# OLA BUSINESS CASE

**Problem Statement:** To predict whether a driver will be leaving the company or not based on their attributes like

- Demographics (city, age, gender etc.)
- Tenure information (joining date, Last Date)
- Historical data regarding the performance of the driver (Quarterly rating, Monthly business acquired, grade, Income)
- **Dataset:**
- Dataset Link: [ola_driver.csv]()

**Column Profiling:**

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

## PRELIMINARY ANALYSIS

Importing Python libraries

```
[ ]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
```

Data is loaded as pandas dataframe and top 5 rows are obtained using df.head()

```python
df = pd.read_csv("ola.txt")
```

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

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

```python
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
```

**DESCRIPTIVE ANALYSIS**

```
df.describe()
```

| | Unnamed: 0 | Driver_ID | Age | Gender | Education_Level | Income | Joining Designation | Gr |
|---|---|---|---|---|---|---|---|---|
| count | 19104.000000 | 19104.000000 | 19043.000000 | 19052.000000 | 19104.000000 | 19104.000000 | 19104.000000 | 19104.00( |
| mean | 9551.500000 | 1415.591133 | 34.668435 | 0.418749 | 1.021671 | 65652.025126 | 1.690536 | 2.252 |
| std | 5514.994107 | 810.705321 | 6.257912 | 0.493367 | 0.800167 | 30914.515344 | 0.836984 | 1.02€ |
| min | 0.000000 | 1.000000 | 21.000000 | 0.000000 | 0.000000 | 10747.000000 | 1.000000 | 1.00( |
| 25% | 4775.750000 | 710.000000 | 30.000000 | 0.000000 | 0.000000 | 42383.000000 | 1.000000 | 1.00( |
| 50% | 9551.500000 | 1417.000000 | 34.000000 | 0.000000 | 1.000000 | 60087.000000 | 1.000000 | 2.00( |
| 75% | 14327.250000 | 2137.000000 | 39.000000 | 1.000000 | 2.000000 | 83969.000000 | 2.000000 | 3.00( |
| max | 19103.000000 | 2788.000000 | 58.000000 | 1.000000 | 2.000000 | 188418.000000 | 5.000000 | 5.00( |

From the describe method it can be inferred that there are few outliers in income colum which will be treated later.

```
# Here unnammed is not required hecne dropping it
df.drop("Unnamed: 0" , inplace = True , axis = 1)
```

```
[ ] # convert the date columns to date time format
df["MMM-YY"] = pd.to_datetime(df["MMM-YY"])
df["Dateofjoining"] = pd.to_datetime(df["Dateofjoining"])
```

```
[ ] df["LastWorkingDate "] = pd.to_datetime(df["LastWorkingDate"])
```

Explanation of code:

- Heere unnamed column is dropped as it is not required for analysis.
- All the date columns are converted to date time format

Data Preprocessing

Treating missing values

```
df.isna().sum()
```

```
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
LastWorkingDate\t       17488
dtype: int64
```

Observation : Clearly there are columns which has missing values.

```
[ ]  df.duplicated().sum()
```

```
0
```

There are no duplicate values

The null values in gender feature is treated with the mode.

As there are missing values .The numerical columns with missing values are treated with KNNImputer.

```
# fill gender with mode

df["Gender"].fillna(df["Gender"].mode()[0], inplace=True)
```

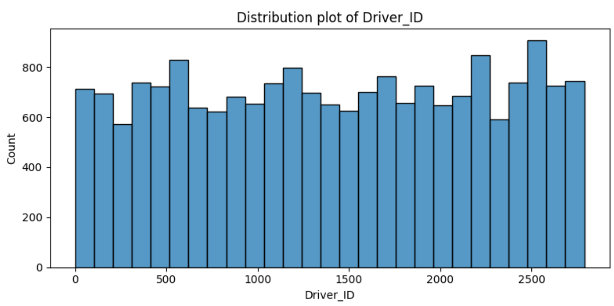
```python
# perform knn imputation on numerical columns

from sklearn.impute import KNNImputer

# Define the KNN imputer with k=5
imputer = KNNImputer(n_neighbors=5)

# Impute missing values in numerical columns
df[num_cols] = imputer.fit_transform(df[num_cols])

# Check for missing values again
df.isna().sum()
```

