Problem statement

Recruiting and retaining drivers is seen by industry watchers as a tough battle for Ola. Churn among drivers is high and it's very easy for drivers to stop working for the service on the fly or jump to Uber depending on the rates. You are provided with the monthly information for a segment of drivers for 2019 and 2020 and tasked to predict whether a driver will be leaving the company or not based on their attributes given.

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

```
ola_data = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/002/492/original/ola_driver_scaler.cs
```

```
ola_data.head(10)
```

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dat
0	0	01/01/19	1	28.0	0.0	C23	2	57387	
1	1	02/01/19	1	28.0	0.0	C23	2	57387	
2	2	03/01/19	1	28.0	0.0	C23	2	57387	
3	3	11/01/20	2	31.0	0.0	C7	2	67016	
4	4	12/01/20	2	31.0	0.0	C7	2	67016	
5	5	12/01/19	4	43.0	0.0	C13	2	65603	
6	6	01/01/20	4	43.0	0.0	C13	2	65603	
7	7	02/01/20	4	43.0	0.0	C13	2	65603	
8	8	03/01/20	4	43.0	0.0	C13	2	65603	
9	9	04/01/20	4	43.0	0.0	C13	2	65603	

```
ola_data.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
```

```
ola_data.shape
```

```
(19104, 14)
```

```
ola_data.drop("Unnamed: 0", axis = 1, inplace = True)
```

```
ola_data.isnull().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
dtype: int64
```

Statistical Summary of original data

```
ola_data.describe(include = 'all')
```

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	
count	19104	19104.000000	19043.000000	19052.000000	19104	19104.000000	1
unique	24	NaN	NaN	NaN	29	NaN	
top	01/01/19	NaN	NaN	NaN	C20	NaN	
freq	1022	NaN	NaN	NaN	1008	NaN	
mean	NaN	1415.591133	34.668435	0.418749	NaN	1.021671	6
std	NaN	810.705321	6.257912	0.493367	NaN	0.800167	3
min	NaN	1.000000	21.000000	0.000000	NaN	0.000000	1
25%	NaN	710.000000	30.000000	0.000000	NaN	0.000000	4
50%	NaN	1417.000000	34.000000	0.000000	NaN	1.000000	6
75%	NaN	2137.000000	39.000000	1.000000	NaN	2.000000	8
max	NaN	2788.000000	58.000000	1.000000	NaN	2.000000	18

```
ola_data[["MMM-YY","Dateofjoining","LastWorkingDate"]]= ola_data[["MMM-YY","Dateofjoining","LastWorkingDate"]].apply(pd.to_c
```

Removing multiple occurances of data for same driver

- Since multiple records exist for same drivers, data frame is grouped by Driver_ID and aggreagtion is done to remove multiple occurances.

```
df = ola_data.copy()
```

```
def max_rep(x):
  x["Max_report_date"] = x["MMM-YY"].max()
  return x

df = df.groupby("Driver_ID").apply(max_rep)
```

```
<ipython-input-161-9b0d83e751bc>:5: FutureWarning: Not prepending group keys to the result index of transform-like apply
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  df = df.groupby("Driver_ID").apply(max_rep)
```

```
def avg_business_values(x):
  x["AvgTotal Business Value"] = x["Total Business Value"].mean()
  return x

df = df.groupby("Driver_ID").apply(avg_business_values)
```

```
<ipython-input-162-7513920e3e38>:5: FutureWarning: Not prepending group keys to the result index of transform-like apply
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  df = df.groupby("Driver_ID").apply(avg_business_values)
```

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

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining
0	2019-01-01	1	28.0	0.0	C23	2	57387	2018-12-24
1	2019-02-01	1	28.0	0.0	C23	2	57387	2018-12-24
2	2019-03-01	1	28.0	0.0	C23	2	57387	2018-12-24
3	2020-11-01	2	31.0	0.0	C7	2	67016	2020-11-06
4	2020-12-01	2	31.0	0.0	C7	2	67016	2020-11-06
5	2019-12-01	4	43.0	0.0	C13	2	65603	2019-12-07
6	2020-01-01	4	43.0	0.0	C13	2	65603	2019-12-07
7	2020-02-01	4	43.0	0.0	C13	2	65603	2019-12-07
8	2020-03-01	4	43.0	0.0	C13	2	65603	2019-12-07
9	2020-04-01	4	43.0	0.0	C13	2	65603	2019-12-07

```
df = df[(df["MMM-YY"]== df["Max_report_date"])]
```

```
df.drop(["Total Business Value","Max_report_date"], axis =1, inplace = True)
```

```
df.head()
```

