Problem Statement - To predict drivers that have good probability of leaving based on the driver's details. Churned Driver can lead to Churned Customers as if no of drivers are less in area, finding cabs can be difficult. Also, finding new drivers (by Ola) is more costly than retaining old ones.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixD
import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)
```

```
!gdown 1qQGMqO8uN217NfpMZkpOpYLP6hgbvxjA
```

```
Downloading...
From: https://drive.google.com/uc?id=1qQGMqO8uN217NfpMZkpOpYLP6hgbvxjA
To: /content/ola.csv
100% 1.13M/1.13M [00:00<00:00, 123MB/s]
```

```
df = pd.read_csv('ola.csv')
df
```

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

19104 rows × 14 columns

Column Profiling:

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

EDA

```
df_2 = df.drop(['Unnamed: 0'], axis=1, inplace=False)
df_2['Churn'] = df_2['LastWorkingDate'].apply(lambda x: 0 if pd.isna(x) else 1) # when we aggregate, we will max this value
df_2
```

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

```
df=df_2.copy()
```

```
df.info()
```

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

```
df.describe(include=['object','int64','float64'])
```

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate
count	19104	19104.000000	19043.000000	19052.000000	19104	19104.000000	19104.000000	19104	1616
unique	24	NaN	NaN	NaN	29	NaN	NaN	869	493
top	01/01/19	NaN	NaN	NaN	C20	NaN	NaN	23/07/15	29/07/20
freq	1022	NaN	NaN	NaN	1008	NaN	NaN	192	70
mean	NaN	1415.591133	34.668435	0.418749	NaN	1.021671	65652.025126	NaN	NaN
std	NaN	810.705321	6.257912	0.493367	NaN	0.800167	30914.515344	NaN	NaN
min	NaN	1.000000	21.000000	0.000000	NaN	0.000000	10747.000000	NaN	NaN
25%	NaN	710.000000	30.000000	0.000000	NaN	0.000000	42383.000000	NaN	NaN
50%	NaN	1417.000000	34.000000	0.000000	NaN	1.000000	60087.000000	NaN	NaN
75%	NaN	2137.000000	39.000000	1.000000	NaN	2.000000	83969.000000	NaN	NaN
max	NaN	2788.000000	58.000000	1.000000	NaN	2.000000	188418.000000	NaN	NaN

```
df['Driver_ID'].nunique()
```

    2381

```
df['City'].nunique()
```

    29

```
# we see for each driver_id multiple rows of data is present for different months
```

```
df[df['Driver_ID'] == 1]
```

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	To Busin Va
0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	2381
1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	-665

```
df.corr()
```

	Driver_ID	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	Churn
Driver_ID	1.000000	0.005457	0.030349	-0.016132	-0.035767	-0.035166	-0.025712	0.003896	0.017917	-0.000675
Age	0.005457	1.000000	0.040261	-0.010245	0.191112	-0.006641	0.210702	0.108835	0.171818	-0.063562
Gender	0.030349	0.040261	1.000000	-0.010123	0.013229	-0.050878	0.002076	0.008909	0.008099	-0.002908
Education_Level	-0.016132	-0.010245	-0.010123	1.000000	0.115008	0.002041	-0.039552	-0.007504	0.026064	-0.007058
Income	-0.035767	0.191112	0.013229	0.115008	1.000000	0.380878	0.778383	0.234044	0.116897	-0.100896
Joining Designation	-0.035166	-0.006641	-0.050878	0.002041	0.380878	1.000000	0.559854	-0.044446	-0.237791	0.020249
Grade	-0.025712	0.210702	0.002076	-0.039552	0.778383	0.559854	1.000000	0.220955	0.014445	-0.089486
Total Business	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Feature Engineering

```
def increase_fn(x):
  if len(x) >= 2:
    for i in range(len(x)):
      if x[-1] > x[-2]:
        return 1
      else:
        return 0
  else:
    return 0
```

```
df_4 = df.sort_values(by=['Driver_ID', 'MMM-YY'])
df_4 = df_4.reset_index(drop=True)
df_4
```

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	B
0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	

```
df_2 = pd.merge(left = df_4.groupby("Driver_ID")["Quarterly Rating"].unique().apply(increase_fn).rename("Quarterly_Rating_Incr
            right = df_4,
            on = "Driver_ID",
            how="outer")
df_3 = pd.merge(left = df_2.groupby("Driver_ID")["Income"].unique().apply(increase_fn).rename("Income_Increased"),
            right = df_2,
            on = "Driver_ID",
            how="outer")
df_3
```

	Driver_ID	Income_Increased	Quarterly_Rating_Increased	MMM-YY	Age	Gender	City	Education_Level	Income	Dateofj
0	1	0	0	01/01/19	28.0	0.0	C23	2	57387	
1	1	0	0	02/01/19	28.0	0.0	C23	2	57387	
2	1	0	0	03/01/19	28.0	0.0	C23	2	57387	
3	2	0	0	11/01/20	31.0	0.0	C7	2	67016	
4	2	0	0	12/01/20	31.0	0.0	C7	2	67016	
...	...	...	...	...	...	...	...	...	...	
19099	2788	0	0	08/01/20	30.0	0.0	C27	2	70254	
19100	2788	0	0	09/01/20	30.0	0.0	C27	2	70254	
19101	2788	0	0	10/01/20	30.0	0.0	C27	2	70254	
19102	2788	0	0	11/01/20	30.0	0.0	C27	2	70254	
19103	2788	0	0	12/01/20	30.0	0.0	C27	2	70254	

19104 rows × 16 columns

```
df=df_3.copy()
```

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

```
0    12442
1     6662
Name: Quarterly_Rating_Increased, dtype: int64
```

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

```
0    18114
1      990
Name: Income_Increased, dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 19104 entries, 0 to 19103
Data columns (total 16 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Driver_ID                   19104 non-null  int64  
 1   Income_Increased            19104 non-null  int64  
 2   Quarterly_Rating_Increased  19104 non-null  int64  
 3   MMM-YY                      19104 non-null  object 
 4   Age                         19043 non-null  float64
 5   Gender                      19052 non-null  float64
 6   City                        19104 non-null  object 
 7   Education_Level             19104 non-null  int64  
 8   Income                      19104 non-null  int64  
 9   Dateofjoining               19104 non-null  object 
 10  LastWorkingDate             1616 non-null   object 
 11  Joining Designation         19104 non-null  int64  
 12  Grade                       19104 non-null  int64  
 13  Total Business Value        19104 non-null  int64  
 14  Quarterly Rating            19104 non-null  int64  
 15  Churn                       19104 non-null  int64  
dtypes: float64(2), int64(10), object(4)
memory usage: 2.5+ MB
```

```
df['dt'] = pd.to_datetime(df['MMM-YY'])
df['last_dt'] = pd.to_datetime(df['LastWorkingDate'])
df['joining_dt'] = pd.to_datetime(df['Dateofjoining'])
df.head(1)
```

	Driver_ID	Income_Increased	Quarterly_Rating_Increased	MMM-YY	Age	Gender	City	Education_Level	Income	Dateofjoin
0	1	0	0	01/01/19	28.0	0.0	C23	2	57387	24/12

```
df.drop(['MMM-YY', 'LastWorkingDate', 'Dateofjoining'], axis=1, inplace=True)
```

```
df.head(5)
```

	Driver_ID	Income_Increased	Quarterly_Rating_Increased	Age	Gender	City	Education_Level	Income	Joining Designation	Grade
0	1	0	0	28.0	0.0	C23	2	57387	1	1
1	1	0	0	28.0	0.0	C23	2	57387	1	1
2	1	0	0	28.0	0.0	C23	2	57387	1	1
3	2	0	0	31.0	0.0	C7	2	67016	2	2
4	2	0	0	31.0	0.0	C7	2	67016	2	2

Feature Engineering

```
# lets do aggregation now
df_agg = df.groupby(["Driver_ID"]).agg({"dt": [len,'max'],
                                          "Age": 'max',
                                          "Gender": 'last',
                                          "City": 'last',
                                          "Education_Level": 'max',
                                          "Income": ['mean','max'],
                                          "Income_Increased": ['sum','max'],
                                          "joining_dt": 'min',
                                          "last_dt": 'max',
                                          "Joining Designation": ['min','max'],
                                          "Grade": ['mean', 'max', 'min'],
                                          "Total Business Value": ['max', 'mean','std'],
                                          "Quarterly Rating": ['mean','max', 'min'],
                                          "Quarterly_Rating_Increased": ['sum','max'],
                                          "Churn":'max'
                                    }).reset_index()

# Rename columns for clarity
df_agg.columns = [' '.join(col).strip() for col in df_agg.columns.values]
df_agg
```

	Driver_ID	dt len	dt max	Age max	Gender last	City last	Education_Level max	Income mean	Income max	Income_Increased sum	Income_Increased max	joining_ r
0	1	3	2019-03-01	28.0	0.0	C23	2	57387.0	57387	0	0	2018-12
1	2	2	2020-12-01	31.0	0.0	C7	2	67016.0	67016	0	0	2020-11
2	4	5	2020-04-01	43.0	0.0	C13	2	65603.0	65603	0	0	2019-12
3	5	3	2019-03-01	29.0	0.0	C9	0	46368.0	46368	0	0	2019-01
[illegible]	[illegible]	[illegible]	2020-	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