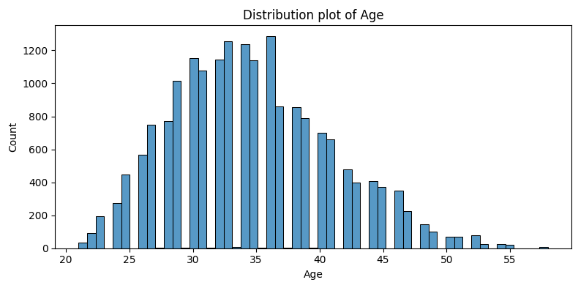
```python
# here the columns lastworkingdate a d lastworkingdate\t are same hence can drop one of them
df.drop("LastWorkingDate\t" , inplace = True , axis = 1)
```

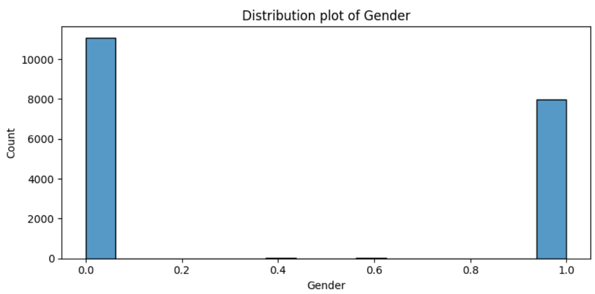
Explanation of the above code :

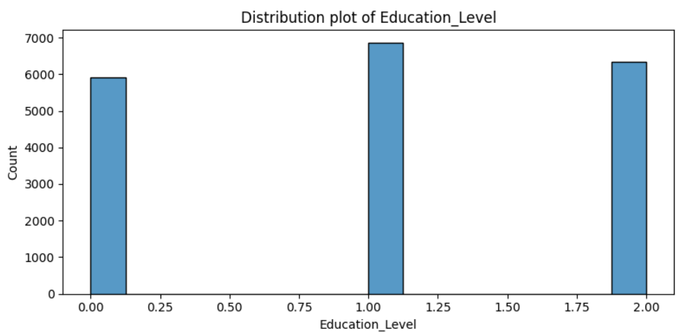
Here lastWorkingDate\t has same information as of lastWorkingDate columns . Hence it can be dropped.