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining
2	2019-03-01	1	28.0	0.0	C23	2	57387	2018-12-24
4	2020-12-01	2	31.0	0.0	C7	2	67016	2020-11-06
9	2020-04-01	4	43.0	0.0	C13	2	65603	2019-12-07
12	2019-03-01	5	29.0	0.0	C9	0	46368	2019-01-09
17	2020-	6	31.0	1.0	C11	1	78728	2020-07-31

```
df.reset_index(inplace = True)
```

```
df.shape
```

```
(2381, 14)
```

```
df.isnull().sum()
```

```
index                    0
MMM-YY                   0
Driver_ID                0
Age                      7
Gender                  10
City                     0
Education_Level          0
Income                   0
Dateofjoining            0
LastWorkingDate        765
Joining Designation      0
Grade                    0
Quarterly Rating         0
AvgTotal Business Value  0
dtype: int64
```

Creatig Target variable

```
df["Churn"] = df["LastWorkingDate"].fillna(1, inplace = True)
df["Churn"] = df["LastWorkingDate"].apply(lambda x:0 if x!= 1 else x)
df.drop("LastWorkingDate", axis = 1, inplace = True)
```

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

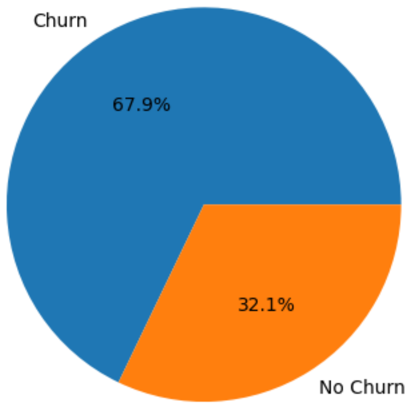
```
0    1616
1     765
Name: Churn, dtype: int64
```

```
#ola_data["Dateofjoining"] = pd.to_datetime(ola_data["Dateofjoining"])
#ola_data["Dateofjoining"] = ola_data["Dateofjoining"].dt.year + ola_data["Dateofjoining"].dt.month/12
```

Univariate Analysis

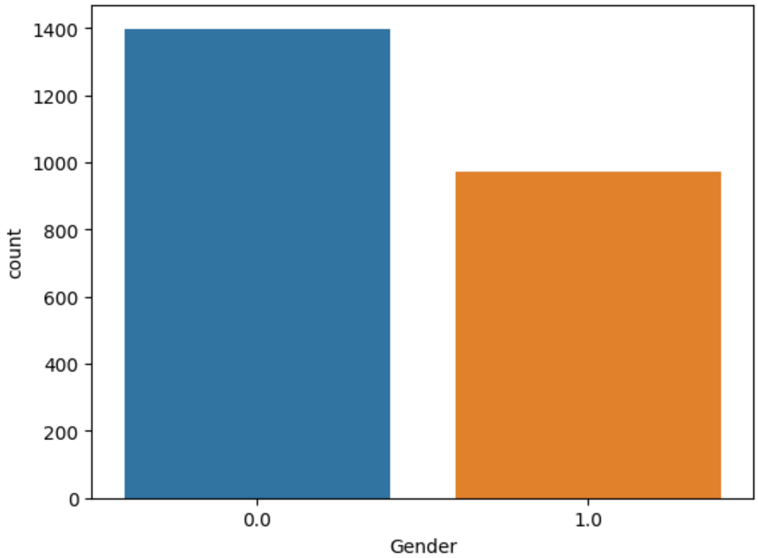
```
plt.pie(df["Churn"].value_counts(), labels = ["Churn", "No Churn"], autopct = "%2.1f%%")
```

```
([<matplotlib.patches.Wedge at 0x7bf79e71e7a0>,
  <matplotlib.patches.Wedge at 0x7bf79e71d660>],
 [Text(-0.5856301893031244, 0.9311483670053805, 'Churn'),
  Text(0.5856301021227194, -0.9311484218360322, 'No Churn')],
 [Text(-0.3194346487107951, 0.5078991092756621, '67.9%'),
  Text(0.3194346011578469, -0.5078991391832902, '32.1%')])
```



