In [1]:
```
# Importing required libraries -
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import ttest_ind # T-test for independent samples
from scipy.stats import shapiro # Shapiro-Wilk's test for Normality
#from scipy.stats import levene # Levene's test for Equality of Variance
from scipy.stats import f_oneway # One-way ANOVA
from scipy.stats import chi2_contingency # Chi-square test of independence
#from scipy.stats import kruskal


import warnings
warnings.filterwarnings("ignore")


###################
# Display Changes
###################
# cf.go_offline()
from matplotlib import rc, rcParams
%matplotlib inline
rc('axes', linewidth=2)
rc('font', weight='bold')
sns.set_style('whitegrid')
#pd.set_option('max_rows',2000)
#pd.set_option('max_columns',255)
#rcParams['figure.figsize'] = 24,8
```

In [2]:
```
import matplotlib_inline
matplotlib_inline.backend_inline.set_matplotlib_formats('svg')
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [3]:
```
import os
os.getcwd()
```

Out[3]: 'C:\\Users\\gurme\\OneDrive\\Desktop\\scaler'

In [4]:
```
import urllib.request
url = 'https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/002/492/original/ola_driver_scaler.csv'
urllib.request.urlretrieve(url, 'Ola.csv')
```

Out[4]:
```
('Ola.csv', <http.client.HTTPMessage at 0x1d7d9f18f10>)
```

In [5]:
```
df=pd.read_csv('Ola.csv')
```

Defining Problem Statement and Analysing basic metrics

Using the monthly information for a segment of drivers for 2019 and 2020 and 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). etc '

In [ ]:
```

```

In [ ]:
```

```

Observations on the shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required), missing value detection, statistical summary (10 Points)

In [6]:
```
df.shape
```

Out[6]:
```
(19104, 14)
```

In [7]:
```
# none of the variables have null values
# 3 categorical clumns and 6 numerical columns
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 [8]:
```
# precentage of null values for different variables
def check_missing(df):
    missing = np.round(df.isnull().sum()/len(df)*100 ,2)
    return missing[missing>0]
check_missing(df)
#missing values for source_name and destination_name
```

Out[8]:
```
Age                 0.32
Gender              0.27
LastWorkingDate    91.54
dtype: float64
```

In [9]:
```
#
# will explore further to check this
df.describe()
```

Out[9]:

	Unnamed: 0	Driver_ID	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating
count	19104.000000	19104.000000	19043.000000	19052.000000	19104.000000	19104.000000	19104.000000	19104.000000	1.910400e+04	19104.000000
mean	9551.500000	1415.591133	34.668435	0.418749	1.021671	65652.025126	1.690536	2.252670	5.716621e+05	2.008899
std	5514.994107	810.705321	6.257912	0.493367	0.800167	30914.515344	0.836984	1.026512	1.128312e+06	1.009832
min	0.000000	1.000000	21.000000	0.000000	0.000000	10747.000000	1.000000	1.000000	-6.000000e+06	1.000000
25%	4775.750000	710.000000	30.000000	0.000000	0.000000	42383.000000	1.000000	1.000000	0.000000e+00	1.000000
50%	9551.500000	1417.000000	34.000000	0.000000	1.000000	60087.000000	1.000000	2.000000	2.500000e+05	2.000000
75%	14327.250000	2137.000000	39.000000	1.000000	2.000000	83969.000000	2.000000	3.000000	6.997000e+05	3.000000
max	19103.000000	2788.000000	58.000000	1.000000	2.000000	188418.000000	5.000000	5.000000	3.374772e+07	4.000000

In [10]:
```
df.head()
```

Out[10]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterly Rating
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	2381060	2
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	-665480	2
2	2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18	03/11/19	1	1	0	2
3	3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2	0	1
4	4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2	0	1

In [11]:
```
df['MMM-YY'] =pd.to_datetime(df['MMM-YY'])

df['Dateofjoining'] =pd.to_datetime(df['Dateofjoining'])

```

In [12]:
```
1 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  datetime64[ns]
 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  datetime64[ns]
 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: datetime64[ns](2), float64(2), int64(8), object(2)
memory usage: 2.0+ MB
```

Missing Values imputation

In [13]:
```
1 df['LastWorkingDate']=df['LastWorkingDate'].fillna(0)
```

In [14]:
```
1 df.head()
```

Out[14]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterly Rating
0	0	2019-01-01	1	28.0	0.0	C23	2	57387	2018-12-24	0	1	1	2381060	2
1	1	2019-02-01	1	28.0	0.0	C23	2	57387	2018-12-24	0	1	1	-665480	2
2	2	2019-03-01	1	28.0	0.0	C23	2	57387	2018-12-24	03/11/19	1	1	0	2
3	3	2020-11-01	2	31.0	0.0	C7	2	67016	2020-11-06	0	2	2	0	1
4	4	2020-12-01	2	31.0	0.0	C7	2	67016	2020-11-06	0	2	2	0	1

In [15]:
```
df[df['Driver_ID']==11]
```

Out[15]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterly Rating
21	21	2020-12-01	11	28.0	1.0	C19	2	42172	2020-12-07	0	1	1	0	1

In [16]:
```
check_missing(df)
```

Out[16]:
```
Age       0.32
Gender    0.27
dtype: float64
```

In [17]:
```
def KNNimpute(df):
    # define imputer
    from sklearn.impute import KNNImputer
    imputer = KNNImputer()
    imputer.fit(df)
    return imputer.transform(df)
```

In [18]:
```
cat_col_list= [col for col in df.columns if df[col].dtype=='object' ]
```

In [19]:
```
for col in cat_col_list:
    print(f'the number of unique items in {col} is {df[col].nunique()}')
```

```
the number of unique items in City is 29
the number of unique items in LastWorkingDate is 494
```

In [20]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [21]:
```
num_col_list =[col for col in df.columns if df[col].dtype=='float64' or df[col].dtype=='int64']
num_col_list =[
 'Age',
 'Income',
 'Total Business Value']
```

In [22]:
```
num_df= df[num_col_list]
num_df
```

Out[22]:

	Age	Income	Total Business Value
0	28.0	57387	2381060
1	28.0	57387	-665480
2	28.0	57387	0
3	31.0	67016	0
4	31.0	67016	0
...	...	...	...
19099	30.0	70254	740280
19100	30.0	70254	448370
19101	30.0	70254	0
19102	30.0	70254	200420
19103	30.0	70254	411480

19104 rows × 3 columns

In [23]:
```
num_df = KNNimpute(num_df)
```

In [24]:
```
num_col_list
```

Out[24]:
```
['Age', 'Income', 'Total Business Value']
```

In [25]:
```
num_df[:5]
```

Out[25]:
```
array([[ 2.80000e+01,  5.73870e+04,  2.38106e+06],
       [ 2.80000e+01,  5.73870e+04, -6.65480e+05],
       [ 2.80000e+01,  5.73870e+04,  0.00000e+00],
       [ 3.10000e+01,  6.70160e+04,  0.00000e+00],
       [ 3.10000e+01,  6.70160e+04,  0.00000e+00]])
```

In [26]:
```
df[num_col_list]= num_df
```

In [27]:
```
check_missing(df)
```

Out[27]:
```
Gender    0.27
dtype: float64
```

In [28]:
```
df['Gender'].mode()[0]
```

Out[28]: 0.0

In [29]:
```
df['Gender'].fillna(df['Gender'].mode()[0],inplace=True)
```

Data Aggregation

In [30]:
```
df.head()
```

Out[30]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterly Rating
0	0	2019-01-01	1	28.0	0.0	C23	2	57387.0	2018-12-24	0	1	1	2381060.0	2
1	1	2019-02-01	1	28.0	0.0	C23	2	57387.0	2018-12-24	0	1	1	-665480.0	2
2	2	2019-03-01	1	28.0	0.0	C23	2	57387.0	2018-12-24	03/11/19	1	1	0.0	2
3	3	2020-11-01	2	31.0	0.0	C7	2	67016.0	2020-11-06	0	2	2	0.0	1
4	4	2020-12-01	2	31.0	0.0	C7	2	67016.0	2020-11-06	0	2	2	0.0	1

