Loan Default Prediction

Team 17

Xin Feng Kaimi Huang

Yuanchen Cui Boqian Wang

CONTENT

Project Introduction

Loan default rate prediction is **a critical aspect of financial risk management** for banks, credit unions, and other lending institutions.

01

The ability to accurately predict loan default rate can significantly **impact the financial health of these institutions**.

02

It enables lenders to **mitigate risks, set appropriate interest rates, allocate reserves for potential losses, and make informed decisions** about loan approvals.

03

It also helps in **tailoring loan products** to suit different risk profiles, **enhancing customer satisfaction**, and **fostering financial stability** in the broader economy.

Data Collection and Preprocessing

Data Source:

Lending Club Loan Dataset
(https://www.scaler.com/topics/data-science/loan-default-prediction/)
20000 records, 15 columns

Target:

bad_loan

Features:

id、grade、annual_income、short_employee、emp_length_num、home_ownership、Debt-To-Income Ratio、purpose、term、last_deling_none、last_major_derog_none、revol_util、total_rec_late_fee、od_ratio、bad loan

Missing Values:

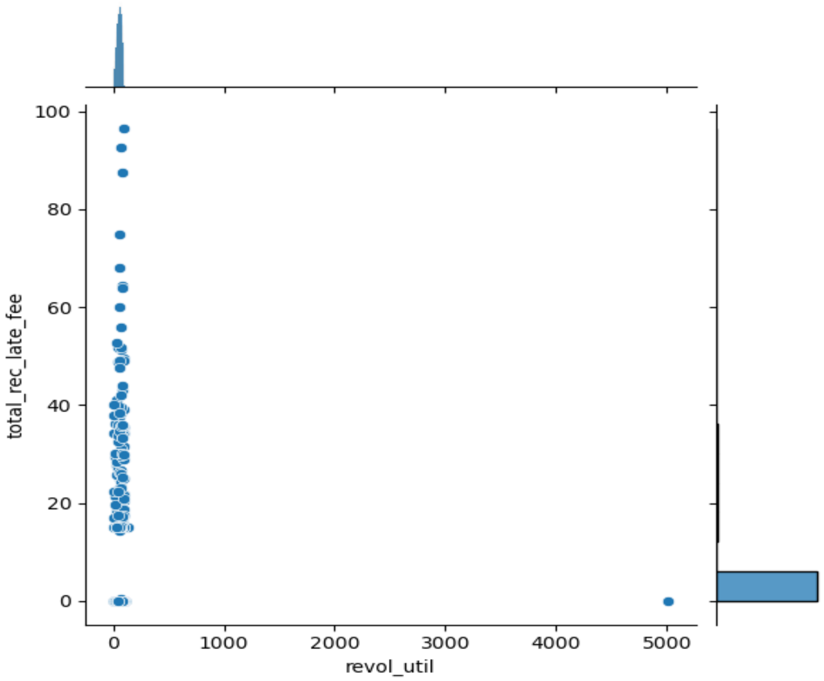
```
df.info()
```

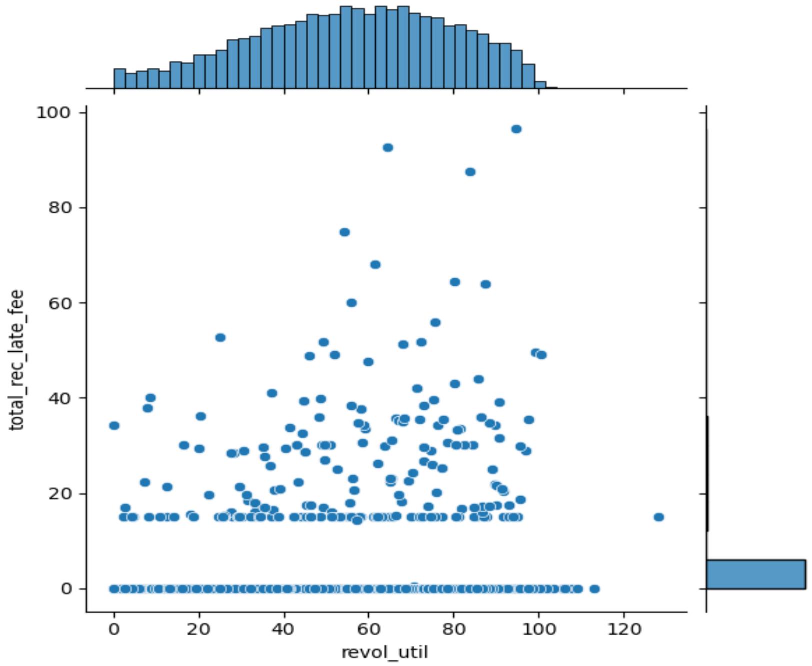
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   id                     20000 non-null  int64
 1   grade                  20000 non-null  object
 2   annual_income          20000 non-null  int64
 3   short_employee         20000 non-null  int64
 4   emp_length_num         20000 non-null  int64
 5   home_ownership         18509 non-null  object
 6   Debt-To-Income Ratio   19846 non-null  float64
 7   purpose                20000 non-null  object
 8   term                   20000 non-null  object
 9   last_delinq_none       20000 non-null  int64
 10  last_major_derog_none  574 non-null    float64
 11  revol_util             20000 non-null  float64
 12  total_rec_late_fee     20000 non-null  float64
 13  od_ratio               20000 non-null  float64
 14  bad_loan               20000 non-null  int64
dtypes: float64(5), int64(6), object(4)
memory usage: 2.3+ MB
```

Outlier

