Problem Statement

Recruiting and retaining drivers is seen by industry watchers as a tough battle for Ola. Churn among drivers is high and it's very easy for drivers to stop working for the service on the fly or jump to Uber depending on the rates.

As the companies get bigger, the high churn could become a bigger problem. To find new drivers, Ola is casting a wide net, including people who don't have cars for jobs. But this acquisition is really costly. Losing drivers frequently impacts the morale of the organization and acquiring new drivers is more expensive than retaining existing ones.

You are working as a data scientist with the Analytics Department of Ola, focused on driver team attrition. You are provided with the monthly information for a segment of drivers for 2019 and 2020 and tasked to predict whether a driver will be leaving the company or not based on their attributes

In [1]:
```
#Importing libraries to be used later
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
#Reading the dataset and print first 5 rows
df=pd.read_csv('ola.csv')
df.head()
```

Out[2]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjo
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/
2	2	03/01/19	1	28.0	0.0	C23	2	57387	24/
3	3	11/01/20	2	31.0	0.0	C7	2	67016	11/(
4	4	12/01/20	2	31.0	0.0	C7	2	67016	11/(

In [3]:
```
df.shape
```

Out[3]:
```
(19104, 14)
```

In [4]:
```
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 [5]:
```
df.describe(include='all')
```

Out[5]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Educa
count	19104.000000	19104	19104.000000	19043.000000	19052.000000	19104	19
unique	NaN	24	NaN	NaN	NaN	29	
top	NaN	01/01/19	NaN	NaN	NaN	C20	
freq	NaN	1022	NaN	NaN	NaN	1008	
mean	9551.500000	NaN	1415.591133	34.668435	0.418749	NaN	
std	5514.994107	NaN	810.705321	6.257912	0.493367	NaN	
min	0.000000	NaN	1.000000	21.000000	0.000000	NaN	
25%	4775.750000	NaN	710.000000	30.000000	0.000000	NaN	
50%	9551.500000	NaN	1417.000000	34.000000	0.000000	NaN	
75%	14327.250000	NaN	2137.000000	39.000000	1.000000	NaN	
max	19103.000000	NaN	2788.000000	58.000000	1.000000	NaN	

Handling Null values

In [6]:
```
from sklearn.impute import KNNImputer
imp=KNNImputer( n_neighbors=3)
vals=df['Age'].values
df['Age']=imp.fit_transform(vals.reshape(-1,1))
```

In [7]:
```
from sklearn.impute import SimpleImputer
simp=SimpleImputer(strategy='most_frequent')
vals=df['Gender'].values
df['Gender']=simp.fit_transform(vals.reshape(-1,1))
```

In [8]:
```
#Creating the target variable from the Lastworkingdate column
df['target']=np.where(df['LastWorkingDate'].isnull() ,0,1)
```

Non Graphical Analysis

In [9]:
```
df.groupby('target')['City'].agg(pd.Series.mode)
```

Out[9]:
```
target
0    C20
1    C20
Name: City, dtype: object
```

In [10]:
```
df.groupby('target')['MMM-YY'].agg(pd.Series.mode)
```

Out[10]:
```
target
0    01/01/19
1    05/01/19
Name: MMM-YY, dtype: object
```

In [11]:
```
df.groupby('target')['Education_Level'].agg(pd.Series.mode)
```

Out[11]:
```
target
0    1
1    2
Name: Education_Level, dtype: int64
```

In [12]:
```
df.groupby('target')['Dateofjoining'].agg(pd.Series.mode)
```

Out[12]:
```
target
0    23/07/15
1    31/10/19
Name: Dateofjoining, dtype: object
```

In [13]:
```
df.groupby('target')['Gender'].agg(pd.Series.mode)
```

Out[13]:
```
target
0    0.0
1    0.0
Name: Gender, dtype: float64
```

In [14]:
```
df.groupby('target')['LastWorkingDate'].agg(pd.Series.mode)
```

Out[14]:
```
target
0          []
1    29/07/20
Name: LastWorkingDate, dtype: object
```

In [15]:
```
df.groupby('target')['Joining Designation'].agg(pd.Series.mode)
```

Out[15]:
```
target
0    1
1    1
Name: Joining Designation, dtype: int64
```

In [16]:
```
df.groupby('target')['Grade'].agg(pd.Series.mode)
```

Out[16]:
```
target
0    2
1    2
Name: Grade, dtype: int64
```

In [17]:
```
df.groupby('target')['Age'].median()
```

Out[17]:
```
target
0    34.0
1    33.0
Name: Age, dtype: float64
```

In [18]:
```
df.groupby('target')['Income'].median()
```

Out[18]:
```
target
0    61291.0
1    51630.0
Name: Income, dtype: float64
```

In [19]:
```
df.groupby('target')['Total Business Value'].median()
```

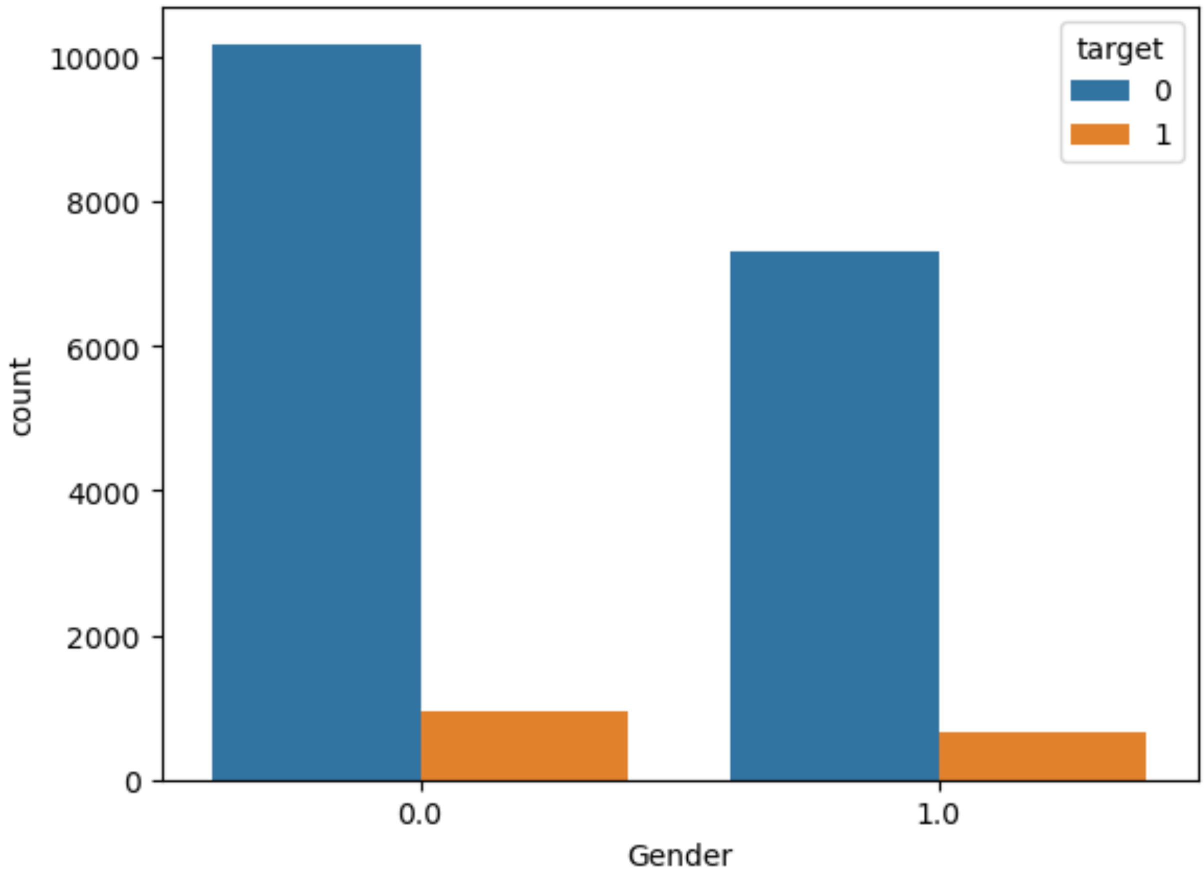
Out[19]:
```
target
0    300520.0
1         0.0
Name: Total Business Value, dtype: float64
```

Graphical Analysis

In [20]:
```
sns.countplot(df['Gender'],hue=df['target'])
```

```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

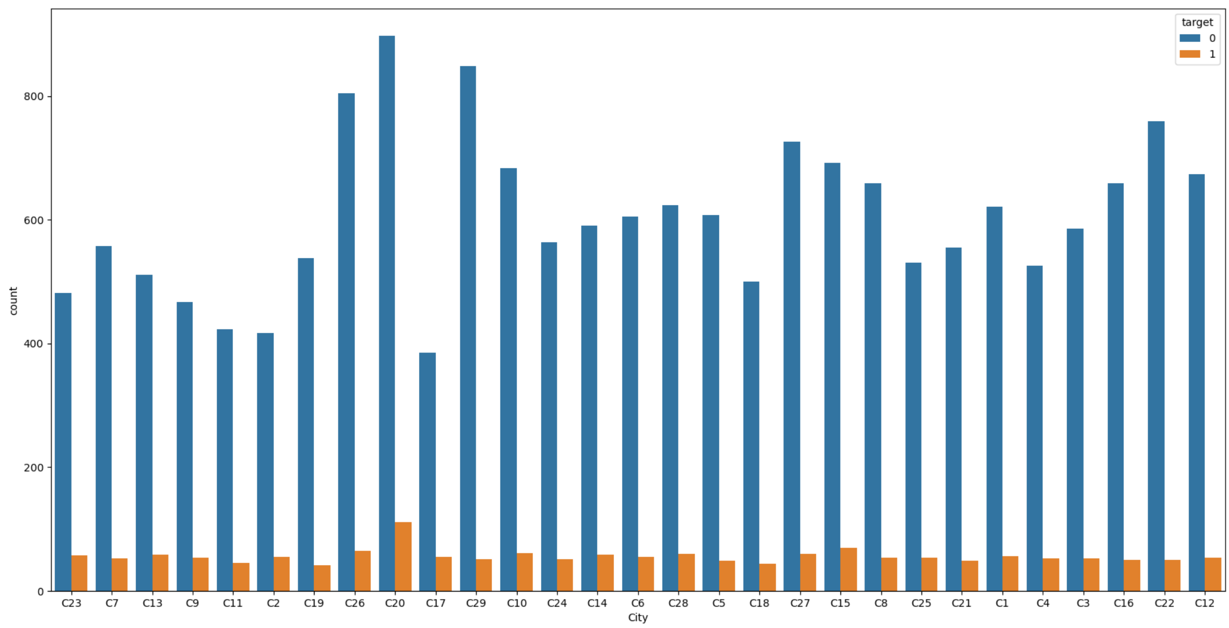
Out[20]:
```
<AxesSubplot:xlabel='Gender', ylabel='count'>
```