In [31]:
```
df_groupby= df.groupby('Driver_ID')
```

In [32]:
```
df_clean =pd.DataFrame()
```

In [33]:
```
df_clean[['Age','Gender','City','Education_Level','Income','Dateofjoining','LastWorkingDate','Joining Designation','Grade'
                                , 'Quarterly Rating']] = df_groupby[['Age','Gender','City','Education_Level','Income','Dateofjoini
                                , 'Quarterly Rating']].last()
```

In [ ]:
```

```

Feature Engineeering

In [34]:
```
df_clean['Total Business Value'] = df_groupby['Total Business Value'].sum()
df_clean['Tenure'] =  df_groupby['MMM-YY'].last()-df_groupby['MMM-YY'].first()
```

In [35]:
```
df_clean['rating_change'] = df_groupby['Quarterly Rating'].last()-df_groupby['Quarterly Rating'].first()
df_clean['income_change'] = df_groupby['Income'].last()-df_groupby['Income'].first()

```

In [36]:
```
df_clean.head()
```

Out[36]:

Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Quarterly Rating	Total Business Value	Tenure	rating_change	income_chang
1	28.0	0.0	C23	2	57387.0	2018-12-24	03/11/19	1	1	2	1715580.0	59 days	0	0
2	31.0	0.0	C7	2	67016.0	2020-11-06	0	2	2	1	0.0	30 days	0	0
4	43.0	0.0	C13	2	65603.0	2019-12-07	27/04/20	2	2	1	350000.0	122 days	0	0
5	29.0	0.0	C9	0	46368.0	2019-01-09	03/07/19	1	1	1	120360.0	59 days	0	0
6	31.0	1.0	C11	1	78728.0	2020-07-31	0	3	3	2	1265000.0	122 days	1	0

In [37]:
```
df_clean['rating_change'].unique()
```

Out[37]:
```
array([ 0,  1, -3, -1, -2,  2,  3], dtype=int64)
```

In [38]:
```
df_clean['rating_change']= df_clean['rating_change'].apply(lambda x: 'Increased' if x>0 else 'Same' if x==0 else "Decreased")
df_clean['income_change']= df_clean['income_change'].apply(lambda x: 'Increased' if x>0 else 'Same' if x==0 else "Decreased")
```

In [39]:
```
df_clean.head()
```

Out[39]:

Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Quarterly Rating	Total Business Value	Tenure	rating_change	income_chang
1	28.0	0.0	C23	2	57387.0	2018-12-24	03/11/19	1	1	2	1715580.0	59 days	Same	Sam
2	31.0	0.0	C7	2	67016.0	2020-11-06	0	2	2	1	0.0	30 days	Same	Sam
4	43.0	0.0	C13	2	65603.0	2019-12-07	27/04/20	2	2	1	350000.0	122 days	Same	Sam
5	29.0	0.0	C9	0	46368.0	2019-01-09	03/07/19	1	1	1	120360.0	59 days	Same	Sam
6	31.0	1.0	C11	1	78728.0	2020-07-31	0	3	3	2	1265000.0	122 days	Increased	Sam

In [40]:
```
df_clean['Target']= np.where(df_clean['LastWorkingDate'] !=0,1,0)
df_clean.reset_index(inplace=True)
```

In [41]:
```
# Creating some more features
df_clean['Joining Month'] = df_clean['Dateofjoining'].dt.month
df_clean['Joining year'] = df_clean['Dateofjoining'].dt.year

df_clean['Tenure'] = df_clean['Tenure'].dt.days
```

In [42]:
```
df_clean.head()
```

Out[42]:

	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Quarterly Rating	Total Business Value	Tenure	rating_change	income_ch
0	1	28.0	0.0	C23	2	57387.0	2018-12-24	03/11/19	1	1	2	1715580.0	59	Same	S
1	2	31.0	0.0	C7	2	67016.0	2020-11-06	0	2	2	1	0.0	30	Same	S
2	4	43.0	0.0	C13	2	65603.0	2019-12-07	27/04/20	2	2	1	350000.0	122	Same	S
3	5	29.0	0.0	C9	0	46368.0	2019-01-09	03/07/19	1	1	1	120360.0	59	Same	S
4	6	31.0	1.0	C11	1	78728.0	2020-07-31	0	3	3	2	1265000.0	122	Increased	S

In [43]:
```
df_clean.head()
```

Out[43]:

	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Quarterly Rating	Total Business Value	Tenure	rating_change	income_ch
0	1	28.0	0.0	C23	2	57387.0	2018-12-24	03/11/19	1	1	2	1715580.0	59	Same	
1	2	31.0	0.0	C7	2	67016.0	2020-11-06	0	2	2	1	0.0	30	Same	
2	4	43.0	0.0	C13	2	65603.0	2019-12-07	27/04/20	2	2	1	350000.0	122	Same	
3	5	29.0	0.0	C9	0	46368.0	2019-01-09	03/07/19	1	1	1	120360.0	59	Same	
4	6	31.0	1.0	C11	1	78728.0	2020-07-31	0	3	3	2	1265000.0	122	Increased	

In [44]:
```
#Dropping unnecassry columns
df_clean.drop(['Driver_ID','LastWorkingDate'],inplace=True,axis=1)
```

In [45]:
```
df_clean.drop(['Dateofjoining'],inplace=True,axis=1
              )
```

Data Exploration -

In [46]:
```
df_clean.head()
```

Out[46]:

	Age	Gender	City	Education_Level	Income	Joining Designation	Grade	Quarterly Rating	Total Business Value	Tenure	rating_change	income_change	Target	Joining Month	Joining year
0	28.0	0.0	C23	2	57387.0	1	1	2	1715580.0	59	Same	Same	1	12	2018
1	31.0	0.0	C7	2	67016.0	2	2	1	0.0	30	Same	Same	0	11	2020
2	43.0	0.0	C13	2	65603.0	2	2	1	350000.0	122	Same	Same	1	12	2019
3	29.0	0.0	C9	0	46368.0	1	1	1	120360.0	59	Same	Same	1	1	2019
4	31.0	1.0	C11	1	78728.0	3	3	2	1265000.0	122	Increased	Same	0	7	2020

In [ ]:
```

```

In [47]:
```
1 df_clean.groupby(by='Target')['Income'].describe()
```

Out[47]:

Target	count	mean	std	min	25%	50%	75%	max
0	765.0	67662.90719	29578.579157	12938.0	46131.0	64154.0	84573.00	188418.0
1	1616.0	55391.40099	26924.959518	10747.0	36117.5	51630.0	69816.75	167758.0

In [48]:
```
1 df_clean['Education_Level'].value_counts()
```

Out[48]:
```
2    802
1    795
0    784
Name: Education_Level, dtype: int64
```

In [49]:
```
1 df_clean.groupby(by='Target')['Total Business Value'].describe()
```

Out[49]:

Target	count	mean	std	min	25%	50%	75%	max
0	765.0	9.620626e+06	1.323157e+07	0.0	240920.0	2636210.0	16290800.0	95331060.0
1	1616.0	2.203746e+06	4.717754e+06	-1385530.0	0.0	465025.0	2345702.5	60153830.0

In [50]:
```
1 df_clean.groupby(by='Target')['Tenure'].describe()
```

Out[50]:

Target	count	mean	std	min	25%	50%	75%	max
0	765.0	317.666667	274.769928	0.0	61.0	183.0	700.0	700.0
1	1616.0	164.916460	140.248902	0.0	61.0	122.0	214.0	700.0