```
# add more features like time_before_leaving days & time_in_job for those who didn't leave
df_agg['time_before_leaving'] = df_agg['last_dt max'] - df_agg['joining_dt min']
df_agg['time_in_job'] = df_agg['dt max'] - df_agg['joining_dt min']
df_agg
```

	Driver_ID	dt len	dt max	Age max	Gender last	City last	Education_Level max	Income mean	Income max	Income_Increased sum	Income_Increased max	joining_ r
0	1	3	2019-03-01	28.0	0.0	C23	2	57387.0	57387	0	0	2018-12
1	2	2	2020-12-01	31.0	0.0	C7	2	67016.0	67016	0	0	2020-11
2	4	5	2020-04-01	43.0	0.0	C13	2	65603.0	65603	0	0	2019-12
3	5	3	2019-03-01	29.0	0.0	C9	0	46368.0	46368	0	0	2019-01
4	6	5	2020-12-01	31.0	1.0	C11	1	78728.0	78728	0	0	2020-07
...	...	...	...	...	...	...	...	...	...	...	...	
2376	2784	24	2020-12-01	34.0	0.0	C24	0	82815.0	82815	0	0	2015-10
2377	2785	3	2020-10-01	34.0	1.0	C9	0	12105.0	12105	0	0	2020-08
2378	2786	9	2019-09-01	45.0	0.0	C19	0	35370.0	35370	0	0	2018-07
2379	2787	6	2019-06-01	28.0	1.0	C20	2	69498.0	69498	0	0	2018-07
2380	2788	7	2020-12-01	30.0	0.0	C27	2	70254.0	70254	0	0	2020-06

2381 rows × 29 columns

```
# extracted info from dates, drop dates now
df_agg['job_time'] = np.where(df_agg['time_before_leaving'].isna(), df_agg['time_in_job'], df_agg['time_before_leaving'])
df_agg_2 = df_agg.drop(['time_before_leaving', 'time_in_job', 'joining_dt min', 'last_dt max', 'dt max'], axis=1, inplace=Fals
df_agg_2
```

	Driver_ID	dt len	Age max	Gender last	City last	Education_Level max	Income mean	Income max	Income_Increased sum	Income_Increased max	Joining Designation min
0	1	3	28.0	0.0	C23	2	57387.0	57387	0	0	1

```
df_agg_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 25 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Driver_ID                       2381 non-null   int64
 1   dt len                          2381 non-null   int64
 2   Age max                         2381 non-null   float64
 3   Gender last                     2381 non-null   float64
 4   City last                       2381 non-null   object
 5   Education_Level max             2381 non-null   int64
 6   Income mean                     2381 non-null   float64
 7   Income max                      2381 non-null   int64
 8   Income_Increased sum            2381 non-null   int64
 9   Income_Increased max            2381 non-null   int64
 10  Joining Designation min         2381 non-null   int64
 11  Joining Designation max         2381 non-null   int64
 12  Grade mean                      2381 non-null   float64
 13  Grade max                       2381 non-null   int64
 14  Grade min                       2381 non-null   int64
 15  Total Business Value max        2381 non-null   int64
 16  Total Business Value mean       2381 non-null   float64
 17  Total Business Value std        2200 non-null   float64
 18  Quarterly Rating mean           2381 non-null   float64
 19  Quarterly Rating max            2381 non-null   int64
 20  Quarterly Rating min            2381 non-null   int64
 21  Quarterly_Rating_Increased sum  2381 non-null   int64
 22  Quarterly_Rating_Increased max  2381 non-null   int64
 23  Churn max                       2381 non-null   int64
 24  job_time                        2381 non-null   timedelta64[ns]
dtypes: float64(7), int64(16), object(1), timedelta64[ns](1)
memory usage: 465.2+ KB
```

```
df_agg_2['job_days'] = df_agg_2['job_time']/pd.Timedelta(days=1)
df_agg_3 = df_agg_2.drop(['job_time'], axis=1, inplace=False)
df_agg_3
```

	Driver_ID	dt len	Age max	Gender last	City last	Education_Level max	Income mean	Income max	Income_Increased sum	Income_Increased max	Joining Designation min
0	1	3	28.0	0.0	C23	2	57387.0	57387	0	0	1
1	2	2	31.0	0.0	C7	2	67016.0	67016	0	0	2
2	4	5	43.0	0.0	C13	2	65603.0	65603	0	0	2
3	5	3	29.0	0.0	C9	0	46368.0	46368	0	0	1
4	6	5	31.0	1.0	C11	1	78728.0	78728	0	0	3
...	...	...	...	...	...	...	...	...	...	...	...
2376	2784	24	34.0	0.0	C24	0	82815.0	82815	0	0	2
2377	2785	3	34.0	1.0	C9	0	12105.0	12105	0	0	1
2378	2786	9	45.0	0.0	C19	0	35370.0	35370	0	0	2
2379	2787	6	28.0	1.0	C20	2	69498.0	69498	0	0	1
2380	2788	7	30.0	0.0	C27	2	70254.0	70254	0	0	2

2381 rows × 25 columns

```
df_final=df_agg_3.copy()
```

```
df_final.describe()
```

	Driver_ID	dt len	Age max	Gender last	Education_Level max	Income mean	Income max	Income_Increased sum	Income_I
count	2381.000000	2381.00000	2381.000000	2381.000000	2381.00000	2381.000000	2381.000000	2381.000000	23
mean	1397.559009	8.02352	33.663167	0.410332	1.00756	59232.460484	59336.159597	0.415792	
std	806.161628	6.78359	5.983375	0.491997	0.81629	28298.214012	28383.012146	3.062127	
min	1.000000	1.00000	21.000000	0.000000	0.00000	10747.000000	10747.000000	0.000000	

```
df_final.describe(include='object')
```

	City last
count	2381
unique	29
top	C20
freq	152

```
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 25 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Driver_ID                       2381 non-null   int64
 1   dt len                          2381 non-null   int64
 2   Age max                         2381 non-null   float64
 3   Gender last                     2381 non-null   float64
 4   City last                       2381 non-null   object
 5   Education_Level max             2381 non-null   int64
 6   Income mean                     2381 non-null   float64
 7   Income max                      2381 non-null   int64
 8   Income_Increased sum            2381 non-null   int64
 9   Income_Increased max            2381 non-null   int64
 10  Joining Designation min         2381 non-null   int64
 11  Joining Designation max         2381 non-null   int64
 12  Grade mean                      2381 non-null   float64
 13  Grade max                       2381 non-null   int64
 14  Grade min                       2381 non-null   int64
 15  Total Business Value max        2381 non-null   int64
 16  Total Business Value mean       2381 non-null   float64
 17  Total Business Value std        2200 non-null   float64
 18  Quarterly Rating mean           2381 non-null   float64
 19  Quarterly Rating max            2381 non-null   int64
 20  Quarterly Rating min            2381 non-null   int64
 21  Quarterly_Rating_Increased sum  2381 non-null   int64
 22  Quarterly_Rating_Increased max  2381 non-null   int64
 23  Churn max                       2381 non-null   int64
 24  job_days                        2381 non-null   float64
dtypes: float64(8), int64(16), object(1)
memory usage: 465.2+ KB
```

```
# null value treatment
# df_final['Total Business Value std'].fillna(0, inplace=True)

from sklearn.impute import SimpleImputer, KNNImputer
imputer = KNNImputer(n_neighbors = 5)

missing_data_columns = ['Total Business Value std']

for i in missing_data_columns:
  df_final[i] = imputer.fit_transform(df_final[i].values.reshape(-1, 1))

df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 25 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Driver_ID                       2381 non-null   int64
 1   dt len                          2381 non-null   int64
 2   Age max                         2381 non-null   float64
 3   Gender last                     2381 non-null   float64
 4   City last                       2381 non-null   object
 5   Education_Level max             2381 non-null   int64
 6   Income mean                     2381 non-null   float64
 7   Income max                      2381 non-null   int64
 8   Income_Increased sum            2381 non-null   int64
```

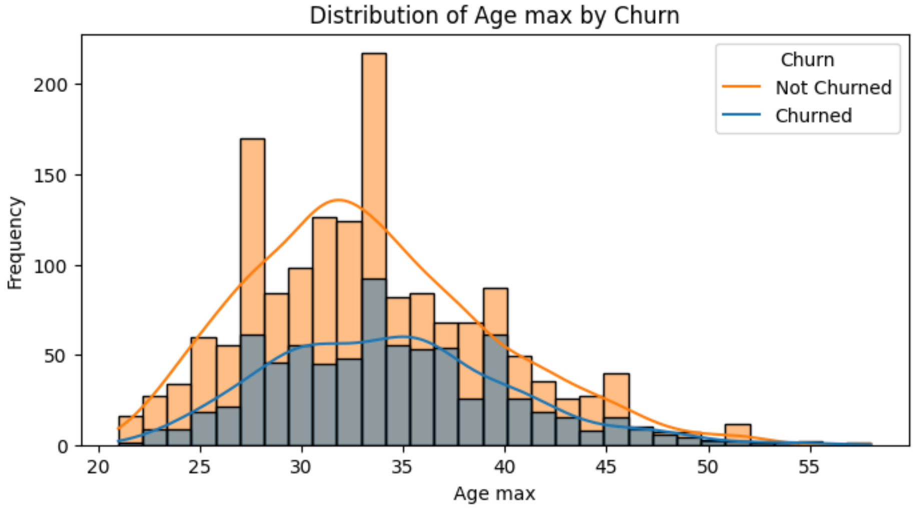
```
 9   Income_Increased max            2381 non-null   int64
 10  Joining Designation min         2381 non-null   int64
 11  Joining Designation max         2381 non-null   int64
 12  Grade mean                      2381 non-null   float64
 13  Grade max                       2381 non-null   int64
 14  Grade min                       2381 non-null   int64
 15  Total Business Value max        2381 non-null   int64
 16  Total Business Value mean       2381 non-null   float64
 17  Total Business Value std        2381 non-null   float64
 18  Quarterly Rating mean           2381 non-null   float64
 19  Quarterly Rating max            2381 non-null   int64
 20  Quarterly Rating min            2381 non-null   int64
 21  Quarterly_Rating_Increased sum  2381 non-null   int64
 22  Quarterly_Rating_Increased max  2381 non-null   int64
 23  Churn max                       2381 non-null   int64
 24  job_days                        2381 non-null   float64
dtypes: float64(8), int64(16), object(1)
memory usage: 465.2+ KB
```

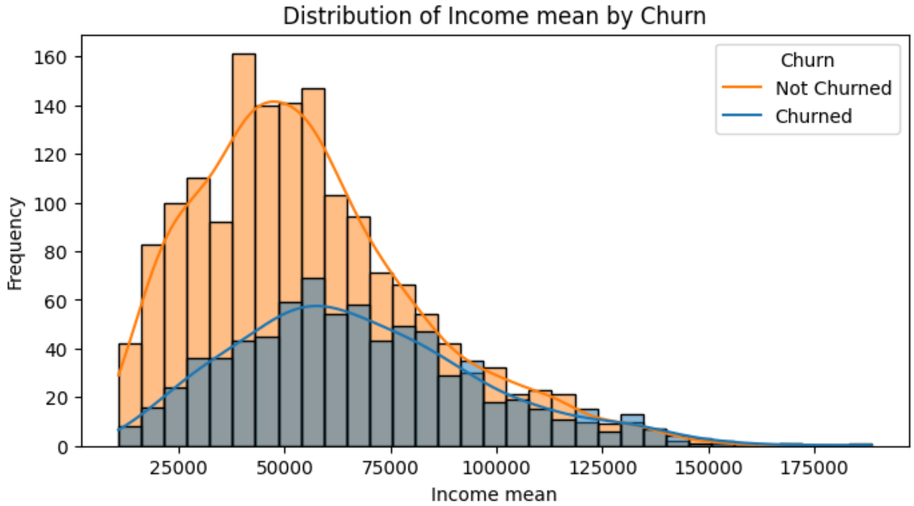
```
df_final['Churn max'].value_counts()
```

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

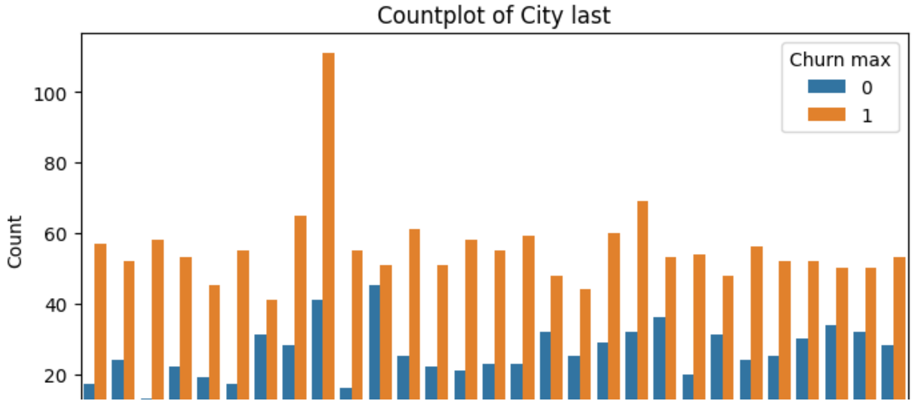
Univariate & BiVariate Analysis

