Business Problem

Financial institutions incur significant losses due to the default of vehicle loans. This has led to the tightening up of vehicle loan underwriting and an increase in vehicle loan rejection rates. The need for a better credit risk scoring model is also raised by these institutions. This warrants a study to estimate the determinants of vehicle loan default.

There is one dataset with data that has 41 attributes.

You are required to determine and examine factors that affect the ratio of vehicle loan defaulters. Also, use the findings to create a model to predict the potential defaulters.

Project Overview

- The objective of the problem is to accurately predict the probability of loanee/borrower defaulting on a vehicle loan in the first EMI (Equated Monthly Instalments) on the due date.
- Performed EDA to understand the relation of target variable with the other features.
- Statistical Analysis techniques like ANOVA for numerical and Chi-square for the categorical variables were performed to find the significance of the features with respect to the target.
- Base Models were built in Logistic Regression, Random Forest, KNN and LightGBM with Kfold cross-validation.
- Created new features like age at the time of disbursement, disbursement month, etc. The dataset represents Vehicle Loan - - Default of L&T financial institution.

Dataset Description

The dataset represents Vehicle Loan Default of L&T financial institution. There are 2,33,134 records and 42 columns Following Information regarding the loan and loanee are provided in the datasets:

- Loanee Information (Demographic data like age, Identity proof etc.)
- Loan Information (Disbursal details, loan to value ratio etc.)
- Bureau data & history (Bureau score, number of active accounts, the status of other loans, credit history etc.)

Importing Libaries

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score

from sklearn.metrics import confusion_matrix,accuracy_score,classification_report,roc_auc_score,roc_curve
```

In [2]:
```python
import seaborn as sns
```

Import the data

In [3]:
```python
df = pd.read_csv(r"C:\Users\BAPS\Downloads\BankigCapstoneProjectDataSet\Dataset\BakingDataset.csv")
```

In [4]:
```python
df.head()
```

Out[4]:

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Current_pincode_ID	Date.of.Birth	Employment.Type	...	SEC.S
0	420825	50578	58400	89.55	67	22807	45	1441	01/01/1984	Salaried	...	
1	417566	53278	61360	89.63	67	22807	45	1497	24/08/1985	Self employed	...	
2	539055	52378	60300	88.39	67	22807	45	1495	09/12/1977	Self employed	...	
3	529269	46349	61500	76.42	67	22807	45	1502	01/06/1988	Salaried	...	
4	563215	43594	78256	57.50	67	22744	86	1499	14/07/1994	Self employed	...	

5 rows × 41 columns

In [5]:
```
print("The dataset has %d rows"%df.shape[0])
print("The dataset has %d columns"%df.shape[1])
```

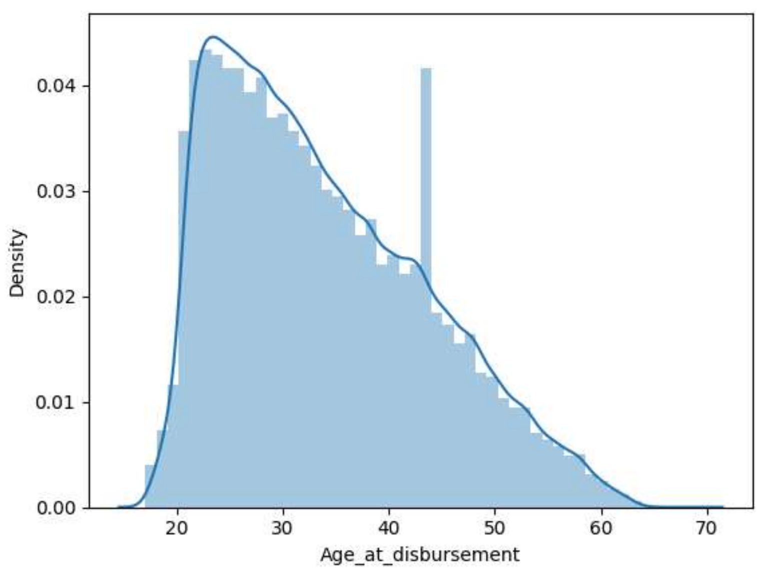
```
The dataset has 233154 rows
The dataset has 41 columns
```

In [6]:
```
def cns_desc(x):
    if x<300:
        return 0
    elif (x>=300) and (x<=350):
        return 1
    elif (x>350) and (x<=570):
        return 2
    elif (x>570) and (x<=630):
        return 3
    elif (x>630) and (x<=705):
        return 4
    else:
        return 5
```

In [7]:
```
df["Date.of.Birth"]=pd.to_datetime(df["Date.of.Birth"])

df["DisbursalDate"]=pd.to_datetime(df["DisbursalDate"])
df["DisbursalDate_Month"]=df["DisbursalDate"].dt.month

df["Age_at_disbursement"]=((df['DisbursalDate']-df["Date.of.Birth"])/365).apply(lambda x:x.days)

df["Employment.Type"]=df["Employment.Type"].map({'Salaried':0,"Self employed":1,np.nan:2})

df["PERFORM_CNS.SCORE.DESCRIPTION"]=df["PERFORM_CNS.SCORE"].apply(cns_desc)

df["AVERAGE.ACCT.AGE"]=df["AVERAGE.ACCT.AGE"].apply(lambda x:(int((x.split("y"))[0])*12)+(int((x.split())[1].split("m")[

df["CREDIT.HISTORY.LENGTH"]=df["CREDIT.HISTORY.LENGTH"].apply(lambda x:(int((x.split("y"))[0])*12)+(int((x.split())[1].s
```

In [8]:
```
sns.distplot(df['Age_at_disbursement'])
```

Out[8]: <Axes: xlabel='Age_at_disbursement', ylabel='Density'>



In [9]:
```
df['NO.OF.ACCTS']=df['PRI.NO.OF.ACCTS']+df['SEC.NO.OF.ACCTS']
df['ACTIVE.ACCTS']=df['PRI.ACTIVE.ACCTS']+df['SEC.ACTIVE.ACCTS']
df['OVERDUE.ACCTS']=df['PRI.OVERDUE.ACCTS']+df['SEC.OVERDUE.ACCTS']
df['CURRENT.BALANCE']=df['PRI.CURRENT.BALANCE']+df['SEC.CURRENT.BALANCE']
df['SANCTIONED.AMOUNT']=df['PRI.SANCTIONED.AMOUNT']+df['SEC.SANCTIONED.AMOUNT']
df['DISBURSED.AMOUNT']=df['PRI.DISBURSED.AMOUNT']+df['SEC.DISBURSED.AMOUNT']
df['INSTAL.AMT']=df['PRIMARY.INSTAL.AMT']+df['SEC.INSTAL.AMT']

df=df.drop(['PRI.NO.OF.ACCTS','SEC.NO.OF.ACCTS','PRI.ACTIVE.ACCTS','SEC.ACTIVE.ACCTS','PRI.OVERDUE.ACCTS','SEC.OVERDUE.AC
```

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

Out[10]:
```
UniqueID                               233154
disbursed_amount                        24565
asset_cost                              46252
ltv                                      6579
branch_id                                  82
supplier_id                              2953
manufacturer_id                            11
Current_pincode_ID                       6698
Date.of.Birth                           15433
Employment.Type                             3
DisbursalDate                              84
State_ID                                   22
Employee_code_ID                         3270
MobileNo_Avl_Flag                           1
Aadhar_flag                                 2
PAN_flag                                    2
VoterID_flag                                2
Driving_flag                                2
Passport_flag                               2
PERFORM_CNS.SCORE                         573
PERFORM_CNS.SCORE.DESCRIPTION               6
NEW.ACCTS.IN.LAST.SIX.MONTHS               26
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS        14
AVERAGE.ACCT.AGE                          192
CREDIT.HISTORY.LENGTH                     294
NO.OF_INQUIRIES                            25
loan_default                                2
DisbursalDate_Month                        12
Age_at_disbursement                        49
NO.OF.ACCTS                               108
ACTIVE.ACCTS                               41
OVERDUE.ACCTS                              22
CURRENT.BALANCE                         72483
SANCTIONED.AMOUNT                       45367
DISBURSED.AMOUNT                        48958
INSTAL.AMT                              28540
dtype: int64
```

In [11]:
```
df.columns
```

Out[11]:
```
Index(['UniqueID', 'disbursed_amount', 'asset_cost', 'ltv', 'branch_id',
       'supplier_id', 'manufacturer_id', 'Current_pincode_ID', 'Date.of.Birth',
       'Employment.Type', 'DisbursalDate', 'State_ID', 'Employee_code_ID',
       'MobileNo_Avl_Flag', 'Aadhar_flag', 'PAN_flag', 'VoterID_flag',
       'Driving_flag', 'Passport_flag', 'PERFORM_CNS.SCORE',
       'PERFORM_CNS.SCORE.DESCRIPTION', 'NEW.ACCTS.IN.LAST.SIX.MONTHS',
       'DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS', 'AVERAGE.ACCT.AGE',
       'CREDIT.HISTORY.LENGTH', 'NO.OF_INQUIRIES', 'loan_default',
       'DisbursalDate_Month', 'Age_at_disbursement', 'NO.OF.ACCTS',
       'ACTIVE.ACCTS', 'OVERDUE.ACCTS', 'CURRENT.BALANCE', 'SANCTIONED.AMOUNT',
       'DISBURSED.AMOUNT', 'INSTAL.AMT'],
      dtype='object')
```

In [12]:
```
df=df.drop(columns=['Date.of.Birth',"DisbursalDate"])
```

In [13]:
```
cols=df.columns.to_list()
```

In [14]:
```
cat_cols=['UniqueID','branch_id','supplier_id', 'manufacturer_id', 'Current_pincode_ID','Employment.Type', 'State_ID',
          'Employee_code_ID','MobileNo_Avl_Flag', 'Aadhar_flag', 'PAN_flag', 'VoterID_flag','Driving_flag', 'Passport_fl
       'PERFORM_CNS.SCORE.DESCRIPTION','DisbursalDate_Month']
```

In [15]:
```
target_col="loan_default"
```

In [16]:
```
num_cols=['disbursed_amount','asset_cost','ltv','PERFORM_CNS.SCORE','NEW.ACCTS.IN.LAST.SIX.MONTHS',
          'DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS','AVERAGE.ACCT.AGE','CREDIT.HISTORY.LENGTH','NO.OF_INQUIRIES',
          'Age_at_disbursement','NO.OF.ACCTS','ACTIVE.ACCTS','OVERDUE.ACCTS','CURRENT.BALANCE','SANCTIONED.AMOUNT',
             'DISBURSED.AMOUNT','INSTAL.AMT']
```

In [17]:
```
df[cat_cols].nunique()
```

Out[17]:
```
UniqueID                       233154
branch_id                          82
supplier_id                      2953
manufacturer_id                    11
Current_pincode_ID               6698
Employment.Type                     3
State_ID                           22
Employee_code_ID                 3270
MobileNo_Avl_Flag                   1
Aadhar_flag                         2
PAN_flag                            2
VoterID_flag                        2
Driving_flag                        2
Passport_flag                       2
PERFORM_CNS.SCORE.DESCRIPTION       6
DisbursalDate_Month                12
dtype: int64
```

```
As we see no of unique elements in some categorical columns is higher,we couldn't take dummies as it will cause curse of dimensionality.Let's drop them.
```

In [21]:
```
linear_models_df=df.copy()
```

In [22]:
```
linear_models_df=linear_models_df.drop(columns=['UniqueID','supplier_id','Current_pincode_ID','Employee_code_ID',"Mobile
```

In [24]:
```
cols_to_be_dummied=["branch_id","State_ID","manufacturer_id",'Employment.Type','DisbursalDate_Month','PERFORM_CNS.SCORE.
```

In [25]:
```
linear_models_final_df=pd.get_dummies(data=linear_models_df,columns=cols_to_be_dummied,drop_first=True)
```

In [26]:
```
linear_models_df.corr()['loan_default']
```

Out[26]:
```
disbursed_amount                       0.077675
asset_cost                             0.014261
ltv                                    0.098208
branch_id                              0.030193
manufacturer_id                       -0.025039
Employment.Type                        0.024823
State_ID                               0.048075
Aadhar_flag                           -0.041593
PAN_flag                               0.002046
VoterID_flag                           0.043747
Driving_flag                          -0.005821
Passport_flag                         -0.007602
PERFORM_CNS.SCORE                     -0.057929
PERFORM_CNS.SCORE.DESCRIPTION         -0.067798
NEW.ACCTS.IN.LAST.SIX.MONTHS          -0.029400
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS    0.034462
AVERAGE.ACCT.AGE                      -0.024781
CREDIT.HISTORY.LENGTH                 -0.042126
NO.OF_INQUIRIES                        0.043678
loan_default                           1.000000
DisbursalDate_Month                    0.011411
Age_at_disbursement                   -0.036384
NO.OF.ACCTS                           -0.035963
ACTIVE.ACCTS                          -0.041511
OVERDUE.ACCTS                          0.039469
CURRENT.BALANCE                       -0.027839
SANCTIONED.AMOUNT                     -0.011749
DISBURSED.AMOUNT                      -0.011591
INSTAL.AMT                            -0.010707
Name: loan_default, dtype: float64
```

Since Current balance has negative values we will take scaled between 0 and 1

In [27]:
```
scalar=MinMaxScaler()
linear_models_final_df["CURRENT.BALANCE"]=scalar.fit_transform(linear_models_final_df["CURRENT.BALANCE"].values.reshape(
```

In [28]:
```python
val=[]
for i in num_cols:
    val.append(linear_models_df[i].skew())
skew_df=pd.DataFrame(index=num_cols,data=val,columns=["Scores"])
skew_df
```

Out[28]:

	Scores
disbursed_amount	4.492240
asset_cost	6.133485
ltv	-1.075766
PERFORM_CNS.SCORE	0.445150
NEW.ACCTS.IN.LAST.SIX.MONTHS	4.839326
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS	6.641996
AVERAGE.ACCT.AGE	3.285142
CREDIT.HISTORY.LENGTH	2.969155
NO.OF_INQUIRIES	7.870683
Age_at_disbursement	0.608667
NO.OF.ACCTS	9.474425
ACTIVE.ACCTS	5.278660
OVERDUE.ACCTS	7.312614
CURRENT.BALANCE	28.589453
SANCTIONED.AMOUNT	320.010642
DISBURSED.AMOUNT	318.903561
INSTAL.AMT	68.804991

In [29]:
```python
cols_to_taken_log=skew_df[skew_df["Scores"]>2].index
```

In [30]:
```python
for i in cols_to_taken_log:
    linear_models_final_df[i]=np.log(linear_models_final_df[i]+1)
```

In [31]:
```python
X=linear_models_final_df.drop('loan_default',axis=1)
y=linear_models_final_df['loan_default']
model_scores={}
```

In [32]:
```python
linear_models_final_df.head()
```

Out[32]:

	disbursed_amount	asset_cost	ltv	Aadhar_flag	PAN_flag	VoterID_flag	Driving_flag	Passport_flag	PERFORM_CNS.SCORE	NEW.ACCTS.IN.LAST.SIX.I
0	10.831292	10.975088	89.55	1	0	0	0	0	0	
1	10.883298	11.024530	89.63	1	0	0	0	0	0	
2	10.866261	11.007104	88.39	1	0	0	0	0	0	
3	10.743977	11.026809	76.42	1	0	0	0	0	0	
4	10.682698	11.267754	57.50	1	0	0	0	0	0	

5 rows × 153 columns

Model Building

In [38]:
```python
#model selection from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
#ensemble
from sklearn.ensemble import RandomForestClassifier,BaggingClassifier,AdaBoostClassifier,GradientBoostingClassifier
from xgboost import XGBClassifier
#metrics
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, accuracy_score, classification_repor
```

In [39]:
```
from sklearn.model_selection import train_test_split,KFold,cross_val_score
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)
```

In [40]:
```
y_train.shape
```

Out[40]: (163207,)

In [41]:
```
X_test.shape,y_test.shape
```

Out[41]: ((69947, 152), (69947,))

In [42]:
```
#dict to store various model results for comparison
results = {}
```

In [43]:
```
#model result is a function for model building and calculating scores
def modelresult(model_name,i):
    print(i,"\n")
    classifier = i
    classifier.fit(X_train, y_train)
    y_predt=classifier.predict(X_train)
    y_pred=classifier.predict(X_test)
    y_pred_prob=classifier.predict_proba(X_test)[:,1]
    results[model_name] = {'Accuracy' : accuracy_score(y_test,y_pred), 'roc_auc_score' : roc_auc_score(y_test,y_pred_prol
    print("\n Accuracy for the Train:",accuracy_score(y_train,y_predt))
    print("\n Accuracy for the Test",accuracy_score(y_test,y_pred),"\n")
    print("\n ROC AUC score",roc_auc_score(y_test,y_pred_prob),"\n")
    from sklearn.metrics import classification_report
    print(classification_report(y_test,y_pred))
```

Logistic Regression

In [44]:
```
lr = LogisticRegression(random_state=1)
modelresult("Linear Reg",lr)
```

```
LogisticRegression(random_state=1)


 Accuracy for the Train: 0.7822397323644207

 Accuracy for the Test 0.7848084978626675


 ROC AUC score 0.626169361004602

              precision    recall  f1-score   support

           0       0.78      1.00      0.88     54893
           1       0.53      0.00      0.00     15054

    accuracy                           0.78     69947
   macro avg       0.66      0.50      0.44     69947
weighted avg       0.73      0.78      0.69     69947
```

In [45]:
```
pd.DataFrame(results)
```

Out[45]:

	Linear Reg
Accuracy	0.784808
roc_auc_score	0.626169

Decision Tree classifier

In [56]:
```
dcg = DecisionTreeClassifier(criterion='entropy',random_state=1)
modelresult("DT-Entropy" ,dcg)
```

```
DecisionTreeClassifier(criterion='entropy', random_state=1)


 Accuracy for the Train: 0.9996752590268799

 Accuracy for the Test 0.6791999656883069


 ROC AUC score 0.5325419028239513

              precision    recall  f1-score   support

           0       0.80      0.79      0.79     54893
           1       0.26      0.27      0.27     15054

    accuracy                           0.68     69947
   macro avg       0.53      0.53      0.53     69947
weighted avg       0.68      0.68      0.68     69947
```

Decision Tree classifier with gini

In [47]:
```
dcg = DecisionTreeClassifier(criterion='gini',random_state=1)
modelresult("DT-Gini",dcg)
```

```
DecisionTreeClassifier(random_state=1)


 Accuracy for the Train: 0.9996752590268799

 Accuracy for the Test 0.6745821836533376


 ROC AUC score 0.5298714189214917

              precision    recall  f1-score   support

           0       0.80      0.78      0.79     54893
           1       0.26      0.28      0.27     15054

    accuracy                           0.67     69947
   macro avg       0.53      0.53      0.53     69947
weighted avg       0.68      0.67      0.68     69947
```

RandomForest Classifier

In [48]:
```
rfc = RandomForestClassifier(n_estimators=100,random_state=1)
modelresult("RandomForest",rfc)
```

```
RandomForestClassifier(random_state=1)


 Accuracy for the Train: 0.9996446230860196

 Accuracy for the Test 0.7791470684947174


 ROC AUC score 0.6305002753390946

              precision    recall  f1-score   support

           0       0.79      0.98      0.87     54893
           1       0.40      0.05      0.09     15054

    accuracy                           0.78     69947
   macro avg       0.59      0.51      0.48     69947
weighted avg       0.71      0.78      0.71     69947
```

Bagging Classifier

In [49]:
```
bgc = BaggingClassifier(random_state=1)
modelresult("Bagging",bgc)
```

```
BaggingClassifier(random_state=1)


 Accuracy for the Train: 0.9746395681557777

 Accuracy for the Test 0.7690680086351095


 ROC AUC score 0.5922704236487604
              precision    recall  f1-score   support

           0       0.79      0.96      0.87     54893
           1       0.35      0.09      0.14     15054

    accuracy                           0.77     69947
   macro avg       0.57      0.52      0.50     69947
weighted avg       0.70      0.77      0.71     69947
```

AdaBoost Classifier

In [50]:
```
ada = AdaBoostClassifier(random_state=1)
modelresult("Adaboost", ada)
```

```
AdaBoostClassifier(random_state=1)


 Accuracy for the Train: 0.7815044697837715

 Accuracy for the Test 0.7850801321000186


 ROC AUC score 0.6489863472474202
              precision    recall  f1-score   support

           0       0.79      1.00      0.88     54893
           1       0.52      0.02      0.03     15054

    accuracy                           0.79     69947
   macro avg       0.65      0.51      0.46     69947
weighted avg       0.73      0.79      0.70     69947
```

XGBoost Classifier

In [51]:
```
xgb =  XGBClassifier(random_state=1, n_jobs=-1, learning_rate=0.2,n_estimators=100, max_depth=3)
modelresult("XGboost",xgb)
```

```
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=0.2, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=-1,
              num_parallel_tree=None, random_state=1, ...)


 Accuracy for the Train: 0.7830362668267905

 Accuracy for the Test 0.7852373940269061


 ROC AUC score 0.6600821452440933

              precision    recall  f1-score   support

           0       0.79      1.00      0.88     54893
           1       0.56      0.01      0.02     15054

    accuracy                           0.79     69947
   macro avg       0.67      0.50      0.45     69947
weighted avg       0.74      0.79      0.69     69947
```

In [52]:
```
pd.DataFrame(results).T
```

Out[52]:

	Accuracy	roc_auc_score
Linear Reg	0.784808	0.626169
DT-Entropy	0.679200	0.532542
DT-Gini	0.674582	0.529871
RandomForest	0.779147	0.630500
Bagging	0.769068	0.592270
Adaboost	0.785080	0.648986
XGboost	0.785237	0.660082

```
We can observe that Xgboost classifier has highest accuracy score of 78.3% and roc_auc_score of 66% of all models built.
```

Evaluation Metric

- ROC AUC-Score was chosen as the metric for the models.

Models

Here we are trying Linear, distance and tree-based models in the conviction which splits the target variables at its best. Since the metric of interest for the problem statement is AUC, from the below output we can conclude that tree based generally outperforms linear based models hence we would be using tree-based model for our further analysis.

- Logistic Regression ROC AUC SCORE: 0.55
- KNN ROC AUC SCORE: 0.52
- Random Forest ROC AUC SCORE: 0.60
- KNN Classifier ROC AUC SCORE: 0.55

Conclusion

- In this project Vehicle loan defaulters in the first EMI for L&T have been determined. The best performing models were ensemble-based models.
- The data seems to exactly mimic the real-life scenario which is very evident since there many zero values present which corresponds to first-time customers.