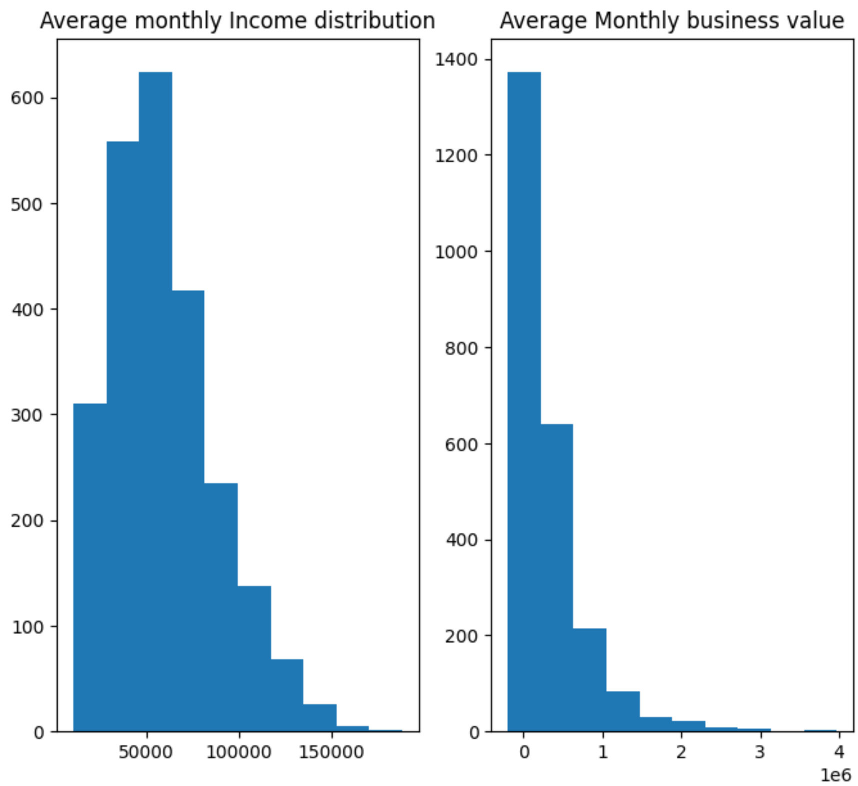
- Data is sligtly imbalanced but workable. 68% of drivers left & 32% of drivers are still working in the comapny

```
sns.countplot(data = df, x = "Gender")
plt.show()

sns.countplot(data = df, x = "City")
plt.xticks(rotation = 90)
plt.show()
```



- Both male & female counts are significant but male's count is slighltly higher than female's count.
- Number of drivers from city C20 is more compared to number of drivers from other cities. And almost equal number of drivers from other cities.

```
# Average monthly Income & Business value distribution
plt.figure(figsize=(8,7))
plt.subplot(1,2,1)
plt.hist(df["Income"])
plt.title("Average monthly Income distribution")

plt.subplot(1,2,2)
plt.hist(df["AvgTotal Business Value"])
plt.title("Average Monthly business value")
plt.show()
```

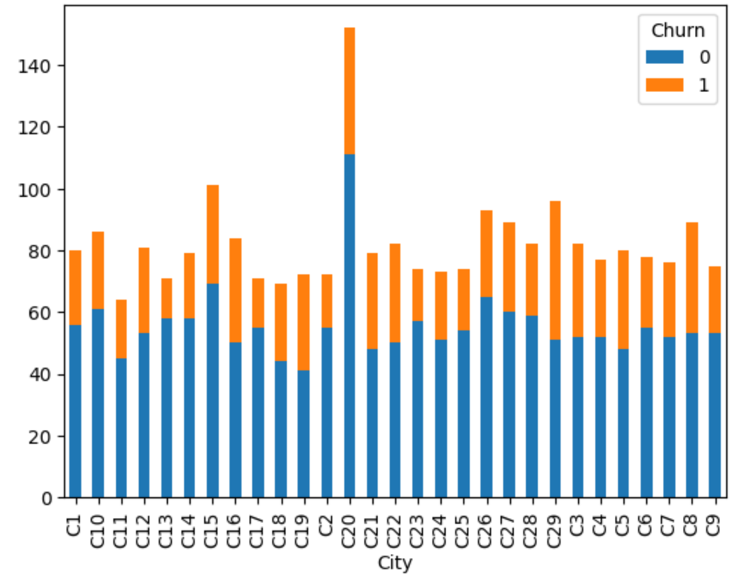


- Average monthly income of most of the drivers lie in the range [30000, 150000]

Bivariate Analysis

```
# Categarical features Vs Target variable
cat = ["City", "Gender", "Grade", "Quarterly Rating"]
for i in range(1, len(cat)+1):
  plt.figure(figsize = (4,4))
  pd.crosstab(df[cat[i-1]], df["Churn"]).plot(kind = "bar", stacked =True)
```

```
<Figure size 400x400 with 0 Axes>
```

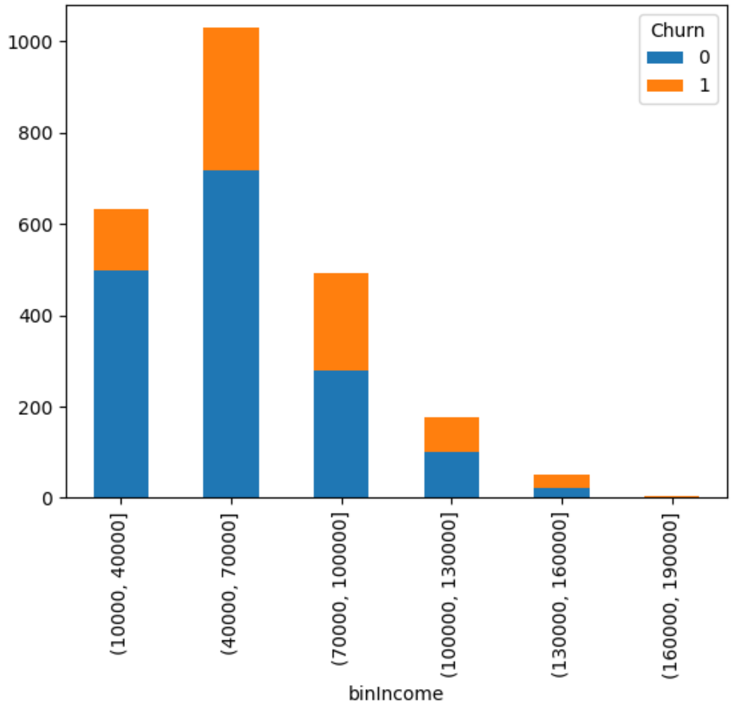



```
<Figure size 400x400 with 0 Axes>
```