In [ ]:
```
1
```

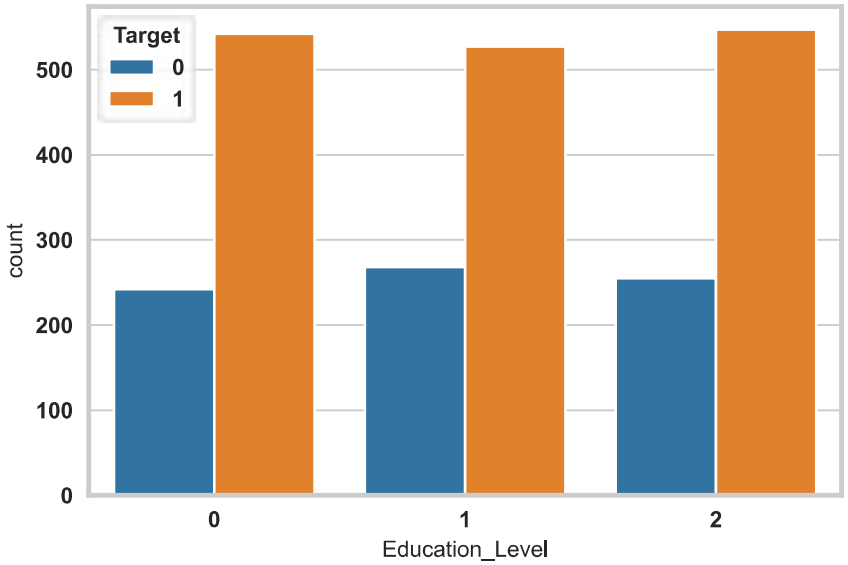
In [ ]:
```
1
```

In [ ]:
```
1
```

Data Visualization

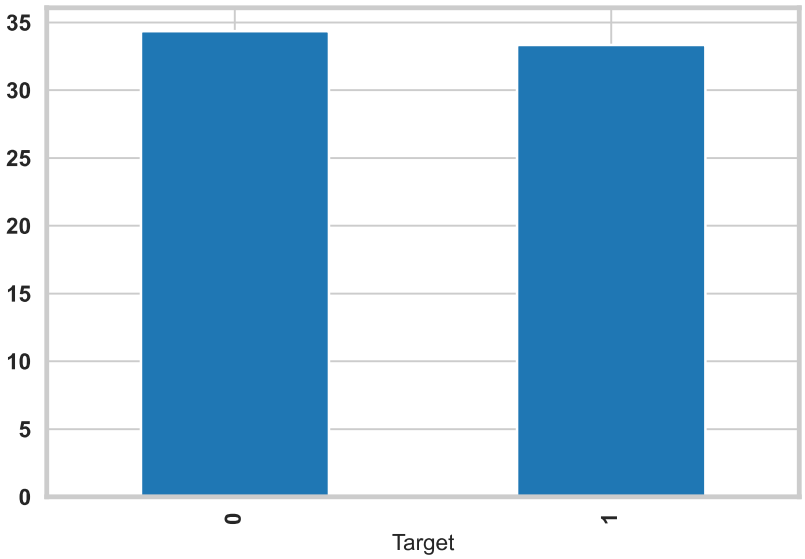
In [51]:
```
1 sns.countplot(data=df_clean,x='Education_Level',hue='Target')
2 plt.xticks()
```

Out[51]: (array([0, 1, 2]), [Text(0, 0, '0'), Text(1, 0, '1'), Text(2, 0, '2')])



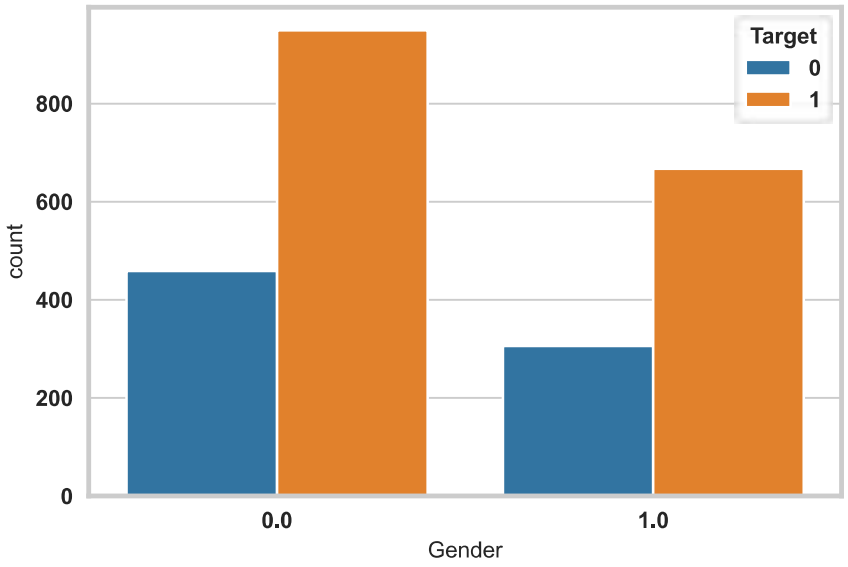
In [52]:
```
1
2 df_clean.groupby('Target')['Age'].mean().plot(kind='bar')
```

Out[52]: <AxesSubplot:xlabel='Target'>



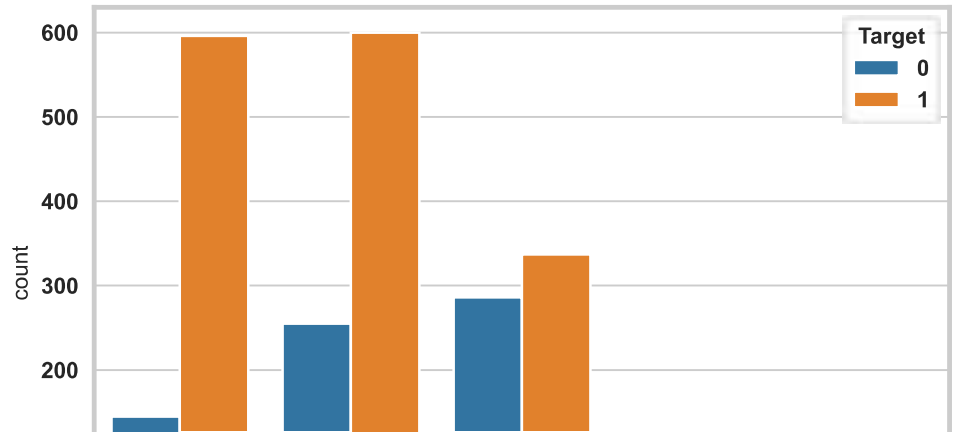
In [53]:
```
1 sns.countplot(data=df_clean,x='Gender',hue='Target')
```

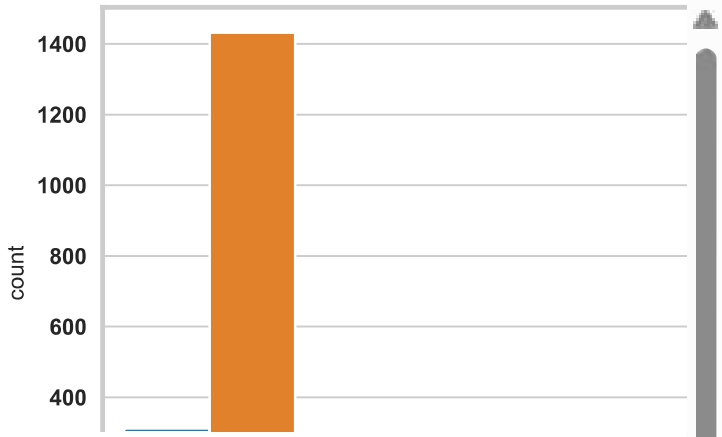
Out[53]: <AxesSubplot:xlabel='Gender', ylabel='count'>