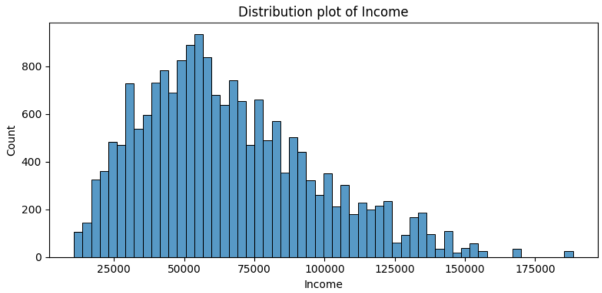
## UNIVARIATE ANALYSIS

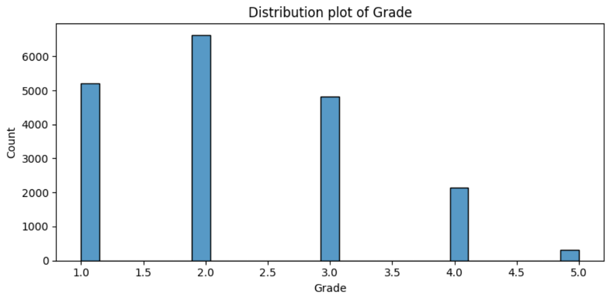


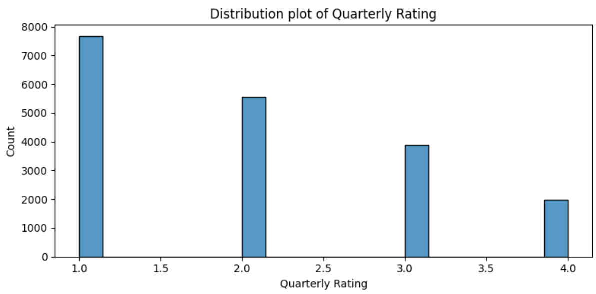



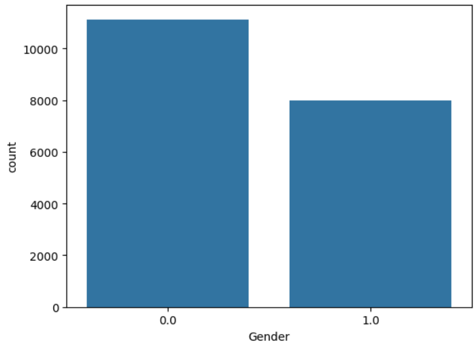





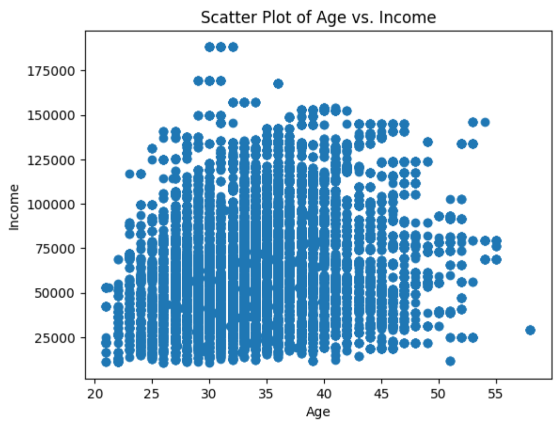



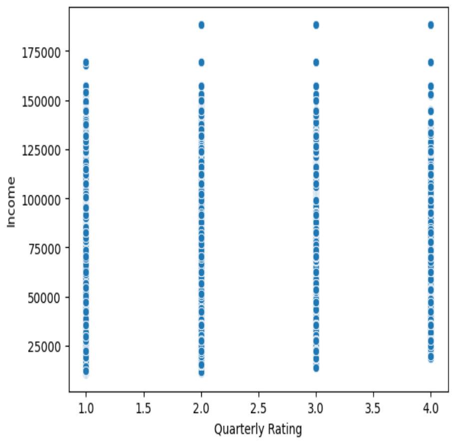



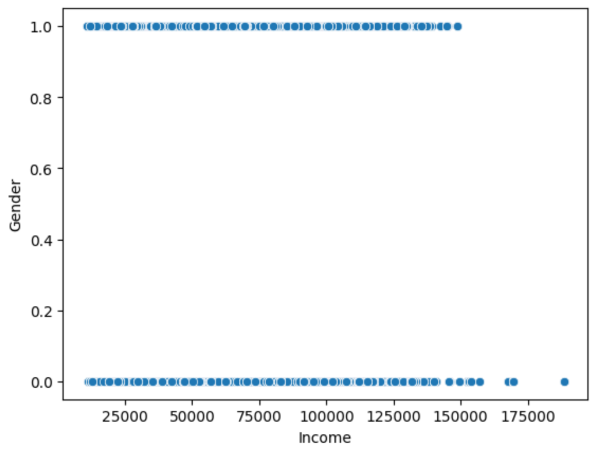

Observation:

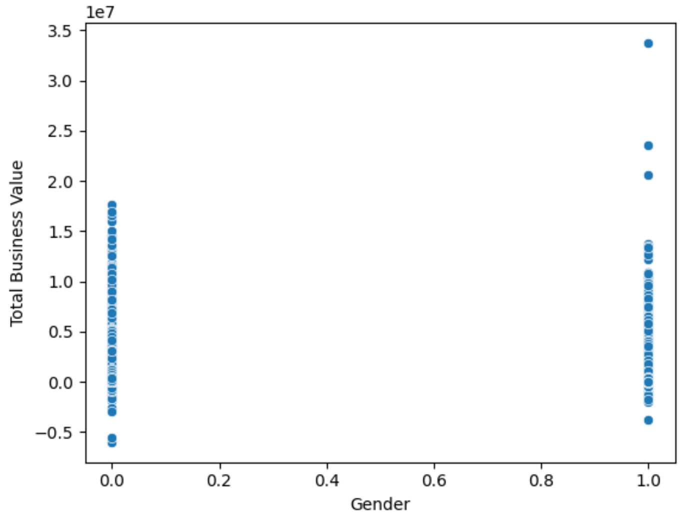
- Based on the distribution plot of age it can be inferred that most of the drivers are between the age group of 31-40. As the plot follows normal distribution.
- In ola maximum drivers are males.
- Income distribution is rightly skewed . Most of the drivers receive the income in the range of 25-80k.
- Mostly the joining designation given to drivers is 1.
- Grade 2 being the highest provided to most of the drivers.
- Most of the driver received 1 rating.