In [21]:
```
plt.figure(figsize=(20,10))
sns.countplot(df['City'],hue=df['target'])
```

```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

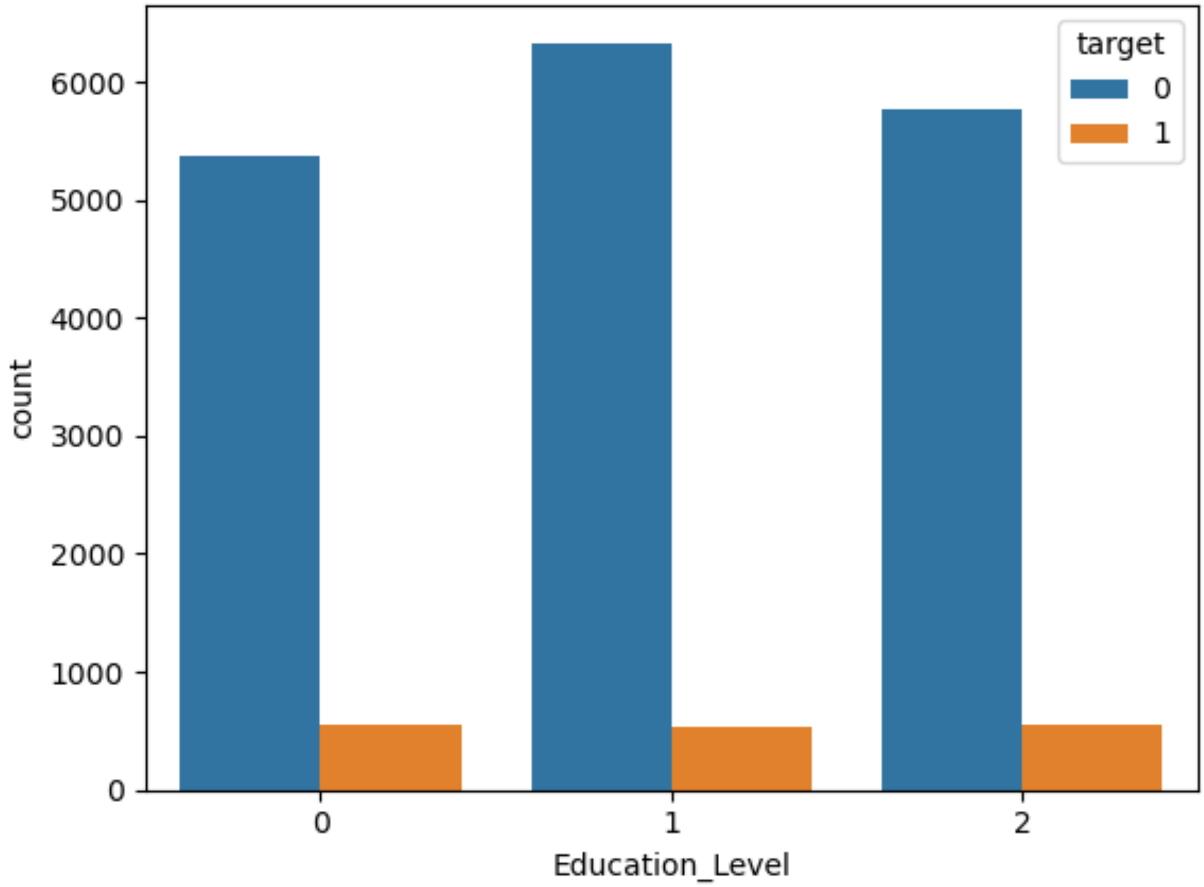
Out[21]:
```
<AxesSubplot:xlabel='City', ylabel='count'>
```



In [22]:
```
sns.countplot(df['Education_Level'],hue=df['target'])
```

```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

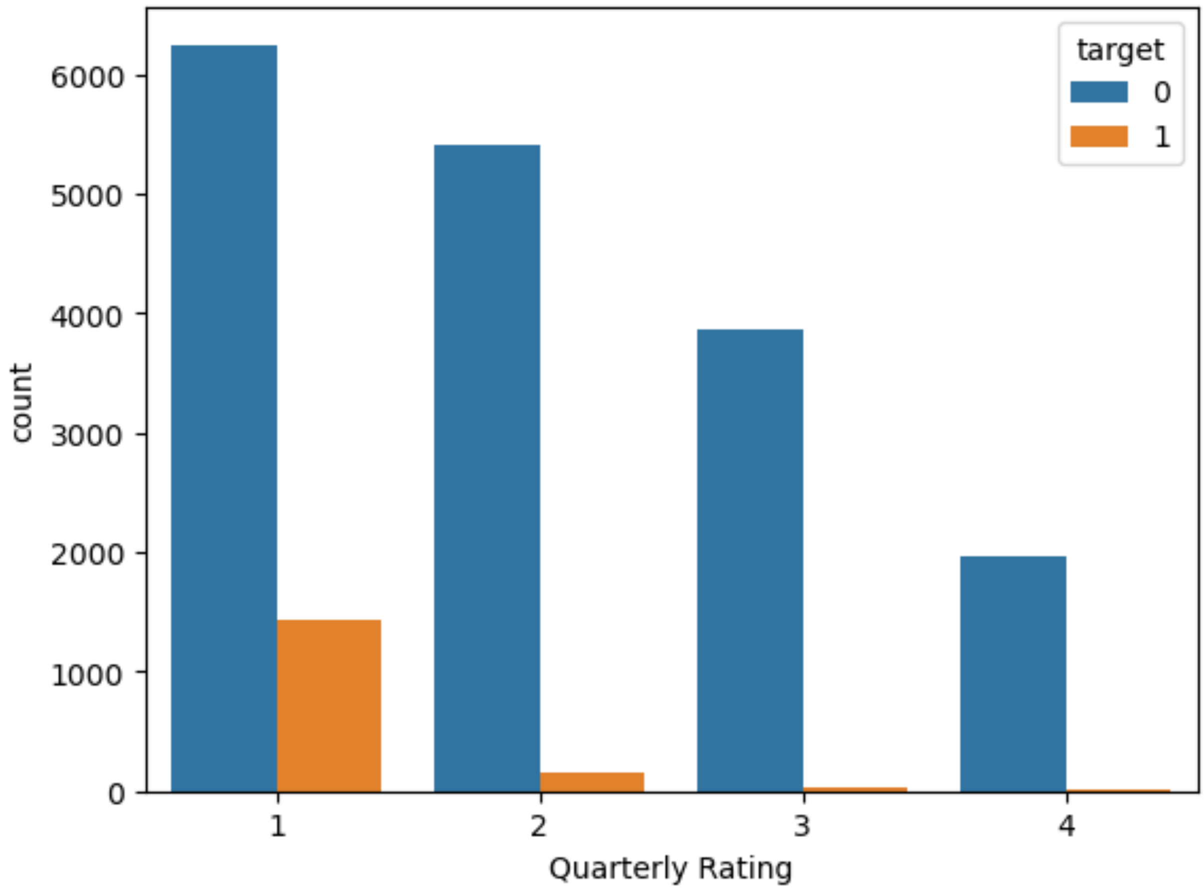
Out[22]: <AxesSubplot:xlabel='Education_Level', ylabel='count'>



In [23]:
```
sns.countplot(df['Quarterly Rating'],hue=df['target'])
```

```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

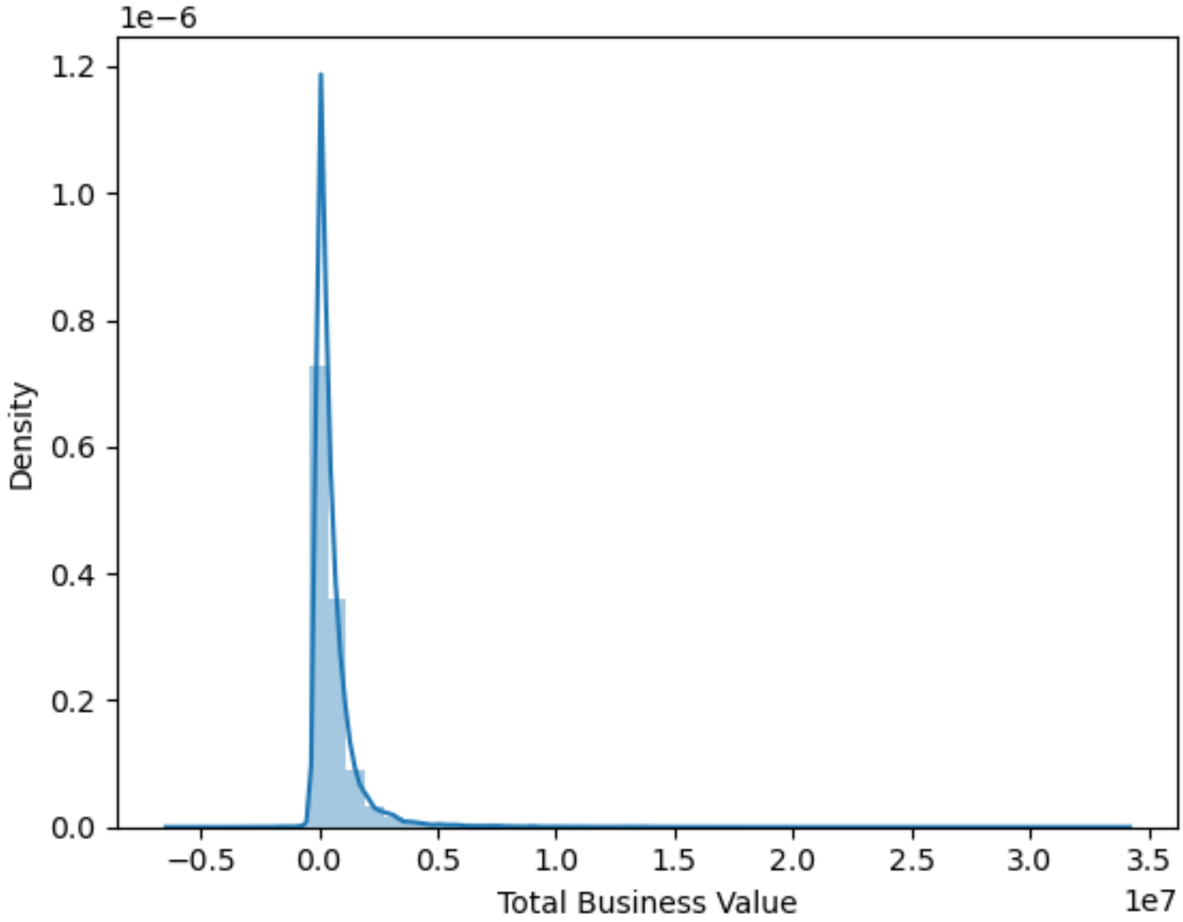
Out[23]: <AxesSubplot:xlabel='Quarterly Rating', ylabel='count'>



In [24]:
```
sns.distplot(df['Total Business Value'])
```