- Churn ration is almost same for all the cities that is city column does not significantly impact the target variable.
- Churn rate(Number of drivers leaving the job/total number of drivers belonging to same grade) will be high for drivers having grade 1 and 2.
- Churn rate(Number of drivers leaving the job) will be high for drivers having quarterly rating 1.

```
# Income Vs Churn
df["binIncome"] = pd.cut(df['Income'], bins = [10000, 40000, 70000,100000, 130000,160000,190000])
pd.crosstab(df['binIncome'],df["Churn"]).plot(kind = "bar",stacked = True)
```

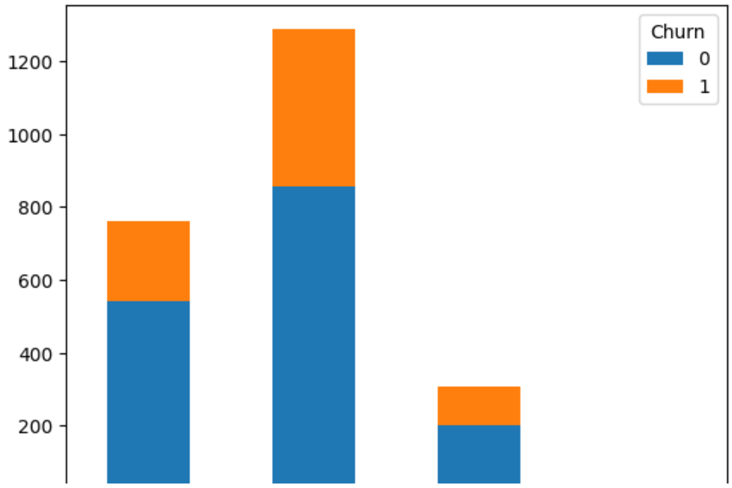
```
<Axes: xlabel='binIncome'>
```



- Churn rate is high among the drivers having average income in the range[10000, 70000]

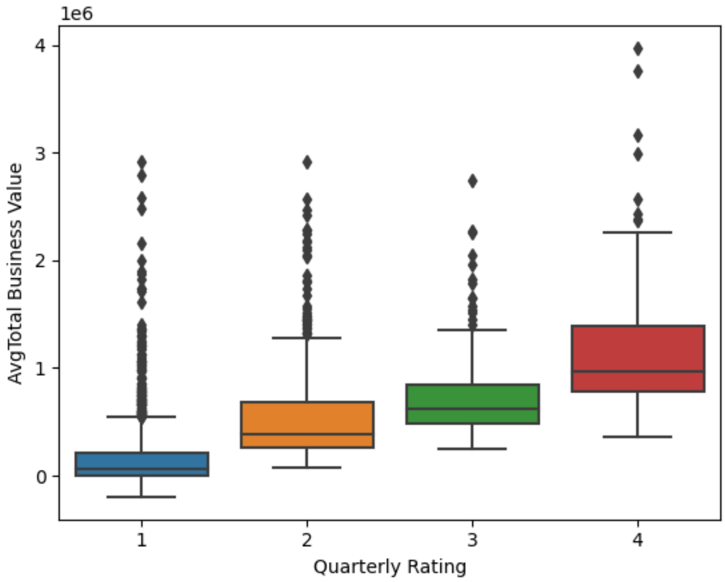
```
# Age Vs Churn
df["Agebin"] = pd.cut(df['Age'], bins = [20, 30, 40, 50, 60])
pd.crosstab(df['Agebin'],df["Churn"]).plot(kind = "bar",stacked = True)
```

<Axes: xlabel='Agebin'>




```
sns.boxplot(data = df,x = "Quarterly Rating", y = "AvgTotal Business Value")
```