In [54]:
```
plt.figure(figsize=(15, 20))

plt.subplot(4, 2, 1)
sns.countplot(x='Grade', data=df_clean, hue='Target')

plt.subplot(4, 2, 2)
sns.countplot(x='Quarterly Rating', data=df_clean, hue='Target')

plt.subplot(4, 2, 3)
sns.countplot(x='Joining Designation', data=df_clean, hue='Target')

plt.subplot(4, 2, 4)
sns.countplot(x='rating_change', data=df_clean, hue='Target')

plt.subplot(4, 2, 5)
sns.countplot(x='income_change', data=df_clean, hue='Target')
```

Out[54]:
```
<AxesSubplot:xlabel='income_change', ylabel='count'>
```



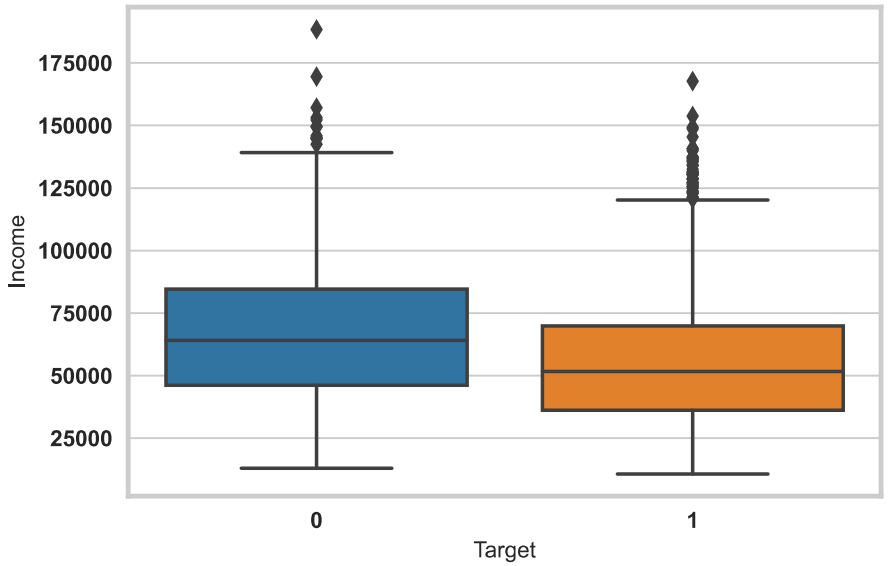


as expected , People who had increased rating change and income change are less likely to churn

In [ ]:
```
1
```

In [55]:
```
1 sns.boxplot(x='Target',y='Income',data=df_clean)
```

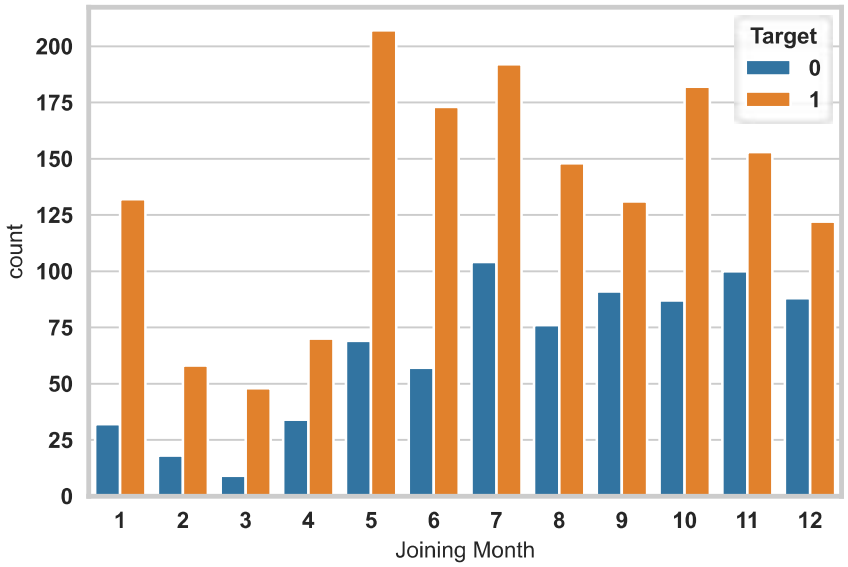
Out[55]:
```
<AxesSubplot:xlabel='Target', ylabel='Income'>
```



Clealy People who churn have lower median income

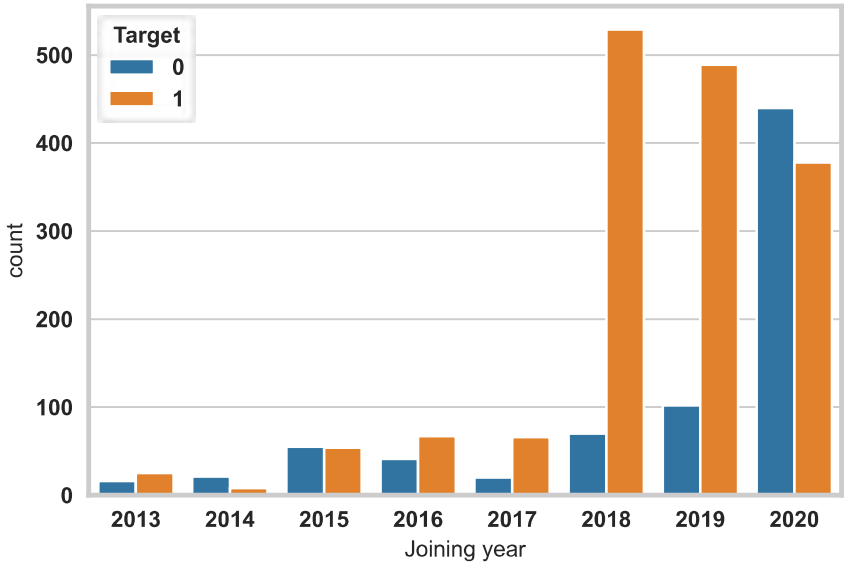
In [56]:
```
1 sns.countplot(data=df_clean,x='Joining Month',hue='Target')
```