```
df.describe()
```

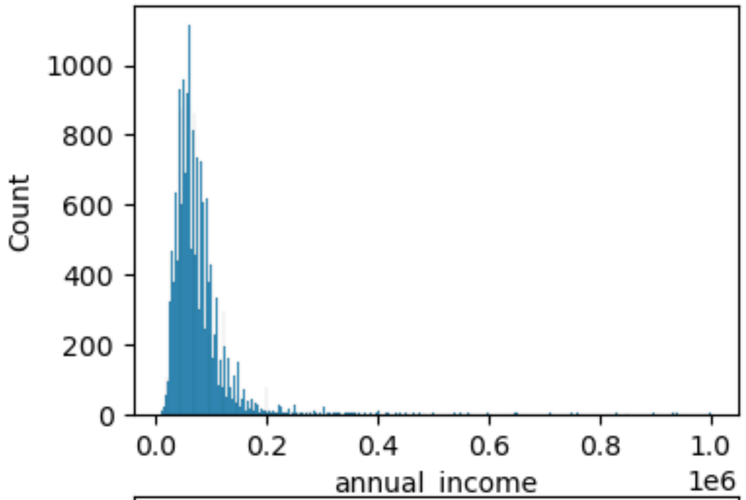
	id	annual_income	short_employee	emp_length_num	Debt-To-Income Ratio	revol_util	total_rec_late_fee	od_ratio	bad_loan
count	1.837100e+04	18371.000000	18371.000000	18371.000000	18371.000000	18371.000000	18371.000000	18371.000000	18371.000000
mean	7.594628e+06	73421.273257	0.112297	6.827609	16.590894	56.001801	0.293404	0.504941	0.200479
std	1.609952e+06	45612.958798	0.315740	3.769322	7.582902	43.411698	3.140913	0.287800	0.400370
min	5.860400e+05	8412.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000077	0.000000
25%	6.206280e+06	47000.000000	0.000000	3.000000	10.850000	38.750000	0.000000	0.257495	0.000000
50%	7.379923e+06	65000.000000	0.000000	7.000000	16.220000	57.100000	0.000000	0.507883	0.000000
75%	8.776061e+06	88000.000000	0.000000	11.000000	22.060000	74.000000	0.000000	0.753875	0.000000
max	1.145464e+07	1000000.000000	1.000000	11.000000	34.990000	5010.000000	96.466600	0.999894	1.000000

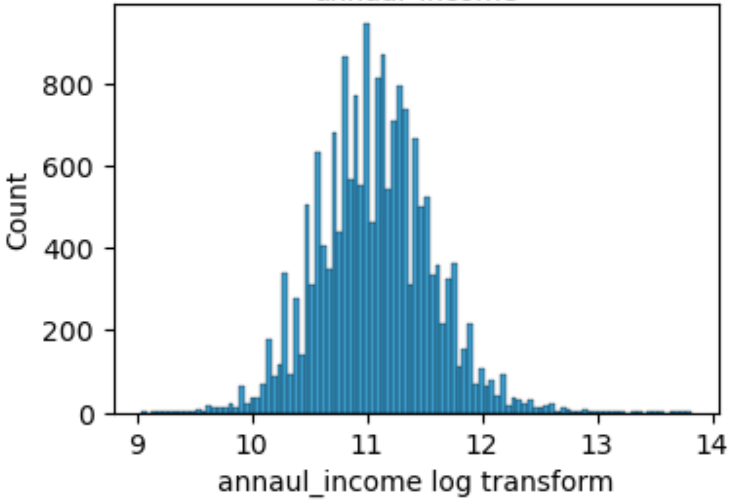




Feature Transformation & Engineering

1. Transform annual income to log (annual income)



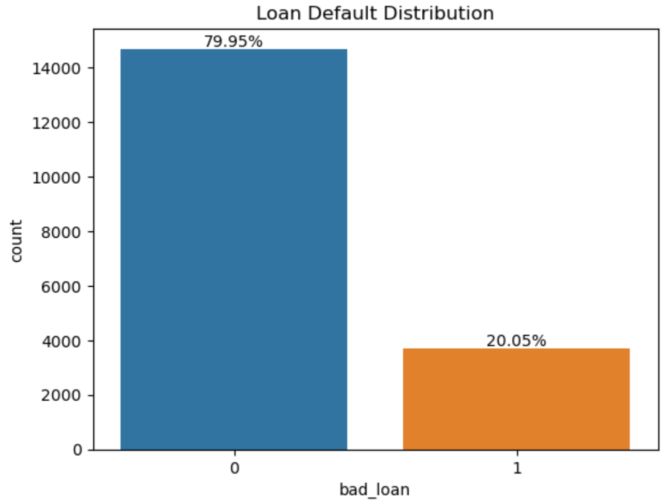


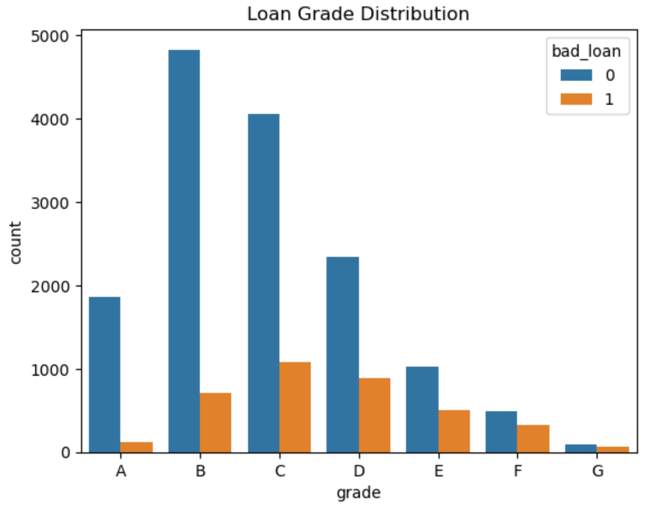
2. One-Hot Encoding for Categorical Features

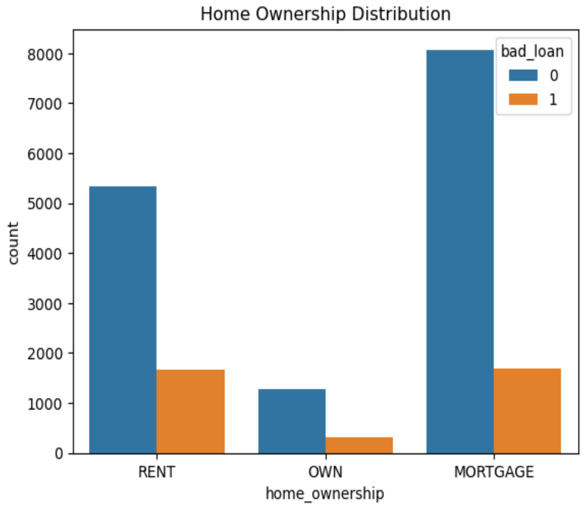
```
X.info()
```

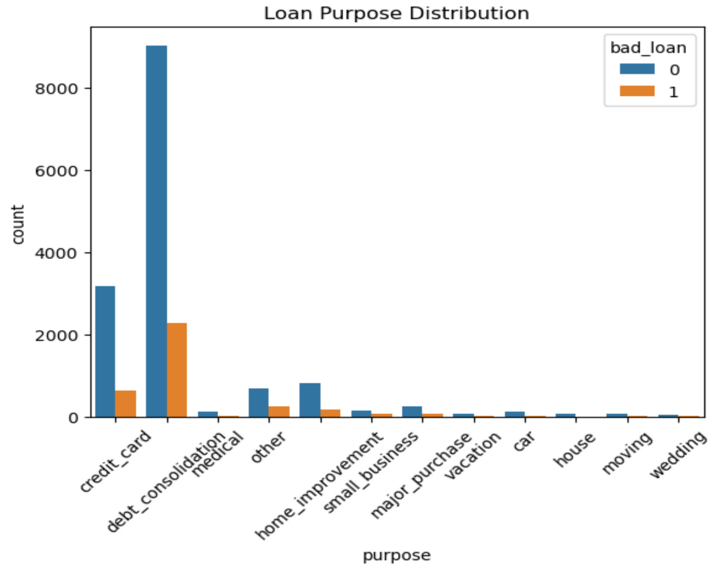
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18370 entries, 0 to 18369
Data columns (total 28 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   short_employee               18370 non-null  int64
 1   emp_length_num               18370 non-null  int64
 2   Debt-To-Income Ratio         18370 non-null  float64
 3   revol_util                   18370 non-null  float64
 4   total_rec_late_fee           18370 non-null  float64
 5   od_ratio                     18370 non-null  float64
 6   annual_income_log            18370 non-null  float64
 7   grade_B                      18370 non-null  uint8
 8   grade_C                      18370 non-null  uint8
 9   grade_D                      18370 non-null  uint8
 10  grade_E                      18370 non-null  uint8
 11  grade_F                      18370 non-null  uint8
 12  grade_G                      18370 non-null  uint8
 13  home_ownership_OWN           18370 non-null  uint8
 14  home_ownership_RENT          18370 non-null  uint8
 15  purpose_credit_card          18370 non-null  uint8
 16  purpose_debt_consolidation   18370 non-null  uint8
 17  purpose_home_improvement     18370 non-null  uint8
 18  purpose_house                18370 non-null  uint8
 19  purpose_major_purchase       18370 non-null  uint8
 20  purpose_medical              18370 non-null  uint8
 21  purpose_moving               18370 non-null  uint8
 22  purpose_other                18370 non-null  uint8
 23  purpose_small_business       18370 non-null  uint8
 24  purpose_vacation             18370 non-null  uint8
 25  purpose_wedding              18370 non-null  uint8
 26  term_60 months               18370 non-null  uint8
 27  last_delinq_none_1           18370 non-null  uint8
dtypes: float64(5), int64(2), uint8(21)
```

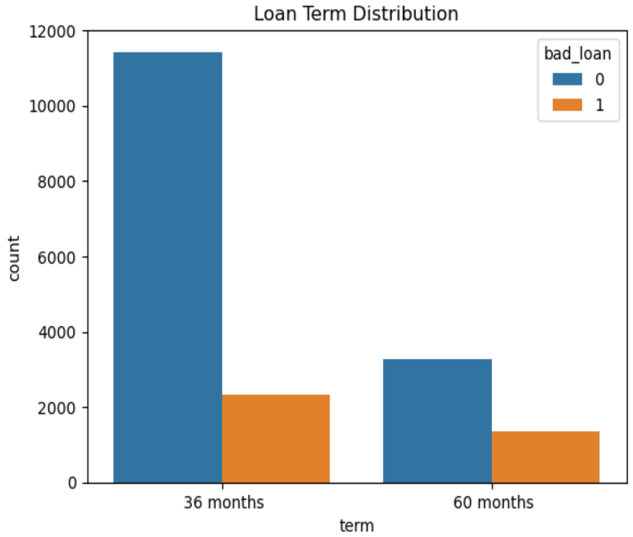
Distributions of Target and Categorical Features

