In [1]:
```python
# useful imports
import numpy as np, seaborn as sns, pandas as pd, matplotlib.pyplot as plt, scipy
df = pd.read_csv('driver.csv')
# df.head()
```

In [3]:
```python
df.drop(['Unnamed: 0'],axis=1,inplace=True)
```

In [4]:
```python
df.info() # some missing values are seen
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   MMM-YY                19104 non-null  object 
 1   Driver_ID             19104 non-null  int64  
 2   Age                   19043 non-null  float64
 3   Gender                19052 non-null  float64
 4   City                  19104 non-null  object 
 5   Education_Level       19104 non-null  int64  
 6   Income                19104 non-null  int64  
 7   Dateofjoining         19104 non-null  object 
 8   LastWorkingDate       1616 non-null   object 
 9   Joining Designation   19104 non-null  int64  
 10  Grade                 19104 non-null  int64  
 11  Total Business Value  19104 non-null  int64  
 12  Quarterly Rating      19104 non-null  int64  
dtypes: float64(2), int64(7), object(4)
memory usage: 1.9+ MB
```

In [5]:
```python
df.columns[(df.dtypes == "float64") | (df.dtypes == "int64")]
```

Out[5]:
```
Index(['Driver_ID', 'Age', 'Gender', 'Education_Level', 'Income',
       'Joining Designation', 'Grade', 'Total Business Value',
       'Quarterly Rating'],
      dtype='object')
```

The categorical columns are:

In [6]:
```python
df.columns[df.dtypes =="object"]
```

Out[6]:
```
Index(['MMM-YY', 'City', 'Dateofjoining', 'LastWorkingDate'], dtype='object')
```

In [7]:
```python
df.shape
```

Out[7]:
```
(19104, 13)
```

In [8]:
```python
df.isna().sum()*100/len(df)
```

Out[8]:
```
MMM-YY                   0.000000
Driver_ID                0.000000
Age                      0.319305
Gender                   0.272194
City                     0.000000
Education_Level          0.000000
Income                   0.000000
Dateofjoining            0.000000
LastWorkingDate         91.541039
Joining Designation      0.000000
Grade                    0.000000
Total Business Value     0.000000
Quarterly Rating         0.000000
dtype: float64
```

checking for duplicated values

There are no duplicate entries.

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
0
```

changing date columns to DateTime format

In [10]:
```
df['MMM-YY'] = pd.to_datetime(df['MMM-YY'])
df['Dateofjoining'] = pd.to_datetime(df['Dateofjoining'])
df['LastWorkingDate'] = pd.to_datetime(df['LastWorkingDate'])
```

In [15]:
```
cont = df[['Age', 'Gender']].values

