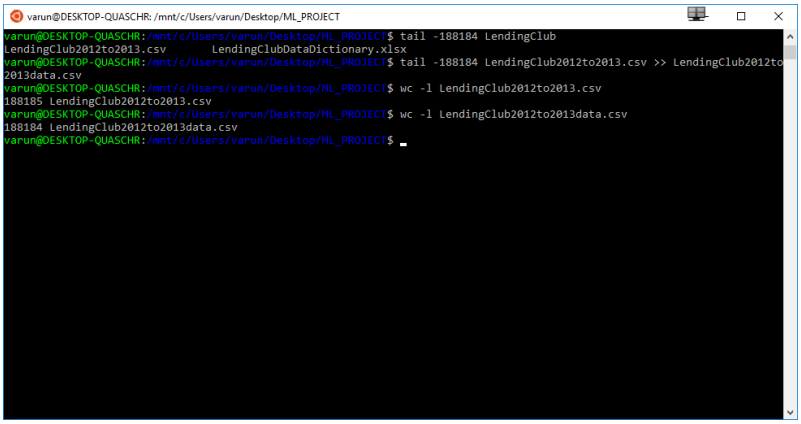
Remove first row from the file to make it ready for processing:

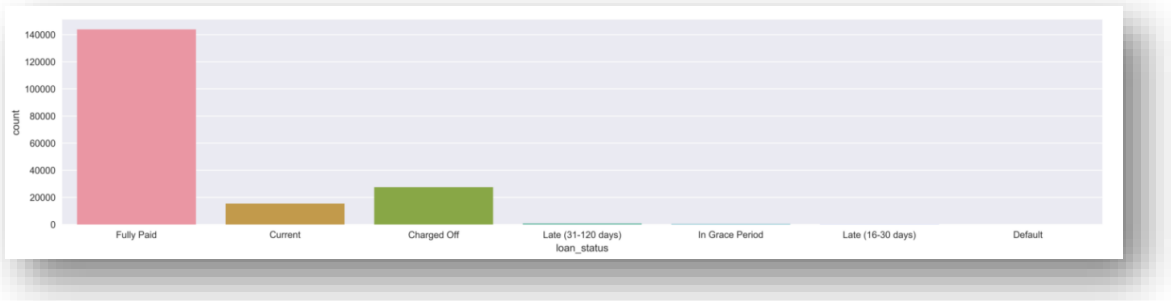


1. **Perform a through feature analysis of the dataset. For instance, what is the distribution of each feature? If the feature is categorical, plot a histogram to show the feature distribution. If it is continuous, use appropriate plots to show the distribution. Include this analysis in your report. Use your intuition to also explain which features are best for classification. Propose new features not in the dataset that you think will be useful. This should be a good 2-3-page analysis of the dataset.**

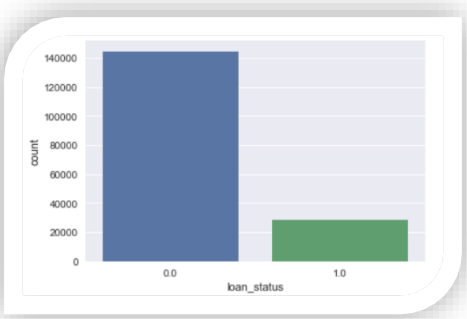
After importing the data, after initial pre-processing, we have visualized both continuous and categorical features:

A] **Target Variable (Problem definition)**:

- We define our problem as detection of bad loans. Lending club can use this classifier in order to identify the loan applications with higher probability of negative outcome for that loan.
- We are dropping rows corresponding to missing values in variable 'loan_status'
- For this reason, we can visualize the distribution of our dependent variable 'loan_status'

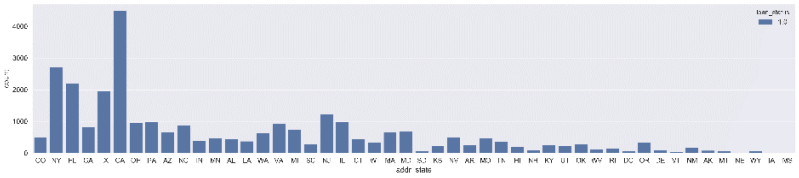


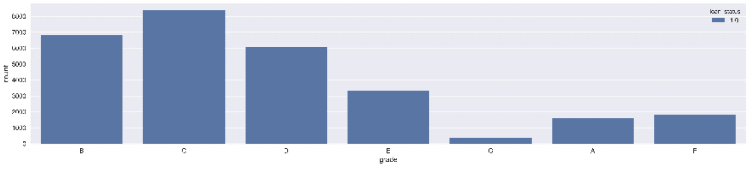
- It appears that we have multiple categories for loan status in our data.
- As lending club, we are defining our problem as identification of 'Good' or 'Bad' loans.
- We have mapped categories fully paid, and in grade period as "Acceptable" and considering these as 'Good loans'.
- We are ignoring loans with "Current" status as we do not know their outcome yet and they do not provide any useful information in order to train our classifier.
- Categories, Charged Off, Default and Late are considered 'Bad' (Unacceptable loans)
- After this mapping, we can visualize distribution for our independent variable:

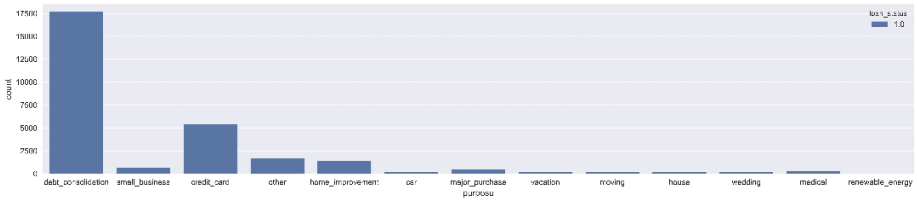


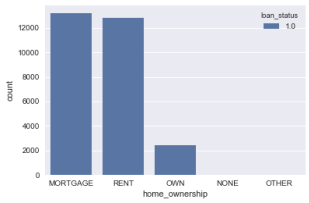
In this Figure,
1 – Unacceptable status
0 – Acceptable Status (Good loan)
It is evident that our dependent variable is highly imbalanced with very less proportion of bad loans. Since this will keep our classifiers biased towards prediction of good loans, we must handle this problem. We will use Oversampling procedure on this variable in order to balance it. (With current number of data points, it is preferable to under-sampling)

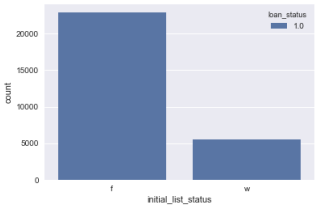
B] Visualization of Categorical features under consideration of bad loans:

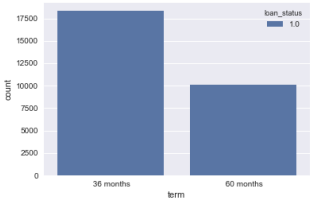


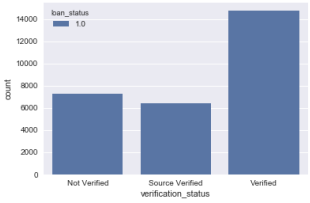




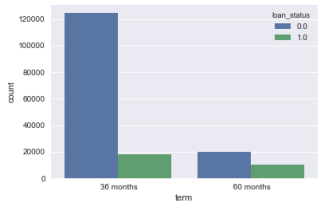


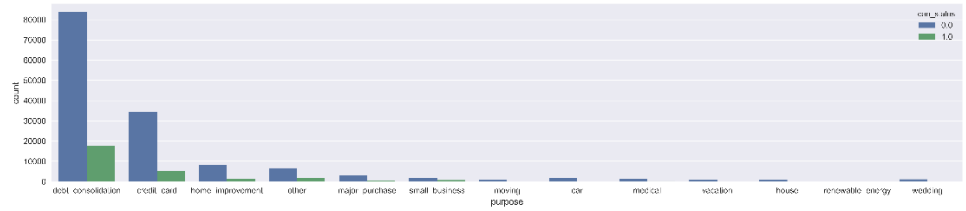


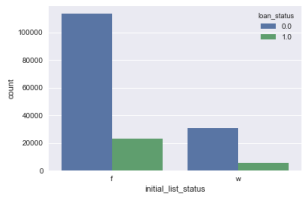


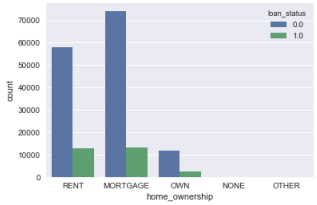


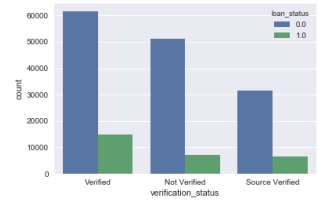
C] Visualization of Categorical features under consideration (Comparison of both categories):

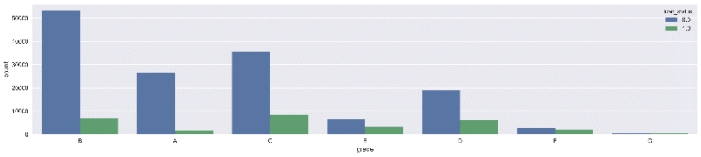


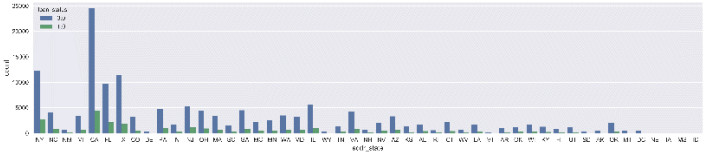




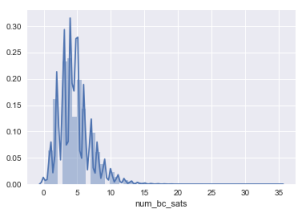


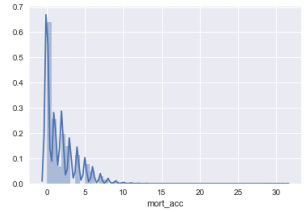


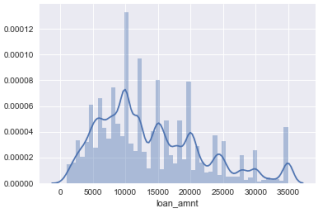


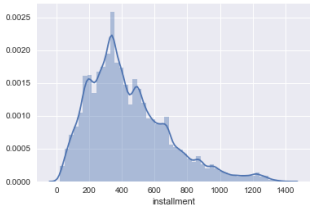


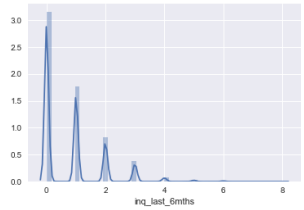
D] Visualization of Continuous(Numeric) features under consideration:

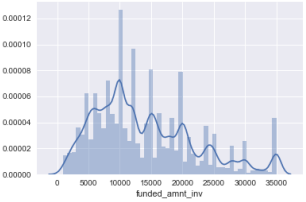


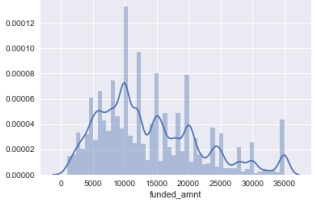


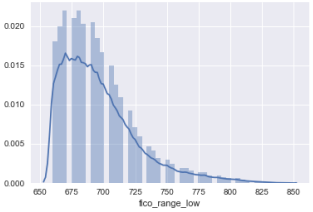


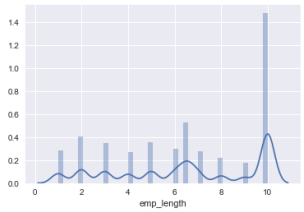


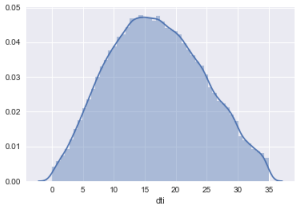


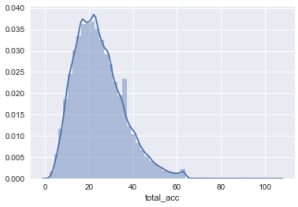


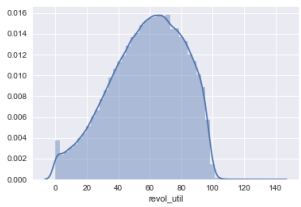


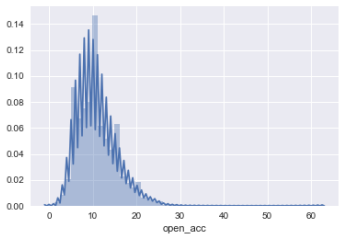


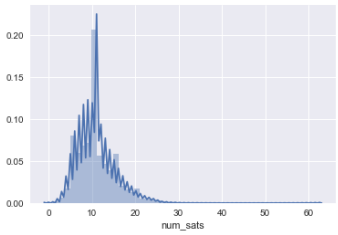


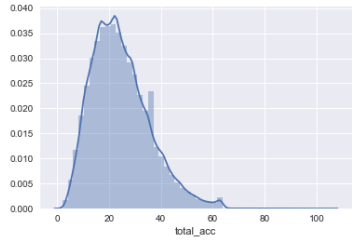












*Taking into account the size limitation of this report, only important features are included for visualization (Remainaing visualizations are separately uploaded on Canvas.

E] Analysis of features:

- As seen from above visualizations, although most of the important features are close to normal distribution, there seems to be multi-collinearity between features.
- Please refer to Jupyter notebook for feature Analysis
- We have imputed missing values, dropped the features with more than 50% missing values and pefromed additional pre-processing on data.
- As we are using classification algorithms which are sensitive to scale, we are using standardscaler to transform the variable into uniform scale of variance by their standard deviation.

F] Leakage:

- Taking into account that, at the time of sanctioning the loan, many of the features in the dataset will not be available, we must drop these features in order to handle the leakage issue.
- Please refer to Jupyter notebook for detailed analysis of this and feature selection

G] Intuition to also explain which features are best for classification

- We have used the data dictionary to identify the leakage feature. Also we can intuitively comment that features such as ‘grade’, ‘loan_amount’,’emp_length ,’Fico_score’ ,’dti’,’income’,’Housing_status’ etc. must be important features in deciding the quality of the loan
- Out of these, the visualizations for categorical features explain some high level correlation with bad loans. E.g. for grade, the relation looks significant
- We will compare this analysis after training our classifiers to see feature importance.

H] New features not in the dataset that you think will be useful:

- If we have features which explain ratio of income and expenses for an individual, it can serve as an important feature in addition to fico scores which only capture the credit worthiness. (There can be individuals who do not possess credit cards yet, they have good financial habits.
- Information about the tax paid by an individual in past years can also serve as a significant feature.

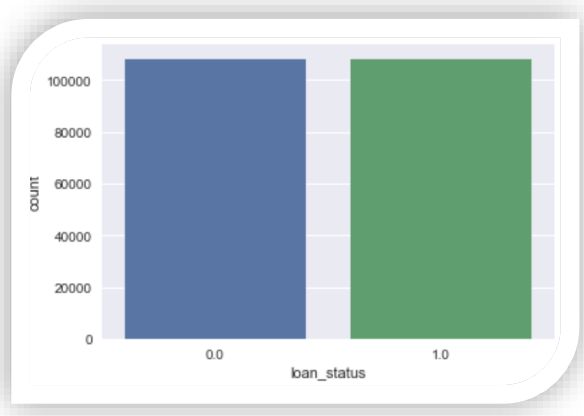
The exercise is to implement a suite of classifiers learned in the course thus far (decision tree, k-nearest neighbor, naïve Bayes, logistic regression, etc.). Split the data into 75-25 and hold out the 25. On the 75%, using an 80-20 split conduct your experiments and perform 10-fold cross validation

I] Separating training and test data

- As specified, we are separating 25% of the test data and using remaining 75% which is used as training data. (for cross validation and training)

J] Balancing the dependent variable using SMOTE:

- So far, we have already cleaned the dataset, performed preliminary feature selection, imputed the missing values, and performed standard scaling.
- In order to handle the imbalance problem in our dependent variable, we are using the package 'imbalanced-learn' in order to perform oversampling for the bad loans category (create new data points) using the SMOTE procedure.
- Please refer to Jupyter notebook for further details. Below is the distribution of our dependent variable after oversampling:



- Please not that we are applying this procedure only on Training set and NOT on test set. This procedure is used only in order to avoid bias during training of our classifier.

2) Which classifier gave the best results? Intuitively explain why?

- Please refer to Jupyter notebook for further details about individual classifier perfomeance
- Just to have initial judgement, we have run the classifier on whole training set without hyper-parameter optimization and judged the performance (This is ONLY for understanding purpose and not part of standard process.
- The best performance at this point seems to be Logistic Regression.

Logistic Regression

```
In [53]: lc = LogisticRegression()

In [54]: print_classification_report(lc,X_train,y_train,X_test,y_test)
```

	precision	recall	f1-score	support
0.0	0.90	0.68	0.78	36034
1.0	0.28	0.62	0.38	7154
avg / total	0.80	0.67	0.71	43188

K] Dimensionality Reduction and Principal Component Analysis:

- After dummy encoding (One Hot Encoding), our feature space grew from 36 to 102 features.
- For processor intensive algorithms such as K nearest neighbours (where we need to find our number of best K nearest neighbours using trail and error), training is very slow process with such a high level of dimensionality.
- In order to improve performance and also to minimize the degrees of freedom for our algorithm (which may lead to overfitting), we have applied PCA in order to reduce the dimensionality of the data to either 4 or 5 (principal components). The number is chosen based on the type of classifier under question and more details can be explored in Jupyter notebook solution.

L] 10 Fold Cross Validation and optimal set of Hyper-Parameters:

- As we have already performed Oversampling and balanced our dependent variable, we do not need to perform 'stratified K fold cross validation'.
- Also, we need to try many combinations of the hyper-parameters in order to find out model with optimal set of hyper-parameters.
- E.g. for Logistic regression, we have either lasso or ridge regularization parameters ('l1' and 'l2') with range of values of C (0.0 – 1.0).

Below table summarizes few examples. (we have experimented with very large number of such parameters for this assignment. However, in order to keep the code readable, only few examples have been included in Jupyter notebook for representation.

Classifier	Hyper-Parameters/ features to experiment with
Logistic regression	lasso or ridge regularization parameters ('l1' and 'l2'); range of values of C (0.0 – 1.0)
K Nearest Neighbours	Number of K neighbours *(5,10,30,50,100)
DecisionTreeClassifier	criterion=['entropy','gini']; max_depth=[None,10,20,40,60]
Naive Bayes Classifier	Priors; different number of PCA components
RandomForestClassifier	N_estimators

- For such large number of experiments, it is not pragmatic to handle it manually
- We are using sklearn's pipeline and GridSearchCV methods, in order to handle this experimentation as well as 10 fold cross validation simultaneously.

3) Do a feature selection and analysis to come up with the optimal combination of features that yield highest accuracy. (Hint: You shouldn't be using all the features for this exercise!)

M] Feature Selection:

- Similar to the issues listed above, we face same problems while determining best features to use in order to train our models.
- Keeping large number of features may lead to overfitting and keeping too few features may reduce accuracy of our models.
- Below is the custom class written which can be integrated in Pipeline (and eventually GridSearchCV) in order to find top k best features

```
from sklearn.base import BaseEstimator, TransformerMixin
class TopKDecisionTreeFeatures(BaseEstimator, TransformerMixin):
    def __init__(self,k=50):
        self.k = k
        self.dt = DecisionTreeClassifier()
    def fit(self, X_train, y_train):
        self.dt.fit(X_train,y_train)
        self.features = pd.DataFrame(data= list(self.dt.feature_importances_),
                                     index=list(X_train.columns),columns=['importance']).reset_index()
        self.sorted_features = self.features.sort_values(by=['importance'],ascending=False)
        self.topk = list(self.sorted_features['index'])[0:self.k]
        return self
    def transform(self, x):
        return(x[self.topk])
```

- Here, we are using decision tree in order to find the feature importance for our selection.
- GridSearchCV can make use of this class in order to experiment with the number of k we provide in list and find optimal number of top k features which provide best performance during cross validation.
- However, after experimenting with large number of K, we have the output that all the features are important. (Our selection of features based on intuition matches with the machine learning algorithm. The statistical significance is matching with the business significance for this case)
- We are also using the SelectKBest with different values of K best features using GridSearchCV just to compare that our custom selector class is matching the overall statistical method.
- The final answer we have is all the selected 36 features are important (They are basically ≈36 features from original dataset with correspond to 103 features after dummy coding)

4) What is the classification accuracy for the classifiers (confusion matrix comprising of F1-score, recall, precision)?

Please refer below:

Decision Tree Classifier:

	precision	recall	f1-score	support
0.0	0.86	0.61	0.71	36034
1.0	0.20	0.51	0.29	7154
avg / total	0.75	0.59	0.64	43188

Naïve Bayes Classifier:

	precision	recall	f1-score	support
0.0	0.89	0.34	0.50	36034
1.0	0.19	0.78	0.31	7154
avg / total	0.77	0.42	0.47	43188

Logistic Regression Classifier:

	precision	recall	f1-score	support
0.0	0.90	0.68	0.78	36034
1.0	0.28	0.62	0.38	7154
avg / total	0.80	0.67	0.71	43188

K-Nearest Neighbours Classifier:

	precision	recall	f1-score	support
0.0	0.84	0.59	0.70	36034
1.0	0.18	0.45	0.26	7154
avg / total	0.73	0.57	0.62	43188

Test your classifiers (all classifiers + optimal combination of features; essentially the best version of each classifier) on the hold out data (the 25%)
5) Report the accuracy scores (F-1, precision, and recall) of each classifier.

- Please refer below for the details:

NOTE: Below results may seem somewhat bad in comparison. However, we have used PCA for dimensionality reduction in order to have massive amount of experimentation with hyper-parameters. It improves processing speed at the cost of accuracy.

Decision Tree Classifier:

	precision	recall	f1-score	support
0.0	0.86	0.61	0.71	36034
1.0	0.20	0.51	0.29	7154
avg / total	0.75	0.59	0.64	43188

Naïve Bayes Classifier:

	precision	recall	f1-score	support
0.0	0.88	0.26	0.41	36034
1.0	0.18	0.82	0.30	7154
avg / total	0.76	0.36	0.39	43188

Logistic Regression Classifier:

	precision	recall	f1-score	support
0.0	0.89	0.60	0.72	36034
1.0	0.24	0.62	0.34	7154
avg / total	0.78	0.61	0.66	43188

K-Nearest Neighbours Classifier:

	precision	recall	f1-score	support
0.0	0.85	0.48	0.62	36034
1.0	0.18	0.58	0.28	7154
avg / total	0.74	0.50	0.56	43188

Just to showcase the actual results with optimized hyper-paramters, we have below execution for logistic regression with all data without dimensionality reduction (This ran for 8 hours 48 minutes):

Parameters tried:

```
logisticregression__penalty=['l2','l1'],logisticregression__C =[1.0,0.1,0.3,0.5,0.8,0.05,0.01,0.001]
```

Best parameters:

```
lrg.best_estimator_
```

```
Pipeline(memory=None,
     steps=[('logisticregression', LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interc
ept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False))])
```

Performance:

	precision	recall	f1-score	support
0.0	0.90	0.68	0.78	36034
1.0	0.28	0.62	0.38	7154
avg / total	0.80	0.67	0.71	43188

As we expected before, Logistic Regression Classifier both in terms of F1 score and also on the precision for detecting bad loans. Most of our feature selection after data processing have continuous features with close to normal distribution. This is why statistically regorous methods such as Logistic Regression is providing best perfromance.

NOTE: Please refer top jupyter notebok for any further details including the best choice of hyperparameters used for this algorithms along with the selection process.

This submission is under assumption that grading is based on the process and not accuracy. If the PCA step is skipped in this solution and the the steps are exeucted on powerful server environmet, the performance metrics shuld improve drastically.