DS675 Machine Learning

Project

Loan Application Approval Predictor

Ojaswi Ghate (org4)
Dinesh Lankepillewar (dl92)

Dataset URL:
https://www.kaggle.com/datasets/abhishek14398/loan-dataset

Source Code:
https://colab.research.google.com/drive/1IV2h7YE6dDxbeq5aFFT9h8gKwfFQJDVX

Demo Link:
https://www.youtube.com/watch?v=FItqyDsC3nk

Table of Contents

Introduction and milestone 2 report details

Coincidentally, Dinesh and Ojaswi selected the same data from Kaggle and so both of us merged our report to represent our analysis we performed during milestone 2 and below are the details.

This dataset available in loan-dataset contains information about customers who have applied for loans, such as their personal and financial details. Based on this information, we performed classification tasks, where the goal was to predict whether a loan application should be accepted or rejected. We have trained the machine learning models using this dataset which will take the details of the customers as input and output a binary label based on the data we see.

Data details:

Size: 115 feature columns with 39417 data rows

Usability rating: 10.00

Observations from available implementation Kaggle (loan-dataset)

While observing the available implementations **there was only one implementation** that is from a user by name called Allena. She has implemented the project only for exploring data science techniques with the name “Lending Club Case Study” (Link: lending-club-case-study) to check how data can be used effectively to solve business problems like defaulters prediction in Loan Lending club but **she did not apply any ML algorithms.**

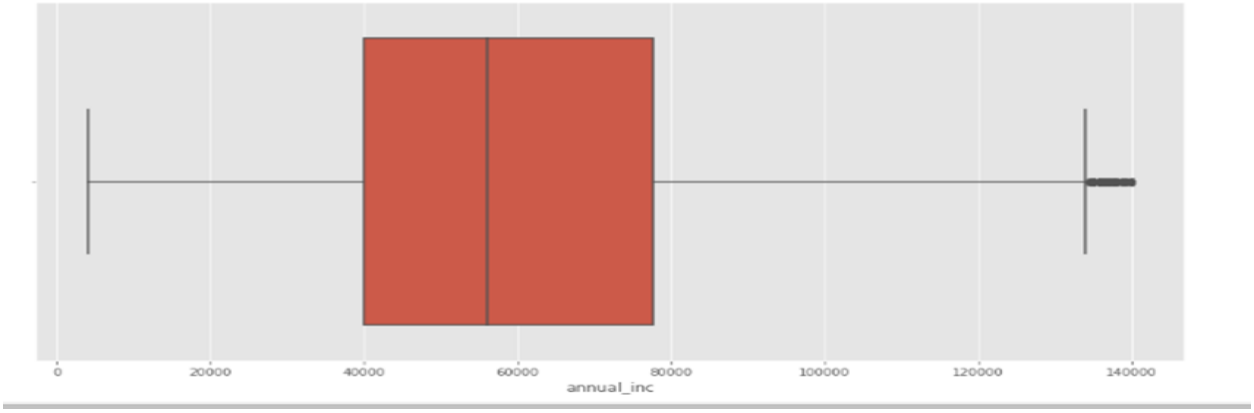
While exploring the data, Allena considered two types of risks are associated with the bank’s decision:

- If the applicant is likely to repay the loan, then not approving the loan results in a loss of business to the company.
- If the applicant is not likely to repay the loan, i.e., he/she is likely to default, then approving the loan may lead to a financial loss for the company.

She had put some great observations after cleaning the data which may help to judge the model results. Some of the observation were as below:

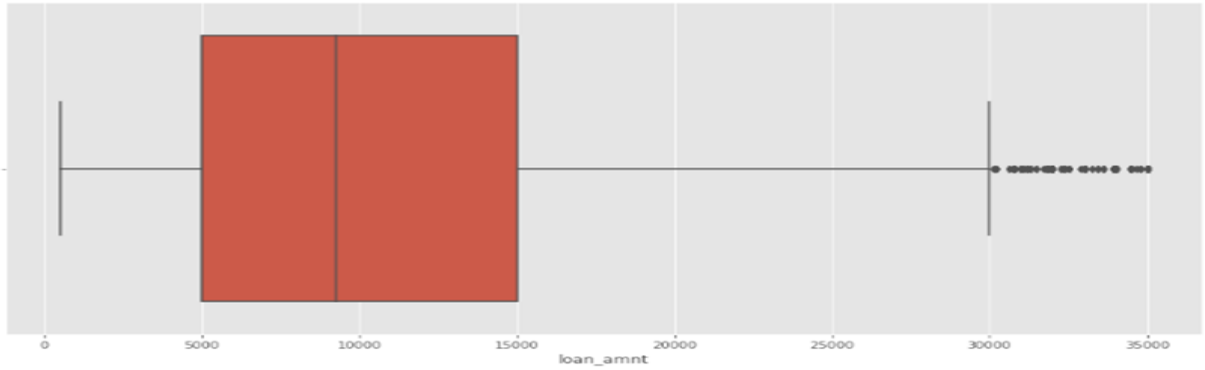
- Income range in available data set after removing outliers that are above 95 percentile:

```
# Drawing Boxplot for Annual Income
plt.figure(figsize=(15,7))
with plt.style.context('ggplot'):
  sns.boxplot(loan_df.annual_inc)
```



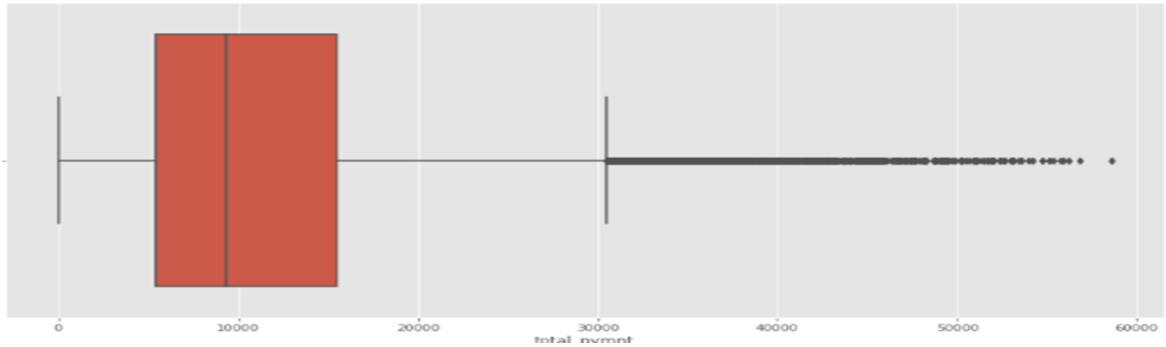
- Her observations on total loan disbursement was as below:

```
#Drawing Box Plot for loan amount
plt.figure(figsize=(15,7))
with plt.style.context('ggplot'):
  sns.boxplot(loan_df.loan_amnt)
```



- Her observations on total payments received as per available data is as below:

```
# Drawing Boxplot for Total Payment
plt.figure(figsize=(15,7))
with plt.style.context('ggplot'):
  sns.boxplot(loan_df.total_pymnt)
```

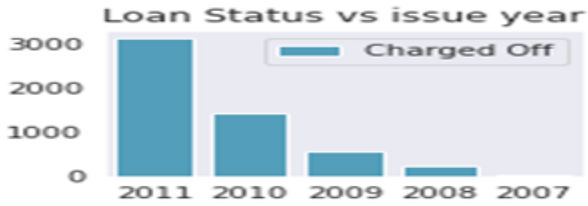


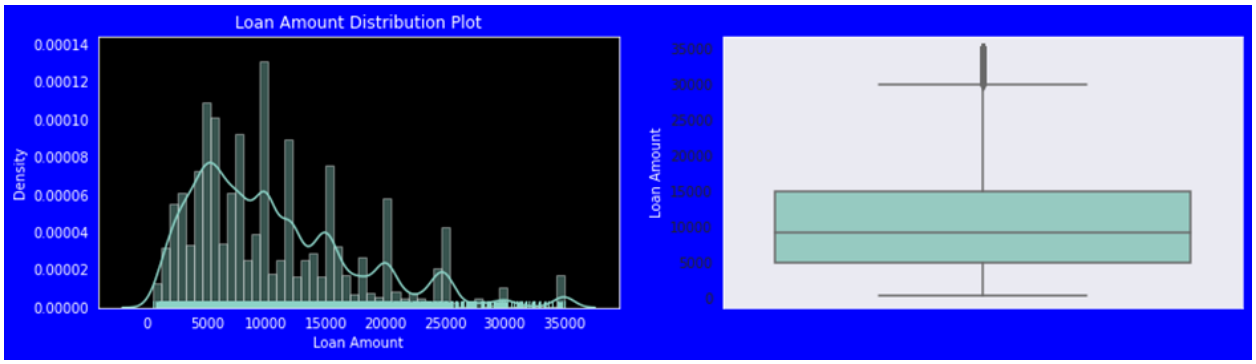
She performed various univariate, bivariate, and multivariate analysis.

- **Univariate Analysis for Loan Amount Variable**

```
## Extracting month and year
df_month_year = loan_df['issue_d'].str.partition("-", True)
loan_df['issue_month']=df_month_year[0]
loan_df['issue_year']='20' + df_month_year[2]
```

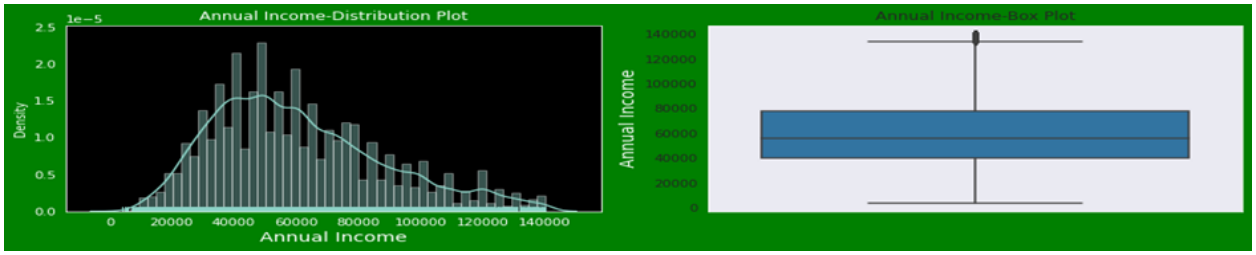
```
plt.figure(figsize=(10,6),facecolor='b')

plt.subplot(221)
ax = sns.countplot(x='issue_month', data=loan_df[loan_df['loan_status']=='Charged Off'],hue='loa
n_status',palette='GnBu_d')
ax.set(title='Loan Status vs issue month')
ax.set_xlabel('issue month',fontsize=14,color = 'w')
ax.set_ylabel('Loan Status',fontsize=14,color = 'w')
ax.legend(bbox_to_anchor=(1, 1))
plt.show()

plt.subplot(222)
ax = sns.countplot(x='issue_year', data=loan_df[loan_df['loan_status']=='Charged Off'],hue='loan
_status',palette='GnBu_d')
ax.set(title='Loan Status vs issue year')
ax.set_xlabel('issue year',fontsize=14,color = 'w')
ax.set_ylabel('Loan Status',fontsize=14,color = 'w')
ax.legend(bbox_to_anchor=(1, 1))
plt.show()
```





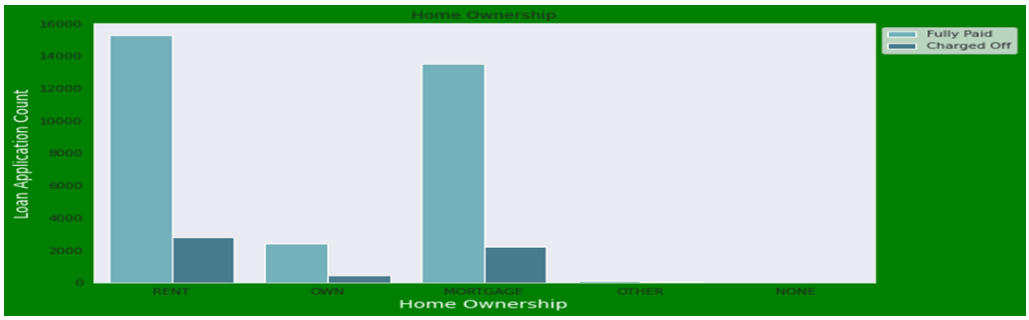
Observations - Most of the loan amounts are in the range 5000-15000

- **Univariate Analysis of Annual Income**



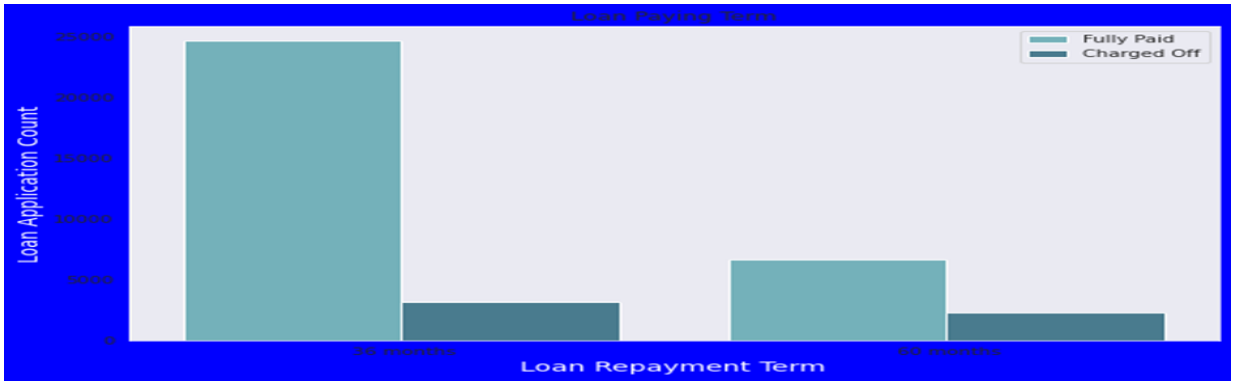
Observation - Most of the Annual Income are in the range 25000-75000

- **Univariate Analysis - Unordered Categorical Variable - Home Ownership**



Observations – The Count plot shows that most of them live in rented homes or mortgaged their home. Applicant numbers are high from these categories so charged off is high too.

- **Univariate analysis- Loan Paying Term**



Observations -

1. Below plot shows that those who had taken loan to repay in 60 months had more % of number of applicants getting
2. charged off as compared to applicants who had taken a loan for 36 months.

This both data statistical analysis and insights drawn from univariate analysis is impressive.

Dataset Statistics

Number of Variables	44
Number of Rows	36642
Missing Cells	1634
Missing Cells (%)	0.1%
Duplicate Rows	0
Duplicate Rows (%)	0.0%
Total Size in Memory	71.7 MB
Average Row Size in Memory	2.0 KB
Variable Types	Categorical: 36 Numerical: 8

Dataset Insights

loan_amnt and funded_amnt have similar distributions	Similar Distribution
interest_rat… has 673 (1.84%) missing values	Missing
dti_categori… has 784 (2.14%) missing values	Missing
emp_length is skewed	Skewed
id has a high cardinality: 36642 distinct values	High Cardinality
issue_d has a high cardinality: 55 distinct values	High Cardinality
title has a high cardinality: 18483 distinct values	High Cardinality
earliest_cr_… has a high cardinality: 522 distinct values	High Cardinality
revol_bal has a high cardinality: 20215 distinct values	High Cardinality
revol_util has a high cardinality: 1082 distinct values	High Cardinality

Bivariate Analysis

Correlation Analysis-Bivariate Matrix

```
loan_correlation=loan_df.corr()
loan_correlation
```

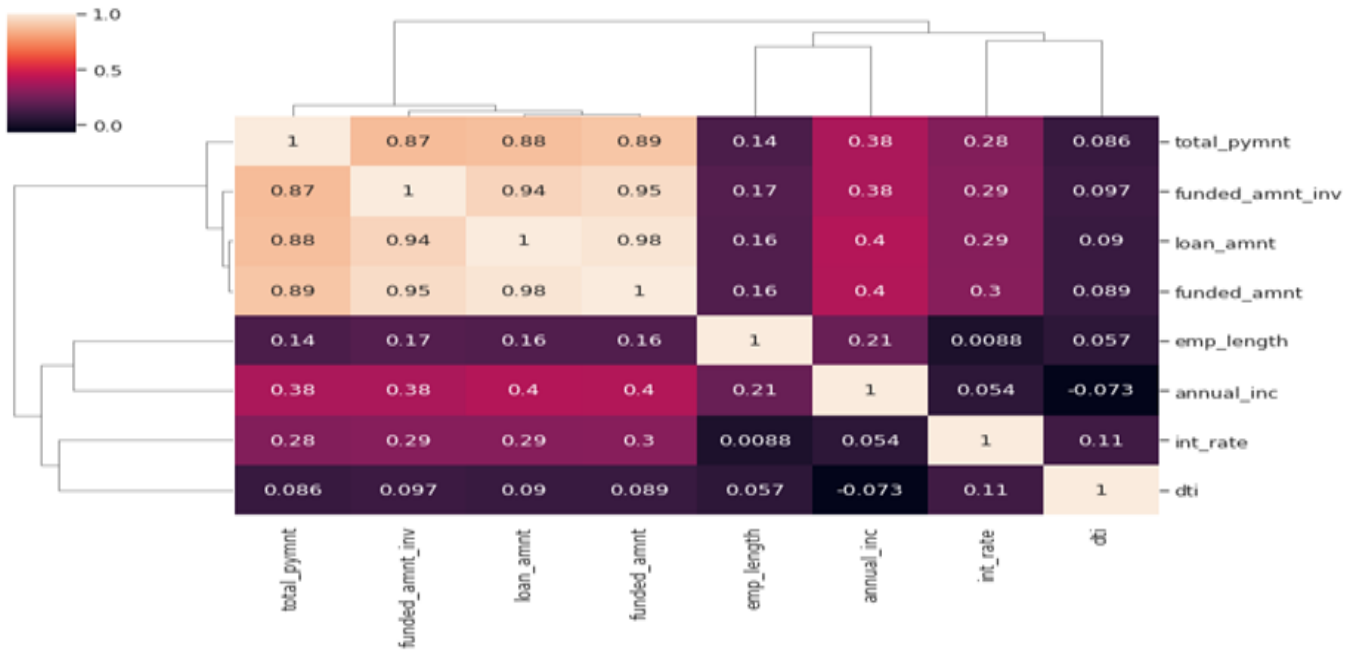
	loan_amnt	funded_amnt	funded_amnt_inv	int_rate	emp_length	annual_inc	dti	total_pymnt
loan_amnt	1.000000	0.981940	0.936901	0.291145	0.157226	0.403358	0.090237	0.876601
funded_amnt	0.981940	1.000000	0.954829	0.296134	0.156766	0.399203	0.089438	0.893239
funded_amnt_inv	0.936901	0.954829	1.000000	0.288743	0.165194	0.382540	0.097037	0.868781
int_rate	0.291145	0.296134	0.288743	1.000000	0.008820	0.053952	0.111587	0.279973
emp_length	0.157226	0.156766	0.165194	0.008820	1.000000	0.213528	0.057098	0.140505
annual_inc	0.403358	0.399203	0.382540	0.053952	0.213528	1.000000	-0.072526	0.382223
dti	0.090237	0.089438	0.097037	0.111587	0.057098	-0.072526	1.000000	0.086089
total_pymnt	0.876601	0.893239	0.868781	0.279973	0.140505	0.382223	0.086089	1.000000

Observations -

1. Annual Income to Debt To Income Ratio i.e. dti are negatively correlated.
2. Loan Amount, Investor Amount and Funding Amount are strongly correlated.
3. Positive correlation between Annual Income and employment years.
4. Positive correlation between annual income and funded amount that means people with high income get a high funded amount.
5. Positive correlation between annual income and total payment.

Heatmap to show correlation amongst various variables

```
sns.set(font_scale=1.1)
sns.clustermap(loan_correlation,annot=True,figsize=(12,8))
plt.show()
```

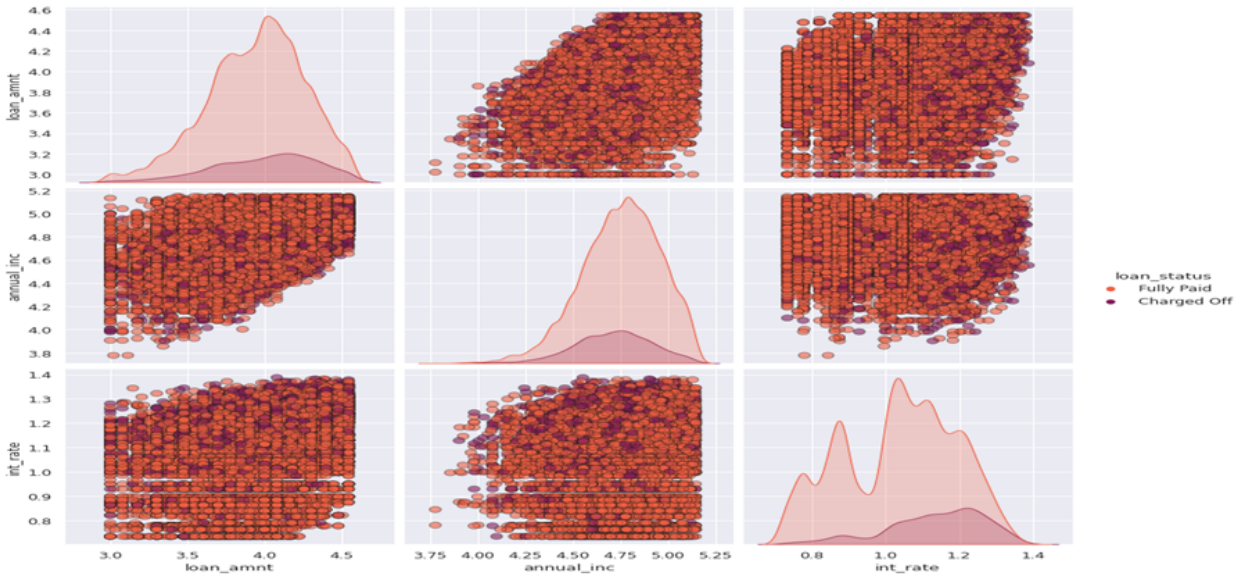


Observations -

1. It is clear that the interest rate is increasing with loan amount increase.
2. probably when loan amount is more it is taken for longer loan term, we saw earlier that longer the loan term more the interest rate.

Multivariate Analysis

```
sns.pairplot(loan_multivariate_dataframe[loan_multivariate_dataframe['issue_year'] == "2011"],
             vars = ['loan_amnt', 'annual_inc', 'int_rate'],
             hue = 'loan_status',
             palette = "rocket_r",
             diag_kind = 'kde',
             plot_kws = {'alpha': 0.6, 's': 80, 'edgecolor': 'k'},
             size = 4);
```



Observations –

1. Higher is the charged off ratio for higher interest rate.
2. Higher the annual income higher the loan amount slightly
3. Interest rate is increasing with loan amount & in increases there is high charged off.

From the above, it is noticed that no one has created any ML models before on this data so there is scope to perform various experiments. We got a great opportunity to implement various ML models. We can implement the model by selecting the features and by implementing the model which can give a decision on a customer's loan application. It is possible to create models using KNN Classifier, Decision Tree Classifier, Perceptron and also with Random Forest Classifier, and we can experiment which classifier renders accuracy.

Data Description

In the data I picked up from Kaggle, there are a total 117 columns and 39718 rows. Out of 117 columns, 116 columns are features and one is label with the name “loan_status” which is used as target variable. Out of 116 columns, below are used as featured to implement the models :

loan_amnt, term, grade, emp_length, home_ownership, annual_inc, verification_status, dti, delinq_2yrs, inq_last_6mths, open_acc, pub_rec, revol_bal, total_acc and application_type.

Description of used column and label is as below:

loan_amnt	The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
Term	The number of payments on the loan. Values are in months and can be either 36 or 60.
Grade	LC assigned loan grade
emp_length	Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
home_ownership	The home ownership status provided by the borrower during registration. Our values are: RENT, OWN, MORTGAGE, OTHER.
annual_inc	The self-reported annual income provided by the borrower during registration.
verification_status	Indicates if income was verified by LC, not verified, or if the income source was verified
Dti	A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.

delinq_2yrs	The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years
inq_last_6mths	The number of inquiries in past 6 months (excluding auto and mortgage inquiries)
open_acc	The number of open credit lines in the borrower's credit file.
pub_rec	Number of derogatory public records
revol_bal	Total credit revolving balance
total_acc	The total number of credit lines currently in the borrower's credit file
application_type	Indicates whether the loan is an individual application or a joint application with two co-borrowers
loan_status	Current status of the loan

Purpose

No one has implemented any model on the dataset we have picked but only performed the data exploration. We took the opportunity to try various machine learning models and check the accuracy of those models.

We have implemented below ML models and divided responsibilities as below :

1. Decision Trees - Dinesh
2. Random Forest - Dinesh
3. K-Nearest Neighbors - Ojaswi
4. Perceptron - Ojaswi

Model implementation

Implementation using sk-learn libraries in python and output of implemented models.

Imported the required libraries :

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier,
plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

Loaded data from the CSV file and selected only the relevant columns for the analysis

```
df =
pd.read_csv('/content/drive/MyDrive/ColabProjectTexts/l
oan.csv', usecols=['loan_amnt', 'term', 'grade',
'emp_length', 'home_ownership', 'annual_inc',
'verification_status', 'dti', 'delinq_2yrs',
'inq_last_6mths', 'open_acc', 'pub_rec', 'revol_bal',
'total_acc', 'application_type', 'loan_status'])
```

Removed rows with missing values

```
df.dropna(inplace=True)
```

Label encoded the categorical features

```
le = LabelEncoder()
df['term'] = le.fit_transform(df['term'])
df['grade'] = le.fit_transform(df['grade'])
df['emp_length'] = le.fit_transform(df['emp_length'])
```

```
df['home_ownership'] =
le.fit_transform(df['home_ownership'])
df['verification_status'] =
le.fit_transform(df['verification_status'])
df['application_type'] =
le.fit_transform(df['application_type'])
```

Created target variable

```
df['status'] = (df['loan_status'] == 'Fully
Paid').astype(int)
```

Selected features are targeted variables

```
X = df.drop(['loan_status', 'status'], axis=1)
y = df['status']
```

Split data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)
```

Decision Tree

Created decision tree classifier and trained it on training data

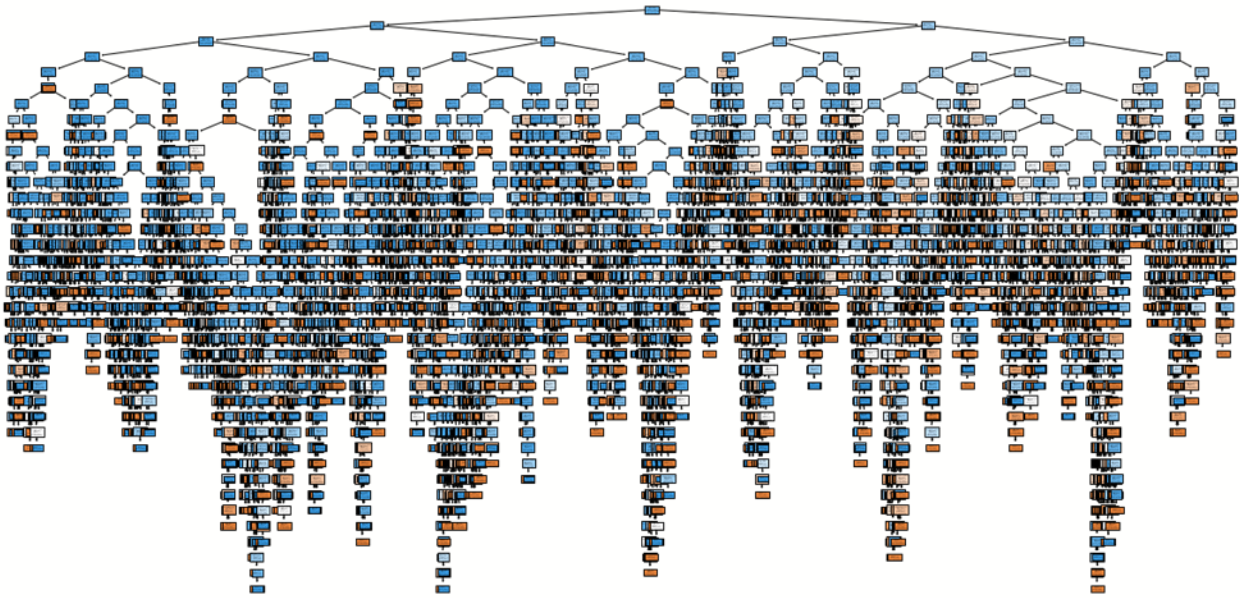
```
dt_clf = DecisionTreeClassifier()
dt_clf.fit(X_train, y_train)
```

Created plot of decision tree

```
plt.figure(figsize=(20, 10))
plot_tree(dt_clf, filled=True, feature_names=X.columns,
class_names=['Not Fully Paid', 'Fully Paid'])
plt.show()
```

Plotted decision tree of data

As the data is huge, it's tough to observe the details of the decision tree but it can be cleanly observed from pictorial data that data available is good for decision making using the decision tree till last node.



Evaluated the performance of the trained Decision Tree Model on the test data

```
y_dt_pred = dt_clf.predict(X_test)
```

Generated a text report showing the precision, recall, F1 score, and support for each class based on the predicted labels and the true labels.

```
print("Decision Tree Report:\n",
classification_report(y_test, y_dt_pred))
dt_cm = confusion_matrix(y_test, y_dt_pred)
sns.heatmap(dt_cm, annot=True, cmap="YlGnBu")
plt.show()
```

Generated a text report showing the precision, recall, F1 score, and support for each class based on the predicted labels and the true labels.

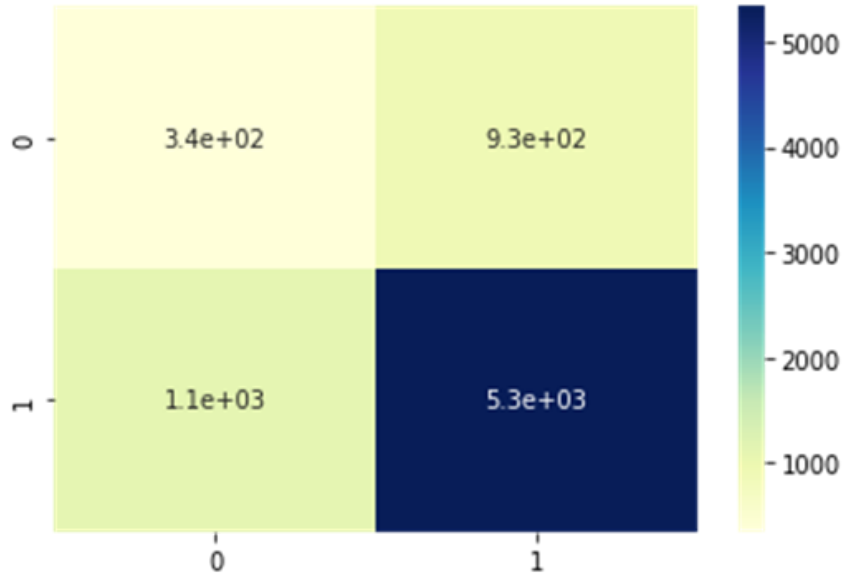
From the decision tree performance report, we can observe that, accuracy of decision tree classifier model is 0.73 i.e. 73% where as obtained precision with weighted average as 75%, recall with weighted average as 73%, f1-score with weighted average as 0.74 and support values with weighted average as 6457 for decision tree classifier and can be seen in below report :

Decision Tree Report:

	precision	recall	f1-score	support
0	0.24	0.27	0.25	1272
1	0.85	0.83	0.84	6457
accuracy			0.73	7729
macro avg	0.54	0.55	0.55	7729
weighted avg	0.75	0.73	0.74	7729

Created a confusion matrix, the rows correspond to the true classes, and the columns correspond to the predicted classes.

Below heatmap is created using the Seaborn library. We can see that from the diagonal of the below matrix which represents the correct predictions made by the decision tree classifier while the off-diagonal elements represent the misclassifications made by the decision tree classifier.



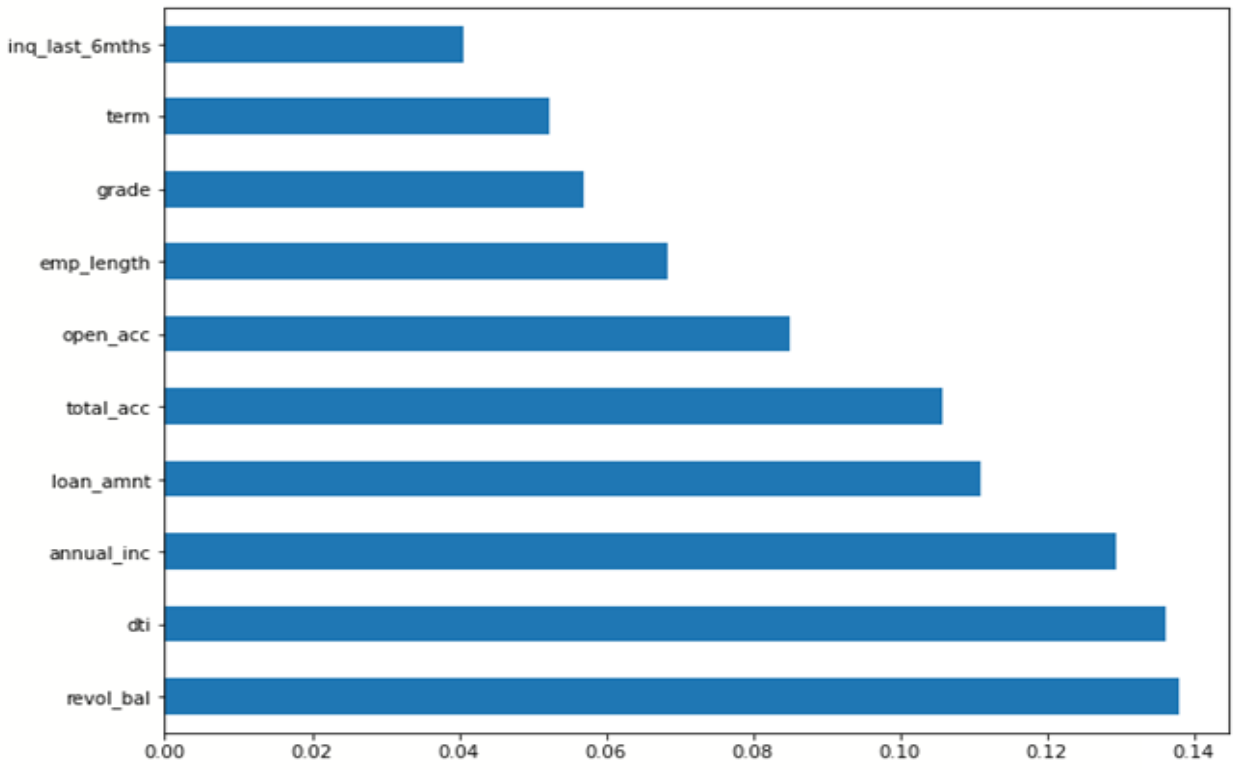
Random Forest

Created random forest classifier and trained it on training data

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

Plot feature importance for random forest

```
plt.figure(figsize=(10, 8))
feat_importances =
pd.Series(rf_clf.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```



From the plot below, we can observe the feature importance scores and we can see the most important feature in the model which expresses the underlying patterns and relationships of the data.

Evaluated the performance of the trained Random Forest Model on the test data

```
y_rf_pred = rf_clf.predict(X_test)
```

Generated a text report showing the precision, recall, F1 score, and support for each class based on the predicted labels and the true labels.

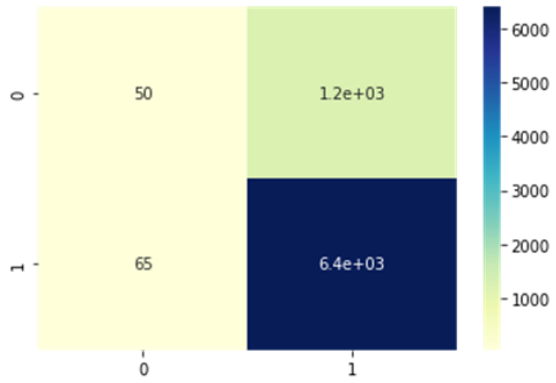
```
print("Random Forest Report:\n",
classification_report(y_test, y_rf_pred))
rf_cm = confusion_matrix(y_test, y_rf_pred)
sns.heatmap(rf_cm, annot=True, cmap="YlGnBu")
plt.show()
```

From the random forest performance report, we can observe that, accuracy of random forest classifier model is 0.83 i.e. 83% whereas obtained precision with weighted average as 77%, recall with weighted average as 83%, f1-score with weighted average as 0.77 and support values with weighted average as 7729 for random forest classifier and can be seen in below report :

```
Random Forest Report:
              precision    recall  f1-score   support

           0       0.43      0.04      0.07      1272
           1       0.84      0.99      0.91      6457

    accuracy                           0.83      7729
   macro avg       0.64      0.51      0.49      7729
weighted avg       0.77      0.83      0.77      7729
```

Created a confusion matrix, the rows correspond to the true classes, and the columns correspond to the predicted classes.

Below heatmap is also created using the Seaborn library. Here also, we can see that from the diagonal of the below matrix which represents the correct predictions made by the random forest classifier while the off-diagonal elements represent the misclassifications made by the random forest classifier.



KNN Classifier

Created KNN classifier object with 5 neighbors and trained it on training data

```
knn_clf = KNeighborsClassifier(n_neighbors=5)
knn_clf.fit(X_train, y_train)
```

Evaluated the performance of the trained KNN classifier on the test data

```
knn_y_pred = knn_clf.predict(X_test)
```

Computed the accuracy of the KNN classifier on the test data

```
print('KNN Accuracy Score:', accuracy_score(y_test,
knn_y_pred))
```

Generated a text report showing the precision, recall, F1 score, and support for each class based on the predicted labels and the true labels.

```
print('KNN Classification Report:')
print(classification_report(y_test, knn_y_pred))
```

Created a confusion matrix, the rows correspond to the true classes, and the columns correspond to the predicted classes.

```
print('KNN Confusion Matrix:')
```

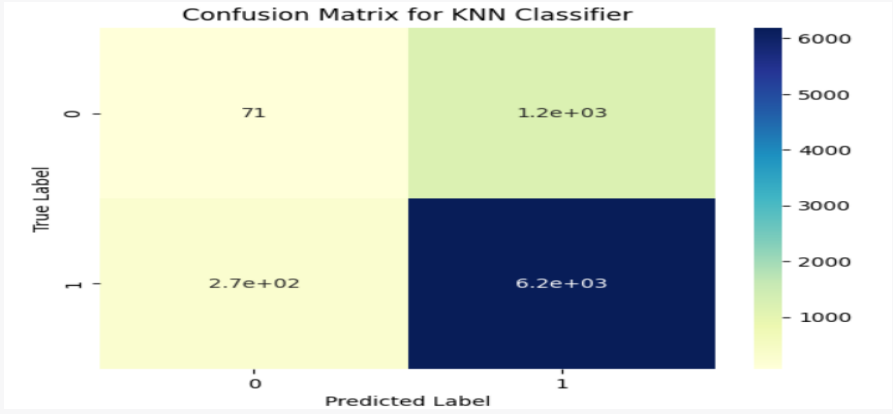
```
print(confusion_matrix(y_test, knn_y_pred))
```

```
KNN Accuracy Score: 0.7170953101361573
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.32      0.16      0.21       320
           1       0.77      0.90      0.83      1002

    accuracy                           0.72      1322
   macro avg       0.55      0.53      0.52      1322
weighted avg       0.66      0.72      0.68      1322

KNN Confusion Matrix:
[[ 50 270]
 [104 898]]
```

Heat Map of KNN Classifier confusion matrix

The heatmap of the confusion matrix is a useful tool for evaluating the performance of the KNN classifier and for identifying areas where the model may need improvement.

```
knn_cm = confusion_matrix(y_test, knn_y_pred)
sns.heatmap(knn_cm, annot=True, cmap="YlGnBu")
plt.title('Confusion Matrix for KNN classifier')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



Perceptron

Created an instance of the Perceptron class

```
perceptron_clf = Perceptron()
```

Trained the Perceptron model on a training set

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

Evaluated the performance of the trained Perceptron classifier on the test data

```
perceptron_y_pred = perceptron_clf.predict(X_test)
```

Computed the accuracy of the Perceptron classifier on the test data

```
print('Perceptron Accuracy Score:', accuracy_score(y_test,
perceptron_y_pred))
```

```
print('Perceptron Classification Report:')
```

```
Perceptron Accuracy Score: 0.8352956397981628
Perceptron Classification Report:
              precision    recall  f1-score   support

           0       0.43      0.00      0.00      1272
           1       0.84      1.00      0.91      6457

    accuracy                           0.84      7729
   macro avg       0.63      0.50      0.46      7729
weighted avg       0.77      0.84      0.76      7729
```

Generated a text report showing the precision, recall, F1 score, and support for each class based on the predicted labels and the true labels.

```
print(classification_report(y_test, perceptron_y_pred))
```

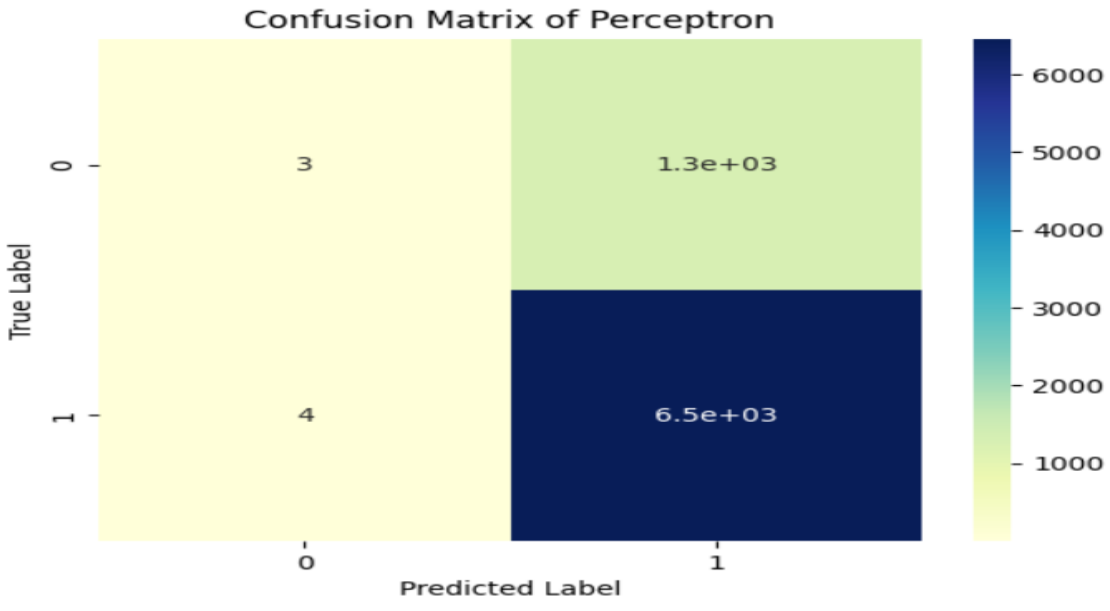
Created a confusion matrix, the rows correspond to the true classes, and the columns correspond to the predicted classes.

```
print('Perceptron Confusion Matrix:')

print(confusion_matrix(y_test,perceptron_y_pre
```

```
Perceptron Confusion Matrix:
[[   3 1269]
 [   4 6453]]
```

Heat Map of Perceptron Classifier confusion matrix

```
pc_cm = confusion_matrix(y_test, perceptron_y_pred)

sns.heatmap(pc_cm, annot=True, cmap="YlGnBu")

plt.title('Confusion Matrix of Perceptron')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()
```



There are more examples of one class than the other, which can cause the classifier to be biased towards the majority class. The classifier should perform well on all classes, not just one or a few.
Therefore our approach for this should be as, oversampling the minority class or undersampling the majority class to balance the number of examples in each class.
With this technique we may improve the performance of the classifier and ensure that it performs well on all classes.

```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=42)

X_train_resampled,y_train_resampled=rus.fit_resample(X_train
, y_train)

perceptron_clf = Perceptron()
```

Trained the Perceptron model on the resampled training set

```
perceptron_clf.fit(X_train_resampled, y_train_resampled)
```

Evaluated the performance of the trained Perceptron classifier on the test data

```
perceptron_y_pred = perceptron_clf.predict(X_test)
```

Computed the accuracy of the Perceptron classifier on the test data

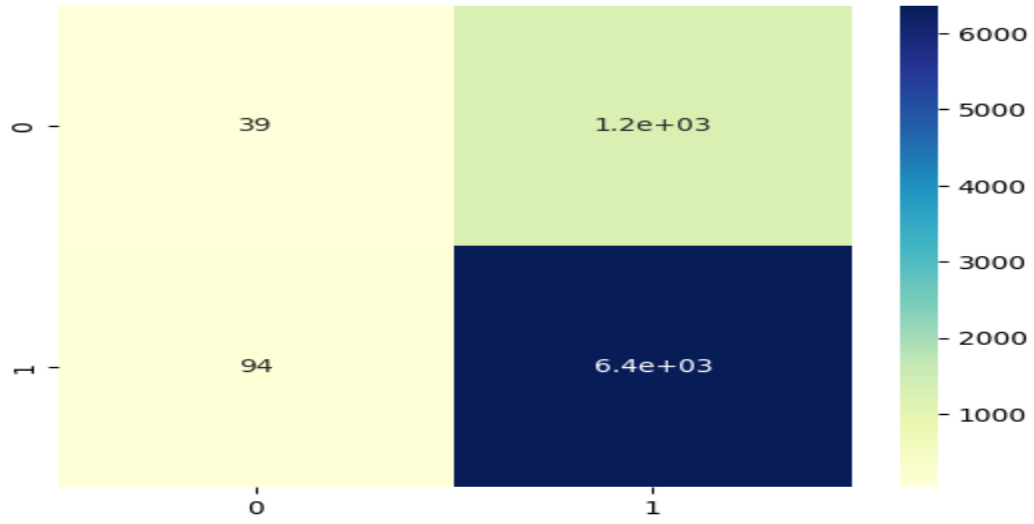
```
print('Perceptron Accuracy Score:', accuracy_score(y_test,
perceptron_y_pred))
```

Printed the classification report for the Perceptron classifier on the test data

```
print('Perceptron Classification Report:')

print(classification_report(y_test, perceptron_y_pred))

pc_cm = confusion_matrix(y_test, perceptron_y_pred)
```

```
sns.heatmap(pc_cm, annot=True, cmap="YlGnBu")

plt.show()
```

```
Perceptron Accuracy Score: 0.8283089662310777
Perceptron Classification Report:
              precision    recall  f1-score   support

           0       0.29      0.03      0.06      1272
           1       0.84      0.99      0.91      6457

    accuracy                           0.83      7729
   macro avg       0.57      0.51      0.48      7729
weighted avg       0.75      0.83      0.77      7729
```



Conclusion

While implementing this project, we have considered four classification models i.e. Decision Trees, Random Forest, K-Nearest Neighbour and Perceptron on the loan classification dataset.

As we can see from the initial data analysis using decision tree plot, the selected features were very distinct and were well separable. We first tried a decision tree model and achieved a reasonable accuracy of 73%, precision with weighted average of 75% and recall with weighted average of 73%.

Secondly, we tried to improve the model by training data on random forest and there we observed significant improvement in the performance compared to decision tree model as accuracy of 83%, precision with weighted average of 77% and recall with weighted average of 83%.

As a third model, we tried K-Nearest Neighbour. It is observed that the performance has accuracy of 81%, precision with weighted average of 73% and recall with weighted average of 81%.

And finally, we implemented a Perceptron classifier for which the performance accuracy we got is 84% and precision with weighted average of 77% and recall with weighted average of 84%. Where as when we performed undersampling technique to perceptron classifier, we got the accuracy of 82%

So, from above, after executing the four different models, we observed that the Perceptron classifier model is the suitable model for this data where as other model like Random Forest or Decision Tree can also be used as its accuracy is nearly same as perceptron one as well as we have scope to improve models further by applying various strategies.