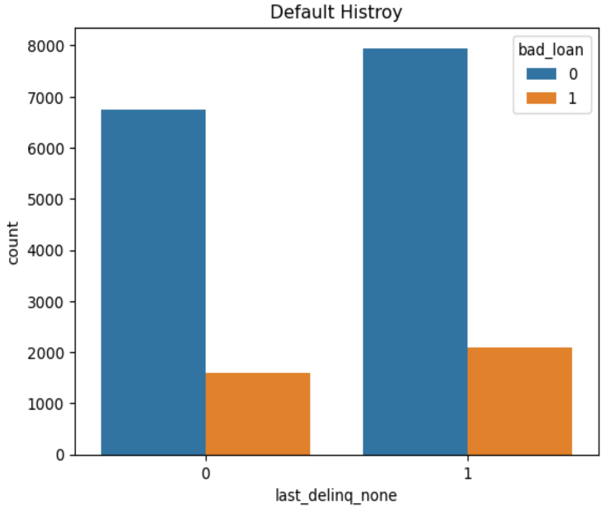




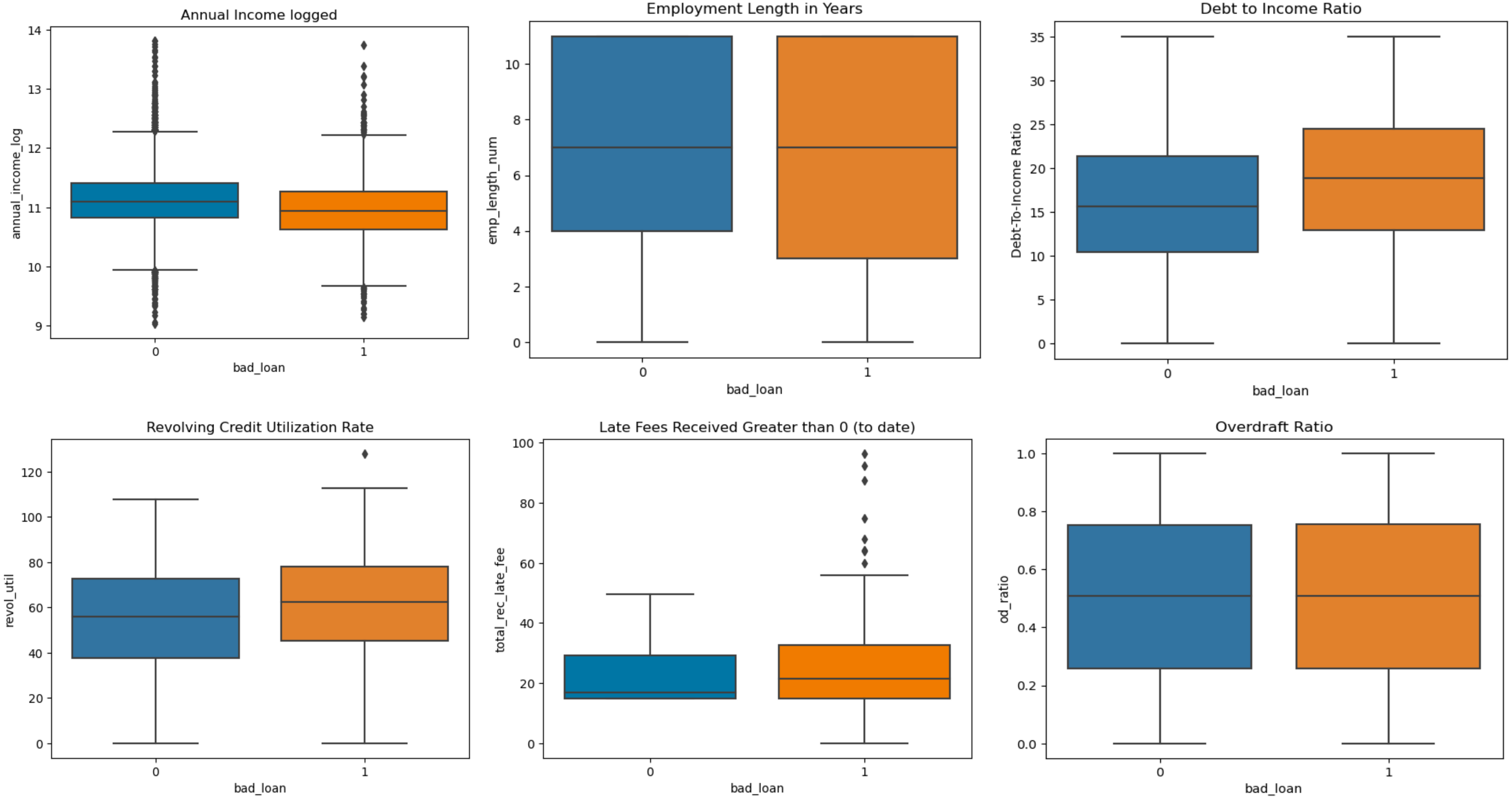








Distributions of Continuous Features



Training Models

- X = df.drop(columns=['id', 'annual_income','bad_loan','grade_A',

  'home_ownership_MORTGAGE','purpose_car','term_36 months',

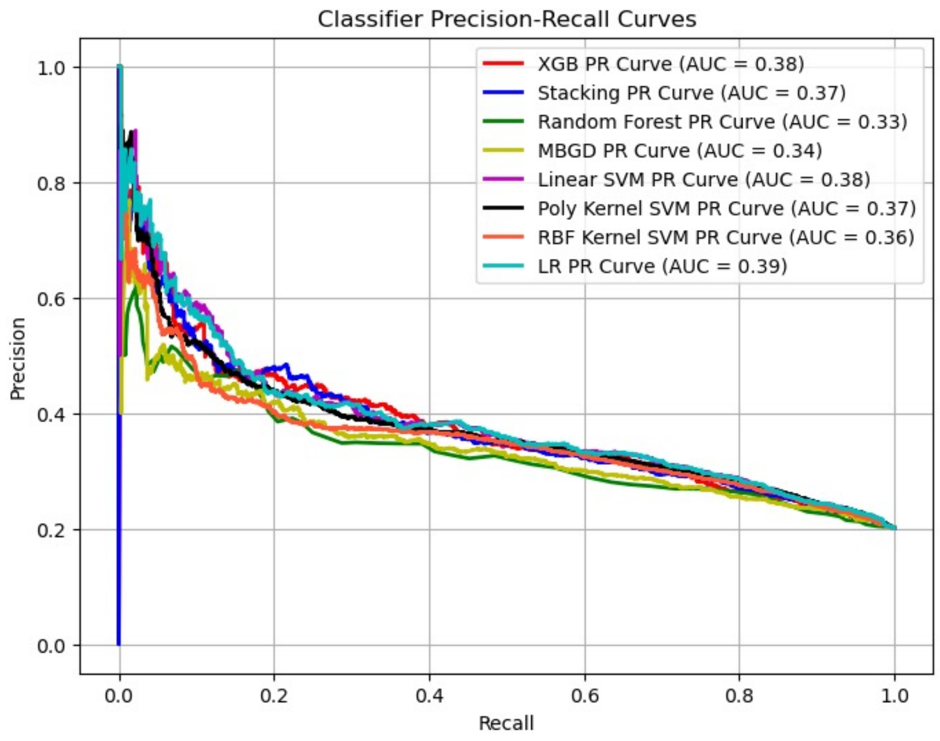
  'last_delinq_none_0'])
- y = df['bad_loan'].astype(int) *#1 represents bad loan. 0 good loan.*
- StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=123)
- scaler = StandardScaler()
- X_train = scaler.fit_transform(X_train)
- X_test = scaler.transform(X_test)
- class_weight='balanced' *#hyperparameter for all classifier instances*
- GridSearchCV() *#used for finding optimal hyperparameters for all classifiers*

