In [132...

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.model_selection import StratifiedKFold
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report,
    roc_auc_score, roc_curve, auc,
    plot_confusion_matrix, plot_roc_curve,
    precision_recall_curve
)
```

Problem Statement

- **Primary Goal**
  - **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)
  - Recognizing **significant features** that will drive more attrition of drivers.
  - How well those features describe the attrition of drivers
  - How to **reduce new driver acquisition cost**
- **Long term benefits** : **customer growth** , **More market penetration** where (i.e. states, place, cities) there are less volume of requests, **Customer acquisition** , **Balance short and long trips** and **driver team and customer retention**

Exploratory Analysis

In [133...

```
df = pd.read_csv("driver.csv")
```

Data types - structure & characteristics of the dataset

In [134...

```
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
```

In [135…

```
df.shape
```

Out[135]:

```
(19104, 14)
```

In [136…

```
df.head()
```

Out[136]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate
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

Dropping Unnamed column

In [137…

```
df.drop(['Unnamed: 0'], axis=1,inplace=True)
```

In [138…

```
df.head()
```

Out[138]:

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

Non Graphical Analysis

In [139…

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

Out[139]:

```
0.0    11074
1.0     7978
Name: Gender, dtype: int64
```

In [140…

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

Out[140]:

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

In [141…

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

Out[141]:
```
C20    1008
C29     900
C26     869
C22     809
C27     786
C15     761
C10     744
C12     727
C8      712
C16     709
C28     683
C1      677
C6      660
C5      656
C14     648
C3      637
C24     614
C7      609
C21     603
C25     584
C19     579
C4      578
C13     569
C18     544
C23     538
C9      520
C2      472
C11     468
C17     440
Name: City, dtype: int64
```

In [142…
```
df["Education_Level"].value_counts()
```

Out[142]:
```
1    6864
2    6327
0    5913
Name: Education_Level, dtype: int64
```

Convert 'Dateofjoining' feature to date type

In [143…
```
df['Dateofjoining'] = pd.to_datetime(df['Dateofjoining'])
```

Convert MMM-YY to date type

In [144…
```
df['MMM-YY'] = pd.to_datetime(df['MMM-YY'])
```

Data Preprocessing

Feature Engineering - Part I

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

In [145…
```
df['LastWorkingDate'] = df['LastWorkingDate'].replace(np.nan, 0)
```

In [146…
```
def generate_feature_transform(x):
    if x==0 :
        return 1 # Charged Drivers as +ve target class because that's more concerning fo
```

```
    else:
        return 0
```

In [147…

```
df["is_charned"] = df["LastWorkingDate"].apply(generate_feature_transform)
```

In [148…

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

Out[148]:

```
1    17488
0     1616
Name: is_charned, dtype: int64
```

In [149…

```
df['LastWorkingDate'] = df['LastWorkingDate'].replace(0, np.nan)
```

Convert 'LastWorkingDate' feature to date type

In [150…

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

Extract Month and year from MMM-YY

In [151…

```
df["year"] = df["MMM-YY"].dt.year
df["month"] = df["MMM-YY"].dt.month_name()
# Removing 'MMM-YY' post year amd month extraction as ''MMM-YY'' will be redundant featu
df.drop(['MMM-YY'], axis=1,inplace=True)
```

Age to generations

In [152…

```
def age_to_generation(age):
    if age >= 10 and age <= 25:
        return 'Gen-Z'
    elif age >= 26 and age <= 32:
        return 'Younger-Millennials'
    elif age >= 33 and age <= 41:
        return 'Older-Millennials'
    else:
        return 'Gen-X'
```

In [153…

```
df["generation"] = df["Age"].apply(age_to_generation)
```

Check Missing values (Only numerical features)

In [154…

```
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df.columns,
                                 'percent_missing': percent_missing})
missing_value_df.sort_values('percent_missing', ascending=False)
```

Out[154]:

	column_name	percent_missing
LastWorkingDate	LastWorkingDate	91.541039
Age	Age	0.319305
Gender	Gender	0.272194
Driver_ID	Driver_ID	0.000000
City	City	0.000000
Education_Level	Education_Level	0.000000
Income	Income	0.000000
Dateofjoining	Dateofjoining	0.000000
Joining Designation	Joining Designation	0.000000
Grade	Grade	0.000000
Total Business Value	Total Business Value	0.000000
Quarterly Rating	Quarterly Rating	0.000000
is_charned	is_charned	0.000000
year	year	0.000000
month	month	0.000000
generation	generation	0.000000

In [155…

```
df_numeric = df.select_dtypes(include='number')
df_non_numeric = df.select_dtypes(exclude='number')
```

Non Graphical Analysis - Part 2

In [156…

```
pd.crosstab(df['is_charned'], df['month'], margins=True,normalize=True)*100
```

Out[156]:

month	April	August	December	February	January	July	June	March	May	November
is_charned										
0	0.492044	0.612437	0.753769	0.769472	0.790410	0.759003	0.612437	0.701424	0.858459	0.675251
1	7.480109	7.584799	7.694724	8.155360	8.652638	7.422529	7.218384	7.616206	7.150335	7.626675
All	7.972152	8.197236	8.448492	8.924832	9.443049	8.181533	7.830821	8.317630	8.008794	8.301926

- **Insights**
  - Driver churning is consistent across months
  - Churning Month specific information can be fetched from last date month
  - Also months of tenure can be fetched from derived feature - diff of days joining and leaving date

In [157…

```
pd.crosstab(df['is_charned'], df['year'], margins=True,normalize=True)*100
```

Out[157]:

year	2019	2020	All
is_charned			
0	4.318467	4.140494	8.458961
1	46.498116	45.042923	91.541039
All	50.816583	49.183417	100.000000

- **Insights**
  - Nearly same percentage of churning in year 2019 and 2020
  - Hence **data processing will be considered wrtt. Driver Id , excluding months and years**

KNN Imputation

In [158…

```
#### Numerical missing value treatment - KNN Imputer
from sklearn.impute import KNNImputer
imputer = KNNImputer(missing_values = np.nan, n_neighbors=7)
df_numeric = pd.DataFrame(imputer.fit_transform(df_numeric),columns = df_numeric.columns
```

In [159…

```
percent_missing = df_numeric.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df_numeric.columns,
                                 'percent_missing': percent_missing})
missing_value_df.sort_values('percent_missing', ascending=False)
```

Out[159]:

	column_name	percent_missing
Driver_ID	Driver_ID	0.0
Age	Age	0.0
Gender	Gender	0.0
Education_Level	Education_Level	0.0
Income	Income	0.0
Joining Designation	Joining Designation	0.0
Grade	Grade	0.0
Total Business Value	Total Business Value	0.0
Quarterly Rating	Quarterly Rating	0.0
is_charned	is_charned	0.0
year	year	0.0

In [160…

```
df_numeric.shape
```

Out[160]:

```
(19104, 11)
```

In [161…

```
df = pd.merge(df_numeric, df_non_numeric, left_index=True, right_index=True)
df.shape
```

Out[161]:

```
(19104, 16)
```

In [162…

```
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df.columns,
                                 'percent_missing': percent_missing})
missing_value_df.sort_values('percent_missing', ascending=False)
```

Out[162]:

	column_name	percent_missing
LastWorkingDate	LastWorkingDate	91.541039
Driver_ID	Driver_ID	0.000000
Age	Age	0.000000
Gender	Gender	0.000000
Education_Level	Education_Level	0.000000
Income	Income	0.000000
Joining Designation	Joining Designation	0.000000
Grade	Grade	0.000000
Total Business Value	Total Business Value	0.000000
Quarterly Rating	Quarterly Rating	0.000000
is_charned	is_charned	0.000000
year	year	0.000000
City	City	0.000000
Dateofjoining	Dateofjoining	0.000000
month	month	0.000000
generation	generation	0.000000

Aggregate data in order to remove multiple occurrences of same driver data

In [163…

```
df_agg = df.groupby(["Driver_ID"])[["Dateofjoining","LastWorkingDate","Total Business Va
df_agg.reset_index(inplace=True)
df_agg.columns = ['_'.join(col) for col in df_agg.columns]
df_agg.rename(columns = {'Driver_ID_':'Driver_ID','Total Business Value_sum':'Total_Bus
df_agg.head(500)
```

Out[163]:

	Driver_ID	Dateofjoining_first	LastWorkingDate_last	Total_Business_Value_sum	Income_first	Income_last	Q Rat
0	1.0	2018-12-24	2019-03-11	1715580.0	57387.0	57387.0	
1	2.0	2020-11-06	NaT	0.0	67016.0	67016.0	
2	4.0	2019-12-07	2020-04-27	350000.0	65603.0	65603.0	
3	5.0	2019-01-09	2019-03-07	120360.0	46368.0	46368.0	
4	6.0	2020-07-31	NaT	1265000.0	78728.0	78728.0	
...	...	...	...	...	...	...	
495	575.0	2020-12-21	NaT	0.0	39391.0	39391.0	
496	576.0	2020-05-08	2020-10-26	242560.0	68356.0	68356.0	
497	577.0	2020-05-03	2020-10-26	158570.0	108091.0	108091.0	
498	578.0	2018-12-21	2019-01-19	0.0	46169.0	46169.0	
499	579.0	2017-06-30	2019-06-29	1583300.0	70570.0	70570.0	

500 rows × 8 columns

In [164…