## BIVARIATE ANALYSIS





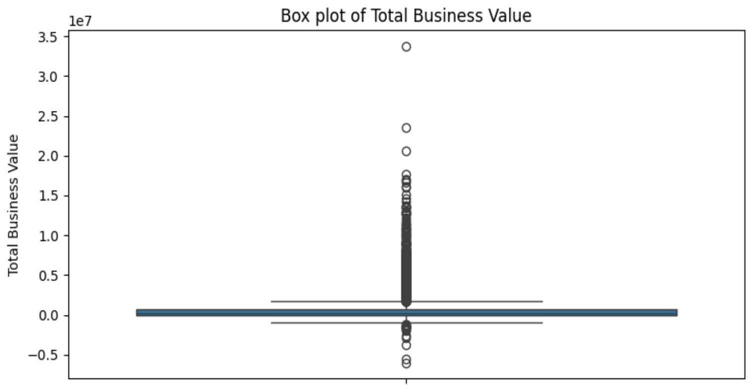



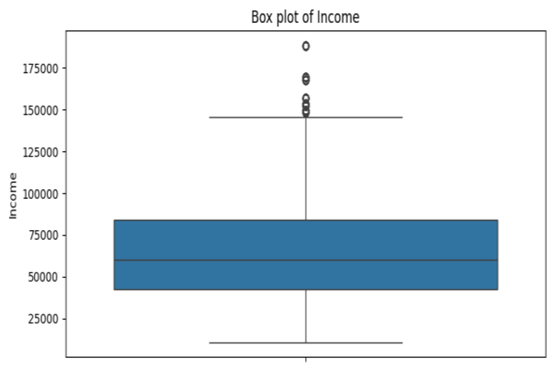



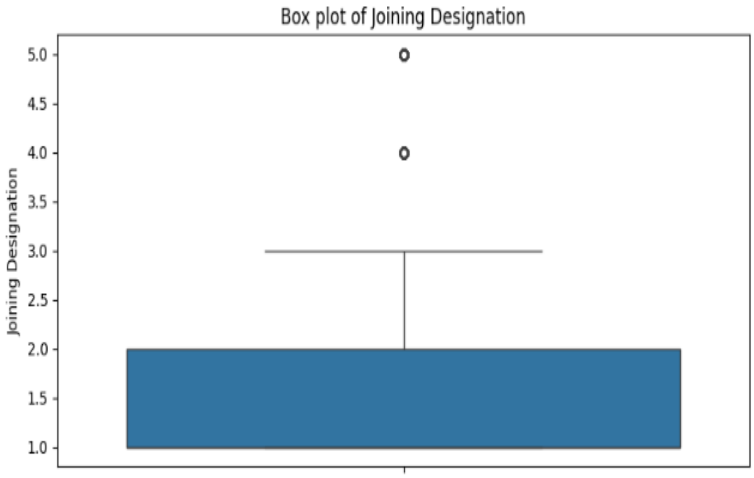

OBSERVATION:

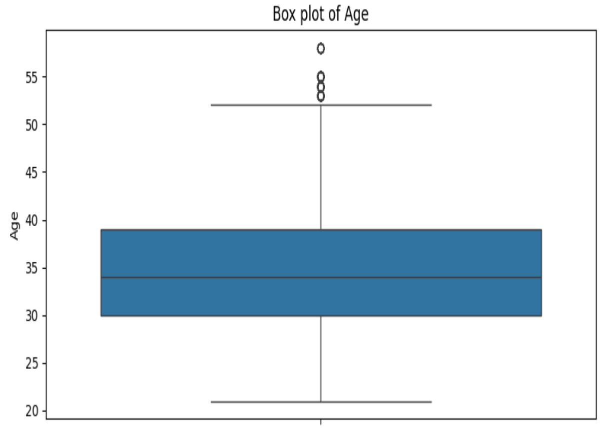
- Clearly age and income are not corelated to each other.
- Similarly gender and quarterly ratings are also not corelated to income.
- Total business value generated by males is slightly more than females.

# OUTLIERS TREATMENT






OBSERVATION:

From the above plots it can be inferred that there are few outliers present in business value , income , age , designation. Hence they are treated with the IQR method.

## Aggregating data in order to remove multiple occurrences of same driver data

```
df = df.groupby('Driver_ID').agg(
    Age=('Age', 'mean'),
    Gender=('Gender', 'first'),
    City=('City', 'first'),
    Income=('Income', 'mean'),
    Quarterly_Rating=('Quarterly Rating', 'mean'),
    Education_Level=('Education_Level', 'first'),
    Dateofjoining=('Dateofjoining', 'first'),
    LastWorkingDate=('LastWorkingDate', 'first')
).reset_index()
```

OBSERVATION

The driver id column is aggregated so as to avoid the duplicate values of driver id and all the features are brought to the same level .