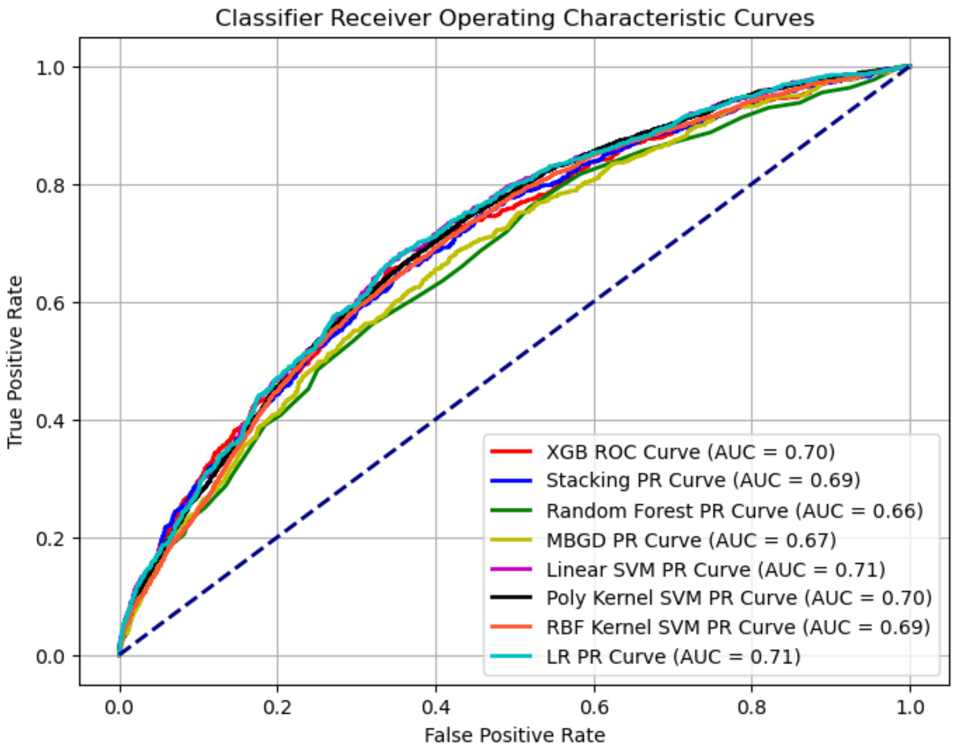
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18370 entries, 0 to 18369
Data columns (total 28 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   short_employee              18370 non-null  int64
 1   emp_length_num              18370 non-null  int64
 2   Debt-To-Income Ratio        18370 non-null  float64
 3   revol_util                  18370 non-null  float64
 4   total_rec_late_fee          18370 non-null  float64
 5   od_ratio                    18370 non-null  float64
 6   annual_income_log           18370 non-null  float64
 7   grade_B                     18370 non-null  uint8
 8   grade_C                     18370 non-null  uint8
 9   grade_D                     18370 non-null  uint8
 10  grade_E                     18370 non-null  uint8
 11  grade_F                     18370 non-null  uint8
 12  grade_G                     18370 non-null  uint8
 13  home_ownership_OWN          18370 non-null  uint8
 14  home_ownership_RENT         18370 non-null  uint8
 15  purpose_credit_card         18370 non-null  uint8
 16  purpose_debt_consolidation  18370 non-null  uint8
 17  purpose_home_improvement    18370 non-null  uint8
 18  purpose_house               18370 non-null  uint8
 19  purpose_major_purchase      18370 non-null  uint8
 20  purpose_medical             18370 non-null  uint8
 21  purpose_moving              18370 non-null  uint8
 22  purpose_other               18370 non-null  uint8
 23  purpose_small_business      18370 non-null  uint8
 24  purpose_vacation            18370 non-null  uint8
 25  purpose_wedding             18370 non-null  uint8
 26  term_60 months              18370 non-null  uint8
 27  last_delinq_none_1          18370 non-null  uint8
dtypes: float64(5), int64(2), uint8(21)
```

Model Performances

Model	PR AUC	ROC AUC	Accuracy
Logistic Regression	0.39	0.71	0.66
XGBoost	0.38	0.7	0.8
Random Forest	0.33	0.66	0.8
Stochastic Gradient Descent	0.34	0.67	0.61
Mini-batch Gradient Descent	0.34	0.67	0.61
Linear Support Vector Machine	0.38	0.71	0.66
Polynomial Kernel Support Vector Classifier	0.37	0.7	0.65
Gaussian RBF Kernel Support Vector Classifier	0.36	0.69	0.72
Stacking/Stacked Generalization	0.37	0.69	0.65



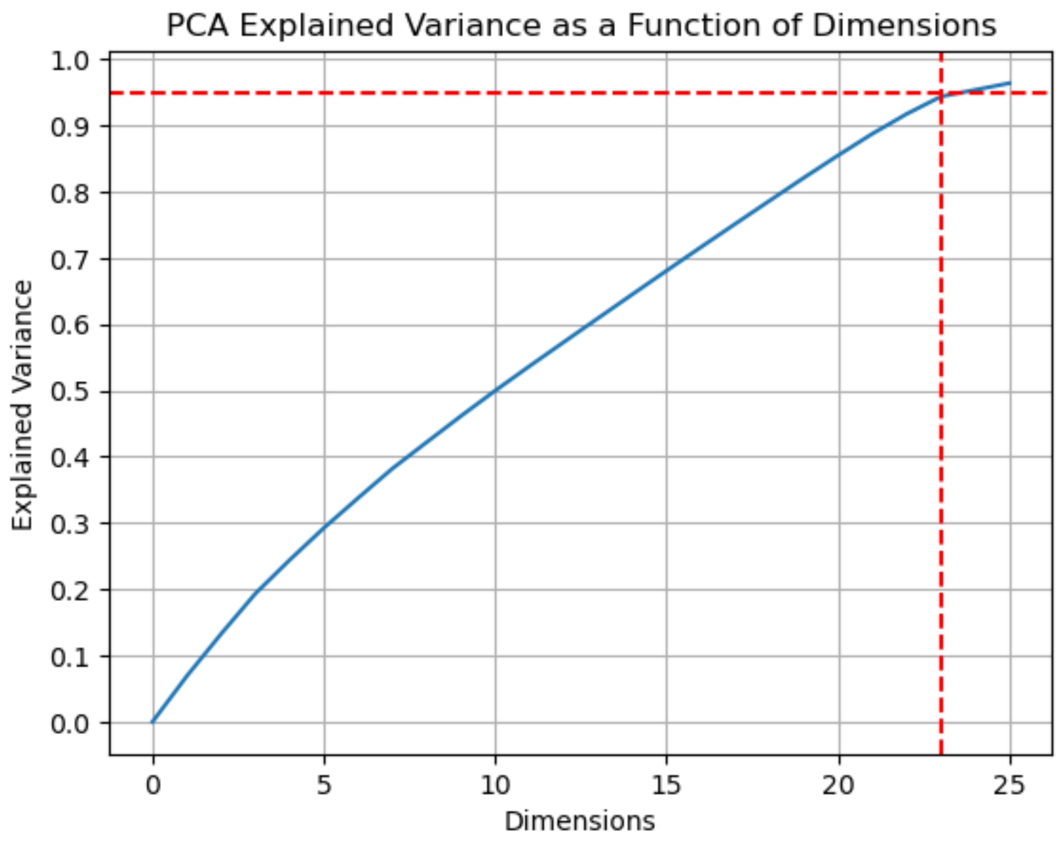


Attempts to Improve Model Performance

1. Dimensionality Reduction

- **Idea:** reduce noise via PCA
- In order to preserve 95% of the variance, kept 23 of the 25 features
- pca = PCA(n_components=0.95)
- **Result:** PR AUCs did not improve

  but dropped slightly



Attempts to Improve Model Performance

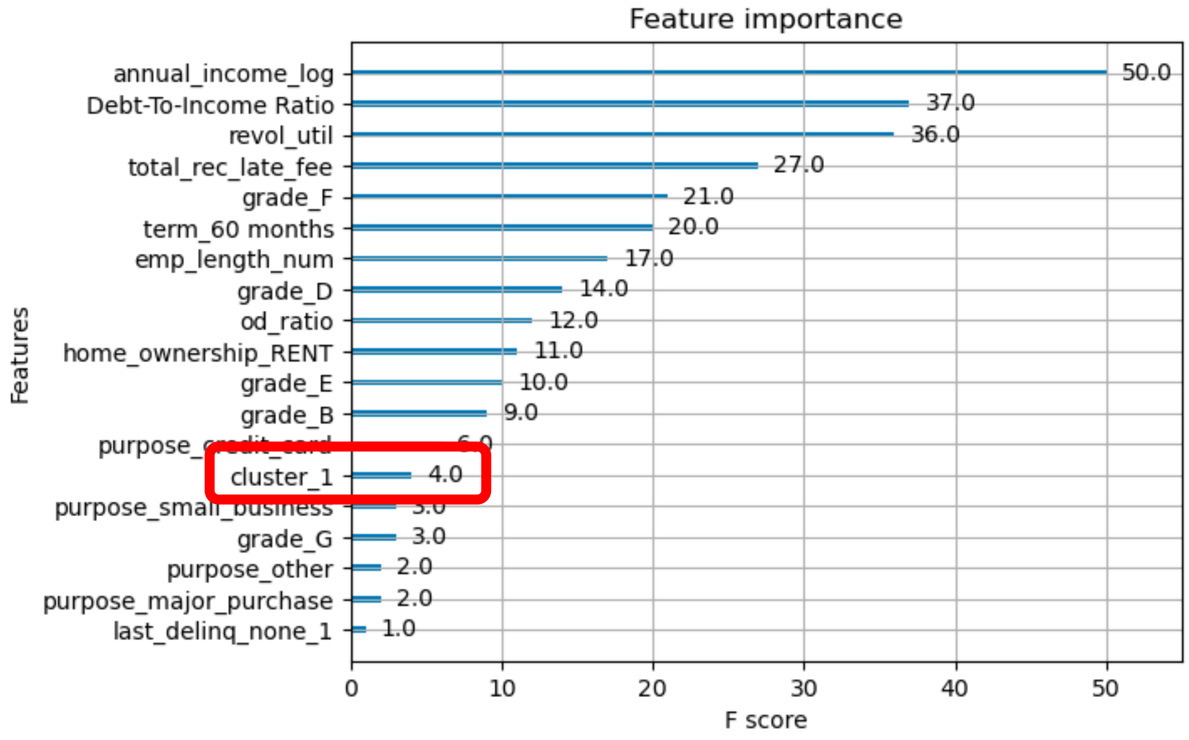
2. Create new categorical features, “cluster_0” & “cluster_1” , using K-means

- **Idea:** grouping loans based on features
  might add information that help classification

- pipeline = Pipeline([
  ('kmeans', KMeans()),
  ('log_reg', lg_clf)])

- Grid search showed the optimal number of cluster was
  {'kmeans__n_clusters’: 3}

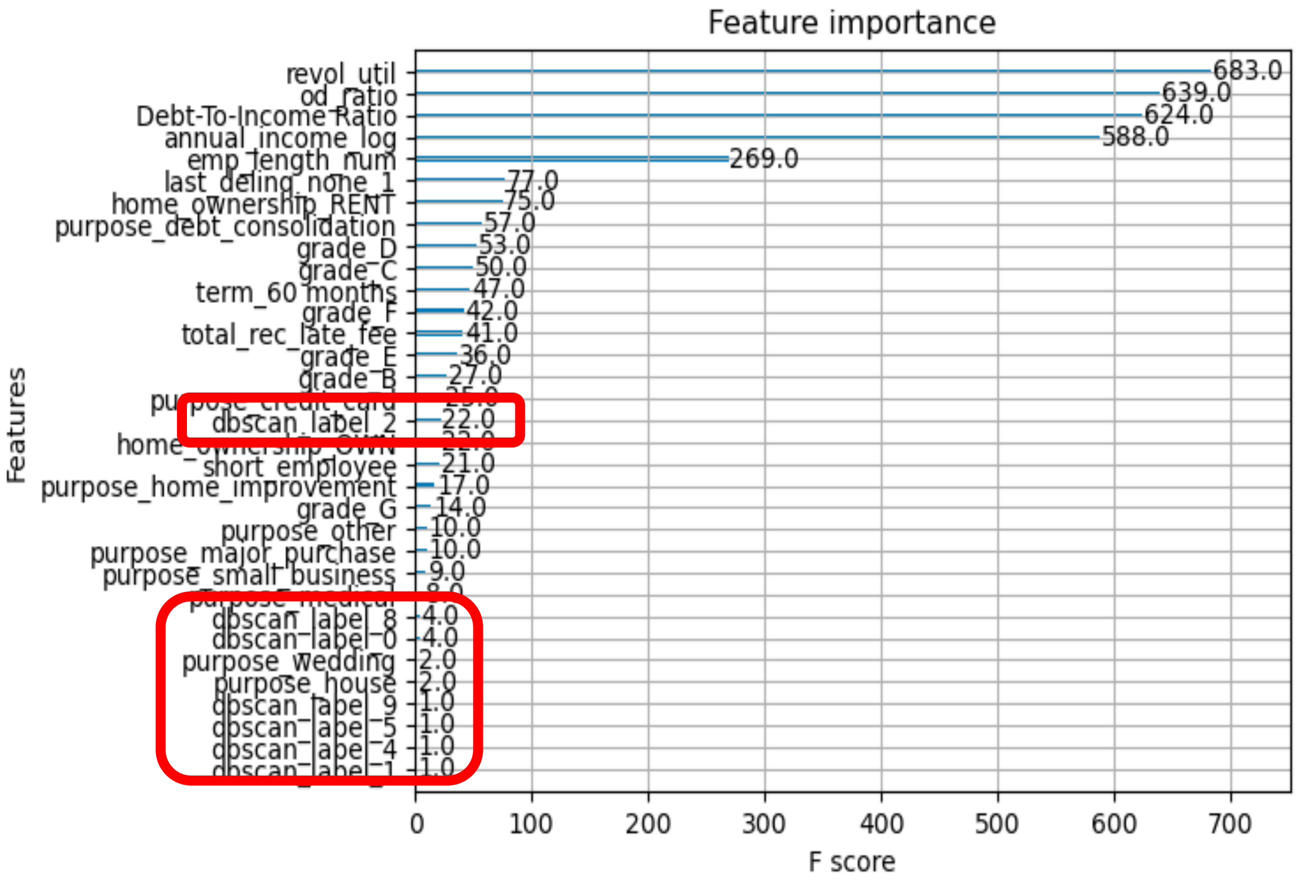
- **Result:** PR AUCs did not improve.
  New feature’s importance was low.



Attempts to Improve Model Performance

3. Create new categorical features, “dbscan_label_”, using DBSCAN

- **Idea:** grouping loans based on features might add information that help classification
- dbscan = DBSCAN(eps = 6, min_samples = 3)
- X_train was clustered into 13 groups
- **Result:** PR AUC was worse.
  New features’ importance was low



What if the target labels were wrong?

- Separate df into two datasets by target label
- Find anomalies in each set using Gaussian Mixture
  - setting n_components=2 because of the assumption that some of the observations in the good loans are mislabeled and should have been labeled as bad loans, and vice versa