from sklearn.impute import KNNImputer

imputer = KNNImputer(missing_values=np.nan, n_neighbors=3)
imputer = imputer.fit(cont)
cont = imputer.transform(cont).astype('int')
cont[:5]
```

Out[15]:
```
array([[28,  0],
       [28,  0],
       [28,  0],
       [31,  0],
       [31,  0]])
```

In [16]:
```
df[['Age', 'Gender']] = cont
```

Data aggregation

In [ ]:
```
# grouping based on Driver ID
agg_df = df.groupby(['Driver_ID']).agg({'MMM-YY':'last','Age':'first','Gender':'first','City':'f
        'Education_Level':'first','Income':['first','last','mean'],'Dateofjoining':'first','LastW
         ,'Joining Designation':'first','Grade':'first','Total Business Value':'sum',
                                        'Quarterly Rating':['first','last','mean']})
df2 = agg_df.reset_index()
```

In [110…
```
df2.columns = ['_'.join(col) for col in df2.columns.values]
```

In [111…
```
df2.columns
```

Out[111]:
```
Index(['Driver_ID_', 'MMM-YY_last', 'Age_first', 'Gender_first', 'City_first',
       'Education_Level_first', 'Income_first', 'Income_last', 'Income_mean',
       'Dateofjoining_first', 'LastWorkingDate_last', 'LastWorkingDate_any',
       'Joining Designation_first', 'Grade_first', 'Total Business Value_sum',
       'Quarterly Rating_first', 'Quarterly Rating_last',
       'Quarterly Rating_mean'],
      dtype='object')
```

In [112…
```
df2['Churn'] = df2['LastWorkingDate_any']
df2['Churn'] = df2['Churn'].astype('int')
```

In [113…
```
df2['RateIncrease'] = np.where((df2['Quarterly Rating_last'] > df2['Quarterly Rating_first']),1,(
```

In [114…
```
df2['Income_Increase'] = np.where((df2['Income_last'] > df2['Income_first']),1,0)
```

In [115…
```
df2['LastDate'] = np.where(df2['LastWorkingDate_last'].notnull(),
                           df2['LastWorkingDate_last'].dt.strftime('%Y-%m-%d'),
                           df2['MMM-YY_last'].dt.strftime('%Y-%m-%d'))
# replacing NaT values in last Working date column with Last reporting date 'MMM-YY'
```

In [116…
```
df2[['MMM-YY_last','LastWorkingDate_last','LastDate']].head()
```

Out[116]:

	MMM-YY_last	LastWorkingDate_last	LastDate
0	2019-03-01	2019-03-11	2019-03-11
1	2020-12-01	NaT	2020-12-01
2	2020-04-01	2020-04-27	2020-04-27
3	2019-03-01	2019-03-07	2019-03-07
4	2020-12-01	NaT	2020-12-01

In [117…
```
df2['tenure'] = pd.to_datetime(df2['LastDate']) - df2['Dateofjoining_first']
```

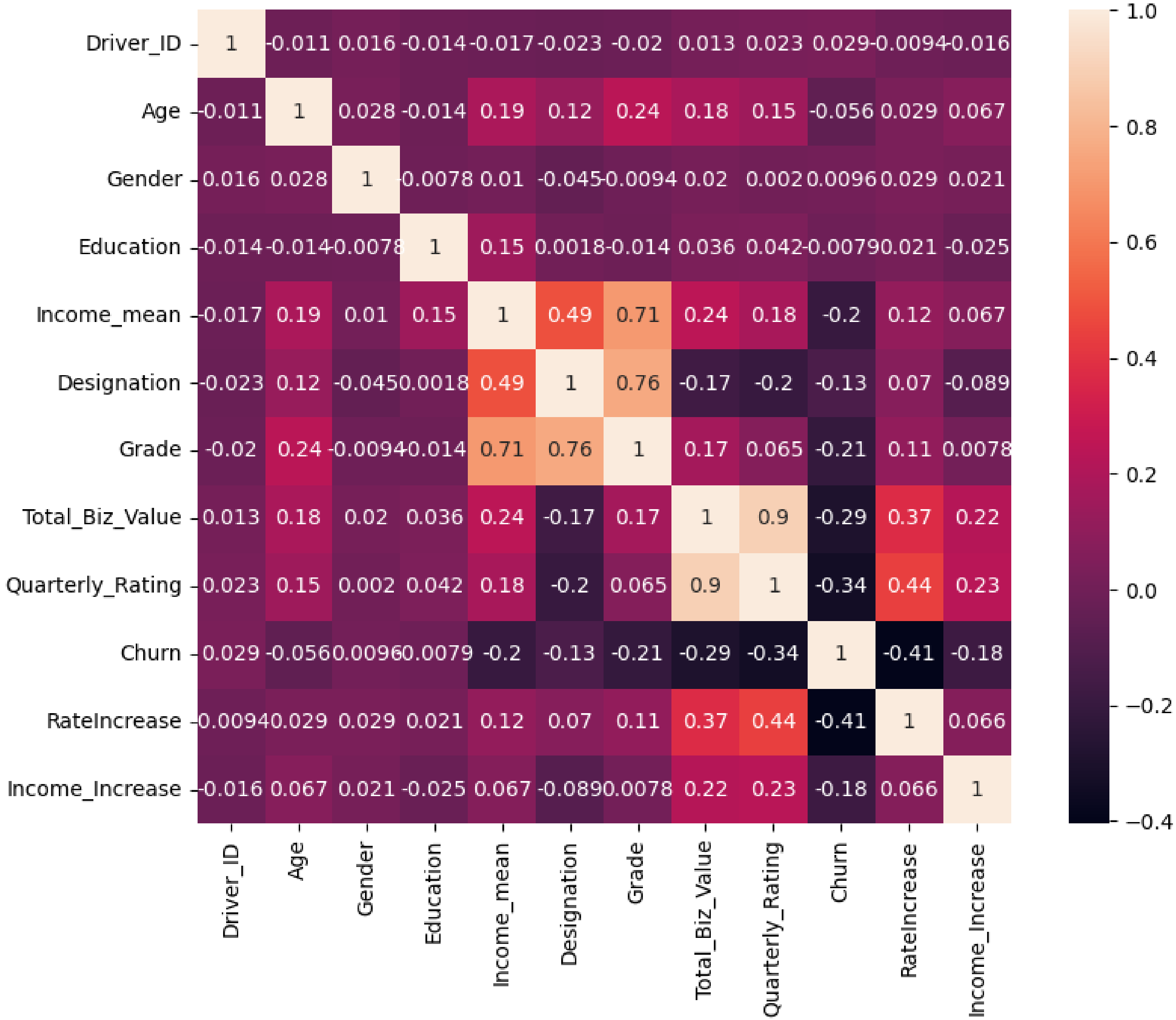
In [118…
```
df2 = df2.drop(['Income_first','Income_last','Quarterly Rating_first','Quarterly Rating_last','La
        ,'LastWorkingDate_any','MMM-YY_last','Dateofjoining_first'],axis=1)
```

In [119…
```
df2.rename(columns = {'Driver_ID_':'Driver_ID','Age_first':'Age','Gender_first':'Gender','City_fi
                      'Education_Level_first':'Education','Joining Designation_first':'Designati
                      'Total Business Value_sum':'Total_Biz_Value','Quarterly Rating_mean':'Quar
                     }, inplace=True)
```

In [80]:
```
# Spearman's Rank Correlation Coefficient
plt.figure(figsize=(10,7))
sns.heatmap(df2.corr(method='spearman'), square=True,annot=True)
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_7664\1015870712.py:3: FutureWarning: The default val
ue of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fals
e. Select only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(df2.corr(method='spearman'), square=True,annot=True)
```

Out[80]: <AxesSubplot: >



The variables 'Quarterly Rating' and 'Total Business Value' are highly correlated (0.9), we will ignore one of them - Quarterly rating as we are also considering it in RateIncrease variable.

Grade and Designation are also correlated (0.76), and we would ignore Grade variable.

In [121…
```
df2.drop(['Grade','Quarterly_Rating','Driver_ID'],axis=1,inplace=True) # driver ID is not relate
```

In [122…
```
df2['LastDate'] = pd.to_datetime(df2['LastDate']).dt.year
```

In [125…
```
df2.describe() # numerical features
```

Out[125]:

	Age	Gender	Education	Income_mean	Designation	Total_Biz_Value	Churn	RateIncreas
count	2381.000000	2381.000000	2381.00000	2381.000000	2381.000000	2.381000e+03	2381.000000	2381.00000
mean	33.089038	0.410752	1.00756	59232.460484	1.820244	4.586742e+06	0.678706	0.15035
std	5.839201	0.492074	0.81629	28298.214012	0.841433	9.127115e+06	0.467071	0.35749
min	21.000000	0.000000	0.00000	10747.000000	1.000000	-1.385530e+06	0.000000	0.00000
25%	29.000000	0.000000	0.00000	39104.000000	1.000000	0.000000e+00	0.000000	0.00000
50%	33.000000	0.000000	1.00000	55285.000000	2.000000	8.176800e+05	1.000000	0.00000
75%	37.000000	1.000000	2.00000	75835.000000	2.000000	4.173650e+06	1.000000	0.00000
max	58.000000	1.000000	2.00000	188418.000000	5.000000	9.533106e+07	1.000000	1.00000

In [126…

```
df2['tenure'] = df2['tenure'].dt.days
df2['tenure'].describe()
```

Out[126]:

```
count    2381.000000
mean      424.540109
std       564.404943
min       -27.000000
25%        91.000000
50%       180.000000
75%       467.000000
max      2801.000000
Name: tenure, dtype: float64
```

In [127…