```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level f
unction for histograms).
  warnings.warn(msg, FutureWarning)
```

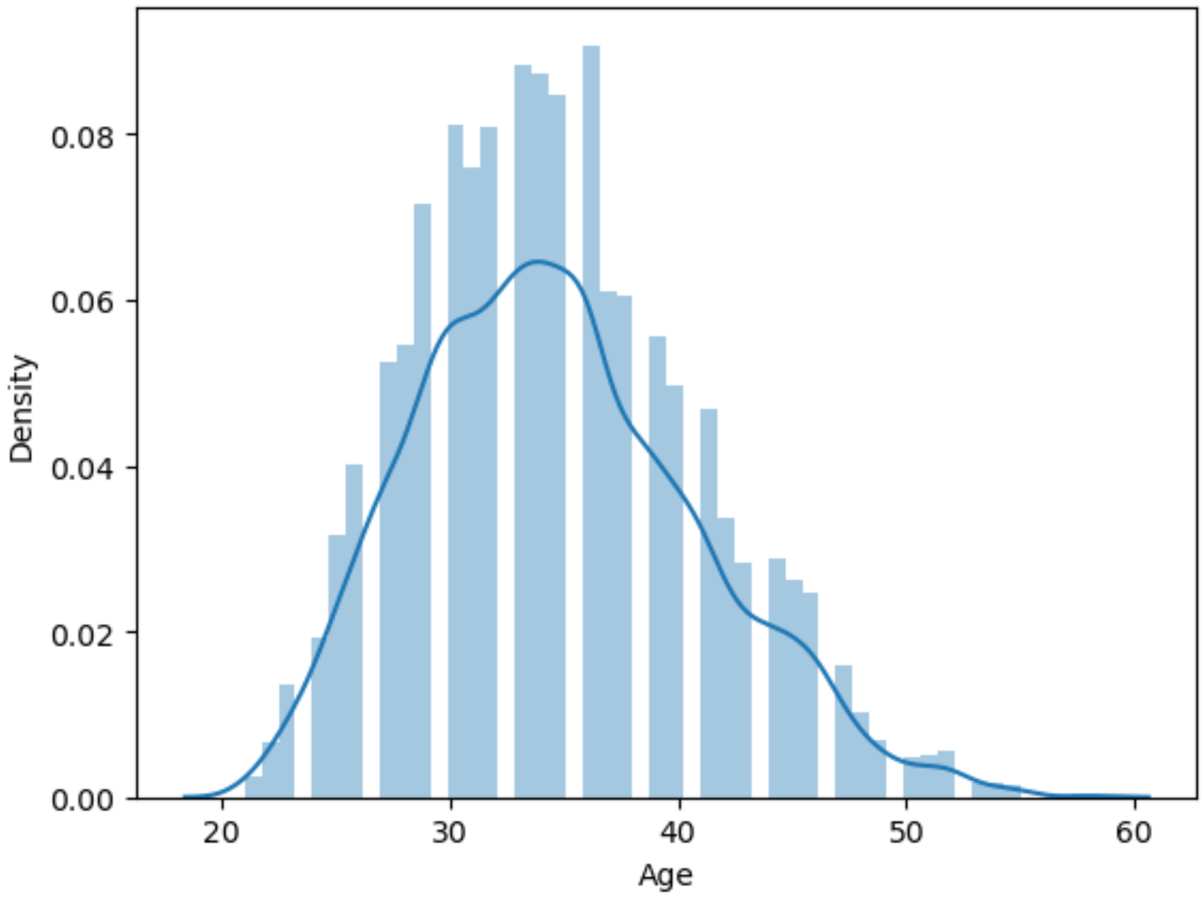
Out[24]:
```
<AxesSubplot:xlabel='Total Business Value', ylabel='Density'>
```



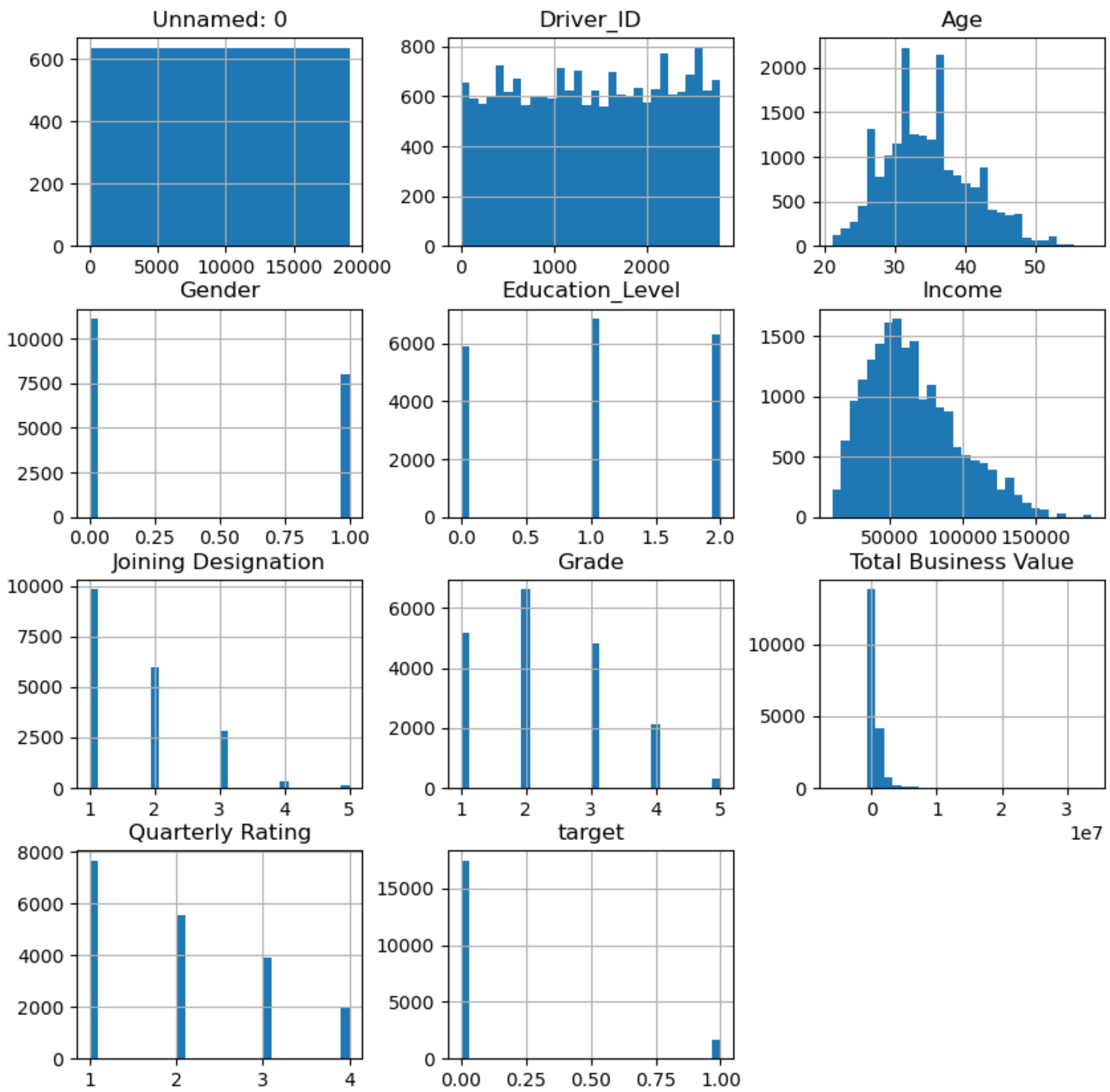
In [25]:
```
sns.distplot(df['Age'])
```

```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[25]:
```
<AxesSubplot:xlabel='Age', ylabel='Density'>
```