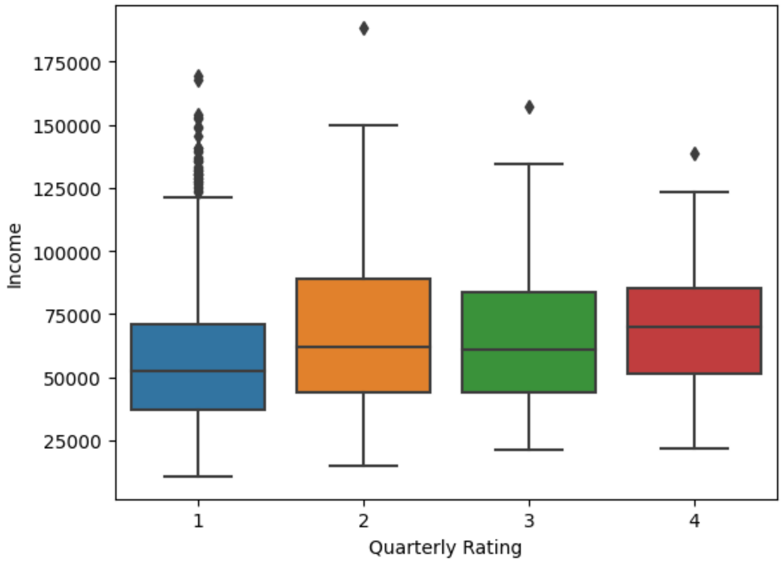
<Axes: xlabel='Quarterly Rating', ylabel='AvgTotal Business Value'>



- Avg business value of drivers & theire quartely rating are related to each other. Average business of value of drivers having high quarterly rating has high average business value.

```
# Continous Vs continous
sns.boxplot(data = df,x = "Quarterly Rating", y = "Income")
```

<Axes: xlabel='Quarterly Rating', ylabel='Income'>



```
df.drop(["binIncome","Agebin"], axis = 1, inplace = True)
```

Data cleaning

Missing value Treatment

KNN Imputation

```
# Gender & Age are the two columns which consists of null values

from sklearn.impute import KNNImputer

columns = ["Age", "Gender"]
df_num = df[["Age", "Gender"]]

Imputer = KNNImputer(n_neighbors = 4, weights = 'uniform', metric = 'nan_euclidean' )
Imputer.fit(df_num)
df_new_num = Imputer.transform(df_num)

df_new_num = pd.DataFrame(df_new_num, columns = columns)
```

```
df[["Age", "Gender"]] = df_new_num
```

```
df.drop("index", axis =1, inplace = True)
```

```
df.head()
```

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining
0	2019-03-01	1	28.0	0.0	C23	2	57387	2018-12-24
1	2020-12-01	2	31.0	0.0	C7	2	67016	2020-11-06
2	2020-04-01	4	43.0	0.0	C13	2	65603	2019-12-07
3	2019-03-01	5	29.0	0.0	C9	0	46368	2019-01-09
4	2020-	6	31.0	1.0	C11	1	78728	2020-07-31

Removing Outliers

```
num = ["Age", "Income", "AvgTotal Business Value"]

def outlierremoval(df,col):
  q1 = np.percentile(df[col], 25)
  q3 = np.percentile(df[col], 75)
  iqr = q3 - q1
  ll = q1 - 1.5*iqr
  ul = q3+1.5*iqr
  df = df[(df[col]>ll )& (df[col]<ul)]
  return df

for col in num:
  df = outlierremoval(df,col)
```

Statistical Summary of derived data after data cleaning

```
df.describe()
```

	Driver_ID	Age	Gender	Education_Level	Income	Jo Designa
count	2158.000000	2158.000000	2158.000000	2158.000000	2158.000000	2158.0
mean	1394.732159	33.159291	0.406627	1.006024	55614.132994	1.8

Data preprocessing

```
df["Dateofjoining"] = df["Dateofjoining"].dt.year
```

	Driver_ID	Age	Gender	Education_Level	Income	Jo Designa
75%	2090.750000	37.000000	1.000000	2.000000	70726.750000	2.0

Feature Engineering

```
# Since reporting year of the all drivers lie in [2019, 2020], it doenot impact the target variable
df.drop("MMM-YY", axis = 1, inplace = True)

# Since churn ration remains almost same for all cities, city is removed
df.drop("City", axis = 1, inplace = True)

#Dropping driverID
df.drop("Driver_ID", axis = 1, inplace = True)
```

Checking of class imbalance

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

```
0    1538
1     620
Name: Churn, dtype: int64
```

- Final derived data distribution is 71%-29%, which is slightly imbalanced but workable.
- But this imbalance will be accounted while bulding the model by hyperparameter tuning.