```
df2['tenure'] = df2['tenure'].clip(lower=0)
df2['tenure'].describe()
```

Out[127]:

```
count    2381.000000
mean      424.852163
std       564.165833
min         0.000000
25%        91.000000
50%       180.000000
75%       467.000000
max      2801.000000
Name: tenure, dtype: float64
```

In [128…

```
df2['City'].describe()
```

Out[128]:

```
count      2381
unique       29
top         C20
freq        152
Name: City, dtype: object
```

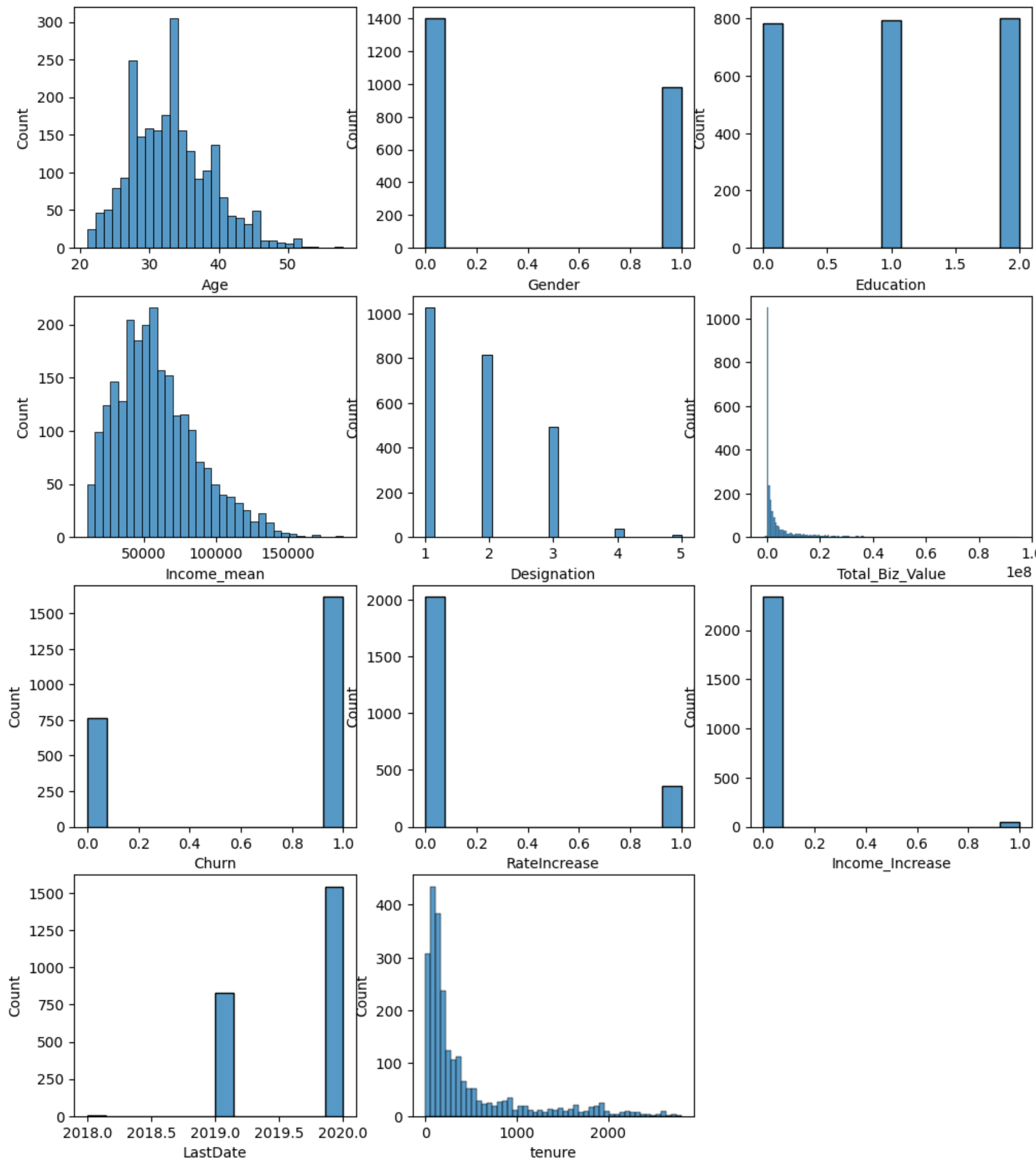
City has 29 unique city codes, more common being C20.

In [131…

```
continuous_cols = df2.columns[(df2.dtypes == 'int64')|(df2.dtypes == 'float64')|(df2.dtypes == ':
continuous_cols
```

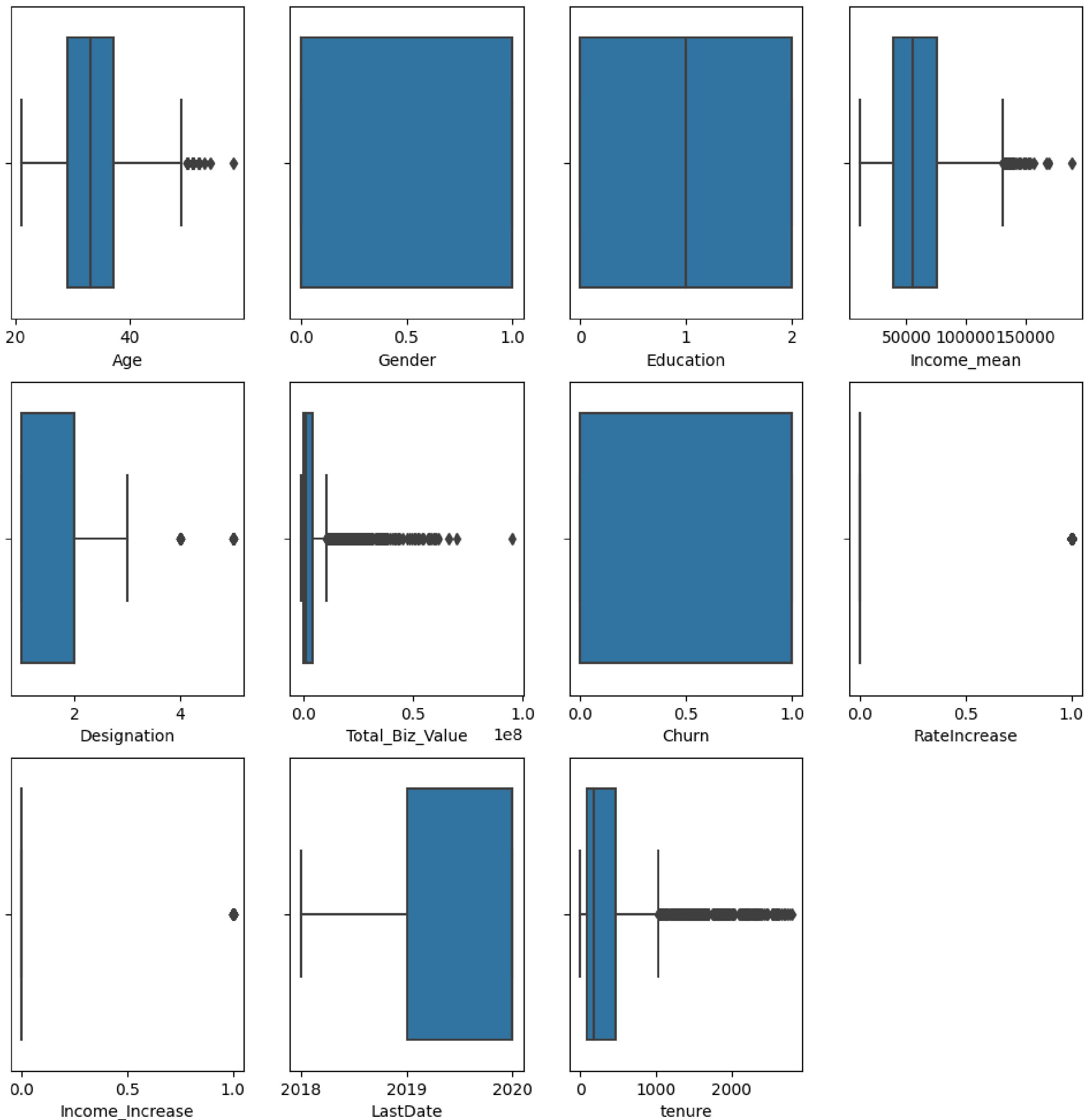
```
Out[131]: Index(['Age', 'Gender', 'Education', 'Income_mean', 'Designation',
       'Total_Biz_Value', 'Churn', 'RateIncrease', 'Income_Increase',
       'LastDate', 'tenure'],
      dtype='object')
```

```
In [133…
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(14)
n = len(continuous_cols)

for i in range(n):
    plt.subplot(4,(n//4)+1,i+1)
    sns.histplot(data=df2, x=continuous_cols[i])
plt.show()
```



In [135…

```
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(12)
n = len(continuous_cols)

for i in range(n):
    plt.subplot(3,4,i+1)
    sns.boxplot(data=df2, x=continuous_cols[i])
plt.show()
```



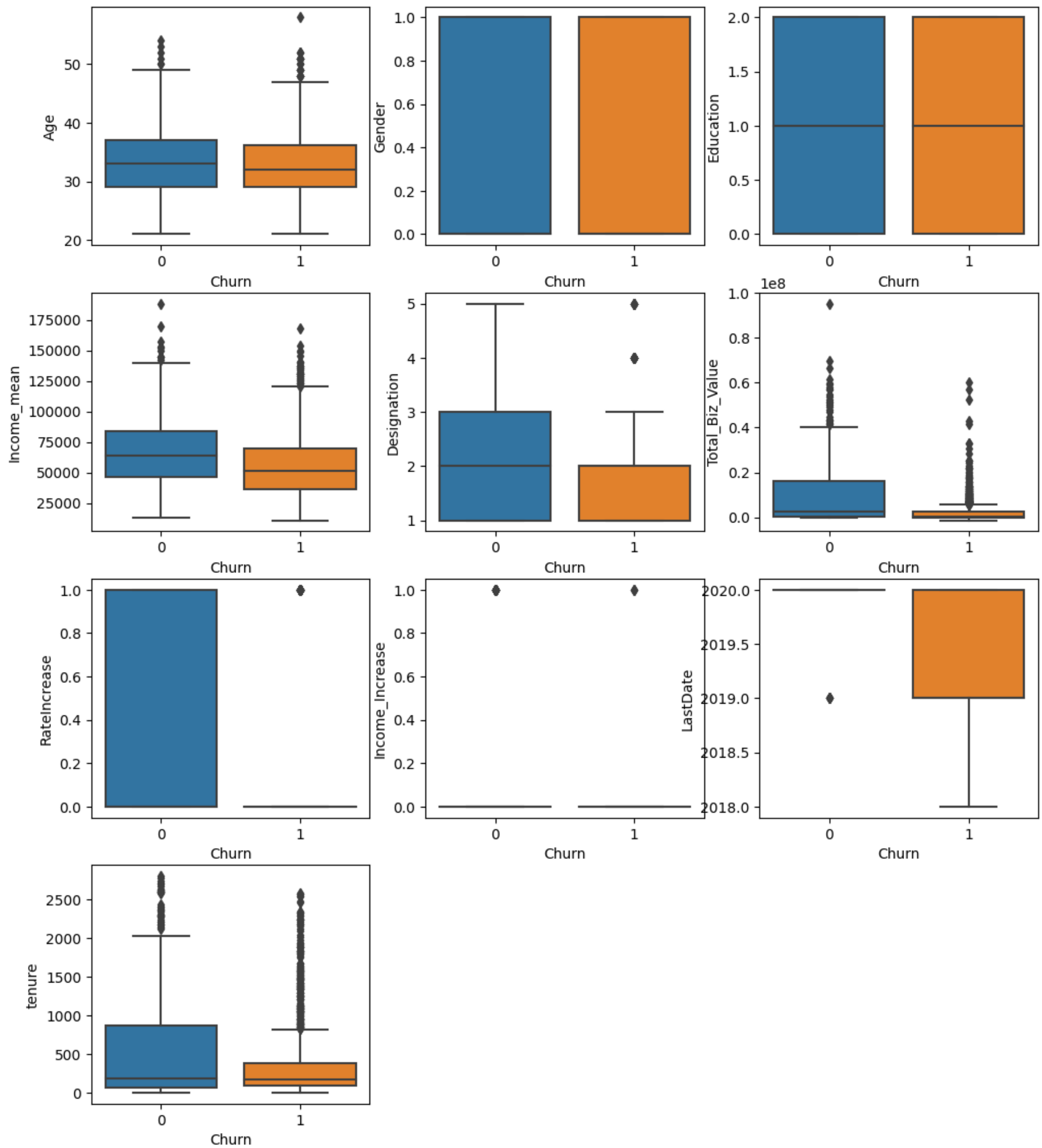
There are many outliers but bagging and boosting algorithms do not assume normality.

In [143…