```
df_good_loan = df[df.bad_loan==0]
df_bad_loan = df[df.bad_loan==1]
```

```
gm_good = GaussianMixture(n_components=2, n_init=10, random_state=123)
gm_good.fit(df_good_loan)
```

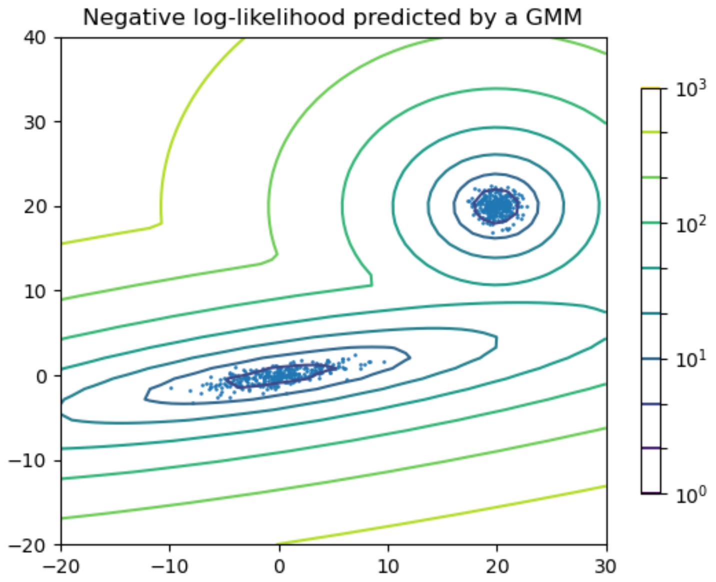
```
GaussianMixture(n_components=2, n_init=10, random_state=123)
```