# FEATURE ENGINEERING

## Creating 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.

```
# Create a column which tells whether the quarterly rating has increased for that driver - for those whose quarterly rating has

df['Increased_Rating'] = np.where(df['Quarterly Rating'] > df.groupby('Driver_ID')['Quarterly Rating'].shift(1), 1, 0)
```

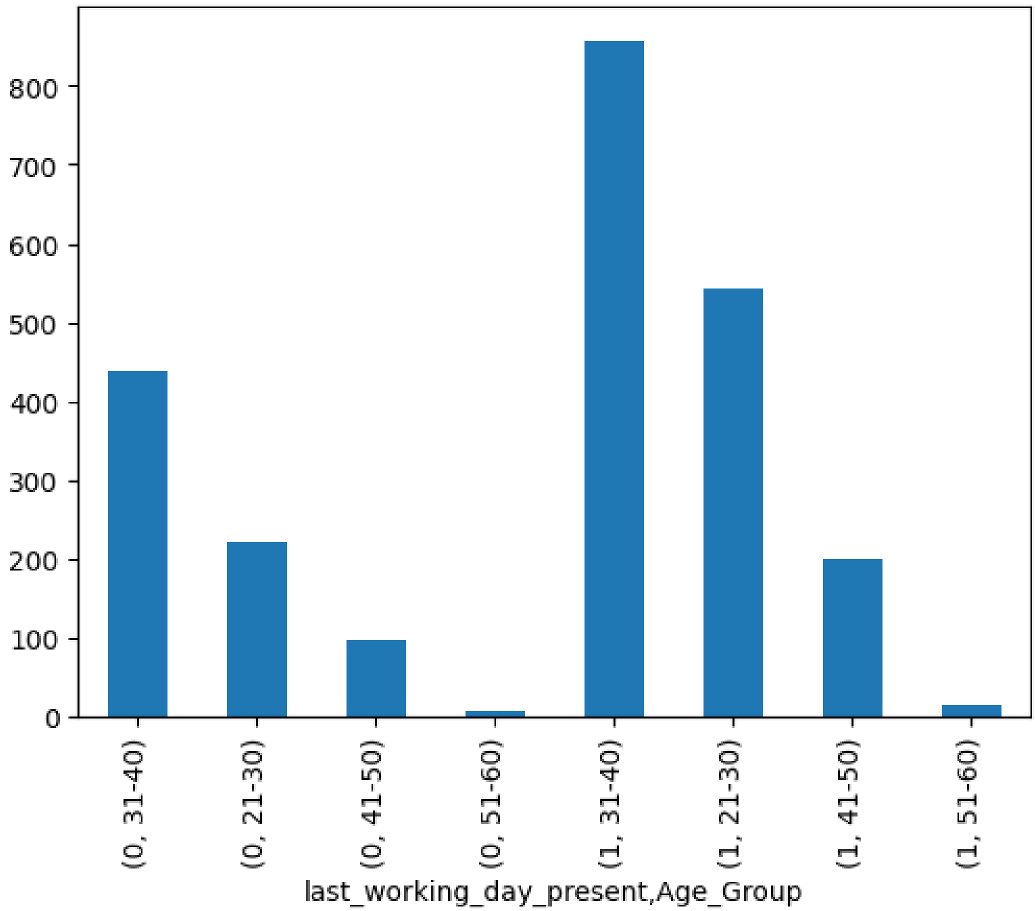
**Target variable creation: Creating  a column called target which tells whether the driver has left the company- driver whose last working day is present will have the value 1 and dropping the alst working day column**

```
# encode the last working date if not null as 1 else 0

df['last_working_day_present'] = df['LastWorkingDate'].apply(lambda x: 1 if pd.notna(x) else 0)
```

## Binning the Age column and dropping the age column

```
#minimum age is 21 and maximum 52.5
#  binnig the age column
bins = [21, 30, 40, 50, 60]
labels = ['21-30', '31-40', '41-50', '51-60']
df['Age_Group'] = pd.cut(df['Age'], bins=bins, labels=labels)
df.head()
```

| | Driver_ID | Age | Gender | City | Income | Quarterly_Rating | Education_Level | Dateofjoining | LastWorkingDate | working_or_not | Age_Group |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | 28.0 | 0.0 | C23 | 57387.0 | 2.0 | 2.0 | 2018-12-24 | 03/11/19 | 0 | 21-30 |
| 1 | 2.0 | 31.0 | 0.0 | C7 | 67016.0 | 1.0 | 2.0 | 2020-11-06 | None | 1 | 31-40 |
| 2 | 4.0 | 43.0 | 0.0 | C13 | 65603.0 | 1.0 | 2.0 | 2019-12-07 | 27/04/20 | 0 | 41-50 |
| 3 | 5.0 | 29.0 | 0.0 | C9 | 46368.0 | 1.0 | 0.0 | 2019-01-09 | 03/07/19 | 0 | 21-30 |
| 4 | 6.0 | 31.0 | 1.0 | C11 | 78728.0 | 1.6 | 1.0 | 2020-07-31 | None | 1 | 31-40 |




## OBSERVATION:

From the above plot it can be inferred drivers of age group 31-40 ha the highest attrition , followed by drivers with age group 21-30 .