Out[56]: <AxesSubplot:xlabel='Joining Month', ylabel='count'>



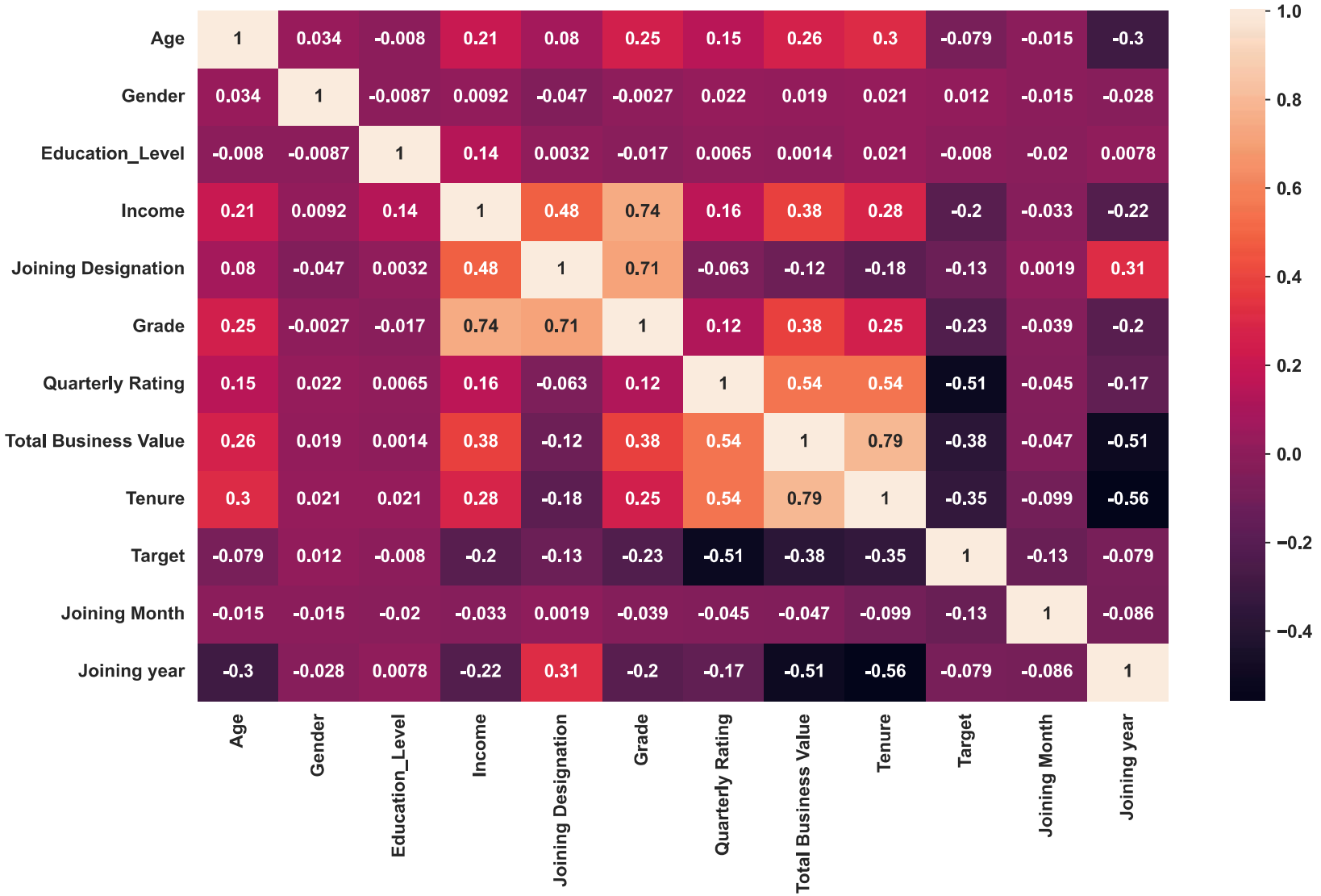
In [57]:
```
1 sns.countplot(data=df_clean,x='Joining year',hue='Target')
```

Out[57]: <AxesSubplot:xlabel='Joining year', ylabel='count'>



In [58]:
```
plt.figure(figsize=(12,7))
sns.heatmap(df_clean.corr(),annot=True)

```

Out[58]: <AxesSubplot:>



NOne of the variables seems highly correlated, lets continue for now

In [ ]:
```
1
```

Encoding & Standardization

In [59]:
```
1 from imblearn.over_sampling import SMOTE
2 from sklearn.model_selection import train_test_split
3 from sklearn.preprocessing import StandardScaler ,OneHotEncoder
4 from sklearn.pipeline import make_pipeline
5 from sklearn.compose import ColumnTransformer
```

In [60]:
```
1 df_clean.columns
```

Out[60]:
```
Index(['Age', 'Gender', 'City', 'Education_Level', 'Income',
       'Joining Designation', 'Grade', 'Quarterly Rating',
       'Total Business Value', 'Tenure', 'rating_change', 'income_change',
       'Target', 'Joining Month', 'Joining year'],
      dtype='object')
```

In [61]:
```
1 categorical_columns = ['City','rating_change', 'income_change']
2 non_categorical_columns = ['Age', 'Gender',  'Education_Level', 'Income',
3        'Joining Designation', 'Grade', 'Quarterly Rating',
4        'Total Business Value', 'Tenure',
5         'Joining Month', 'Joining year']
```

In [62]:
```
1 preprocessor = ColumnTransformer(
2     transformers=[
3         ('cat', OneHotEncoder(), categorical_columns),  # One-hot encode categorical columns
4         ('num', 'passthrough', non_categorical_columns)  # Leave non-categorical columns as-is
5     ])
6 scaler =StandardScaler()
7 pipeline = make_pipeline(preprocessor, scaler)
```

In [63]:
```
1 X = df_clean.drop(['Target'],axis=1)
2 y= df_clean['Target']
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [64]:
```
1 X_train_scaled= pipeline.fit_transform(X_train)
2 X_test_scaled= pipeline.transform(X_test)
3
```

Class Imbalance

we have class imbalance, lets use SMOTE

In [65]:
```
1 y_train.value_counts()
```

Out[65]:
```
1    1289
0     615
Name: Target, dtype: int64
```

In [66]:
```
1 ros= SMOTE()
```

In [67]:
```
1 X_resampled, y_resampled = ros.fit_resample(X_train_scaled, y_train)
```

In [ ]:
```
1
```

Model Building

Ensemble - Bagging Algorithm Randon forest

In [68]:
```
1 from sklearn.ensemble import RandomForestClassifier
2 from sklearn.model_selection import KFold, cross_validate
```

In [69]:
```
1 rfc = RandomForestClassifier(random_state=7, n_estimators=100) ## i am letting each DT overfit
2
3 kfold = KFold(n_splits=10)
4 cv_acc_results = cross_validate(rfc, X_resampled, y_resampled, cv = kfold,
5                                 scoring = 'accuracy',
6                                 return_train_score = True)
7
8 print(f"K-Fold Accuracy Mean: Train: {cv_acc_results['train_score'].mean().round(3)*100} Validation: {cv_acc_results['test_score'].
9
```

```
K-Fold Accuracy Mean: Train: 100.0 Validation: 93.60000000000001
```

In [70]:

```
# lets try Grid search to tune hyperparameters

from sklearn.model_selection import GridSearchCV , RandomizedSearchCV
params = {
    'max_depth' : [7,10, 15, 20],
    'n_estimators' : [100,200,300,400],
    'max_features' : [5, 10, 20, 30]
}

```

In [71]:
```
model = RandomizedSearchCV(rfc, param_distributions =  params, scoring="accuracy", cv = 3, verbose=5)

