Settle take home assessment

Loan default risk prediction

1. Problem statement:

The LoanData.csv file consists of various features of the borrower like employment length, fico score etc and of the nature of the loan like loan amount, interest rates. Build a loan default risk prediction model which predicts the probability of loan charge off.

Notations:

Model prediction probability = 0: safe account.

Model prediction probability = 1: risky account (the account which has high chance to default the loan).

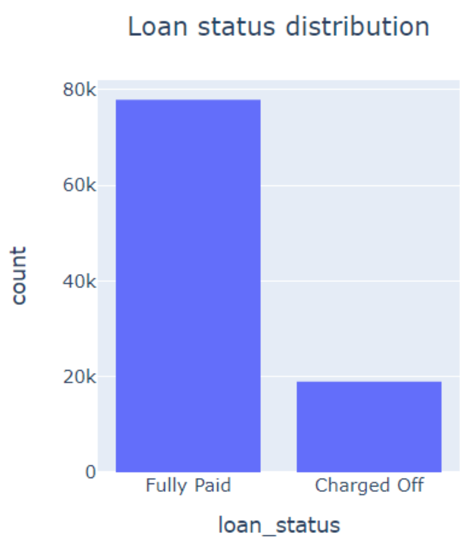
2. Overview of the data and Exploratory data analysis:

'LoanData.csv' – Initially I have tried to get an overview of the data. The loandata.csv file consists of 150 features and close to 100k (99,998) accounts. All the loans are originated between October 2015 and December 2015.

Target label:

The target label is 'loan_status' where "Fully Paid" indicates successful repayment and "Charged Off" indicates a default.

There are 77,883 fully paid accounts and 18,956 charged off accounts.



There are 77883 fully paid accounts and 18956 charged off accounts. 80.4% of the loans are fully paid and 19.6% loans are charged off. The data is skewed. So, if we use accuracy as performance metrics, if the model predicts fully paid every time, the accuracy will be 80%. Which is why I choose to use F1 score as performance metrics and monitor both precision and recall. Using confusion matrix, I will monitor the model performance on both fully paid and charged off categories.

b. Missing values:

14 features have 100 % missing values. Member id, next payment day are empty in all rows and 12 secondary applicant features are missing totally. So I decided to remove these 14 features.

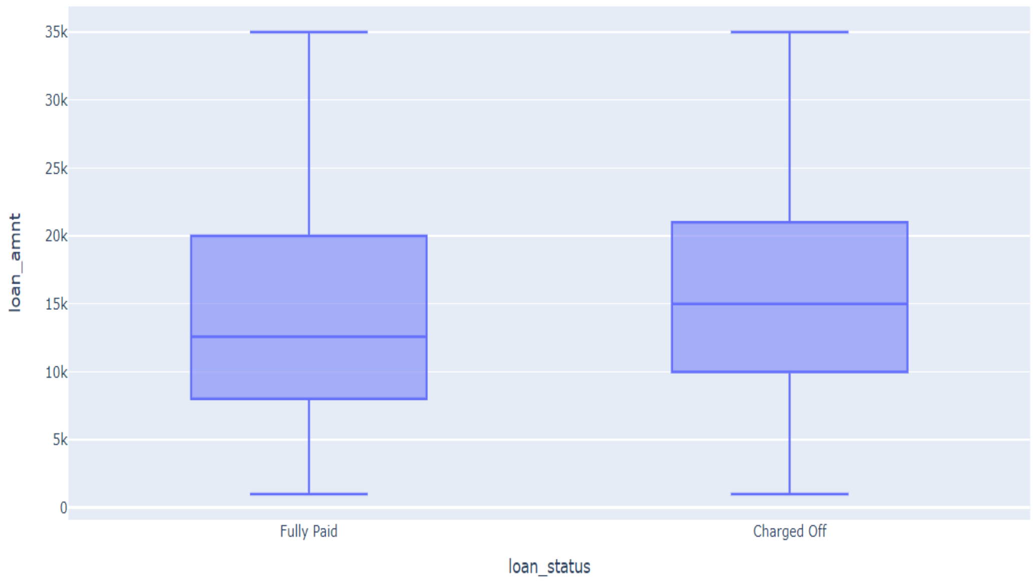
Also hardship based features are missing over 99% of the times. As of now I will remove all hardship related features except hardship_status feature. Hardship_status feature may have correlation with account charge off. If time permits, I will work on other hardship related features.

If there are any previous settlement issues, then the risk involved maybe high. So I will assume 'settlement_status' as 1 if it is not null else 0.

Missing values with more than 90% missing values are removed.

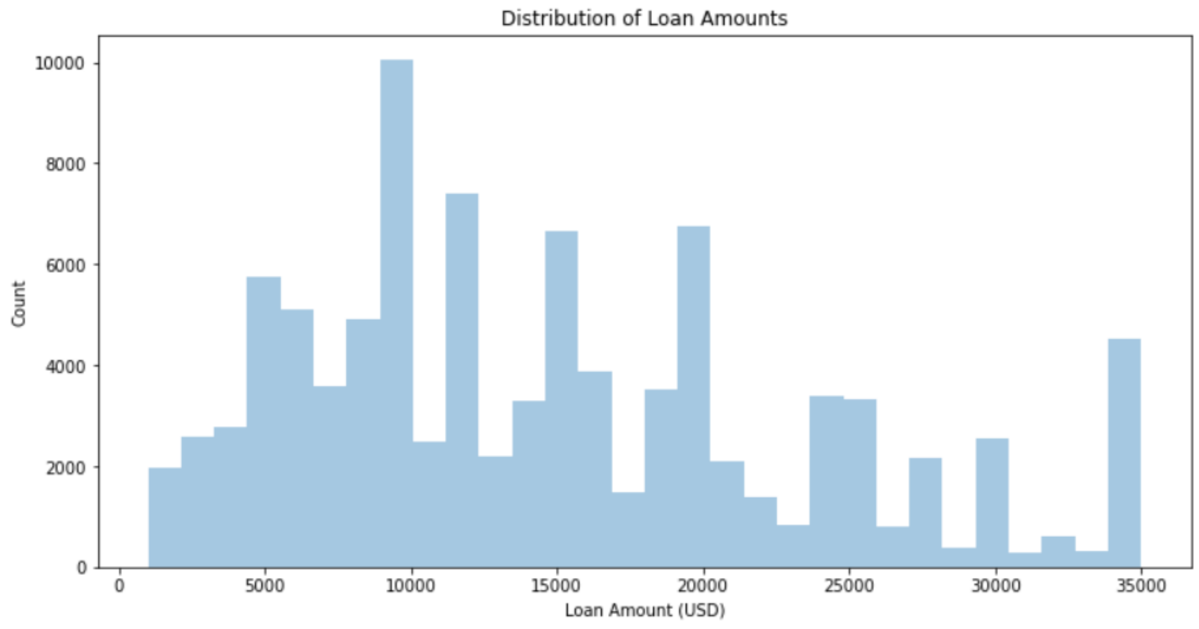
c. Loan amount vs loan status:

The median loan amount of charged off accounts(15k) is a bit higher than median loan amount of fully paid accounts(12.5k).



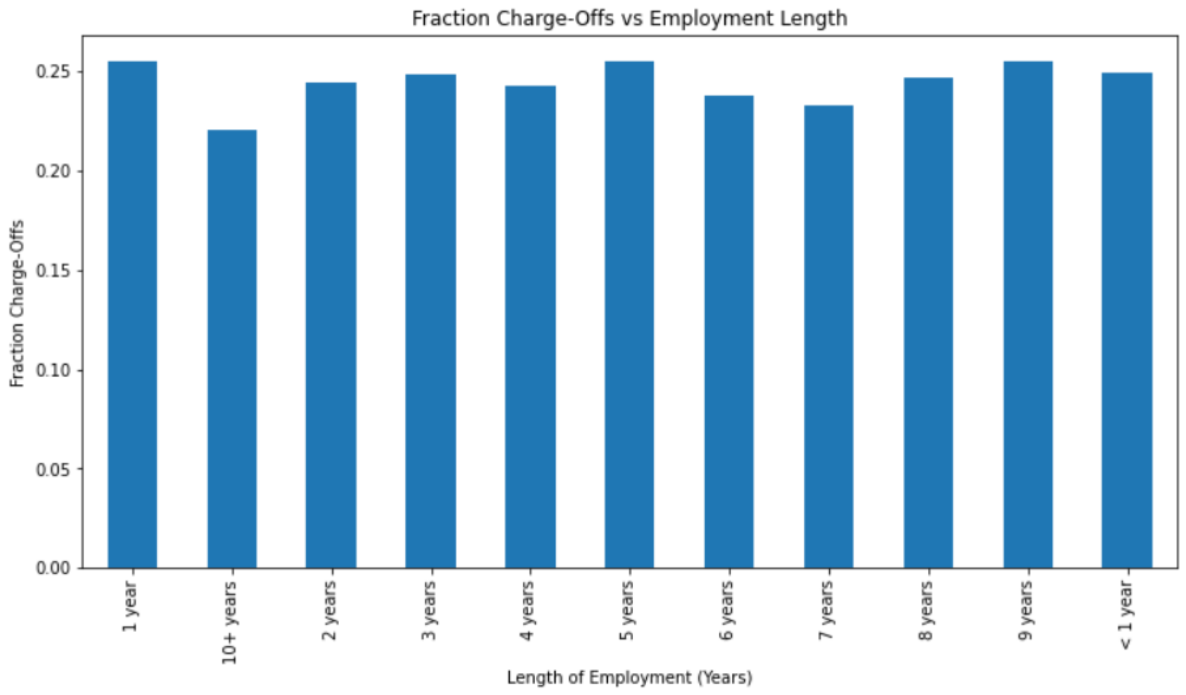
d. Distribution of loan amounts:

In our data, loans of 10k$ are distributed most.



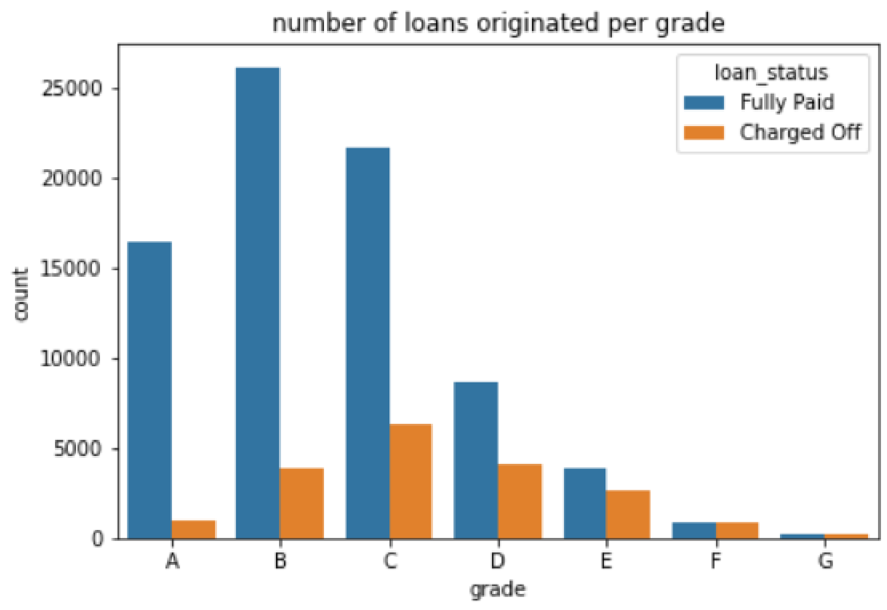
d. Years of employment vs charge off:

People with over 10 plus years of employment have lowest charge off ratio. Also, people with less than 1 year of employment have highest charge off ratio.



e. Number of loans originated per grade:

We can see loans of B grade have more loans originated than any other grade.



3. Data preprocessing:

After removing high missing value features, there are 113 features left. Of these, there are 48 features are with integer and 41 features are with float datatype. There are over 24 features with object datatype which can either be string or dates.

a. Features with 'object 'datatype:

Object datatype consists of both string and date-based features. So, I have built a object feature generator which preprocess the features based on the content present.

Some of the preprocessing tasks performed:

i.) Feature 'term' indicates number of loan installments and consists of '36 months' or '60 months'. Strip months and convert to int.
# Example: '36 months' -> 36, '60 months' -> 60

ii.) Remove '%' symbol for features like interest rate(int_rate) and revolving utilization percentage(revol_util).

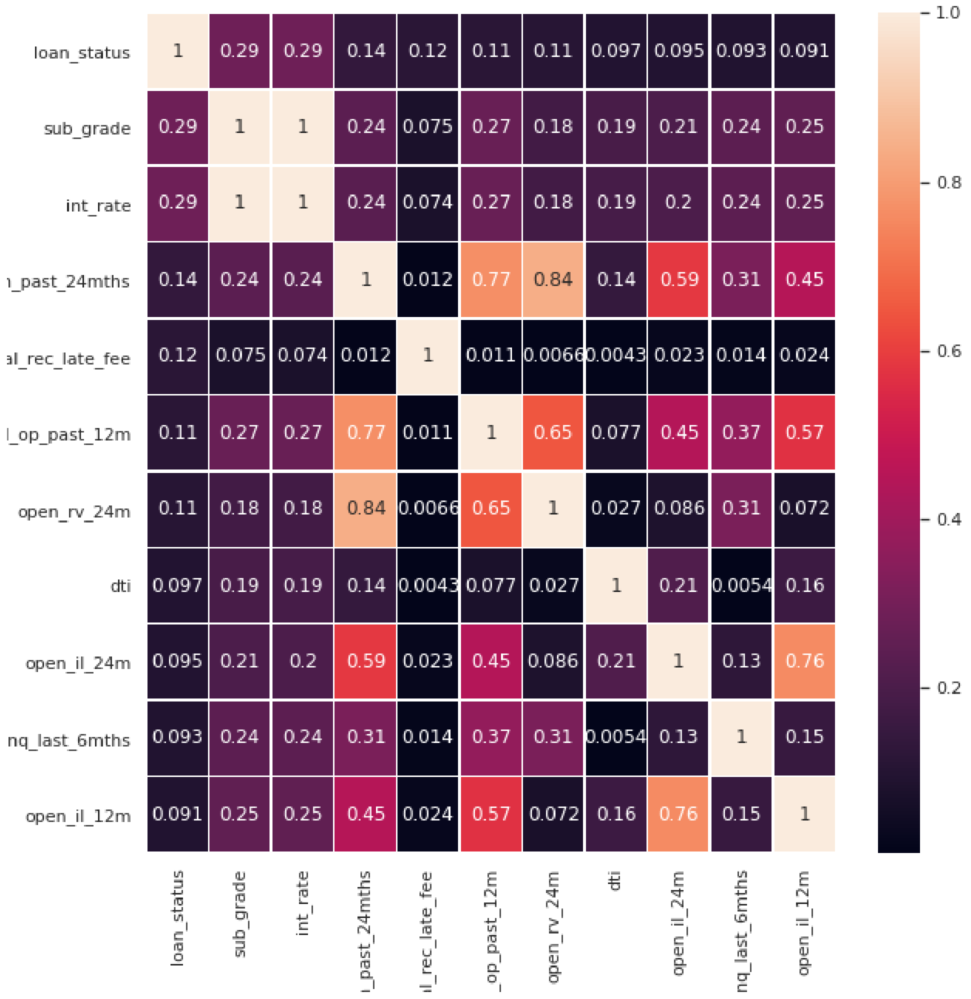
iii.) Convert features like subgrade to ordinal values using label encoding. (A is considered as best grade and G is considered as lowest grade).

iv.) Extract employment length from text feature ‘emp_length’. < 1 year is assumed as 0 and 10 + is assumed as 10 years of experience.

v.) One hot encode categorical features like ‘purpose’ which indicates the purpose of the loan like debt consolidation etc.

b. Correlation:

I observed few features are highly correlated with target feature ‘loan status’. Later I realized many features like recoveries, collection recovery fees, settlement related features will not be available during loan origination. So, these features will not be available at initial stages or during loan initiation. So, I removed features like settlement status. I observed that grade of the loan and interest rates are highly correlated with loan status.



c. Treating missing values:

There are 29 features which have null/missing values. I used different techniques like mean imputing, filling missing values with zero etc depending on the feature. Of the 100k loans, few features like 'dti'(debt to income ratio) and 'revol_util'(revolving utilization) has missing values less than 50. So I imputed them with zero. Also many month related features like months since last delinquency have many missing values. I assumed they are null as those accounts are never defaulted before and so I imputed the month related with zero.

d. Dimensionality reduction:

I noticed many features are identical to others or multicollinear. For example, feature 'loan_amnt' is same as 'funded_amnt' for all the loan accounts. So I identified such features and deleted such features.

e. Removal of multiple input features like settlement, recoveries while building the model:

I realized many features shouldn't be used while training the model. Features like recoveries, settlement, last payment etc occurs after distribution of the loan. For example, if settlement feature is set as true, it is pretty obvious that the loan is defaulted. I used settlement and recovery features initially to build the model and I got an F1 score of 0.98(precision = 0.99 and recall = 0.98) and accuracy of 99%. I realized using just 7 features like recoveries, we can achieve an F1 score of 0.98. But during loan origination, we will not have information about recovery or settlement information of current loan as these events occur later. So I removed those features and started building the model again.

Some of the features removed: debt settlement flag, debt settlement flag date, settlement date, Settlement amount, Settlement percentage, Settlement term, last payment amount, last payment date,recoveries etc.

4. Model selection:

Model building- As we have to classify whether each loan will be defaulted(1) or paid perfectly(0) , I choose this as binary classification problem.

Evaluation metrics- I used macro f1 score as base evaluation metrics. As the data is skewed, using f1 and confusion matrix, we can monitor per category performance. I evaluated model accuracy, precision and recall along with F1.

Train test split- As there is no explicit testing data provided, I used train_test_split to create separate train, validation and test data sets. Throughout modeling, I used train_test_split ratio of 0.2 and random_state=9 to obtain consistent results. There are 77471 train samples and 15494 test samples.

Model selection-

My idea was to start small and build a logistic regression model and see how it performs on the data. This serves as baseline model. Later depending on the performance, I will move to more complex models like xgboosting which performs well on tabular data format. Depending on it's performance, I will try to do hyperparameter tuning or try out different techniques like doing advanced feature engineering like using domain knowledge to create more features etc to improve the model performance.

5. ML modeling:

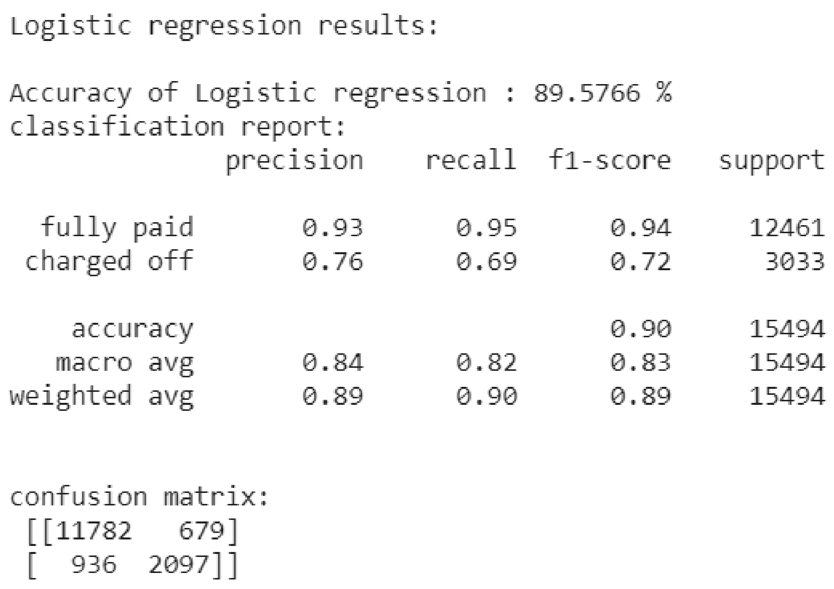
ML models:

Initial I have built simple base line models to check how it fits the data.

i.) *Logistic regression:*

Logistic regression is performing decently on the data. We got an accuracy of 89%. It is predicting fully paid accounts well (f1 score of 0.94).

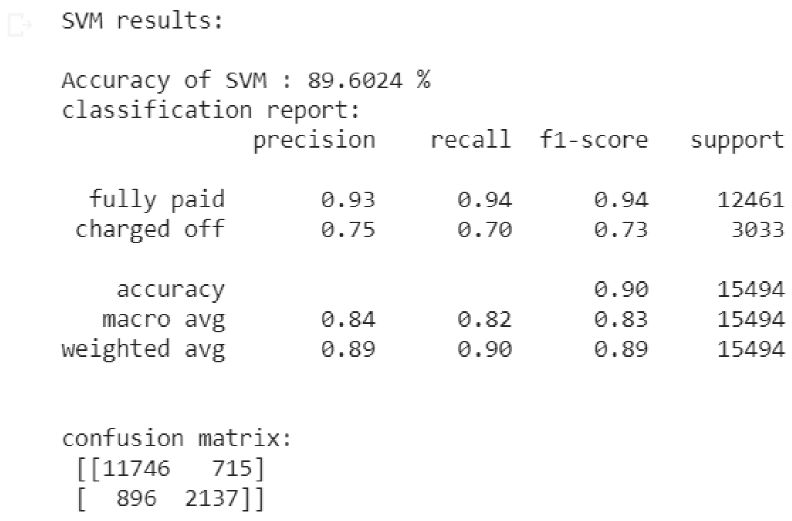
We can notice, out of 12461 fully paid accounts, the model is able to predict 11782 correctly. Whereas out of 3033 charged off accounts, it is able to predict 2097 well and it missed 936 accounts. It resulted in recall of 0.69 for charged off accounts. To improve on recall, I tried out more complex models.



```
Logistic regression results:

Accuracy of Logistic regression : 89.5766 %
classification report:
              precision    recall  f1-score   support

  fully paid       0.93      0.95      0.94     12461
 charged off       0.76      0.69      0.72      3033

    accuracy                           0.90     15494
   macro avg       0.84      0.82      0.83     15494
weighted avg       0.89      0.90      0.89     15494


confusion matrix:
 [[11782   679]
 [  936  2097]]
```

ii.) SVM:

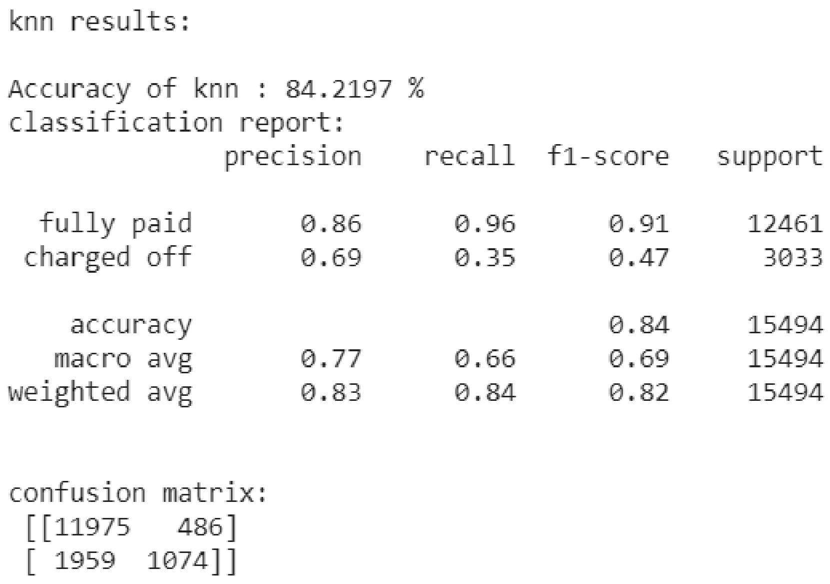
SVM is performing almost similar to logistic regression with an accuracy score of 89.6% and F1 score on fully paid accounts is 0.94. There is slight improvement on recall value on charged off accounts. Recall improved from 0.69 to 0.7.



```
SVM results:

Accuracy of SVM : 89.6024 %
classification report:
              precision    recall  f1-score   support

  fully paid       0.93      0.94      0.94     12461
 charged off       0.75      0.70      0.73      3033

    accuracy                           0.90     15494
   macro avg       0.84      0.82      0.83     15494
weighted avg       0.89      0.90      0.89     15494


confusion matrix:
 [[11746   715]
 [  896  2137]]
```

iii.) knn:

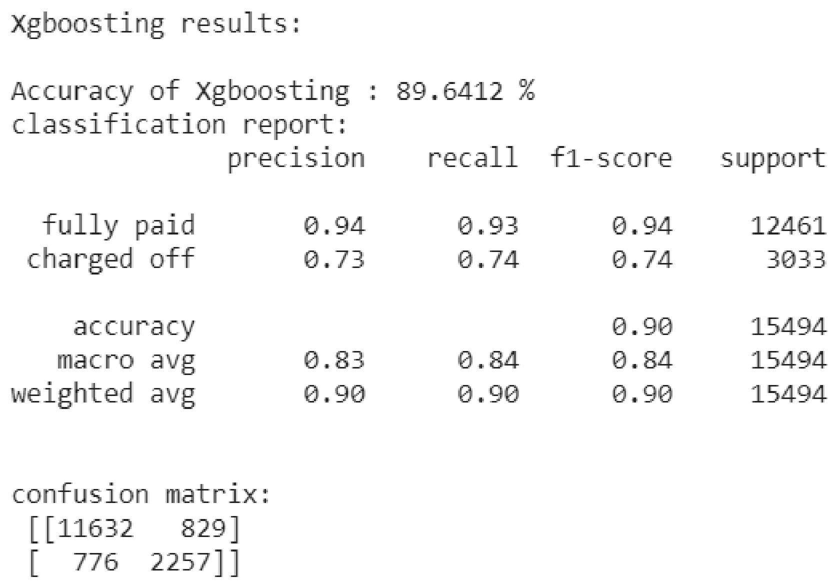
KNN seems to perform worser than logistic regression and SVM. The accuracy dropped to 84% and Recall value of charged off accounts is 0.35. The reason is it is suffering from curse of dimensionality. As there are more than 100 features, the search space increases and knn often stumbles in such instances as all points will be almost equidistant in higher dimensional models.



```
knn results:

Accuracy of knn : 84.2197 %
classification report:
              precision    recall  f1-score   support

  fully paid       0.86      0.96      0.91     12461
 charged off       0.69      0.35      0.47      3033

    accuracy                           0.84     15494
   macro avg       0.77      0.66      0.69     15494
weighted avg       0.83      0.84      0.82     15494


confusion matrix:
 [[11975   486]
 [ 1959  1074]]
```

iv.) *Xgboosting:*

I moved to Xgboosting which often works best on tabular data. Vanilla Xgboosting hasn't improved accuracy much(89%) but improved recall value on charged off accounts significantly(0.74). Also features like interest rate are impacting the model performance more.



```
Xgboosting results:

Accuracy of Xgboosting : 89.6412 %
classification report:
              precision    recall  f1-score   support

  fully paid       0.94      0.93      0.94     12461
 charged off       0.73      0.74      0.74      3033

    accuracy                           0.90     15494
   macro avg       0.83      0.84      0.84     15494
weighted avg       0.90      0.90      0.90     15494


confusion matrix:
 [[11632   829]
 [  776  2257]]
```

v.) *Xgboosting with Polynomial features:*

To further improve model performance, I have built polynomial features by varying degree of the polynomials. I have built polynomials of degrees 2 and I checked how these features are impacting the xgboosting model. In general, polynomial features capture the signal between multiple features and can improve model performance. On this data, I haven't found any significant improvement. So I moved to Hyperparameter tuning.

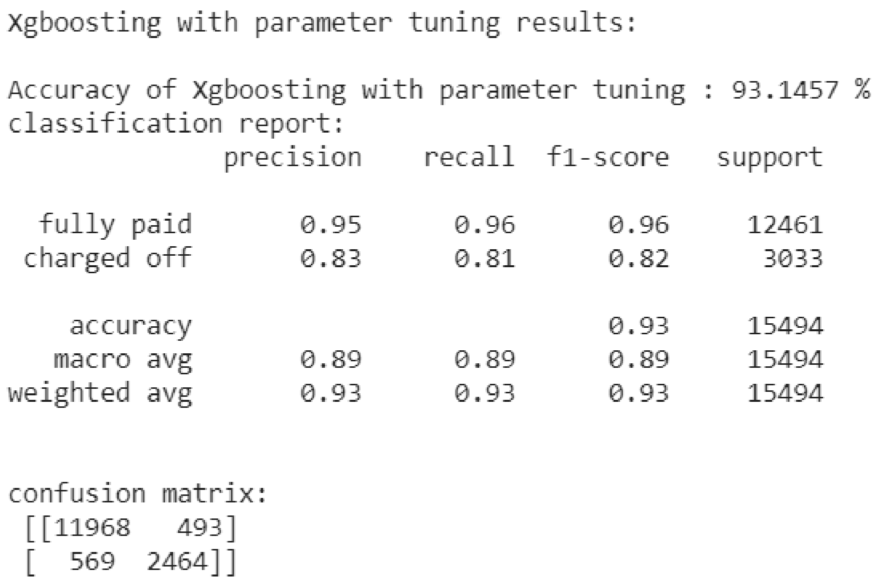
6. Hyperparameter tuning:

As there is still lots of scope to improve the model capacity, I turned on to hyperparameter tuning. Initial idea was to use a combination of RandomSearchCV and GridSearchCV. Due to time constraints, I limited to grid search.

Some of the parameters I tuned are like number of estimators or trees, maximum depth of the tree, minimum child weight, regularization constant etc. After hyperparameter tuning, the accuracy increased to 93% and recall value on charged off accounts bumped up to 0.81.

Parameter	Cross validation input	Best parameter identified
Min_child_weight	[3,4,5]	5
Gamma	[0.4,0.5,0.6]	0.5
Learning_rate	[0.05,0.1,1]	0.1
Max_depth	[5,6,7,8]	7
N_estimators	[100,200,300]	300

Table 1. Hyperparameters



```
Xgboosting with parameter tuning results:

Accuracy of Xgboosting with parameter tuning : 93.1457 %
classification report:
              precision    recall  f1-score   support

  fully paid       0.95      0.96      0.96     12461
 charged off       0.83      0.81      0.82      3033

    accuracy                           0.93     15494
   macro avg       0.89      0.89      0.89     15494
weighted avg       0.93      0.93      0.93     15494


confusion matrix:
 [[11968   493]
 [  569  2464]]
```

7. Model performance:

Base line model	accuracy	f1-fully paid	f1-charged off	Recall-fully paid	Recall-charged off
Logistic regression	89.60%	0.94	0.72	0.95	0.69
SVM	89.60%	0.94	0.73	0.94	0.70
knn	84.2%	0.91	0.47	0.96	0.35
xgboosting	89.64%	0.94	0.74	0.93	0.74
Xgboosting,polynomial features	89.6%	0.94	0.74	0.93	0.74
Xgboosting-after hyperparameter tuning	**93.14%**	**0.96**	**0.82**	0.96	**0.81**

Table 2. Comparison of ML models

8. Conclusion:

The main moto of the problem is to predict the loan charge off accounts. The data has 100k loans originated between oct 2015 and dec 2015. The loans which are paid properly are labelled 'Fully paid' and default loans are labelled as 'charged off'. I found 80% of the loans are fully paid and the rest are charged off accounts. So I used scale_pos_weight parameter later in Xgboosting to handle the data imbalance. I have done some Exploratory data analysis to get an overview of the data. I have built the model as a binary classification model where 1 refers charge off and 0 refers the account will be fully paid. I observed different distributions of the data, how they are responding to the loan status, correlation between different input features and output feature etc. Based on these, I moved to data preprocessing. There are 150 features. Of these I removed multiple features initially. Over 14 features have 100% missing/null values. Many features have zero variance or close to zero variance. I removed such features. Many features are related to closing events or later events like loan recovery, loan settlement status etc which are also removed. If I include one of the features related to later events like recovery amount, I am getting close to perfect predictions (accuracy of 99% and F1 score of 0.98).

As next steps, I generated a text feature generator which extracts features from text data. Some of the things done are like removing special characters like '%' symbol in interest rate, '>, <, +' symbols in employment length (+10 years) etc. I label encoded few features which have ordinal relationship like grades, subgrades. Grade A is considered better than grad F. I one hot encoded some features which doesn't have any ordinal relationship like purpose of loan origination (debt consolidation, car loan etc.). Also, I removed features which are multicollinear. I imputed some missing values in features with appropriate values.

Later I moved to model building. I used logistic regression as a baseline model using which we got an accuracy of 89% and macro F1 score of 0.82. One thing I noticed is the model is predicting well for fully paid accounts whereas it is stumbling on predicting charged off accounts perfectly. So I decided to concentrate on recall value of charged off accounts. The recall on charged off accounts using regression is 0.69. I then tried out SVM which performed slightly better than logistic regression model. Knn performed worse than both the models because of curse of dimensionality(recall is 0.35).

I then choose Xgboosting which often works well on tabular data. It performs well due to multiple factors like building the tree in more regularized manner, using ensemble of models(boosting), sparse awareness and weighted quantile search. Vanilla Xgboosting perfomed better than logistic regression, SVM and knn. The accuracy is same(89%) but macro f1 score improved to 0.84 and the recall on charged off accounts improved to 0.74. It is increased by 0.5 when compared to logistic regression. So I decided to work more on Xgboosting and I initially tried polynomial features with a degree of 2. In general, the interaction terms of polynomials boost the performance. But on our data, I couldn't find much difference. So I later moved to hyperparameter tuning. I used grid search cv. I finetuned multiple parameters like learning rate, max tree depth, minimum child weight etc. I also finetuned scale_pos_weight to handle data imbalance. Using tuning, the accuracy improved from 89% to **93%**. Macro F1 score improved from 0.84 to **0.88**. Recall on charged off accounts increased from 0.74 to 0.81 which is a significant improvement.

There is still lots of scope for model performance improvement. But given time constraints, I stopped going further. These are some of the places of improvement:

1) I removed missing features above threshold of 0.97. I would love to see if there are any patterns in these missing features. They may have important relations with the output data.
2) Use Bayesian optimization instead of grid search cv to perform hyperparameter tuning. But in general Bayesian optimization takes more time. Also I want to try out more broad range of hyperparameters. Due to time constraints, I limited to basic search.
3) Do better feature engineering. There are over 150 features. Feature engineering will be time consuming, but proper feature engineering will almost always improve the model performance. I would love to go over each feature and see how to use it better in model building. I haven't used any domain based knowledge and it often helps in building better models.
4) Use neural network variants and see whether they perform better than traditional ML models like regression and boosting. Building neural network variant from scratch will take significant amount of time, so I limited to xgboosting.
5) Also, I think more data might have been better. Many features seem to be very close between fully paid and charged off accounts. So more data might have provided better signal or distinguished what traits of an account will more likely to be charged off.