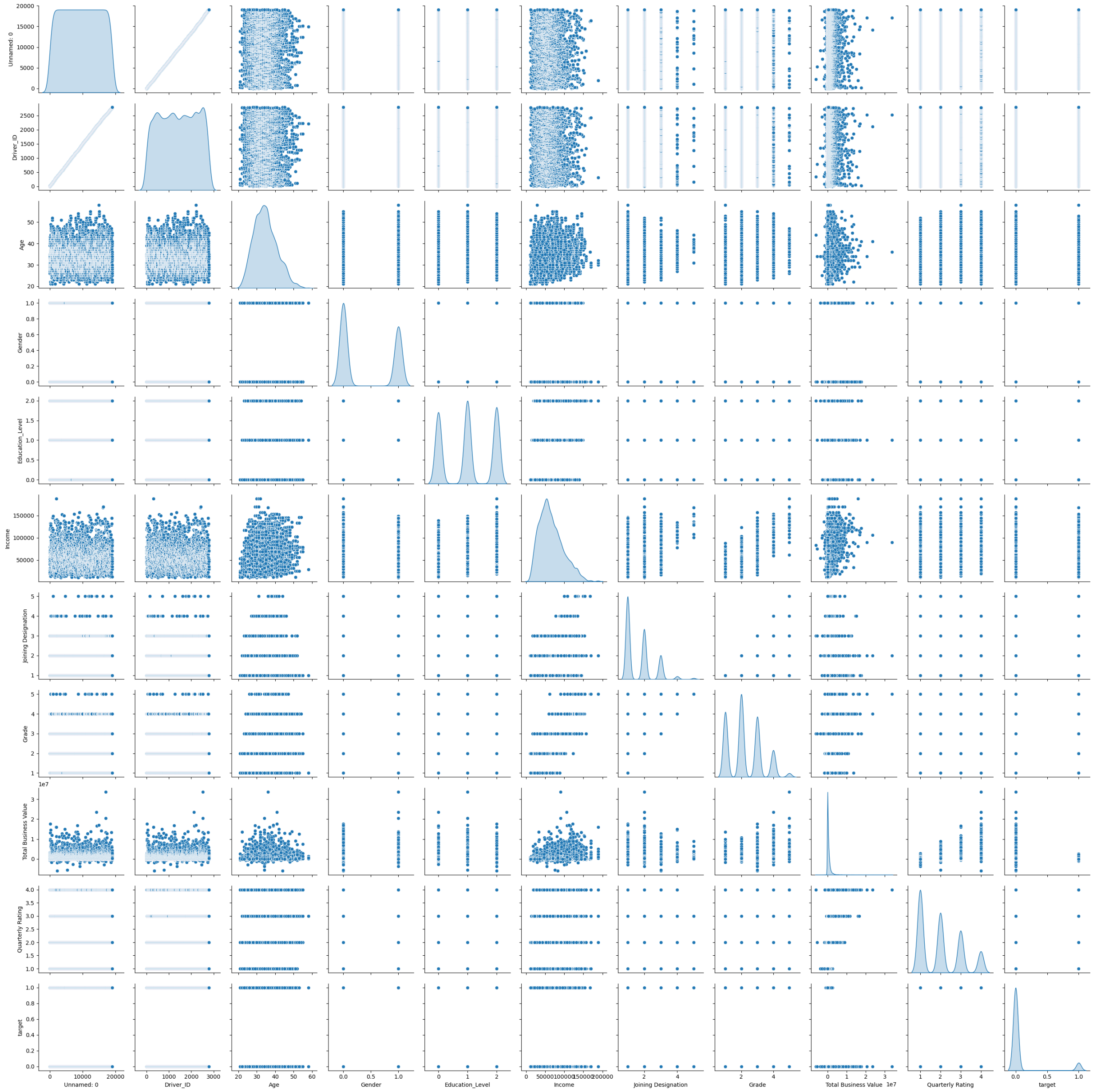


In [26]:
```
df.hist(figsize=(10,10),bins=30);
```



In [27]:
```python
sns.pairplot(df, diag_kind='kde')
```

Out[27]:
```
<seaborn.axisgrid.PairGrid at 0x7fd9e9592610>
```



Feature Engineering

In [28]:
```
#Dropping the unnamed feature
df.drop('Unnamed: 0',axis=1, inplace=True)
```

In [29]:
```
#Converting the reporting date to datetime format
df['MMM-YY']=pd.to_datetime(df['MMM-YY'])
```

In [30]:
```
#Converting the Dateofjoining to datetime format
df['Dateofjoining']=pd.to_datetime(df['Dateofjoining'])
```

In [31]:
```
#Converting the LastWorkingDate feature to datetime format
df['LastWorkingDate']=pd.to_datetime(df['LastWorkingDate'])
```

In [32]:
```
n=df['LastWorkingDate'].max()
```

In [33]:
```
#Filling the null values of Lastworkingdate with today
df['LastWorkingDate'].fillna(n, inplace=True)
```

In [34]:
```
#Derived a feature called tenure which depicts the number of day the driv
temp=df['LastWorkingDate']-df['Dateofjoining']
df['tenure']=temp.dt.days
```

In [35]:
```
#Derived a feature called tenure_y which contains the number of year(s)th
#temp1=df['LastWorkingDate'].dt.year-df['Dateofjoining'].dt.year
#df['tenure_y']=temp1
```

In [36]:
```
#Derving the year, month and date of joining from the dateofjoining featu
df['yoj']=df['Dateofjoining'].dt.year
df['moj']=df['Dateofjoining'].dt.month
df['doj']=df['Dateofjoining'].dt.day
```

In [37]:
```
df['yoj']=df['yoj'].map(str)
df['moj']=df['moj'].map(str)
df['doj']=df['doj'].map(str)
```

In [38]:
```
#Derving the year, month and date of leaving from the LastWorkingDate fea
#df['ly']=df['LastWorkingDate'].dt.year
#df['lm']=df['LastWorkingDate'].dt.month
#df['ld']=df['LastWorkingDate'].dt.day
```

In [39]:
```
#df['ly']=df['ly'].map(str)
#df['lm']=df['lm'].map(str)
#df['ld']=df['ld'].map(str)
```

In [40]:
```
#Derving the year, month and date of leaving from the LastWorkingDate fea
df['ry']=df['MMM-YY'].dt.year
df['rm']=df['MMM-YY'].dt.month
df['rd']=df['MMM-YY'].dt.day
```

In [41]:
```
df['ry']=df['ry'].map(str)
df['rm']=df['rm'].map(str)
df['rd']=df['rd'].map(str)
```

In [42]:
```
df['MMM-YY']=df['MMM-YY'].map(str)
df['Dateofjoining']=df['Dateofjoining'].map(str)
df['LastWorkingDate']=df['LastWorkingDate'].map(str)
df.drop(['MMM-YY','Dateofjoining','LastWorkingDate'],axis=1,inplace=True)
```

Create a column which tells whether the quarterly rating has increased for that driver - for those whose quarterly rating has increased we assign the value 1

In [43]:
```
qr=df.groupby(['Driver_ID'])['Quarterly Rating'].agg(list)
```

In [44]:
```
df[df['Driver_ID']==2784]
```

Out[44]:

	Driver_ID	Age	Gender	City	Education_Level	Income	Joining Designation	Grade	Bus
19055	2784	33.0	0.0	C24	0	82815	2	3	122
19056	2784	33.0	0.0	C24	0	82815	2	3	20
19057	2784	33.0	0.0	C24	0	82815	2	3	449
19058	2784	33.0	0.0	C24	0	82815	2	3	10
19059	2784	33.0	0.0	C24	0	82815	2	3	2
19060	2784	33.0	0.0	C24	0	82815	2	3	
19061	2784	33.0	0.0	C24	0	82815	2	3	10
19062	2784	33.0	0.0	C24	0	82815	2	3	2
19063	2784	33.0	0.0	C24	0	82815	2	3	2
19064	2784	33.0	0.0	C24	0	82815	2	3	99
19065	2784	33.0	0.0	C24	0	82815	2	3	5
19066	2784	33.0	0.0	C24	0	82815	2	3	1
19067	2784	34.0	0.0	C24	0	82815	2	3	130
19068	2784	34.0	0.0	C24	0	82815	2	3	85
19069	2784	34.0	0.0	C24	0	82815	2	3	412
19070	2784	34.0	0.0	C24	0	82815	2	3	15
19071	2784	34.0	0.0	C24	0	82815	2	3	15
19072	2784	34.0	0.0	C24	0	82815	2	3	9
19073	2784	34.0	0.0	C24	0	82815	2	3	25
19074	2784	34.0	0.0	C24	0	82815	2	3	126
19075	2784	34.0	0.0	C24	0	82815	2	3	40
19076	2784	34.0	0.0	C24	0	82815	2	3	308
19077	2784	34.0	0.0	C24	0	82815	2	3	
19078	2784	34.0	0.0	C24	0	82815	2	3	50

In [45]:

```
inc_qr=[]
for x in qr:
    if x[0]>=x[-1]:
        inc_qr.append(0)
    else:
        inc_qr.append(1)
```

Create a column which tells whether the monthly income has increased for that driver - for those whose monthly income has increased we assign the value 1

In [46]:
```
income=df.groupby(['Driver_ID'])['Income'].agg(list)
```

In [47]:
```
inc_inc=[]
for y in income:
    if y[0]>=y[-1]:
        inc_inc.append(0)
    else:
        inc_inc.append(1)
```

Aggregate data in order to remove multiple occurrences of same driver data (We did something similar in Delhivery business Case)

In [48]:
```
sel={
    'Age':'max',
    'Gender':'first',
    'City':'first',
    'Education_Level':'last',
    'Income':'last',
    'Joining Designation':'last',
    'tenure':'last',
    #'tenure_y':'last',
    'Grade':'last',
    'target':'last',
    #'MMM-YY':'last',
    #'Dateofjoining':'first',
    #'LastWorkingDate':'last',
    'Total Business Value':'sum',
    'Quarterly Rating':'last',
     'doj':'first',
     'moj':'first',
     'yoj': 'first',
     'ry':'first',
     'rm':'first',
     'rd': 'first'
    #'ly':'last',
     #'lm':'last',
     #'ld': 'last'
}
```

In [49]:
```
df=pd.DataFrame(df.groupby(['Driver_ID']).agg(sel))
```

In [50]:
```
df['Increased_income']=inc_inc
```

In [51]:
```
df['Increased_qr']=inc_qr
```

In [52]:
```
#Applying target encoding on the categorical feature 'City'
import category_encoders as ce
tenc=ce.TargetEncoder()
df['City']=tenc.fit_transform(df['City'],df['target'])
```

```
/opt/anaconda3/lib/python3.9/site-packages/category_encoders/target_encod
er.py:122: FutureWarning: Default parameter min_samples_leaf will change
in version 2.6.See https://github.com/scikit-learn-contrib/category_encod
ers/issues/327
  warnings.warn("Default parameter min_samples_leaf will change in versio
n 2.6."
/opt/anaconda3/lib/python3.9/site-packages/category_encoders/target_encod
er.py:127: FutureWarning: Default parameter smoothing will change in vers
ion 2.6.See https://github.com/scikit-learn-contrib/category_encoders/iss
ues/327
  warnings.warn("Default parameter smoothing will change in version 2.6."
```

In [53]:
```
l=['doj', 'moj', 'yoj', 'ry','rm', 'rd'] #'ly', 'lm', 'ld']#'MMM-YY','Dat
```

In [54]:
```
for i in l:
    df[i]=tenc.fit_transform(df[i],df['target'])
```

In [55]:
```
pd.options.display.max_columns = None
df.head()
```

Out[55]:

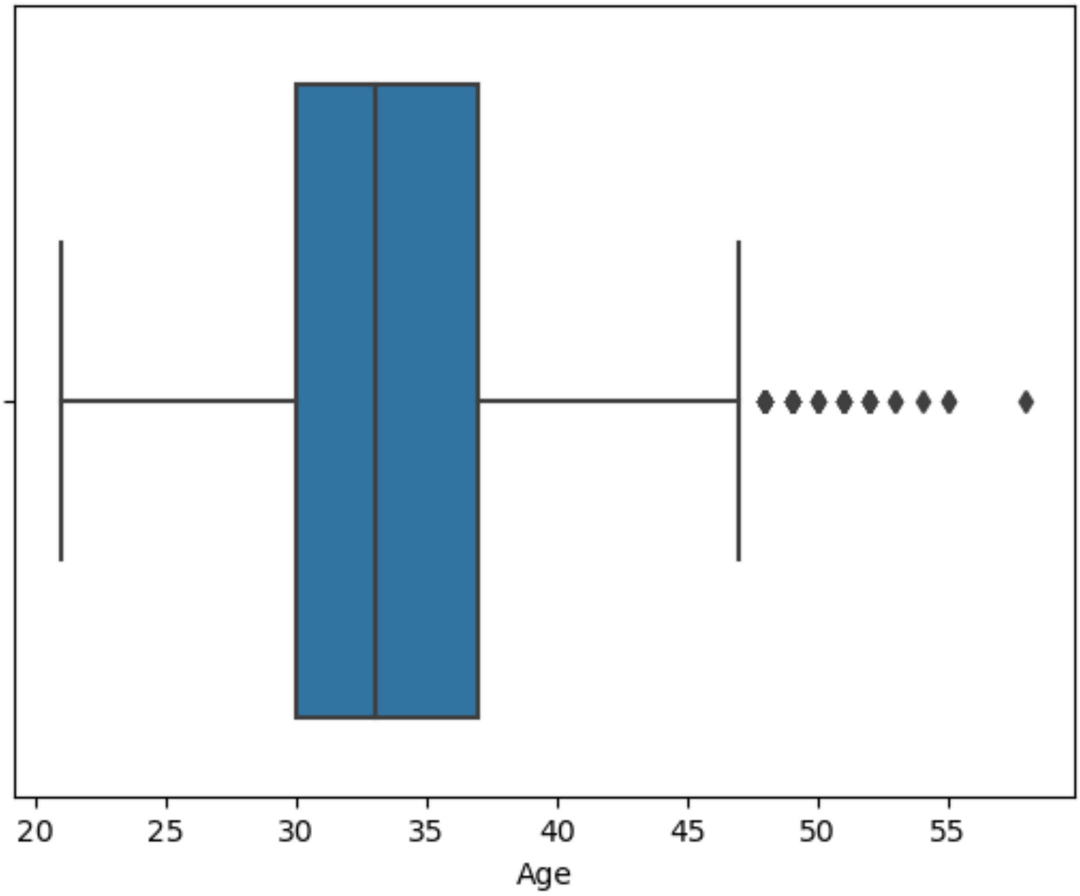
	Age	Gender	City	Education_Level	Income	Joining Designation	tenure	Grade
Driver_ID								
1	28.0	0.0	0.770270	2	57387	1	77	1
2	31.0	0.0	0.684211	2	67016	2	52	2
4	43.0	0.0	0.816901	2	65603	2	142	2
5	29.0	0.0	0.706667	0	46368	1	57	1
6	31.0	1.0	0.703125	1	78728	3	150	3

Outlier detection and removal

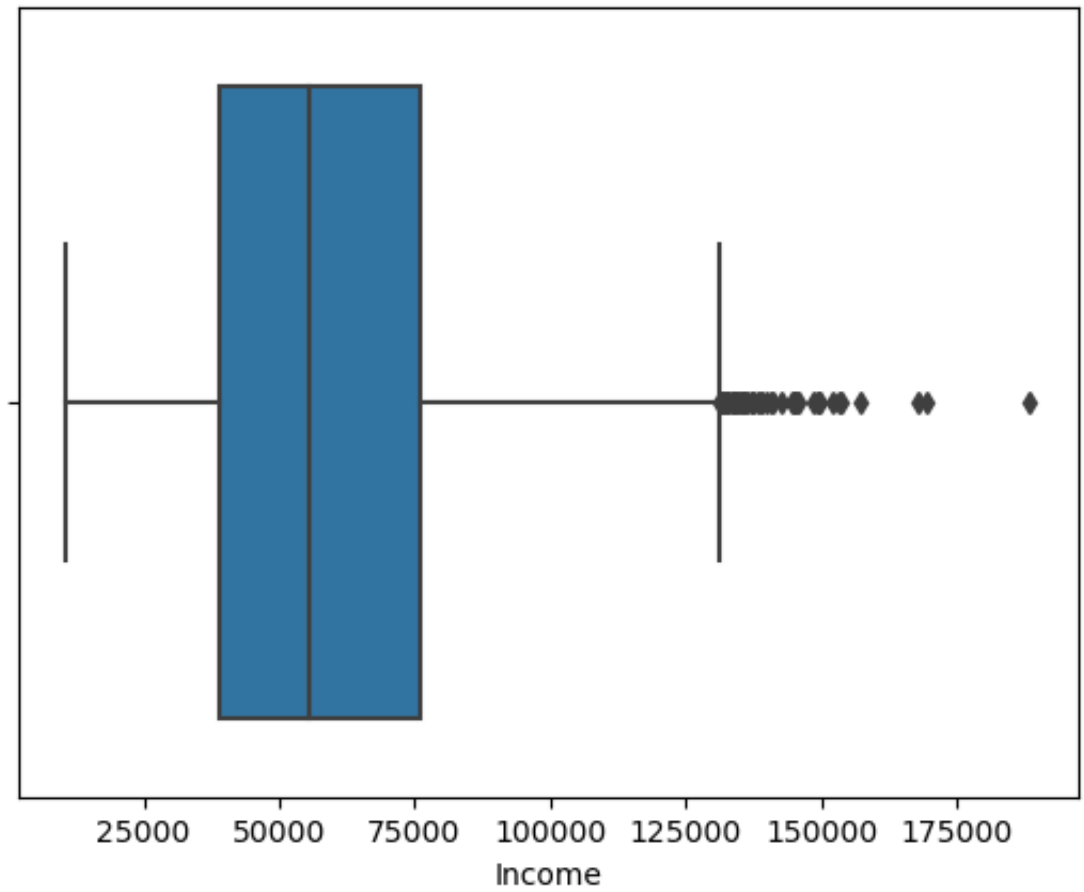
In [56]:
```
a=['Age', 'Income',
       'Joining Designation',
       'Total Business Value', 'Quarterly Rating']
```

In [57]:
```
for i in a:
    sns.boxplot(df[i])
    plt.show()
```

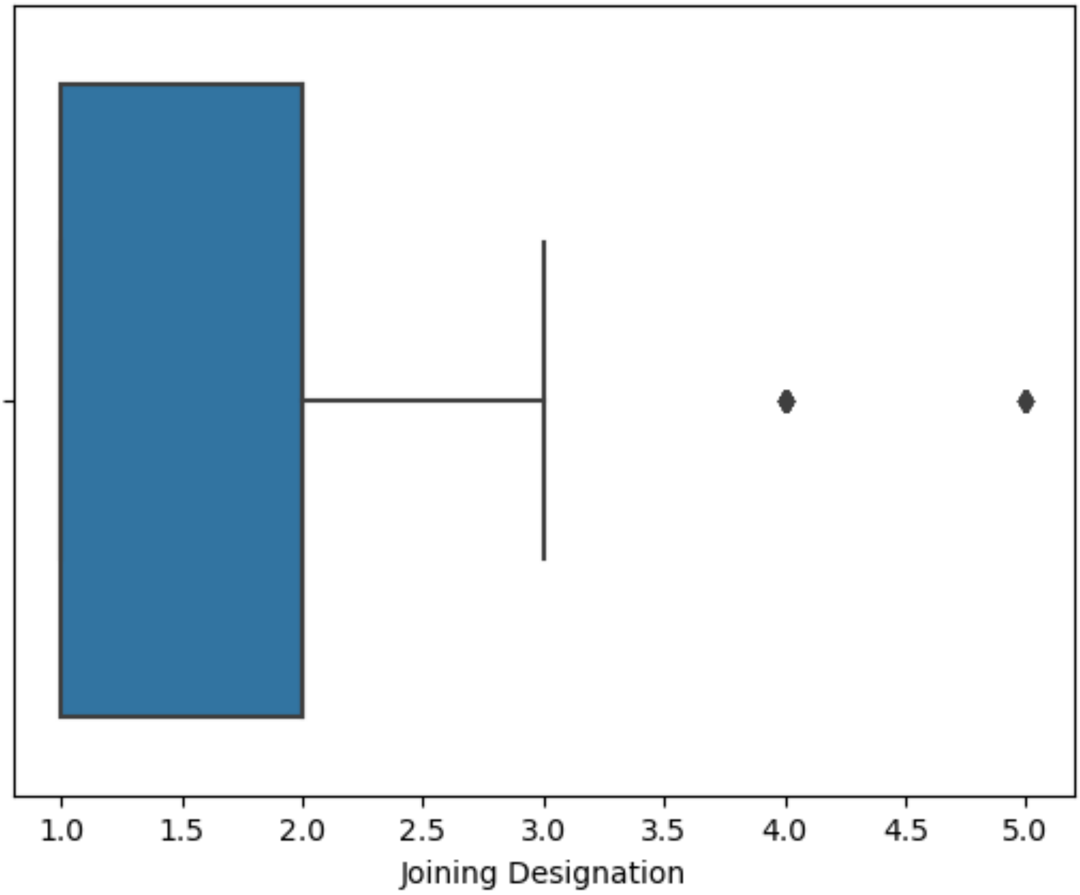
```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing othe
r arguments without an explicit keyword will result in an error or misint
erpretation.
  warnings.warn(
```



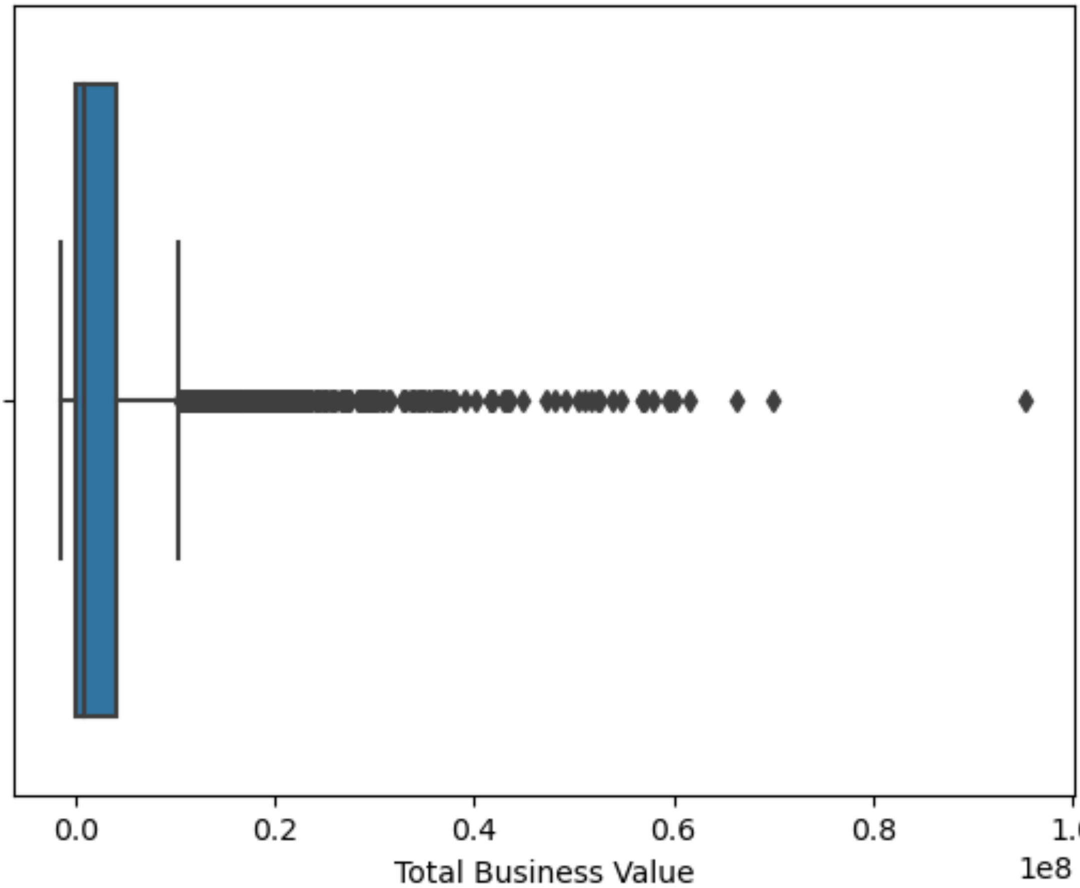

```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing othe
r arguments without an explicit keyword will result in an error or misint
erpretation.
  warnings.warn(
```



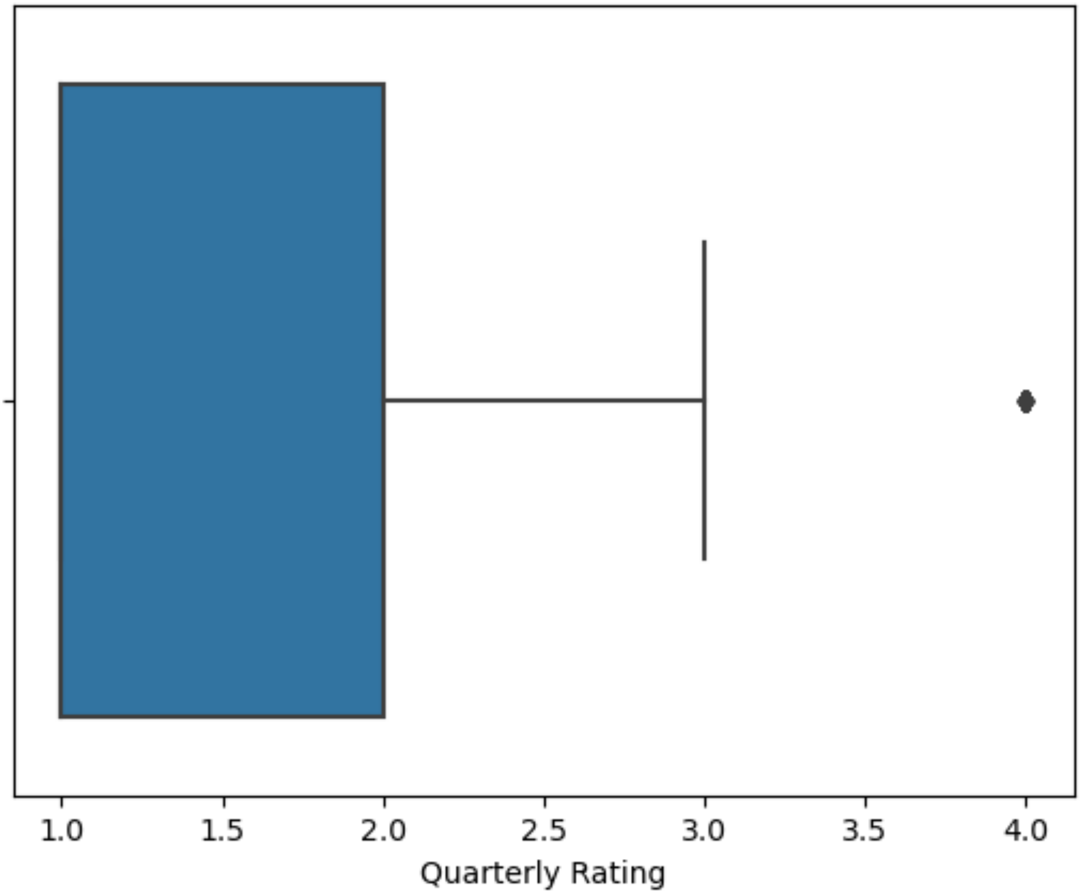

```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing othe
r arguments without an explicit keyword will result in an error or misint
erpretation.
  warnings.warn(
```




```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing othe
r arguments without an explicit keyword will result in an error or misint
erpretation.
  warnings.warn(
```




```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing othe
r arguments without an explicit keyword will result in an error or misint
erpretation.
  warnings.warn(
```



In [58]:
```
df.shape
```

Out[58]:
```
(2381, 19)
```

In [59]:
```
def outliers(data,feature):
  q1=data[feature].quantile(0.05)
  q3=data[feature].quantile(0.95)
  iqr=q3-q1
  ul=q3+1.5*iqr
  ll=q1-1.5*iqr
  return ul,ll
```

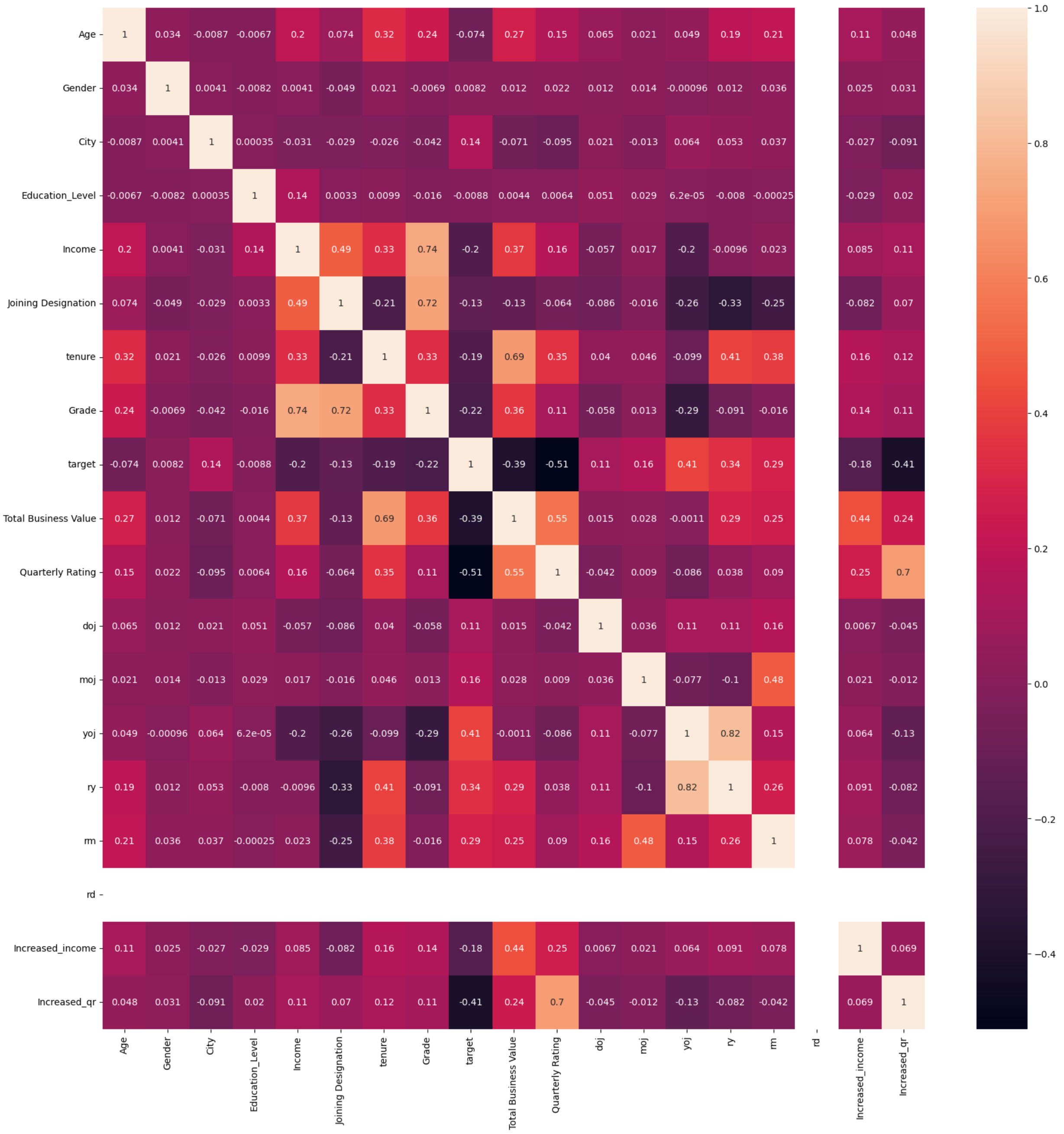
In [60]:
```
for i in a:
  ul,ll=outliers(df,i)
  df=df[(df[i]<ul) & (df[i]>ll)]
```

In [61]:
```
df.shape
```

Out[61]:
```
(2374, 19)
```

In [62]:
```
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(),annot=True)
```

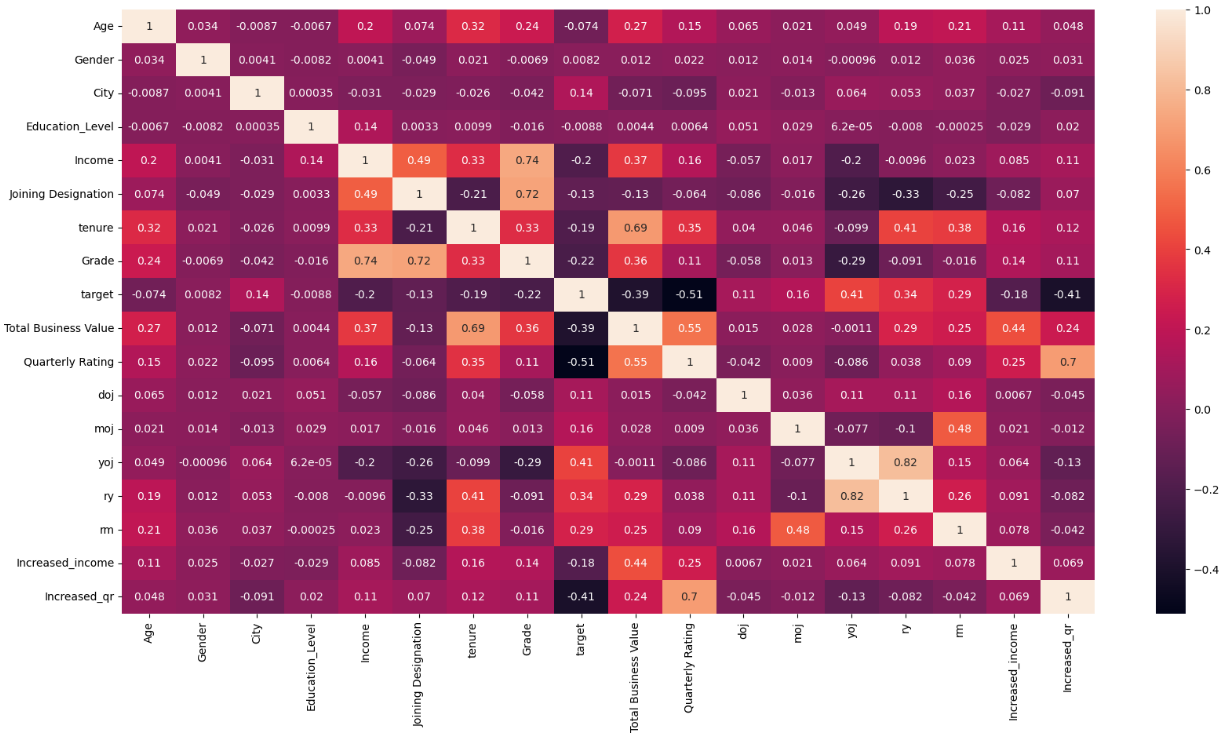
Out[62]:
```
<AxesSubplot:>
```



In [63]:
```
#Dropping the rd feature as aa the values of rd are same
df.drop(['rd'],axis=1,inplace=True)
```