# model
model.fit(X_resampled, np.ravel(y_resampled))
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[CV 1/3] END max_depth=10, max_features=5, n_estimators=400;, score=0.908 total time=   2.2s
[CV 2/3] END max_depth=10, max_features=5, n_estimators=400;, score=0.929 total time=   2.2s
[CV 3/3] END max_depth=10, max_features=5, n_estimators=400;, score=0.939 total time=   2.2s
[CV 1/3] END max_depth=10, max_features=10, n_estimators=300;, score=0.909 total time=   2.2s
[CV 2/3] END max_depth=10, max_features=10, n_estimators=300;, score=0.932 total time=   2.1s
[CV 3/3] END max_depth=10, max_features=10, n_estimators=300;, score=0.942 total time=   2.0s
[CV 1/3] END max_depth=20, max_features=5, n_estimators=400;, score=0.903 total time=   2.5s
[CV 2/3] END max_depth=20, max_features=5, n_estimators=400;, score=0.928 total time=   2.4s
[CV 3/3] END max_depth=20, max_features=5, n_estimators=400;, score=0.951 total time=   2.5s
[CV 1/3] END max_depth=20, max_features=30, n_estimators=400;, score=0.902 total time=   5.3s
[CV 2/3] END max_depth=20, max_features=30, n_estimators=400;, score=0.929 total time=   5.1s
[CV 3/3] END max_depth=20, max_features=30, n_estimators=400;, score=0.959 total time=   5.1s
[CV 1/3] END max_depth=15, max_features=20, n_estimators=100;, score=0.910 total time=   1.0s
[CV 2/3] END max_depth=15, max_features=20, n_estimators=100;, score=0.931 total time=   0.9s
[CV 3/3] END max_depth=15, max_features=20, n_estimators=100;, score=0.956 total time=   0.9s
[CV 1/3] END max_depth=15, max_features=30, n_estimators=100;, score=0.899 total time=   1.4s
[CV 2/3] END max_depth=15, max_features=30, n_estimators=100;, score=0.929 total time=   1.2s
[CV 3/3] END max_depth=15, max_features=30, n_estimators=100;, score=0.958 total time=   1.2s
[CV 1/3] END max_depth=10, max_features=10, n_estimators=200;, score=0.912 total time=   1.5s
[CV 2/3] END max_depth=10, max_features=10, n_estimators=200;, score=0.930 total time=   1.4s
[CV 3/3] END max_depth=10, max_features=10, n_estimators=200;, score=0.943 total time=   1.3s
[CV 1/3] END max_depth=20, max_features=20, n_estimators=300;, score=0.908 total time=   3.1s
[CV 2/3] END max_depth=20, max_features=20, n_estimators=300;, score=0.931 total time=   3.0s
[CV 3/3] END max_depth=20, max_features=20, n_estimators=300;, score=0.959 total time=   3.0s
[CV 1/3] END max_depth=10, max_features=10, n_estimators=400;, score=0.907 total time=   2.9s
[CV 2/3] END max_depth=10, max_features=10, n_estimators=400;, score=0.934 total time=   2.7s
[CV 3/3] END max_depth=10, max_features=10, n_estimators=400;, score=0.945 total time=   2.8s
[CV 1/3] END max_depth=20, max_features=30, n_estimators=300;, score=0.899 total time=   4.0s
[CV 2/3] END max_depth=20, max_features=30, n_estimators=300;, score=0.927 total time=   3.9s
[CV 3/3] END max_depth=20, max_features=30, n_estimators=300;, score=0.959 total time=   3.8s
```

Out[71]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(random_state=7),
                   param_distributions={'max_depth': [7, 10, 15, 20],
                                        'max_features': [5, 10, 20, 30],
                                        'n_estimators': [100, 200, 300, 400]},
                   scoring='accuracy', verbose=5)
```

In [72]:
```
model.best_params_
```

Out[72]:
```
{'n_estimators': 300, 'max_features': 20, 'max_depth': 20}
```

In [73]:
```
print(model.best_score_)
```

```
0.9329033218723772
```

In [74]:
```
best_model =model.best_estimator_
```

In [79]:
```
from sklearn.metrics import accuracy_score,confusion_matrix, classification_report, roc_curve, roc_auc_score, precision_recall_curv

y_pred = best_model.predict(X_test_scaled)
accuracy_score(y_test,y_pred)

```

Out[79]:
```
0.9329140461215933
```

In [80]:
```
# we have the accuracy of  93 % on test data which is pretty decent ,
```

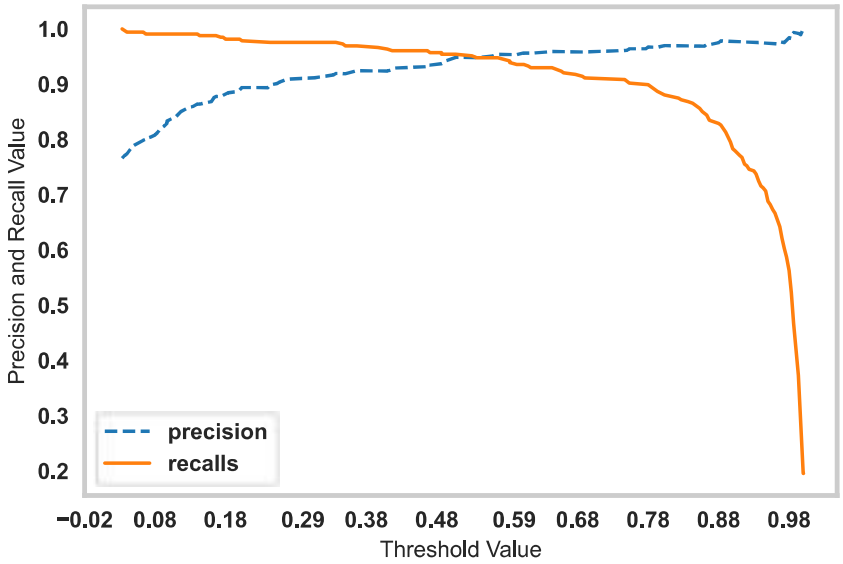
In [81]:
```
y_prob = best_model.predict_proba(X_test_scaled)

```

In [82]:
```
# Evaluate performance using accuracy score and confusion matrix
print(f"Accuracy score: {round(accuracy_score(y_test, y_pred), 3)}\n")
print(f"Confusion matrix\n {confusion_matrix(y_test, y_pred)}\n")
print(f"Classification report\n {classification_report(y_test, y_pred)}\n")
print(f"ROC-AUC Score\n {round(roc_auc_score(y_test, y_prob[:,1]), 3)}")
```

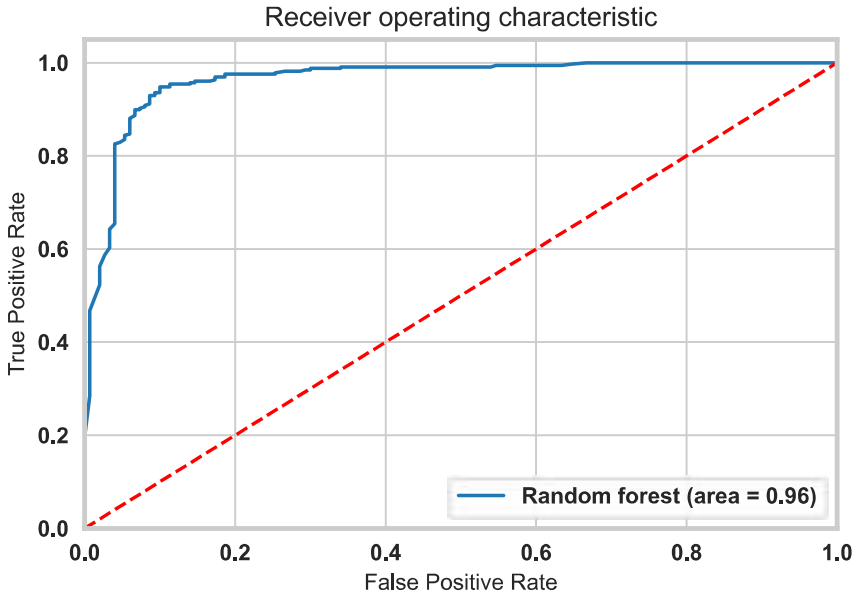
```
Accuracy score: 0.933

Confusion matrix
 [[133  17]
 [ 15 312]]

Classification report
              precision    recall  f1-score   support

           0       0.90      0.89      0.89       150
           1       0.95      0.95      0.95       327

    accuracy                           0.93       477
   macro avg       0.92      0.92      0.92       477
weighted avg       0.93      0.93      0.93       477


ROC-AUC Score
 0.964
```

In [ ]:
```
1
```

In [83]:
```
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()

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



In [84]:
```
roc_auc = roc_auc_score(y_test, y_prob[:,1])
fpr, tpr, thresholds = roc_curve(y_test, y_prob[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Random forest (area = %0.2f)' % 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()
```



In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Boosting - LightGBM & XGboost

XGBoost

In [85]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\gurme\anaconda3\lib\site-packages (2.0.0)
Requirement already satisfied: numpy in c:\users\gurme\anaconda3\lib\site-packages (from xgboost) (1.22.4)
Requirement already satisfied: scipy in c:\users\gurme\anaconda3\lib\site-packages (from xgboost) (1.7.3)
```

In [86]:
```
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_score, precision_recall_curve

```

In [ ]:
```

```

In [ ]:
```

```

In [87]:
```
params = {
        'min_child_weight': [1, 5, 10],
        'gamma': [0.5, 1, 1.5, 2, 5],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5]
        }
```

In [88]:
```
xgb = XGBClassifier(learning_rate=0.02, n_estimators=600, objective='binary:logistic',
                    silent=True, nthread=1)
```

In [89]:
```
folds = 3
param_comb = 5
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 1001)

random_search = RandomizedSearchCV(xgb, param_distributions=params, n_iter=param_comb, scoring='accuracy',
                                   n_jobs=4, cv=3, verbose=3, random_state=1001 )
```

In [90]:
```
1
2 random_search.fit(X_resampled, y_resampled)
```

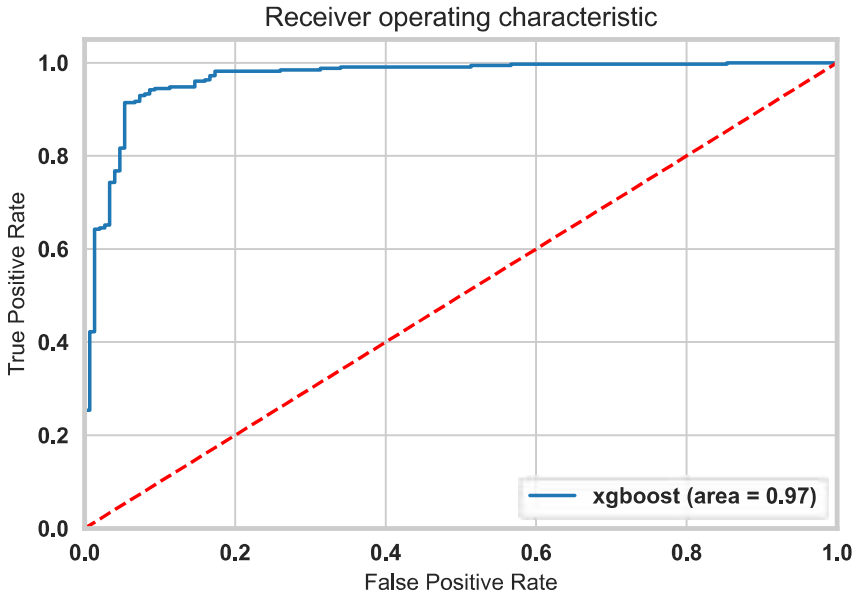
```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

Out[90]:
```
RandomizedSearchCV(cv=3,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           callbacks=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, device=None,
                                           early_stopping_rounds=None,
                                           enable_categorical=False,
                                           eval_metric=None, feature_types=None,
                                           gamma=None, grow_policy=None,
                                           importance_type=None,
                                           interaction_constraints=None,
                                           learning_rate...
                                           min_child_weight=None, missing=nan,
                                           monotone_constraints=None,
                                           multi_strategy=None,
                                           n_estimators=600, n_jobs=None,
                                           nthread=1, num_parallel_tree=None, ...),
                   n_iter=5, n_jobs=4,
                   param_distributions={'colsample_bytree': [0.6, 0.8, 1.0],
                                        'gamma': [0.5, 1, 1.5, 2, 5],
                                        'max_depth': [3, 4, 5],
                                        'min_child_weight': [1, 5, 10],
                                        'subsample': [0.6, 0.8, 1.0]},
                   random_state=1001, scoring='accuracy', verbose=3)
```

In [91]:

```
print('\n All results:')
print(random_search.cv_results_)
print('\n Best estimator:')
print(random_search.best_estimator_)
print('\n Best normalized gini score for %d-fold search with %d parameter combinations:' % (folds, param_comb))
print(random_search.best_score_ * 2 - 1)
print('\n Best hyperparameters:')
print(random_search.best_params_)
results = pd.DataFrame(random_search.cv_results_)
results.to_csv('xgb-random-grid-search-results-01.csv', index=False)
```

In [97]:
```
roc_auc = roc_auc_score(y_test, y_prob[:,1])
fpr, tpr, thresholds = roc_curve(y_test, y_prob[:,1])
plt.figure()
plt.plot(fpr, tpr, label='xgboost (area = %0.2f)' % 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()
```



LightBGM

In [98]:
```
param_grid = {
"num_leaves": [31, 63, 127],
"max_depth": [-1, 3, 5],
"subsample": [0.8, 1.0],
"colsample_bytree": [0.8, 1.0]
}
```

In [99]:
```
!pip install lightgbm
```

```
Requirement already satisfied: lightgbm in c:\users\gurme\anaconda3\lib\site-packages (4.1.0)
Requirement already satisfied: numpy in c:\users\gurme\anaconda3\lib\site-packages (from lightgbm) (1.22.4)
Requirement already satisfied: scipy in c:\users\gurme\anaconda3\lib\site-packages (from lightgbm) (1.7.3)
```

In [100]:
```
import lightgbm as lgb
```

In [101]:
```
# Create an instance of the LightGBM classifier
lgbm = lgb.LGBMClassifier(objective="binary", metric="accuracy", random_state=42)
```

In [102]:
```
random_search = RandomizedSearchCV(lgbm, param_distributions=param_grid, n_iter=5, scoring='accuracy',
                                   n_jobs=4, cv=3, verbose=3, random_state=1001 )
```

In [103]:
```
random_search.fit(X_resampled,y_resampled)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
[LightGBM] [Info] Number of positive: 1289, number of negative: 1289
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001140 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2723
[LightGBM] [Info] Number of data points in the train set: 2578, number of used features: 45
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[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
```

Out[103]:
```
RandomizedSearchCV(cv=3,
                   estimator=LGBMClassifier(metric='accuracy',
                                            objective='binary',
                                            random_state=42),
                   n_iter=5, n_jobs=4,
                   param_distributions={'colsample_bytree': [0.8, 1.0],
                                        'max_depth': [-1, 3, 5],
                                        'num_leaves': [31, 63, 127],
                                        'subsample': [0.8, 1.0]},
                   random_state=1001, scoring='accuracy', verbose=3)
```

In [104]:

```
print('\n All results:')
print(random_search.cv_results_)
print('\n Best estimator:')
print(random_search.best_estimator_)
print('\n Best normalized gini score for %d-fold search with %d parameter combinations:' % (folds, param_comb))
print(random_search.best_score_ * 2 - 1)
print('\n Best hyperparameters:')
print(random_search.best_params_)
results = pd.DataFrame(random_search.cv_results_)
results.to_csv('xgb-random-grid-search-results-01.csv', index=False)
```

```
 All results:
{'mean_fit_time': array([0.71018442, 0.27947617, 0.2317001 , 0.22844442, 0.17791533]), 'std_fit_time': array([0.02739269, 0.09187242,
0.02029487, 0.01629035, 0.04128941]), 'mean_score_time': array([0.01041738, 0.01041834, 0.00838017, 0.        , 0.01041722]), 'std_sco
re_time': array([0.0073662 , 0.00736688, 0.00642829, 0.        , 0.00736609]), 'param_subsample': masked_array(data=[1.0, 0.8, 0.8, 1.
0, 1.0],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_num_leaves': masked_array(data=[63, 127, 127, 127, 127],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_max_depth': masked_array(data=[5, 3, 3, 3, 3],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_colsample_bytree': masked_array(data=[0.8, 1.0, 0.8, 0.8, 1.0],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'subsample': 1.0, 'num_leaves': 63, 'max_depth': 5, 'colsample_bytree': 0.8}, {'subsample': 0.
8, 'num_leaves': 127, 'max_depth': 3, 'colsample_bytree': 1.0}, {'subsample': 0.8, 'num_leaves': 127, 'max_depth': 3, 'colsample_bytre
e': 0.8}, {'subsample': 1.0, 'num_leaves': 127, 'max_depth': 3, 'colsample_bytree': 0.8}, {'subsample': 1.0, 'num_leaves': 127, 'max_d
epth': 3, 'colsample_bytree': 1.0}], 'split0_test_score': array([0.90116279, 0.90813953, 0.91744186, 0.91744186, 0.90813953]), 'split1
_test_score': array([0.93946449, 0.93597206, 0.94062864, 0.94062864, 0.93597206]), 'split2_test_score': array([0.96274738, 0.95343423,
0.95925495, 0.95925495, 0.95343423]), 'mean_test_score': array([0.93445822, 0.93251527, 0.93910848, 0.93910848, 0.93251527]), 'std_tes
t_score': array([0.02538979, 0.01865233, 0.01710393, 0.01710393, 0.01865233]), 'rank_test_score': array([3, 4, 1, 1, 4])}