```
gm_good.weights_
```

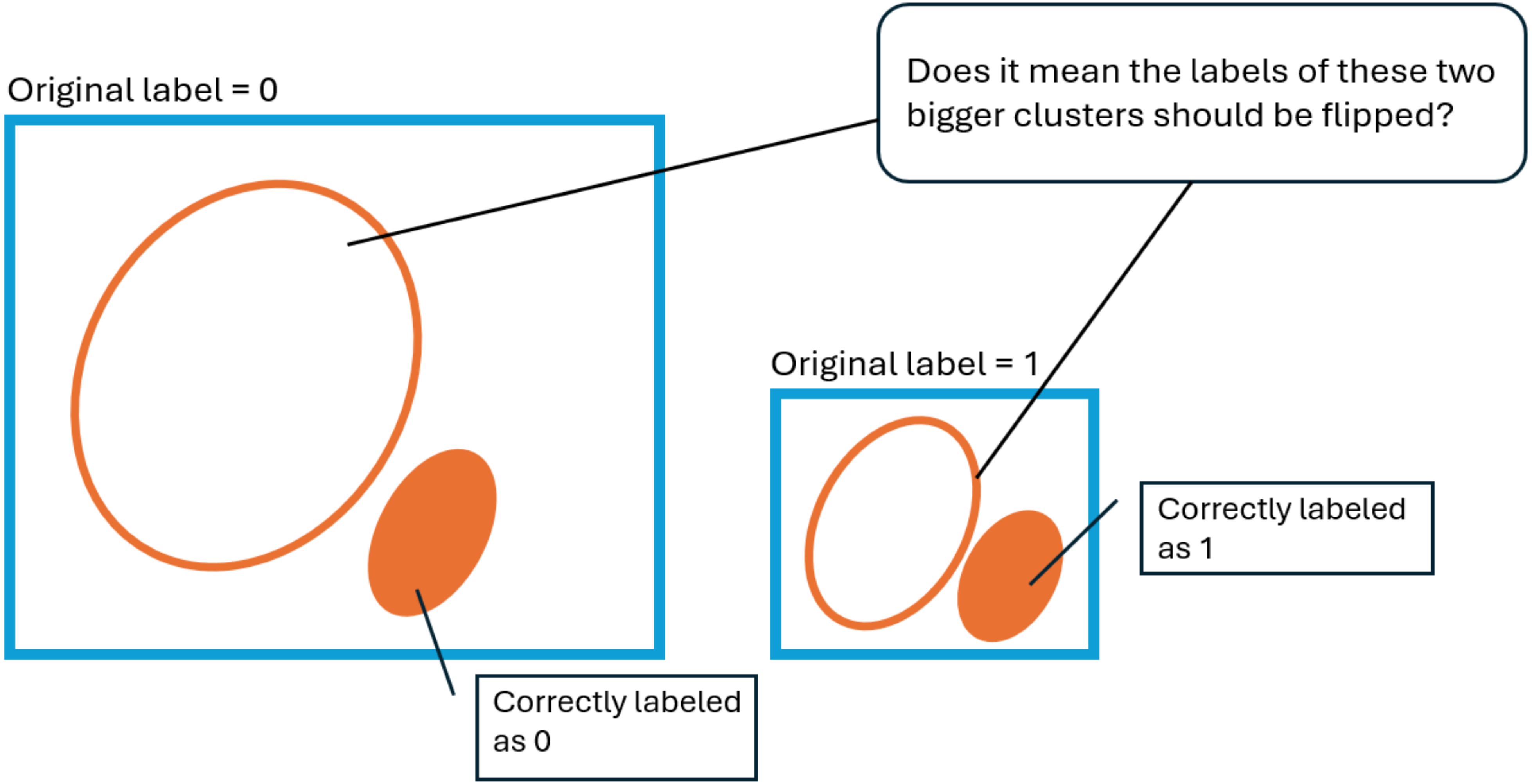
```
array([0.06910874, 0.93089126])
```