```
df_agg["Income_increased"] = df_agg["Income_last"] - df_agg["Income_first"]
df_agg["Quarterly_Rating_increased"] = df_agg["Quarterly Rating_last"] - df_agg["Quarte
df_agg.head()
```

Out[164]:

	Driver_ID	Dateofjoining_first	LastWorkingDate_last	Total_Business_Value_sum	Income_first	Income_last	Qua Rating
0	1.0	2018-12-24	2019-03-11	1715580.0	57387.0	57387.0	
1	2.0	2020-11-06	NaT	0.0	67016.0	67016.0	
2	4.0	2019-12-07	2020-04-27	350000.0	65603.0	65603.0	
3	5.0	2019-01-09	2019-03-07	120360.0	46368.0	46368.0	
4	6.0	2020-07-31	NaT	1265000.0	78728.0	78728.0	

Feature Engineering - Part II

- Create feature which tells whether **quarterly rating has increased** for that driver
- Create feature which tells whether the **monthly income has increased** for that driver

In [165…

```
def boolean_transform(x):
    if x > 0:
        return 1
    else:
        return 0
```

In [166…

```
df_agg["Income_increased"] = df_agg["Income_increased"].apply(boolean_transform)
df_agg["Quarterly_Rating_increased"] = df_agg["Quarterly_Rating_increased"].apply(boolea
```

In [167…

```
df_agg.head()
```

Out[167]:

	Driver_ID	Dateofjoining_first	LastWorkingDate_last	Total_Business_Value_sum	Income_first	Income_last	Qua Ratin
0	1.0	2018-12-24	2019-03-11	1715580.0	57387.0	57387.0	
1	2.0	2020-11-06	NaT	0.0	67016.0	67016.0	
2	4.0	2019-12-07	2020-04-27	350000.0	65603.0	65603.0	
3	5.0	2019-01-09	2019-03-07	120360.0	46368.0	46368.0	
4	6.0	2020-07-31	NaT	1265000.0	78728.0	78728.0	

In [168…

```
# Selecting specific features from non aggregated data , before merging
df_org = df[["Driver_ID","Age","generation","Gender","City","Education_Level","Income",'
df_org.shape
```

Out[168]: (19104, 11)

In [169…

```
# Removing duplicate datas , keep first occurance of the data
df_org = df_org.drop_duplicates(keep='first')
df_org.shape
```

Out[169]: (7024, 11)

In [170…

```
# viewing selected rows post removing duplicate rows
df_org.head()
```

Out[170]:

	Driver_ID	Age	generation	Gender	City	Education_Level	Income	Joining Designation	Grade	Quarterly Rating	is_charned
0	1.0	28.0	Younger-Millennials	0.0	C23	2.0	57387.0	1.0	1.0	2.0	1.0
2	1.0	28.0	Younger-Millennials	0.0	C23	2.0	57387.0	1.0	1.0	2.0	0.0
3	2.0	31.0	Younger-Millennials	0.0	C7	2.0	67016.0	2.0	2.0	1.0	1.0
5	4.0	43.0	Gen-X	0.0	C13	2.0	65603.0	2.0	2.0	1.0	1.0
9	4.0	43.0	Gen-X	0.0	C13	2.0	65603.0	2.0	2.0	1.0	0.0

In [171…

```
# Removing duplicate records post aggregation
df_agg = df_agg.drop_duplicates(keep='first')
df_agg.shape
```

Out[171]: (2381, 10)

In [172…

```
# Merging aggregated and raw features
df_merged = pd.merge(df_agg, df_org, how="inner", on=["Driver_ID"])
df_merged.shape
```

Out[172]: (7024, 20)

In [173…

```
df_merged.head()
```

Out[173]:

	Driver_ID	Dateofjoining_first	LastWorkingDate_last	Total_Business_Value_sum	Income_first	Income_last	Qua Rating
0	1.0	2018-12-24	2019-03-11	1715580.0	57387.0	57387.0	
1	1.0	2018-12-24	2019-03-11	1715580.0	57387.0	57387.0	
2	2.0	2020-11-06	NaT	0.0	67016.0	67016.0	
3	4.0	2019-12-07	2020-04-27	350000.0	65603.0	65603.0	
4	4.0	2019-12-07	2020-04-27	350000.0	65603.0	65603.0	

In [174…

```
df_merged["is_charned"].value_counts(normalize=True)*100
```

Out[174]:

```
1.0    76.993166
0.0    23.006834
Name: is_charned, dtype: float64
```

Fill empty last dates by today's date

In [175…

```
df_merged["LastWorkingDate_last"].fillna(pd.to_datetime('today'),inplace=True)
```

In [176…

```
df_merged.head()
```

Out[176]:

	Driver_ID	Dateofjoining_first	LastWorkingDate_last	Total_Business_Value_sum	Income_first	Income_last	Qua Rating
0	1.0	2018-12-24	2019-03-11 00:00:00.000000	1715580.0	57387.0	57387.0	
1	1.0	2018-12-24	2019-03-11 00:00:00.000000	1715580.0	57387.0	57387.0	
2	2.0	2020-11-06	2022-12-03 23:00:57.568019	0.0	67016.0	67016.0	
3	4.0	2019-12-07	2020-04-27 00:00:00.000000	350000.0	65603.0	65603.0	
4	4.0	2019-12-07	2020-04-27 00:00:00.000000	350000.0	65603.0	65603.0	

In [177…

```
df_merged.drop(['Income_first','Income_last','Quarterly Rating_first','Quarterly Rating_
```

In [178…

```
df_merged.head()
```

Out[178]:

	Driver_ID	Dateofjoining_first	LastWorkingDate_last	Total_Business_Value_sum	Income_increased	Quarterly_Rati
0	1.0	2018-12-24	2019-03-11 00:00:00.000000	1715580.0	0	
1	1.0	2018-12-24	2019-03-11 00:00:00.000000	1715580.0	0	
2	2.0	2020-11-06	2022-12-03 23:00:57.568019	0.0	0	
3	4.0	2019-12-07	2020-04-27 00:00:00.000000	350000.0	0	
4	4.0	2019-12-07	2020-04-27 00:00:00.000000	350000.0	0	

In [179…

```
df_merged.shape
```

Out[179]:

```
(7024, 16)
```

Driver Tenure information (Months between Last Date and Joining date)

In [180…

```
df_merged["tenure_months"] = (df_merged["LastWorkingDate_last"] - df_merged["Dateofjoin
```

Fill Empty Dates before month , day , year extraction

In [181…

```
df_merged["year_of_last_date"] = df_merged["LastWorkingDate_last"].dt.year
df_merged["month_of_last_date"] = df_merged["LastWorkingDate_last"].dt.month_name()
df_merged["day_of_last_date"] = df_merged["LastWorkingDate_last"].dt.day_name()
```

- **Extract Day ,Month and Year from joining and last date**

In [182…

```
df_merged["year_of_joining"] = df_merged["Dateofjoining_first"].dt.year
df_merged["month_of_joining"] = df_merged["Dateofjoining_first"].dt.month_name()
df_merged["day_of_joining"] = df_merged["Dateofjoining_first"].dt.day_name()
```

In [183…

```
# Removing date features as we've extracted day , month and year features , which will l
df_merged.drop(["Dateofjoining_first","LastWorkingDate_last"], axis=1, inplace=True)
```

Checking Target Class Imbalance

In [184…

```
df_merged["is_charned"].value_counts()
```

Out[184]:

```
1.0    5408
0.0    1616
Name: is_charned, dtype: int64
```

Non Graphical Analysis - Part 3

In [185…

```
pd.crosstab(df_merged['is_charned'], df_merged["Quarterly Rating"], margins=True,normal:
```

Out[185]:

Quarterly Rating	1.0	2.0	3.0	4.0	All
is_charned					
0.0	20.387244	2.078588	0.398633	0.142369	23.006834
1.0	33.812642	21.099089	14.336560	7.744875	76.993166
All	54.199886	23.177677	14.735194	7.887244	100.000000

- **Insights**
  - **33.8% Drivers** who are leaving has **Quarterly Rating 1**
  - Drivers with Quarterly Rating 2(21%) and 3(~14%) are also contributing significantly for churning

In [186…