 Best estimator:
LGBMClassifier(colsample_bytree=0.8, max_depth=3, metric='accuracy',
               num_leaves=127, objective='binary', random_state=42,
               subsample=0.8)

 Best normalized gini score for 3-fold search with 5 parameter combinations:
0.8782169640198176

 Best hyperparameters:
{'subsample': 0.8, 'num_leaves': 127, 'max_depth': 3, 'colsample_bytree': 0.8}
```

In [105]:
```
1 best_model =random_search.best_estimator_
```

In [106]:
```
1 y_pred = best_model.predict(X_test_scaled)
2 accuracy_score(y_test,y_pred)
```

Out[106]:
```
0.9203354297693921
```

In [107]:
```
1 y_prob = best_model.predict_proba(X_test_scaled)
2
```

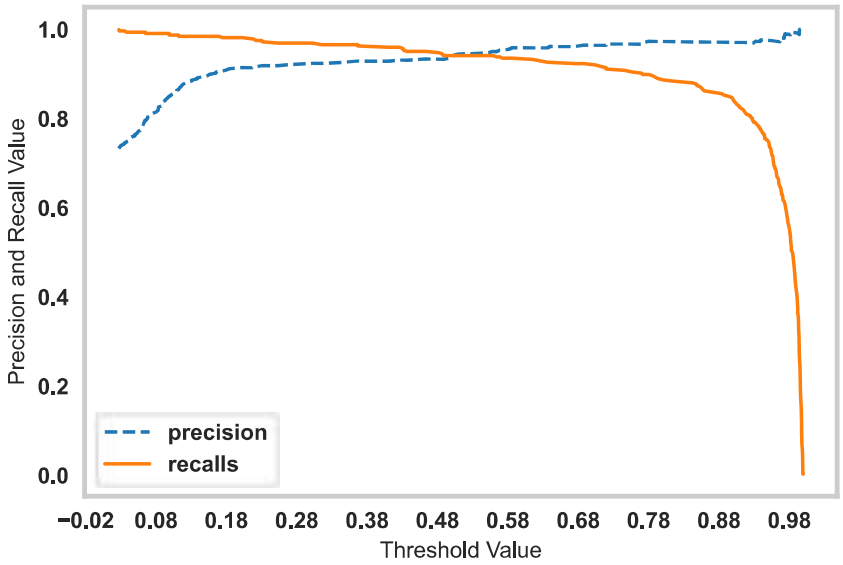
In [108]:
```
1 # Evaluate performance using accuracy score and confusion matrix
2 print(f"Accuracy score: {round(accuracy_score(y_test, y_pred), 3)}\n")
3 print(f"Confusion matrix\n {confusion_matrix(y_test, y_pred)}\n")
4 print(f"Classification report\n {classification_report(y_test, y_pred)}\n")
5 print(f"ROC-AUC Score\n {round(roc_auc_score(y_test, y_prob[:,1]), 3)}")
```

```
Accuracy score: 0.92

Confusion matrix
 [[131  19]
 [ 19 308]]

Classification report
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       150
           1       0.94      0.94      0.94       327

    accuracy                           0.92       477
   macro avg       0.91      0.91      0.91       477
weighted avg       0.92      0.92      0.92       477


ROC-AUC Score
 0.965
```

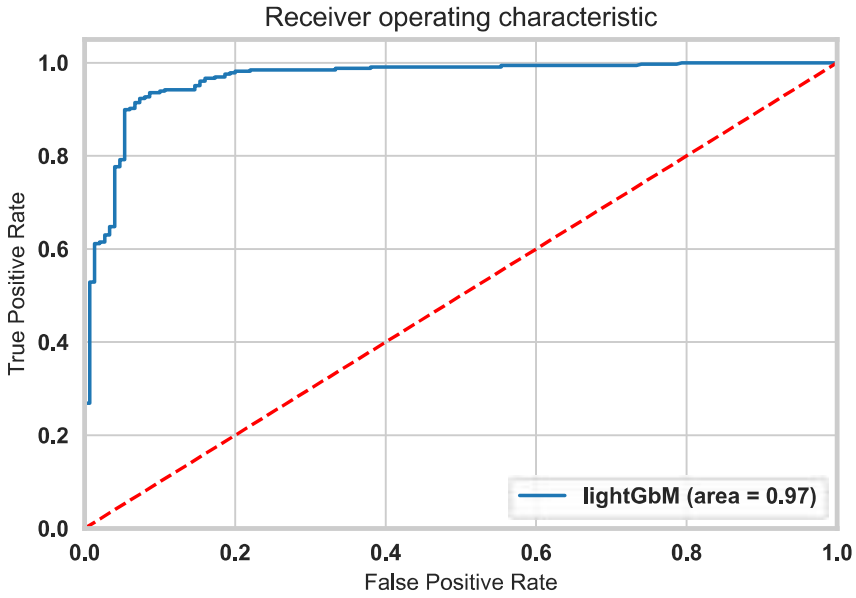
In [ ]:
```
1
```

In [109]:

```
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()

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



In [110]:
```
roc_auc = roc_auc_score(y_test, y_prob[:,1])
fpr, tpr, thresholds = roc_curve(y_test, y_prob[:,1])
plt.figure()
plt.plot(fpr, tpr, label='lightGbM (area = %0.2f)' % 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()
```



In [ ]:
```
```

So XGBooost gives the best performance out of all the three models with ROC-AUC score of 97%

In [ ]:
```
```

```
## Observation
People who had increased rating change and income change are less likely to churn

People who haveless median income are less likely to churn

```

```
people with good grade are less likely to churn

People getting good quarterly ratings are less likely to churn and vice versa

People with better designation are less likely to shurn and vice versa

We are gettting a good precision, recall and F1 score for the all the models we tried.

Best model that was selected was LightGBM although there was only marginal increaese in the accuracy and other metrics.
```

In [ ]:

```
```

Recommendations

Focus on Happy Customers: Pay extra attention to customers who are reporting increased satisfaction (higher ratings) and income growth. Engage with them, offer incentives, and build loyalty programs.

Support Lower-Income Customers: Customers with lower median incomes tend to stay longer. Create special offers or tailored services to retain their loyalty.

Reward Good Grades: Encourage and reward customers who consistently achieve good grades or ratings. This can reinforce their loyalty.

Quarterly Ratings: Keep an eye on quarterly ratings. Reach out to customers with high ratings to nurture their loyalty and address issues with lower-rated customers promptly.

Value Customers with Better Designations: Offer exclusive benefits to customers with prestigious designations to make them feel valued.

Opt for Efficient Models: Stick with LightGBM since it's efficient, even if it only marginally improved accuracy. This will enable faster predictions and quicker action to prevent churn.

Segment and Personalize: Create customer segments based on these factors and tailor retention strategies to each group.

Collect Feedback: Gather feedback from churned customers to understand why they left and make improvements based on their input.

Continuous Monitoring: Keep an eye on customer behavior and model performance. Adapt strategies as needed to minimize churn

In [ ]:

```
```