```
# Univariate Analysis (Continuous Variables)
continuous_vars = ['Age max', 'Income mean', 'Grade mean', 'job_days']
for col in continuous_vars:
    plt.figure(figsize=(8, 4))
    sns.histplot(data=df_final, x=col, kde=True, hue='Churn max')
    plt.title(f'Distribution of {col} by Churn')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.legend(title='Churn', labels=['Not Churned', 'Churned'])
    plt.show()
```

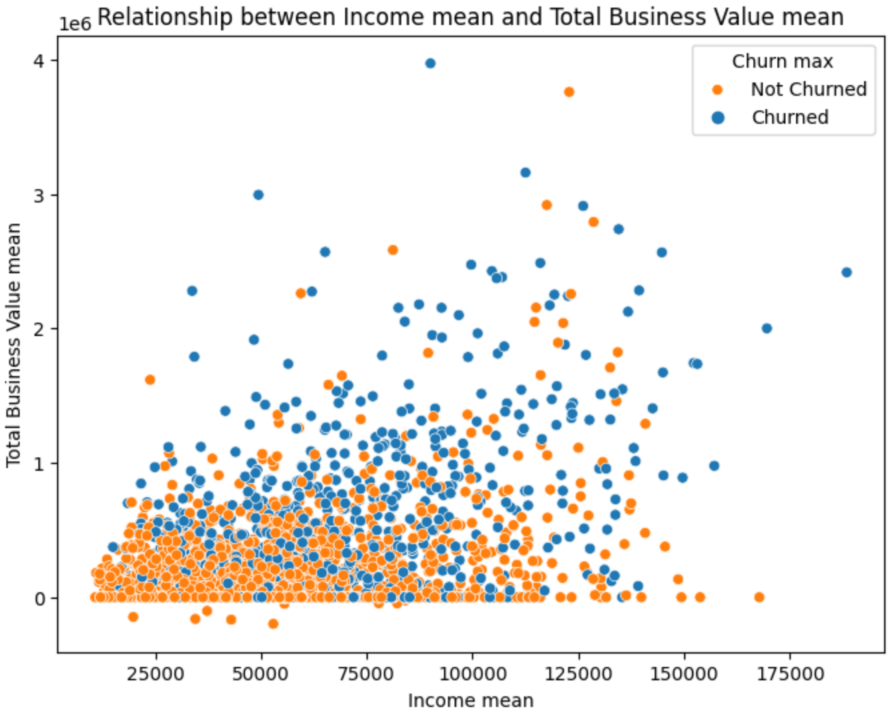




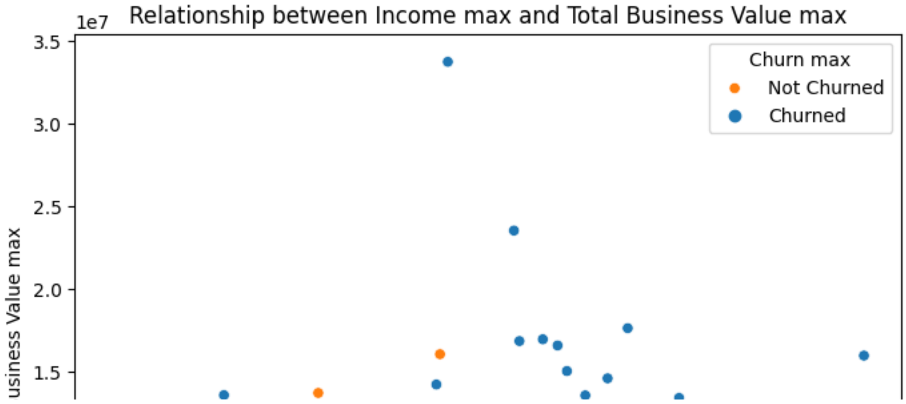

```
# Univariate Analysis (Categorical Variables)
categorical_vars = ['City last', 'Education_Level max']
for col in categorical_vars:
    plt.figure(figsize=(8, 4))
    sns.countplot(data=df_final, x=col, hue='Churn max')
    plt.title(f'Countplot of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45)
    plt.show()
```



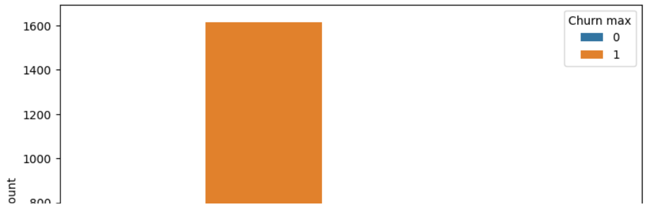

```
# Bivariate Analysis (Relationships between important variables)
# Relationship between 'Income mean' and 'Total Business Value mean'
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df_final, x='Income mean', y='Total Business Value mean', hue='Churn max')
plt.title('Relationship between Income mean and Total Business Value mean')
plt.xlabel('Income mean')
plt.ylabel('Total Business Value mean')
plt.legend(title='Churn max', loc='upper right', labels=['Not Churned', 'Churned'])
plt.show()
```

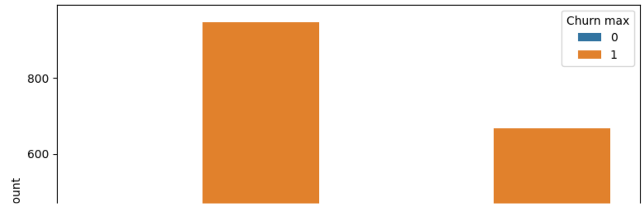



```
# Bivariate Analysis (Relationships between important variables)
# Relationship between 'Income sum' and 'Total Business Value sum'
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df_final, x='Income max', y='Total Business Value max', hue='Churn max')
plt.title('Relationship between Income max and Total Business Value max')
plt.xlabel('Income max')
plt.ylabel('Total Business Value max')
plt.legend(title='Churn max', loc='upper right', labels=['Not Churned', 'Churned'])
plt.show()
```



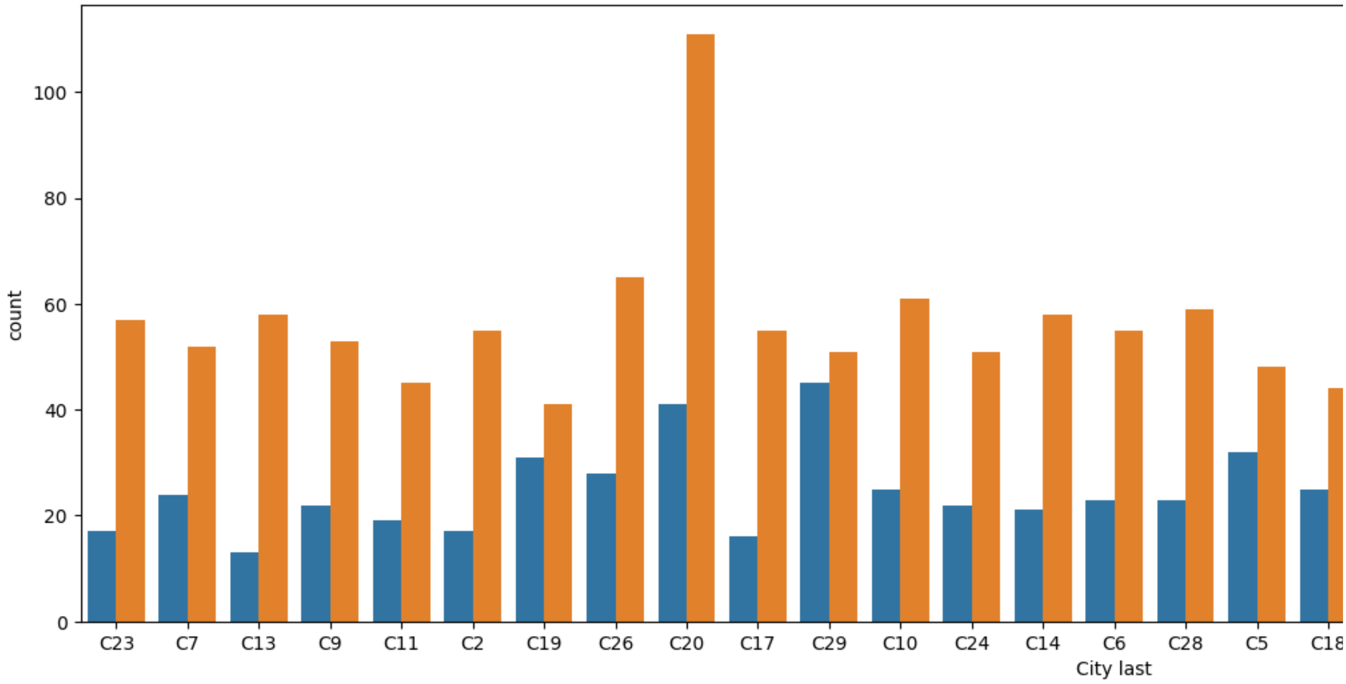

```
features = ['Income_Increased max', 'Gender last', 'Education_Level max', 'Joining Designation mean', 'Quarterly_Rating_Increa

fig, axs = plt.subplots(3, 2, figsize = (20, 20))
for i, ax in zip(features, axs.ravel()):
  sns.countplot(data = df_final, x = i, hue = 'Churn max', ax = ax)
```

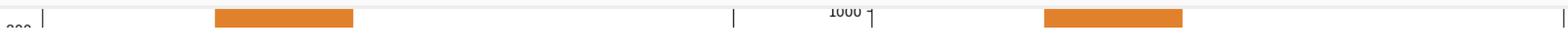





```
plt.figure(figsize = (20, 6))
sns.countplot(data = df_final, x = 'City last', hue = 'Churn max')
```

```
<Axes: xlabel='City last', ylabel='count'>
```





Data Insights from above graphs

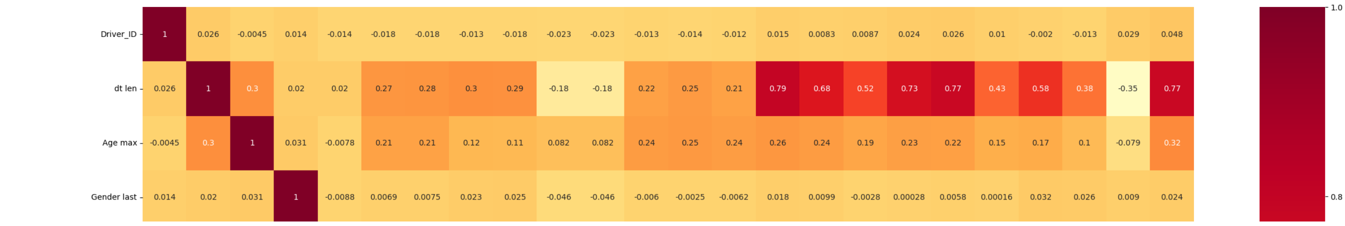
- Churn customers are more in dataset, nearly twice the non churned customers.
- Ages are majorly from 25 to 40. Age 35+ years are more likely to Churn.
- mostly drivers have income 25000 to 100000. From incomes > 75000, churn customers are seen in more percentage
- we see much more churned customers after 2 years (700 days)
- ratio of male to female drivers is 60:40
- more income & less business value also increases chances of Churn
- quarterly ratings & income is increased recently for Churn customers
- joining designation 1 is more prone to Churn while joining designation 3 retains the drivers more.
- Cities like C20 & C17 have concerning churn rates. Ola should try to retain drivers here.
- High business value drivers have mostly churned which is concerning. Income should be increased proportionately for them.

```
# we notice c20 & c17 city ids have high churn rate %
# lets encode city to target variable churn

city_churn_means = df_final.groupby('City last')['Churn max'].mean().reset_index()
df = df_final.merge(city_churn_means, on='City last', how='left')
df.rename(columns={'Churn max_x': 'Churn', 'Churn max_y': 'city_encoded'}, inplace=True)
df.drop(['City last','Driver_ID'], axis=1, inplace=True)
df
```

	dt len	Age max	Gender last	Education_Level max	Income mean	Income max	Income_Increased sum	Income_Increased max	Joining Designation min	Joining Designation max	Grad mea
0	3	28.0	0.0	2	57387.0	57387	0	0	1	1	1.(
1	2	31.0	0.0	2	67016.0	67016	0	0	2	2	2.(
2	5	43.0	0.0	2	65603.0	65603	0	0	2	2	2.(
3	3	29.0	0.0	0	46368.0	46368	0	0	1	1	1.(
4	5	31.0	1.0	1	78728.0	78728	0	0	3	3	3.(

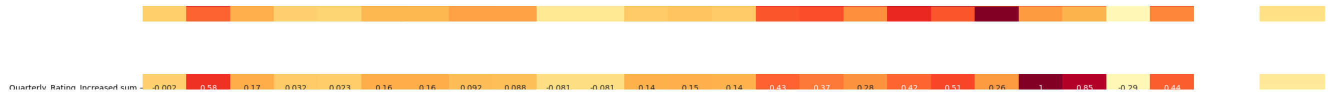
```
plt.figure(figsize=(30, 30))
sns.heatmap(df_final.corr(), annot = True, cmap = "YlOrRd", annot_kws = {"size": 10})
```

```
<Axes: >
```




```
# all are numerical features now
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2381 entries, 0 to 2380
Data columns (total 24 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   dt len                          2381 non-null   int64  
 1   Age max                         2381 non-null   float64
 2   Gender last                     2381 non-null   float64
 3   Education_Level max             2381 non-null   int64  
 4   Income mean                     2381 non-null   float64
 5   Income max                      2381 non-null   int64  
 6   Income_Increased sum            2381 non-null   int64  
 7   Income_Increased max            2381 non-null   int64  
 8   Joining Designation min         2381 non-null   int64  
 9   Joining Designation max         2381 non-null   int64  
 10  Grade mean                      2381 non-null   float64
 11  Grade max                       2381 non-null   int64  
 12  Grade min                       2381 non-null   int64  
 13  Total Business Value sum        2381 non-null   int64  
 14  Total Business Value mean       2381 non-null   float64
 15  Total Business Value std        2381 non-null   float64
 16  Quarterly Rating mean           2381 non-null   float64
 17  Quarterly Rating max            2381 non-null   int64  
 18  Quarterly Rating min            2381 non-null   int64  
 19  Quarterly_Rating_Increased sum  2381 non-null   int64  
 20  Quarterly_Rating_Increased max  2381 non-null   int64  
 21  Churn                           2381 non-null   int64  
 22  job_days                        2381 non-null   float64
 23  city_encoded                    2381 non-null   float64
dtypes: float64(9), int64(15)
memory usage: 465.0 KB
```



Data Preprocessing

- KNN Imputation - done on Total Business Value std later in notebook
- Feature Engineering - derived many features from aggregated driver data
- Class Imbalance treatment - used SMOT & algorith level hyperparameter (scale_pos_weight)
- Standardization - done as preprocessing to get X & y features
- Encoding - Target encoding done on city

[ ] ↳ *22 cells hidden*

Model building using:

- Logistic Regression
- Decision Tree (DT)
- Random Forest (RF)
- Random Forest with gridsearch CV for hyperparameter tuning
- Gradient Boosting DT (GBDT)
- GBDT with randomisedsearch CV for hyperparameter tuning
- LightGBM with randomisedsearch CV for hyperparameter tuning
- **Result evaluation done after each Model**

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Create and train the logistic regression model
```

```
logistic_regression_model = LogisticRegression()
logistic_regression_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = logistic_regression_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:')
print(conf_matrix)
print('Classification Report:')
print(classification_rep)
```

```
Accuracy: 0.79
Confusion Matrix:
[[ 81  72]
 [ 29 295]]
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.53      0.62       153
           1       0.80      0.91      0.85       324

    accuracy                           0.79       477
   macro avg       0.77      0.72      0.73       477
weighted avg       0.78      0.79      0.78       477
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
# Create and train the Decision Tree classifier
decision_tree_model = DecisionTreeClassifier(random_state=42)
decision_tree_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = decision_tree_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:')
print(conf_matrix)
print('Classification Report:')
print(classification_rep)
```

```
Accuracy: 0.82
Confusion Matrix:
[[104  49]
 [ 35 289]]
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.68      0.71       153
           1       0.86      0.89      0.87       324

    accuracy                           0.82       477
   macro avg       0.80      0.79      0.79       477
weighted avg       0.82      0.82      0.82       477
```

Random Forest (DT gave recall of 90%, lets try RF)

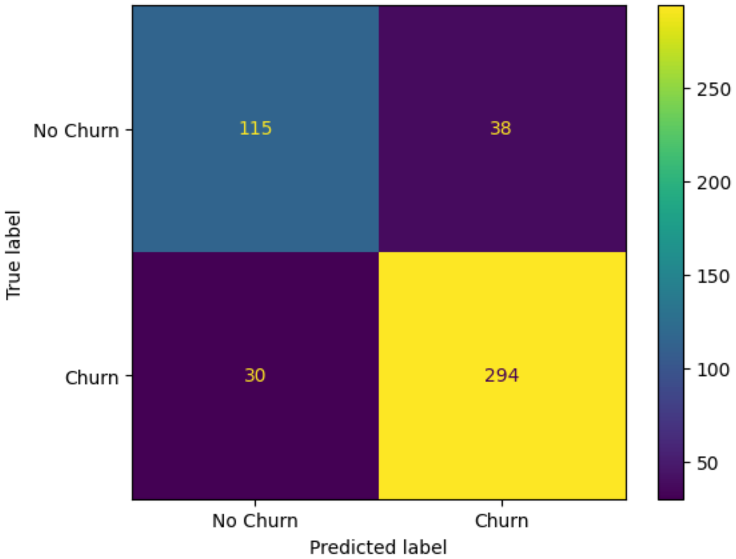
```
RF = RandomForestClassifier(n_estimators = 100,
    criterion = 'entropy',
    max_depth = 20,
    min_samples_split = 3,
    min_samples_leaf = 4,
    min_weight_fraction_leaf = 0.0,
    max_features = 'sqrt',
    max_leaf_nodes = None,
    min_impurity_decrease = 0.0,
    bootstrap = True,
    oob_score = False,
    n_jobs = None,
```

```
    random_state = None,
    verbose = 0,
    warm_start = False,
    class_weight = 'balanced',
    max_samples = None)

RF.fit(X_train, y_train)
```

```
▾                          RandomForestClassifier
RandomForestClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=20, min_samples_leaf=4, min_samples_split=3)
```

```
y_pred = RF.predict(X_test)
y_pred_train = RF.predict(X_train)

cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(cm, display_labels = ['No Churn', 'Churn']).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b44b0451270>
```




```
print(classification_report(y_train, y_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.87      0.90      0.88       612
           1       0.95      0.94      0.94      1292

    accuracy                           0.92      1904
   macro avg       0.91      0.92      0.91      1904
weighted avg       0.92      0.92      0.92      1904
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.75      0.77       153
           1       0.89      0.91      0.90       324

    accuracy                           0.86       477
   macro avg       0.84      0.83      0.83       477
weighted avg       0.86      0.86      0.86       477
```

```
train_accuracy = RF.score(X_train, y_train)
test_accuracy = RF.score(X_test, y_test)
precision_train = precision_score(y_train, y_pred_train)
precision_test = precision_score(y_test, y_pred)
recall_train = recall_score(y_train, y_pred_train)
recall_test = recall_score(y_test, y_pred)
training_f1_score = f1_score(y_train, y_pred_train)
test_f1_score = f1_score(y_test, y_pred)

training_data_metrics = pd.DataFrame(index = ['Accuracy', 'Precision', 'Recall', 'F1-Score'], data = [train_accuracy, precisic
training_data_metrics.rename(columns = {'index': 'Training Data Metrics'}, inplace = True)
test_data_metrics = pd.DataFrame(index = ['Accuracy', 'Precision', 'Recall', 'F1-Score'], data = [test_accuracy, precision_tes
test_data_metrics.rename(columns = {'index': 'Test Data Metrics'}, inplace = True)
```

```
training_data_metrics
```

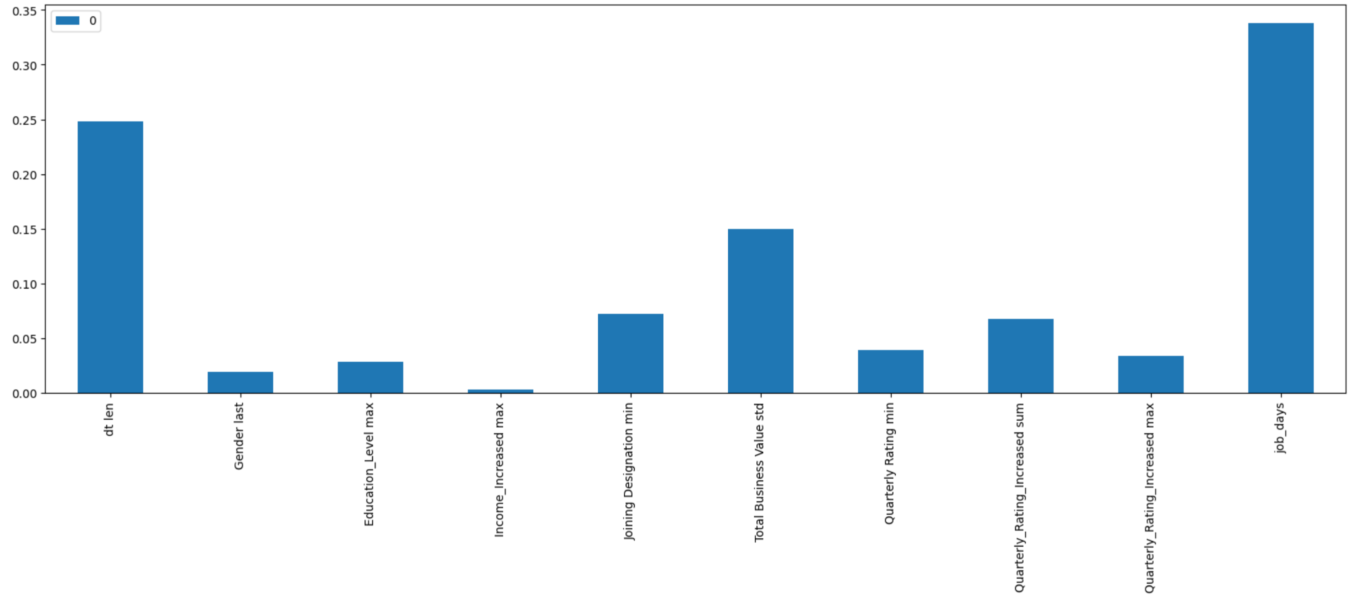
	Training Data Metrics	Values
0	Accuracy	0.923845
1	Precision	0.950511
2	Recall	0.936533
3	F1-Score	0.943470

```
test_data_metrics
```

	Test Data Metrics	Values
0	Accuracy	0.857442
1	Precision	0.885542
2	Recall	0.907407
3	F1-Score	0.896341

```
# feature importance
pd.DataFrame(data = RF.feature_importances_,
            index = X.columns).plot(kind = "bar", figsize = (20, 6))
```

```
<Axes: >
```



Oversample Churn data & use RF

> we have decent recall of ~92% using RF, lets try using smote

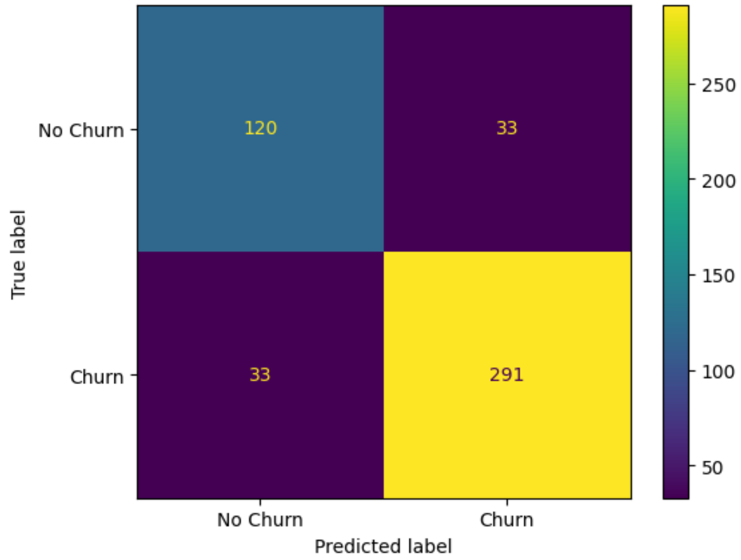
```
from imblearn.over_sampling import SMOTE

smoteBalance = SMOTE(k_neighbors = 7)
X_smote, y_smote = smoteBalance.fit_resample(X_train, y_train)
X_smote.shape, y_smote.shape
```

```
((2584, 10), (2584,))
```

```
RF.fit(X_smote, y_smote)
```

```
y_train_pred_bal = RF.predict(X_smote)
y_pred_bal = RF.predict(X_test)

cm = confusion_matrix(y_test, y_pred_bal)

ConfusionMatrixDisplay(cm, display_labels = ['No Churn', 'Churn']).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b44b034a800>
```




```
train_accuracy = RF.score(X_smote, y_smote)
test_accuracy = RF.score(X_test, y_test)
precision_train = precision_score(y_smote, y_train_pred_bal)
precision_test = precision_score(y_test, y_pred_bal)
recall_train = recall_score(y_smote, y_train_pred_bal)
recall_test = recall_score(y_test, y_pred_bal)
training_f1_score = f1_score(y_smote, y_train_pred_bal)
test_f1_score = f1_score(y_test, y_pred_bal)

training_data_metrics = pd.DataFrame(index = ['Accuracy', 'Precision', 'Recall', 'F1-Score'], data = [train_accuracy, precisic
training_data_metrics.rename(columns = {'index': 'Training Data Metrics'}, inplace = True)
test_data_metrics = pd.DataFrame(index = ['Accuracy', 'Precision', 'Recall', 'F1-Score'], data = [test_accuracy, precision_tes
test_data_metrics.rename(columns = {'index': 'Test Data Metrics'}, inplace = True)
```

```
training_data_metrics
```

	Training Data Metrics	Values
0	Accuracy	0.946981
1	Precision	0.937169
2	Recall	0.958204
3	F1-Score	0.947570

```
test_data_metrics
```

	Test Data Metrics	Values
0	Accuracy	0.865828
1	Precision	0.898773
2	Recall	0.904321
3	F1-Score	0.901538

Apply Grid search & use RF on smote data

```
from sklearn.model_selection import GridSearchCV

parameters = {"max_depth":[7, 10, 15, 20, 25, 30],
              "n_estimators":[50, 100, 200, 300, 400],
              "max_features":[4, 7, 10],
```

```
              "ccp_alpha":[0.0005, 0.00075, 0.001]}

RFC = RandomForestClassifier()
grid_search = GridSearchCV(
    estimator = RFC,
    param_grid = parameters,
    scoring = "accuracy",
    n_jobs = -1,
    refit=True,                   # need not to train again after grid search
    cv=3,
    pre_dispatch='2*n_jobs',
    return_train_score=False)
```

```
grid_search.fit(X_smote,y_smote.values.ravel())
```

```
▸           GridSearchCV
▸ estimator: RandomForestClassifier
      ▸ RandomForestClassifier
```

```
grid_search.best_estimator_
```

```
▾                         RandomForestClassifier
RandomForestClassifier(ccp_alpha=0.0005, max_depth=15, max_features=7,
                       n_estimators=200)
```

```
grid_search.best_score_
```

```
0.8773373646895236
```

```
grid_search.best_params_
```

```
{'ccp_alpha': 0.0005, 'max_depth': 15, 'max_features': 7, 'n_estimators': 200}
```

```
RF_best_params = RandomForestClassifier(n_estimators = 300,
    criterion = 'entropy',
    max_depth = 15,
    min_samples_split = 2,
    min_samples_leaf = 1,
    max_features = 10,
    class_weight = "balanced",
    ccp_alpha = 0.001,
    max_samples = None)
```

```
RF_best_params.fit(X_train, y_train)
```

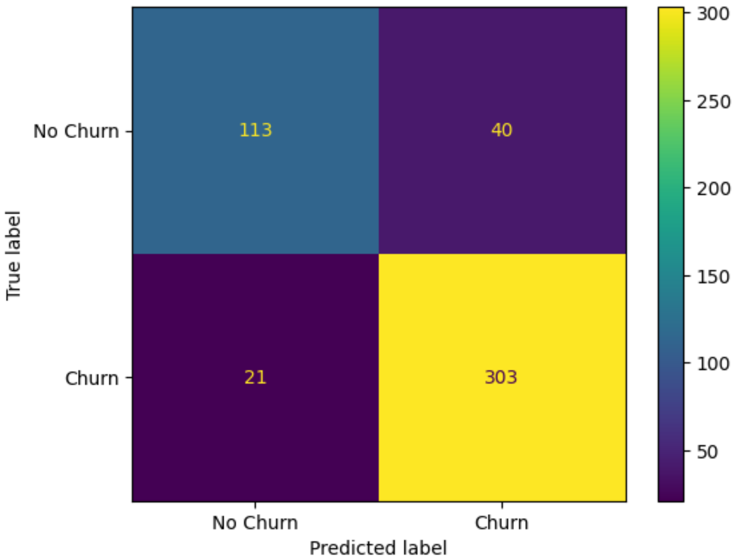
```
▾                         RandomForestClassifier
RandomForestClassifier(ccp_alpha=0.001, class_weight='balanced',
                       criterion='entropy', max_depth=15, max_features=10,
                       n_estimators=300)
```

```
def roc_curve_plot(y_test, pred_prob, model_name):
  logit_roc_auc = roc_auc_score(y_test, pred_prob)
  fpr, tpr, thresholds = roc_curve(y_test, pred_prob)
  plt.figure()
  plt.plot(fpr, tpr, label= model_name + ' (Area under ROC = %0.2f)' % logit_roc_auc)
  plt.plot([0, 1], [0, 1],'r--')
  plt.ylim([0.0, 1.05])
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title('Receiver Operating Characteristic (ROC) Curve')
  plt.legend(loc="lower right")
  plt.savefig('ROC_for_' + model_name)
  plt.show()

def precision_recall_curve_plot(y_test, pred_prob):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_prob)
    prec_rec_auc = np.round(auc(recalls, precisions), 2)

    threshold_boundary = thresholds.shape[0]

    #Precision Plot
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label = 'Precisions')

    #Recall Plot
```

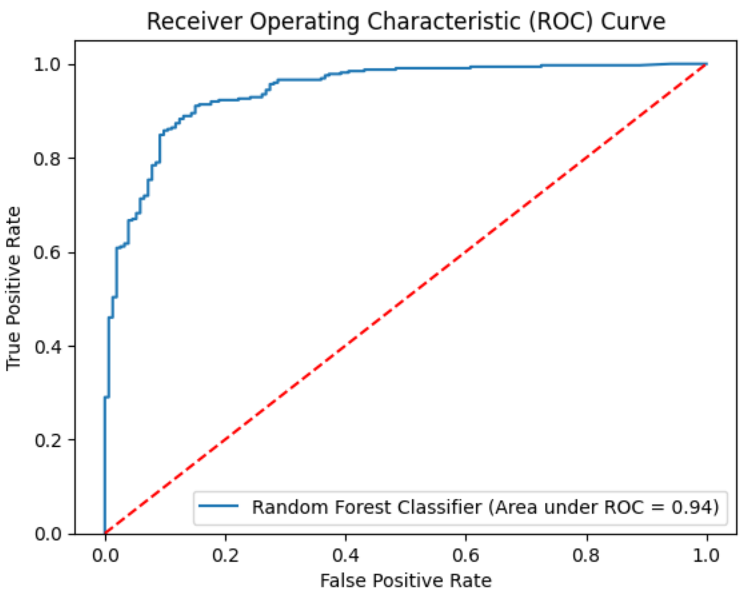
```
    plt.plot(thresholds, recalls[0:threshold_boundary], label='Recalls')

    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1), 2))

    plt.xlabel('Threshold Value')
    plt.ylabel('Precision and Recall Value')
    plt.title(f'Precision recall curve with AUC : {prec_rec_auc}')
    plt.legend()
    plt.grid()
    plt.show()
```

```
y_pred = RF_best_params.predict(X_test)
y_pred_train = RF_best_params.predict(X_train)

cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(cm, display_labels = ['No Churn', 'Churn']).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7a46e10837c0>
```




```
roc_curve_plot(y_test, RF_best_params.predict_proba(X_test)[:, 1], 'Random Forest Classifier')
```




```
print(classification_report(y_train, y_pred_train))
```

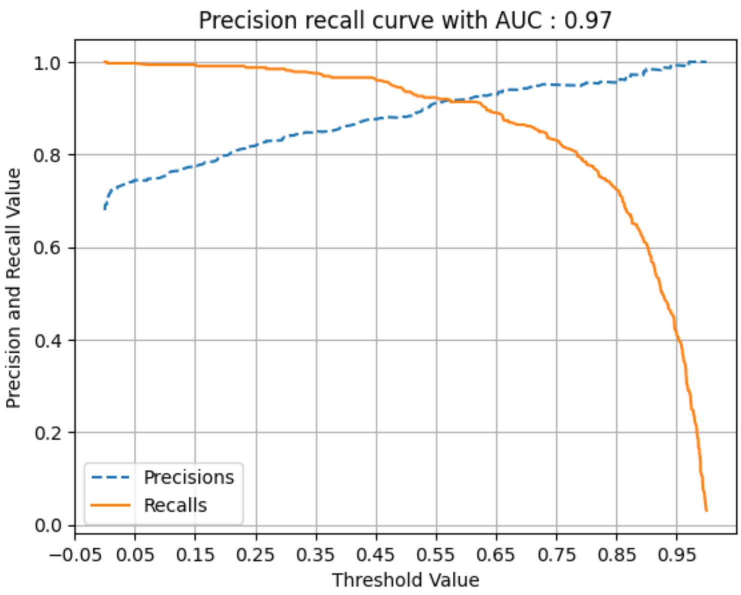
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       612
           1       1.00      1.00      1.00      1292

    accuracy                           1.00      1904
   macro avg       1.00      1.00      1.00      1904
```

```
weighted avg       1.00      1.00      1.00      1904
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.74      0.79       153
           1       0.88      0.94      0.91       324

    accuracy                           0.87       477
   macro avg       0.86      0.84      0.85       477
weighted avg       0.87      0.87      0.87       477
```

```
precision_recall_curve_plot(y_test, RF_best_params.predict_proba(X_test)[:, 1])
```



Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
gbt = GradientBoostingClassifier()
gbt.fit(X_train, y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

```
y_pred = gbt.predict(X_test)
y_pred_train = gbt.predict(X_train)
prob = gbt.predict_proba(X_test)
cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(cm, display_labels = ['No Churn', 'Churn']).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b44b01039d0>
```


```
train_accuracy = gbt.score(X_train, y_train)
test_accuracy = gbt.score(X_test, y_test)
precision_train = precision_score(y_train, y_pred_train)
precision_test = precision_score(y_test, y_pred)
recall_train = recall_score(y_train, y_pred_train)
recall_test = recall_score(y_test, y_pred)
training_f1_score = f1_score(y_train, y_pred_train)
test_f1_score = f1_score(y_test, y_pred)

training_data_metrics = pd.DataFrame(index = ['Accuracy', 'Precision', 'Recall', 'F1-Score'], data = [train_accuracy, precisic
training_data_metrics.rename(columns = {'index': 'Training Data Metrics'}, inplace = True)
test_data_metrics = pd.DataFrame(index = ['Accuracy', 'Precision', 'Recall', 'F1-Score'], data = [test_accuracy, precision_tes
test_data_metrics.rename(columns = {'index': 'Test Data Metrics'}, inplace = True)
```

```
training_data_metrics
```

	Training Data Metrics	Values
0	Accuracy	0.904937
1	Precision	0.904588
2	Recall	0.961300
3	F1-Score	0.932083

```
test_data_metrics
```

	Test Data Metrics	Values
0	Accuracy	0.886792
1	Precision	0.894737
2	Recall	0.944444
3	F1-Score	0.918919

gbdt grid search

```
parameters = {"max_depth":[7, 10, 15],
             "n_estimators":[100, 200, 300, 400],
             "max_features":[4, 7, 10],
             "ccp_alpha":[0.0005, 0.00075, 0.001]}

gbt = GradientBoostingClassifier()
grid_search = GridSearchCV(
    estimator = gbt,
    param_grid = parameters,
    scoring = "accuracy",
    n_jobs = -1,
    refit=True,                   # need not to train again after grid search
    cv=3,
    pre_dispatch='2*n_jobs',
    return_train_score=False)
```

```
grid_search.fit(X_train,y_train.values.ravel())
```

```
▸ GridSearchCV
▸ estimator: GradientBoostingClassifier
    ▸ GradientBoostingClassifier
```

```
grid_search.best_estimator_
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier(ccp_alpha=0.0005, max_depth=7, max_features=7,
                           n_estimators=200)
```

```
grid_search.best_score_
```

```
0.8666062246950993
```

```
grid_search.best_params_
```

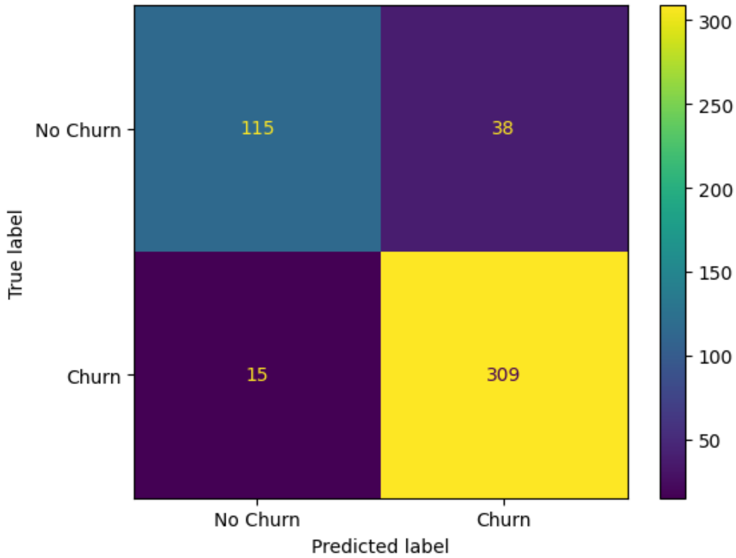
```
{'ccp_alpha': 0.0005, 'max_depth': 7, 'max_features': 7, 'n_estimators': 200}
```

```
gbt = GradientBoostingClassifier(ccp_alpha = 0.0005,
                                 max_depth= 7,
                                 max_features = 10,
                                 n_estimators = 200)
gbt.fit(X_train, y_train)
```

```
▾                    GradientBoostingClassifier
GradientBoostingClassifier(ccp_alpha=0.0005, max_depth=7, max_features=10,
                           n_estimators=200)
```

```
y_pred = gbt.predict(X_test)
y_pred_train = gbt.predict(X_train)
prob = gbt.predict_proba(X_test)

cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(cm, display_labels = ['No Churn', 'Churn']).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7a46dd4ebf10>
```




```
train_accuracy = gbt.score(X_train, y_train)
test_accuracy = gbt.score(X_test, y_test)
precision_train = precision_score(y_train, y_pred_train)
precision_test = precision_score(y_test, y_pred)
recall_train = recall_score(y_train, y_pred_train)
recall_test = recall_score(y_test, y_pred)
training_f1_score = f1_score(y_train, y_pred_train)
test_f1_score = f1_score(y_test, y_pred)

training_data_metrics = pd.DataFrame(index = ['Accuracy', 'Precision', 'Recall', 'F1-Score'], data = [train_accuracy, precisic
training_data_metrics.rename(columns = {'index': 'Training Data Metrics'}, inplace = True)
test_data_metrics = pd.DataFrame(index = ['Accuracy', 'Precision', 'Recall', 'F1-Score'], data = [test_accuracy, precision_tes
test_data_metrics.rename(columns = {'index': 'Test Data Metrics'}, inplace = True)
```

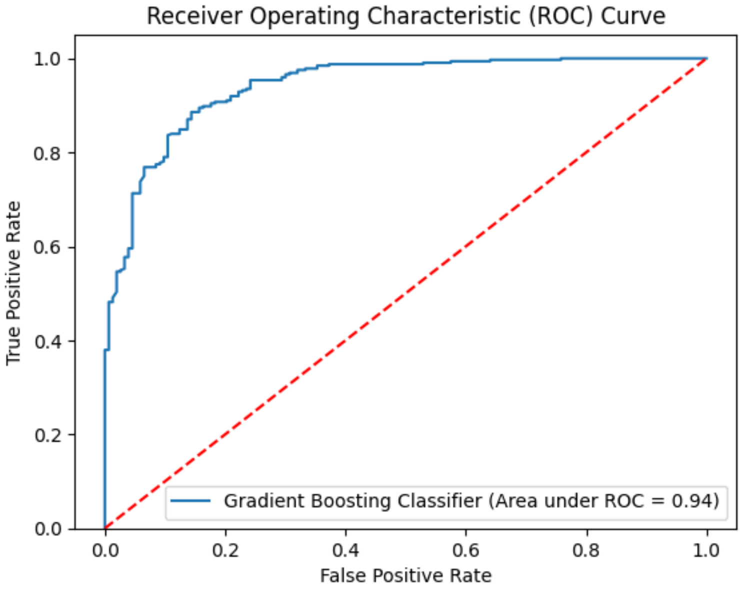
```
training_data_metrics
```

	Training Data Metrics	Values
0	Accuracy	0.904937
1	Precision	0.897638
2	Recall	0.970588
3	F1-Score	0.932689

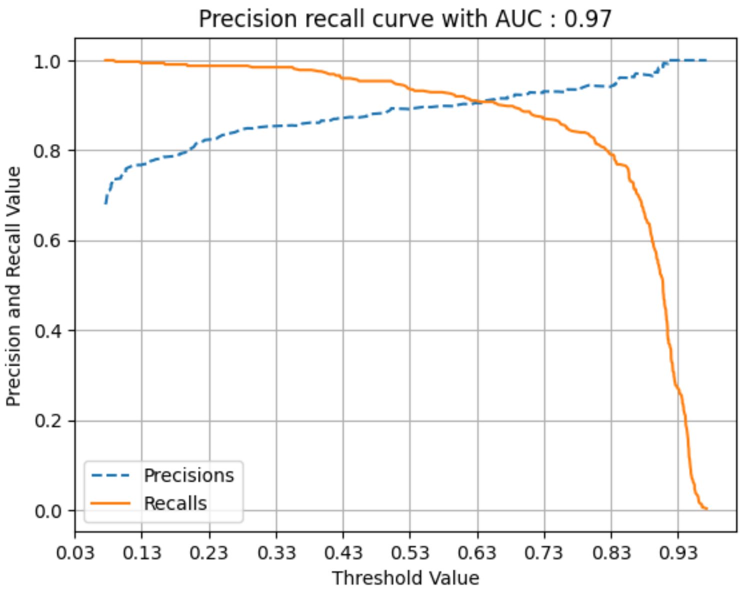
```
test_data_metrics
```

	Test Data Metrics	Values
0	Accuracy	0.888889
1	Precision	0.890490
2	Recall	0.953704
3	F1-Score	0.921013

```
# AUC ROC plot
prob = gbt.predict_proba(X_test)
roc_curve_plot(y_test, prob[:, 1], 'Gradient Boosting Classifier')
```




```
# precision recall plot
precision_recall_curve_plot(y_test, prob[:, 1])
```



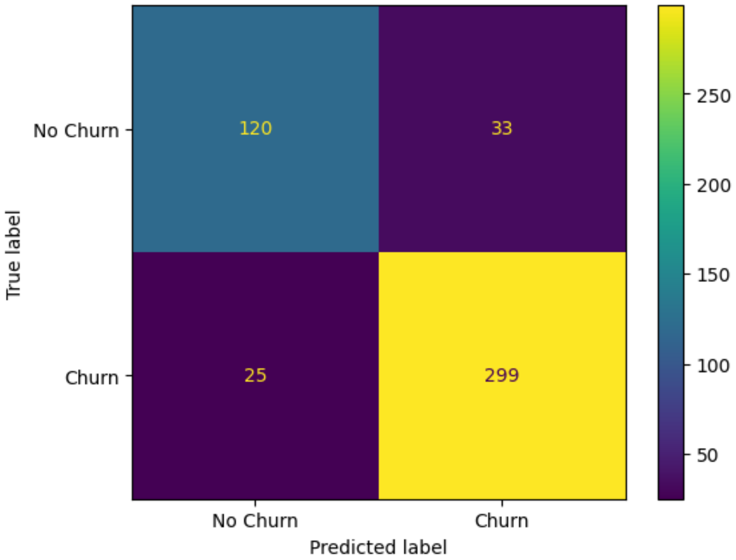
xgboost classifier

```
from xgboost import XGBClassifier

xgb = XGBClassifier(n_estimators = 100,
                    max_depth = 5)
xgb.fit(X_train, y_train)
```

```
▾                                    XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
```

```
y_pred = xgb.predict(X_test)
y_train_pred = xgb.predict(X_train)

cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(cm, display_labels = ['No Churn', 'Churn']).plot()
```

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b44c40c75b0>




```
print(classification_report(y_train, y_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.93      0.77      0.84       612
           1       0.90      0.97      0.93      1292

    accuracy                           0.90      1904
   macro avg       0.91      0.87      0.89      1904
weighted avg       0.91      0.90      0.90      1904
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.83      0.85       153
           1       0.92      0.94      0.93       324

    accuracy                           0.91       477
   macro avg       0.90      0.89      0.89       477
weighted avg       0.90      0.91      0.91       477
```

```
train_accuracy = xgb.score(X_train, y_train)
test_accuracy = xgb.score(X_test, y_test)
precision_train = precision_score(y_train, y_train_pred)
precision_test = precision_score(y_test, y_pred)
recall_train = recall_score(y_train, y_train_pred)
recall_test = recall_score(y_test, y_pred)
training_f1_score = f1_score(y_train, y_train_pred)
test_f1_score = f1_score(y_test, y_pred)

training_data_metrics = pd.DataFrame(index = ['Accuracy', 'Precision', 'Recall', 'F1-Score'], data = [train_accuracy, precisio
training_data_metrics.rename(columns = {'index': 'Training Data Metrics'}, inplace = True)
test_data_metrics = pd.DataFrame(index = ['Accuracy', 'Precision', 'Recall', 'F1-Score'], data = [test_accuracy, precision_tes
test_data_metrics.rename(columns = {'index': 'Test Data Metrics'}, inplace = True)
```

```
training_data_metrics
```

	Training Data Metrics	Values
0	Accuracy	0.996849
1	Precision	0.995378
2	Recall	1.000000
3	F1-Score	0.997683

```
test_data_metrics
```

	Test Data Metrics	Values
0	Accuracy	0.905660
1	Precision	0.921450
2	Recall	0.941358
3	F1-Score	0.931298

xgb randomised search

```
from sklearn.model_selection import RandomizedSearchCV


# Define hyperparameter grid for random search
parameters = {"max_depth":[2, 4, 5],
             "n_estimators":[100, 200, 300, 400]}



# Create and train t with random search
xgb = XGBClassifier()
random_search = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=parameters,
    scoring="recall",
    n_iter=10,  # Number of parameter settings to sample
    n_jobs=-1,
    cv=3,
    refit=True,
    random_state=42  # Set a random seed for reproducibility
)

random_search.fit(X_train, y_train)

# Make predictions on the test data using the best model from the random search
best_model = random_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f'Best Model Parameters: {random_search.best_params_}')
print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:')
print(conf_matrix)
print('Classification Report:')
print(classification_rep)
```

```
Best Model Parameters: {'n_estimators': 100, 'max_depth': 2}
Accuracy: 0.89
Confusion Matrix:
[[120  33]
 [ 19 305]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.78      0.82       153
           1       0.90      0.94      0.92       324

    accuracy                           0.89       477
   macro avg       0.88      0.86      0.87       477
weighted avg       0.89      0.89      0.89       477
```

```
y_train.value_counts()
```

```
1    1292
0     612
Name: Churn, dtype: int64
```

BalancedRandomForest (since there is class imbalance)

```
from imblearn.ensemble import BalancedRandomForestClassifier

model = BalancedRandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
print(f'Best Model Parameters: {random_search.best_params_}')
print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:')
print(conf_matrix)
print('Classification Report:')
print(classification_rep)
```

```
Best Model Parameters: {'n_estimators': 50, 'max_depth': 4, 'learning_rate': 0.1}
Accuracy: 0.86
Confusion Matrix:
[[127  26]
 [ 42 282]]
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.83      0.79       153
           1       0.92      0.87      0.89       324

    accuracy                           0.86       477
   macro avg       0.83      0.85      0.84       477
weighted avg       0.86      0.86      0.86       477
```

LightGBM

```
import lightgbm as lgb
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Define hyperparameter grid for random search
parameters = {
    "n_estimators": [50, 100, 150, 200, 300, 400, 500, 600],
    "max_depth": [2, 3, 4, 5, 7],
    "learning_rate": [0.1, 0.2, 0.4, 0.5]
}

num_positive_samples = sum(y_train == 1)  # Count of positive samples
num_negative_samples = sum(y_train == 0)  # Count of negative samples

imbalance_ratio = num_positive_samples / num_negative_samples # majority/minority

# Create and train with random search
lgb_model = lgb.LGBMClassifier(scale_pos_weight=imbalance_ratio)  # Specify scale_pos_weight here
random_search = RandomizedSearchCV(
    estimator=lgb_model,
    param_distributions=parameters,
    scoring="recall",
    n_iter=20,  # Number of parameter settings to sample
    n_jobs=-1,
    cv=3,
    refit=True,
    random_state=42  # Set a random seed for reproducibility
)

random_search.fit(X_train, y_train)

res = random_search.cv_results_

for i in range(len(res["params"])):
    print(f"Parameters:{res['params'][i]} Mean_score: {res['mean_test_score'][i]} Rank: {res['rank_test_score'][i]}")

# Make predictions on the test data using the best model from the random search
```

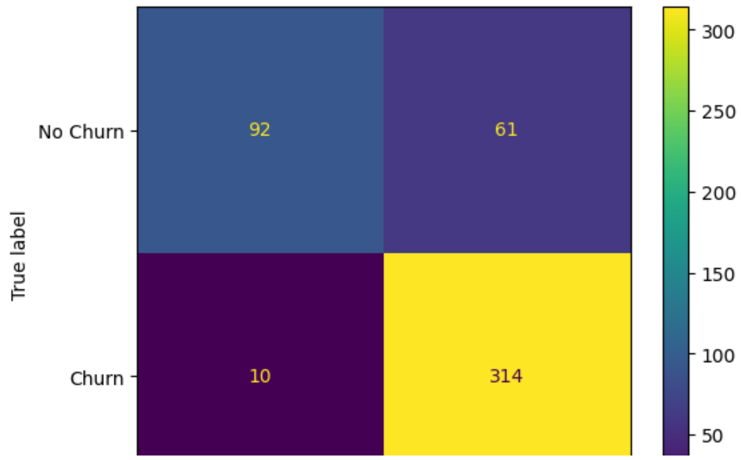
```
best_model = random_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f'Best Model Parameters: {random_search.best_params_}')
print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:')
print(conf_matrix)
print('Classification Report:')
print(classification_rep)
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Parameters:{'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.4} Mean_score: 0.9295706757315779 Rank: 11
Parameters:{'n_estimators': 300, 'max_depth': 5, 'learning_rate': 0.4} Mean_score: 0.916413964279933 Rank: 17
Parameters:{'n_estimators': 400, 'max_depth': 4, 'learning_rate': 0.5} Mean_score: 0.9125397938811849 Rank: 18
Parameters:{'n_estimators': 600, 'max_depth': 3, 'learning_rate': 0.2} Mean_score: 0.9287936833396283 Rank: 12
Parameters:{'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.4} Mean_score: 0.9171855608913829 Rank: 16
Parameters:{'n_estimators': 400, 'max_depth': 5, 'learning_rate': 0.1} Mean_score: 0.934214644148276 Rank: 9
Parameters:{'n_estimators': 400, 'max_depth': 4, 'learning_rate': 0.4} Mean_score: 0.9109948020647853 Rank: 19
Parameters:{'n_estimators': 200, 'max_depth': 3, 'learning_rate': 0.2} Mean_score: 0.9481411536178709 Rank: 5
Parameters:{'n_estimators': 300, 'max_depth': 4, 'learning_rate': 0.4} Mean_score: 0.919507545099732 Rank: 15
Parameters:{'n_estimators': 500, 'max_depth': 4, 'learning_rate': 0.5} Mean_score: 0.9078976240579867 Rank: 20
Parameters:{'n_estimators': 200, 'max_depth': 4, 'learning_rate': 0.1} Mean_score: 0.9543301138509687 Rank: 3
Parameters:{'n_estimators': 300, 'max_depth': 2, 'learning_rate': 0.4} Mean_score: 0.9442687818126224 Rank: 6
Parameters:{'n_estimators': 600, 'max_depth': 3, 'learning_rate': 0.1} Mean_score: 0.9434989837946727 Rank: 7
Parameters:{'n_estimators': 150, 'max_depth': 5, 'learning_rate': 0.2} Mean_score: 0.9357614345581755 Rank: 8
Parameters:{'n_estimators': 50, 'max_depth': 5, 'learning_rate': 0.1} Mean_score: 0.968262019101063 Rank: 2
Parameters:{'n_estimators': 500, 'max_depth': 5, 'learning_rate': 0.1} Mean_score: 0.9326660551448768 Rank: 10
Parameters:{'n_estimators': 50, 'max_depth': 3, 'learning_rate': 0.5} Mean_score: 0.9535603158330185 Rank: 4
Parameters:{'n_estimators': 200, 'max_depth': 5, 'learning_rate': 0.5} Mean_score: 0.921831327901581 Rank: 14
Parameters:{'n_estimators': 150, 'max_depth': 4, 'learning_rate': 0.4} Mean_score: 0.9249195129408802 Rank: 13
Parameters:{'n_estimators': 50, 'max_depth': 4, 'learning_rate': 0.1} Mean_score: 0.9775553517149588 Rank: 1
Best Model Parameters: {'n_estimators': 50, 'max_depth': 4, 'learning_rate': 0.1}
Accuracy: 0.85
Confusion Matrix:
[[ 92  61]
 [ 10 314]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.60      0.72       153
           1       0.84      0.97      0.90       324

    accuracy                           0.85       477
   macro avg       0.87      0.79      0.81       477
weighted avg       0.86      0.85      0.84       477
```

recall = 0.97 for (Churn=1) which means we would be able to reduce False negatives, with 97% accuracy we will be able to predict if driver is about to Churn. We have good enough 0.84 precision as well.

```
y_pred = best_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(cm, display_labels = ['No Churn', 'Churn']).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b44a013f100>
```

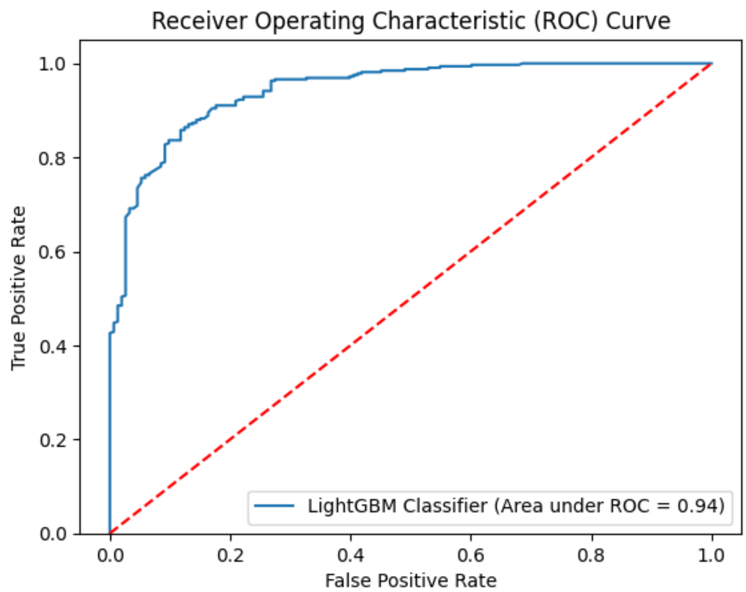



```
y_test.shape
```

```
(477,)
```

Only 10 customers out of 477 test customers, we were not able to predict that they will Churn.

```
prob = best_model.predict_proba(X_test)
roc_curve_plot(y_test, prob[:, 1], 'LightGBM Classifier')
```




```
precision_recall_curve_plot(y_test, prob[:, 1])
```