## LABEL ENCODING

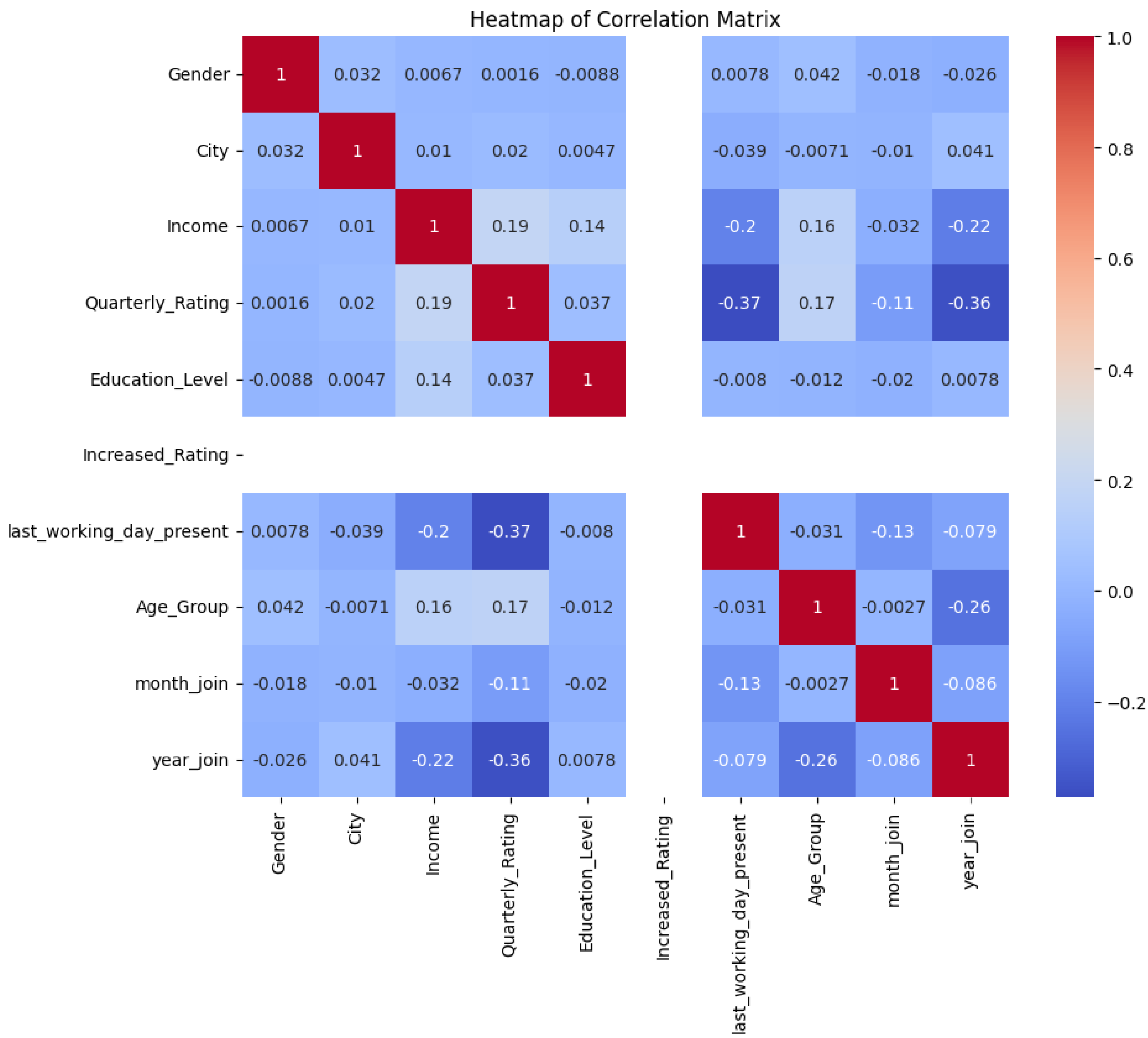
```
# Using dataframe df: label encoder for City

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['City'] = le.fit_transform(df['City'])
df
```

```
[ ] # Using dataframe df: convert the age_group to label encoders
    from sklearn.preprocessing import LabelEncoder
    le = LabelEncoder()
    df['Age_Group'] = le.fit_transform(df['Age_Group'])
    df.head()
```

CODE EXPLANATION

Here Feature encoding is performed on city , Age group

FEATURE CORRELATION




OBSERVATION:

- From the above plot it can be inferred that there  is no strong positive corelation between features.
- There is a negative correlation between features “Quaterly_Rating”  and “last_working_day_present”.
- Also “Quaterly_Rating’ and “year_join” are negatively corelated.

## CHECKING FOR IMBALANCED DATASET

```
#check whether the dataset is imbalanced or not

df["last_working_day_present"].value_counts()
```

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

## OBSERVATION

From the above code it can be inferred that there re 50% more driver who left the company than the drivers remaining. Hence there is a imbalance in dataset.

## TO BALANCE THE DATASET SMOTE TECHNIQUE IS USED

## First the dataset is split based on independent feature and target variable . Making as train and test data.

```
# train test split

from sklearn.model_selection import train_test_split

# Separate features and target
X = df.drop('last_working_day_present', axis=1)
y = df['last_working_day_present']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Print the shapes of the train and test sets
print('Shape of X_train:', X_train.shape)
print('Shape of X_test:', X_test.shape)
print('Shape of y_train:', y_train.shape)
print('Shape of y_test:', y_test.shape)
```

```
Shape of X_train: (1785, 9)
Shape of X_test: (596, 9)
Shape of y_train: (1785,)
Shape of y_test: (596,)
```

## FEATURE SCALING

```
# perform  standard scaler

from sklearn.preprocessing import StandardScaler

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

## PERFORMING SMOTE

```
# Working with an imbalanced dataset
from imblearn.over_sampling import SMOTE
# Oversample the minority class using SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Check the class distribution after oversampling
print(f"Class distribution after oversampling: {y_train_resampled.value_counts()}")
```

```
Class distribution after oversampling: 0    1213
1    1213
Name: last_working_day_present, dtype: int64
```

Now the data is balanced can performing ensembling techniques

# MODEL BUILDING

ENSEMBLING

For  ensembling following algorithms are implemented

```python
# perform ensemble model on df

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import VotingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

# Define the individual models
model1 = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=0)
model2 = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=2, random_state=0)
model3 = AdaBoostClassifier(n_estimators=100, learning_rate=1.0, random_state=0)
model4 = XGBClassifier(random_state=42)

# Create a voting classifier
voting_clf = VotingClassifier(estimators=[('rf', model1), ('gbdt', model2), ('ada', model3)], voting='hard')

# Train the voting classifier
voting_clf.fit(X_train, y_train)

# Predict on the test set
y_pred = voting_clf.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

Ensemble Algorithms used

- Random Forest Algorithm
- Gradient Boosting Algorithm
- Ada Boost
- Voting Classifier
- XGboost Algorithm

```
Accuracy: 0.8422818791946308
```

```
[65] # print classification report
     from sklearn.metrics import classification_report
     # Get the classification report
     report = classification_report(y_test, y_pred)
     # Print the classification report
     print(report)
```

```
              precision    recall  f1-score   support

           0       0.79      0.70      0.74       193
           1       0.87      0.91      0.89       403

    accuracy                           0.84       596
   macro avg       0.83      0.81      0.81       596
weighted avg       0.84      0.84      0.84       596
```

INFERENCE

Based on the ensembled techniques used the accuracy obtained is 84.2%

Precision for lastworkingday present is 87%

Recall for last working day present  is 91%

F1 score  is 0.89

How ever the model did not perform quiet well for last_working day not present means for  the driver that are still in the company

The precision is 79%

Recall is 70%

F1 score is  0.74

## TO IMPROVE SOME ACCURACY HYPER PARAMENTER TUNING IS PERFORMED

Gridsearch CV is used to perform hyper parameter tuning and following report is obtained .

```
# Print the classification report
print(report)
```

