Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from scipy.stats import uniform
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge, LogisticRegression
from sklearn.metrics import r2_score, precision_score, recall_score, f1_score
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, roc_curve
from sklearn.metrics import ConfusionMatrixDisplay, roc_auc_score, precision_recall_curve
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.impute import KNNImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold, cross_validate, GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import GradientBoostingClassifier
```

```
from imblearn.over_sampling import SMOTE
from statsmodels.stats.outliers_influence import variance_inflation_factor

#import warnings
#warnings.filterwarnings('ignore')
```

Loading Dataset

```
!wget https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/002/492/original/ola_driver_scaler.csv
```

```
--2024-03-19 12:32:57--  https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/002/492/original/ola_driver_scaler.csv
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 18.172.139.61, 18.172.139.46, 18.172.139.94, ...
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|18.172.139.61|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1127673 (1.1M) [text/plain]
Saving to: ‘ola_driver_scaler.csv’

ola_driver_scaler.c 100%[===================>]   1.08M  --.-KB/s    in 0.06s

2024-03-19 12:32:57 (17.5 MB/s) - ‘ola_driver_scaler.csv’ saved [1127673/1127673]
```

Analysing the Dataset

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

```
df.shape
```

```
(19104, 14)
```

```
df.columns
```

```
Index(['Unnamed: 0', 'MMM-YY', 'Driver_ID', 'Age', 'Gender', 'City',
       'Education_Level', 'Income', 'Dateofjoining', 'LastWorkingDate',
       'Joining Designation', 'Grade', 'Total Business Value',
       'Quarterly Rating'],
      dtype='object')
```

```
df.head()
```

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	238106

Next steps: Generate code with df View recommended plots

1	1	02/01/19	1	[illegible].0	0.0	C23	2	57387	24/12/18	NaN	1	1	-665480
2	2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18	03/11/19	1	1	

Basic Data Cleaning and Exploration

> ***Removing Unnecessary Column***

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

> ***Duplicate rows in the dataset***

```
df.duplicated().sum()
```

```
0
```

> ***Missing values in the dataset***

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

```
MMM-YY                   0
Driver_ID                0
Age                     61
Gender                  52
City                     0
Education_Level          0
Income                   0
Dateofjoining            0
LastWorkingDate      17488
Joining Designation      0
Grade                    0
Total Business Value     0
Quarterly Rating         0
dtype: int64
```

> ***Missing values Treatment***

> > **KNN Imputation**

> > Preparing data for KNN imputation

```
#consider only numerical features
num_df = df.select_dtypes('number')

#droping driver_id(reason is unique)
num_df.drop('Driver_ID', axis = 1, inplace = True)
```

> > KNN Imputation

```
imputer = KNNImputer(n_neighbors = 5)

num_df = pd.DataFrame(imputer.fit_transform(num_df), columns = num_df.columns)
num_df.head()
```

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating
0	28.0	0.0	2.0	57387.0	1.0	1.0	2381060.0	2.0
1	28.0	0.0	2.0	57387.0	1.0	1.0	-665480.0	2.0
2	28.0	0.0	2.0	57387.0	1.0	1.0	0.0	2.0
3	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0
4	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0

Next steps: Generate code with num_df View recommended plots

Concatening dataframes

```
#remaining columns
rem_col = list(df.columns.difference(num_df.columns))

new_df = pd.concat([df[rem_col], num_df], axis=1)

new_df.shape
```

```
(19104, 13)
```

Merging of rows and aggregation of fields

```
data = new_df.groupby(['Driver_ID']).agg({'MMM-YY': 'count', 'Age': 'max', 'Gender': 'max',
                                         'City': 'max', 'Education_Level': 'max', 'Income': 'mean',
                                         'Dateofjoining': 'first', 'LastWorkingDate': 'last', 'Joining Designation': 'max',
                                         'Grade': 'mean', 'Total Business Value': 'sum', 'Quarterly Rating': 'mean'})
```

```
data = data.rename(columns = {'MMM-YY': 'No_of_records', 'Dateofjoining': 'Date_of_joining',
                              'LastWorkingDate': 'Last_working_date', 'Joining Designation': 'Joining_designation',
                              'Total Business Value': 'Total_Business_value', 'Quarterly Rating': 'Quarterly_rating'})
```

```
data['Grade'] = np.round(data['Grade'])
data['Quarterly_rating'] = np.round(data['Quarterly_rating'])
```

```
data.shape
```

```
(2381, 12)
```

Column-wise unique entries:

```
for i in data.columns:
  print(f"Unique entries for column {i:20} = {data[i].nunique()}")
```

```
Unique entries for column No_of_records        = 24
Unique entries for column Age                  = 61
Unique entries for column Gender               = 6
Unique entries for column City                 = 29
Unique entries for column Education_Level      = 3
Unique entries for column Income               = 2339
Unique entries for column Date_of_joining      = 869
Unique entries for column Last_working_date    = 493
Unique entries for column Joining_designation  = 5
Unique entries for column Grade                = 5
Unique entries for column Total_Business_value = 1629
Unique entries for column Quarterly_rating     = 4
```

Updating Date-time Columns

```
data['Date_of_joining'] = pd.to_datetime(data['Date_of_joining'])
data['Last_working_date'] = pd.to_datetime(data['Last_working_date'])
```

> ***Conversion of Categorical Attributes to Category***

```
categorical_col = ['Gender', 'City', 'Education_Level', 'Joining_designation', 'Grade', 'Quarterly_rating']

for i in categorical_col:
  data[i] = data[i].astype('category')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2381 entries, 1 to 2788
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   No_of_records        2381 non-null   int64
 1   Age                  2381 non-null   float64
 2   Gender               2381 non-null   category
 3   City                 2381 non-null   category
 4   Education_Level      2381 non-null   category
 5   Income               2381 non-null   float64
 6   Date_of_joining      2381 non-null   datetime64[ns]
 7   Last_working_date    1616 non-null   datetime64[ns]
 8   Joining_designation  2381 non-null   category
 9   Grade                2381 non-null   category
 10  Total_Business_value  2381 non-null   float64
 11  Quarterly_rating     2381 non-null   category
dtypes: category(6), datetime64[ns](2), float64(3), int64(1)
memory usage: 146.4 KB
```

Feature Engineering

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

> ***Whether the Quarterly Rating has increased for that driver -***

> > for those whose Quarterly Rating has increased we assign the value 1

```
data['Quarterly_rating_increased'] = df.groupby(['Driver_ID'])['Quarterly Rating'].unique().apply(check_value)
```

```
data.Quarterly_rating_increased.value_counts()
```

```
0    1789
1     592
Name: Quarterly_rating_increased, dtype: int64
```

> ***Whether the monthly Income has increased for that driver -***

> > for those whose monthly Income has increased we assign the value 1

```
data['Income_increased'] = df.groupby(['Driver_ID'])['Income'].unique().apply(check_value)
```

```
data.Income_increased.value_counts()
```

```
0    2337
1      44
Name: Income_increased, dtype: int64
```

> ***Target variable creation:***

> > Driver whose last working day is present will have the value 1

```
def check_day(x):
  if x == 0:
    return 0
  else:
    return 1
```

```
data['Target'] = (data['Last_working_date'].fillna(0)).apply(check_day)
```

```
data.Target.value_counts()
```

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

> ***Number of Months working***

> > Creating column based on their months of working

```
data['No_of_months'] = (data['Last_working_date'].dt.year - data['Date_of_joining'].dt.year) * 12 + (data['Last_working_date'].dt.month - da
```

```
data['No_of_months'].fillna(0, inplace = True)
```

> ***Extract Year from Date***

```
data['Joining_year'] = data['Date_of_joining'].dt.year

data['Leaving_year'] = data['Last_working_date'].dt.year
```

```
data['Joining_year'].fillna(0, inplace = True)
data['Leaving_year'].fillna(0, inplace = True)
```

Final Dataset after removing unnecessary columns

```
data.head()
```

	No_of_records	Age	Gender	City	Education_Level	Income	Date_of_joining	Last_working_date	Joining_designation	Grade	T
Driver_ID											
1	3	28.0	0.0	C23	2.0	57387.0	2018-12-24	2019-03-11	1.0	1.0	
2	2	31.0	0.0	C7	2.0	67016.0	2020-11-06	NaT	2.0	2.0	
4	5	43.0	0.0	C13	2.0	65603.0	2019-12-07	2020-04-27	2.0	2.0	
5	3	29.0	0.0	C9	0.0	46368.0	2019-01-09	2019-03-07	1.0	1.0	
6	5	31.0	1.0	C11	1.0	78728.0	2020-07-31	NaT	3.0	3.0	

Next steps: Generate code with `data` View recommended plots

```
f_data = data.drop(columns=['Date_of_joining', 'Last_working_date', 'Quarterly_rating'])
```

```
f_data.shape
```

```
(2381, 15)
```

Statistical Summary

```
f_data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
No_of_records	2381.0	8.023520e+00	6.783590e+00	1.0	3.0	5.0	10.0	24.0
Age	2381.0	3.377018e+01	5.933265e+00	21.0	30.0	33.0	37.0	58.0
Income	2381.0	5.923246e+04	2.829821e+04	10747.0	39104.0	55285.0	75835.0	188418.0
Total_Business_value	2381.0	4.586742e+06	9.127115e+06	-1385530.0	0.0	817680.0	4173650.0	95331060.0
Quarterly_rating_increased	2381.0	2.486350e-01	4.323126e-01	0.0	0.0	0.0	0.0	1.0
Income_increased	2381.0	1.847963e-02	1.347062e-01	0.0	0.0	0.0	0.0	1.0
Target	2381.0	6.787064e-01	4.670713e-01	0.0	0.0	1.0	1.0	1.0
No_of_months	2381.0	7.941621e+00	1.351620e+01	0.0	0.0	3.0	9.0	85.0
Joining_year	2381.0	2.018536e+03	1.609597e+00	2013.0	2018.0	2019.0	2020.0	2020.0
Leaving_year	2381.0	1.370636e+03	9.432429e+02	0.0	0.0	2019.0	2020.0	2020.0

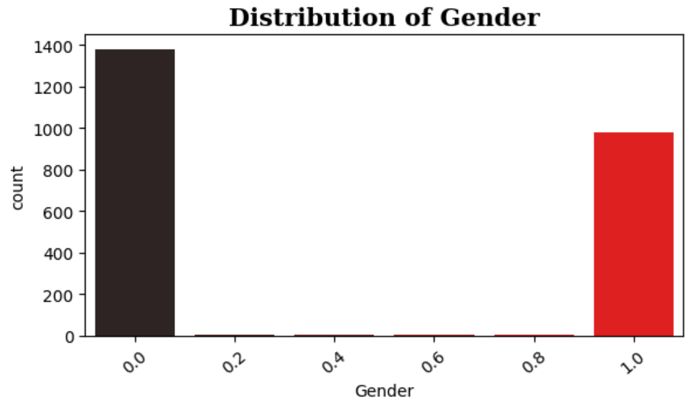
```
f_data.describe(include = 'category').T
```

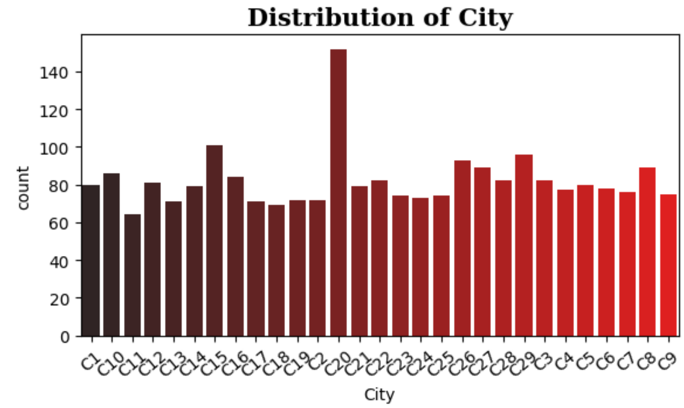
	count	unique	top	freq
Gender	2381.0	6.0	0.0	1382.0
City	2381	29	C20	152
Education_Level	2381.0	3.0	2.0	802.0
Joining_designation	2381.0	5.0	1.0	1026.0
Grade	2381.0	5.0	2.0	866.0

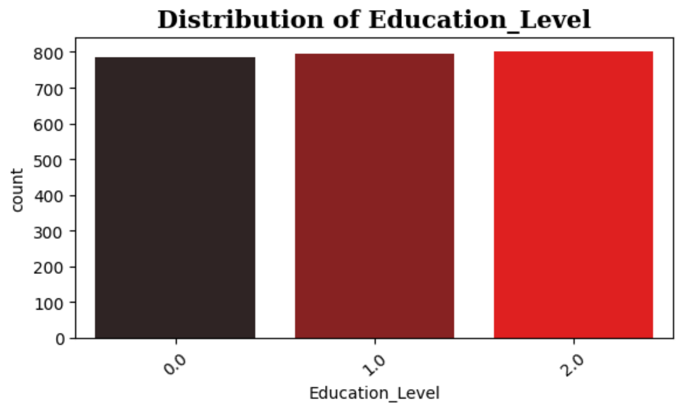
Univeriate Analysis

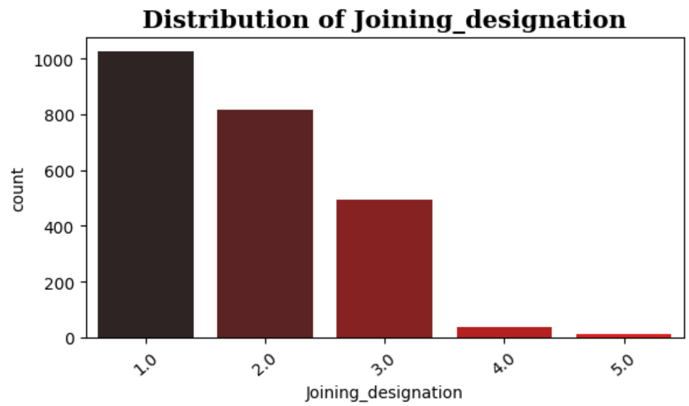
> ***Distribution Plot for Categorical variables***

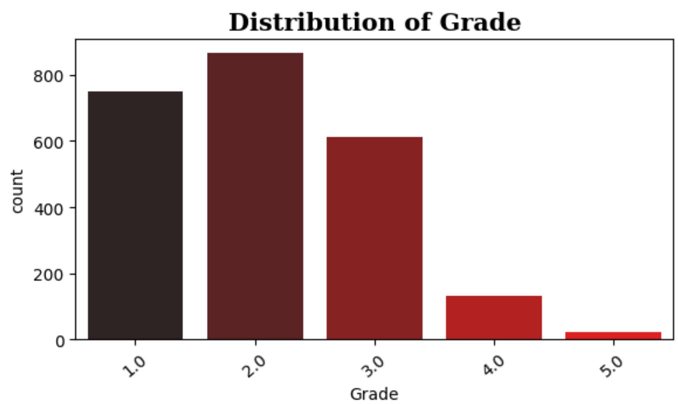
```
categorical = ['Gender', 'City', 'Education_Level', 'Joining_designation',
               'Grade', 'Quarterly_rating_increased', 'Income_increased',
               'Joining_year', 'Leaving_year']

plt.figure(figsize=(12,18))
for i in range(1,10):
  ax=plt.subplot(5,2,i)
  sns.countplot(x=f_data[categorical[i-1]], hue = f_data[categorical[i-1]], legend= False, palette = 'dark:red')
  plt.title(f'Distribution of {categorical[i-1]}', font = 'serif', weight = 'bold', size = 15)
  plt.xticks(rotation = 40)

plt.tight_layout()
plt.show()
```

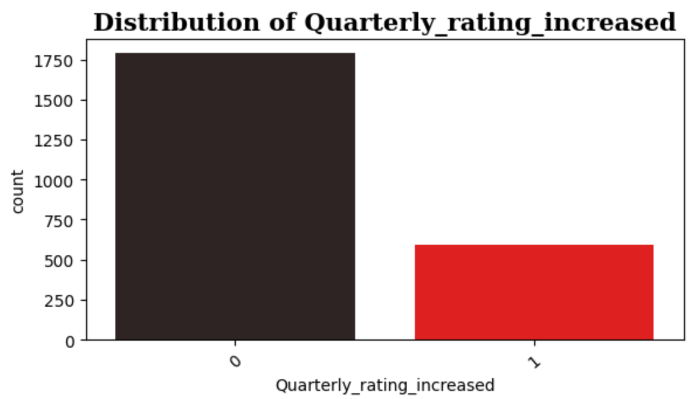

Distribution of Gender
count
0
200
400
600
800
1000
1200
1400
0.0
0.2
0.4
0.6
0.8
1.0
Gender

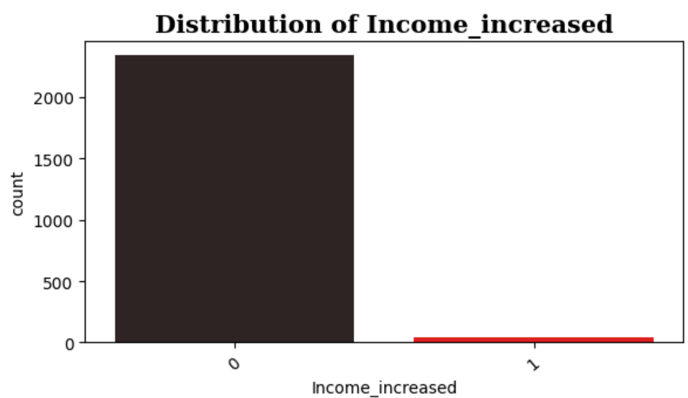

Distribution of City
count
0
20
40
60
80
100
120
140
C1 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C2 C20 C21 C22 C23 C24 C25 C26 C27 C28 C29 C3 C4 C5 C6 C7 C8 C9
City

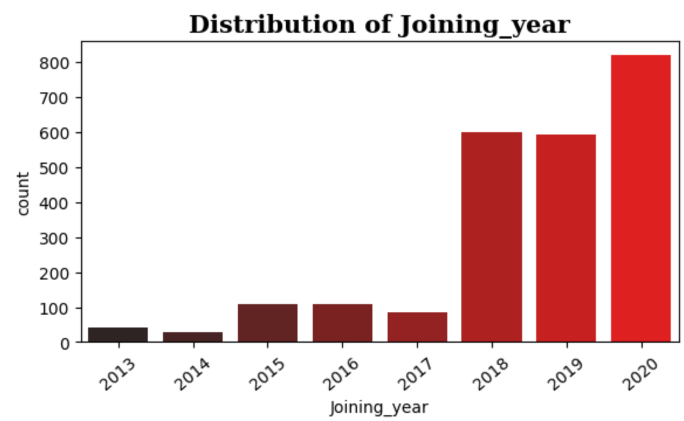

Distribution of Education_Level
count
0
100
200
300
400
500
600
700
800
0.0
1.0
2.0
Education_Level

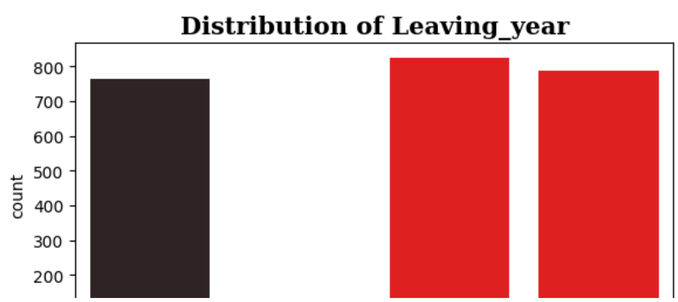

Distribution of Joining_designation
count
0
200
400
600
800
1000
1.0
2.0
3.0
4.0
5.0
Joining_designation


Distribution of Grade
count
0
200
400
600
800
1.0
2.0
3.0
4.0
5.0
Grade


Distribution of Quarterly_rating_increased
count
0
250
500
750
1000
1250
1500
1750
0
1
Quarterly_rating_increased


Distribution of Income_increased
count
0
500
1000
1500
2000
0
1
Income_increased


Distribution of Joining_year
count
0
100
200
300
400
500
600
700
800
2013
2014
2015
2016
2017
2018
2019
2020
Joining_year

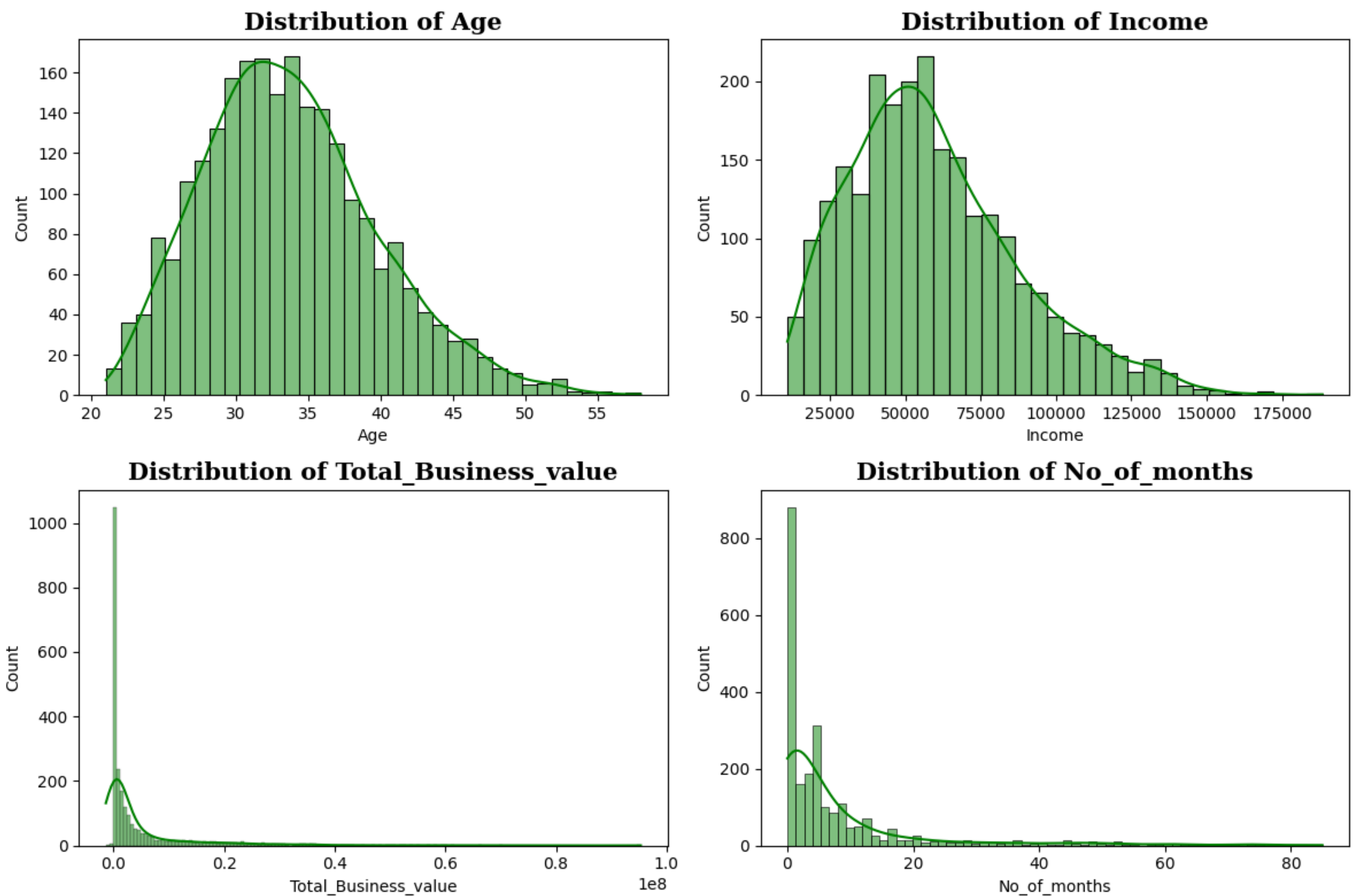

Distribution of Leaving_year
count
200
300
400
500
600
700
800

> ***Distribution Plot For Contineous Variables***

```
f_data.select_dtypes('number').columns
```

```
Index(['No_of_records', 'Age', 'Income', 'Total_Business_value',
       'Quarterly_rating_increased', 'Income_increased', 'Target',
       'No_of_months', 'Joining_year', 'Leaving_year'],
      dtype='object')
```

```
contineous = ['Age', 'Income', 'Total_Business_value', 'No_of_months']

plt.figure(figsize=(12,8))
for i in range(1,5):
  ax=plt.subplot(2,2,i)
  sns.histplot(f_data[contineous[i-1]], kde = True, color = 'green')
  plt.title(f'Distribution of {contineous[i-1]}', font = 'serif', weight = 'bold', size = 15)

plt.tight_layout()
plt.show()
```



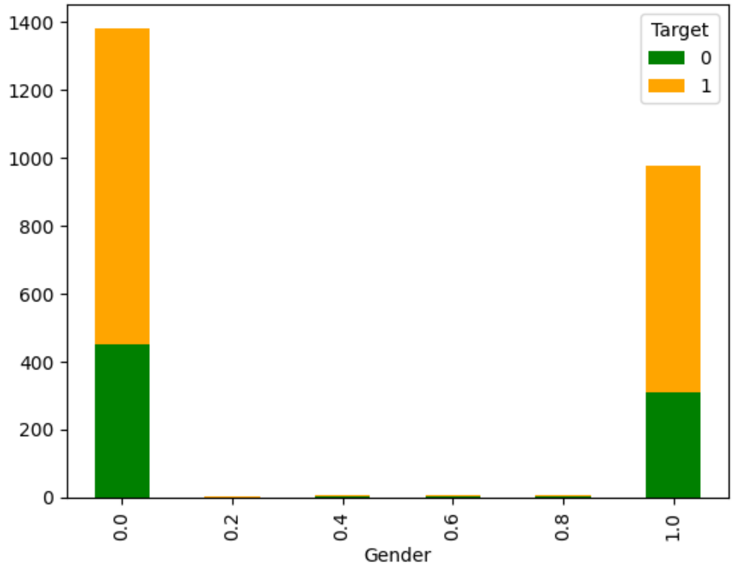
Bi-variate Analysis

> ***Impact of Categorical variables on Target variable***

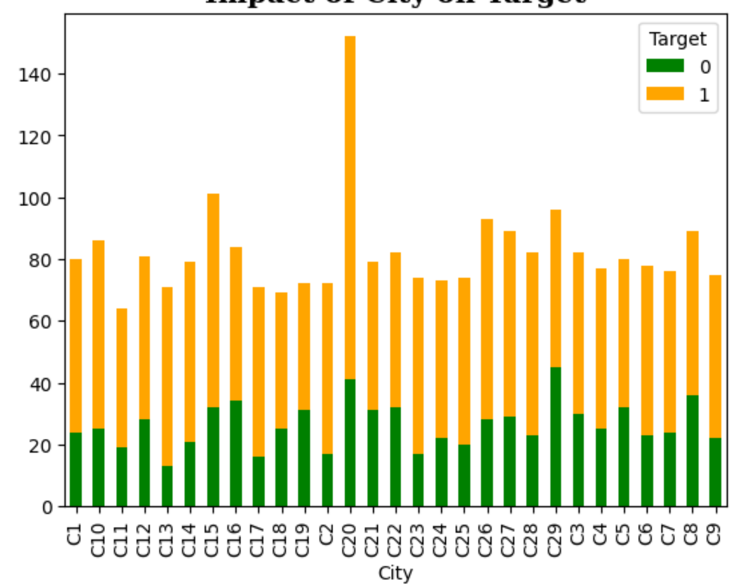
```
#categorical variables
def bar_plot(cat):
  bar = f_data.groupby(cat)['Target'].value_counts().unstack().plot(kind='bar', stacked=True, color = ['green', 'orange']
```

```
cat = ['Gender', 'City', 'Education_Level', 'Joining_designation',
               'Grade', 'Quarterly_rating_increased', 'Income_increased',
               'Joining_year']
for i in cat:
  bar_plot(i)
```

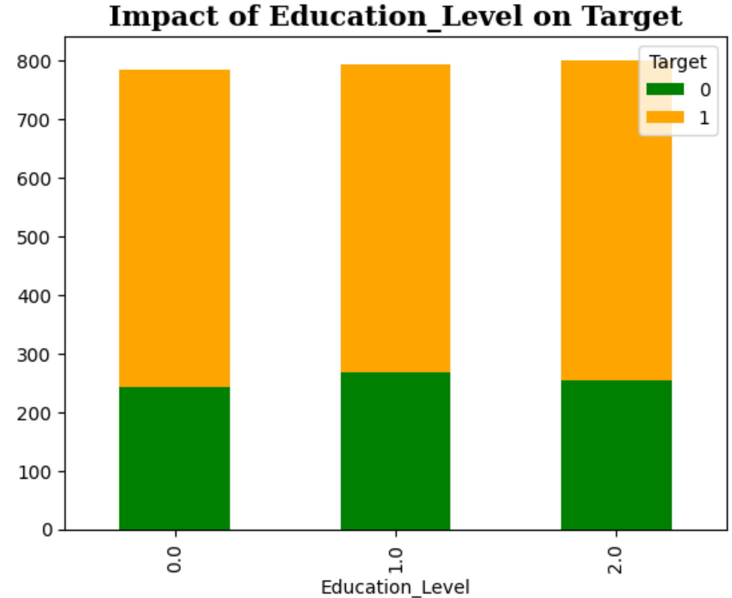
Impact of Gender on Target



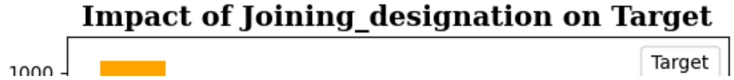
Impact of City on Target



Impact of Education_Level on Target

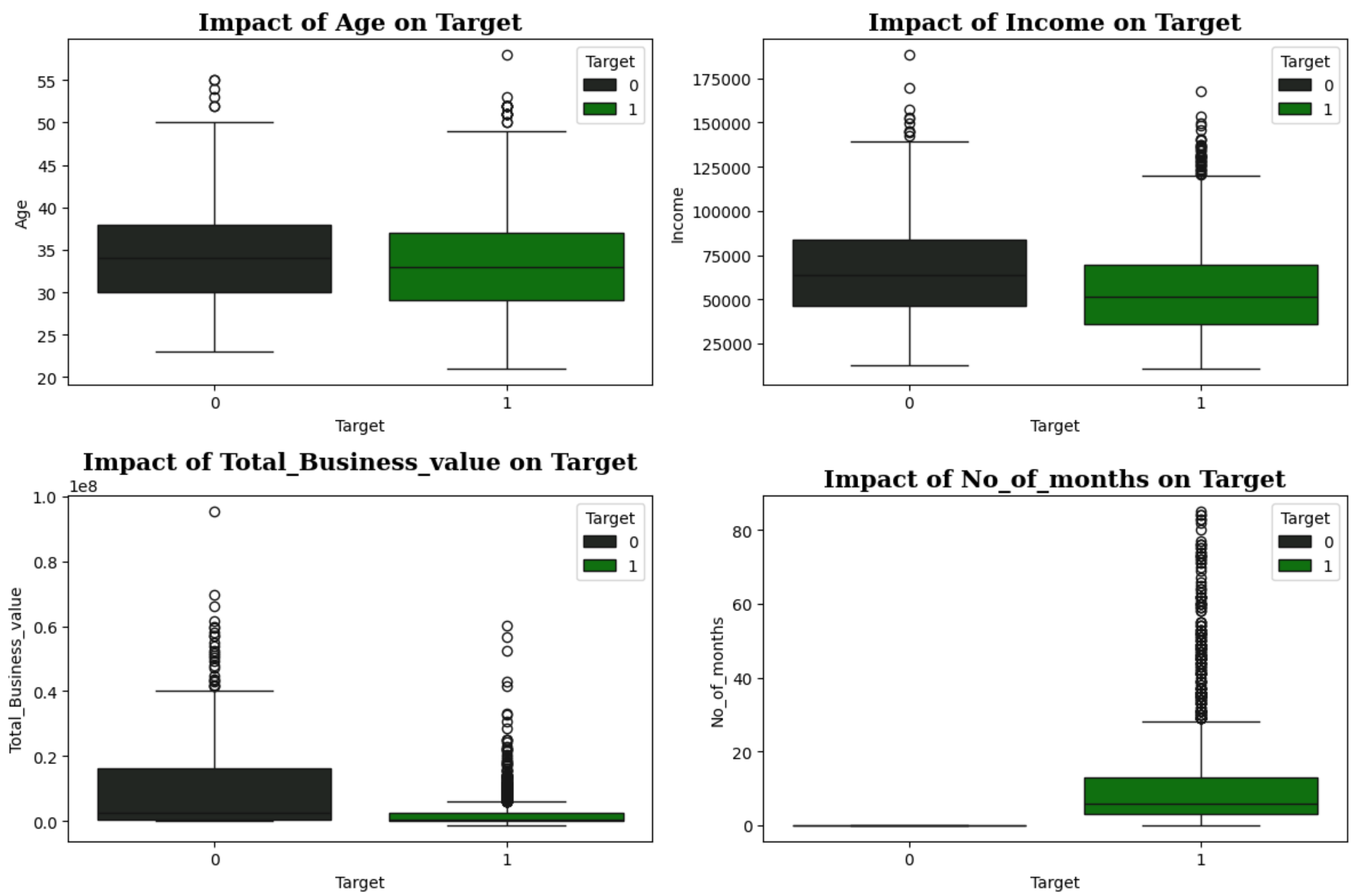


Impact of Joining_designation on Target



> ***Impact of Contineous variables on Target variable***

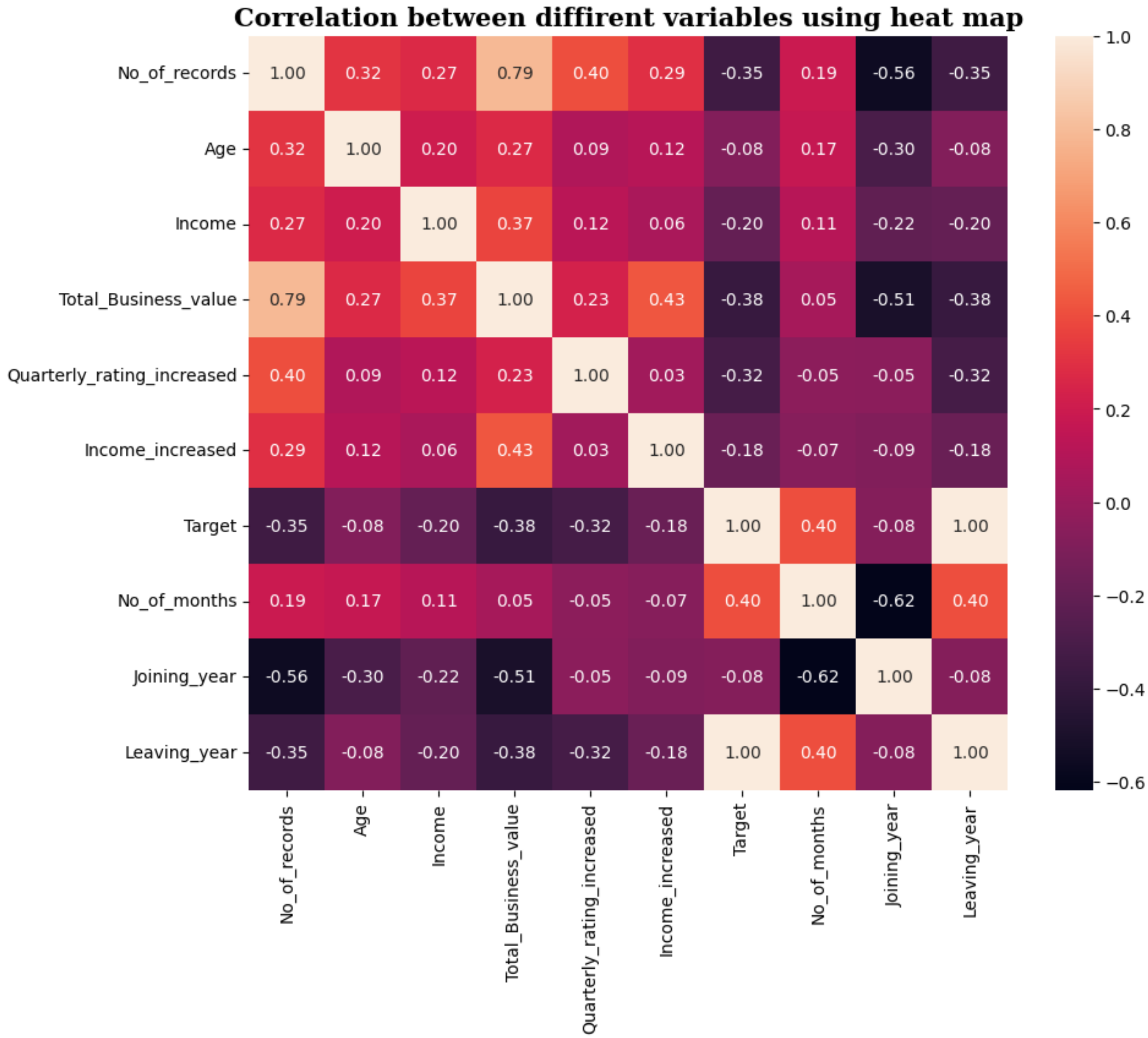
```
num_col = ['Age', 'Income', 'Total_Business_value', 'No_of_months']

plt.figure(figsize=(12, 8))

for i in range(1, 5):
  ax = plt.subplot(2, 2, i)
  sns.boxplot(x = f_data['Target'], y = f_data[num_col[i-1]], hue = f_data['Target'], palette='dark:green')
  plt.title(f'Impact of {num_col[i-1]} on Target', font = 'serif', weight = 'bold', size = 15)

plt.tight_layout()
plt.show()
```



Multi_variate Analysis

```
plt.figure(figsize = (10, 8))

sns.heatmap(data = f_data.corr(), annot = True, fmt = '0.2f')

plt.title('Correlation between diffirent variables using heat map', font='serif', size=15, weight='bold')

plt.show()
```

```
<ipython-input-46-847ba243bca9>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future versior
  sns.heatmap(data = f_data.corr(), annot = True, fmt = '0.2f')
```




```
f_data.shape
```

```
(2381, 15)
```

Model Building

> ***Data Preparetion for Model Building***

```
#pre-processing of Data
x = f_data.drop('Target', axis = 1)
y = f_data['Target']
```

```
print("Shape of x: ", x.shape)
print("Shape of y: ", y.shape)
```

```
Shape of x:  (2381, 14)
Shape of y:  (2381,)
```

> ***Transforming Categorical Columns by OneHotEncoding***

```
x = pd.get_dummies(x, columns = ['City'])

x.head()
```

	No_of_records	Age	Gender	Education_Level	Income	Joining_designation	Grade	Total_Business_value	Quarterly_rating_incr
Driver_ID									
1	3	28.0	0.0	2.0	57387.0	1.0	1.0	1715580.0	
2	2	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	
4	5	43.0	0.0	2.0	65603.0	2.0	2.0	350000.0	
5	3	29.0	0.0	0.0	46368.0	1.0	1.0	120360.0	
6	5	31.0	1.0	1.0	78728.0	3.0	3.0	1265000.0	

5 rows × 42 columns

Split the data

```
#further split the validation-train data to train and test data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state = 1)
```

```
print("Shape of x_train, y_train: ", x_train.shape, y_train.shape)
print("Shape of x_test, y_test: ", x_test.shape, y_test.shape)
```

```
Shape of x_train, y_train:  (1904, 42) (1904,)
Shape of x_test, y_test:  (477, 42) (477,)
```

Class Imbalance Treatment

Oversampling by SMOTE

```
#checking train data Imbalance
y_train.value_counts()
```

```
1    1287
0     617
Name: Target, dtype: int64
```

```
#oversampling by SMOTE technique
smt = SMOTE()
```

```
#fit SMOTE to training data
X_sm, y_sm = smt.fit_resample(x_train, y_train)
```

```
#After oversampling imbalance check
y_sm.value_counts()
```

```
1    1287
0    1287
Name: Target, dtype: int64
```

Standardization of training data

```
scaler = StandardScaler()

#standardization
X_train = pd.DataFrame(scaler.fit_transform(x_train), columns = x.columns)

X_test = pd.DataFrame(scaler.transform(x_test), columns = x.columns)
```

```
X_train.head()
```

	No_of_records	Age	Gender	Education_Level	Income	Joining_designation	Grade	Total_Business_value	Quarterly_rating_:
0	0.138382	-1.124970	-0.451046	0.006426	-1.104915	-0.972756	-1.153776	-0.380985	
1	-0.596363	-1.291532	-0.857864	-1.217060	-1.482449	-0.972756	-1.153776	-0.516264	
2	-0.302465	-0.958409	1.176224	1.229912	0.150893	0.234628	-0.070012	-0.418969	
3	-0.596363	0.707203	1.176224	-1.217060	-0.227892	-0.972756	-1.153776	-0.497448	
4	2.342618	1.540009	-0.857864	1.229912	-0.623027	-0.972756	-1.153776	0.995936	

5 rows × 42 columns

Ensemble Learning - Bagging Algorithm

> **Implementation of Random Forest**

```
rf_clf = RandomForestClassifier(class_weight='balanced', random_state=7, max_depth=4, n_estimators=100)
```

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

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=4, random_state=7)
```

```
print("Training Score:", rf_clf.score(X_train, y_train)*100)
print("Testing Score:", rf_clf.score(X_test, y_test)*100)
```

```
Training Score: 100.0
Testing Score: 100.0
```

```
y_train_pred = rf_clf.predict(X_train)
y_test_pred = rf_clf.predict(X_test)

print("f1 Score for Train data:", f1_score(y_train, y_train_pred))
print("f1 Score for Test data:", f1_score(y_test, y_test_pred))
print("Prescision Score for Train data:", precision_score(y_train, y_train_pred))
print("Precision Score for Test data:", precision_score(y_test, y_test_pred))
print("Recall Score for Train data:", recall_score(y_train, y_train_pred))
print("Recall Score for Test data:", recall_score(y_test, y_test_pred))
```

```
f1 Score for Train data: 1.0
f1 Score for Test data: 1.0
Prescision Score for Train data: 1.0
Precision Score for Test data: 1.0
Recall Score for Train data: 1.0
Recall Score for Test data: 1.0
```

> > Using Cross validation for better result

```
kfold = KFold(n_splits=10)
cv_acc_results = cross_validate(rf_clf, X_train, y_train, cv=kfold, scoring='accuracy', return_train_score=True)

print(f"K-Fold Accuracy Mean: \n Train: {cv_acc_results['train_score'].mean()*100:.2f} \n Test: {cv_acc_results['test_score'].mean()*100:.2f
print(f"K-Fold Accuracy Std: \n Train: {cv_acc_results['train_score'].std()*100:.2f}, \n Test: {cv_acc_results['test_score'].std()*100:.2f}'
```

```
K-Fold Accuracy Mean:
 Train: 100.00
 Test: 100.00
K-Fold Accuracy Std:
 Train: 0.00,
 Test: 0.00
```

> **Hyperparameter Tuning**

Grid Search CV

```
# Defining parameters -

params = {
          'n_estimators' : [100,200,300,400],
          'max_depth' : [3,5,10],
          'criterion' : ['gini', 'entropy'],
          'bootstrap' : [True, False],
          'max_features' : [8,9,10]
         }
```

```
grid = GridSearchCV(estimator = RandomForestClassifier(),
                    param_grid = params,
                    scoring = 'accuracy',
                    cv = 3,
                    n_jobs=-1
                    )
```

```
grid.fit(X_train, y_train)

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

```
Best params:  {'bootstrap': True, 'criterion': 'gini', 'max_depth': 3, 'max_features': 8, 'n_estimators': 100}
Best score:  1.0
```

```
rf_clf2 = RandomForestClassifier(class_weight='balanced', random_state=7, bootstrap=True, criterion='gini',
                              max_depth=2, max_features=4, n_estimators=100)

kfold = KFold(n_splits=10)
cv_acc_results = cross_validate(rf_clf2, X_train, y_train, cv=kfold, scoring='accuracy', return_train_score=True)

print(f"K-Fold Accuracy Mean: \n Train: {cv_acc_results['train_score'].mean()*100:.3f} \n Test: {cv_acc_results['test_score'].mean()*100:.3f
print(f"K-Fold Accuracy Std: \n Train: {cv_acc_results['train_score'].std()*100:.3f}, \n Test: {cv_acc_results['test_score'].std()*100:.3f}'
```

```
K-Fold Accuracy Mean:
 Train: 99.568
 Test: 99.633
K-Fold Accuracy Std:
 Train: 0.098,
 Test: 0.409
```

Randomized Search

```
# Defining parameters -
params = {'ccp_alpha': uniform(loc=0, scale=0.4)}
```

```
random = RandomizedSearchCV(estimator = RandomForestClassifier(class_weight='balanced', random_state=7, bootstrap=True, criterion='gini',
                                                               max_depth=2, max_features=4, n_estimators=100),
                            param_distributions = params,
                            scoring = 'accuracy',
                            cv = 3,
                            n_iter=15,
                            n_jobs=-1
                            )
```

```
random.fit(X_train, y_train)

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

```
Best param:  {'ccp_alpha': 0.2861641672327031}
Best score:  1.0
```

```
rf_clf3 = RandomForestClassifier(class_weight='balanced', random_state=7, bootstrap=True, criterion='gini',
                              max_depth=2, max_features=4, ccp_alpha = 0.28, n_estimators=100)
```

```
kfold = KFold(n_splits=10)
cv_acc_results = cross_validate(rf_clf3, X_train, y_train, cv=kfold, scoring='accuracy', return_train_score=True)

print(f"K-Fold Accuracy Mean: \n Train: {cv_acc_results['train_score'].mean()*100:.3f} \n Test: {cv_acc_results['test_score'].mean()*100:.
```

```
K-Fold Accuracy Mean:
 Train: 99.796
 Test: 99.843
K-Fold Accuracy Std:
 Train: 0.409,
 Test: 0.335
```

Result Evaluation

Classification Report & Confusion Matrix

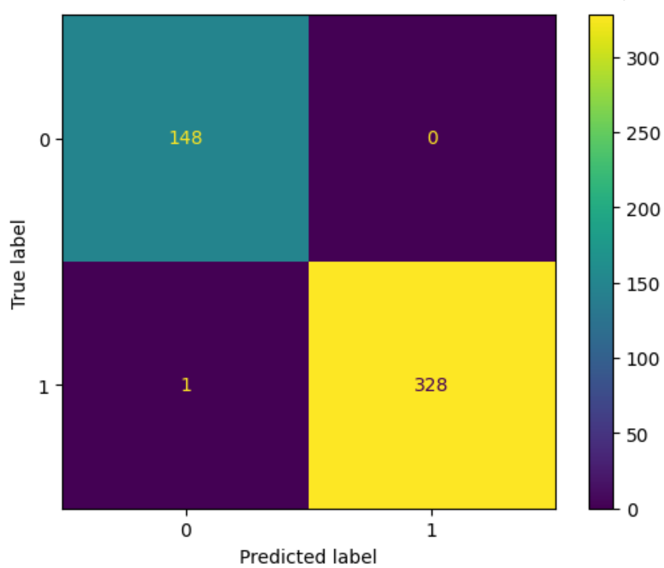
```
rf_clf3.fit(X_train, y_train)
```

```
                     RandomForestClassifier
RandomForestClassifier(ccp_alpha=0.04, class_weight='balanced', max_depth=2,
                       max_features=4, random_state=7)
```

```
y_pred = rf_clf3.predict(X_test)

print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rf_clf3.classes_).plot()
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       148
           1       1.00      1.00      1.00       329

    accuracy                           1.00       477
   macro avg       1.00      1.00      1.00       477
weighted avg       1.00      1.00      1.00       477
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ec348279ed0>
```

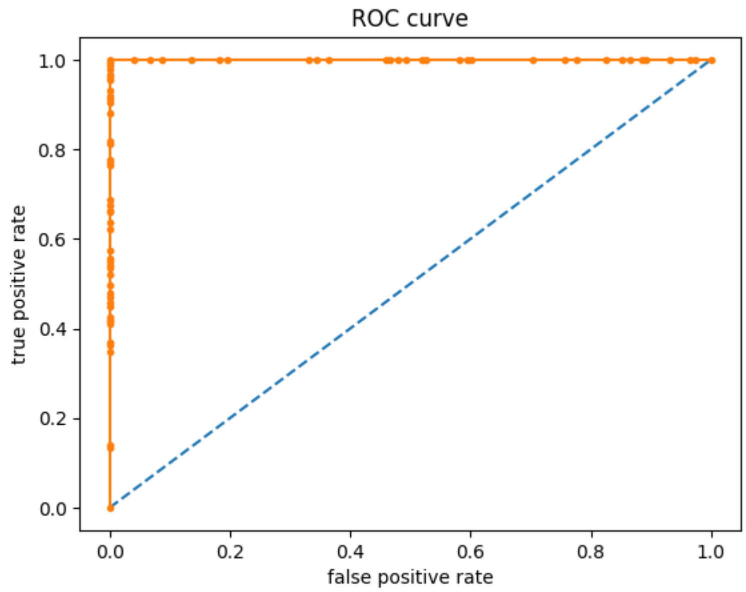


ROC-AUC Curve

```
y_pred = rf_clf3.predict(X_test)
prob = rf_clf3.predict_proba(X_test)
probs = prob[:,1]
```

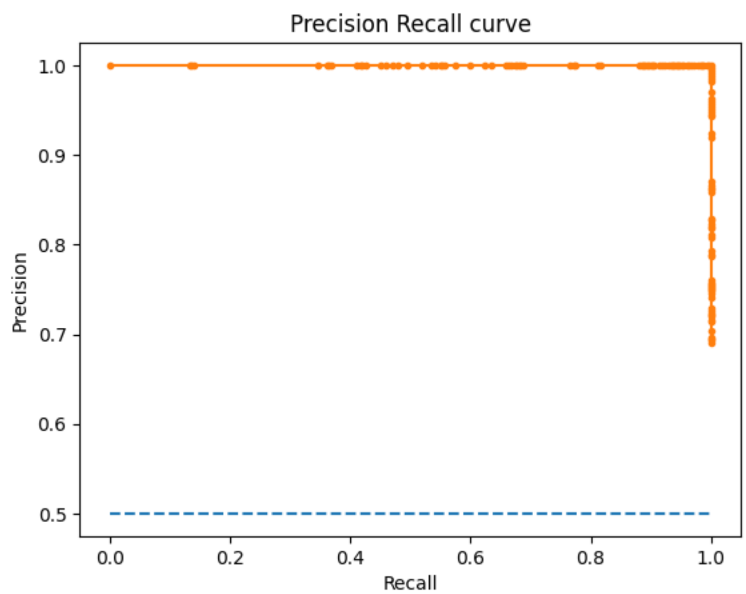
```
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
```

```
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
plt.title("ROC curve")
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
# show the plot
plt.show()
```



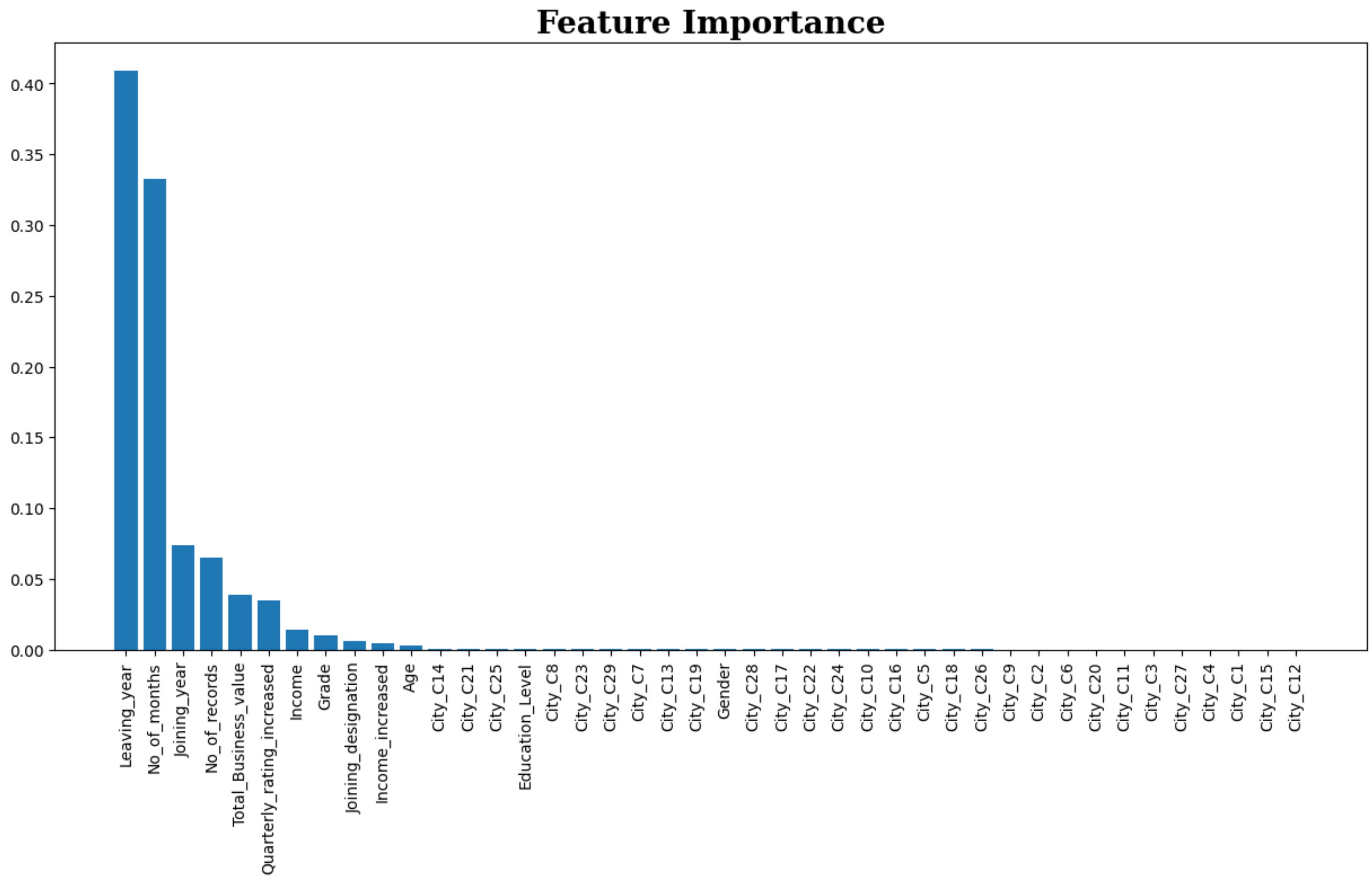
Precision Recall Curve

```
precision, recall, thresholds = precision_recall_curve(y_test, probs)
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
plt.title("Precision Recall curve")
plt.xlabel('Recall')
plt.ylabel('Precision')
# show the plot
plt.show()
```



Comuting Feature Importance

```
# Feature Importance
rf_clf.fit(X_train, y_train)
importances = rf_clf.feature_importances_

indices = np.argsort(importances)[::-1] # Sort feature importances in descending order
names = [X_train.columns[i] for i in indices] # Rearrange feature names so they match the sorted feature importances

plt.figure(figsize=(15, 7)) # Create plot
plt.title("Feature Importance", font = "serif", weight = 'bold', size = 20) # Create plot title
plt.bar(range(X_train.shape[1]), importances[indices]) # Add bars
plt.xticks(range(X_train.shape[1]), names, rotation=90) # Add feature names as x-axis labels
plt.show() # Show plot
```



Ensemble Learning- Boosting Algorithm

> **Gradient Boosting Classifier**

```
parameters = {
    "n_estimators": [50,80,100],
    "max_depth" : [2, 3, 4],
    "max_leaf_nodes" : [5, 10, 20],
    "learning_rate": [0.1, 0.2]
}
```

```
gbc = GradientBoostingClassifier()

clf = RandomizedSearchCV(gbc, parameters, scoring = "accuracy", cv=3, n_jobs = -1, verbose = 1)
```

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

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

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
▸ RandomizedSearchCV
▸ estimator: GradientBoostingClassifier
  ▸ GradientBoostingClassifier
```

```
res = clf.cv_results_

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

```
Parameters:{'n_estimators': 100, 'max_leaf_nodes': 10, 'max_depth': 4, 'learning_rate': 0.2} Mean_score: 1.0 Rank: 1
Parameters:{'n_estimators': 80, 'max_leaf_nodes': 10, 'max_depth': 4, 'learning_rate': 0.1} Mean_score: 1.0 Rank: 1
Parameters:{'n_estimators': 50, 'max_leaf_nodes': 5, 'max_depth': 3, 'learning_rate': 0.2} Mean_score: 1.0 Rank: 1
Parameters:{'n_estimators': 100, 'max_leaf_nodes': 5, 'max_depth': 3, 'learning_rate': 0.2} Mean_score: 1.0 Rank: 1
Parameters:{'n_estimators': 80, 'max_leaf_nodes': 20, 'max_depth': 3, 'learning_rate': 0.2} Mean_score: 1.0 Rank: 1
Parameters:{'n_estimators': 80, 'max_leaf_nodes': 20, 'max_depth': 4, 'learning_rate': 0.1} Mean_score: 1.0 Rank: 1
Parameters:{'n_estimators': 50, 'max_leaf_nodes': 10, 'max_depth': 3, 'learning_rate': 0.1} Mean_score: 1.0 Rank: 1
Parameters:{'n_estimators': 50, 'max_leaf_nodes': 5, 'max_depth': 4, 'learning_rate': 0.1} Mean_score: 1.0 Rank: 1
Parameters:{'n_estimators': 100, 'max_leaf_nodes': 5, 'max_depth': 2, 'learning_rate': 0.1} Mean_score: 1.0 Rank: 1
Parameters:{'n_estimators': 100, 'max_leaf_nodes': 10, 'max_depth': 4, 'learning_rate': 0.1} Mean_score: 1.0 Rank: 1
```

```
print(clf.best_estimator_)
```

```
GradientBoostingClassifier(learning_rate=0.2, max_depth=4, max_leaf_nodes=10)
```

```
gbc = clf.best_estimator_

gbc.fit(X_train, y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.2, max_depth=4, max_leaf_nodes=10)
```

Result Evaluation

Classification Report and Confusion Matrix

```
y_pred = gbc.predict(X_test)
prob = gbc.predict_proba(X_test)
probs = prob[:,1]
```

```
cm = confusion_matrix(y_test, y_pred)
print('Train Score : ', gbc.score(X_train, y_train), '\n')
print('Test Score : ', gbc.score(X_test, y_test), '\n')
print('Accuracy Score : ', accuracy_score(y_test, y_pred), '\n')
print(cm, "---> confusion Matrix ", '\n')
print("ROC-AUC score  test dataset:  ", roc_auc_score(y_test, prob[:, 1]),'\n')
print("precision score  test dataset:  ", precision_score(y_test, y_pred),'\n')
print("Recall score  test dataset:  ", recall_score(y_test, y_pred), '\n')
print("f1 score  test dataset :  ", f1_score(y_test, y_pred), '\n')
```

```
Train Score :  1.0

Test Score :  1.0

Accuracy Score :  1.0

[[148   0]
 [  0 329]] ---> confusion Matrix

ROC-AUC score  test dataset:   1.0
```