In [64]:
```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(),annot=True)
```

Out[64]: <AxesSubplot:>



In [65]:
```
#Splitting features and target
X=df.drop('target',axis=1)
y=df['target']
```

Handling imbalanced Data using SMOTE

In [66]:
```
y.value_counts()
```

Out[66]:
```
1    1615
0     759
Name: target, dtype: int64
```

In [67]:
```
from imblearn.over_sampling import SMOTE
sm=SMOTE()
X_sm,y_sm=sm.fit_resample(X,y)
```

Standardizing the data using StandardScaler

In [68]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_sm=sc.fit_transform(X_sm)
```

Splitting the data into training and testing sets

In [69]:
```
from sklearn.model_selection import train_test_split as tts
X_train,X_test,y_train,y_test=tts(X_sm,y_sm,test_size=0.3)
```

Model Creation

In [70]:
```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay,class
```

Creating the Bagging Model using Random Forest

In [71]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
y_hat=rf.predict(X_test)
recall_score(y_test,y_hat)
```

Out[71]:
```
0.9191489361702128
```

As the Random Forest model used above is just a random model with no hyperparameter settings, hyperparameter tuning is done on the Random Forest Model below

Hyperparameter Tuning of Random Forest

In [72]:
```
params = {
            'n_estimators' : [100,200,300,400],
            'max_depth' : [10,15,20],
            'criterion' : ['gini', 'entropy'],
            'bootstrap' : [True, False],
            'max_features' : [8,13,17]
         }
```

In [73]:
```
from sklearn.model_selection import GridSearchCV

tuning_function = GridSearchCV(estimator = RandomForestClassifier(),
                               param_grid = params,
                               scoring = 'recall',
                               cv = 5,
                               n_jobs=-1
                               )
```

In [74]:
```
tuning_function.fit(X_train, y_train)

parameters = tuning_function.best_params_
score = tuning_function.best_score_
print(parameters)
print(score)
```

```
{'bootstrap': False, 'criterion': 'gini', 'max_depth': 20, 'max_features'
: 8, 'n_estimators': 100}
0.9371179039301311
```

In [75]:
```
yh=tuning_function.predict(X_test)
recall_score(y_test,yh)
```

Out[75]:
```
0.9425531914893617
```

Final Random Forest Model with best hyperparameter settings

In [76]:
```
rf=RandomForestClassifier(bootstrap=False,criterion='entropy',max_depth=1
```

In [77]:
```
rf.fit(X_train,y_train)
yh=rf.predict(X_test)
recall_score(y_test,yh)
```

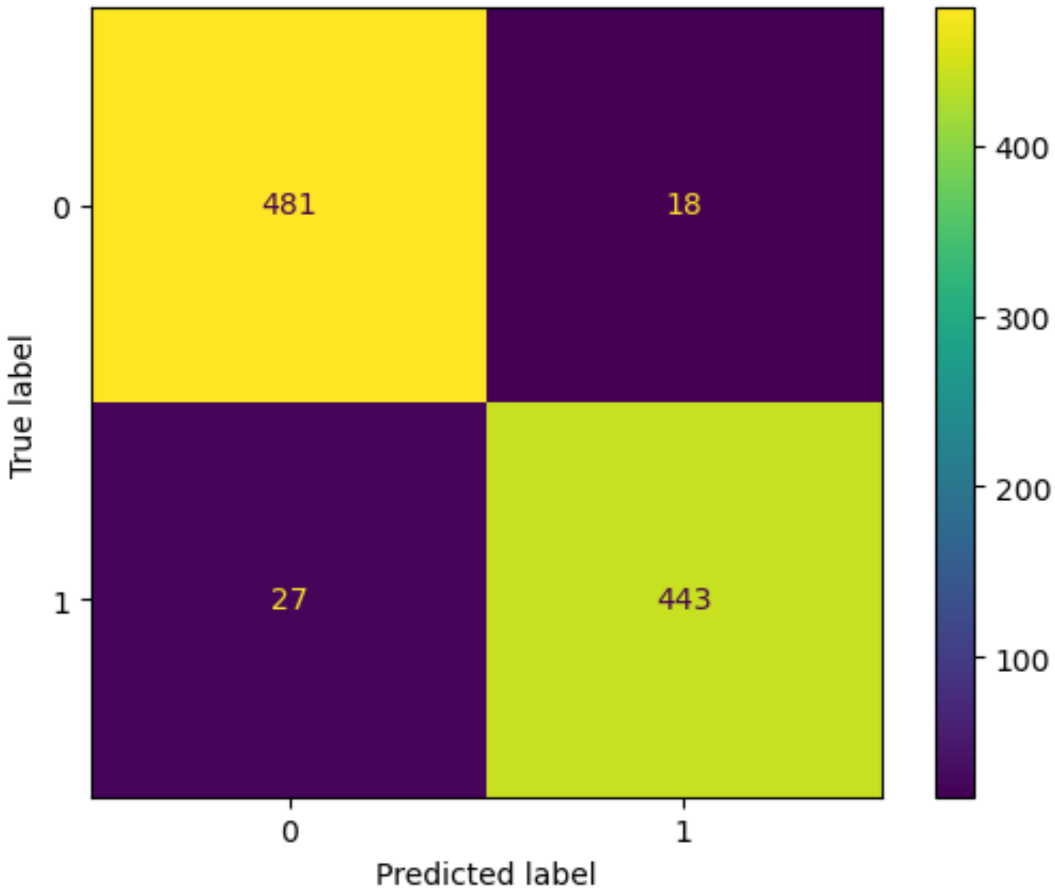
Out[77]:
```
0.9425531914893617
```

In [78]:
```
print(classification_report(y_test,yh))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       499
           1       0.96      0.94      0.95       470

    accuracy                           0.95       969
   macro avg       0.95      0.95      0.95       969
weighted avg       0.95      0.95      0.95       969
```

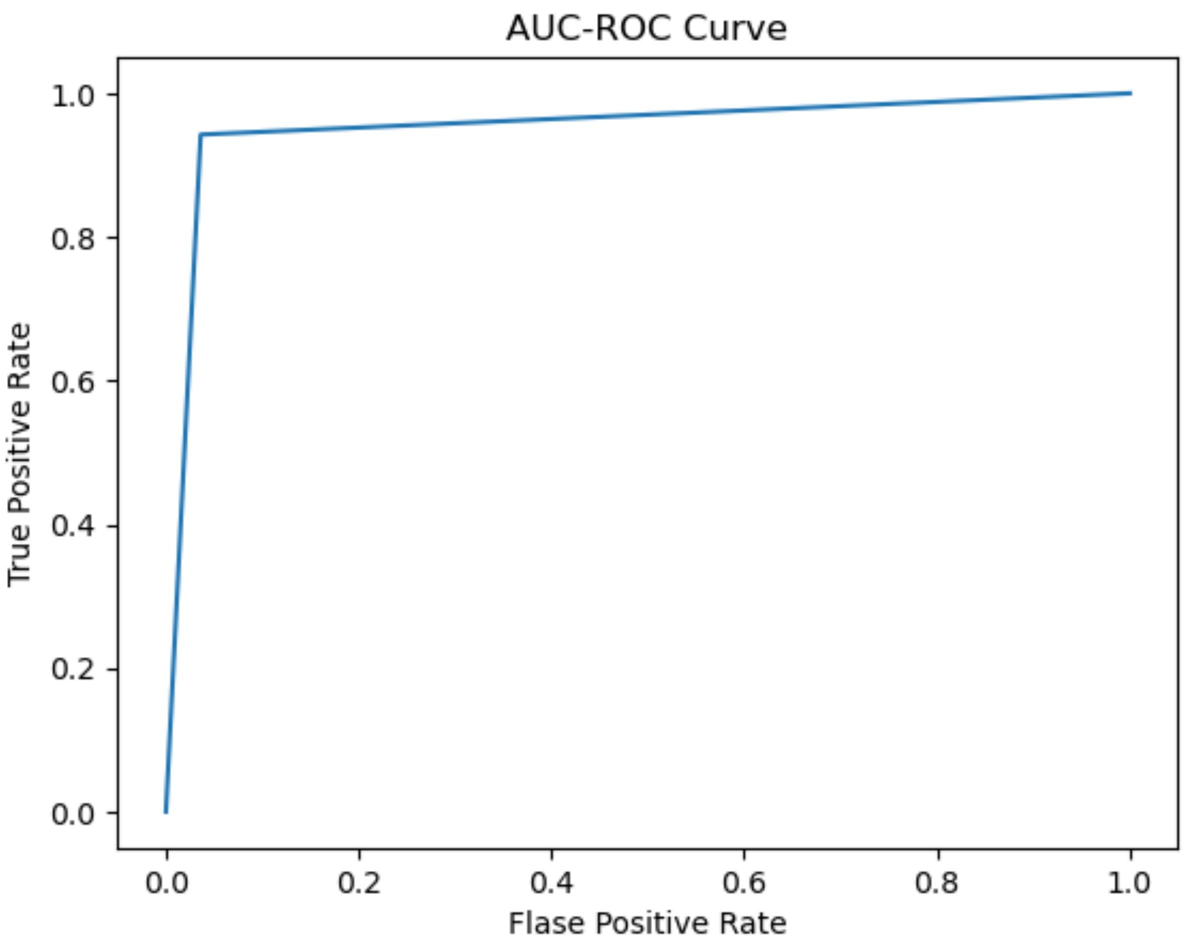
In [79]:
```
cm=confusion_matrix(y_test,yh)
ConfusionMatrixDisplay(cm).plot()
```

Out[79]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fd9d
015bac0>
```



In [80]:
```
fpr, tpr, thresholds = roc_curve(y_test, yh)
plt.plot(fpr,tpr)
plt.xlabel('Flase Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('AUC-ROC Curve')
```

Out[80]:
```
Text(0.5, 1.0, 'AUC-ROC Curve')
```



Creating the Boosting Model using LIGHTGBM

In [81]:
```
from lightgbm import LGBMClassifier
lgbm=LGBMClassifier()
lgbm.fit(X_train,y_train)
y_hat1=lgbm.predict(X_test)
recall_score(y_test,y_hat)
```

Out[81]: 0.9191489361702128

Hyperparameter tuning of the LGBM Classifier

In [82]:
```
hp={
    'boosting_type':['gbdt','goss'],
    'max_depth':[10,50,100],
    'learning_rate':[0.1,0.5,0.8],
    'n_estimators':[400,500,600],
    'subsample':[0.3,0.5,0.8],
    'colsample_bytree':[0.3,0.5,0.8]
}
```

In [83]:
```
tuning_function = GridSearchCV(estimator = LGBMClassifier(),
                               param_grid = hp,
                               scoring = 'recall',
                               cv = 5,
                               n_jobs=-1
                               )
```

In [84]:
```
tuning_function.fit(X_train, y_train)

parameters = tuning_function.best_params_
score = tuning_function.best_score_
print(parameters)
print(score)
```

```
{'boosting_type': 'gbdt', 'colsample_bytree': 0.8, 'learning_rate': 0.5,
'max_depth': 50, 'n_estimators': 500, 'subsample': 0.3}
0.9502183406113538
```

In [85]:
```
yh=tuning_function.predict(X_test)
recall_score(y_test,yh)
```

Out[85]:
```
0.9468085106382979
```

FInal model after hyperparameter tuning

In [86]:
```
lgbm=LGBMClassifier(boosting_type='gbdt', colsample_bytree=0.3, max_depth
                    n_estimators=500, subsample=0.3,learning_rate=0.8)
lgbm.fit(X_train,y_train)
pred=lgbm.predict(X_test)
recall_score(y_test,pred)
```

Out[86]:
```
0.951063829787234
```

Classification Report

In [87]:
```
print(classification_report(y_test,pred))
```

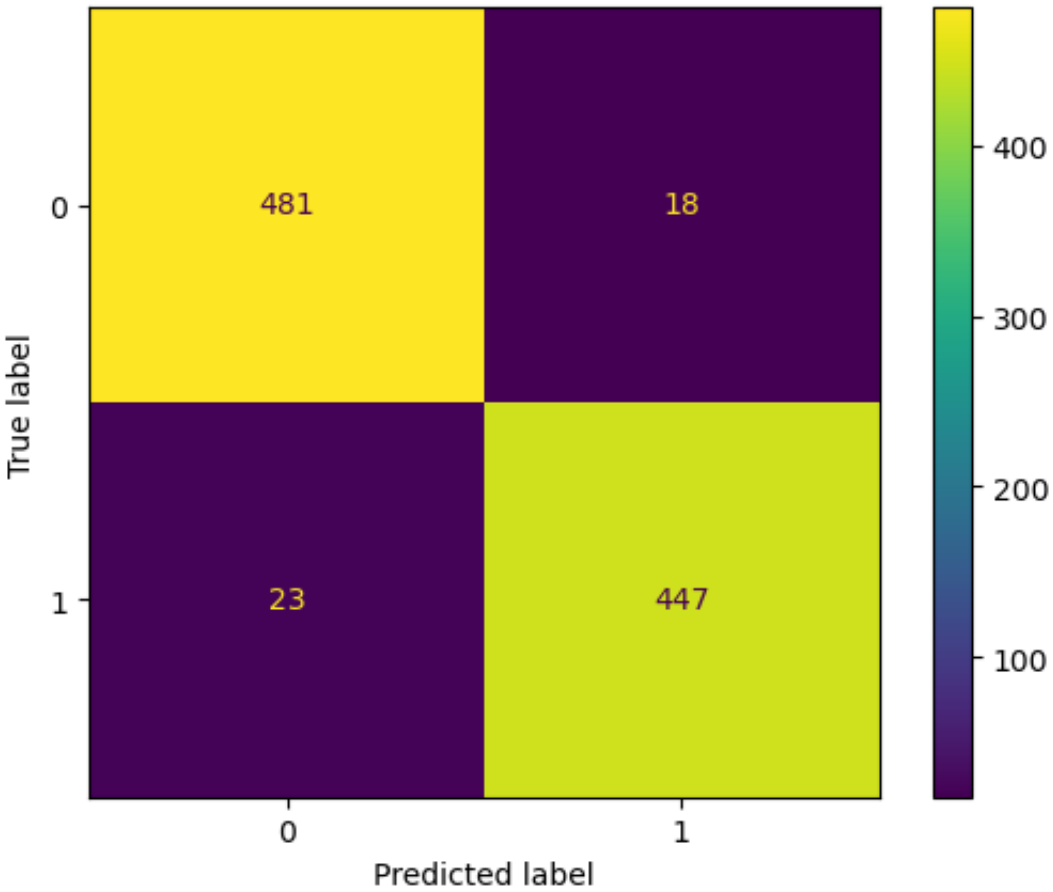
```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       499
           1       0.96      0.95      0.96       470

    accuracy                           0.96       969
   macro avg       0.96      0.96      0.96       969
weighted avg       0.96      0.96      0.96       969
```

Confusion Matrix

In [88]:
```
cm=confusion_matrix(y_test,pred)
ConfusionMatrixDisplay(cm).plot()
```

Out [88]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fd9a
803b190>
```



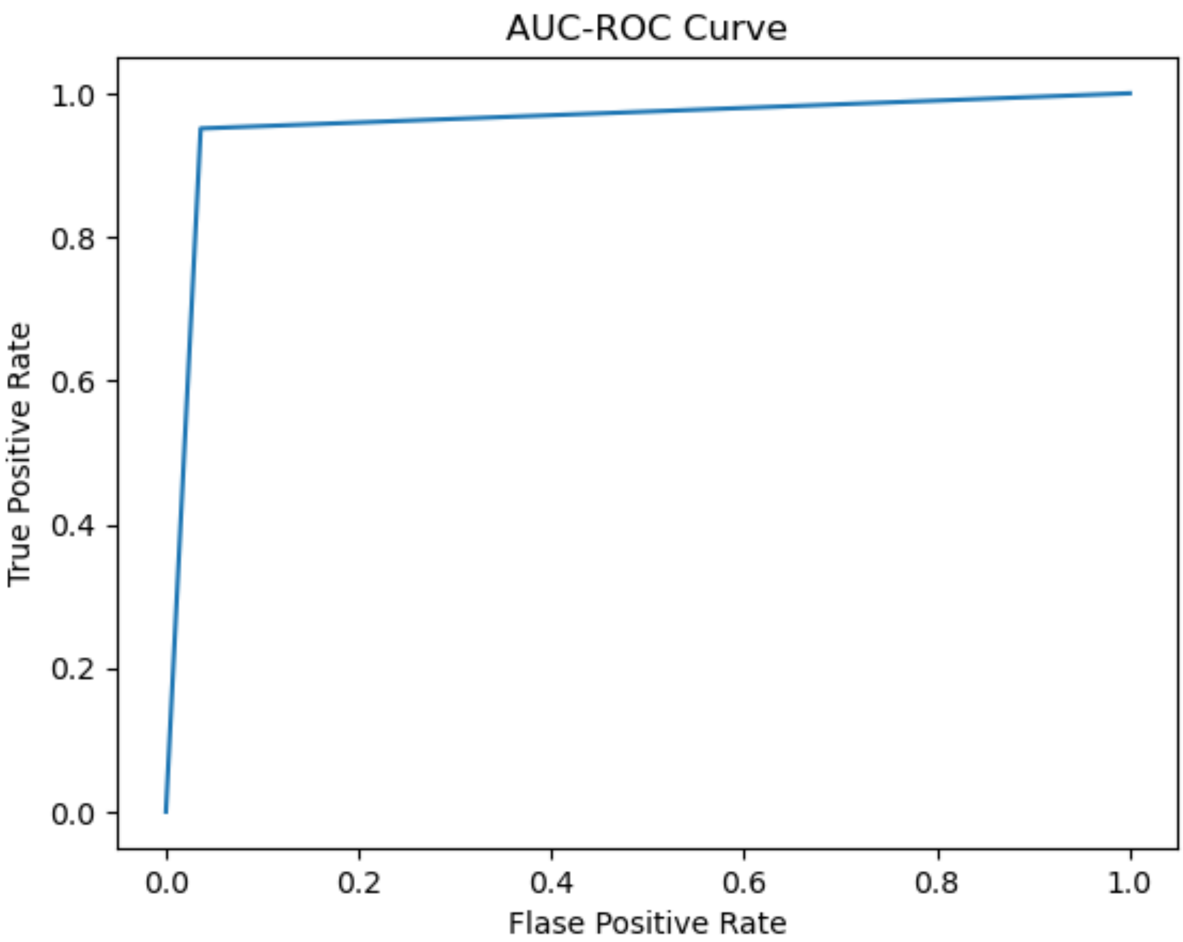
AUC-ROC Curve

In [89]:

```
fpr, tpr, thresholds = roc_curve(y_test, pred)
plt.plot(fpr,tpr)
plt.xlabel('Flase Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('AUC-ROC Curve')
```

Out[89]:

```
Text(0.5, 1.0, 'AUC-ROC Curve')
```



In [91]:
```
!pip install pandoc
```

```
Collecting pandoc
  Downloading pandoc-2.3.tar.gz (33 kB)
  Preparing metadata (setup.py) ... done
Collecting plumbum
  Downloading plumbum-1.8.0-py3-none-any.whl (117 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 117.5/117.5 kB 2.1 MB/s eta
0:00:00a 0:00:01
Requirement already satisfied: ply in /opt/anaconda3/lib/python3.9/site-p
ackages (from pandoc) (3.11)
Building wheels for collected packages: pandoc
  Building wheel for pandoc (setup.py) ... done
  Created wheel for pandoc: filename=pandoc-2.3-py3-none-any.whl size=332
61 sha256=b7950e52c6a0ea90cc905a797496c1818df885faf7a5404eb96ad940bbc44da
d
  Stored in directory: /Users/lms/Library/Caches/pip/wheels/69/e6/a1/1daa
96d919c9e09a71473649b717b8da286f3f8d7719d1cfc5
Successfully built pandoc
Installing collected packages: plumbum, pandoc
Successfully installed pandoc-2.3 plumbum-1.8.0
```

In [ ]: