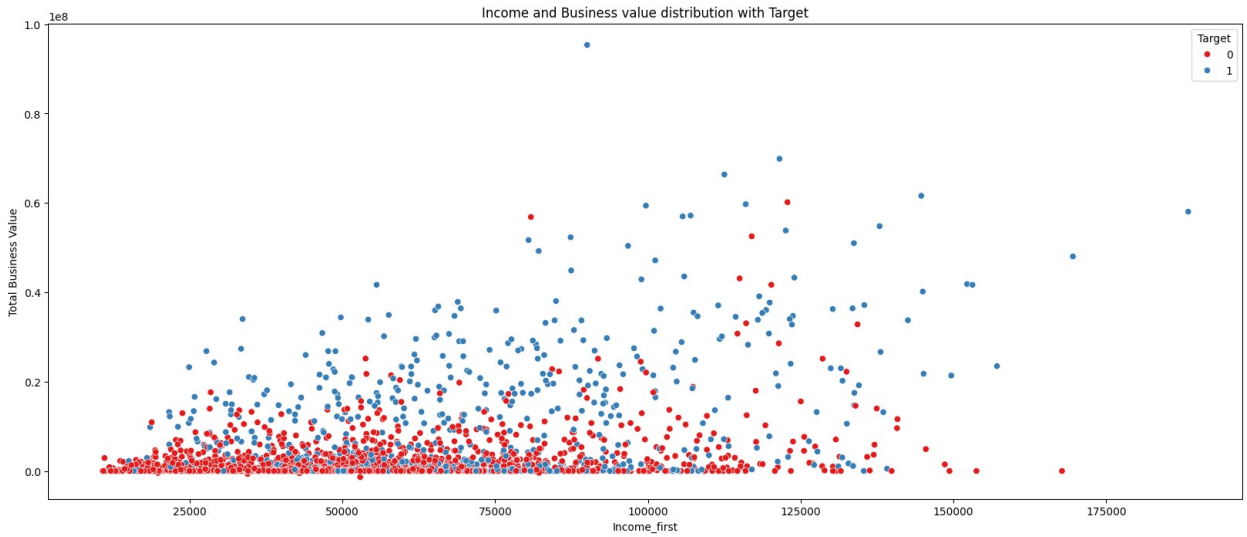

Income and Business value distribution with DoJ_month
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Total Business Value
25000
50000
75000
100000
125000
150000
175000
Income_first
DoJ_month
1
2
3
4
5
6
7
8
9
10
11
12


Income and Business value distribution with DoJ_year
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Total Business Value
25000
50000
75000
100000
125000
150000
175000
Income_first
DoJ_year
2013
2014
2015
2016
2017
2018
2019
2020

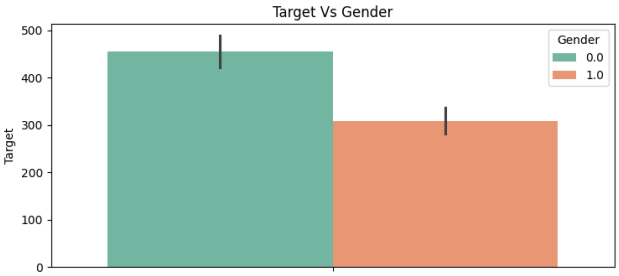

Income and Business value distribution with LWD_dayname
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Total Business Value
25000
50000
75000
100000
125000
150000
175000
Income_first
LWD_dayname
Sunday
Still Active
Monday
Wednesday
Saturday
Friday
Tuesday
Thursday

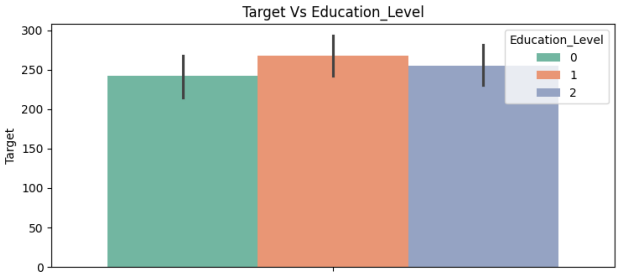

Income and Business value distribution with LWD_month
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Total Business Value
25000
50000
75000
100000
125000
150000
175000
Income_first
LWD_month
11.0
Still Active
4.0
7.0
12.0
2.0
1.0
10.0
5.0
3.0
6.0
9.0
8.0

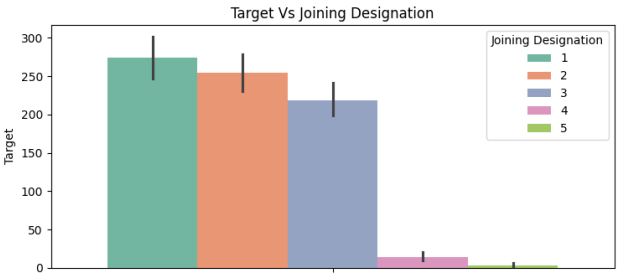

Income and Business value distribution with LWD_year
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Total Business Value
25000
50000
75000
100000
125000
150000
175000
Income_first
LWD_year
2019.0
Still Active
2020.0
2018.0

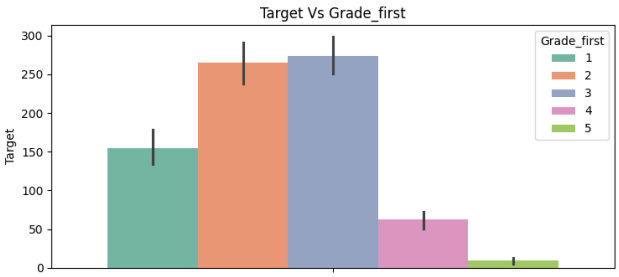


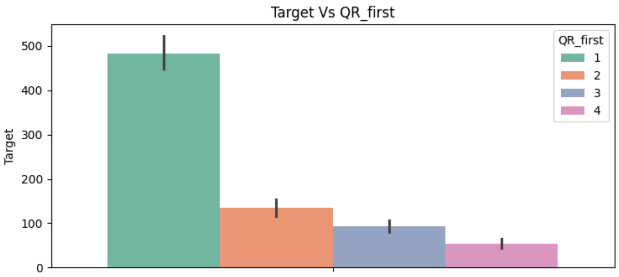

```
def bi_plot(col_list):
    plt.figure(figsize=(15,(len(col_list)*4)))
    for i,c in enumerate(col_list):
        #plt.figure(figsize = (10,5))
        plt.subplot(len(col_list)+1, 2, i+1)
        sns.barplot(data = df_2, y = 'Target', hue = c, estimator =
'sum', palette = 'Set2')
        plt.title("Target Vs {}".format(c))
    plt.tight_layout()
    plt.show()

bi_plot(col_list_2[:-1])
```