```
pd.crosstab(df_merged['is_charned'], df_merged["Joining Designation"], margins=True,norr
```

Out[186]:

Joining Designation	1.0	2.0	3.0	4.0	5.0	All
is_charned						
0.0	10.706150	7.972665	3.900911	0.313212	0.113895	23.006834
1.0	38.154897	24.316629	12.784738	1.380979	0.355923	76.993166
All	48.861048	32.289294	16.685649	1.694191	0.469818	100.000000

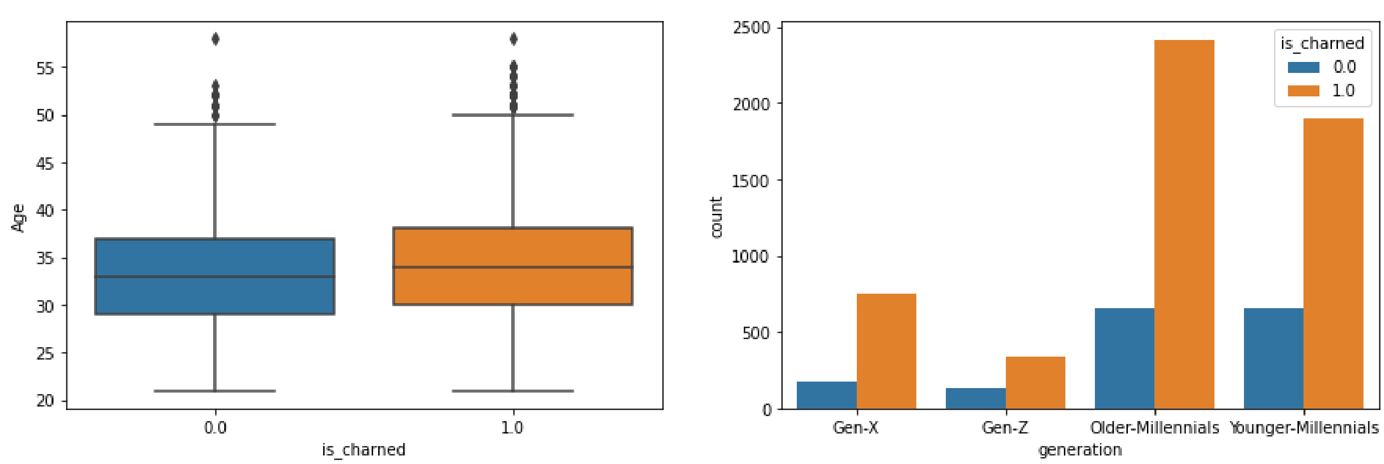
- **Insights**
  - **38.15% churn** is contributed by Drivers with **Joining Designation 1**
  - **24% churn** is contributed by Drivers with **Joining Designation 2**

Visual Analysis - Part 1

Age and generation based churn

In [187…

```
plt.figure(figsize=(15, 10))
plt.subplot(2, 2, 1)
sns.boxplot(y="Age", x="is_charned", data=df_merged)
plt.subplot(2, 2, 2)
generation = sorted(df_merged.generation.unique().tolist())
g = sns.countplot(x="generation",data=df_merged,hue="is_charned",order=generation)
g.set_xticklabels(g.get_xticklabels());
```



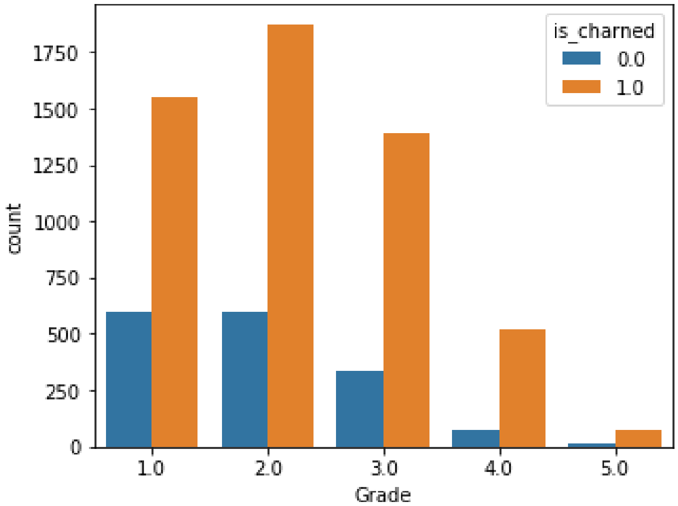
- **Insights**

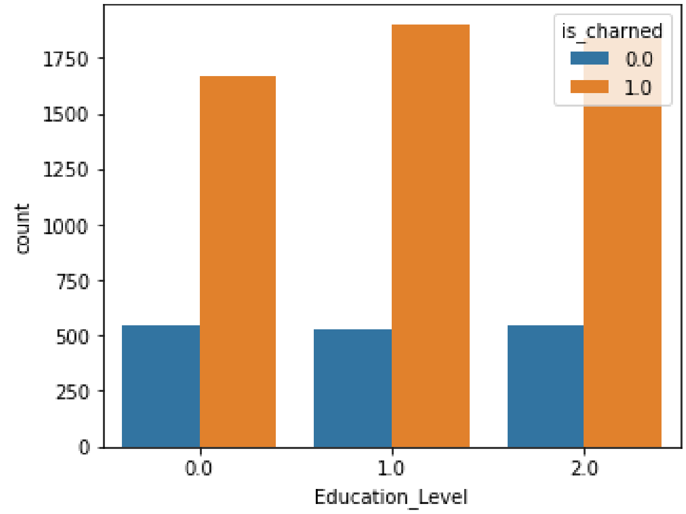
- Mean age of drivers nearly equal, across leavers and non leavers
- **Younger Millenials (26-32) and Older Millenials (33-41)** drivers are leaving most

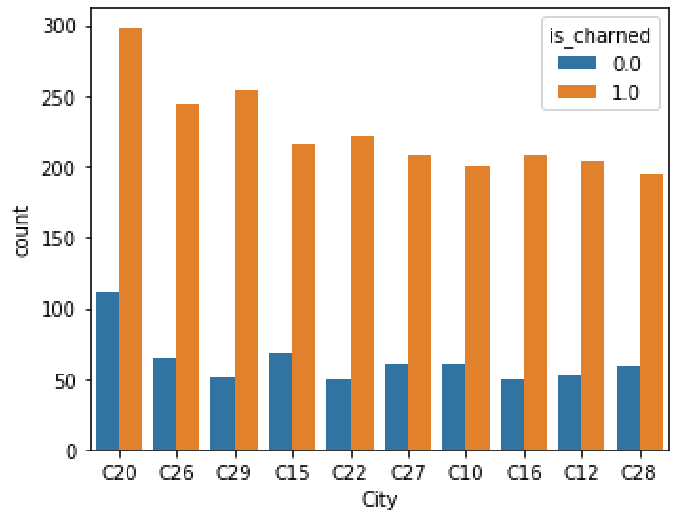
Grade, Education_Level,City, Joining Designation influences to churn

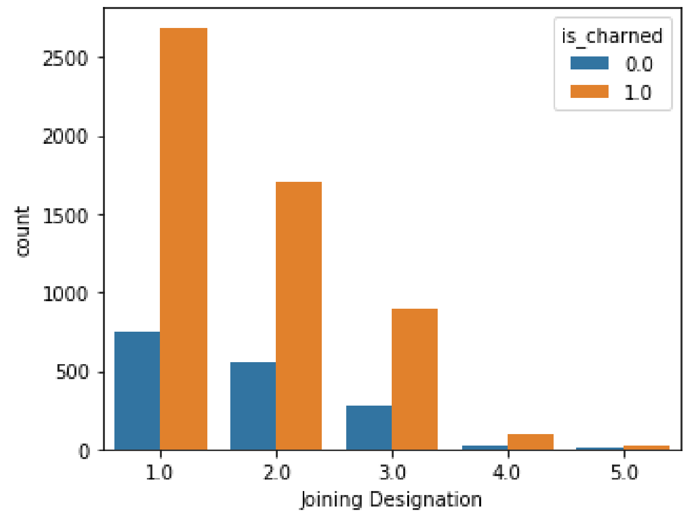
In [197…

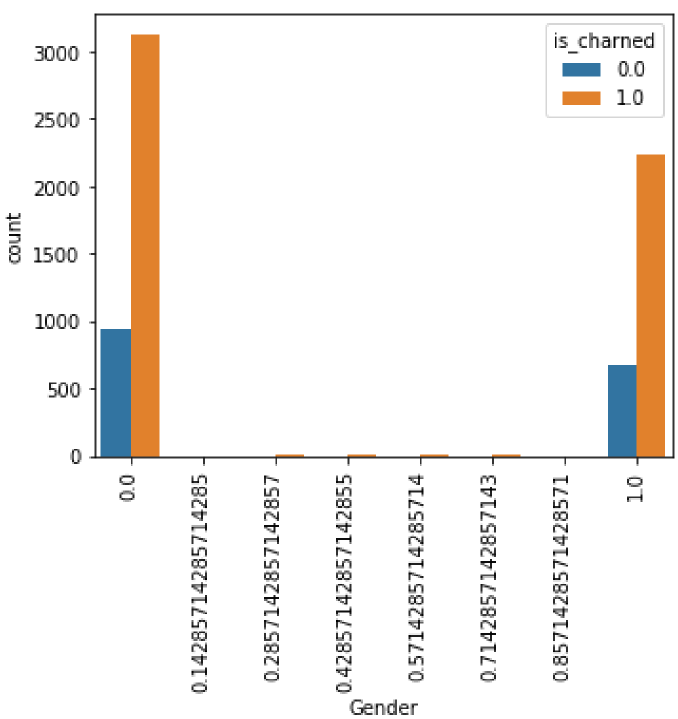
```
plt.figure(figsize=(12, 30))
plt.subplot(6, 2, 1)
sns.countplot(x='Grade', data=df_merged, hue='is_charned')
plt.subplot(6, 2, 2)
sns.countplot(x='Education_Level', data=df_merged, hue='is_charned')
plt.subplot(6, 2, 3)
sns.countplot(x='City', data=df_merged, hue='is_charned',order=df_merged["City"].value_c
plt.subplot(6, 2, 4)
sns.countplot(x='Joining Designation', data=df_merged, hue='is_charned')
plt.subplot(6, 2, 5)
g= sns.countplot(x='Gender', data=df_merged, hue='is_charned')
g.set_xticklabels(g.get_xticklabels(), rotation=90);

plt.show()
```







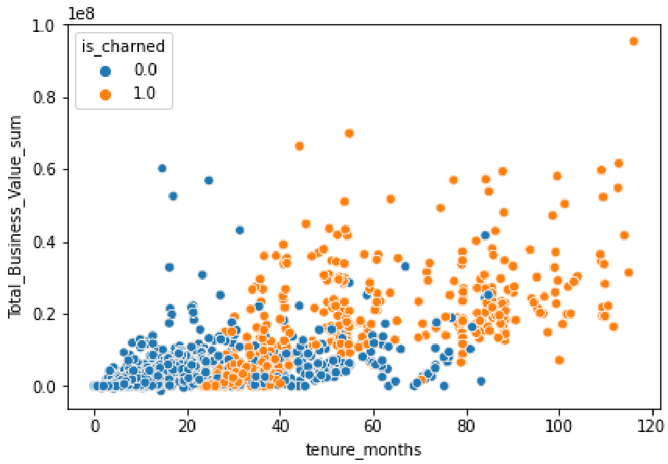


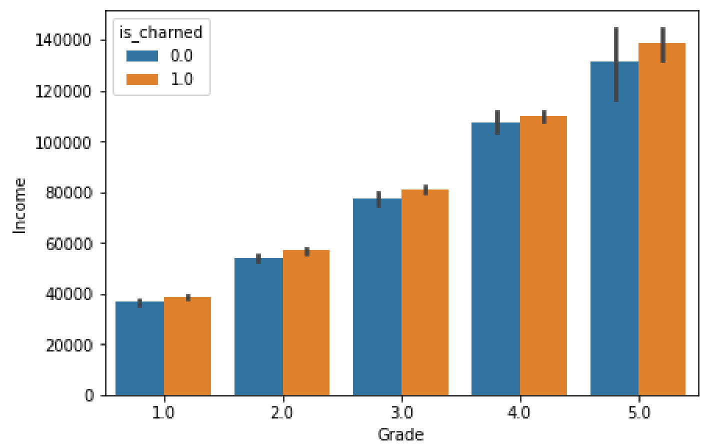


- **Insights**
  - **Drivers of Grade 1,2 and 3 are being charned more** than that of remaining grades
  - **C20, and c29 are the cities where charned rate is highest**
  - **Drivers of Joining Designation 1,2 and 3 are being charned more** than that of remaining designations
  - **Male drivers are charning more than Female drivers**

In [193…

```
plt.figure(figsize=(15, 10))
plt.subplot(2, 2, 1)
sns.scatterplot(x="tenure_months",y="Total_Business_Value_sum", data=df_merged, hue="is_
plt.subplot(2, 2, 2)
g = sns.barplot(x="Grade", y="Income", hue="is_charned", data=df_merged, estimator=np.me
g.set_xticklabels(g.get_xticklabels());
```





- **Insights**
  - **More churned rate when driver is working more than 20 months of tenure**
  - **More churned rate at higher business values beyond 20 months of tenure**
  - **Churn rate is consistent across Grade and Incomes**

Feature Transformation - categorical variable

In [58]:

```
# Converting categorical type of features from Numerical to Object for Target encoding
df_merged[["Gender","Education_Level","Joining Designation","Grade","Quarterly Rating",'
```

Filtering features by data type

Numerical features

In [59]:

```
continious_features = df_merged.select_dtypes(include=['int64','float64']).columns
continious_features
```

Out[59]:

```
Index(['Driver_ID', 'Total_Business_Value_sum', 'Age', 'Income', 'is_charned',
       'tenure_months'],
      dtype='object')
```

Categorical features

In [60]:

```
categorical_features = df_merged.select_dtypes(include=['object']).columns
categorical_features
```

Out[60]:

```
Index(['Income_increased', 'Quarterly_Rating_increased', 'generation',
       'Gender', 'City', 'Education_Level', 'Joining Designation', 'Grade',
       'Quarterly Rating', 'year_of_last_date', 'month_of_last_date',
       'day_of_last_date', 'year_of_joining', 'month_of_joining',
       'day_of_joining'],
      dtype='object')
```

Encoding

Target/Response encoding of categorical features

In [61]:
```
from category_encoders import TargetEncoder
# Using Target/Response encording for region features as there are more than two levels
te = TargetEncoder()

for feature in categorical_features:
    df_merged[feature+'_new'] = te.fit_transform(df_merged[feature],df_merged['is_charne
    df_merged[feature+'_new']
```

In [62]:
```
df_merged.drop(["Driver_ID","Gender","Education_Level","Joining Designation","Grade","Qu
```

In [63]:
```
percent_missing = df_merged.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df_merged.columns,
                                 'percent_missing': percent_missing})
missing_value_df.sort_values('percent_missing', ascending=False)
```

Out[63]:

	column_name	percent_missing
Total_Business_Value_sum	Total_Business_Value_sum	0.0
Age	Age	0.0
month_of_joining_new	month_of_joining_new	0.0
year_of_joining_new	year_of_joining_new	0.0
day_of_last_date_new	day_of_last_date_new	0.0
month_of_last_date_new	month_of_last_date_new	0.0
year_of_last_date_new	year_of_last_date_new	0.0
Quarterly Rating_new	Quarterly Rating_new	0.0
Grade_new	Grade_new	0.0
Joining Designation_new	Joining Designation_new	0.0
Education_Level_new	Education_Level_new	0.0
City_new	City_new	0.0
Gender_new	Gender_new	0.0
generation_new	generation_new	0.0
Quarterly_Rating_increased_new	Quarterly_Rating_increased_new	0.0
Income_increased_new	Income_increased_new	0.0
tenure_months	tenure_months	0.0
is_charned	is_charned	0.0
Income	Income	0.0
day_of_joining_new	day_of_joining_new	0.0

Statistical summary of the derived dataset

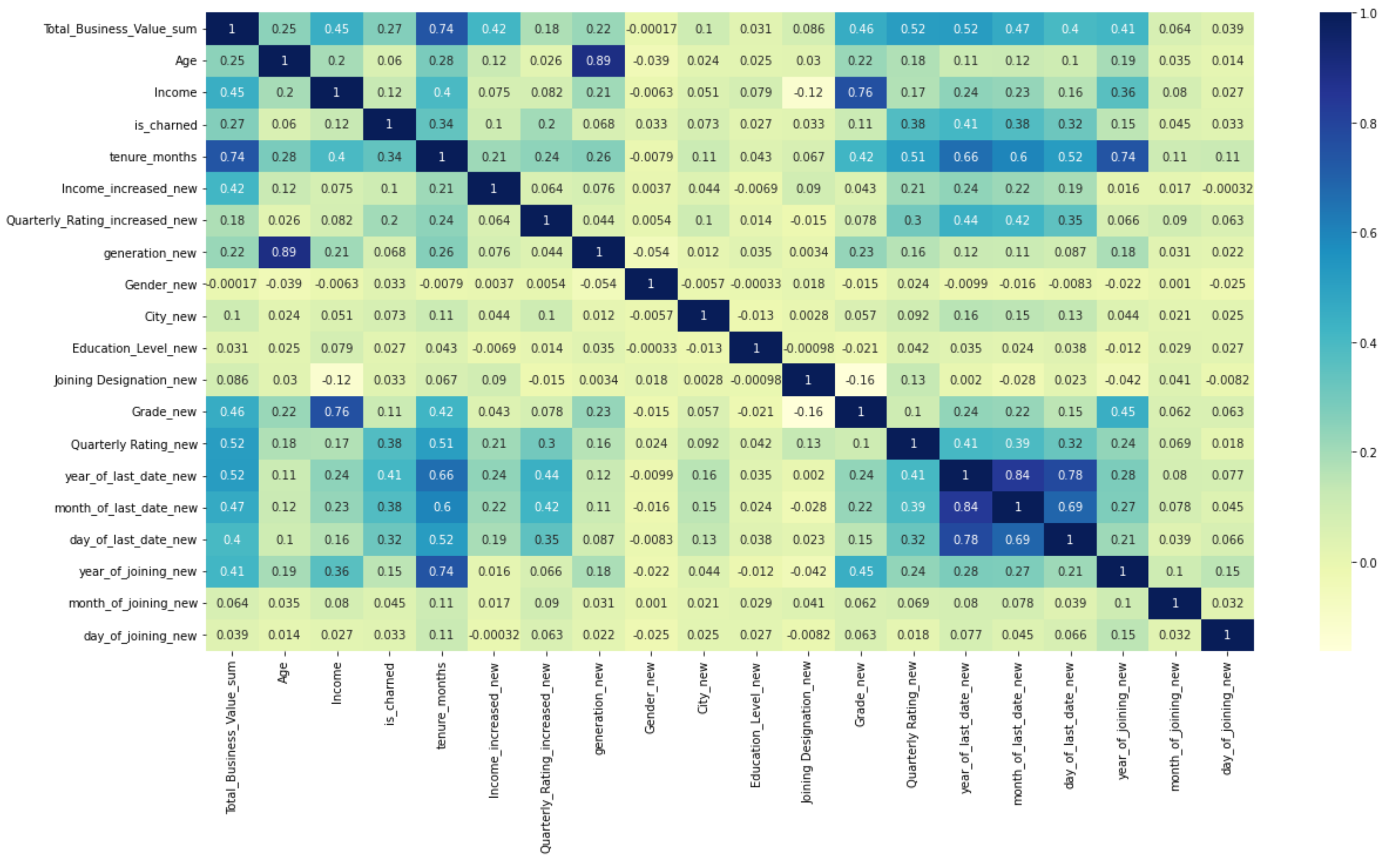
In [64]:
```
display(df_merged.describe())
```

	Total_Business_Value_sum	Age	Income	is_charned	tenure_months	Income_increased_new
count	7.024000e+03	7024.000000	7024.000000	7024.000000	7024.000000	7024.000000
mean	7.885335e+06	34.028209	62105.513240	0.769932	29.124334	0.769932
std	1.166692e+07	6.146468	29782.578527	0.420906	28.060974	0.043036
min	-1.385530e+06	21.000000	10747.000000	0.000000	0.000000	0.761461
25%	3.543600e+05	30.000000	40257.500000	1.000000	5.881024	0.761461
50%	2.613770e+06	33.000000	56848.000000	1.000000	20.287891	0.761461
75%	1.115594e+07	38.000000	79288.000000	1.000000	43.631286	0.761461
max	9.533106e+07	58.000000	188418.000000	1.000000	116.104970	0.988550

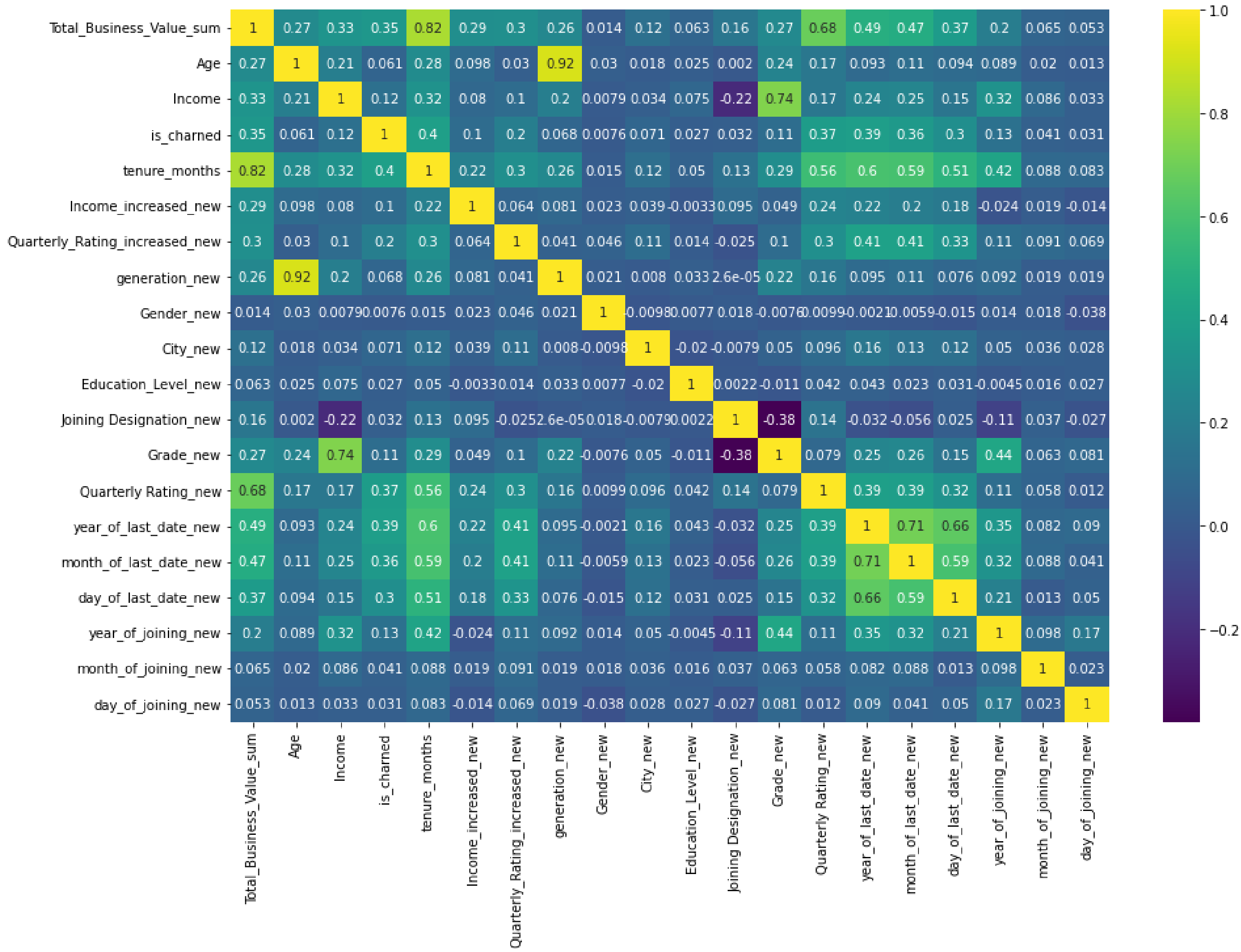
- **Insights** :
  - **outliers exists for following features**
    - **Age**
    - **Income**
    - **tenure_months**
  - **50% of Quarterly ratings is average rating**

Correlation - Independent variables

In [65]:
```
plt.figure(figsize=(20,10))
ax = sns.heatmap(df_merged.corr(method='pearson'), cmap="YlGnBu", annot=True)
```



In [66]:
```
plt.figure(figsize=(15, 10))
sns.heatmap(df_merged.corr(method='spearman'), annot=True, cmap='viridis')
plt.show()
```



- **Insights** :
  - **High correlation between following features**
    - **Age and generation (0.92)** - very likely as it's derived feature of age
    - **Tenur_months and Total Business value(0.82)**
    - **Income and Grade (0.74)**

Train Test Data

In [67]:
```
# Selcted features after first iteration  and evaluation of model .
# Removing features - 'year_of_last_date_new','month_of_last_date_new', 'day_of_last_da
#                     'month_of_joining_new', 'day_of_joining_new'
selected_features = ['Total_Business_Value_sum', 'Age', 'Income',
       'tenure_months', 'Income_increased_new',
       'Quarterly_Rating_increased_new', 'generation_new', 'Gender_new',
       'City_new', 'Education_Level_new', 'Joining Designation_new',
       'Grade_new', 'Quarterly Rating_new']
```

In [68]:
```
#df_X = df_merged.loc[ : , df_merged.columns != 'is_charned']
df_X = df_merged[['Total_Business_Value_sum', 'Age', 'Income',
       'tenure_months', 'Income_increased_new',
       'Quarterly_Rating_increased_new', 'generation_new', 'Gender_new',
       'City_new', 'Education_Level_new', 'Joining Designation_new',
       'Grade_new', 'Quarterly Rating_new']]

df_Y = df_merged["is_charned"]
```

In [69]:
```
from sklearn.model_selection import train_test_split

X_tr_cv, X_test, y_tr_cv, y_test = train_test_split(df_X, df_Y, test_size=0.2, random_st
X_train, X_val, y_train, y_val = train_test_split(X_tr_cv, y_tr_cv, test_size=0.25,rando
X_train.shape
```

Out[69]:
```
(4214, 13)
```

Standardization

In [70]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

Class Imbalance treatment

Strategy # 1 (Implemented here) - No sample data Imputation rather use class weights (for Random forest algorithm) or scale_pos_weight (for XgBoost)

In sample data, we have 76.993166% Non-leavers (i.e. marked as negative class or class 0) and 23.006834% as Leavers (i.e. marked as positive class or class 1) hence classwight / scale_pos_weight = total_negative_examples / total_positive_examples () = 76.993166/23.006834 = 3.3465

Strategy # 2 - Data Imputation i.e. [Oversampling or Undersampling or SMOTE]

- **Undersampling**
  - Selecting majority class in equal proportion to minority class
  - Will reduce data points of majority class that causes information loss
  - Hence **not a best strategy** , specially when we've rich large sample available
- **Oversampling**
  - Replicating the samples of the -ve labels such that it becomes almost same as the +ve labels
  - It will cause fabrication of data , which will **tend to overfitted model**
- **SMOTE**
  - In oversampling, we are simply repeating the data
  - But using SMOTE we are **synthetically creating new data**
  - **Second best strategy to deal with imbalance data**

Model building

Random Forest Base model with class Imbalance treatment as 'class_weight'

In [71]:
```
from sklearn.model_selection import KFold, cross_validate

tree_clf = RandomForestClassifier(random_state=7, class_weight='balanced',bootstrap=Fals
kfold = KFold(n_splits=10)
cv_acc_results = cross_validate(tree_clf, X_train, y_train, cv = kfold, scoring = 'accu
```

```
print(f"K-Fold Accuracy Mean: Train: {cv_acc_results['train_score'].mean()*100} Validat:
print(f"K-Fold Accuracy Std: Train: {cv_acc_results['train_score'].std()*100} Validatior
```

```
K-Fold Accuracy Mean: Train: 78.17327833528563 Validation: 71.66602875122423
K-Fold Accuracy Std: Train: 0.30856059172795547 Validation: 1.613130213817073
```

Hyper-parameter tuning - Bagging (Random Forest)

Randomized Search

Just like Grid Search which is an exhaustive brute for search, we can use a Random Search as well, which will try hyper-parameters randomly from a finite list of options, or from a distribution. It can be used when you want to try a hyper-parameter within a certain range with some asssociated probability

ccp_alpha- cost complexity pruning

- **PRUNING: Sometimes after we make a tree using the greedy approach and maximising information gain at each step, we eventually end up with some redundant or very less usefull branches. Hence after the tree is completed, we can now go back and merge / remove some paths / subtress inside the tree making it simpler and effecient. This is called Pruning**
- This is basicaly used for pruning the base learners
- We can control the overfitting and undefitting of the base learners using the value α, this is almost similar tp λ whixh we used in linear and logistic regression
- so the idea here is to minimise the loss associated with the decision tree and α times the number of terminal nodes (leaves) which controls overfitting
  - min(loss + α * number of leaves in the tree)
- As the depth of tree increases we know the loss decreases, where the number of leaf nodes in increases, this trade-off between the loss and number of leaves can be controlled using α like regularisation.

RandomizedSearch of tunning parameter 'ccp_alpha'

In [100…

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform


params = {'ccp_alpha': uniform(loc=0, scale=0.4)}  # sample from uniform dist between 0

tuning_function = RandomizedSearchCV(estimator = RandomForestClassifier(random_state=7,
                                                                        max_depth=10, max_feat
                               param_distributions = params,  # notice arg changeed fror
                               scoring = 'precision',
                               cv = 3,
                               n_iter=15,  # Number of times to run random combination
                               n_jobs=-1
                               )

tuning_function.fit(X_train, y_train)

parameters = tuning_function.best_params_
score = tuning_function.best_score_
print("RandomSearch RF best_params",parameters)
print("RandomSearch RF precision score", score)
```

```
RandomSearch RF best_params {'ccp_alpha': 0.004991949323641664 5}
RandomSearch RF precision score 0.9581323132842776
```

Grid Search of tunning parameter n_estimators (with best value of 'ccp_alpha')

In [101…

```
from sklearn.model_selection import GridSearchCV

params = {'n_estimators': [100,120,140,160,180,200,220, 240, 260,280,300,320,340,360,38(

tuning_function = GridSearchCV(estimator = RandomForestClassifier(random_state=7, class_
                                                                  max_depth=10, max_feat
                               param_grid = params,  # notice arg changeed from param_g:
                               scoring = 'precision',
                               cv = 3,
                               n_jobs=-1
                               )

tuning_function.fit(X_train, y_train)

parameters = tuning_function.best_params_
score = tuning_function.best_score_
print("GridSearch RF best_params",parameters)
print("GridSearch RF precision score", score)
```

```
GridSearch RF best_params {'n_estimators': 240}
GridSearch RF precision score 0.9584782562623277
```

Grid Search of tunning parameter 'max_depth' (with tunned values of 'ccp_alpha' and 'n_estimators')

In [102…

```
params = {'max_depth' : [3,5,10]}

tuning_function = GridSearchCV(estimator = RandomForestClassifier(random_state=7, class_
                                                                  max_features=8, n_est:
                               param_grid = params,  # notice arg changeed from param_g:
                               scoring = 'precision',
                               cv = 3,
                               n_jobs=-1
                               )
tuning_function.fit(X_train, y_train)

parameters = tuning_function.best_params_
score = tuning_function.best_score_
print("GridSearch RF best_params",parameters)
print("GridSearch RF precision score", score)
```

```
GridSearch RF best_params {'max_depth': 10}
GridSearch RF precision score 0.9584782562623277
```

Grid Search of tunning parameter 'max_features' (with tunned values of 'ccp_alpha' , 'n_estimators' and 'max_depth')

In [103…

```
params = {'max_features' : [8,9,10]}

tuning_function = GridSearchCV(estimator = RandomForestClassifier(random_state=7, class_
                                                                  max_depth=10, n_estima
                               param_grid = params,  # notice arg changeed from param_g:
                               scoring = 'precision',
                               cv = 3,
                               n_jobs=-1
                               )
tuning_function.fit(X_train, y_train)

parameters = tuning_function.best_params_
score = tuning_function.best_score_
print("GridSearch RF best_params",parameters)
print("GridSearch RF precision score", score)
```

```
GridSearch RF best_params {'max_features': 10}
GridSearch RF precision score 0.9623820311443501
```

In [104…:

```
tree_clf_best = RandomForestClassifier(random_state=7, class_weight='balanced',bootstrap
tree_clf_best.fit(X_train, y_train)
```

Out[104]:

```
RandomForestClassifier

RandomForestClassifier(bootstrap=False, ccp_alpha=0.004991949323641664,
                       class_weight='balanced', max_depth=10, max_features=10,
                       n_estimators=240, random_state=7)
```

In [105…:

```
core = tuning_function.best_score_
print("RF best precision score", score)
```

```
RF best precision score 0.9623820311443501
```

XGBoost

- Model with consideration
  - Hyper-parameter tuning
  - Class Imbalance treatment as 'scale_pos_weight' and

In [78]:

```
# scale_pos_weight = total_negative_examples / total_positive_examples () = 76.993166/2.
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.model_selection import StratifiedKFold

import datetime as dt

params = {
        'learning_rate': [0.1, 0.5, 0.8],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5],
        'n_estimators': [100,  200, 300, 400, 500]
        }
xgb = XGBClassifier(n_estimators=100, silent=True,scale_pos_weight = 3.3465)
```

GridSearch

In [79]:

```
from sklearn.model_selection import GridSearchCV

tuning_function = GridSearchCV(estimator = xgb,
                               param_grid = params,  # notice arg changeed from param_g
                               scoring = 'precision',
                               cv = 3,
                               n_jobs=-1
                               )

tuning_function.fit(X_train, y_train)

parameters = tuning_function.best_params_
score = tuning_function.best_score_
print("GridSearch XGBoost best_params",parameters)
print("GridSearch XGBoost precision score", score)
```

```
[21:04:01] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgboost-ci-windows/src/learner.cc:767:
Parameters: { "silent" } are not used.

GridSearch XGBoost best_params {'colsample_bytree': 1.0, 'learning_rate': 0.8, 'max_depth': 5, 'n_estimators': 400, 'subsample': 0.6}
GridSearch XGBoost precision score 0.787690403049603
```

RandomizedSearch

In [80]:

```
folds = 3

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

random_search = RandomizedSearchCV(xgb, param_distributions=params, n_iter=10, scoring=


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

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[21:04:14] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgboost-ci-windows/src/learner.cc:767:
Parameters: { "silent" } are not used.
```

In [81]:

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

```
 Best hyperparameters XGBoost (RandomizedSearch):
{'subsample': 0.6, 'n_estimators': 500, 'max_depth': 4, 'learning_rate': 0.5, 'colsample_bytree': 0.8}
```

In [116…

```
best_xgb = XGBClassifier(colsample_bytree= 1.0, learning_rate= 0.8, max_depth= 5, n_est
best_xgb.fit(X_train, y_train)
```

Out[116]:

```
▾                          XGBClassifier
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1.0,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.8, max_bin=256,
              max_cat_threshold=64, max_cat_to_onehot=4, max_delta_step=0,
              max_depth=5, max_leaves=0, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=400, n_jobs=0,
              num_parallel_tree=1, predictor='auto', random_state=0, ...)
```

In [117…

```
print(f"Time taken for training : {end - start}\nTraining accuracy:{best_xgb.score(X_tra
```

```
Time taken for training : 0:00:12.849315
Training accuracy:0.8830090175605125
Test Accuracy: 0.6149466192170818
```

Results Evaluation:

Predict with RF

In [107…

```
# Predict with Test data
y_pred_rf = tree_clf_best.predict(X_test)
# Predict with validation data
y_pred_rf_val = tree_clf_best.predict(X_val)
```

Predict with XGboot

In [118…

```
# Predict with Test data
y_pred_xgb = best_xgb.predict(X_test)
# Predict with validation data
y_pred_xgb_val = best_xgb.predict(X_val)
```

Classification Report

Bagging Classification report with Test data

In [108…

```
print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

         0.0       0.45      0.90      0.60       336
         1.0       0.96      0.66      0.78      1069

    accuracy                           0.72      1405
   macro avg       0.70      0.78      0.69      1405
weighted avg       0.84      0.72      0.74      1405
```

Bagging Classification report with validation data

In [109…

```
print(classification_report(y_test, y_pred_rf_val))
```

```
              precision    recall  f1-score   support

         0.0       0.23      0.43      0.30       336
         1.0       0.75      0.54      0.63      1069

    accuracy                           0.52      1405
   macro avg       0.49      0.49      0.46      1405
weighted avg       0.63      0.52      0.55      1405
```

- **Observation**:
  - **High Precision score i.e. 0.96 on test data**
  - However **low precision score i.e. 0.75 on validation data**
    - Even after **prunning and hyperparameter tunning (i.e with Grid Search) the model is overfitted model**
    - To **overcome overfitting following approaches can be considered**:
      - **Data cleaning** to clear out garbage input to the model
        - Missing values of **Age, Gender**
        - Outliers treatment of income
      - **Feature Engineering with consultation with domain expert**
      - **Trying SMOTE to treat class imbalance**
      - **Add more data**

Boosting Classification report with test data

In [119…

```
print(classification_report(y_test, y_pred_xgb))
```

```
              precision    recall  f1-score   support

         0.0       0.17      0.16      0.16       336
         1.0       0.74      0.76      0.75      1069

    accuracy                           0.61      1405
   macro avg       0.46      0.46      0.46      1405
weighted avg       0.60      0.61      0.61      1405
```

Boosting Classification report with validation data

In [120…

```
print(classification_report(y_test, y_pred_xgb_val))
```

```
              precision    recall  f1-score   support

         0.0       0.22      0.21      0.21       336
         1.0       0.75      0.77      0.76      1069

    accuracy                           0.63      1405
   macro avg       0.49      0.49      0.49      1405
weighted avg       0.63      0.63      0.63      1405
```

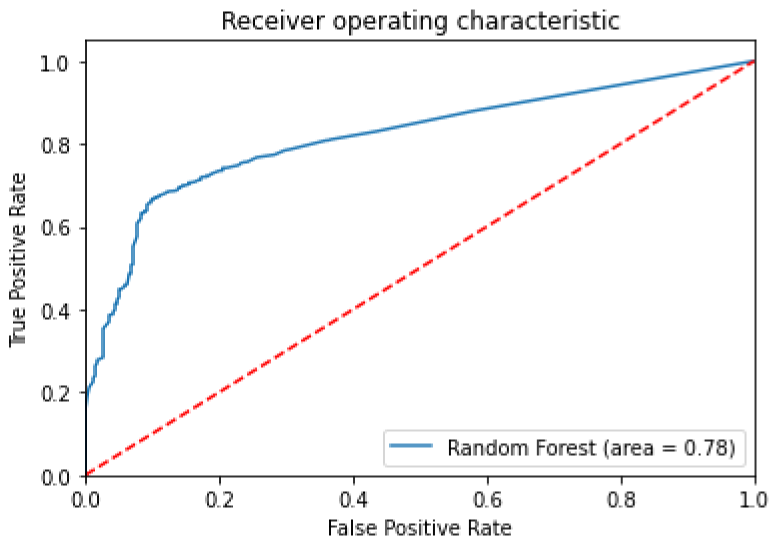
- **Observation**:
  - **Test data Precision score i.e. 0.74, Recall score 0.76 with f1 score as 0.75**
  - **Validation data Precision score i.e. 0.75 on validation data 0.77**
    - Robust model with **low score** , which needs to be improved
    - To **improve score following approaches can be considered**:
      - **Data cleaning** to clear out garbage input to the model
      - **Feature Engineering with consultation with domain expert**
      - **Trying SMOTE to treat class imbalance**
      - **Add more data points /features**
        - **More data point of charned class**
        - **More features like historical income grades , demographic specific information**

ROC AUC curve

Random Forest

In [110…

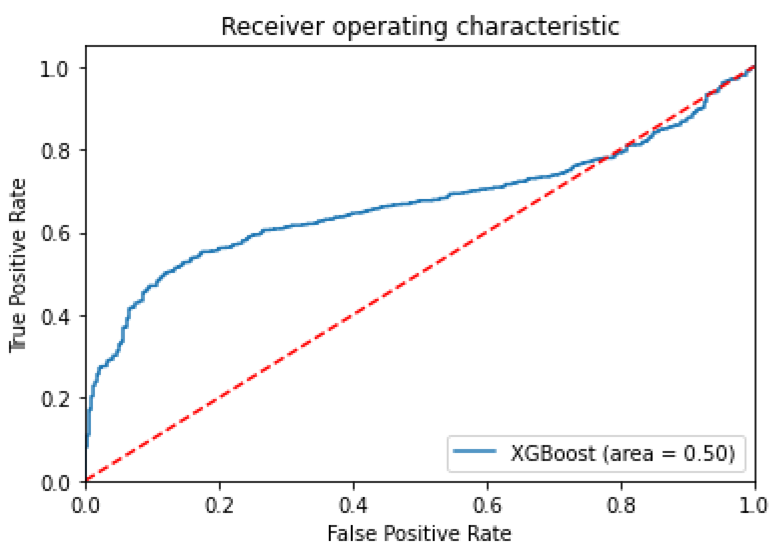
```
logit_roc_auc = roc_auc_score(y_test, y_pred_rf)
fpr, tpr, thresholds = roc_curve(y_test, tree_clf_best.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Random Forest (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



XGBoost

In [88]:

```
logit_roc_auc = roc_auc_score(y_test, y_pred_xgb)
fpr, tpr, thresholds = roc_curve(y_test, best_xgb.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='XGBoost (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



Precision/Recall curve

In [89]:

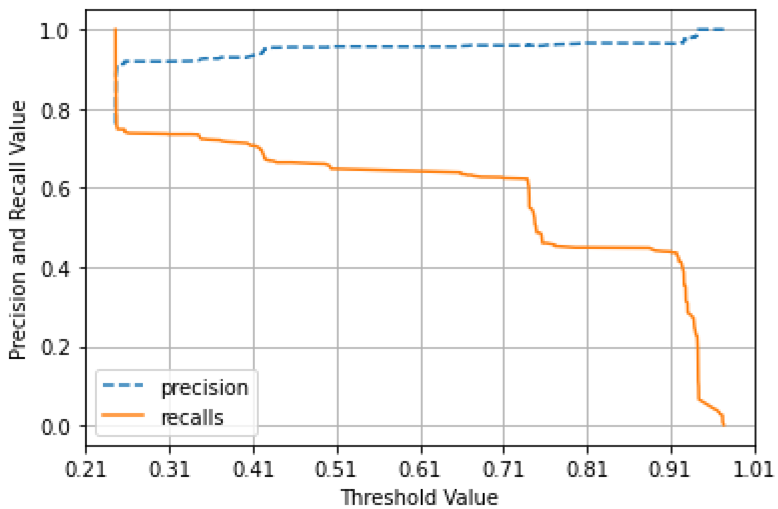
```
def precision_recall_curve_plot(y_test, pred_proba_c1):
 precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_c1)

 threshold_boundary = thresholds.shape[0]
 # plot precision
 plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='precision
 # plot recall
 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.legend(); plt.grid()
plt.show()
```

RF - Precision Recall curve

In [111…

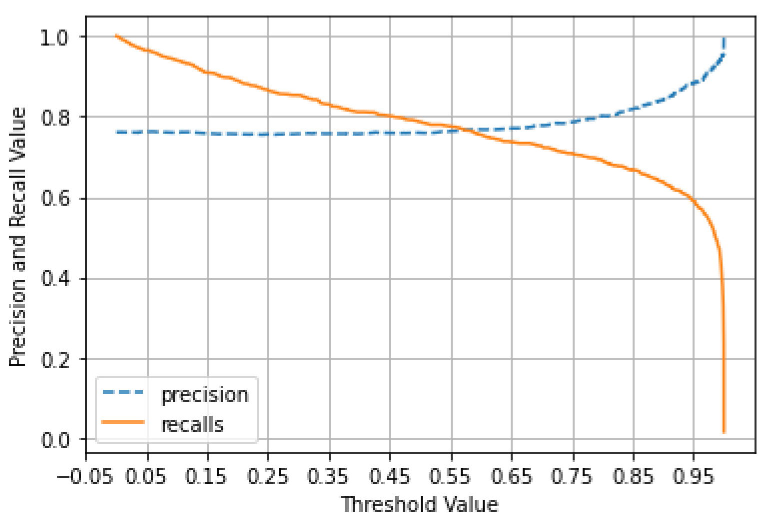
```
precision_recall_curve_plot(y_test, tree_clf_best.predict_proba(X_test)[:,1])
```



XGBoost - Precision Recall curve

In [91]:

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



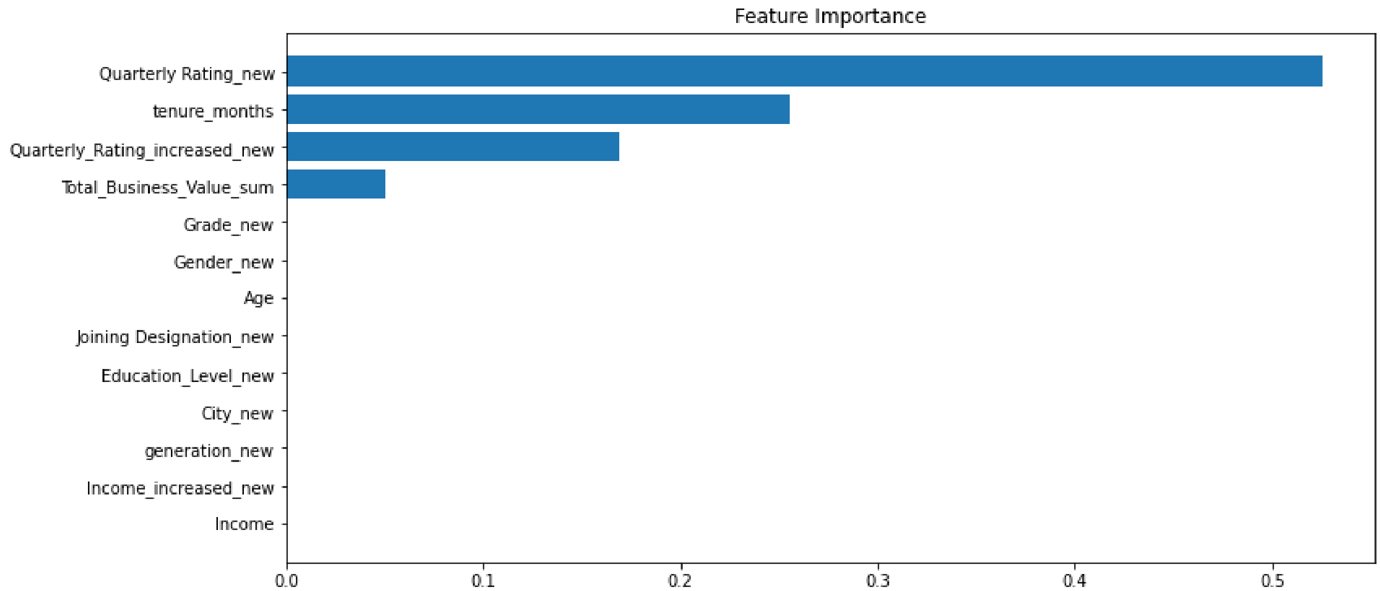
Feature Importance Random Forest

In [114…

```
feature_importance = tree_clf_best.feature_importances_
sorted_idx = np.argsort(feature_importance)
fig = plt.figure(figsize=(12, 6))
plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), np.array(df_X.columns)[sorted_idx])
plt.title('Feature Importance')
```

Out[114]:

```
Text(0.5, 1.0, 'Feature Importance')
```



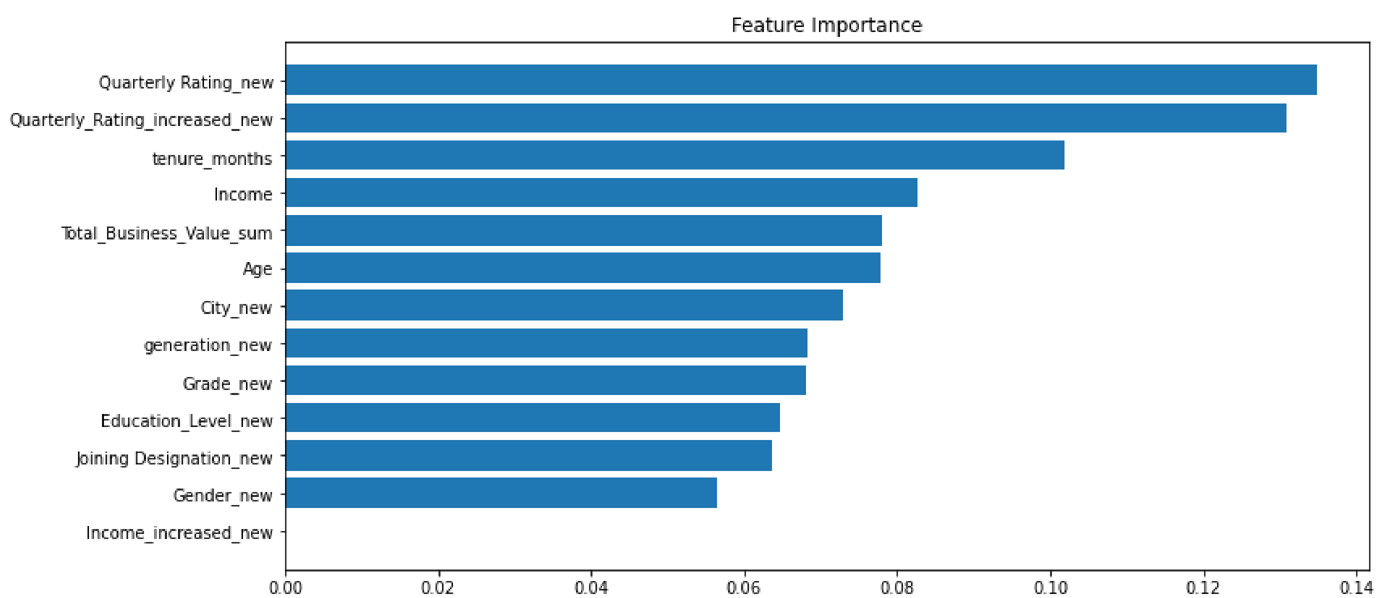
Feature Importance XGBoost

In [115…

```
feature_importance = best_xgb.feature_importances_
sorted_idx = np.argsort(feature_importance)
fig = plt.figure(figsize=(12, 6))
plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), np.array(df_X.columns)[sorted_idx])
plt.title('Feature Importance')
```

Out[115]:

```
Text(0.5, 1.0, 'Feature Importance')
```



Insights

- High correlation between following features, which may affect churn of drivers
    - Age and Generation (0.92)
    - Months driver stayed with Ola and Total Business value contributed (0.82)
    - Income and Grade (0.74)
- Younger Millennials (26-32) and Older Millennials (33-41) drivers are leaving in most numbers
- Drivers of Grade 1,2 and 3 are leaving more than that of remaining grades
- C20, and C29 are the cities where churned rate is highest

- Drivers of Joining Designation 1,2 and 3 are being churned more than that of remaining designations
- Male drivers are churning more than Female drivers
- More churned rate when driver is working more than 20 months of tenure
- More churned rate at higher business values beyond 20 months of tenure
- Churn rate is consistent across Grade and Incomes
- 50% of Quarterly ratings is average rating

Recommendations

- **Recommendations** Key considerations: Below recommendation will be more effective when more appropriate measures taken care wrt. highly correlated features, outliers, data cleaning (e,g. Age , Gender) and feature engineering are taken care as well
- **Actionable items for business**
  - Quarterly ratings, Tenure of stay, Increase in Quarterly rating , Total Business Value etc. play most significant role for predicting chances of churning of driver
  - More focus should be on following areas to reduce the chance of churning
    - Ratings – Regular Analysis of low ratings and train drivers with skills on how they can improve ratings
    - Analysis is required why there is average rating of 50% drivers
    - More benefits or commissions to drivers who have served longer tenure
      - Health benefits for committed drivers and generated high business values
    - Root cause of High churned cities must be analysed e.g. C20,C29
    - Upper Income group of Drivers can be targeted for re- investment in cabs with attractive commissions or loans
    - Target group - Younger and Older Millennials who has high tenured in OLA and have ability to pay back the loan
  - OLA can offer more flexible ‘investment option’ for owning cars to attract drivers
  - OLA can tie up with more corporate offices for good drivers and long-term commitment based on preference of the driver i.e., choice of work life balance over more trips

In [ ]:

```

```