```
gm_bad = GaussianMixture(n_components=2, n_init=20,random_state=123)
gm_bad.fit(df_bad_loan)
gm_bad.weights_
```

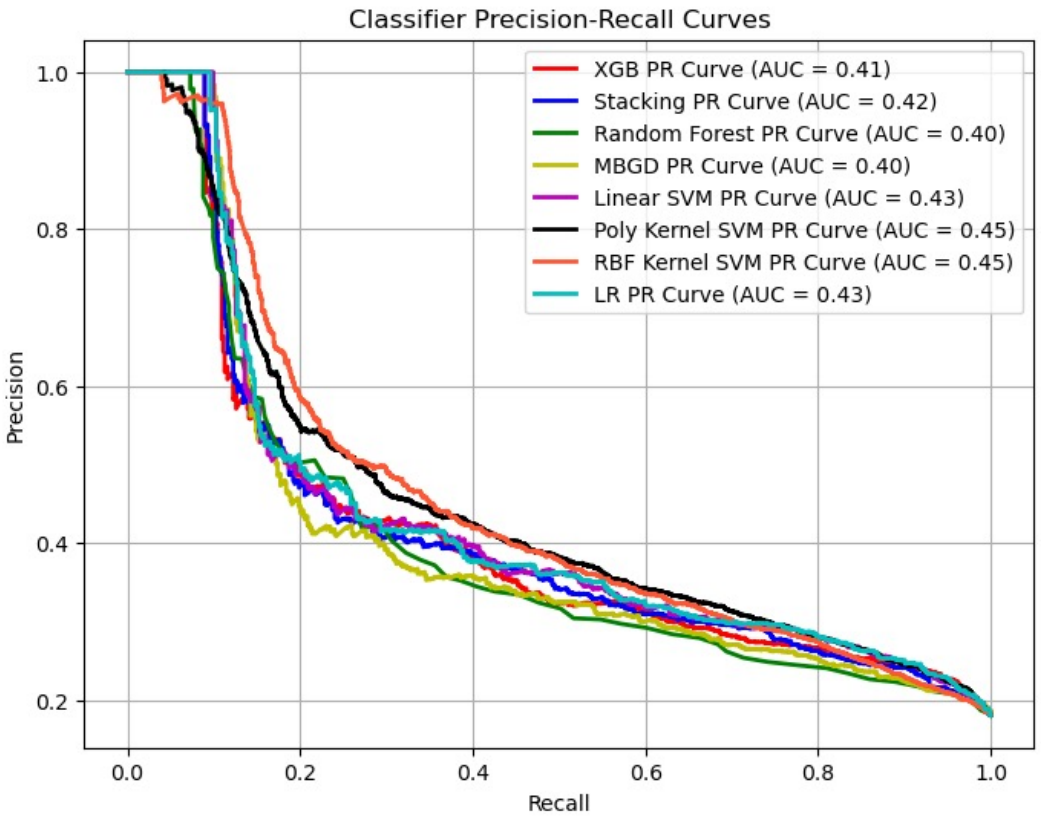
```
array([0.18028848, 0.81971152])
```



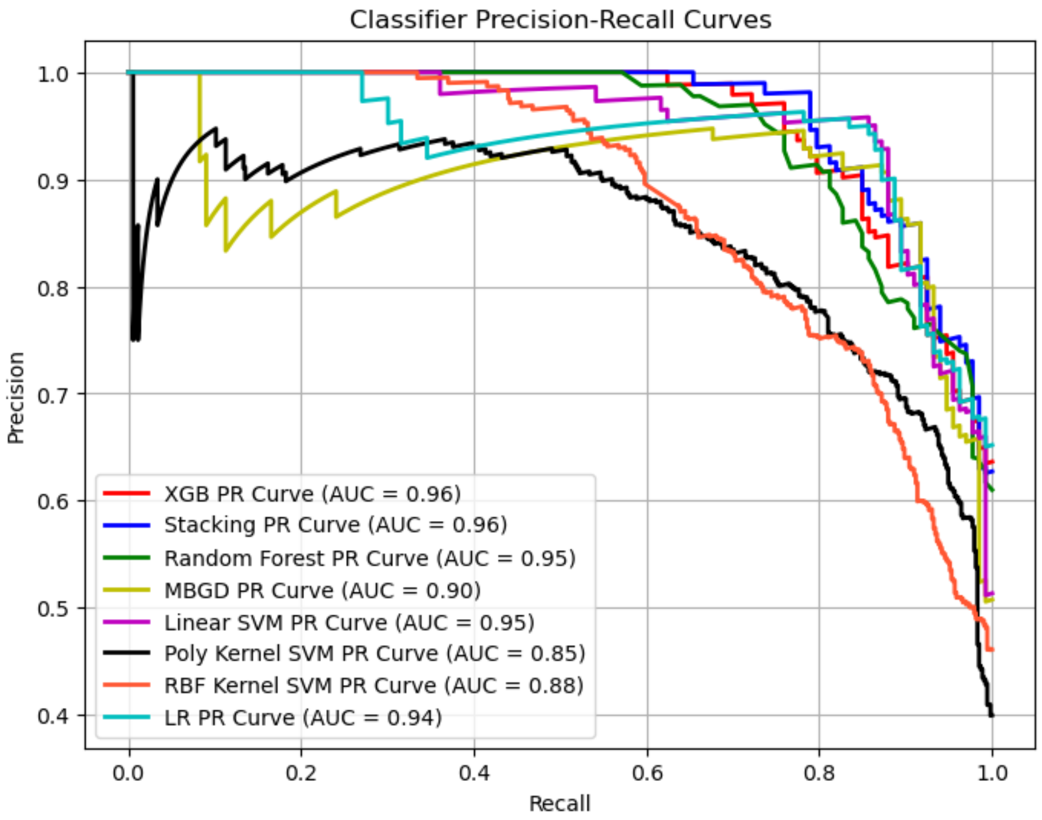
Gaussian Mixture Clusters



Model Performances using the bigger two sets of data

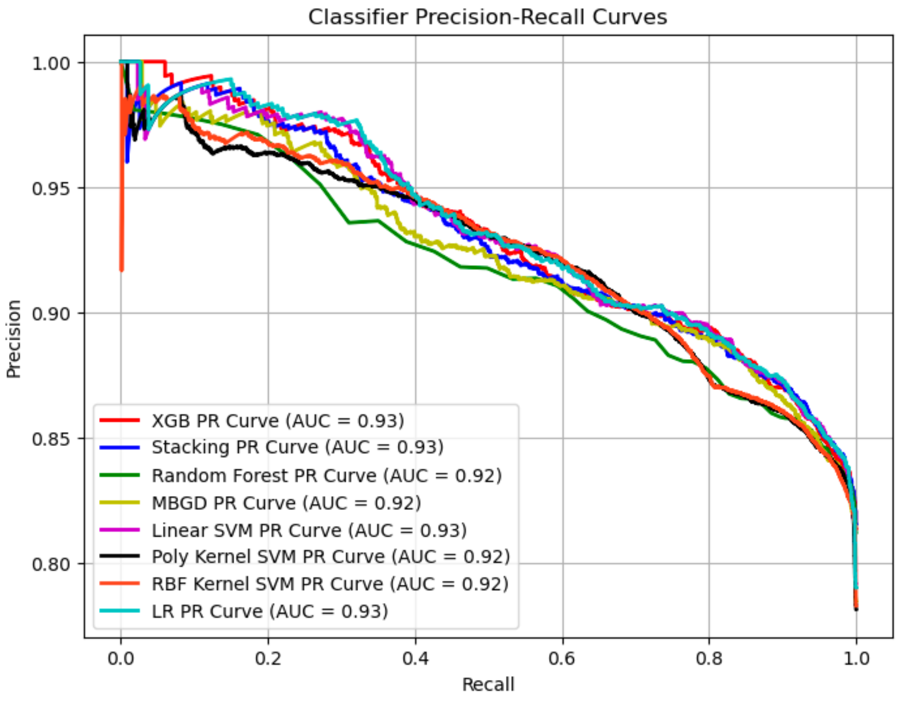


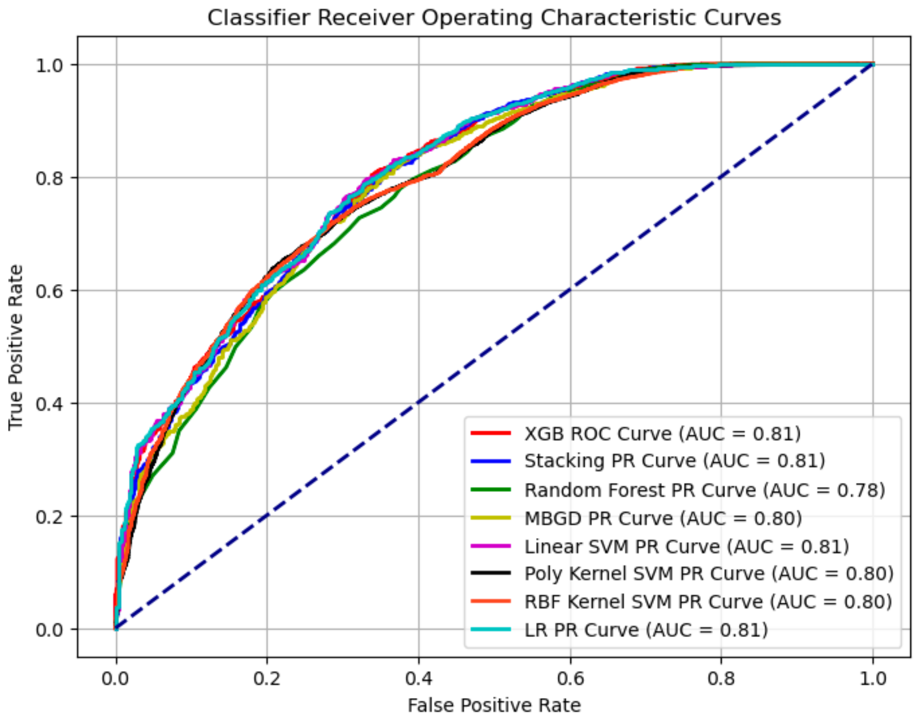
Model Performances using the smaller two sets of data



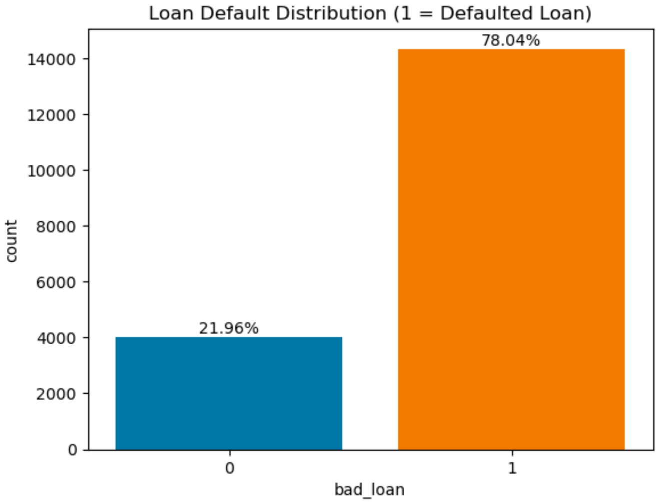
Model Performances after Flipping Labels

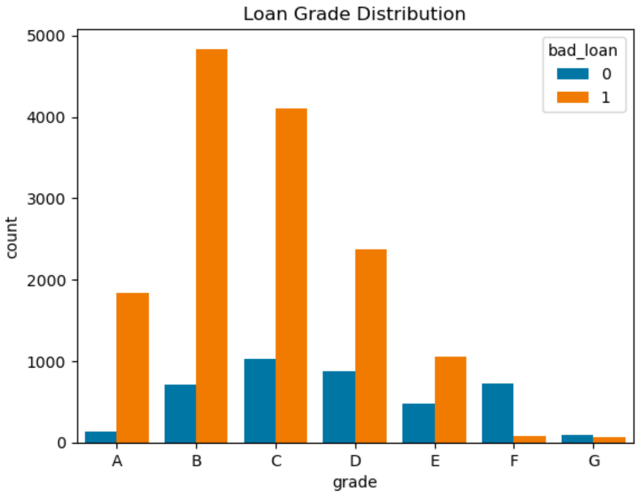
Model	PR AUC	ROC AUC	Accuracy
Logistic Regression	0.93	0.81	0.76
XGBoost	0.93	0.81	0.84
Random Forest	0.92	0.78	0.83