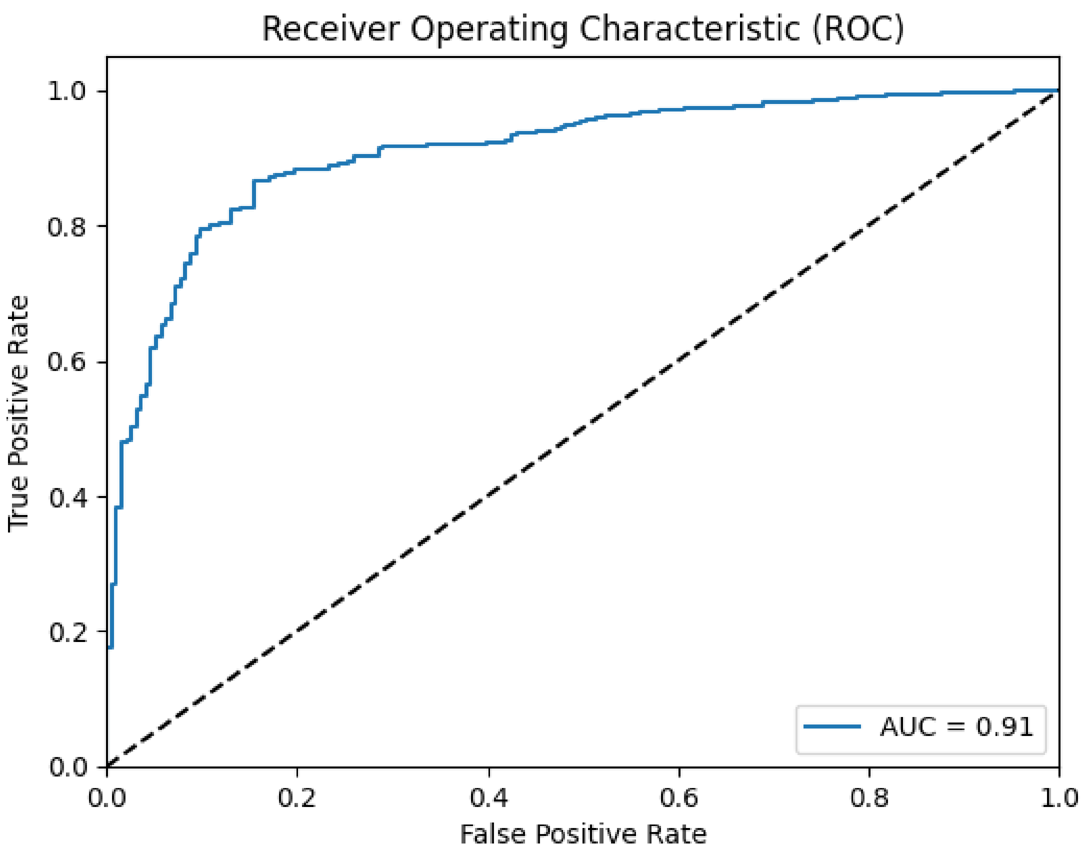
```
Accuracy after hyperparameter tuning: 0.8489932885906041
              precision    recall  f1-score   support

           0       0.77      0.77      0.77       193
           1       0.89      0.89      0.89       403

    accuracy                           0.85       596
   macro avg       0.83      0.83      0.83       596
weighted avg       0.85      0.85      0.85       596
```

The accuracy is improved from 0.842 to 0.848.

There is improvement in F1 score for lastworking_day not present column. Means driver still working.

Also there is improvement in precision recall and f1 score for the column last_workingday_present =1 means for the case where driver left the company.

**PLOTTING ROC AUC CURVE:**




INFERENCE:

- Here AUC is 0.91 which means that there is 91% probability that a driver chosen will leave the company.

- ROC curve corresponding to a more discriminating test are located closer to the upper left hand of the ROC . The highest point at upper left hand of ROC is close to 0.85.
- As per the ROC curve the model will perform decently.

## BUSINESS INSIGHTS

- Out of 2381 drivers 1616 have left the company.
- We need to incentivise the drivers overtime or other perks to overcome churning
- The employees whose quarterly rating has increased are less likely to leave the organization.
- Company needs to implement the reward system for the customer who provide the feedback and rate drivers
- The employees whose monthly salary has not increased are more likely to leave the organization.
- Company needs to get in touch with those drivers whose monthly salary has not increased and help them out to earn more by provider bonus and perks.
- Out of 2381 employees, 1744 employees had their last quarterly rating as 1.
- Out of 2381 employees, the quarterly rating has not increased for 2076 employees. This is red flag for the company which needs to regulate.
- Company needs to look why customers are not rating drivers.
- Last_Quarterly_Rating, Total_Business_Value & Quarterly_Rating_Increased are the most important features. Company needs to tracks these features as predicators
- We observe that we are not getting very high recall on target 0 which may be due to small unbalanced dataset. More data will overcome this issue.
- The Random Forest Classifier attains the Recall score of 91% for the driver who left the company. Which indicates that model is performing the decent job.

## Google Colab Link:

## https://colab.research.google.com/drive/19pibrieGLlgiBaov6let2RZKjPusjVhf?usp=sharing