Decision tree Model

Features and Label

```
y = df["Churn"]
x = df.drop("Churn", axis =1)
```

```
from sklearn.model_selection import train_test_split, KFold, cross_validate, cross_val_score
from sklearn.tree import DecisionTreeClassifier

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,random_state = 7)

tree_clf = DecisionTreeClassifier(random_state = 7)
tree_clf.fit(x_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=7)
```

```
# Accuracy of the model
from sklearn.metrics import accuracy_score, recall_score, precision_score, confusion_matrix
train_acc = accuracy_score(y_train, tree_clf.predict(x_train))
test_acc = accuracy_score(y_test, tree_clf.predict(x_test))

print(f"train_accuracy---->{train_acc}")
print(f"test_accuracy---->{test_acc}")
```

```
train_accuracy---->1.0
test_accuracy---->0.7546296296296297
```

Cross validation - Hyper parameter Tuning

```
from sklearn.model_selection import GridSearchCV

params = {"criterion" :["gini", "entropy"], "max_depth":[2,3,5,7,10],"min_samples_split":[3, 4, 5,]}

grid = GridSearchCV(estimator= DecisionTreeClassifier(), param_grid= params, scoring = 'accuracy', cv = 4,n_jobs =1)
grid.fit(x_train, y_train)

print("Best param:", grid.best_params_)
print("Best score:", grid.best_score_)
```

```
Best param: {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 3}
Best score: 0.8140212791097362
```

Decision treee model with best parameters

```
tree_clf = DecisionTreeClassifier(random_state = 7,max_depth= 3, min_samples_split= 3)
tree_clf.fit(x_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, min_samples_split=3, random_state=7)
```

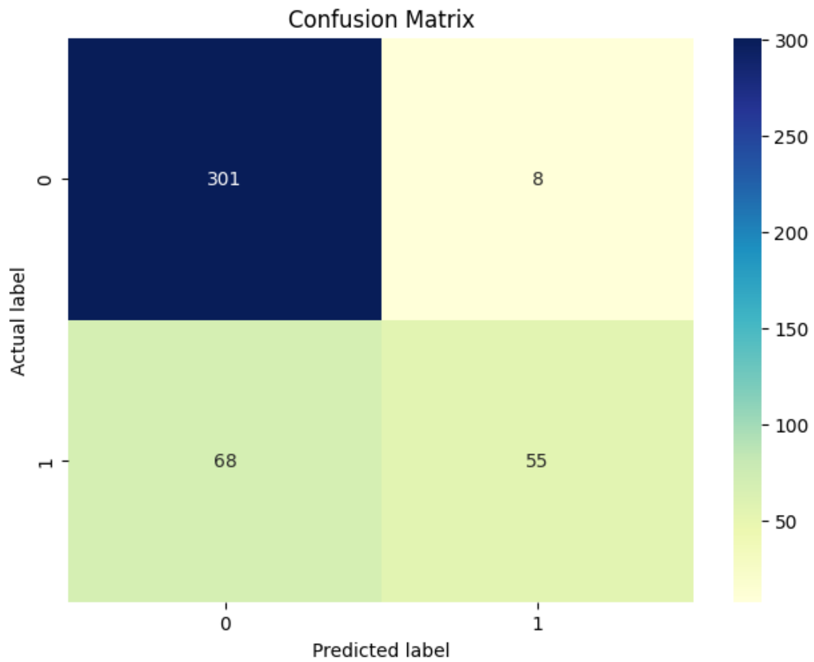
```
train_acc = accuracy_score(y_train, tree_clf.predict(x_train))
test_acc = accuracy_score(y_test, tree_clf.predict(x_test))

print(f"train_accuracy---->{train_acc}")
print(f"test_accuracy---->{test_acc}")
```

```
train_accuracy---->0.8140208574739282
test_accuracy---->0.8240740740740741
```

Confusion matrix

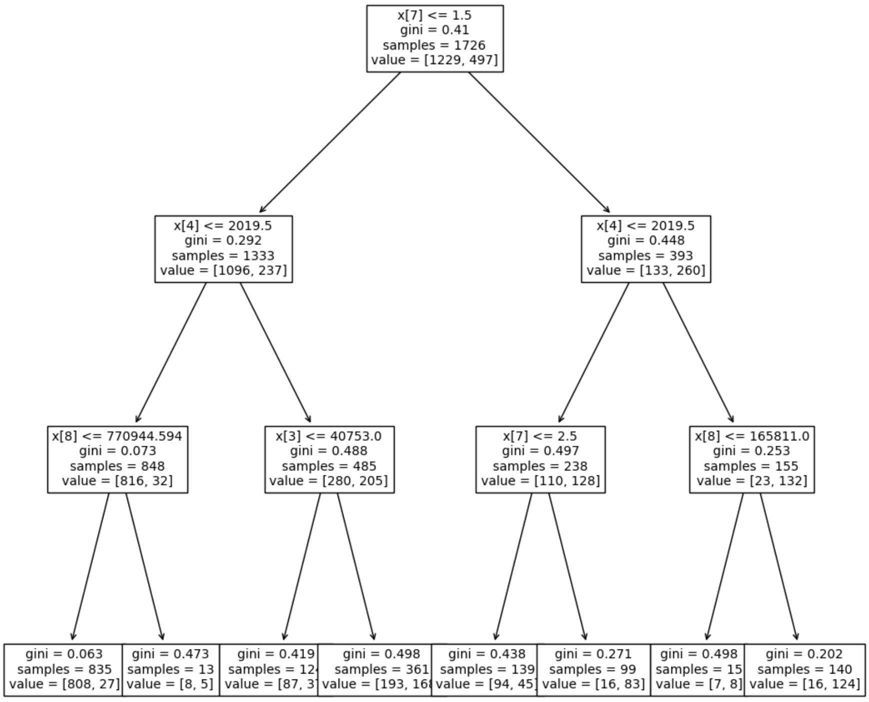
```
cnf_matrix = confusion_matrix(y_test, tree_clf.predict(x_test))
fig, ax = plt.subplots()

# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')

plt.tight_layout()
plt.title('Confusion Matrix')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 23.52222222222222, 'Predicted label')
```



Decision tree plot

```
from sklearn import tree
plt.figure(figsize = (12,12))
tree.plot_tree(tree_clf, fontsize = 10)
plt.show()
```



Class Imbalance treatment

```
x_train_new = x_train
y_train_new = y_train
```

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42, k_neighbors = 3)
X_res, y_res = sm.fit_resample(x_train_new, y_train_new)
```

```
params = {"criterion" :["gini", "entropy"], "max_depth":[9, 10,12,13,15, 17],"min_samples_split":[5, 6, 7, 9,10]}

grid = GridSearchCV(estimator= DecisionTreeClassifier(), param_grid= params, scoring = 'accuracy', cv = 4,n_jobs =1)
grid.fit(X_res, y_res)

print("Best param:", grid.best_params_)
print("Best score:", grid.best_score_)
```

```
Best param: {'criterion': 'gini', 'max_depth': 9, 'min_samples_split': 5}
Best score: 0.8027051720028602
```

```
tree_clf1 = DecisionTreeClassifier(criterion= 'gini',random_state = 7,max_depth= 9, min_samples_split= 6)
tree_clf1.fit(X_res, y_res)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=9, min_samples_split=6, random_state=7)
```

```
train_acc = accuracy_score(y_res, tree_clf1.predict(X_res))
test_acc = accuracy_score(y_test, tree_clf1.predict(x_test))

print(f"train accuracy----->{train_acc}")
print(f"test accuracy----->{test_acc}")
```

```
train accuracy----->0.8950366151342555
test accuracy----->0.7916666666666666
```

Random Forest Classifier

```
# Bulding Random forest model
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(random_state = 7)
rf_clf.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=7)
```

```
train_acc = accuracy_score(y_train, rf_clf.predict(x_train))
test_acc = accuracy_score(y_test, rf_clf.predict(x_test))

print(f"train accuracy --->{train_acc}")
print(f"test accuracy ---->{test_acc}")
```

```
train accuracy --->1.0
test accuracy ---->0.8333333333333334
```

- As shown above, training accuracy of model is 100% & testing accuracy is 83.33% which means model is overfitted. Hence hyper parameter tuning is performed below to avoid overfiting and increase the accuracy.

Cross validation

```
from sklearn.model_selection import GridSearchCV
params = {"n_estimators":[40, 50, 70, 100], "criterion" :["gini", "entropy"], "max_depth":[3,5,7,10], "max_features":[3, 4,

grid = GridSearchCV(estimator = RandomForestClassifier(), param_grid= params, scoring = 'accuracy', cv = 4,n_jobs =1)
grid.fit(x_train, y_train)
```

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

```
print("Best param", grid.best_params_)
print("Best_score", grid.best_score_)
```

```
Best param {'criterion': 'gini', 'max_depth': 3, 'max_features': 3, 'n_estimators': 100}
Best_score 0.8192390113431297
```

```
final_model = RandomForestClassifier(n_estimators = 150, random_state = 7, criterion= 'gini', max_features= 3, max_depth = 4
final_model.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=4, max_features=3, n_estimators=150,
                       random_state=7)
```

```
train_acc = accuracy_score(y_train, final_model.predict(x_train))
test_acc = accuracy_score(y_test, final_model.predict(x_test))

print(f"train accuracy --->{train_acc}")
print(f"test accuracy ---->{test_acc}")
```

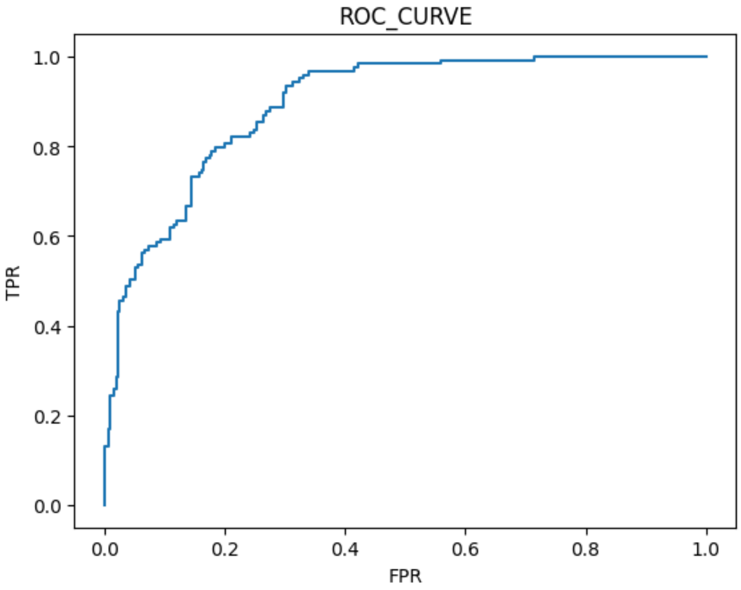
```
train accuracy --->0.8232908458864426
test accuracy ---->0.8240740740740741
```

ROC curve

```
from sklearn.metrics import roc_auc_score, precision_score, roc_curve, precision_recall_curve
```

```
prob = final_model.predict_proba(x_test)
```

```
prob1 = prob[:,1]

fpr, tpr, thresh = roc_curve(y_test, prob1)
plt.plot(fpr, tpr)
plt.title("ROC_CURVE")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



Area Under ROC curve

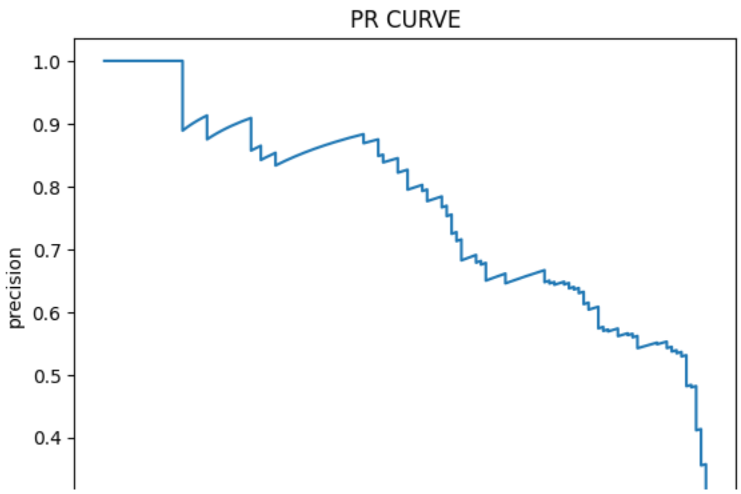
```
# Area under ROC curve

roc_auc_score(y_test, prob1)
```

```
0.8929144631252137
```

- roc_auc_score is 0.89 which tells that model is good

Precision Recall curve

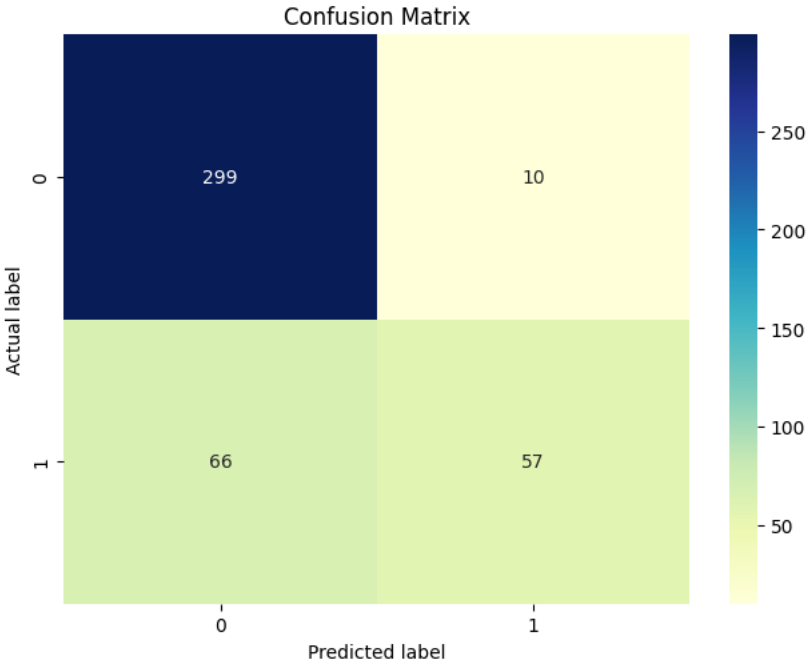
```
p,r,t = precision_recall_curve(y_test, prob1)
plt.plot(r,p)
plt.title("PR CURVE")
plt.xlabel("recall")
plt.ylabel("precision")
plt.show()
```



confusion matrix

```
cnf_matrix = confusion_matrix(y_test, final_model.predict(x_test))
fig, ax = plt.subplots()

# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')

plt.tight_layout()
plt.title('Confusion Matrix')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 23.52222222222222, 'Predicted label')
```



Accuracy & Precision

```
print(f"Accuracy is ------> {accuracy_score(y_test, final_model.predict(x_test))}")
print(f"Precision is ------> {precision_score(y_test, final_model.predict(x_test))}")
```

```
Accuracy is ------> 0.8240740740740741
Precision is ------> 0.8507462686567164
```