DATA MINING

Complex Computing Problem

Report

BS(CS)-7(A)

Imaan Shahid (02-134201-009)

BAHRIA UNIVERSITY KARACHI CAMPUS

Table of Contents

Introduction

The main aim of this report is to represent the problem-solving methodology used to solve the complex computing problem of building a “Predictive Model for Credit Risk Assessment”. In which the models must classify whether an applicant is loan default or not. All necessary steps are discussed from data preprocessing, feature selection, model selection, k-fold cross validation to analyzing and interpreting the results of the models that were implemented.

Dataset

The dataset that was taken from Kaggle included the 9 attributes against 1 target attribute of loan_default. The following represent the information of all the attributes and what they represent.

- loan_id:Unique identifier of a loan
- age:Age of the Applicant
- Education:Applicant Education
- proof_submitted:Type of proof submitted
- loan_amount: Loan Amount Disbursed
- asset_cost: The total asset value of the applicant
- no_of_loans: No. of the loans taken by the applicant
- no_of_curr_loans:No. of active loans held by the applicant
- last_delinq_none:The loan defaulted in at least one of the past loans
- loan_default (Target Variable):0/1 indicating if an applicant will default on the loan or not.

All the columns were used in the dataset except the education one because it wasn’t defined on the Kaggle site what the 1 and 2 values represent in the education column. As well as experimenting with keeping the education column it didn’t have any significant impact on the algorithms thus, I chose to not keep this attribute in the dataframe.

Code and Analysis

Data Preprocessing:

The following code snippets depict all the data processing steps I applied on the dataset. Here are the main steps I implemented which can summarize the whole data preprocessing I applied on the dataset:

1. Firstly, I loaded the dataset into python dataframe and ran the few basic lines of code to understand the nature of the dataset such as checking if there were any missing values in dataset and the datatype of all the attributes present in the dataset.

2. Secondly, I checked if the data was imbalanced or not as an imbalanced dataset can result in biasness in the accuracy of model results. So, when checked the data was very imbalanced as it had a greater number of '0' class rows than '1' class rows. To convert this dataset into balanced dataset I used the under-sampling technique to remove the extra number of '0' rows. And then the final balanced dataset consisted of the equal number of '0' and '1' loan_default class as shown in code line number 13.

3. After balancing the dataset, I applied one-hot encoding to convert the categorical attribute of "proof provided" into numeric format so it can be understood more effectively when we apply the machine learning algorithms.

4. Then I checked for outliers on the dataset by drawing boxplot and understanding whether we should keep or delete the outliers. The boxplot was drawn with respect to the target class, and it didn't show any significant relationship such as if one class had outliers and the other didn't. Due to the absence of any relationship of outliers with respect to the loan_default class I decided to remove the outliers from the dataset.

5. After removing outliers, I saved the cleaned dataset and separated a few rows creating a validation dataset which will be used at the end to evaluate the models on unseen data.

DATA MINING CCP

1. DATA PREPROCESSING

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report,confusion_matrix

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
```

```
In [2]: df = pd.read_csv('loan default train dataset.csv')
        df.head()
```

Out[2]:

	loan_id	age	education	proof_submitted	loan_amount	asset_cost	no_of_loans	no_of_curr_loans	last_delinq_none	loan_default
0	1	27	1.0	Aadhar	504264	820920	2	2	0	0
1	2	48	1.0	Aadhar	728556	831444	6	2	0	0
2	3	30	2.0	VoterID	642936	826092	0	0	0	1
3	4	28	1.0	Aadhar	746556	930924	0	0	0	0
4	5	29	1.0	Aadhar	1139880	1902000	0	0	0	0

```
In [3]: df2=df.drop(["education"],axis=1)
        df2.head()
```

Out[3]:

	loan_id	age	proof_submitted	loan_amount	asset_cost	no_of_loans	no_of_curr_loans	last_delinq_none	loan_default
0	1	27	Aadhar	504264	820920	2	2	0	0
1	2	48	Aadhar	728556	831444	6	2	0	0
2	3	30	VoterID	642936	826092	0	0	0	1
3	4	28	Aadhar	746556	930924	0	0	0	0
4	5	29	Aadhar	1139880	1902000	0	0	0	0

```
In [4]: df2.shape

Out[4]: (7000, 9)
```

```
In [5]: df2.isnull().sum()

Out[5]: loan_id             0
        age                 0
        proof_submitted     0
        loan_amount         0
        asset_cost          0
        no_of_loans         0
        no_of_curr_loans    0
        last_delinq_none    0
        loan_default        0
        dtype: int64
```

```
In [6]: df2.isnull().any()

Out[6]: loan_id             False
        age                 False
        proof_submitted     False
        loan_amount         False
        asset_cost          False
        no_of_loans         False
        no_of_curr_loans    False
        last_delinq_none    False
        loan_default        False
        dtype: bool
```

```
In [7]: df2.describe()
```

Out[7]:

	loan_id	age	loan_amount	asset_cost	no_of_loans	no_of_curr_loans	last_delinq_none	loan_default
count	7000.000000	7000.000000	7.000000e+03	7.000000e+03	7000.000000	7000.000000	7000.000000	7000.000000
mean	3500.500000	36.096571	6.633552e+05	9.162998e+05	2.853286	1.371143	0.013286	0.400000
std	2020.870275	7.587700	1.498128e+05	2.144922e+05	5.471932	2.189278	0.114504	0.489933
min	1.000000	21.000000	1.678800e+05	4.733520e+05	0.000000	0.000000	0.000000	0.000000
25%	1750.750000	29.000000	5.777880e+05	7.979010e+05	0.000000	0.000000	0.000000	0.000000
50%	3500.500000	36.000000	6.571080e+05	8.584260e+05	0.000000	0.000000	0.000000	0.000000
75%	5250.250000	43.000000	7.373640e+05	9.576750e+05	3.000000	2.000000	0.000000	1.000000
max	7000.000000	50.000000	1.781376e+06	2.419200e+06	109.000000	33.000000	1.000000	1.000000

```
In [8]: df2.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7000 entries, 0 to 6999
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   loan_id           7000 non-null   int64
 1   age               7000 non-null   int64
 2   proof_submitted   7000 non-null   object
 3   loan_amount       7000 non-null   int64
 4   asset_cost        7000 non-null   int64
 5   no_of_loans       7000 non-null   int64
 6   no_of_curr_loans  7000 non-null   int64
 7   last_delinq_none  7000 non-null   int64
```

```
 8   loan_default      7000 non-null   int64
dtypes: int64(8), object(1)
memory usage: 492.3+ KB
```

Checking if data is imbalance and converting it into balanced dataset

In [9]:
```
value_counts = df2["loan_default"].value_counts()
print(value_counts)
```

```
0    4200
1    2800
Name: loan_default, dtype: int64
```

In [10]:
```
import imblearn
from collections import Counter
```

In [11]:
```
X=df2.drop(["loan_default"],axis=1)
y=df2["loan_default"]
```

In [12]:
```
from imblearn.under_sampling import RandomUnderSampler
rs=RandomUnderSampler(random_state=42)
```

In [13]:
```
X_res,y_res=rs.fit_resample(X,y)
print("After sampling %s" % Counter(y_res))
```

```
After sampling Counter({0: 2800, 1: 2800})
```

In [14]:
```
sampled_data = pd.DataFrame(X_res, columns=X.columns)
sampled_data['loan_default'] = y_res
```

In [15]:
```
sampled_data
```

Out[15]:

	loan_id	age	proof_submitted	loan_amount	asset_cost	no_of_loans	no_of_curr_loans	last_delinq_none	loan_default
0	2857	35	Aadhar	550188	1055736	8	3	0	0
1	3655	33	Aadhar	580188	749052	0	0	0	0
2	2833	39	Aadhar	542748	614076	0	0	0	0
3	5556	32	Aadhar	675108	1807356	0	0	0	0
4	479	41	Aadhar	750156	975600	0	0	0	0
...	...	...	...	...	...	...	...	...	...
5595	6995	40	Aadhar	696156	868584	0	0	0	1
5596	6996	45	Aadhar	930948	1258344	0	0	0	1
5597	6997	41	Aadhar	681108	791040	4	4	0	1
5598	6998	47	Aadhar	627636	720336	35	11	0	1
5599	6999	39	Aadhar	654708	793860	0	0	0	1

5600 rows × 9 columns

In [16]:
```
sampled_data.to_csv('sampled_data.csv', index=False)
```

Applied One hot encoding

In [17]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
```

In [18]:
```
df = pd.read_csv("sampled_data.csv")
```

In [19]:
```
df.dtypes
```

Out[19]:
```
loan_id              int64
age                  int64
proof_submitted     object
loan_amount          int64
asset_cost           int64
no_of_loans          int64
no_of_curr_loans     int64
last_delinq_none     int64
loan_default         int64
dtype: object
```

In [20]:
```
df["proof_submitted"].unique()
```

Out[20]:
```
array(['Aadhar', 'VoterID', 'Driving', 'PAN', 'Passport'], dtype=object)
```

In [21]:
```
ohe = OneHotEncoder()
```

In [22]:
```
feature_arry = ohe.fit_transform(df[["proof_submitted"]]).toarray()
```

```
In [22]: feature_arry = ohe.fit_transform(df[["proof_submitted"]]).toarray()
```

```
In [23]: print(feature_arry)
```

```
[[1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 ...
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]]
```

```
In [24]: feature_labels = ohe.categories_
         np.array(feature_labels).ravel()
```

```
Out[24]: array(['Aadhar', 'Driving', 'PAN', 'Passport', 'VoterID'], dtype=object)
```

```
In [25]: features=pd.DataFrame(feature_arry, columns = feature_labels)
```

```
In [26]: df1=pd.concat([df, features], axis=1)
```

```
In [27]: df1.head()
```

Out[27]:

	loan_id	age	proof_submitted	loan_amount	asset_cost	no_of_loans	no_of_curr_loans	last_delinq_none	loan_default	(Aadhar,)	(Driving,)	(PAN,)	(Pa
0	2857	35	Aadhar	550188	1055736	8	3	0	0	1.0	0.0	0.0	
1	3655	33	Aadhar	580188	749052	0	0	0	0	1.0	0.0	0.0	
2	2833	39	Aadhar	542748	614076	0	0	0	0	1.0	0.0	0.0	
3	5556	32	Aadhar	675108	1807356	0	0	0	0	1.0	0.0	0.0	
4	479	41	Aadhar	750156	975600	0	0	0	0	1.0	0.0	0.0	

```
In [28]: df1=df1.drop(["proof_submitted"],axis=1)
```

```
In [29]: df1.to_csv('sampled_data.csv', index=False)
```

Deciding if outliers should be kept or removed with respect to their relationship with the target class

```
In [30]: sampled_data2=pd.read_csv("sampled_data.csv")
```

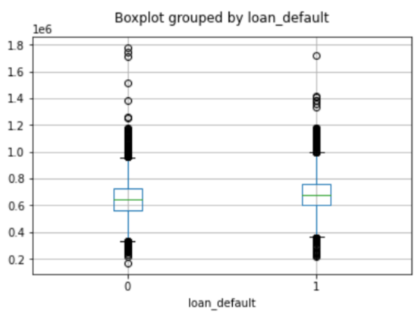
```
In [31]: sampled_data2.head()
```

Out[31]:

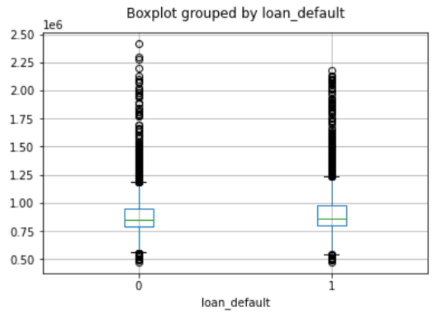
	loan_id	age	loan_amount	asset_cost	no_of_loans	no_of_curr_loans	last_delinq_none	loan_default	('Aadhar',)	('Driving',)	('PAN',)	('Passport',)	('Vot
0	2857	35	550188	1055736	8	3	0	0	1.0	0.0	0.0	0.0	
1	3655	33	580188	749052	0	0	0	0	1.0	0.0	0.0	0.0	
2	2833	39	542748	614076	0	0	0	0	1.0	0.0	0.0	0.0	
3	5556	32	675108	1807356	0	0	0	0	1.0	0.0	0.0	0.0	
4	479	41	750156	975600	0	0	0	0	1.0	0.0	0.0	0.0	

```
In [32]: def plot_boxplot(df,ft):

             boxplot = sampled_data2.boxplot(column=ft,by="loan_default")
             boxplot.set_title("")

             plt.show()
```

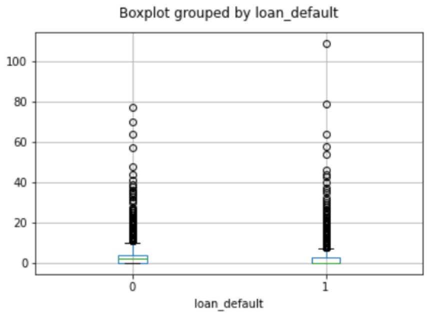
```
In [33]: plot_boxplot(sampled_data2,"loan_amount")
```



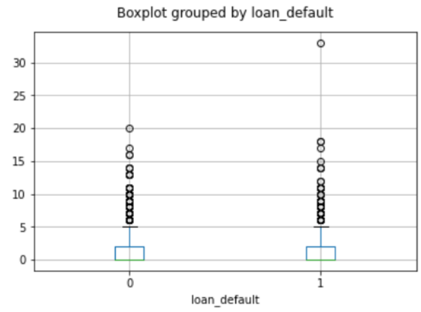
In [34]:
```python
plot_boxplot(sampled_data2,"asset_cost")
```



In [35]:
```python
plot_boxplot(sampled_data2,"no_of_loans")
```



In [36]:
```python
plot_boxplot(sampled_data2,"no_of_curr_loans")
```



Removal of outliers

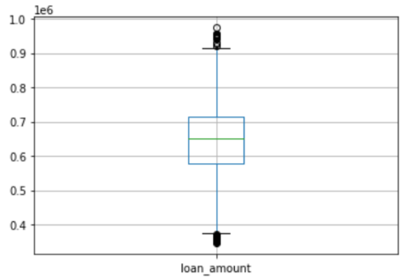
In [37]:
```python
def remove_outliers(df, threshold):
    for column in df.columns:
        # Calculate the quartiles and IQR for the column
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1

        # Filter the column to remove outliers
        df = df[(df[column] >= Q1 - threshold * IQR) & (df[column] <= Q3 + threshold * IQR)]

    return df
```

In [38]:
```python
threshold = 1.5
df3=sampled_data2

# Call the function to remove outliers from the whole DataFrame
df_filtered = remove_outliers(df3, threshold)
```

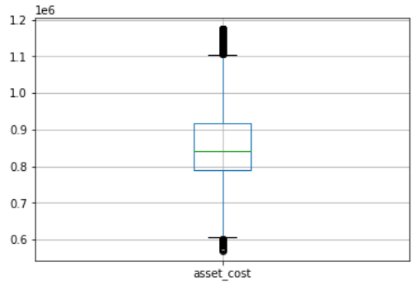
In [39]:
```
print(df_filtered)
```

```
      loan_id  age  loan_amount  asset_cost  no_of_loans  no_of_curr_loans  \
0        2857   35       550188     1055736            8                 3
1        3655   33       580188      749052            0                 0
2        2833   39       542748      614076            0                 0
4         479   41       750156      975600            0                 0
5        3014   47       572988      824460            4                 2
...       ...  ...          ...         ...          ...               ...
5590     6981   41       536940      747840            0                 0
5594     6992   37       639636      984144            0                 0
5595     6995   40       696156      868584            0                 0
5597     6997   41       681108      791040            4                 4
5599     6999   39       654708      793860            0                 0

      last_delinq_none  loan_default  ('Aadhar',)  ('Driving',)  ('PAN',)  \
0                    0             0          1.0           0.0       0.0
1                    0             0          1.0           0.0       0.0
2                    0             0          1.0           0.0       0.0
4                    0             0          1.0           0.0       0.0
5                    0             0          1.0           0.0       0.0
...                ...           ...          ...           ...       ...
5590                 0             1          1.0           0.0       0.0
5594                 0             1          1.0           0.0       0.0
5595                 0             1          1.0           0.0       0.0
5597                 0             1          1.0           0.0       0.0
5599                 0             1          1.0           0.0       0.0

      ('Passport',)  ('VoterID',)
0               0.0           0.0
1               0.0           0.0
5               0.0           0.0
...             ...           ...
5590            0.0           0.0
5594            0.0           0.0
5595            0.0           0.0
5597            0.0           0.0
5599            0.0           0.0

[3795 rows x 13 columns]
```

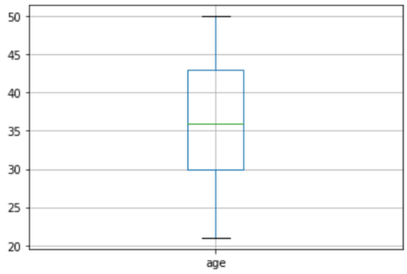
In [40]:
```
boxplot = df_filtered.boxplot(column="loan_amount")
boxplot.set_title("")

plt.show()
```



In [41]:
```
boxplot = df_filtered.boxplot(column="asset_cost")
boxplot.set_title("")

plt.show()
```



In [42]:
```
boxplot = df_filtered.boxplot(column="age")
boxplot.set_title("")

plt.show()
```

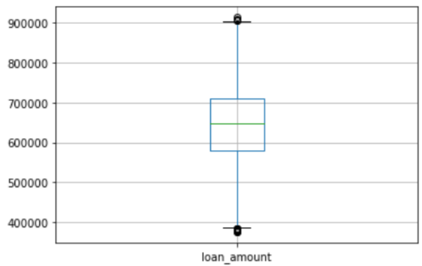


In [43]:
```
threshold = 1.5

df_filtered2 = remove_outliers(df_filtered, threshold)
```

In [44]:
```
print(df_filtered2)
```

```
      loan_id  age  loan_amount  asset_cost  no_of_loans  no_of_curr_loans  \
1        3655   33       580188      749052            0                 0
2        2833   39       542748      614076            0                 0
4         479   41       750156      975600            0                 0
5        3014   47       572988      824460            4                 2
7         573   42       772584      941976            0                 0
...       ...  ...          ...         ...          ...               ...
5590     6981   41       536940      747840            0                 0
5594     6992   37       639636      984144            0                 0
5595     6995   40       696156      868584            0                 0
5597     6997   41       681108      791040            4                 4
5599     6999   39       654708      793860            0                 0

      last_delinq_none  loan_default  ('Aadhar',)  ('Driving',)  ('PAN',)  \
1                    0             0          1.0           0.0       0.0
2                    0             0          1.0           0.0       0.0
4                    0             0          1.0           0.0       0.0
5                    0             0          1.0           0.0       0.0
7                    0             0          1.0           0.0       0.0
...                ...           ...          ...           ...       ...
5590                 0             1          1.0           0.0       0.0
5594                 0             1          1.0           0.0       0.0
5595                 0             1          1.0           0.0       0.0
5597                 0             1          1.0           0.0       0.0
5599                 0             1          1.0           0.0       0.0
7                0.0           0.0
...              ...           ...
5590             0.0           0.0
5594             0.0           0.0
5595             0.0           0.0
5597             0.0           0.0
5599             0.0           0.0

[3471 rows x 13 columns]
```

In [45]:
```
boxplot = df_filtered2.boxplot(column="loan_amount")
boxplot.set_title("")

plt.show()
```



Saving the cleaned dataset into csv file and separating a few data rows for validation steps

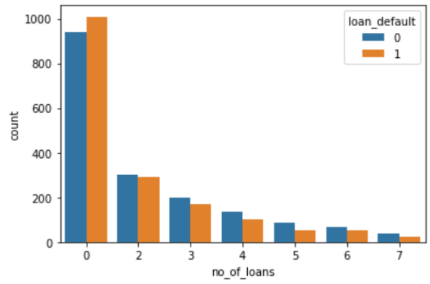
```
In [46]: df_filtered2.to_csv('sampled_data.csv', index=False)
```

```
In [47]:
data = pd.read_csv('sampled_data.csv')
num_rows_to_remove = 10

# Randomly select rows to remove
removed_rows = data.sample(n=num_rows_to_remove, random_state=42)

# Drop the selected rows from the original DataFrame
data_filtered = data.drop(removed_rows.index)

# Save the modified dataset
data_filtered.to_csv('sampled_data.csv', index=False)

# Save the removed rows
removed_rows.to_csv('validation.csv', index=False)
```

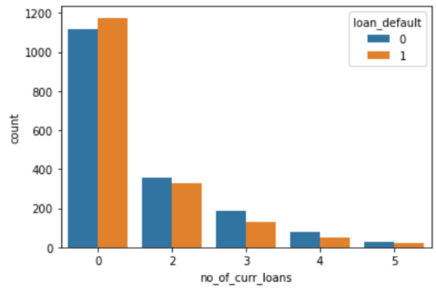
Data Visualization

```
In [48]: df_filtered2["no_of_loans"].unique()
Out[48]: array([0, 4, 3, 5, 2, 6, 7], dtype=int64)
```

```
In [49]: sns.countplot(x='no_of_loans', data=df_filtered2, hue="loan_default")
Out[49]: <AxesSubplot:xlabel='no_of_loans', ylabel='count'>
```




```
In [50]: sns.countplot(x='no_of_curr_loans',data=df_filtered2,hue='loan_default')
Out[50]: <AxesSubplot:xlabel='no_of_curr_loans', ylabel='count'>
```




```
In [51]: df_filtered["no_of_curr_loans"].unique()
Out[51]: array([3, 0, 2, 4, 5], dtype=int64)
```

2. Feature Selection:

Feature selection is a very important step to perform to prevent underfitting and overfitting of the models that will be applied, thus the feature selection methods that I have used can be summarized as follows:

1. I applied 3 various feature selection methods to ensure the correct features and attributes are selected for the model training. The three techniques that were implemented are correlation, chi-squared test, and information gain.

2. The correlation matrix gave insight on the attributes that were highly correlated such as the attribute “loan_amount” was highly correlated with “asset_cost”. As well as the attribute “no_of_curr_loans” and “no_of_loans” were correlated too. Thus, these attributes are so correlated we don’t have to train the model using both attributes we can choose one from each pair of correlated attributes as if we use both attributes, it would be redundant as correlated attributes carry the same amount of value when implemented while training a model.

3. The chi-squared test is used to check how each attribute in the dataset is dependent or independent on the target variable of “loan_default”. Each attribute’s dependence on the target variable is represented by the score of the attribute, the higher the score the more it is dependent on the target variable and thus can be used to train model. The top three attributes that chi-squared listed are “loan_amount”, “no_of_curr_loans” and one of the proofs submitted categories “Aadhar”.

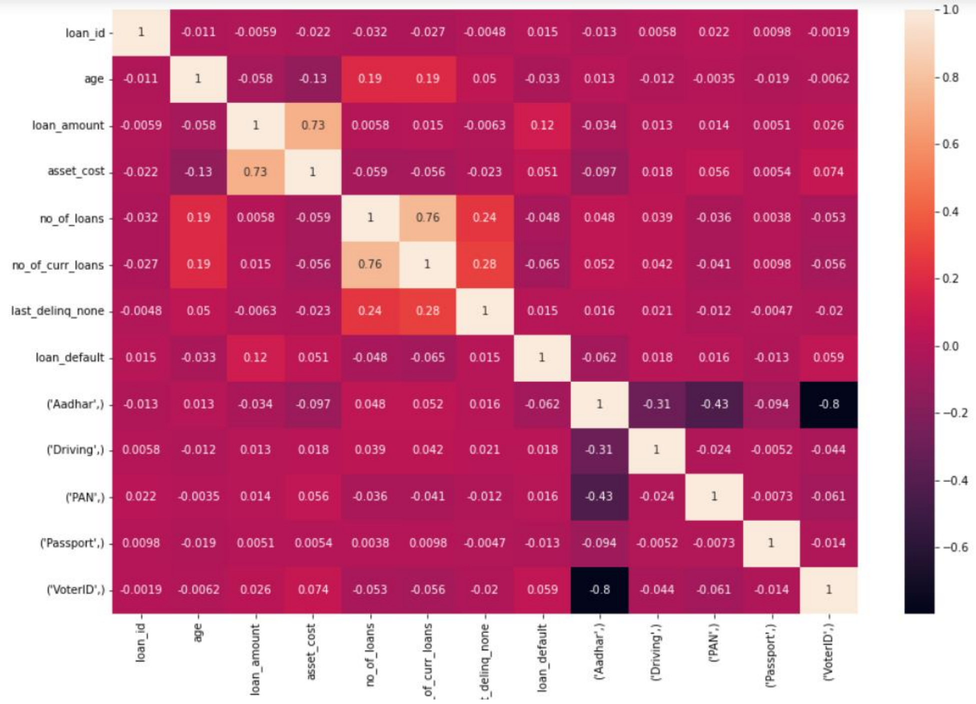
4. Finally, the Information gain quantifies the amount of information or uncertainty that a feature provides about the target variable. The feature with the highest information gain is considered the most informative or valuable. In this case the highest information gain was of “loan_amount”, “no_of_loans” and “Aadhar”.

So, the top three features that we used to train our dataset were “loan_amount”, “no_of_curr_loans” and “Aadhar”. As “no_of_loans” and “no_of_curr_loans” are highly correlated it doesn’t matter which one we choose if we choose of them. The following code snippets represent the feature selection:

2. Feature Selection

CORRELATION:

In [52]:
```
CorrMat=sampled_data2.corr()
plt.figure(figsize=(15,10))

sns.heatmap(CorrMat,annot=True)
```
Out[52]: <AxesSubplot:>



CHI SQUARED TEST

In [53]:
```python
from sklearn.feature_selection import SelectPercentile

from sklearn.feature_selection import SelectKBest, chi2,f_classif
```

In [54]:
```python
sampled_data2.head()
```

Out[54]:

	loan_id	age	loan_amount	asset_cost	no_of_loans	no_of_curr_loans	last_delinq_none	loan_default	('Aadhar',)	('Driving',)	('PAN',)	('Passport',)	('Vot
0	2857	35	550188	1055736	8	3	0	0	1.0	0.0	0.0	0.0	
1	3655	33	580188	749052	0	0	0	0	1.0	0.0	0.0	0.0	
2	2833	39	542748	614076	0	0	0	0	1.0	0.0	0.0	0.0	
3	5556	32	675108	1807356	0	0	0	0	1.0	0.0	0.0	0.0	
4	479	41	750156	975600	0	0	0	0	1.0	0.0	0.0	0.0	

In [55]:
```python
X=sampled_data2.drop(["loan_default"],axis=1)
Y=sampled_data2["loan_default"]
```

In [56]:
```python
chi2_selector=SelectKBest(f_classif,k=3)
kBest=chi2_selector.fit_transform(X,Y)
```

In [57]:
```python
selected_features = X.columns[chi2_selector.get_support()]
```

In [58]:
```python
chi2_scores=pd.DataFrame(list(zip(X.columns,chi2_selector.scores_)),columns=["feature","score"])
```

In [59]:
```python
print("Selected Features:")
print(selected_features)
```

```
Selected Features:
Index(['loan_amount', 'no_of_curr_loans', '('Aadhar',)'], dtype='object')
```

In [60]:
```
chi2_scores
```

Out[60]:

	feature	score
0	loan_id	1.245573
1	age	6.116801
2	loan_amount	79.955283
3	asset_cost	14.353933
4	no_of_loans	12.846898
5	no_of_curr_loans	23.847849
6	last_delinq_none	1.333734
7	('Aadhar',)	21.929216
8	('Driving',)	1.847842
9	('PAN',)	1.485281
10	('Passport',)	1.001431
11	('VoterID',)	19.352055

INFORMATION GAIN

In [61]:
```
from sklearn.feature_selection import mutual_info_classif
```

In [62]:
```
X.head()
```

Out[62]:

	loan_id	age	loan_amount	asset_cost	no_of_loans	no_of_curr_loans	last_delinq_none	('Aadhar',)	('Driving',)	('PAN',)	('Passport',)	('VoterID',)
0	2857	35	550188	1055736	8	3	0	1.0	0.0	0.0	0.0	0.0
1	3655	33	580188	749052	0	0	0	1.0	0.0	0.0	0.0	0.0
2	2833	39	542748	614076	0	0	0	1.0	0.0	0.0	0.0	0.0
3	5556	32	675108	1807356	0	0	0	1.0	0.0	0.0	0.0	0.0
4	479	41	750156	975600	0	0	0	1.0	0.0	0.0	0.0	0.0

In [63]:
```
feature_scores=mutual_info_classif(X,Y,random_state=0)
feature_scores
```

Out[63]:
```
array([0.        , 0.00438302, 0.00923336, 0.00525722, 0.01210109,
       0.        , 0.        , 0.00982624, 0.00041892, 0.0072607 ,
       0.        , 0.        ])
```

3. Model selection:

The problem that we have is classification, thus we can implement supervised algorithms on the dataset. The models that I implemented were SVM, Decision trees and Logistic regression. Before applying the algorithms train-test split was applied to the dataset and then scaled the train test split to ensure that the data is normalized. Then I applied the three algorithms and printed their accuracy, classification report as well as the confusion matrix. Before understanding the results of the models generated lets first understand the fundamental working of the algorithms:

1. Support Vector Machines (SVM):
   SVM is a machine learning algorithm used for classification tasks. The main idea behind SVM is to find a line or a hyperplane that separates different classes of data points as clearly as possible. It tries to maximize the margin, which is the distance between the decision boundary and the nearest data points of each class. SVM works well for both linearly separable and non-linearly separable data by using a technique called the kernel trick, which maps the data to a higher-dimensional space where it can be linearly separable.

2. Logistic Regression:
   Logistic regression is another classification algorithm commonly used in machine learning. It is specifically designed for binary classification problems, where the goal is to predict whether an input belongs to one of two classes (e.g., Yes/No, True/False). Despite its name, logistic regression is a type of regression algorithm used for estimating the probability of an event occurring. It calculates the relationship between the input features and the log-odds of the event happening, which is then transformed into a probability using a sigmoid function. The resulting probability can be interpreted as the likelihood of belonging to a particular class.

3. Decision Tree:
   A decision tree is a flowchart-like structure used for making decisions or predictions in machine learning. It consists of nodes that represent features, branches that represent decisions based on those features, and leaves that represent the outcomes or predictions. Each internal node of the tree tests a specific feature, and the decision is made by following the corresponding branch based on the feature's value. Decision trees are intuitive and easy to interpret, making them useful for both classification and regression tasks. They can handle both numerical and categorical data and can capture complex relationships between variables.

   To understand the performance of the algorithms we must understand the elements of classification reports, specifically precision and recall their meanings are described as follows:

- Precision: It calculates the ratio of true positive predictions to the total number of positive predictions made. In other words, precision tells us how many of the positive predictions were actually correct.

- Recall tells us how many of the actual positive instances were successfully identified by the model. A high recall value indicates that the model has a low rate of false negatives.

Now let’s discuss the overall accuracy, precision and recall of the algorithms implemented:

  - SVM gave an overall accuracy of 0.566, precision of 0.55 for an occurrence of loan default class and recall of 0.69.

  - Decision trees gave the overall accuracy of 0.53, precision of 0.53 as well as recall of 0.48.

  - Logistic regression gave the overall accuracy of 0.58, precision of 0.58 and recall of 0.61.

As we can see the overall accuracy, precision and recall represents that the logistic regression performed the most well, then SVM and lastly decision trees. However, this is the representation of the performance on the train test split that the model had seen and knows the dataset. As well as train test split isn't the most effective way to know the overall accuracy of a model, thus we need to ensure that the right accuracy is evaluated with a more robust measure of performance such as k-fold cross validation.

The below code snippets showcase how the various models were implemented:

3. Model selection

In [64]:
```
from sklearn.model_selection import train_test_split


y = sampled_data2['loan_default']
X = sampled_data2[["loan_amount","no_of_curr_loans","('Aadhar',)"]]



from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# X_train # return dataframe train
print(X_train)
```

```
[[ 0.07763981  0.          0.        ]
 [ 0.09701102  0.          0.        ]
 [ 0.01464464  0.         -1.        ]
 ...
 [ 1.07479224  1.          0.        ]
 [-0.64676598  0.          0.        ]
 [-0.97762625  0.          0.        ]]
```

SVM

In [65]:
```
# Instantiate the Support Vector Classifier (SVC)
# svc = SVC(C=1.0, random_state=1, kernel='linear')
svc=SVC()
# Fit the model
svc.fit(X_train, y_train)
# svc.fit(X_train, y_train)
```

Out[65]:
```
▾ SVC
SVC()
```

In [66]:
```
# Make the predictions
y_predict = svc.predict(X_test)

# # Measure the performance
# print("Accuracy score %.3f" %metrics.accuracy_score(y_test, y_predict))

svc.score(X_test,y_test)
```

Out[66]:
```
0.5660714285714286
```

In [67]:
```
print(classification_report(y_test,y_predict))
print(confusion_matrix(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.59      0.45      0.51       840
           1       0.55      0.69      0.61       840

    accuracy                           0.57      1680
   macro avg       0.57      0.57      0.56      1680
weighted avg       0.57      0.57      0.56      1680

[[374 466]
 [263 577]]
```

Decision Trees

```
In [68]: dt=DecisionTreeClassifier()
         dt.fit(X_train,y_train)
         pred=dt.predict(X_test)
```

```
In [69]: dt.score(X_test,y_test)

Out[69]: 0.530952380952381
```

```
In [70]: print(classification_report(y_test,pred))
         print(confusion_matrix(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.53      0.59      0.56       840
           1       0.53      0.48      0.50       840

    accuracy                           0.53      1680
   macro avg       0.53      0.53      0.53      1680
weighted avg       0.53      0.53      0.53      1680

[[492 348]
 [440 400]]
```

Logistic Regression

```
In [71]: 
         logreg = LogisticRegression()

         logreg.fit(X_train, y_train)

         y_pred = logreg.predict(X_test)


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

```
              precision    recall  f1-score   support

           0       0.59      0.56      0.57       840
           1       0.58      0.61      0.60       840

    accuracy                           0.58      1680
   macro avg       0.58      0.58      0.58      1680
weighted avg       0.58      0.58      0.58      1680

[[467 373]
 [325 515]]
```

```
In [72]: logreg.score(X_test,y_test)

Out[72]: 0.5845238095238096
```

```
In [73]: print(classification_report(y_test,y_pred))
         print(confusion_matrix(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.59      0.56      0.57       840
           1       0.58      0.61      0.60       840

    accuracy                           0.58      1680
   macro avg       0.58      0.58      0.58      1680
weighted avg       0.58      0.58      0.58      1680

[[467 373]
 [325 515]]
```

4. K-fold cross Validation

To get an overall better overview of the performance of all the algorithms we can implement k-fold cross validation. In this case I have applied k-fold cross validation on two datasets, the first being the one the model after the train-test split and the second dataset is the one we separated initially in the data preprocessing step to ensure that those rows were unseen to the model we trained in the later steps.

After applying k-fold on the data the model had seen we can see that overall SVM performed the best, then decision trees and then logistic regression. Which is different from our initial observations from the classification reports.

However, after applying k-fold on the unseen dataset we truly know which algorithm performs the best. Thus, in this case logistic regression and SVM performance were the same whereas decision tree was very low.

Thus, after observing all the three means of checking the performance of models, we can rank the best performance as follows:

1. SVM
2. Logistic regression
3. Decision trees

The snippet below represents the implementation of k-fold cross validation:

4. K-Fold cross Validation

Testing the model on the data we trained:

Logistic regression:

```
In [74]:  cross_val_score(LogisticRegression(), X, y,cv=3)
Out[74]: array([0.49973219, 0.50026781, 0.5       ])
```

SVM:

```
In [75]:  cross_val_score(SVC(), X, y,cv=3)
Out[75]: array([0.56400643, 0.55115158, 0.55841372])
```

Decision tree:

```
In [76]:  cross_val_score(DecisionTreeClassifier(),X, y,cv=3)
Out[76]: array([0.52544189, 0.51633637, 0.51500536])
```

Testing the model on the data on unseen data:

```
In [77]: unseen_data = pd.read_csv('validation.csv')
         X1 = pd.DataFrame(unseen_data, columns=["loan_amount","no_of_curr_loans","('Aadhar',)"])
         y1 = unseen_data['loan_default']
```

Logistic regression:

```
In [78]: cross_val_score(LogisticRegression(), X1, y1,cv=3)

Out[78]: array([0.5       , 0.66666667, 0.66666667])
```

SVM:

```
In [79]: cross_val_score(SVC(), X1, y1,cv=3)

Out[79]: array([0.5       , 0.66666667, 0.66666667])
```

Decision tree:

```
In [80]: cross_val_score(DecisionTreeClassifier(),X1, y1,cv=3)

Out[80]: array([0.75      , 0.33333333, 0.33333333])
```

5. Interpretability

The overall interpretation of how well these different models classify loan_default data can be understood by the performance of the various models implemented as well as the features that were used to train these models. The most significant features that had contributed to the loan_default variable was the "loan_amount","no_of_curr_loans" and "Aadhar" attributes as these had the highest chi-squared score, and as well as had the highest information gain, thus we selected all these. The rest of the attributes were either not dependent on the target variable or they were redundant as we discussed that "loan_amount" and "asset_cost" were both correlated so there was no need to add both to the algorithm as that would cause redundancy. Out of all the three attributes the loan amount attribute was the most significant factor in predicting the loan default as it had the highest chi-squared score.

The impact of choosing these features to train the SVM, logistic regression and decision-tree models was overall poor because it gave an accuracy of equal and above 50% in all three cases- whereas good accuracy is usually between 70-90%- even though performing well in the case of SVM and logistic-regression compared to decision tree we cannot say that our models were good enough.

According to my observations and understanding the choice of attributes wasn't the problem causing the poor performance of models, it was the total number relevant attributes used to train the models which were only 3 out of 9(from the overall dataset). The reason behind choosing only 3 attributes were that while implementing the model I experimented with various number of combinations of features and most of them either were redundant or had no significant relationship represented with the target variable during feature selection. Thus, I was left with just the three attributes to train the dataset. Which seemed relevant enough to train the models. And after all, if we observe the k-fold validation of the models on the unseen dataset on code line 78-80 we can see the SVM, and logistic regression gave an accuracy of

0.66 thus we can say that yes, the overall performance of models was poor but for a model trained on 3 attributes is was a little tilted towards okay-good category than the poor-okay category of performance.

Conclusion

In conclusion, after conducting this study and implementing various data preprocessing techniques, feature selection techniques, machine learning algorithms and performing validation. The most significant insight I have gained is that no matter how much we preprocess, clean, apply various algorithms on dataset if we don't have valuable attributes with respect to the target variable in this case of classification problem, we won't be able to train our models well in understanding the dataset. In this case of classification of loan default, we did not have enough number of attributes to start with, the dataset we chose wasn't sufficient to train a good accurate model on as it had very a smaller number of relevant attributes to look at in terms of classifying whether an object is loan default or not. The choice of dataset really has high implications on the value we derive from training models on that dataset.