```
continuous_cols = continuous_cols.drop(labels = ['Churn'])
continuous_cols
```

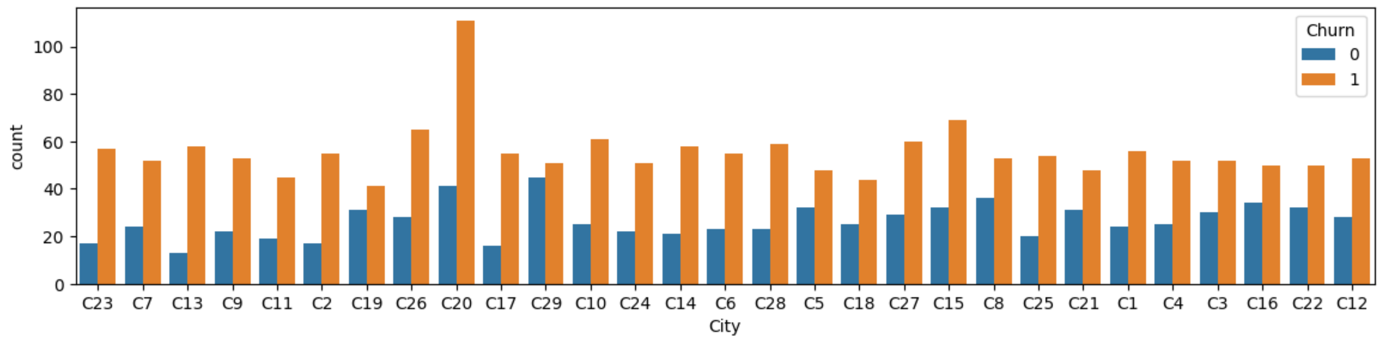
```
Out[143]: Index(['Age', 'Gender', 'Education', 'Income_mean', 'Designation',
                 'Total_Biz_Value', 'RateIncrease', 'Income_Increase', 'LastDate',
                 'tenure'],
                dtype='object')
```

```
In [146…
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(14)
n = len(continuous_cols)

for i in range(n):
    plt.subplot(4,3,i+1)
    sns.boxplot(data=df2, y=continuous_cols[i],x='Churn')
plt.show()
```



In [149…

```
f = plt.figure()
f.set_figwidth(14)
f.set_figheight(3)
sns.countplot(data=df2, x='City',hue='Churn')
plt.show()
```



Encoding categorical variables

In [155…

```
X = df2.drop(['Churn'],axis=1)
Y = np.array(df2['Churn']).reshape(-1,1)
print(X.shape, Y.shape)
```

```
(2381, 11) (2381, 1)
```

In [156…

```
# from sklearn.preprocessing import LabelEncoder
# X['City'] = X['City'].apply(LabelEncoder().fit_transform)
# .join(df.select_dtypes(include=['number']))
X['City'] = X['City'].apply(lambda x:x[1:])
X.head()
```

Out[156]:

	Age	Gender	City	Education	Income_mean	Designation	Total_Biz_Value	RateIncrease	Income_Increase	LastI
0	28	0	23	2	57387.0	1	1715580	0	0	
1	31	0	7	2	67016.0	2	0	0	0	
2	43	0	13	2	65603.0	2	350000	0	0	
3	29	0	9	0	46368.0	1	120360	0	0	
4	31	1	11	1	78728.0	3	1265000	1	0	

Splitting data into training and testing dataset

In [177…

```
from sklearn.model_selection import train_test_split
# Create training and test split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.05, random_state=4) #5% te
```

Class imbalance treatment

In [178…

```
from imblearn.over_sampling import SMOTE
from collections import Counter

smt = SMOTE()
X_sm, y_sm = smt.fit_resample(X_train, y_train)

print('Resampled dataset shape {}'.format(Counter(y_sm)))
```

```
Resampled dataset shape Counter({1: 1533, 0: 1533})
```

Column Standarization

In [179…

```
# Mean centering and Variance scaling (Standard Scaling)
from sklearn.preprocessing import StandardScaler
X_columns = X_sm.columns
scaler = StandardScaler()
X_sm = scaler.fit_transform(X_sm)
X_test = scaler.transform(X_test)
X_sm = pd.DataFrame(X_sm, columns=X_columns)
X_sm.head()
```

Out[179]:

	Age	Gender	City	Education	Income_mean	Designation	Total_Biz_Value	RateIncrease	Income_In
0	0.337000	-0.730594	-1.678926	1.344522	-0.419114	-0.956998	-0.556362	-0.452455	-0.1
1	-1.630738	1.368750	-1.428471	-1.169078	-0.974593	-0.956998	-0.499150	-0.452455	-0.1
2	-0.378541	-0.730594	-1.052790	0.087722	-0.056401	0.277397	-0.556362	-0.452455	-0.1
3	-0.736312	-0.730594	0.449934	1.344522	2.074695	-0.956998	0.416949	-0.452455	-0.1
4	-0.378541	1.368750	0.950842	-1.169078	0.466384	1.511792	-0.556362	-0.452455	-0.1

Ensemble technique - Random Forest Bagging algorithm

In [261…

```
# Defining Parametes
params = {
          'n_estimators' : [100,200,300,400],
          'max_depth' : [5,6,7,8],
          'criterion' : ['gini'],
          'bootstrap' : [True],
          'max_features' : [8,9,10]
         }
```

In [262…

```
from sklearn.model_selection import GridSearchCV

tuning_function = GridSearchCV(estimator = RandomForestClassifier(),
                               param_grid = params,
                               scoring = 'accuracy',
                               cv = 3,
                               n_jobs=-1
                               )
# Now we will fit all combinations, this will take some time to run. (5-6 mins)
tuning_function.fit(X_sm, y_sm)

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

```
{'bootstrap': True, 'criterion': 'gini', 'max_depth': 8, 'max_features': 8, 'n_estimators': 300}
0.8561643835616438
```

In [264…

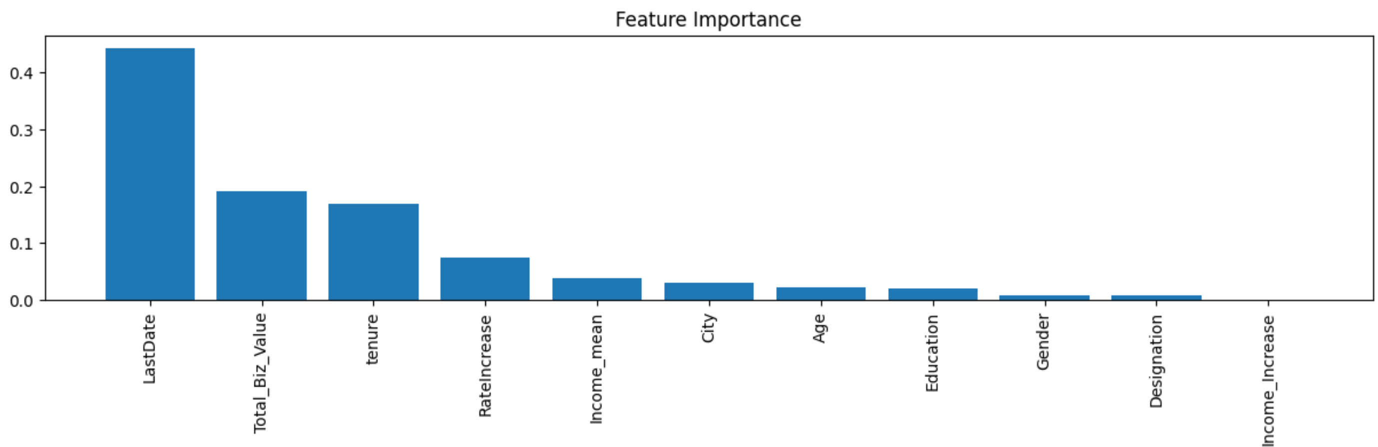
```
tree_clf = RandomForestClassifier(random_state=7, max_depth=8, n_estimators=300, max_features=8)
kfold = KFold(n_splits=3)
cv_acc_results = cross_validate(tree_clf, X_sm, y_sm, cv = kfold, scoring = 'accuracy', return_tr
```

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

```
K-Fold Accuracy Mean: Train: 92.31898238747553 Validation: 77.56033920417482
K-Fold Accuracy Std: Train: 0.4989255884141664 Validation: 4.551976698339208
```

In [265…

```
# Feature importance
tree_clf = RandomForestClassifier(random_state=7, max_depth=8, n_estimators=300, max_features=8)
tree_clf.fit(X_sm, y_sm)
importances = tree_clf.feature_importances_
indices = np.argsort(importances)[::-1] # Sort feature importances in descending order
names = [X_sm.columns[i] for i in indices] # Rearrange feature names so they match the sorted fe
plt.figure(figsize=(15, 3)) # Create plot
plt.title("Feature Importance") # Create plot title
plt.bar(range(X_sm.shape[1]), importances[indices]) # Add bars
plt.xticks(range(X_sm.shape[1]), names, rotation=90) # Add feature names as x-axis labels
plt.show() # Show plot
```



According to the RF bagging algorithm, the churn outcome was most affected by the Last working date, and then the other important features were Total Business value, tenure duration and increase in quarterly ratings.

In [266…

```
y_pred = tree_clf.predict(X_test)
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:450: Us
erWarning: X does not have valid feature names, but RandomForestClassifier was fitted with featu
re names
  warnings.warn(
```

In [267…

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, plot_confusion_matr

testscore = accuracy_score(y_test,y_pred)
print('Test accuracy: ',testscore)

cm = confusion_matrix(y_test, y_pred)

cm_df = pd.DataFrame(cm,index = np.unique(y_test), columns = np.unique(y_test) )

cm_df.head()
```

```
Test accuracy:  0.8833333333333333
```

Out[267]:

	0	1
0	33	4
1	10	73

In [268…

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

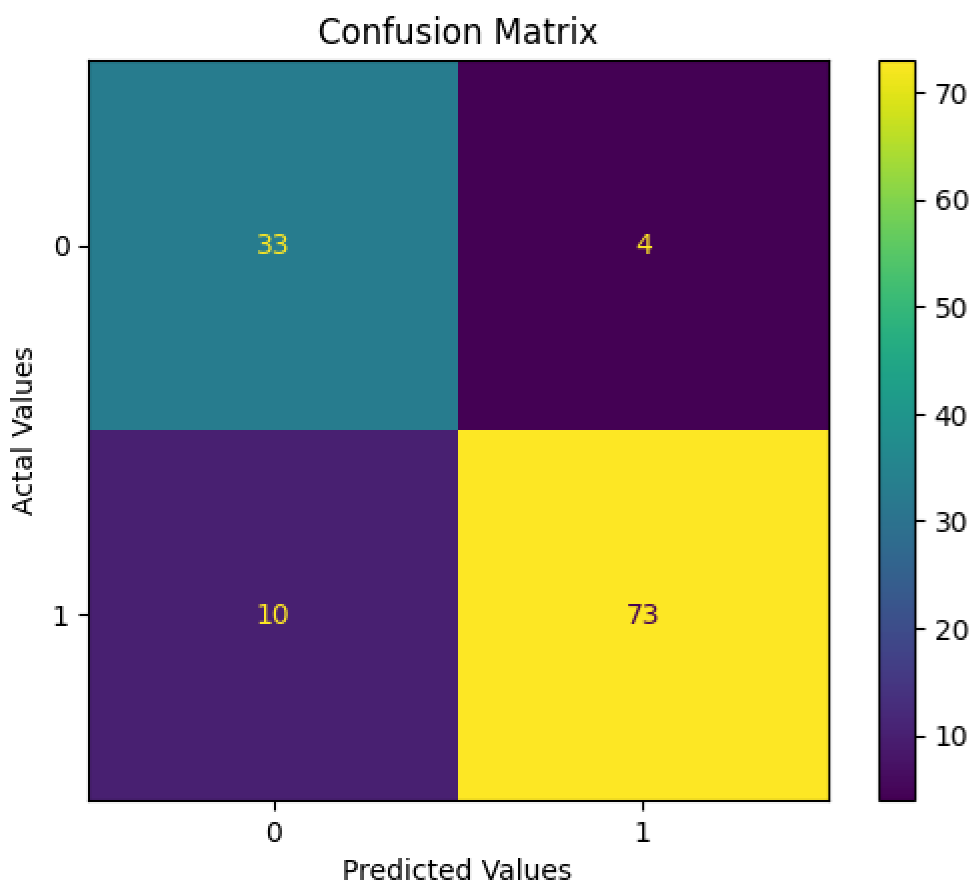
```
              precision    recall  f1-score   support

           0       0.77      0.89      0.82        37
           1       0.95      0.88      0.91        83

    accuracy                           0.88       120
   macro avg       0.86      0.89      0.87       120
weighted avg       0.89      0.88      0.89       120
```

In [269…

```
#Plotting the confusion matrix
plt.figure(figsize=(1,1))
plot_confusion_matrix(tree_clf,X_test,y_test)
#sns.heatmap(cm_df, annot=True,cmap='coolwarm')
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\deprecati
on.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_
matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: Confusion
MatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:450: Us
erWarning: X does not have valid feature names, but RandomForestClassifier was fitted with featu
re names
  warnings.warn(
```

<Figure size 100x100 with 0 Axes>



In [270…

```
from sklearn.metrics import f1_score
print("Precision score is :",precision_score(y_test,y_pred))
```

```
print("Recall score is :",recall_score(y_test,y_pred))
print("F1 score is :",f1_score(y_test,y_pred))
```

```
Precision score is : 0.948051948051948
Recall score is : 0.8795180722891566
F1 score is : 0.9125
```

ROC (Receiver Operating Characteristic curve) and AUC (Area Under Curve)

In [271…

```
y_proba = tree_clf.predict_proba(X_test)
y_proba.shape, y_test.shape
```

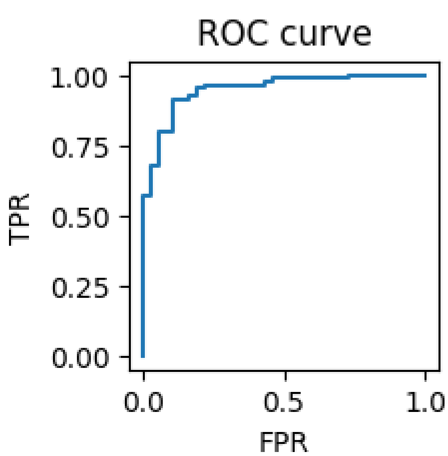
```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

Out[271]:

```
((120, 2), (120, 1))
```

In [272…

```
from sklearn.metrics import roc_curve, roc_auc_score
fpr, tpr, thr = roc_curve(y_test, y_proba[:,1])
plt.figure(figsize=(2,2))
plt.plot(fpr,tpr)
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



There are more true positives than False positives, hence the ROC curve has more area under curve than 0.5.

In [273…

```
roc_auc_score(y_test,y_proba[:,1])
```

Out[273]:

```
0.9488765874308043
```

0.95 is very good area under curve score.

Ensemble Boosting algorithm - XGBoost

In [235…

```
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import StratifiedKFold

params = {
        'learning_rate': [0.1, 0.5, 0.8],
        'subsample': [0.6, 0.8, 1.0],
```

```
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5]
        }
xgb = XGBClassifier(n_estimators=100, objective='multi:softmax', num_class=20, silent=True)

folds = 3

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

random_search = RandomizedSearchCV(xgb, param_distributions=params, n_iter=10, scoring='accuracy
                                   cv=skf.split(X_sm,y_sm), verbose=3, random_state=1001 )


# start = dt.datetime.now()
random_search.fit(X_sm, y_sm)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[23:15:34] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-0fc7796c
793e6356f-1/xgboost/xgboost-ci-windows/src/learner.cc:767:
Parameters: { "silent" } are not used.
```

Out[235]:

▸ **RandomizedSearchCV**

▸ **estimator: XGBClassifier**

▸ XGBClassifier

In [236…

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

```
 Best hyperparameters:
{'subsample': 1.0, 'max_depth': 4, 'learning_rate': 0.5, 'colsample_bytree': 0.8}
```

In [237…

```
best_xgb = XGBClassifier(n_estimators=100, objective='multi:softmax', num_class=20,
                         subsample=1.0, max_depth=4, learning_rate=0.5, colsample_bytree=0.8)
best_xgb.fit(X_sm, y_sm)
```

Out[237]:

▾ XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.5, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [255…

```
y_pred = best_xgb.predict(X_test)
y_pred_train = best_xgb.predict(X_sm)

testscore = accuracy_score(y_test,y_pred)
trscore = accuracy_score(y_sm,y_pred_train)
print('Train accuracy: ',trscore)
print('Test accuracy: ',testscore)
cm = confusion_matrix(y_test, y_pred)
```