Stochastic Gradient Descent	0.92	0.8	0.75
Mini-batch Gradient Descent	0.92	0.8	0.75
Linear Support Vector Machine	0.93	0.81	0.77
Polynomial Kernel Support Vector Classifier	0.92	0.8	0.76
Gaussian RBF Kernel Support Vector Classifier	0.92	0.8	0.76
Stacking/Stacked Generalization	0.93	0.81	0.76

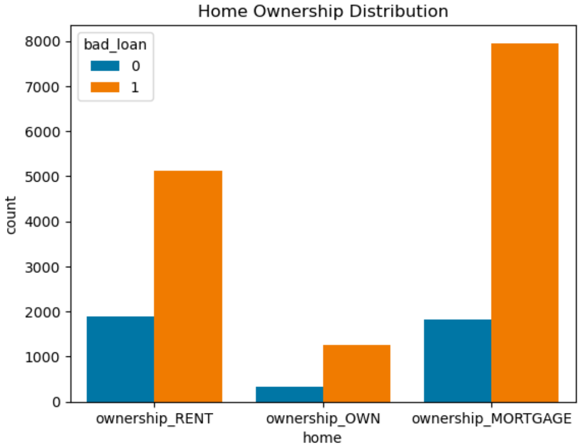


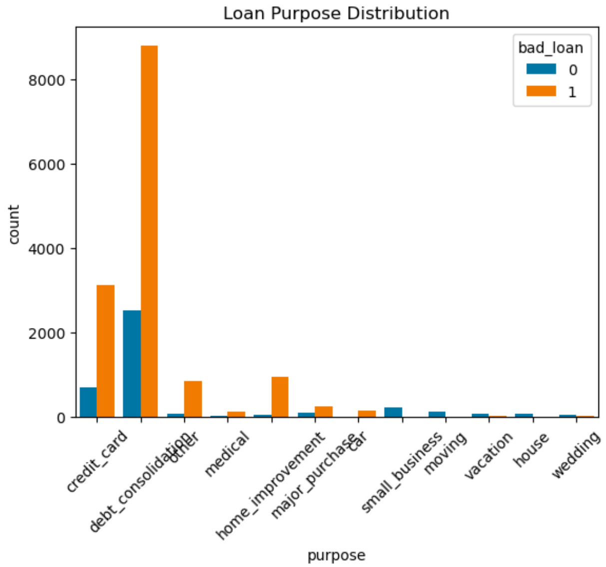


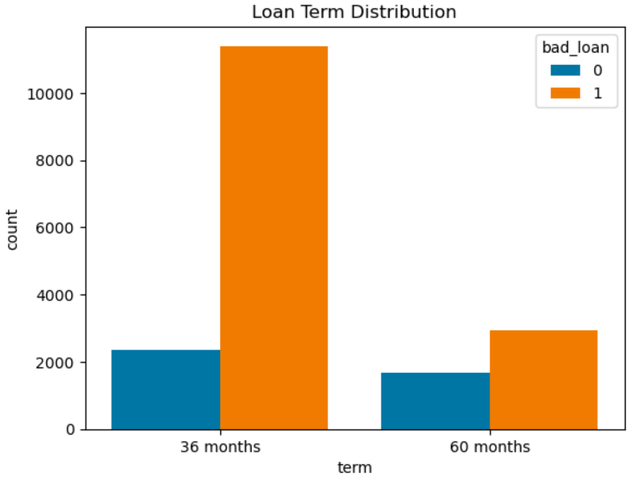
New Distributions of Target and Categorical Features

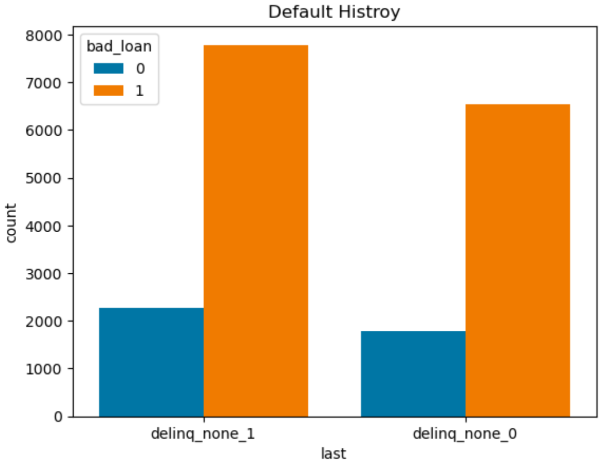




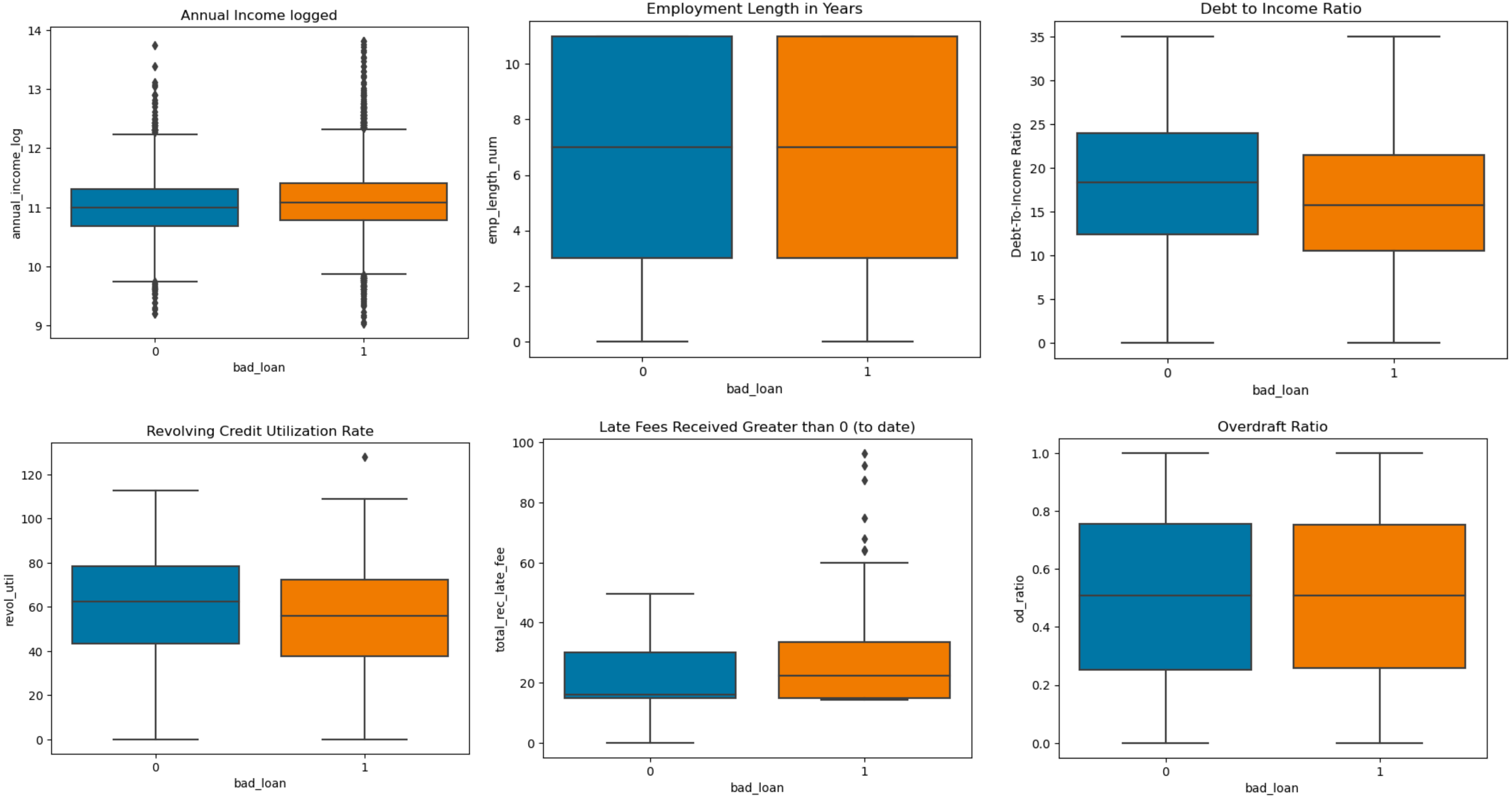








New Distributions of Continuous Features



Does it make sense to flip the labels of the majority of the data points?