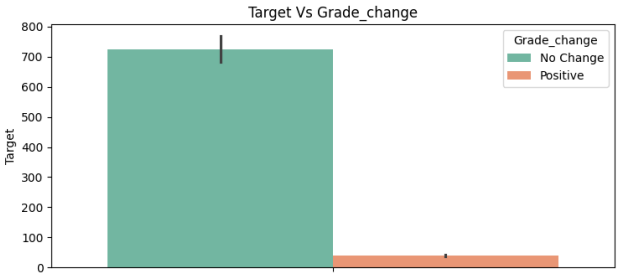

Target Vs Gender
Target
Gender
0.0
1.0

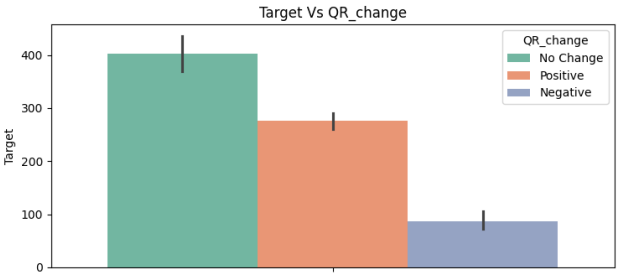

Target Vs Education_Level
Target
Education_Level
0
1
2

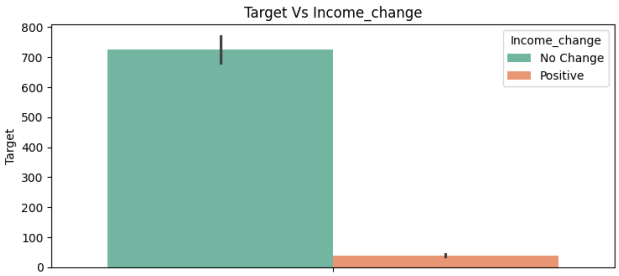

Target Vs Joining Designation
Target
Joining Designation
1
2
3
4
5

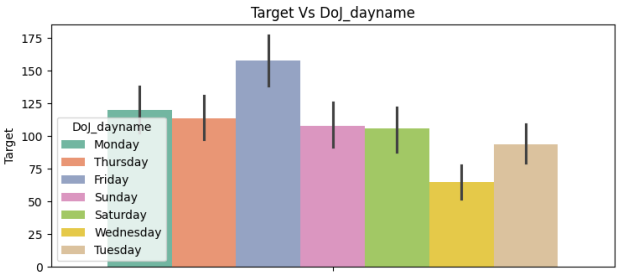

Target Vs Grade_first
Target
Grade_first
1
2
3
4
5

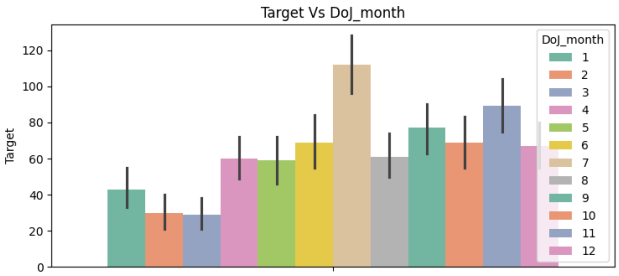

Target Vs QR_first
Target
QR_first
1
2
3
4


Target Vs Grade_change
Target
Grade_change
No Change
Positive


Target Vs QR_change
Target
QR_change
No Change
Positive
Negative


Target Vs Income_change
Target
Income_change
No Change
Positive

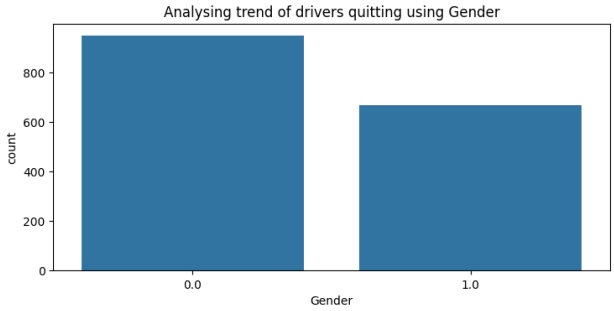

Target Vs DoJ_dayname
Target
DoJ_dayname
Monday
Thursday
Friday
Sunday
Saturday
Wednesday
Tuesday

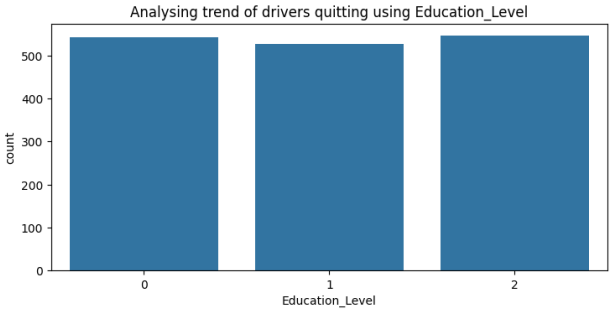

Target Vs DoJ_month
Target
DoJ_month
1
2
3
4
5
6
7
8
9
10
11
12

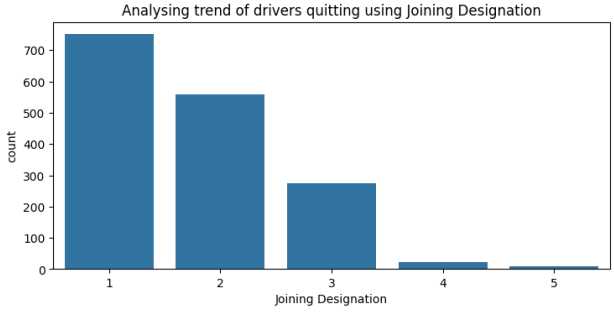

Target Vs DoJ_year
DoJ_year
2013
2014
2015
2016
2017
2018

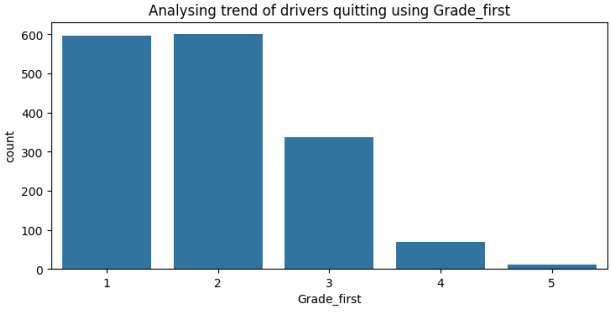

Target Vs LWD_dayname
LWD_dayname
Sunday
Still Active
Monday
Wednesday
Saturday
Friday

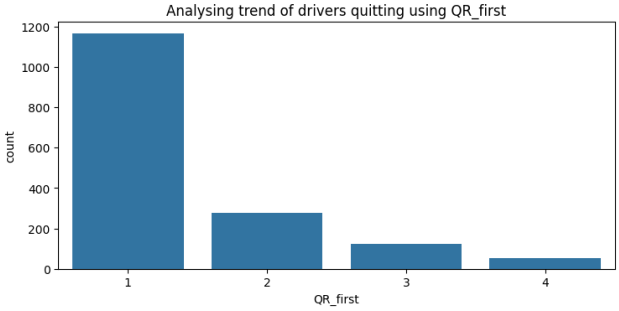
```
# defining a functin to only look at the data where drivers have quit.
def bi_plot_2(col_list):
    plt.figure(figsize=(15,(len(col_list)*4)))
    for i,c in enumerate(col_list):
        #plt.figure(figsize = (10,5))
        plt.subplot(len(col_list)+1, 2, i+1)
        sns.countplot(data = df_2[df_2['Target']==0], x = c)
        plt.title("Analysing trend of drivers quitting using
{}".format(c))
    plt.tight_layout()
    plt.show()

bi_plot_2(col_list_2[:-1])
```

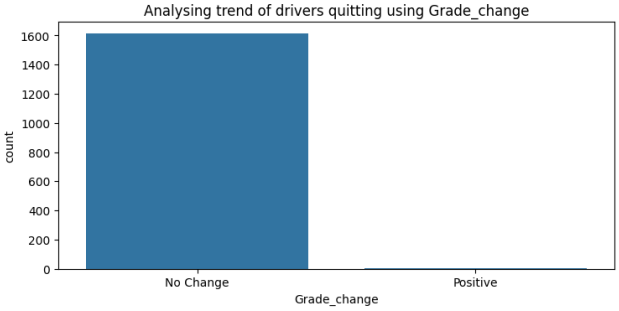

Analysing trend of drivers quitting using Gender
count
Gender
0.0
1.0

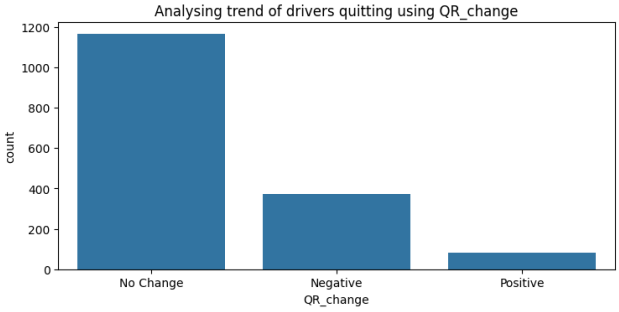

Analysing trend of drivers quitting using Education_Level
count
Education_Level
0
1
2

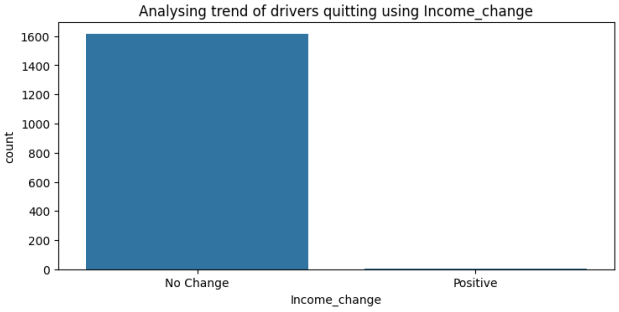

Analysing trend of drivers quitting using Joining Designation
count
Joining Designation
1
2
3
4
5

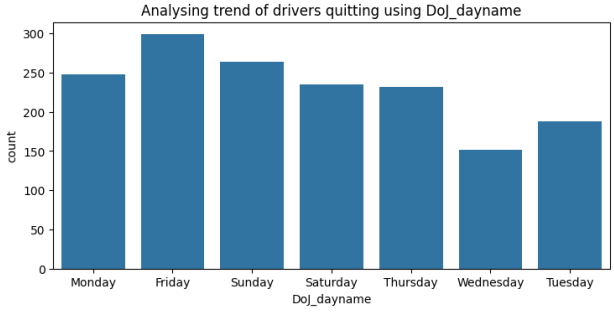

Analysing trend of drivers quitting using Grade_first
count
Grade_first
1
2
3
4
5

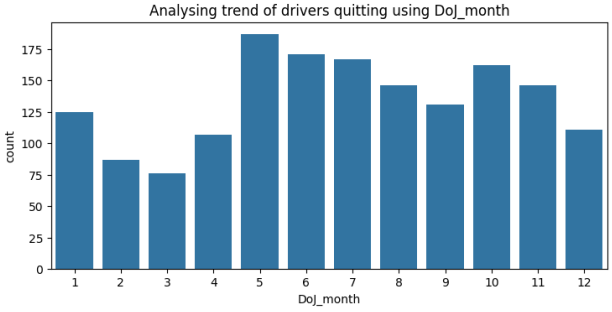

Analysing trend of drivers quitting using QR_first
count
QR_first
1
2
3
4

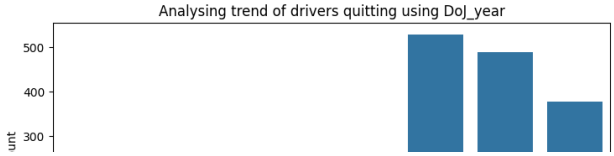

Analysing trend of drivers quitting using Grade_change
count
Grade_change
No Change
Positive

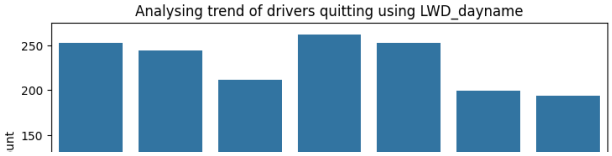

Analysing trend of drivers quitting using QR_change
count
QR_change
No Change
Negative
Positive


Analysing trend of drivers quitting using Income_change
count
Income_change
No Change
Positive


Analysing trend of drivers quitting using DoJ_dayname
count
DoJ_dayname
Monday
Friday
Sunday
Saturday
Thursday
Wednesday
Tuesday


Analysing trend of drivers quitting using DoJ_month
count
DoJ_month
1
2
3
4
5
6
7
8
9
10
11
12


Analysing trend of drivers quitting using DoJ_year


Analysing trend of drivers quitting using LWD_dayname

```
df_2.Target.value_counts(normalize = True)
```

```
Target
0    0.678706
1    0.321294
Name: proportion, dtype: float64
```

Observation Set : 2

1. The dataset is imbalanced, we can observe the sam in the Target column analysis. We will be using balancing techniques later.
2. Total business value has many outliers and is highly skewed with a long right tail.
3. income also has outliers and a long right tail.
4. 41% of the dirvers are female
5. Grade Change seems to have happened only for 1.8% drivers
6. Qurterly rating positive change in only 15% and negative is 19.2%
7. Income change has positively changed only for 1.8% drivers
8. Within the given data, ~68% drivers have quit
9. With higher Grade the income and business values can be seen rising.
10. There is a correlation between 'No of time reported' and Business value
11. 43% is the share of the lowest Joining Designation of 1
12. Education Levels are evenly distributed and seems to be having no affect on income or business value

Section 4 : Data Preparation for Model Building

```
from sklearn.preprocessing import StandardScaler, OrdinalEncoder,
OneHotEncoder
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

df_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 28 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Driver_ID             2381 non-null   int64
 1   No of times reported  2381 non-null   int64
 2   Gender                2381 non-null   float64
 3   City                  2381 non-null   object
 4   Education_Level       2381 non-null   int64
 5   Income_first          2381 non-null   int64
 6   Joining Designation   2381 non-null   int64
 7   Grade_first           2381 non-null   int64
```

```
 8   Total Business Value  2381 non-null   int64
 9   QR_first              2381 non-null   int64
 10  Income_last           2381 non-null   int64
 11  Grade_last            2381 non-null   int64
 12  QR_last               2381 non-null   int64
 13  Grade_diff            2381 non-null   int64
 14  QR_diff               2381 non-null   int64
 15  Income_diff           2381 non-null   int64
 16  Grade_change          2381 non-null   object
 17  QR_change             2381 non-null   object
 18  Income_change         2381 non-null   object
 19  Target                2381 non-null   int32
 20  DoJ_dayname           2381 non-null   object
 21  DoJ_day               2381 non-null   int32
 22  DoJ_month             2381 non-null   int32
 23  DoJ_year              2381 non-null   int32
 24  LWD_dayname           2381 non-null   object
 25  LWD_day               2381 non-null   object
 26  LWD_month             2381 non-null   object
 27  LWD_year              2381 non-null   object
dtypes: float64(1), int32(4), int64(14), object(9)
memory usage: 483.8+ KB
```

Pre-processing steps/actions

1. Driver_ID : to be dropped
2. No of times reported : will be scaled using standard scaler
3. Gender : currently has 0,1 as values, which is good
4. City : Will try to use one-hot encoding inthe first iteration and then drop it in the second iteration. will see what performs better
5. Education_Level, Joining_Designation, Grade, Quarterly Rating, dayname, day, month, year, week : will implement ordinal encoding over it
6. Total Business Value, Income : will be scaled using standard scaler

```
# creating the desired dataset for pre-processsing
df_3 = df_2.drop(columns =
['Income_last','Grade_last','QR_last','Grade_diff','QR_diff','Income_d
iff',

'LWD_dayname','LWD_day','LWD_month','LWD_year'], axis = 1)

df_3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 18 columns):
```

```
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Driver_ID             2381 non-null   int64
 1   No of times reported  2381 non-null   int64
 2   Gender                2381 non-null   float64
 3   City                  2381 non-null   object
 4   Education_Level       2381 non-null   int64
 5   Income_first          2381 non-null   int64
 6   Joining Designation   2381 non-null   int64
 7   Grade_first           2381 non-null   int64
 8   Total Business Value  2381 non-null   int64
 9   QR_first              2381 non-null   int64
 10  Grade_change          2381 non-null   object
 11  QR_change             2381 non-null   object
 12  Income_change         2381 non-null   object
 13  Target                2381 non-null   int32
 14  DoJ_dayname           2381 non-null   object
 15  DoJ_day               2381 non-null   int32
 16  DoJ_month             2381 non-null   int32
 17  DoJ_year              2381 non-null   int32
dtypes: float64(1), int32(4), int64(8), object(5)
memory usage: 297.8+ KB
```

```
oneHot_encoder = OneHotEncoder(sparse_output = False)
ord_encoder = OrdinalEncoder()
std_scaler = StandardScaler()

# Create a DataFrame from the city encoded feature
city_encoded_df =
pd.DataFrame(oneHot_encoder.fit_transform(df_3[['City']]),
columns=oneHot_encoder.get_feature_names_out(['City']))

# Select the features to encode
ordinal_features = df_3[['Grade_change','QR_change',
'Income_change','DoJ_dayname']]

# Fit and transform the features
ord_encoded_features = ord_encoder.fit_transform(ordinal_features)

# Create a new DataFrame with the encoded features
ord_encoded_df = pd.DataFrame(ord_encoded_features,
columns=ordinal_features.columns)

ord_encoded_df
```

```
      Grade_change  QR_change  Income_change  DoJ_dayname
0              0.0        1.0            0.0          1.0
1              0.0        1.0            0.0          4.0
2              0.0        1.0            0.0          0.0
3              0.0        1.0            0.0          3.0
```

```
4               0.0        2.0            0.0          0.0
...             ...        ...            ...          ...
2376            0.0        2.0            0.0          4.0
2377            0.0        1.0            0.0          0.0
2378            0.0        0.0            0.0          5.0
2379            0.0        0.0            0.0          2.0
2380            0.0        2.0            0.0          4.0

[2381 rows x 4 columns]
```

```
# dropping the features which have been encoded
df_3.drop(columns = ['City', 'Grade_change','QR_change',
'Income_change','DoJ_dayname'], axis = 1, inplace = True)

# concatinating the output of encoding with the original dataset
df_4 = pd.concat([df_3, city_encoded_df, ord_encoded_df], axis = 1)

# checking the result of the concatination
df_4.head().T
```

```
                                0        1         2         3
4
Driver_ID                     1.0      2.0       4.0       5.0
6.0
No of times reported          3.0      2.0       5.0       3.0
5.0
Gender                        0.0      0.0       0.0       0.0
1.0
Education_Level               2.0      2.0       2.0       0.0
1.0
Income_first              57387.0  67016.0   65603.0   46368.0
78728.0
Joining Designation           1.0      2.0       2.0       1.0
3.0
Grade_first                   1.0      2.0       2.0       1.0
3.0
Total Business Value    1715580.0      0.0  350000.0  120360.0
1265000.0
QR_first                      2.0      1.0       1.0       1.0
1.0
Target                        0.0      1.0       0.0       0.0
1.0
DoJ_day                      24.0     11.0      12.0       1.0
31.0
DoJ_month                    12.0      6.0       7.0       9.0
7.0
DoJ_year                   2018.0   2020.0    2019.0    2019.0
2020.0
City_C1                       0.0      0.0       0.0       0.0
0.0
```

```
City_C10                    0.0       0.0       0.0       0.0
0.0
City_C11                    0.0       0.0       0.0       0.0
1.0
City_C12                    0.0       0.0       0.0       0.0
0.0
City_C13                    0.0       0.0       1.0       0.0
0.0
City_C14                    0.0       0.0       0.0       0.0
0.0
City_C15                    0.0       0.0       0.0       0.0
0.0
City_C16                    0.0       0.0       0.0       0.0
0.0
City_C17                    0.0       0.0       0.0       0.0
0.0
City_C18                    0.0       0.0       0.0       0.0
0.0
City_C19                    0.0       0.0       0.0       0.0
0.0
City_C2                     0.0       0.0       0.0       0.0
0.0
City_C20                    0.0       0.0       0.0       0.0
0.0
City_C21                    0.0       0.0       0.0       0.0
0.0
City_C22                    0.0       0.0       0.0       0.0
0.0
City_C23                    1.0       0.0       0.0       0.0
0.0
City_C24                    0.0       0.0       0.0       0.0
0.0
City_C25                    0.0       0.0       0.0       0.0
0.0
City_C26                    0.0       0.0       0.0       0.0
0.0
City_C27                    0.0       0.0       0.0       0.0
0.0
City_C28                    0.0       0.0       0.0       0.0
0.0
City_C29                    0.0       0.0       0.0       0.0
0.0
City_C3                     0.0       0.0       0.0       0.0
0.0
City_C4                     0.0       0.0       0.0       0.0
0.0
City_C5                     0.0       0.0       0.0       0.0
0.0
City_C6                     0.0       0.0       0.0       0.0
```

```
0.0
City_C7                         0.0       1.0       0.0       0.0
0.0
City_C8                         0.0       0.0       0.0       0.0
0.0
City_C9                         0.0       0.0       0.0       1.0
0.0
Grade_change                    0.0       0.0       0.0       0.0
0.0
QR_change                       1.0       1.0       1.0       1.0
2.0
Income_change                   0.0       0.0       0.0       0.0
0.0
DoJ_dayname                     1.0       4.0       0.0       3.0
0.0
```

```
# splitting the dependent and independent variables
X = df_4.drop('Target', axis = 1)
y = df_4['Target']

# running train test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, random_state=42)

# since the data was imbalanced, we will resample to balance it
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train,
y_train)

# Scale both the original training data and the resampled training
data
X_train_scaled = std_scaler.fit_transform(X_train_resampled)
X_test_scaled = std_scaler.transform(X_test)
```

Section 5 : Model building

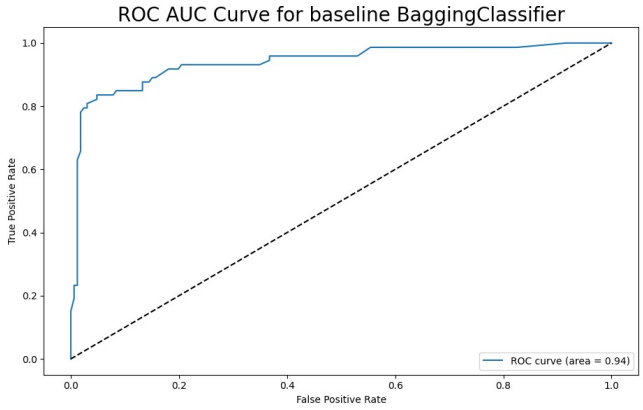
Now that we have the traing and testing data clearly defined with all necessary pre-processing steps taken, we will proceed towards building the models. Our approach will be to first build the baseline models and then move towards tuning those. Later we will compare the models and check the feature importance.

```
# importing required libraries
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier,
BaggingClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
```

```
from sklearn.metrics import accuracy_score, classification_report,
roc_curve, auc



# defining a function to build and get the result of the model
def build_model_get_result(modelname, model, x_train, y_train, x_test,
y_test):
    model.fit(x_train,y_train)
    y_pred = model.predict(x_test)
    # calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    # Calculate ROC AUC curve
    y_pred_proba = model.predict_proba(x_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)


    # Create subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
    # Plot ROC AUC curve in subplot 1
    ax1.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
    ax1.plot([0, 1], [0, 1], 'k--')
    ax1.set_xlabel('False Positive Rate')
    ax1.set_ylabel('True Positive Rate')
    ax1.set_title('ROC AUC Curve for {}'.format(modelname), fontsize =
20)
    ax1.legend(loc="lower right")
    # Print classification report in subplot 2
    report = classification_report(y_test, y_pred) + '\n' + 'Accuracy
of the model is {}'.format(np.round(accuracy,5))
    ax2.text(0.05, 0.5, report, fontsize=15, transform=ax2.transAxes)
    ax2.axis('off')
    ax2.set_title('Classification Report for {}'.format(modelname),
fontsize = 20)
    # Show plot
    plt.tight_layout()
    plt.show()

# defining a function to run the hyper-parameter tuning process for
any given model
def hyper_tuning(model, param, x_train, y_train, cv):
    grid_search = GridSearchCV(estimator = model, param_grid = param,
cv = cv, scoring='accuracy', n_jobs=-1)
    # Fit the GridSearchCV to the training data
    grid_search.fit(x_train, y_train)
    # Print the best hyperparameters
    print("Best Hyperparameters:")
    print(grid_search.best_params_)
```

Bagging Classifier - Baseline

```
bg_clf = BaggingClassifier()
build_model_get_result('baseline BaggingClassifier', rf_clf,
X_train_scaled, y_train_resampled, X_test_scaled, y_test)
```



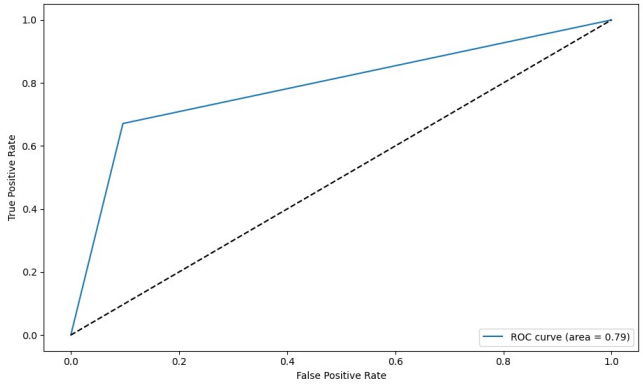
Classification Report for baseline BaggingClassifier

	precision	recall	f1-score	support
0	0.92	0.95	0.94	166
1	0.88	0.82	0.85	73
accuracy			0.91	239
macro avg	0.90	0.89	0.89	239
weighted avg	0.91	0.91	0.91	239

Accuracy of the model is 0.91213

Desicion Tree Classifier - Baseline

```
dt_clf = DecisionTreeClassifier()
build_model_get_result('baseline Decision Tree', dt_clf,
X_train_scaled, y_train_resampled, X_test_scaled, y_test)
```



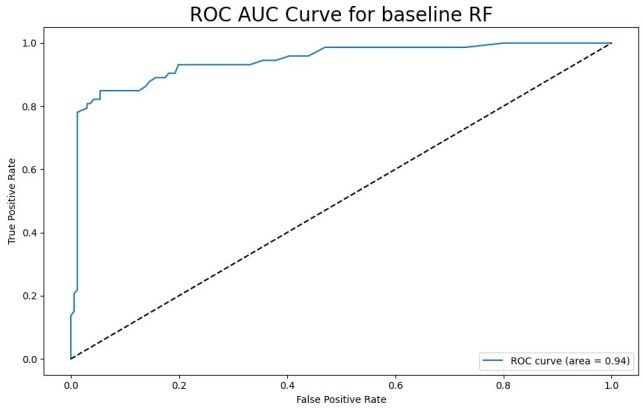
Classification Report for baseline Decision Tree

	precision	recall	f1-score	support
0	0.86	0.90	0.88	166
1	0.75	0.67	0.71	73
accuracy			0.83	239
macro avg	0.81	0.79	0.80	239
weighted avg	0.83	0.83	0.83	239

Accuracy of the model is 0.83264

Random Forest Classifier - Baseline

```
rf_clf = RandomForestClassifier()
build_model_get_result('baseline RF', rf_clf, X_train_scaled,
y_train_resampled, X_test_scaled, y_test)
```



Classification Report for baseline RF

	precision	recall	f1-score	support
0	0.92	0.95	0.93	166
1	0.87	0.82	0.85	73
accuracy			0.91	239
macro avg	0.90	0.88	0.89	239
weighted avg	0.91	0.91	0.91	239

Accuracy of the model is 0.90795

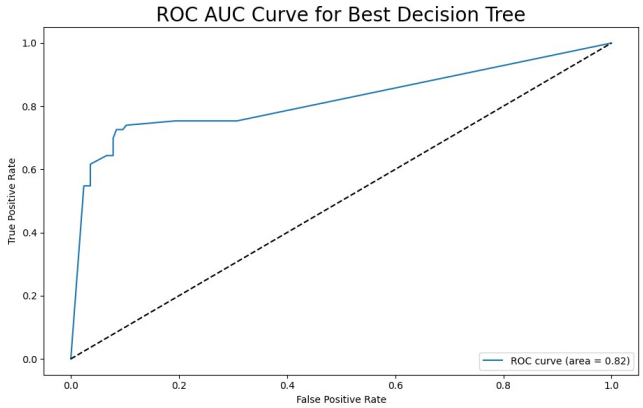
Tuning the Hyper parameter for the Bagging models

Decision Tree Classifier - Tuned

```
# Define the hyperparameter grid for Decision Tree
param_grid_dt = { 'criterion': ['gini', 'entropy'],
                'max_depth': [3, 5, 7, 10],
                'min_samples_split': [2, 5, 10],
                'min_samples_leaf': [1, 2, 4] }
hyper_tuning(dt_clf, param_grid_dt, X_train_scaled, y_train_resampled,
5)
```

```
Best Hyperparameters:
{'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 1,
'min_samples_split': 5}
```

```
# re-defining the Decision Tree with best hyper parameters
dt_clf_best = DecisionTreeClassifier(criterion = 'entropy',
                                     max_depth = 10,
                                     min_samples_leaf = 1,
min_samples_split = 5)

build_model_get_result('Best Decision Tree', dt_clf_best,
X_train_scaled, y_train_resampled, X_test_scaled, y_test)
```



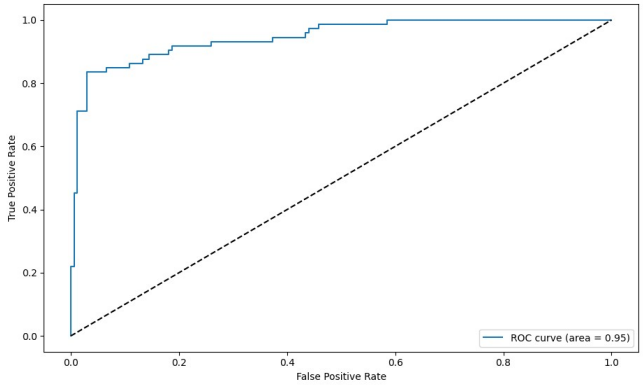
Classification Report for Best Decision Tree

	precision	recall	f1-score	support
0	0.86	0.92	0.89	166
1	0.79	0.66	0.72	73
accuracy			0.84	239
macro avg	0.82	0.79	0.80	239
weighted avg	0.84	0.84	0.84	239

Accuracy of the model is 0.841

Random Forest Classifier - Tuned

```
# Define the hyperparameter grid for Decision Tree
param_grid_rf = { 'n_estimators': [50, 100, 200],
                  'criterion': ['gini', 'entropy'],
                  'max_depth': [3, 7, 15, 30],
                  'min_samples_split': [2, 5, 10], 'min_samples_leaf':
[1, 2, 4]}
hyper_tuning(rf_clf, param_grid_rf, X_train_scaled, y_train_resampled,
5)

Best Hyperparameters:
{'criterion': 'entropy', 'max_depth': 15, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 200}

rf_clf_best = RandomForestClassifier(criterion = 'entropy',
                                     max_depth = 15, min_samples_leaf
= 1,
                                     min_samples_split = 2,
n_estimators = 200)
build_model_get_result('Best RF', rf_clf_best, X_train_scaled,
y_train_resampled, X_test_scaled, y_test)
```



Classification Report for Best RF

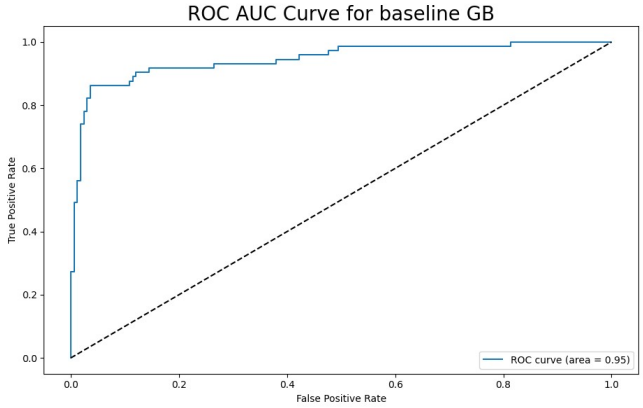
	precision	recall	f1-score	support
0	0.93	0.96	0.95	166
1	0.91	0.84	0.87	73
accuracy			0.92	239
macro avg	0.92	0.90	0.91	239
weighted avg	0.92	0.92	0.92	239

Accuracy of the model is 0.92469

Building Boositing Algrithms

Gradient Boosting - baseline

```
# Initialize GradientBoostingClassifier
gb_clf = GradientBoostingClassifier()
build_model_get_result('baseline GB', gb_clf, X_train_scaled,
y_train_resampled, X_test_scaled, y_test)
```



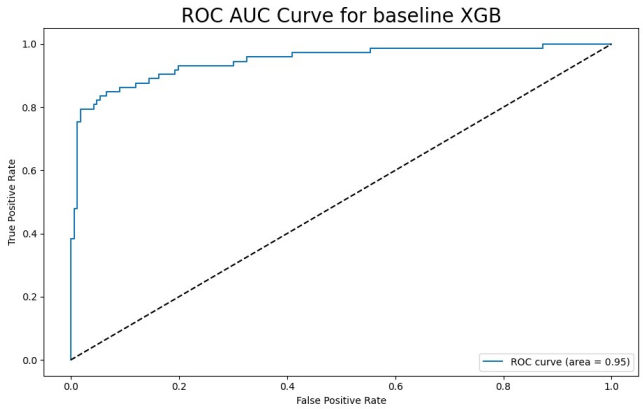
Classification Report for baseline GB

	precision	recall	f1-score	support
0	0.94	0.95	0.94	166
1	0.88	0.86	0.87	73
accuracy			0.92	239
macro avg	0.91	0.90	0.91	239
weighted avg	0.92	0.92	0.92	239

Accuracy of the model is 0.9205

XGBoost - Baseline

```
xgb_clf = XGBClassifier()
build_model_get_result('baseline XGB', xgb_clf, X_train_scaled,
y_train_resampled, X_test_scaled, y_test)
```



Classification Report for baseline XGB

	precision	recall	f1-score	support
0	0.92	0.95	0.93	166
1	0.88	0.81	0.84	73
accuracy			0.91	239
macro avg	0.90	0.88	0.89	239
weighted avg	0.91	0.91	0.91	239

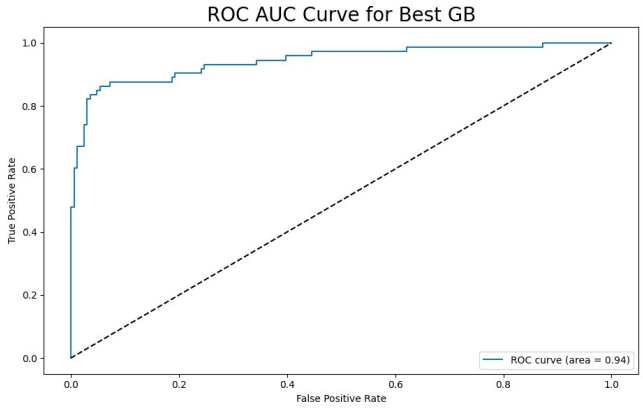
Accuracy of the model is 0.90795

Gradient Boosting - Tuned

```
# Define the hyperparameter grid
param_grid_gb = { 'n_estimators': [50, 100, 200],
                  'learning_rate': [0.01, 0.1, 0.5],
                  'max_depth': [3, 7, 15, 30],
                  'min_samples_split': [2, 5, 10], 'min_samples_leaf':
[1, 3,8,15] }
hyper_tuning(gb_clf, param_grid_gb, X_train_scaled, y_train_resampled,
5)
```

```
Best Hyperparameters:
{'learning_rate': 0.5, 'max_depth': 15, 'min_samples_leaf': 3,
'min_samples_split': 2, 'n_estimators': 200}
```

```
gb_clf_best = GradientBoostingClassifier(learning_rate= 0.5,
                                         max_depth= 15,
min_samples_leaf= 3,
                                         min_samples_split= 2,
n_estimators= 200)
build_model_get_result('Best GB', gb_clf_best, X_train_scaled,
y_train_resampled, X_test_scaled, y_test)
```



Classification Report for Best GB

	precision	recall	f1-score	support
0	0.93	0.95	0.94	166
1	0.89	0.85	0.87	73
accuracy			0.92	239
macro avg	0.91	0.90	0.91	239
weighted avg	0.92	0.92	0.92	239

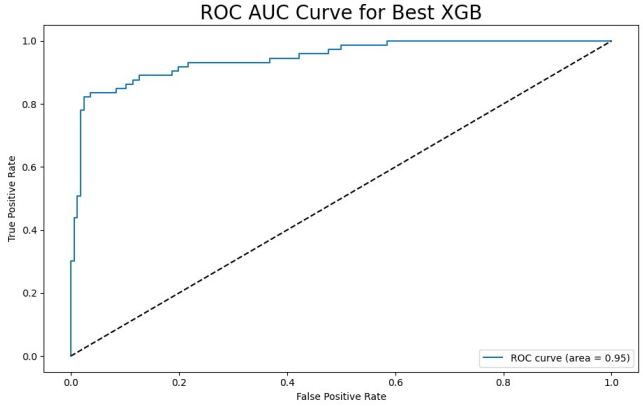
Accuracy of the model is 0.9205

XGBoost - Tuned

```
# Define the hyperparameter grid
param_grid_xgb = { 'n_estimators': [50, 100, 200],
                   'learning_rate': [0.01, 0.1, 0.5],
                   'max_depth': [3, 5, 7],
                   'min_child_weight': [1, 3, 5],
                   'subsample': [0.5, 0.7, 1.0],
                   'colsample_bytree': [0.5, 0.7, 1.0]}
hyper_tuning(xgb_clf, param_grid_xgb, X_train_scaled,
y_train_resampled, 5)
```

```
Best Hyperparameters:
{'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth': 5,
'min_child_weight': 1, 'n_estimators': 200, 'subsample': 0.7}
```

```
xgb_clf_best = XGBClassifier(colsample_bytree= 0.7, learning_rate = 0.1,
                             max_depth= 5, min_child_weight= 1,
                             n_estimators= 200, subsample= 0.7)
build_model_get_result('Best XGB', xgb_clf_best, X_train_scaled, y_train_resampled, X_test_scaled, y_test)
```



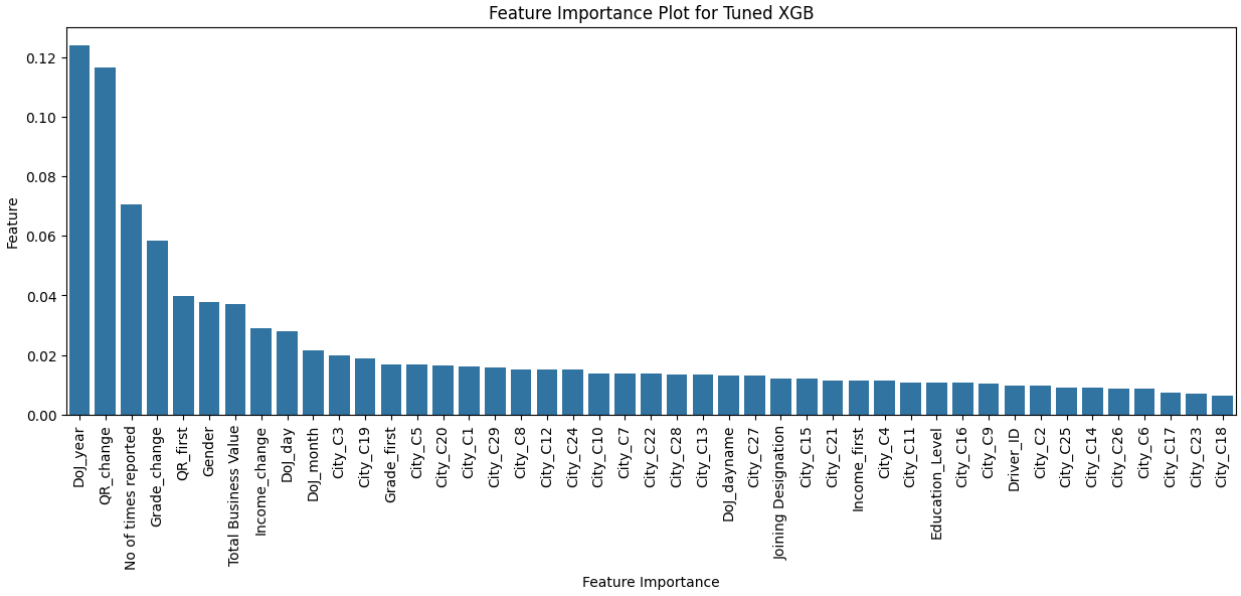
Classification Report for Best XGB

	precision	recall	f1-score	support
0	0.93	0.96	0.95	166
1	0.91	0.84	0.87	73
accuracy			0.92	239
macro avg	0.92	0.90	0.91	239
weighted avg	0.92	0.92	0.92	239

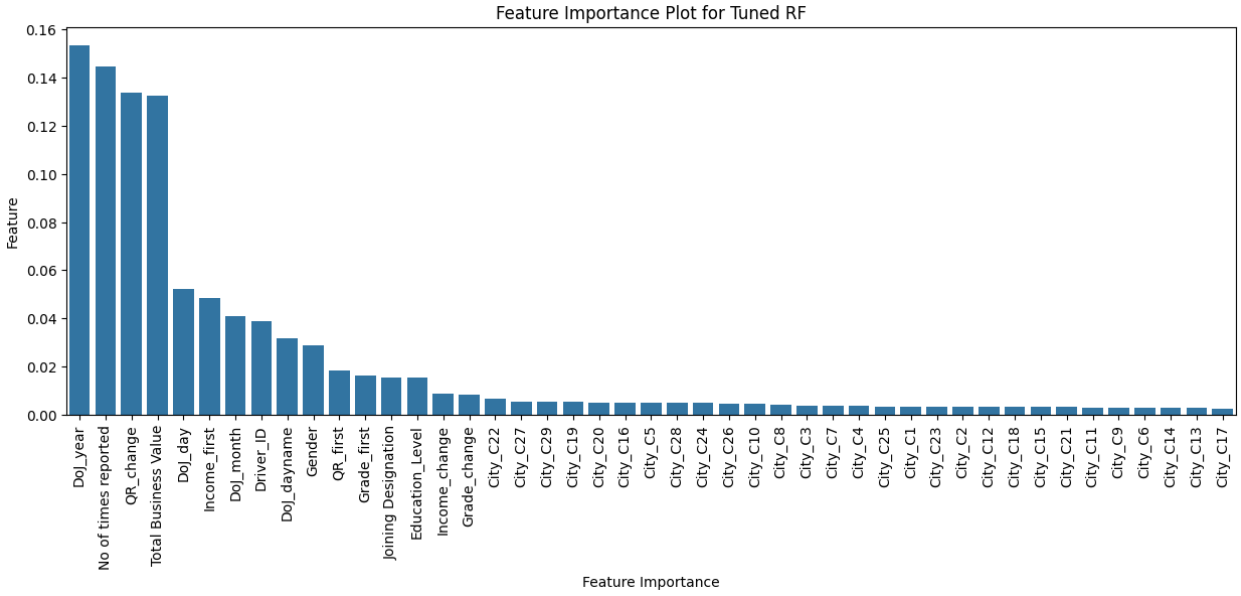
Accuracy of the model is 0.92469

Section 6 : Getting Feature Importance

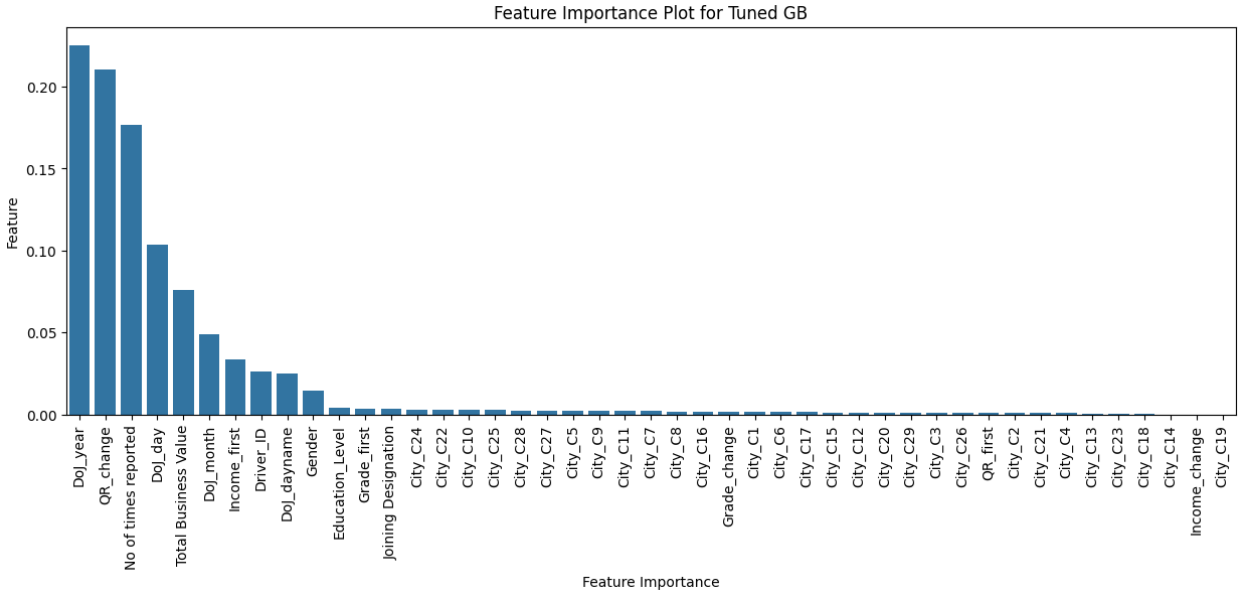
```
def get_feature_imp(modelname, model, cols):
    # Get feature importance
    feature_importances = model.feature_importances_
    # Create a DataFrame for visualization
    feature_importances_df = pd.DataFrame({'Feature': cols, 'Importance': feature_importances})
    # Sort the DataFrame by feature importance
    feature_importances_df = feature_importances_df.sort_values(by='Importance', ascending=False)
    # Visualize feature importance
    plt.figure(figsize = (15,5))
    sns.barplot(y='Importance', x='Feature', data=feature_importances_df)
    plt.xlabel('Feature Importance')
    plt.xticks(rotation = 90)
    plt.ylabel('Feature')
    plt.title('Feature Importance Plot for {}'.format(modelname))
    plt.show()

get_feature_imp('Tuned XGB',xgb_clf_best,X_train_resampled.columns)
```



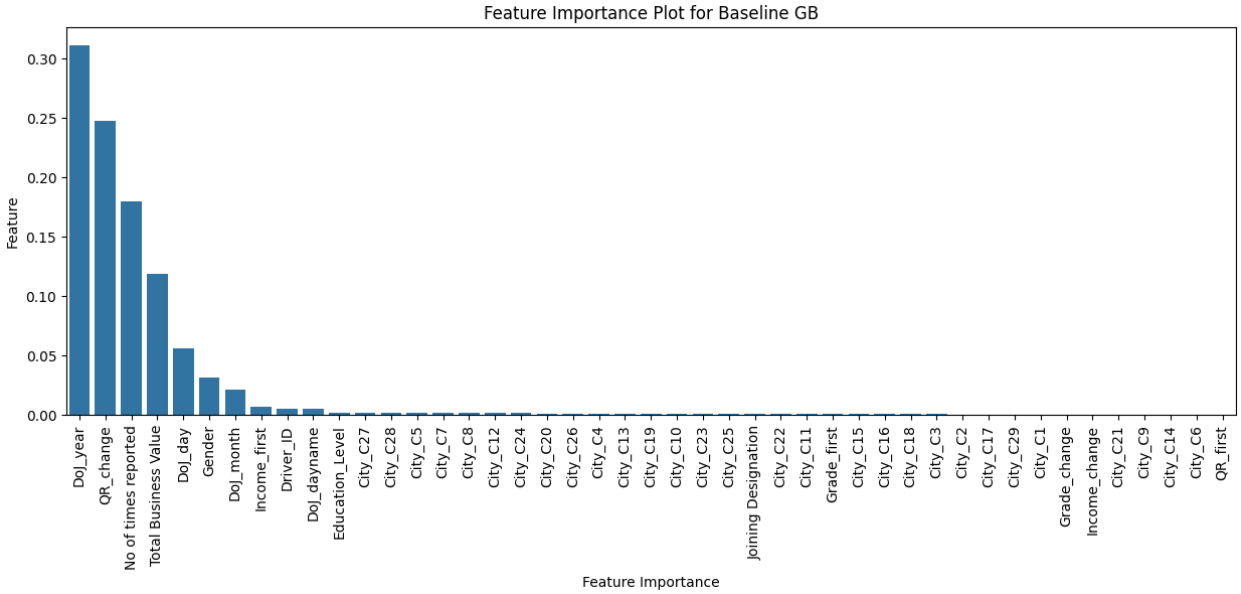

```
get_feature_imp('Tuned RF',rf_clf_best,X_train_resampled.columns)
```




```
get_feature_imp('Tuned GB',gb_clf_best,X_train_resampled.columns)
```




```
get_feature_imp('Baseline GB',gb_clf,X_train_resampled.columns)
```



Section 7 : Insights

1. RF and XGBoost with tuned parameters have the best accuracy of 92.4%
2. Baseline Bagging Classifier has an accuracy of 91%
3. For a trade-off between time to train and accuracy the above 2 points can be used. (Baseline Bagging Classifier was the quickest)
4. The precision recall and f1-score are best for tuned XGBoost

5. DoJ_year, QR_Change, reporting counts are consistently the features of ighest importance.
6. The team should focus on how Quarterly rating to keep a check on the churn
7. Income seems to have some effect on the churn
8. Total Business Value is another feature of much importance, this should be closely watched.
9. Grade change is another factor affecting the churn of the drivers
10. City has no affect on the churn
11. Education level has no effect on the churn