```
cm_df = pd.DataFrame(cm,index = np.unique(y_test), columns = np.unique(y_test) )

cm_df.head()
```

```
Train accuracy:  0.9960861056751468
Test accuracy:  0.8666666666666667
```

Out[255]:

	0	1
0	30	7
1	9	74

True negatives - 30 (drivers who stayed as predicted)

False positives - 7 (drivers who stayed but were predicted to churn)

False negatives - 9 (drivers who churned but were predicted to stay) - need the most attention as can cause huge financial losses

True positives - 74 (drivers who churned as predicted)

In [250…

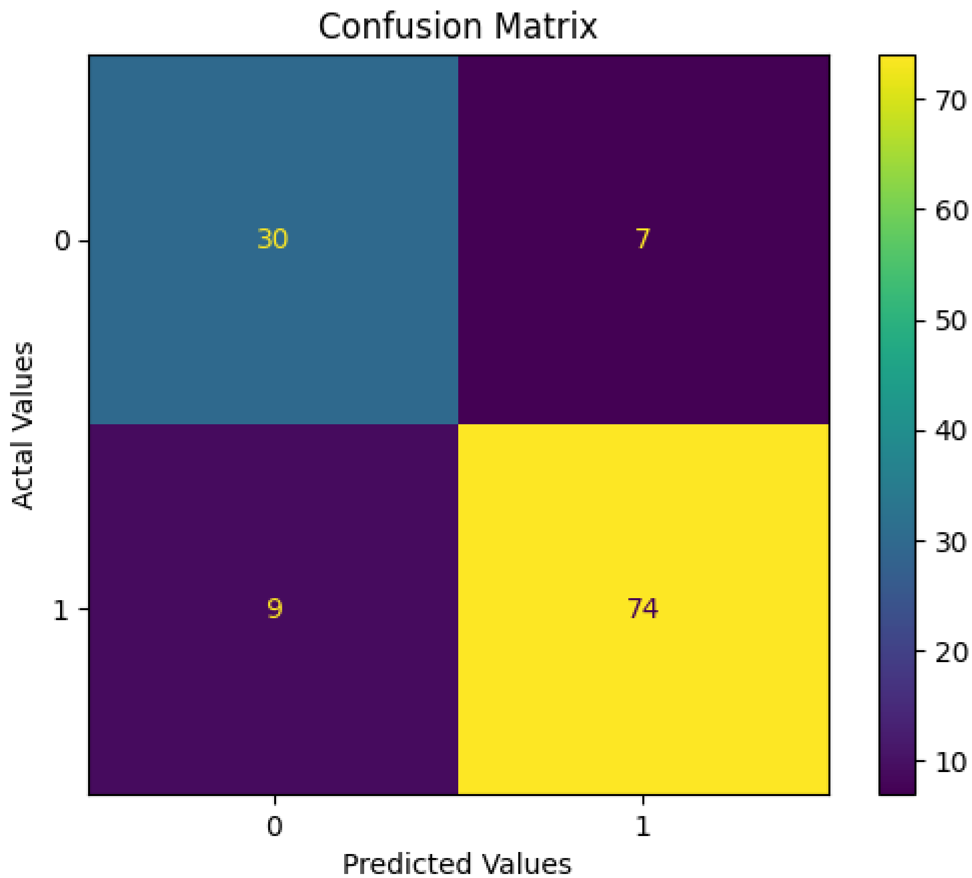
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.81      0.79        37
           1       0.91      0.89      0.90        83

    accuracy                           0.87       120
   macro avg       0.84      0.85      0.85       120
weighted avg       0.87      0.87      0.87       120
```

In [239…

```
#Plotting the confusion matrix
plt.figure(figsize=(1,1))
plot_confusion_matrix(best_xgb,X_test,y_test)
#sns.heatmap(cm_df, annot=True,cmap='coolwarm')
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\deprecati
on.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_
matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: Confusion
MatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
<Figure size 100x100 with 0 Axes>
```



In [240…
```
from sklearn.metrics import f1_score
print("Precision score is :",precision_score(y_test,y_pred))
print("Recall score is :",recall_score(y_test,y_pred))
print("F1 score is :",f1_score(y_test,y_pred))
```

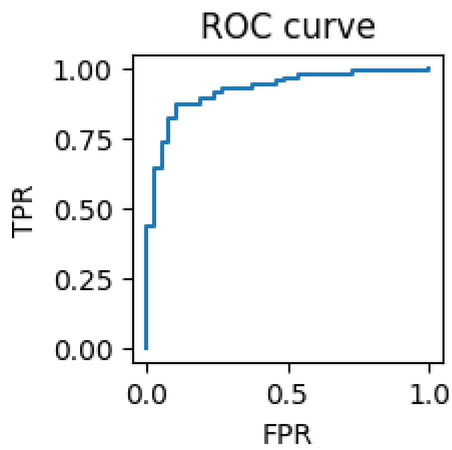
```
Precision score is : 0.9135802469135802
Recall score is : 0.891566265060241
F1 score is : 0.9024390243902438
```

ROC (Receiver Operating Characteristic curve) and AUC (Area Under Curve)

In [241…
```
# from sklearn.linear_model import
y_proba = best_xgb.predict_proba(X_test)
y_proba.shape, y_test.shape
```

Out[241]:
```
((120, 20), (120, 1))
```

In [244…
```
from sklearn.metrics import roc_curve, roc_auc_score
fpr, tpr, thr = roc_curve(y_test, y_proba[:,1])
plt.figure(figsize=(2,2))
plt.plot(fpr,tpr)
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



There are more true positives than False positives, hence the ROC curve has more area under curve than 0.5.

In [243…

```
roc_auc_score(y_test,y_proba[:,1])
```

Out[243]:

```
0.9202214262455226
```

0.92 is a good area under curve.

According to the RF bagging algorithm, train accuracy:92.3% , test accuracy: 88.3%. The churn outcome was most affected by the Last working date, and then the other important features were Total Business value, tenure duration and increase in quarterly ratings. Precision score was 0.95. According to XGBoost algorithm: training accuracy:99.6% , test accuracy 86.67%. Precision is 0.91 which is good as the number of false positives (7) are low, the drivers who stayed but were predicted to churn. Recall-score: 0.89. Recall is high because of low number of False negatives.

Drivers who left were those who did not have any increase in their ratings so maybe they can be encouraged to improve their driving skills and ratings by being given tips on communication with customers and safe driving tips to increase their ratings.

Since we only have data from 2018, 2019 and 2020, out of which 2020 was the time of pandemic and cannot be used for general analysis, we need to collect more data in order to improve our analysis and reduce errors.

In [ ]: