The Lending Club Data Analytics

Wenwei Wu Jianfeng Lin Dreama Wang

1. INTRODUCTION

In this project, we will analyze the Lending Club data to predict whether the borrowers will repay their loan to better manage risk of loan service. Lending Club is an American person-to-person loan company, headquartered in San Francisco, California. We choose to analyze the data of Lending Club because it is the world's largest person-to-person (p2p) lending platform, which has a credible dataset with a huge amount of digital footprints.

The main problem we focus on is to decide whether we should lend money to our potential clients. We built Logistic Regression, Random Forest, Neural Network, and SVM models to predict whether a client was "fully paid" or "charged off" the loan in the year of 2018. Since borrowers who default cause the most losses to the lender, if we could identify these borrowers with models, we can reduce the losses to the lender.

We used accuracy score, false positive rate, and AUC score to evaluate the performance of models.We stated the hypothesis that the best model to predict loan status of fully paid and charged off is the Random Forest model. Since random forest can reduce error, less impact of outliers, and avoid overfitting.

2. METHODOLOGY

2.1 Description of Dataset

We used the Lending Club dataset obtained from Kaggle. We had 56,237 observations, 18 explanatory variables, and 1 response variable for this project. It originally had approximately 2 millions observations and 151 variables. First, we checked the missing values of each variable and dropped variables with a high percentage of missing values. Then, we selected variables based on our objective. In feature selection, we discovered that some variables were highly correlated with the response variable, but could not be used to answer our questions. After the data selection process, we had 56,237 observations and 19 variables.

In exploratory data analysis, we drew histograms to check the relationship between categorical variables and the response. We found that the user tends to repay their loan if their income status were verified. In numerical variables, we made a heatmap and found that there is a high correlation between the installment and the amount of loan. The fico rate has strong correlation with the response.

2.2 Description of algorithms

2.2.1 Logistic Regression

Logistic Regression was a Machine Learning algorithm that prefers to be used in binary classification. We used it for a dataset that has only two outcomes '1' and '0', in our project we determined whether the loan status is fully paid or charged off. We chose logistic regression as one of the models since it was easy to implement and effective to train. However, compared with other models, logistic regression might not be as powerful as other algorithms. In addition, Logistic regression offered an interpretation of the model coefficients, which could be suggested as indicators of feature importance.

$$log(\frac{\pi}{1-\pi}) = \alpha + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p$$

2.2.2 SVM

SVM was a supervised machine learning algorithm that was used in binary classification. It aimed to differentiate the entire dataset into disparate classes and find an optimal boundary between “fully paid” and “charged off”. This boundary was the best separating line, could be linear or nonlinear, that maximizes the distance between the hyperplanes of decision boundaries based on the features we choose. This made the division of vector space into two sets better. SVM was used in high dimension data in binary classification, and dealt with outliers. However, it took time to process the model and caused poor performance for overlapped classes.

2.2.3 K-nearest neighbor algorithm (KNN)

KNN was a mature method in theory and one of the simplest machine learning algorithms. The idea was that a sample belongs to a category if most of the k most similar (that is, the closest neighbors in the feature space) samples in the feature space belong to that category. In the KNN algorithm, the selected neighbors were correctly classified objects. In order to find the nearest “neighbors”, the algorithm applied the concept of Euclidean distance. In the Euclidean plane, if two dimensions, we let point p1 have Cartesian coordinates(p1, p1), and q had coordinates (q1, q2). Then the distance between p and q was given by $d(p,\ q)\ =\ \sqrt{(q1\ -\ p1)^2\ +\ (q2\ -\ p2)^2}$. In higher dimensions, for points given by Cartesian coordinates in n-dimensions, we had $d(p,\ q)\ = \sqrt{(q1\ -\ p1)^2\ +\ (q2\ -\ p2)^2\ +\ (q3\ -\ p3)^2\ +\ \dots\ +\ (qn\ -\ pn)^2}$.

By this mathematical concept, we could easily find its shortest distance neighbor. Based on the parameter k, we could find out its k nearest neighbors. However, it also left some disadvantages. For example, the test sample classification required a large amount of computation and memory overhead, because the distance between each text to be classified and all known samples must be calculated before its K nearest neighbor points could be obtained. At present, the commonly used method is to clip the known sample points in advance, and remove the samples that have little effect on classification in advance. In addition, the selection of parameter k will need deep consideration. For example, when the sample was imbalanced, such as the sample size of one class is very large while the sample size of other classes was very small, it may lead to the majority of the K neighbors of the sample with large volume class when a new sample was imported.

2.2.4 Random Forest

Random Forest was a supervised machine learning algorithm that was based on decision trees. It was based on the logic of bagging. Random Forest served as an ideal method we can implement in this project. First, Random Forest was good at dealing with high dimensional data since it worked with subsets of data. This was good for our project since we have in total 18 predictors which were 18 dimensions. Second, Random Forest could help with finding the importance of variables, so we could view the relationship between predictors and the response. We could find out which predictors affected the outcome of subscription of a term deposit more and which affected less. Also, the prediction of Random Forest was robust to multicollinearity, so this could handle the issue that there were few pairs of variables having multicollinearity in our dataset. In addition, compared with other methods, the random forest tended not to overfit. As the more trees we added into the random forest, the tendency to overfit decreases.

2.2.5 Neural Network

A neural network was a series of algorithms that try to recognize underlying relationships in a set of data through a process that mimics the way the human brain works. A neural network can build nonlinear models of complex relationships. In our dataset, the relationship between predictors and responses was nonlinear and complex. Second, the neural network had a strong predictive ability. The model can effectively infer unknown relationships between unknown data, so that the model can generalize and predict unknown data.

3. IMPLEMENTATION DETAILS

We randomly splited the data with 70% of training data and 30% of test data. We did feature scaling on the training data for better model training. We built models using the training data and predicted with the test data.

3.1 Logistic Regression

We first normalize the data, then all the variables have small p-values close to 0, we can conclude that all the variables are sufficiently important. We build the logistic regression model with default set parameter c =1, inverse of regularization strength, to reduce the generalization error to prevent overfitting the training data.

3.2 SVM

We used SVM with a linear Kernel to separate using a single line since we have a large size of dataset, the linear kernel is faster than other kernels. We try different c regularization parameters when building the model.

3.3 K-nearest neighbor algorithm (KNN)

In order to simulate the situation of finding potential neighbors of a customer to see the proportion of the class between "charged off" and "fully paid" to decide whether we should lend the money to him. We randomly selected 51 rows of our dataset, and chose the first row as our customer information. By our algorithm, we were able to find which class most of his neighbors belong to, and put him in the class with the most people. This helped us to decide whether we should lend money to him.

3.4 Random Forest

We used a random forest algorithm from sklearn. In the selection of tuning parameters, we seted max_features to 6, making the maximum number of features a random forest is allowed to try in individual trees is 6. We set the number of estimators to 100 since it makes the balance between the performance of the model and the speed of the algorithm. We set the minimum sample leaf to 1 since a smaller leaf makes the model more prone to capturing noise in train data. We also found the importance of variables with random forest.

3.5 Neural Network

We used a neural network algorithm from sklearn. It has preset on the input layer and output layers, which are 18 and 1. We set 3 hidden layers, and each contains 10 neurons. We tried multiple combinations of hidden layers and found this is the most powerful. We applied the solver of ‘lbfgs’ as an optimizer because it can converge faster and perform better. We set the L2 penalty to 0.0001 to avoid overfitting. We set the maximum number of iterations to 100.

4. RESULTS AND INTERPRETATION

	Logistic regression	Random Forest	SVM	KNN	Neural Network
Test Accuracy	0.976	0.987	0.976	0.969	0.988
AUC Score	0.950	0.976	0.951	0.925	0.976
False positive/ Total Population	0.014	0.009	0.014	0.022	0.0069

From the above table, we can find that the Neural Network has the highest accuracy. In surprise, we find the performance of Random Forest and Neural Network are approximately equal. They perform well in both classification tasks. In addition, they have a great performance on non-linear relations of our large dataset. They alse balance bias and variance trade-off well.

We decided to choose a Neural Network as our optimal model. According to risk management, we should control the probability of those who are charged off but in the model we predict as fully paid, which are those false positive values. The neural network has the smallest number of false positive values than others. This is better for the objective of our project that reduces the potential losses to the lender.

In KNN, when the sample data is imbalanced, such as the sample size of one class is very large while the sample size of other classes is very small, it may lead to the majority of the K neighbors of the sample with large volume class when a new sample is imported. Because of this problem, it is hard to decide the parameter k and leads to relative bad prediction.

Also, the overlapped classes cause poor performance in SVM, and selecting different kernel functions may cause the accuracy to be different. In the further study, we could try other kernels like the RBF kernel since our dataset may not be linearly divided by plane. Logistic regression has poor performance because the relationship between predictors and response in our dataset is not linear enough.

5. APPENDIX

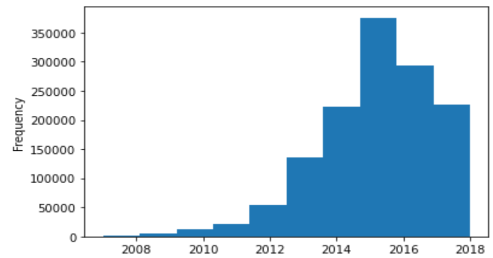


Figure 1. Selection Data by year

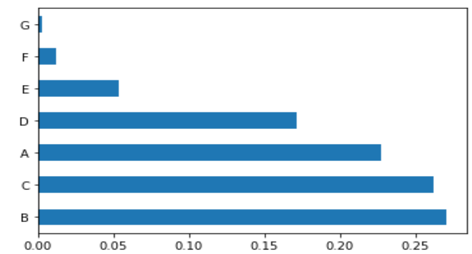


Figure 2. Percentage of each grade

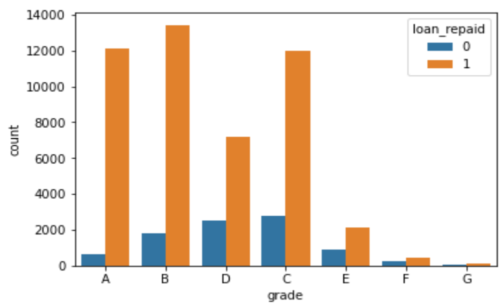


Figure 3. Loan_repaid by grade

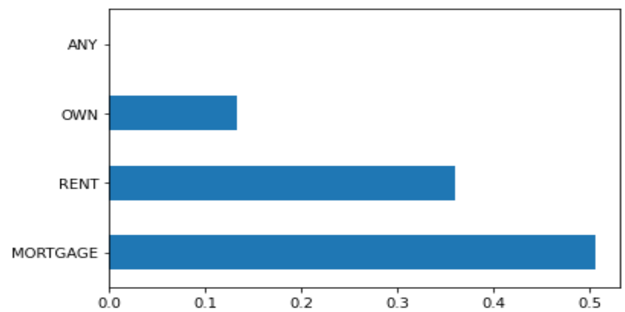


Figure 4. Percentage of home_ownership

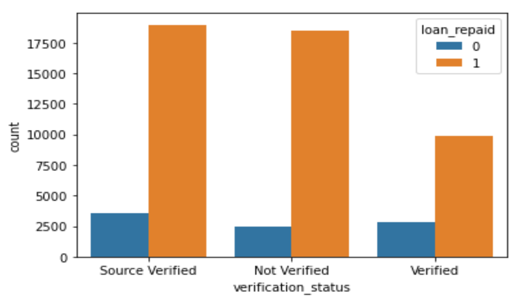


Figure 5. Loan_repaid by verification_status

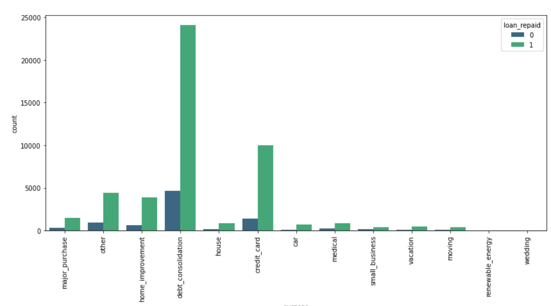


Figure 6. Loan_repaid by purpose

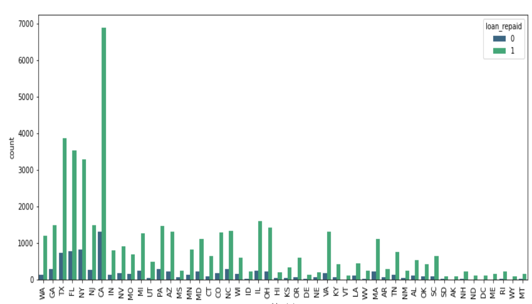


Figure 7. Address state by loan_status

Figure 8. Heatmap

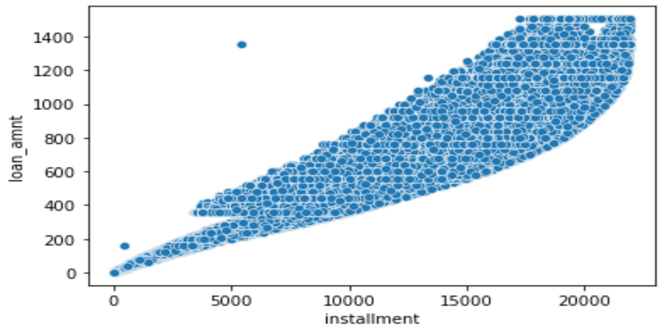


Figure 9. Scatterplot of installment vs loan_repaid

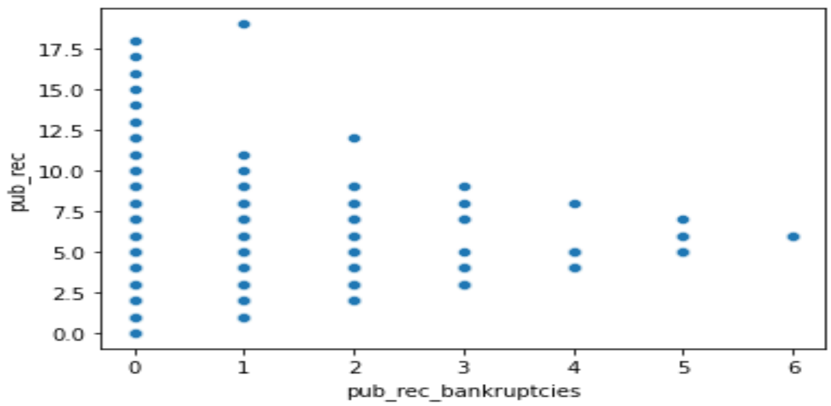


Figure 10. Scatter Plot of pub_rec_bankruptcies vs pub_rec

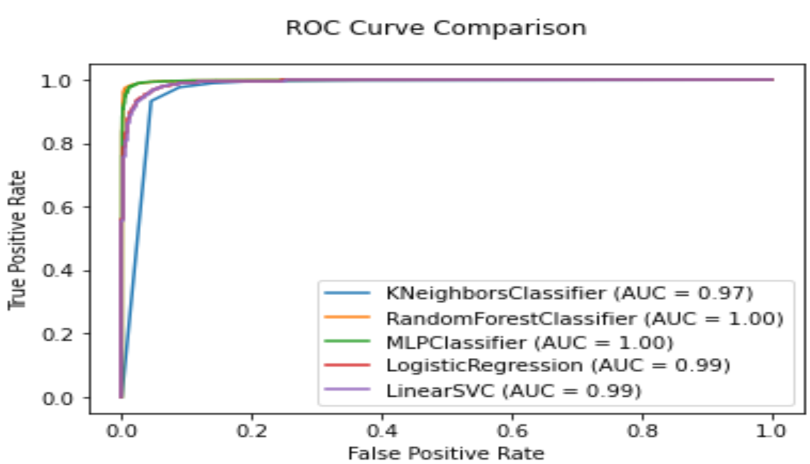


Figure 11. AUC with each model

STA142 DW_JL_WW_FinalProject

January 29, 2023

0.1 1. Data Manipulation

[11]:
```
# import packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[12]:
```
# loading data
# data from https://www.kaggle.com/wordsforthewise/lending-club
data = pd.read_csv("accepted_2007_to_2018Q4.csv")
```

```
<ipython-input-12-fa71728d02e8>:3: DtypeWarning: Columns
(0,19,49,59,118,129,130,131,134,135,136,139,145,146,147) have mixed types.
Specify dtype option on import or set low_memory=False.
  data = pd.read_csv("accepted_2007_to_2018Q4.csv")
```

[13]:
```
data.shape
```

[13]:
```
(2260701, 151)
```

0.1.1 Data Cleaning

We find that there are 151, so we first check Nan for each variable

[14]:
```
# ranking - the number of NaN values
rank_nan = data.isnull().sum().to_frame()
# the percentage of NaN
precent_nan = [round(i/2260701,4) for i in rank_nan[0]]
# add precent_nan to rank_nan
rank_nan["precent_nan"] = precent_nan
```

[15]:
```
rank_nan.sort_values(by=0,ascending=True)
```

[15]:
```
                                          0  precent_nan
id                                        0       0.0000
fico_range_high                          33       0.0000
hardship_flag                            33       0.0000
```

```
revol_bal                                          33       0.0000
initial_list_status                                33       0.0000
…                                                   …         …
hardship_reason                               2249784       0.9952
hardship_dpd                                  2249784       0.9952
hardship_loan_status                          2249784       0.9952
orig_projected_additional_accrued_interest    2252050       0.9962
member_id                                     2260701       1.0000

[151 rows x 2 columns]
```

[16]:

```
rank_nan["precent_nan"].describe()
```

[16]:

```
count    151.000000
mean       0.317794
std        0.416366
min        0.000000
25%        0.000000
50%        0.031100
75%        0.892650
max        1.000000
Name: precent_nan, dtype: float64
```

[17]:

```
rank_nan[rank_nan["precent_nan"] > 0.1].count()
```

[17]:

```
0              59
precent_nan    59
dtype: int64
```

We find that there are 59 out of 151 variables has more than 10% Nan values

0.1.2 Feature Selection

[18]:

```
# only keep Fully Paid and Charged Off for loan_status(y)
df = data[(data['loan_status'] == 'Fully Paid') | (data['loan_status'] ==␣
↪'Charged Off')]
df['loan_repaid'] = df['loan_status'].map({'Fully Paid':1,'Charged Off':0})
df = df.drop('loan_status', axis = 1)
```

```
<ipython-input-18-7c42447ea019>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['loan_repaid'] = df['loan_status'].map({'Fully Paid':1,'Charged Off':0})
```

[19]:
```
# y label
y_label = ['loan_repaid']
# catagorical labels
cata_labels =␣
↪['term','grade','home_ownership','verification_status','purpose','addr_state']
```

[20]:
```
# numerical labels
num_labels =␣
↪['loan_amnt','int_rate','installment','annual_inc','open_acc','pub_rec','revol_bal','revol_
                         'total_acc','pub_rec_bankruptcies',␣
↪'last_fico_range_high','issue_d']
#last_pymnt_amnt
labels = y_label + cata_labels + num_labels
labels
```

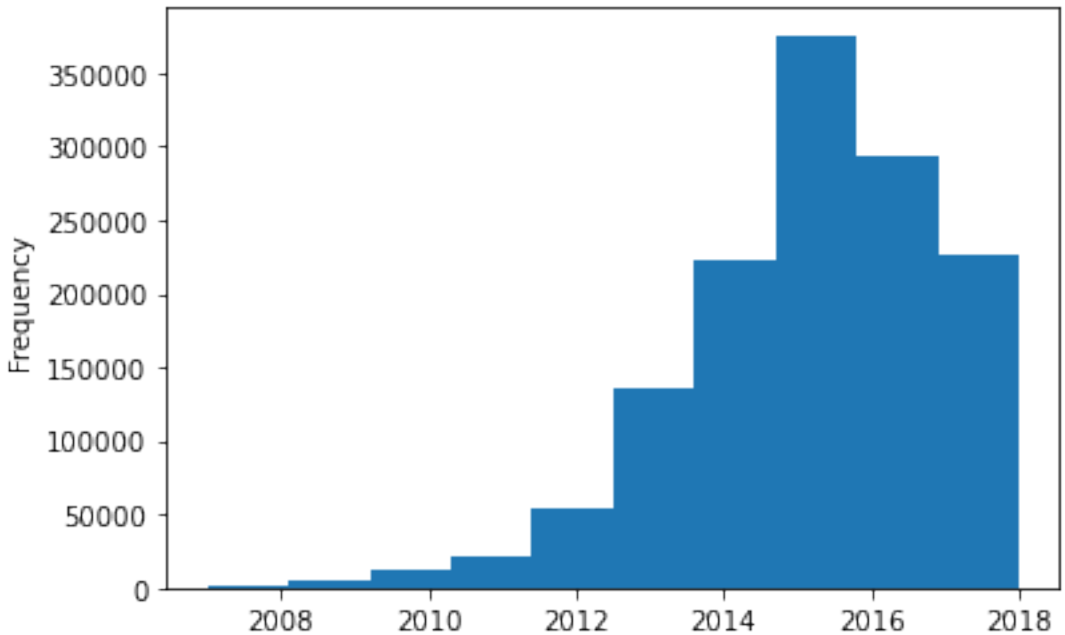
[20]:
```
['loan_repaid',
 'term',
 'grade',
 'home_ownership',
 'verification_status',
 'purpose',
 'addr_state',
 'loan_amnt',
 'int_rate',
 'installment',
 'annual_inc',
 'open_acc',
 'pub_rec',
 'revol_bal',
 'revol_util',
 'total_acc',
 'pub_rec_bankruptcies',
 'last_fico_range_high',
 'issue_d']
```

[21]:
```
# redefine df
df = df[labels]
```

0.1.3 Selection Data by Year

[22]:
```
df.issue_d = pd.to_datetime(df.issue_d)
df['issue_yr'] = df.issue_d.dt.year
df['issue_yr'].plot.hist()
```

[22]:
```
<AxesSubplot:ylabel='Frequency'>
```



[23]:
```
df['issue_yr'].value_counts()
```

[23]:
```
2015    375545
2016    293095
2014    223102
2017    169300
2013    134804
2018     56311
2012     53367
2011     21721
2010     11536
2009      4716
2008      1562
2007       251
Name: issue_yr, dtype: int64
```

[24]:
```
df.head()
```

[24]:
```
   loan_repaid       term grade home_ownership verification_status  \
0            1  36 months     C       MORTGAGE        Not Verified
1            1  36 months     C       MORTGAGE        Not Verified
2            1  60 months     B       MORTGAGE        Not Verified
4            1  60 months     F       MORTGAGE     Source Verified
5            1  36 months     C           RENT     Source Verified

              purpose addr_state  loan_amnt  int_rate  installment  \
```

```
0  debt_consolidation         PA     3600.0      13.99       123.03
1      small_business         SD    24700.0      11.99       820.28
2    home_improvement         IL    20000.0      10.78       432.66
4      major_purchase         PA    10400.0      22.45       289.91
5  debt_consolidation         GA    11950.0      13.44       405.18

   annual_inc  open_acc  pub_rec  revol_bal  revol_util  total_acc  \
0     55000.0       7.0      0.0     2765.0        29.7       13.0
1     65000.0      22.0      0.0    21470.0        19.2       38.0
2     63000.0       6.0      0.0     7869.0        56.2       18.0
4    104433.0      12.0      0.0    21929.0        64.5       35.0
5     34000.0       5.0      0.0     8822.0        68.4        6.0

   pub_rec_bankruptcies  last_fico_range_high    issue_d  issue_yr
0                   0.0                 564.0 2015-12-01      2015
1                   0.0                 699.0 2015-12-01      2015
2                   0.0                 704.0 2015-12-01      2015
4                   0.0                 704.0 2015-12-01      2015
5                   0.0                 759.0 2015-12-01      2015
```

[25]:
```
df = df[df['issue_yr'] == 2018.0].drop(['issue_yr','issue_d'], axis = 1)
```

[26]:
```
df.head()
```

[26]:
```
        loan_repaid        term grade home_ownership verification_status  \
421101            1   36 months     A           RENT     Source Verified
421113            1   36 months     B            OWN        Not Verified
421120            1   36 months     B       MORTGAGE            Verified
421135            1   36 months     D            OWN            Verified
421137            1   60 months     D            OWN     Source Verified

                   purpose addr_state  loan_amnt  int_rate  installment  \
421101      major_purchase         WA     3000.0      7.34        93.10
421113               other         GA     5000.0     11.98       166.03
421120    home_improvement         TX     7000.0     11.98       232.44
421135  debt_consolidation         FL    30000.0     21.85      1143.39
421137               house         NY    21000.0     20.39       560.94

        annual_inc  open_acc  pub_rec  revol_bal  revol_util  total_acc  \
421101     52000.0       7.0      0.0      141.0         0.5       30.0
421113     55000.0      14.0      1.0    11449.0        33.9       24.0
421120     40000.0      13.0      0.0     5004.0        36.0       29.0
421135     57000.0      11.0      0.0    29222.0        53.2       26.0
421137     85000.0      15.0      0.0    14591.0        34.2       27.0

        pub_rec_bankruptcies  last_fico_range_high
421101                   0.0                 764.0
```

```
421113                   1.0                  679.0
421120                   0.0                  644.0
421135                   0.0                  699.0
421137                   0.0                  659.0
```

0.2 2. Exploratory Data Analysis - EDA

0.2.1 Missing Value

[27]:
```python
# Checking the missing values
df.isnull().sum()
```

[27]:
```
loan_repaid              0
term                     0
grade                    0
home_ownership           0
verification_status      0
purpose                  0
addr_state               0
loan_amnt                0
int_rate                 0
installment              0
annual_inc               0
open_acc                 0
pub_rec                  0
revol_bal                0
revol_util              74
total_acc                0
pub_rec_bankruptcies     0
last_fico_range_high     0
dtype: int64
```

[28]:
```python
# data cleaning
df = df.dropna()
```

[29]:
```python
df.isnull().sum()
```

[29]:
```
loan_repaid             0
term                    0
grade                   0
home_ownership          0
verification_status     0
purpose                 0
addr_state              0
loan_amnt               0
int_rate                0
installment             0
```

```
annual_inc              0
open_acc                0
pub_rec                 0
revol_bal               0
revol_util              0
total_acc               0
pub_rec_bankruptcies    0
last_fico_range_high    0
dtype: int64
```

0.2.2 Data Info

[30]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 56237 entries, 421101 to 1611872
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   loan_repaid           56237 non-null  int64
 1   term                  56237 non-null  object
 2   grade                 56237 non-null  object
 3   home_ownership        56237 non-null  object
 4   verification_status   56237 non-null  object
 5   purpose               56237 non-null  object
 6   addr_state            56237 non-null  object
 7   loan_amnt             56237 non-null  float64
 8   int_rate              56237 non-null  float64
 9   installment           56237 non-null  float64
 10  annual_inc            56237 non-null  float64
 11  open_acc              56237 non-null  float64
 12  pub_rec               56237 non-null  float64
 13  revol_bal             56237 non-null  float64
 14  revol_util            56237 non-null  float64
 15  total_acc             56237 non-null  float64
 16  pub_rec_bankruptcies  56237 non-null  float64
 17  last_fico_range_high  56237 non-null  float64
dtypes: float64(11), int64(1), object(6)
memory usage: 8.2+ MB
```

We have 6 catagorical variables and 13 numerical variables.

0.2.3 Catagorical Variables

[31]:
```python
df.select_dtypes(['object']).columns
```

[31]:
```
Index(['term', 'grade', 'home_ownership', 'verification_status', 'purpose',
       'addr_state'],
      dtype='object')
```

0.2.4 term

[32]:
```
# term
df['term'].value_counts()
```

[32]:
```
 36 months    41209
 60 months    15028
Name: term, dtype: int64
```

[33]:
```
df['term'] = df['term'].apply(lambda term: int(term[:3])) # term[:3] take first␣
↪3 values
```

[34]:
```
df['term'].value_counts()
```

[34]:
```
36    41209
60    15028
Name: term, dtype: int64
```

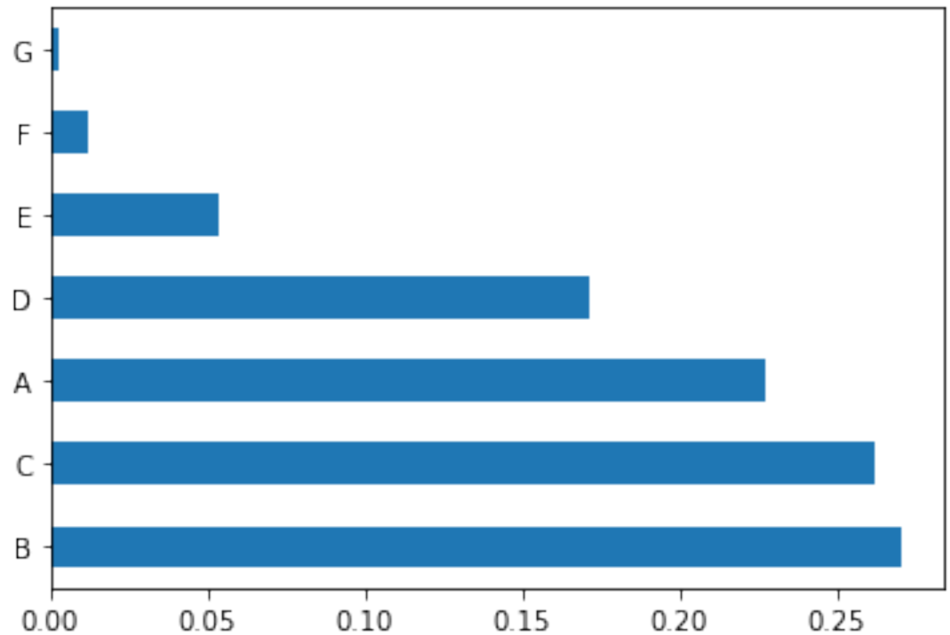
We change variable term from catagorical to numerical

0.2.5 grade

[35]:
```
# grade
df['grade'].value_counts(normalize=True)
```

[35]:
```
B    0.270551
C    0.261998
A    0.227750
D    0.171613
E    0.053808
F    0.011736
G    0.002543
Name: grade, dtype: float64
```

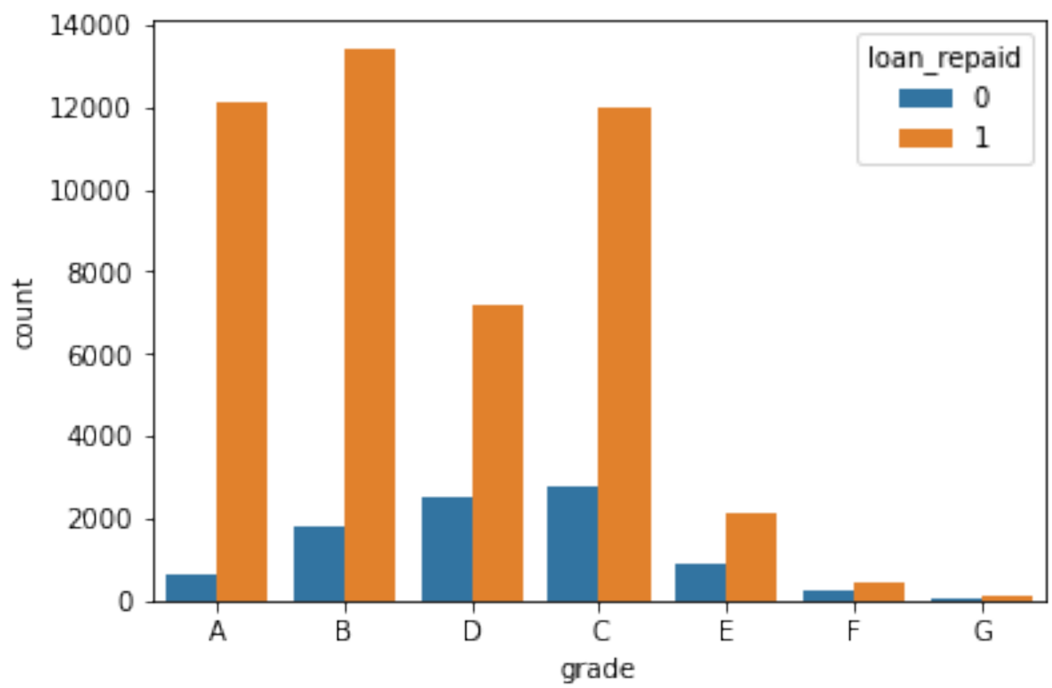
[36]:
```
#plot the bar graph of percentage job categories
df['grade'].value_counts(normalize=True).plot.barh()
plt.show()
```



[37]:

```
sns.countplot(x='grade',data=df,hue='loan_repaid')
```

[37]: <AxesSubplot:xlabel='grade', ylabel='count'>



[38]:
```
# find % 0/1+0 in each grade
g0 = df[df['loan_repaid']==0].groupby("grade").count()['loan_repaid']
g1 = df.groupby("grade").count()['loan_repaid']
g = g0/g1
g
```

[38]:
```
grade
A    0.051530
B    0.117121
C    0.186440
D    0.257797
E    0.290813
F    0.363636
G    0.391608
Name: loan_repaid, dtype: float64
```

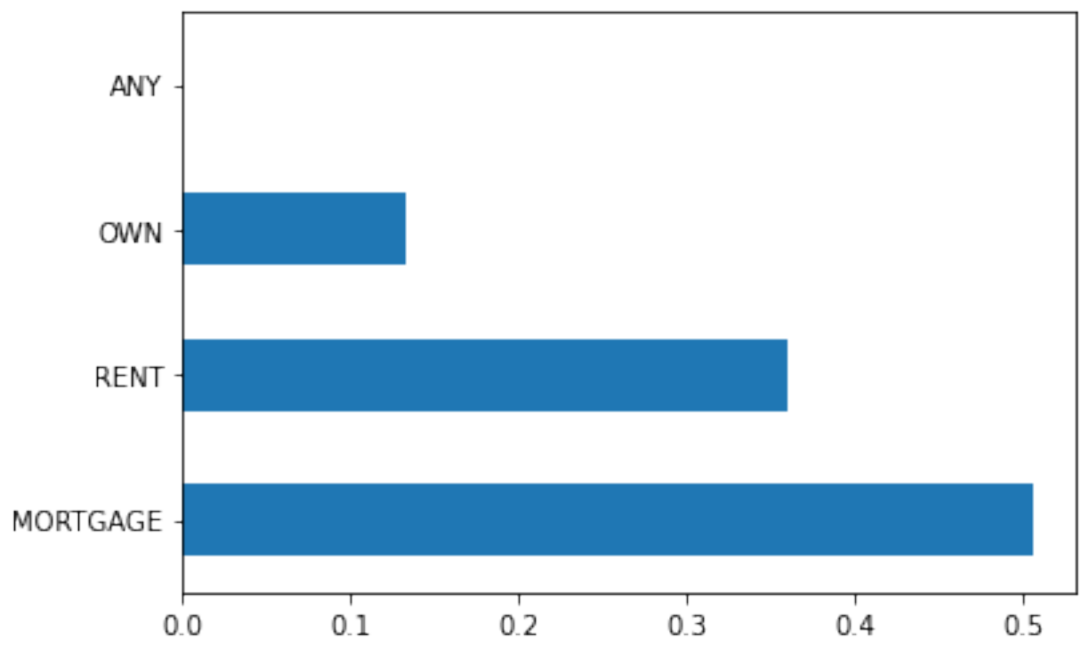
We find that there are 45%+ charge off in Grade F and G; Grade is a valuable factor

0.2.6 home_ownership

[39]:
```
df['home_ownership'].value_counts(normalize=False)
```

[39]:
```
MORTGAGE    28487
RENT        20219
OWN          7508
ANY            23
Name: home_ownership, dtype: int64
```

[40]:
```
#plot the bar graph of percentage job categories
df['home_ownership'].value_counts(normalize=True).plot.barh()
plt.show()
```



[41]:
```
df['home_ownership'] = df['home_ownership'].replace(['NONE','ANY'],'OTHER')
```

[42]:
```
df['home_ownership'].value_counts(normalize=False)
```

[42]:
```
MORTGAGE    28487
RENT        20219
OWN          7508
OTHER          23
Name: home_ownership, dtype: int64
```

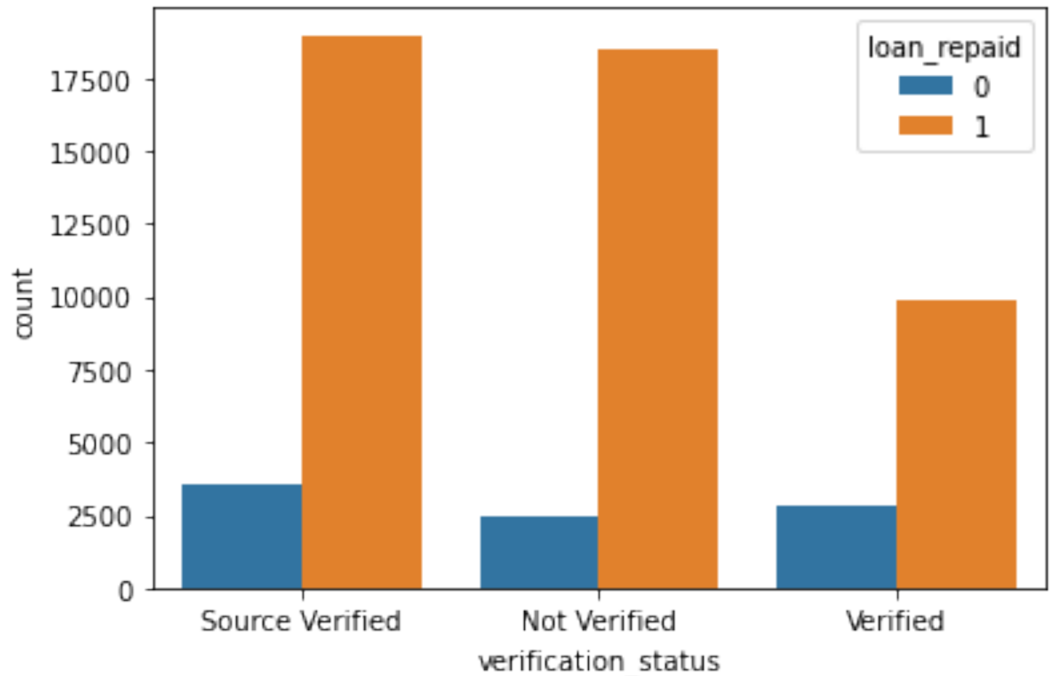
0.2.7 verification_status

[43]:
```
df['verification_status'].value_counts(normalize=False)
```

[43]:
```
Source Verified    22513
Not Verified       20993
Verified           12731
Name: verification_status, dtype: int64
```

[44]:
```
sns.countplot(x='verification_status',data=df,hue='loan_repaid')
```

[44]:
```
<AxesSubplot:xlabel='verification_status', ylabel='count'>
```



[45]:
```
# find % 0/1+0 in each grade
v0 = df[df['loan_repaid']==0].groupby("verification_status").
↪count()['loan_repaid']
v1 = df.groupby("verification_status").count()['loan_repaid']
v = v0/v1
v
```

[45]:
```
verification_status
Not Verified       0.117182
Source Verified    0.157465
Verified           0.223706
Name: loan_repaid, dtype: float64
```

The Not Verified users has higher charge off rate

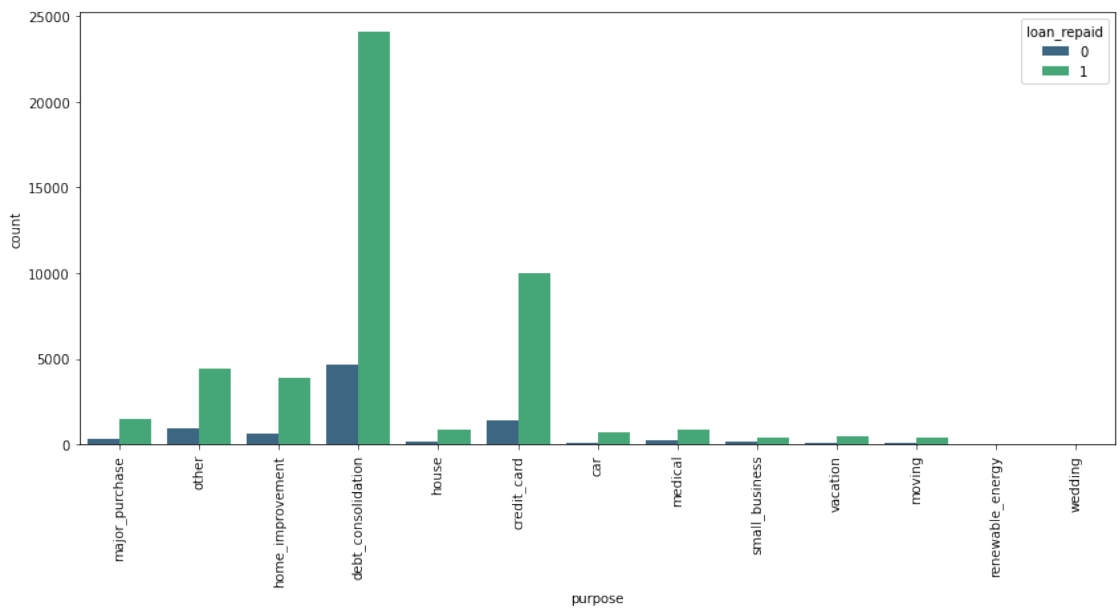
0.2.8 purpose

[46]:
```
df['purpose'].value_counts(normalize=False)
```

[46]:
```
debt_consolidation    28699
credit_card           11388
other                  5383
home_improvement       4514
major_purchase         1804
medical                1031
```

```
house                  996
car                    774
small_business         569
vacation               533
moving                 507
renewable_energy        37
wedding                  2
Name: purpose, dtype: int64
```

[47]:
```python
plt.figure(figsize=(14,6))
sns.countplot(data=df,x='purpose', hue='loan_repaid', palette='viridis');
plt.xticks(rotation=90);
```



[48]:
```python
# find % 0/1+0 in each purpose
p0 = df[df['loan_repaid']==0].groupby("purpose").count()['loan_repaid']
p1 = df.groupby("purpose").count()['loan_repaid']
p = p0/p1
p.sort_values()
```

[48]:
```
purpose
car                   0.120155
credit_card           0.123639
home_improvement      0.140009
debt_consolidation    0.161469
renewable_energy      0.162162
vacation              0.170732
house                 0.178715
```

```
other                   0.178897
major_purchase          0.193459
medical                 0.205626
moving                  0.207101
small_business          0.318102
wedding                 0.500000
Name: loan_repaid, dtype: float64
```

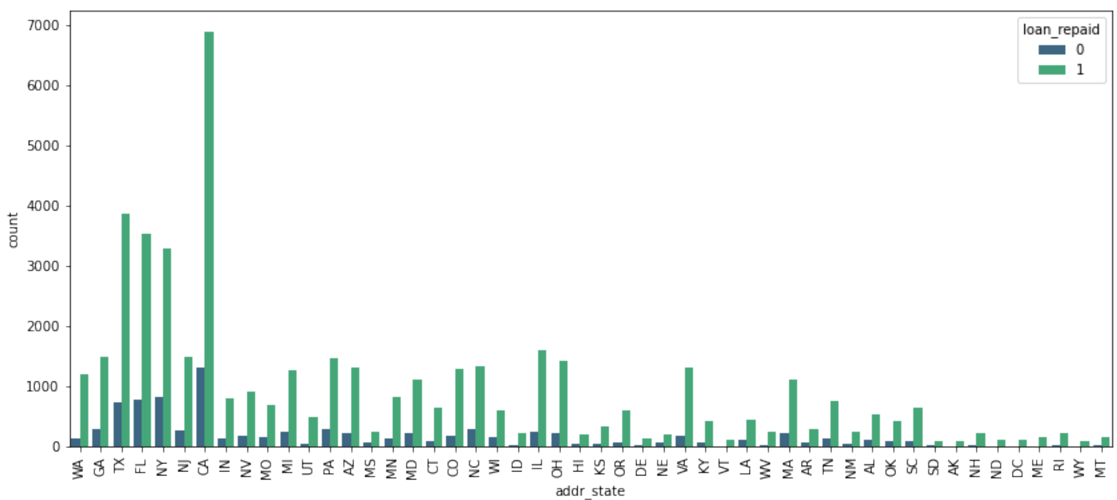
0.2.9 addr_state

[49]:
```
df['addr_state'].value_counts()
```

[49]:
```
CA    8207
TX    4599
FL    4330
NY    4130
IL    1836
GA    1784
NJ    1768
PA    1763
NC    1637
OH    1631
AZ    1526
MI    1513
VA    1499
CO    1477
MD    1348
WA    1332
MA    1331
NV    1091
MN     972
IN     937
TN     894
MO     858
WI     751
SC     747
CT     743
OR     682
AL     662
LA     559
UT     538
OK     531
KY     508
KS     382
AR     377
MS     324
NM     284
```

```
NE      269
WV      264
NH      257
HI      250
ID      250
RI      244
MT      168
DE      164
ME      158
DC      122
ND      117
VT      116
SD      112
WY      101
AK       94
Name: addr_state, dtype: int64
```

[50]:
```python
plt.figure(figsize=(14,6))
sns.countplot(data=df,x='addr_state', hue='loan_repaid', palette='viridis');
plt.xticks(rotation=90);
```



[51]:
```python
# find % 0/1+0 in each states
s0 = df[df['loan_repaid']==0].groupby("addr_state").count()['loan_repaid']
s1 = df.groupby("addr_state").count()['loan_repaid']
s = s0/s1
s.sort_values()
```

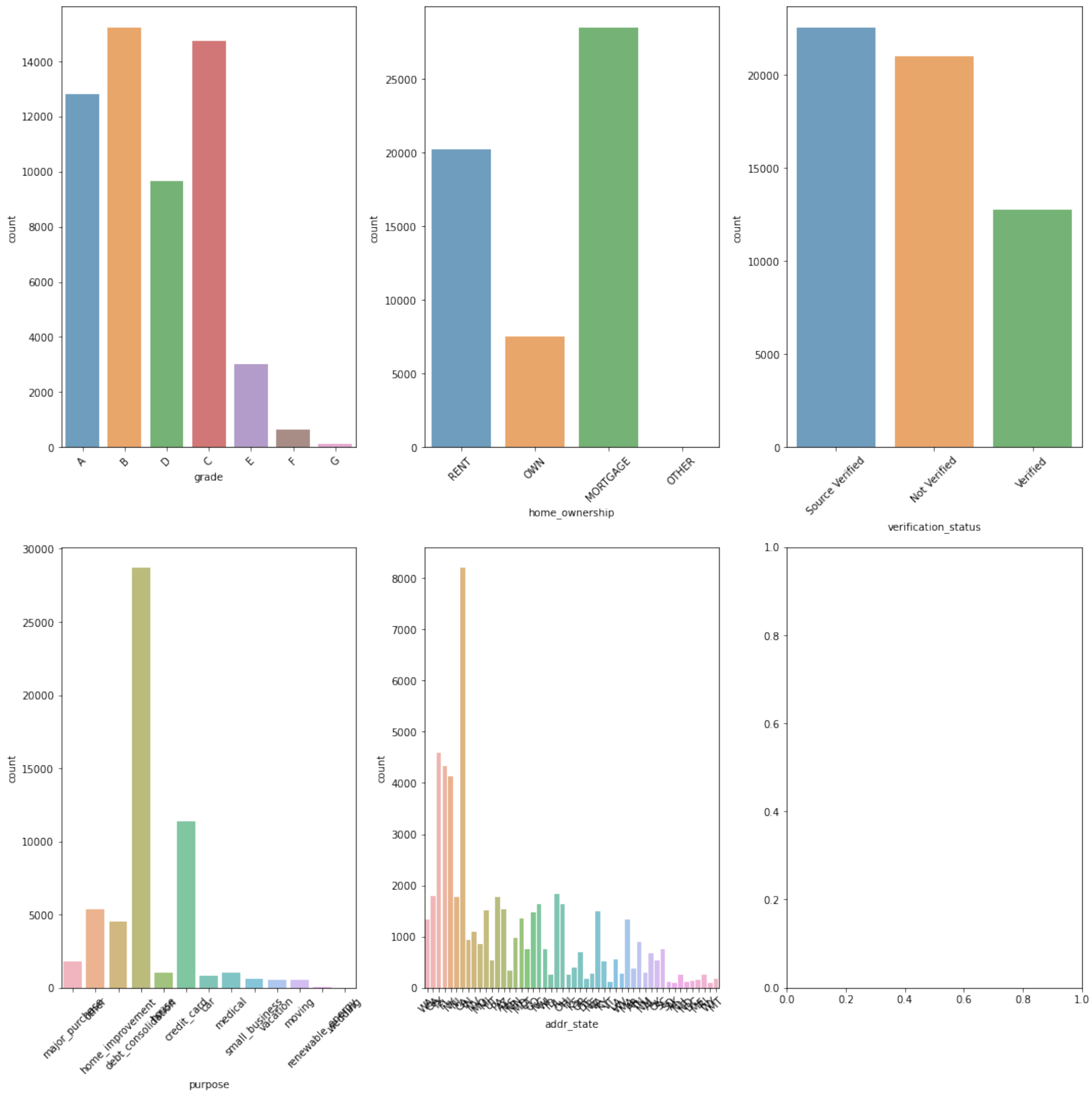
[51]:
```
addr_state
ME     0.050633
UT     0.083643
```

ID	0.084000
VT	0.086207
WV	0.090909
DC	0.098361
WY	0.099010
OR	0.099707
WA	0.103604
ND	0.111111
MT	0.113095
CT	0.114401
RI	0.118852
CO	0.120515
VA	0.125417
NH	0.128405
SC	0.128514
IL	0.132898
OH	0.133047
AK	0.138298
KS	0.138743
MN	0.139918
TN	0.140940
AZ	0.141547
KY	0.141732
IN	0.150480
NJ	0.151018
TX	0.158078
CA	0.159132
MI	0.161930
NM	0.161972
PA	0.162223
MA	0.164538
GA	0.165359
NV	0.168653
MD	0.173591
AL	0.175227
NC	0.178375
OK	0.178908
FL	0.181293
DE	0.182927
LA	0.187835
MO	0.193473
HI	0.196000
WI	0.197071
NY	0.201937
MS	0.206790
AR	0.209549
SD	0.214286

```
NE    0.219331
Name: loan_repaid, dtype: float64
```

[52]:
```python
df_not_num = df.select_dtypes(include=['object'])
fig, axes = plt.subplots(round(len(df_not_num.columns) / 3), 3, figsize=(15,␣
 ↪15))

for i, ax in enumerate(fig.axes):
    if i < len(df_not_num.columns):
        ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
        sns.countplot(x=df_not_num.columns[i], alpha=0.7, data=df_not_num,␣
 ↪ax=ax)

fig.tight_layout()
```

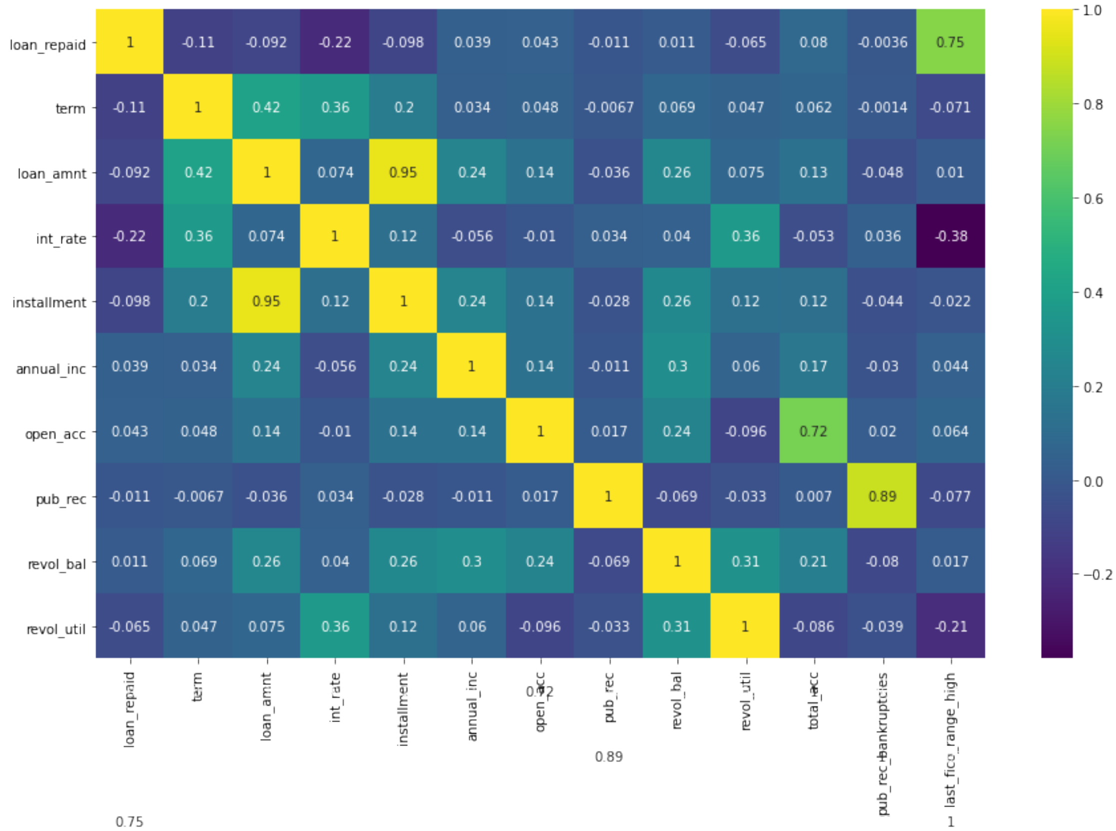
```
<ipython-input-52-042cc3cb982a>:6: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
<ipython-input-52-042cc3cb982a>:6: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
<ipython-input-52-042cc3cb982a>:6: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
<ipython-input-52-042cc3cb982a>:6: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
<ipython-input-52-042cc3cb982a>:6: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
```



0.2.10 Numerical Variables

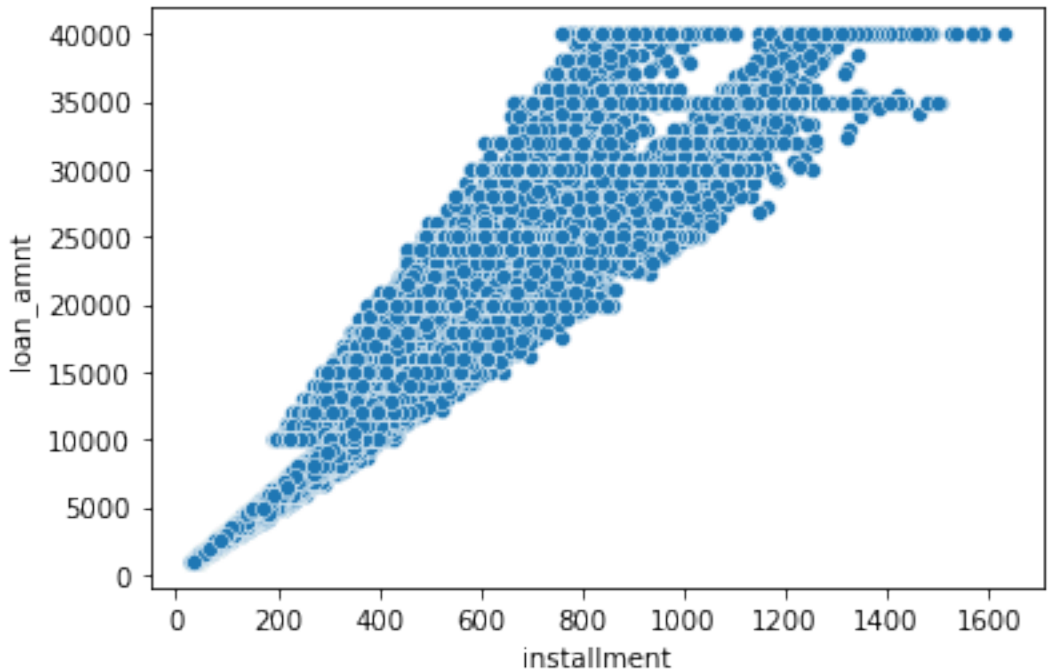
[53]:
```
plt.figure(figsize=(15,9))
sns.heatmap(df.corr(),annot=True,cmap='viridis')
plt.ylim(10, 0)
```

[53]: (10.0, 0.0)




```
[55]: sns.scatterplot(x='installment',y='loan_amnt',data=df)
```

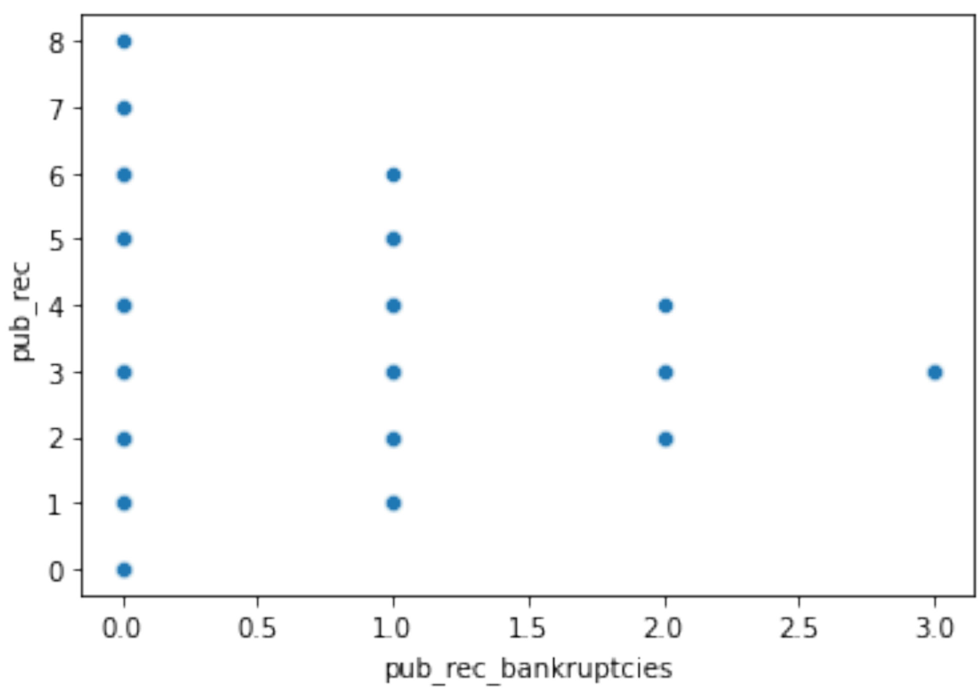
```
[55]: <AxesSubplot:xlabel='installment', ylabel='loan_amnt'>
```



We find a strong correlation between installment and loan_amnt;but why?

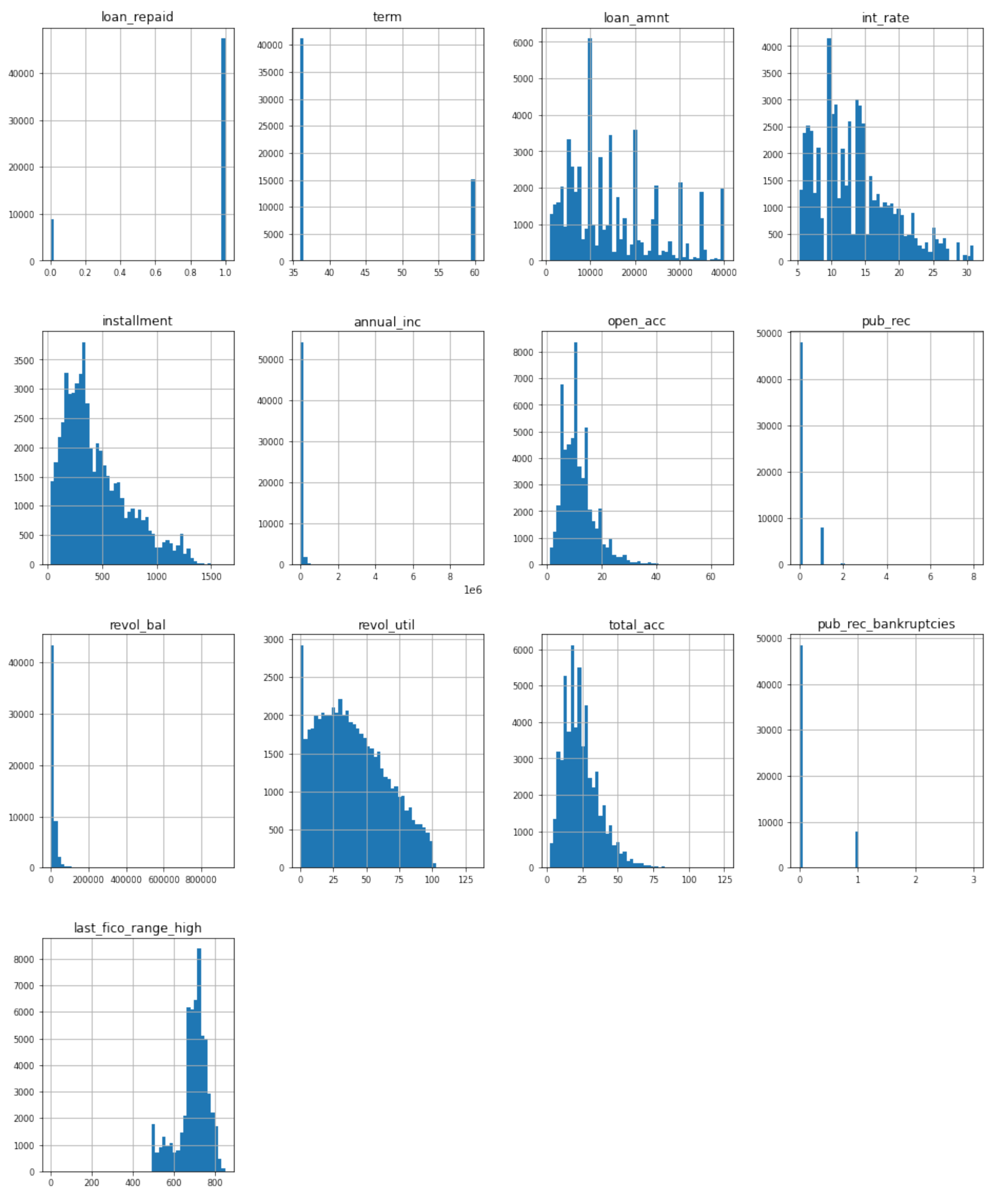
[58]:
```
sns.scatterplot(x='pub_rec_bankruptcies',y='pub_rec',data=df,)
```

[58]: <AxesSubplot:xlabel='pub_rec_bankruptcies', ylabel='pub_rec'>



[59]:

```
df.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8);
```



0.2.11 Transfer catagorical variables to dummy variables

[60]:
```
# transfer catagorical varibales to dummy variables
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
df = df.apply(le.fit_transform)
df.head()
```

[60]:
```
        loan_repaid  term  grade  home_ownership  verification_status  \
421101            1     0      0               3                    1
421113            1     0      1               2                    0
421120            1     0      1               0                    2
421135            1     0      3               2                    2
421137            1     1      3               2                    1

        purpose  addr_state  loan_amnt  int_rate  installment  annual_inc  \
421101        5          46         73        14          956        2179
421113        8          10        144        45         2082        2379
421120        3          42        223        45         3140        1373
421135        2           9       1064        83        11988        2513
421137        4          33        744        80         8231        3851

        open_acc  pub_rec  revol_bal  revol_util  total_acc  \
421101         6        0        140           5         28
421113        13        1      10675         339         22
421120        12        0       4765         360         27
421135        10        0      20995         532         24
421137        14        0      13230         342         25

        pub_rec_bankruptcies  last_fico_range_high
421101                     0                    54
421113                     1                    37
421120                     0                    30
421135                     0                    41
421137                     0                    33
```

[61]:
```
df.shape
```

[61]: (56237, 18)

0.2.12 Split the Data & Standardization - Feature Scaling

[62]:
```
# split the data randomly into 70% training data and 30% test data.
from sklearn.model_selection import train_test_split
# split the data randomly into 70% training data and 30% test data.
X = df.drop('loan_repaid', axis = 1)
y = df['loan_repaid']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=0)
```

[63]:
```
# Feature Scaling - only on the training data and not on test data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

[ ]:
```

```

0.2.13 Random Forest

[64]:
```
from sklearn import ensemble
params = {
    'max_features': 6,
    'n_estimators': 100, # 100 trees in the forest
    'max_depth': 100, # maximum depth of the tree is 100
    'criterion': "gini",
    'n_jobs': -1,
    'min_samples_leaf': 1

}
rf_model = ensemble.RandomForestClassifier(**params)
```

[65]:
```
# fit the data
rf_model.fit(X_train, y_train)
```

[65]:
```
RandomForestClassifier(max_depth=100, max_features=6, n_jobs=-1)
```

RF Result

[66]:
```
print('Training accuracy of RF:', rf_model.score(X_train, y_train))
print('Test accuracy of RF:', rf_model.score(X_test, y_test))
```

```
Training accuracy of RF: 1.0
Test accuracy of RF: 0.9628971076339498
```

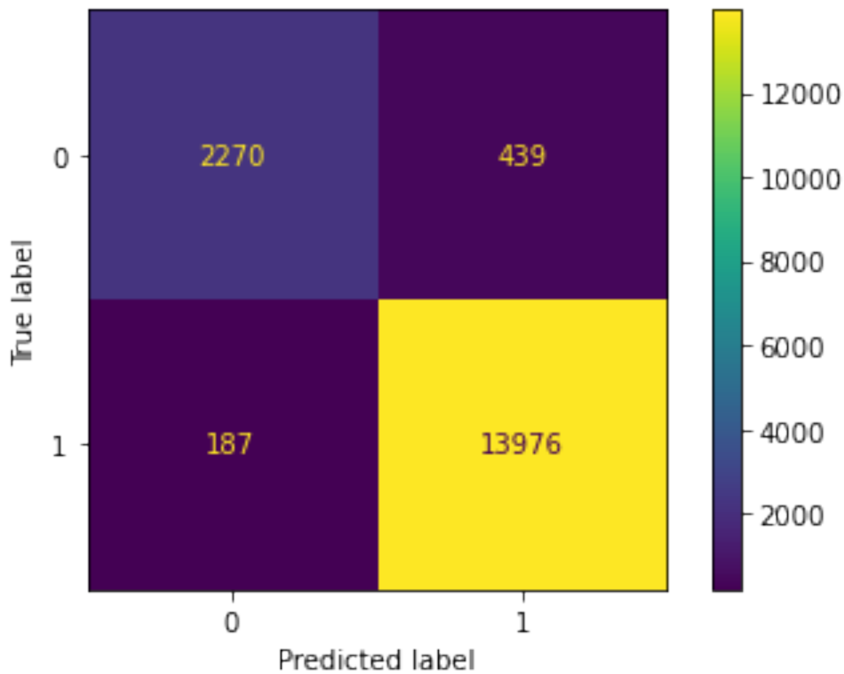
[67]:
```
# result and solution
from sklearn.metrics import␣
↪accuracy_score,plot_confusion_matrix,classification_report
preds = rf_model.predict(X_test)
print(classification_report(y_test,preds))
```

```
              precision    recall  f1-score   support
```

```
              0       0.92      0.84      0.88      2709
              1       0.97      0.99      0.98     14163

       accuracy                           0.96     16872
      macro avg       0.95      0.91      0.93     16872
   weighted avg       0.96      0.96      0.96     16872
```

[68]:
```python
plot_confusion_matrix(rf_model,X_test,y_test)
```

[68]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7faa8a5685b0>

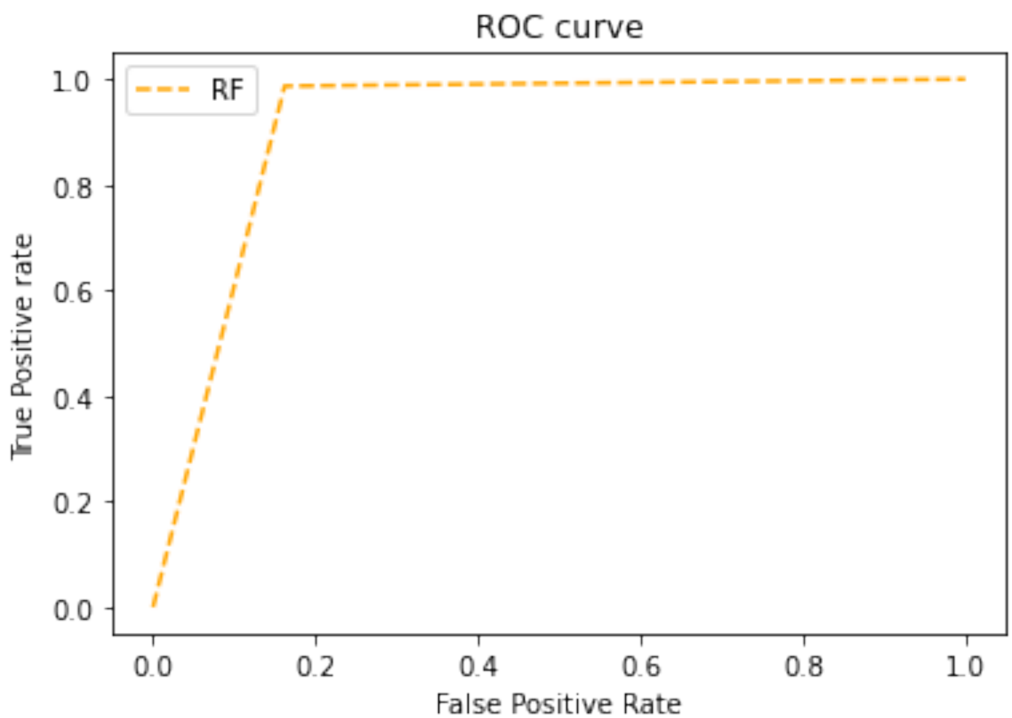


[69]:
```python
from sklearn.metrics import roc_auc_score
# auc scores
auc_score = roc_auc_score(y_test, preds)
# calculate AUC
print('AUC: %.3f' % auc_score)
```

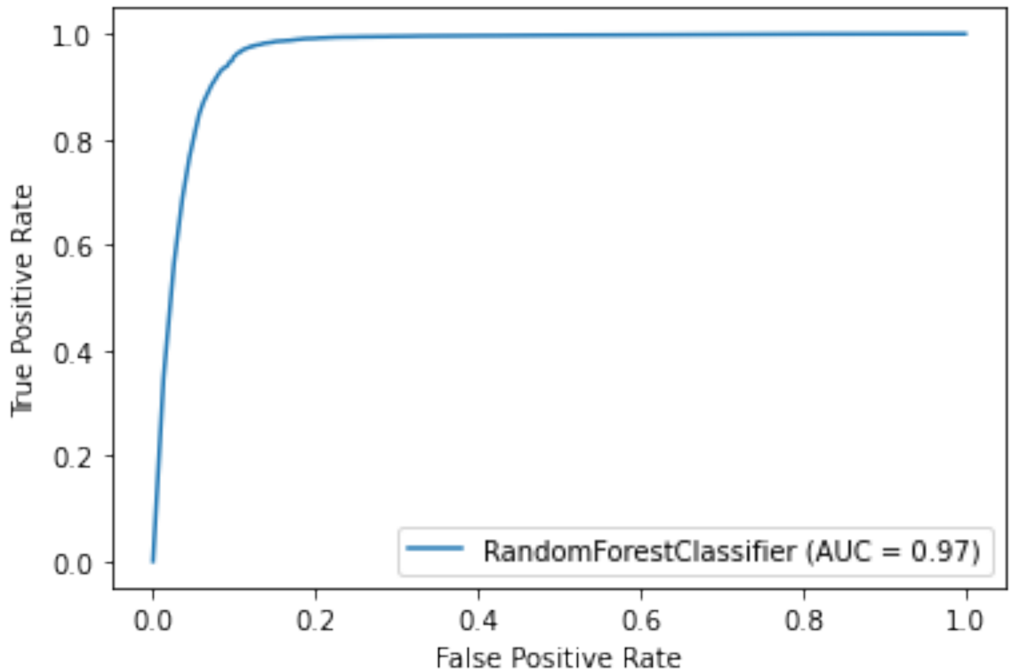
AUC: 0.912

[70]:
```python
# calculate roc curves
from sklearn.metrics import roc_curve
fpr1, tpr1, thresh1 = roc_curve(y_test,preds, pos_label=1)
# plot the roc curve for the model
```

```python
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='RF')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()
# https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/
```



[71]:
```python
from sklearn.metrics import plot_roc_curve

#disp = plot_roc_curve(xgb_clf, X_test, y_test)
plot_roc_curve(rf_model,X_test,y_test) #ax=disp.ax_
plt.show()
```



[72]:
```python
# importances of variables
feat_labels = list(df.columns)
feat_labels.remove('loan_repaid')

variable = feat_labels
importances = rf_model.feature_importances_.tolist()
dic = {"variable":variable, "importances":importances }
show = pd.DataFrame(dic).sort_values(by = "importances", ascending = False)
show.head(18)
```

[72]:

	variable	importances
16	last_fico_range_high	0.743577
7	int_rate	0.031075
8	installment	0.027400
9	annual_inc	0.026595
12	revol_bal	0.026320
13	revol_util	0.025637
14	total_acc	0.019883
6	loan_amnt	0.019349
5	addr_state	0.017003
10	open_acc	0.015384
1	grade	0.014947
4	purpose	0.009766
2	home_ownership	0.007428
3	verification_status	0.006114

```
0                  term     0.003950
11              pub_rec     0.003042
15  pub_rec_bankruptcies     0.002532
```

[73]:
```
show.to_csv('file_name.csv')
```

0.2.14 Neural network

[74]:
```
from sklearn.neural_network import MLPClassifier

nn_model = MLPClassifier(solver='lbfgs',alpha = 0.0001,
                    hidden_layer_sizes=(10, 10, 10), max_iter=1000)
nn_model.fit(X_train, y_train)
```

```
/Users/wenweiwu/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:471:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

[74]:
```
MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000, solver='lbfgs')
```

NN Result

[75]:
```
print('Training accuracy of NN:', nn_model.score(X_train, y_train))
print('Test accuracy of NN:', nn_model.score(X_test, y_test))
```

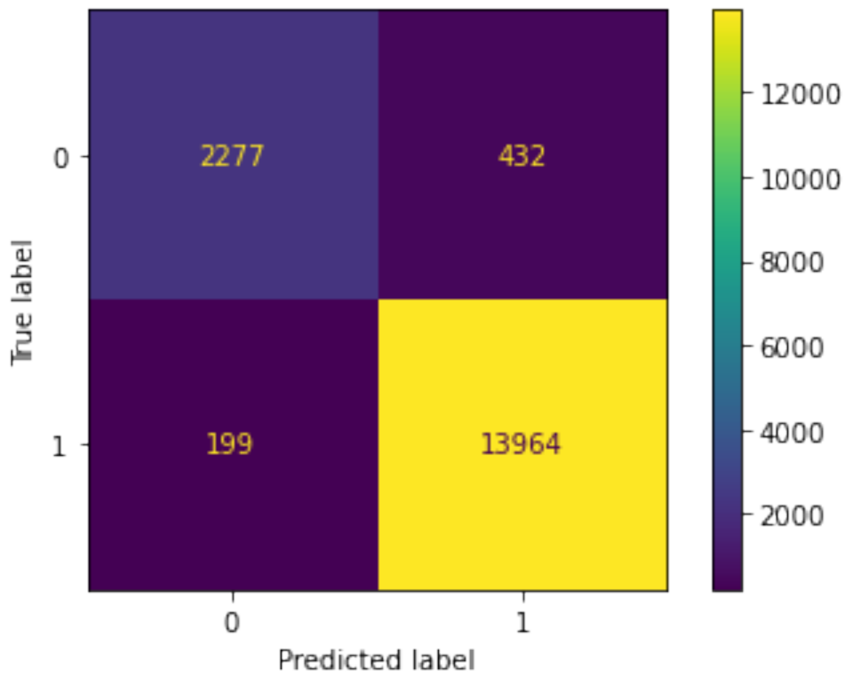
```
Training accuracy of NN: 0.9668995300393751
Test accuracy of NN: 0.9626007586533902
```

[76]:
```
# result and solution
from sklearn.metrics import␣
↪accuracy_score,plot_confusion_matrix,classification_report
preds = nn_model.predict(X_test)
print(classification_report(y_test,preds))
```

```
              precision    recall  f1-score   support

           0       0.92      0.84      0.88      2709
           1       0.97      0.99      0.98     14163

    accuracy                           0.96     16872
   macro avg       0.94      0.91      0.93     16872
weighted avg       0.96      0.96      0.96     16872
```

[77]:
```
plot_confusion_matrix(nn_model,X_test,y_test)
```

[77]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7faa8a34de80>

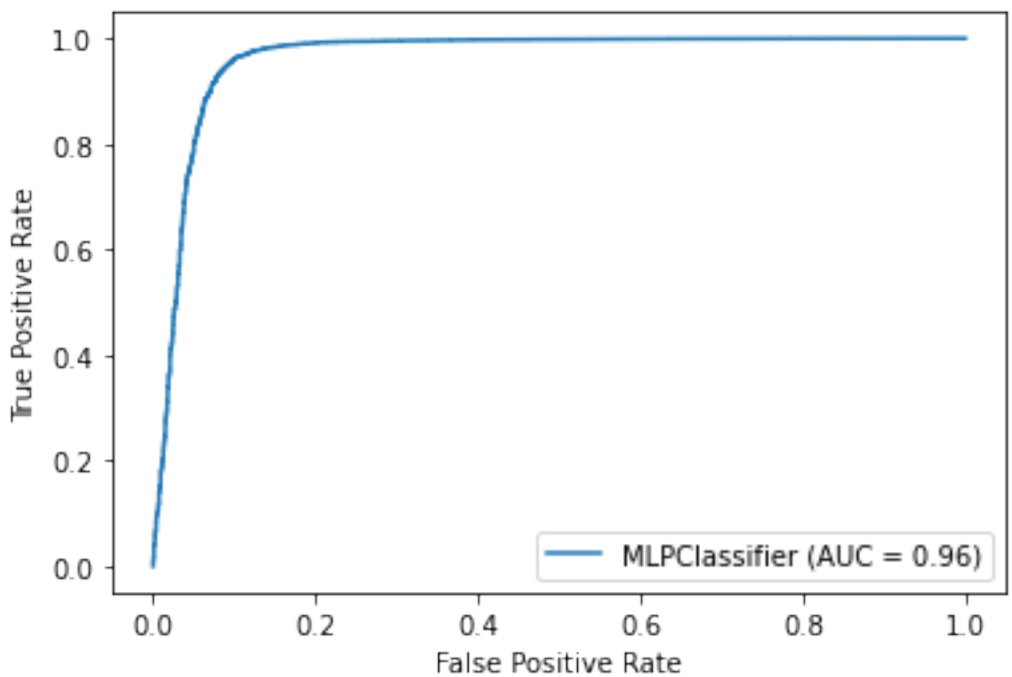


[78]:
```
from sklearn.metrics import roc_auc_score
# auc scores
auc_score = roc_auc_score(y_test, preds)
# calculate AUC
print('AUC: %.3f' % auc_score)
```

```
AUC: 0.913
```

[79]:
```
plot_roc_curve(nn_model,X_test,y_test)
```

[79]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7faa96fea640>



[ ]:

```

```

0.2.15 Logistic Regression

Implementing the model

[236]:
```python
import statsmodels.api as sm
logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.082068
         Iterations 10
                           Results: Logit
=====================================================================
Model:                  Logit             Pseudo R-squared:   0.811
Dependent Variable:     loan_repaid       AIC:                9266.5601
Date:                   2022-03-18 13:11  BIC:                9427.4321
No. Observations:       56237             Log-Likelihood:     -4615.3
Df Model:               17                LL-Null:            -24484.
Df Residuals:           56219             LLR p-value:        0.0000
Converged:              1.0000            Scale:              1.0000
No. Iterations:         10.0000
---------------------------------------------------------------------
                      Coef.  Std.Err.    z     P>|z|    [0.025  0.975]
```

```
-------------------------------------------------------------------
term                 -1.1851   0.0911 -13.0134 0.0000 -1.3636 -1.0066
grade                 1.0154   0.0758  13.3930 0.0000  0.8668  1.1640
home_ownership       -0.3194   0.0202 -15.8016 0.0000 -0.3590 -0.2798
verification_status  -0.2639   0.0368  -7.1635 0.0000 -0.3360 -0.1917
purpose              -0.1670   0.0114 -14.6604 0.0000 -0.1893 -0.1446
addr_state           -0.0211   0.0018 -11.4315 0.0000 -0.0247 -0.0175
loan_amnt             0.0024   0.0004   6.7380 0.0000  0.0017  0.0031
int_rate             -0.0500   0.0037 -13.5946 0.0000 -0.0572 -0.0428
installment          -0.0006   0.0000 -17.3000 0.0000 -0.0006 -0.0005
annual_inc            0.0001   0.0000   3.2830 0.0010  0.0000  0.0001
open_acc             -0.0352   0.0070  -5.0316 0.0000 -0.0489 -0.0215
pub_rec              -0.2520   0.1368  -1.8425 0.0654 -0.5202  0.0161
revol_bal            -0.0000   0.0000  -0.6419 0.5210 -0.0000  0.0000
revol_util           -0.0004   0.0001  -2.8395 0.0045 -0.0007 -0.0001
total_acc            -0.0106   0.0031  -3.3998 0.0007 -0.0168 -0.0045
pub_rec_bankruptcies -0.1030   0.1560  -0.6603 0.5091 -0.4088  0.2027
last_fico_range_high  0.1560   0.0024  66.3000 0.0000  0.1514  0.1607
last_pymnt_amnt       0.0002   0.0000  45.7615 0.0000  0.0002  0.0002
===================================================================
```

[240]:
```python
df2 = df
import scipy.stats as stats
df2.annual_inc = stats.zscore(df2.annual_inc)
df2.revol_bal = stats.zscore(df2.revol_bal)
```

[242]:
```python
X2 = df2.drop('loan_repaid', axis = 1)#.to_numpy()
y2 = df2['loan_repaid'].to_numpy()

import statsmodels.api as sm
logit_model=sm.Logit(y2,X2)
result=logit_model.fit()
print(result.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.078201
         Iterations 10
                          Results: Logit
===================================================================
Model:                 Logit              Pseudo R-squared:  0.820
Dependent Variable:    y                  AIC:               8831.6256
Date:                  2022-03-18 13:12   BIC:               8992.4975
No. Observations:      56237              Log-Likelihood:    -4397.8
Df Model:              17                 LL-Null:           -24484.
Df Residuals:          56219              LLR p-value:       0.0000
Converged:             1.0000             Scale:             1.0000
No. Iterations:        10.0000
```

```
======================================================================
                       Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
----------------------------------------------------------------------
term                  -1.2388   0.0941 -13.1705 0.0000 -1.4232 -1.0545
grade                  0.8374   0.0784  10.6850 0.0000  0.6838  0.9910
home_ownership        -0.2278   0.0210 -10.8334 0.0000 -0.2691 -0.1866
verification_status   -0.1750   0.0380  -4.6079 0.0000 -0.2494 -0.1006
purpose               -0.1553   0.0117 -13.2999 0.0000 -0.1782 -0.1325
addr_state            -0.0165   0.0019  -8.6714 0.0000 -0.0202 -0.0128
loan_amnt              0.0018   0.0004   4.9086 0.0000  0.0011  0.0025
int_rate              -0.0340   0.0038  -8.8458 0.0000 -0.0416 -0.0265
installment           -0.0006   0.0000 -16.7740 0.0000 -0.0006 -0.0005
annual_inc             0.5503   0.0343  16.0564 0.0000  0.4831  0.6174
open_acc              -0.0586   0.0070  -8.3476 0.0000 -0.0724 -0.0448
pub_rec               -0.2819   0.1373  -2.0522 0.0402 -0.5511 -0.0127
revol_bal              0.3468   0.0413   8.4025 0.0000  0.2659  0.4277
revol_util            -0.0013   0.0001  -9.5395 0.0000 -0.0016 -0.0011
total_acc             -0.0140   0.0032  -4.3749 0.0000 -0.0202 -0.0077
pub_rec_bankruptcies   0.0463   0.1574   0.2943 0.7685 -0.2621  0.3548
last_fico_range_high   0.1641   0.0025  66.4095 0.0000  0.1593  0.1690
last_pymnt_amnt        0.0002   0.0000  44.6128 0.0000  0.0002  0.0002
======================================================================
```

Logistic Regression

[243]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
```

Log Result

[244]:
```python
print('Training accuracy of LR:', logreg.score(X_train, y_train))
print('Test accuracy of LR:', logreg.score(X_test, y_test))
```

```
Training accuracy of LR: 0.9770862441254922
Test accuracy of LR: 0.9762328117591276
```

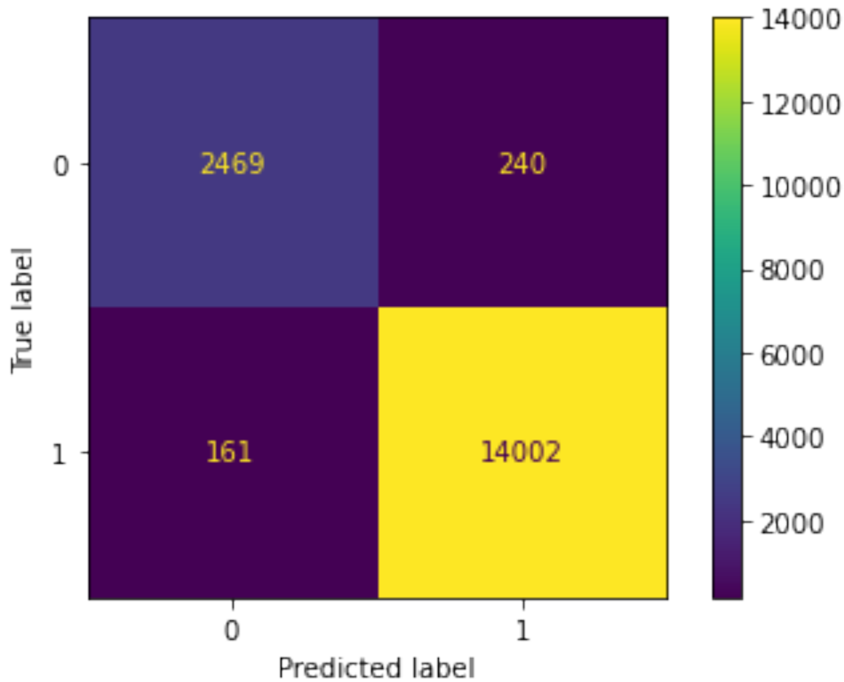
[245]:
```python
# result and solution
from sklearn.metrics import␣
↪accuracy_score,plot_confusion_matrix,classification_report
preds = logreg.predict(X_test)
print(classification_report(y_test,preds))
```

```
              precision    recall  f1-score   support

           0       0.94      0.91      0.92      2709
```

```
            1       0.98      0.99      0.99     14163

     accuracy                           0.98     16872
    macro avg       0.96      0.95      0.96     16872
 weighted avg       0.98      0.98      0.98     16872
```

[246]:
```
plot_confusion_matrix(logreg,X_test,y_test)
```

[246]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff1ea3f0dc0>

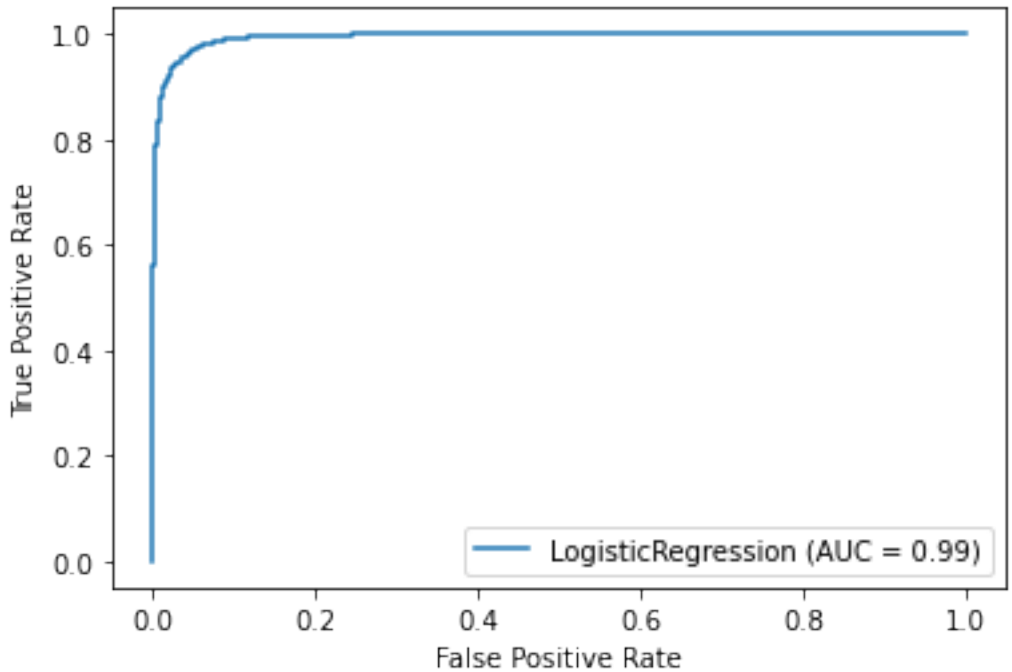


[247]:
```
from sklearn.metrics import roc_auc_score
# auc scores
auc_score = roc_auc_score(y_test, preds)
# calculate AUC
print('AUC: %.3f' % auc_score)
```

```
AUC: 0.950
```

[251]:
```
plot_roc_curve(logreg,X_test,y_test)
```

[251]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7ff1ea3eea00>



[ ]:

```

```

0.2.16 SVM – LinearSVC (linear kernel)

[252]:
```
from sklearn import svm
C = 0.1
ls2 = svm.LinearSVC(C=C, max_iter=10000).fit(X_train, y_train)
```

SVM Result

[253]:
```
print('Training accuracy of SVM:', ls2.score(X_train, y_train))
print('Test accuracy of SVM:', ls2.score(X_test, y_test))
```

```
Training accuracy of SVM: 0.9772132605106059
Test accuracy of SVM: 0.9764698909435752
```

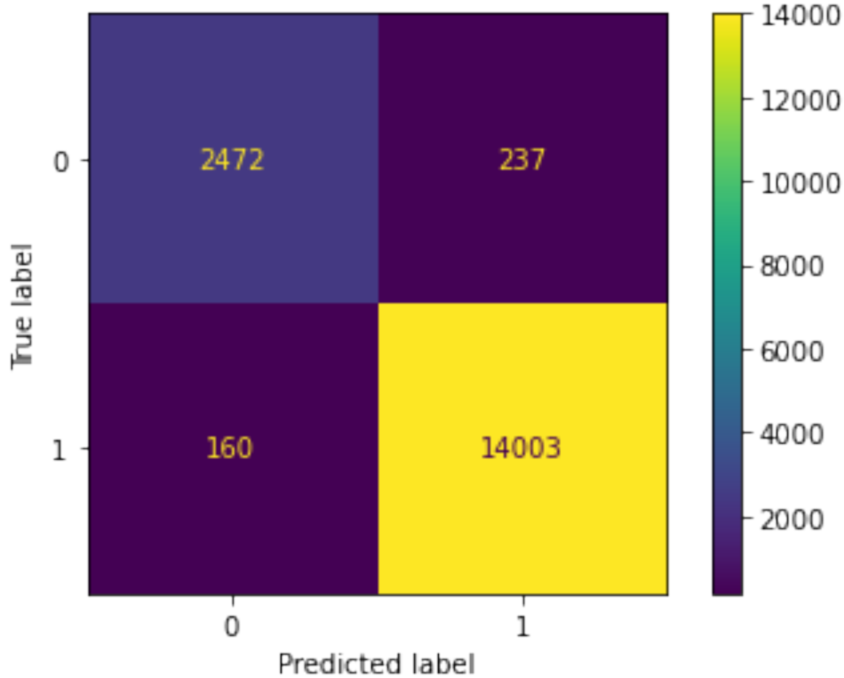
[254]:
```
# result and solution
from sklearn.metrics import␣
↪accuracy_score,plot_confusion_matrix,classification_report
preds = ls2.predict(X_test)
print(classification_report(y_test,preds))
```

```
              precision    recall  f1-score   support

           0       0.94      0.91      0.93      2709
```

```
           1       0.98      0.99      0.99     14163

    accuracy                           0.98     16872
   macro avg       0.96      0.95      0.96     16872
weighted avg       0.98      0.98      0.98     16872
```

[255]:
```
plot_confusion_matrix(ls2,X_test,y_test)
```

[255]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7ff227eb1f40>
```

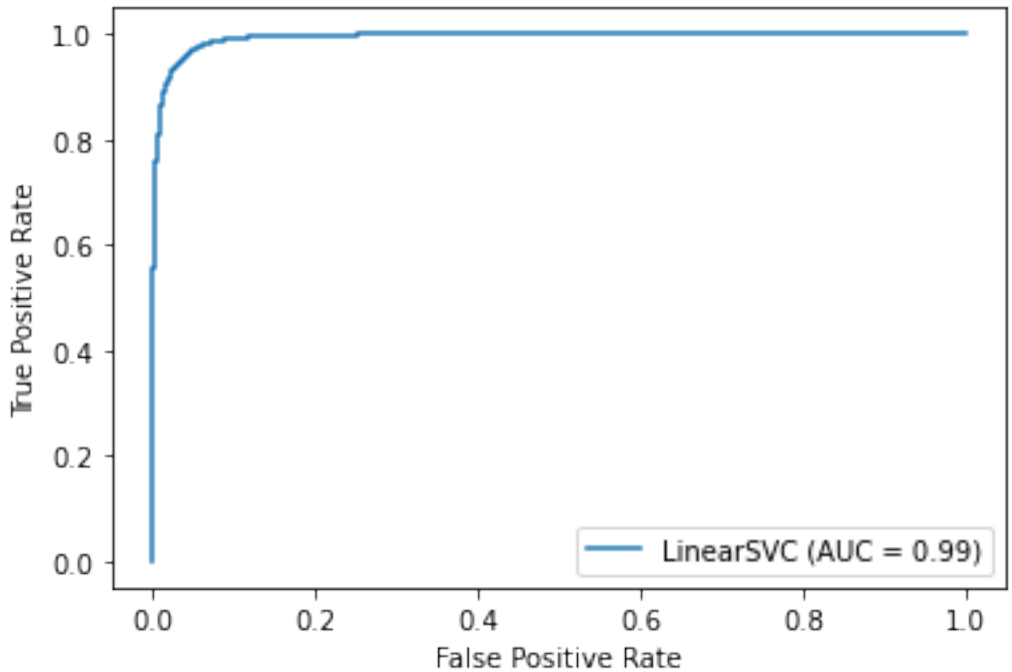


[256]:
```
from sklearn.metrics import roc_auc_score
# auc scores
auc_score = roc_auc_score(y_test, preds)
# calculate AUC
print('AUC: %.3f' % auc_score)
```

```
AUC: 0.951
```

[257]:
```
plot_roc_curve(ls2,X_test,y_test)
```

[257]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7ff22a310a90>
```



[ ]:

0.2.17 KNN

[259]:
```
from sklearn.kernel_ridge import KernelRidge
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

KNN Result

[260]:
```
print('Training accuracy of KNN:', classifier.score(X_train, y_train))
print('Test accuracy of KNN:', classifier.score(X_test, y_test))
```

```
Training accuracy of KNN: 0.9777213260510605
Test accuracy of KNN: 0.9691797060218112
```

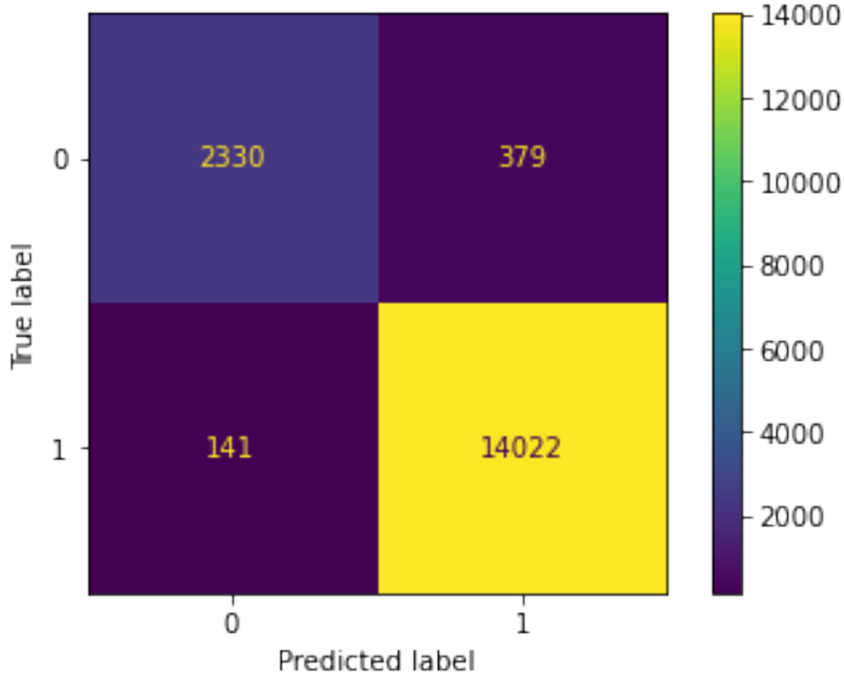
[270]:
```
# result and solution
from sklearn.metrics import␣
 ↪accuracy_score,plot_confusion_matrix,classification_report
preds = classifier.predict(X_test)
print(classification_report(y_test,preds))
```

```
              precision    recall  f1-score   support
```

```
              0       0.94      0.86      0.90      2709
              1       0.97      0.99      0.98     14163

       accuracy                           0.97     16872
      macro avg       0.96      0.93      0.94     16872
   weighted avg       0.97      0.97      0.97     16872
```

[271]:
```
plot_confusion_matrix(classifier,X_test,y_test)
```

[271]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7ff227e8d640>
```

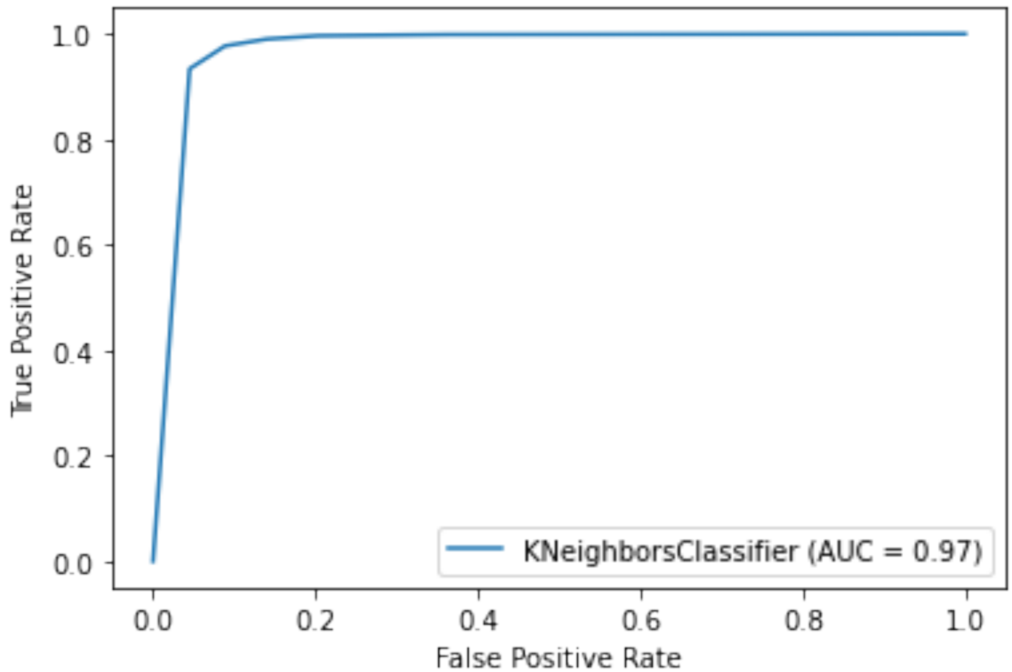


[272]:
```
from sklearn.metrics import roc_auc_score
# auc scores
auc_score = roc_auc_score(y_test, preds)
# calculate AUC
print('AUC: %.3f' % auc_score)
```

```
AUC: 0.925
```

[273]:
```
plot_roc_curve(classifier,X_test,y_test)
```

[273]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7ff2693883d0>
```



[ ]:

Application

- find the potential neighbors of a customer to see the proportion of "charged off" and "fully paid" in order to decide whether we should borrow the money to him.
- For example, randomly get 11 rows of the data.
- Since the dimensions of our data are high, we decide to reduce the dimensions

[262]:
```
random_data = df.sample(n = 51)
random_data.head()
```

[262]:

	loan_repaid	term	grade	home_ownership	verification_status	\
446838	0	0	1	0	0	
794724	1	1	2	0	1	
497271	0	0	2	3	1	
498729	1	0	1	3	0	
1363462	1	0	1	2	0	

	purpose	addr_state	loan_amnt	int_rate	installment	annual_inc	\
446838	2	41	184	28	2485	-1.455619	
794724	2	34	705	53	6976	0.369690	
497271	1	4	519	63	7731	1.626956	
498729	2	41	260	26	3586	-1.423946	
1363462	8	44	1202	51	12064	1.450691	

```
         open_acc  pub_rec  revol_bal  revol_util  total_acc  \
446838         15        0  -0.993191         156         27
794724         10        0   1.064402         705         33
497271          8        0  -0.347015         878         20
498729          8        1  -0.604990         269         15
1363462         6        1  -1.022269         205         22

         pub_rec_bankruptcies  last_fico_range_high  last_pymnt_amnt
446838                      0                     5             3225
794724                      0                    41            40573
497271                      0                     6             5868
498729                      1                    36            18891
1363462                     1                    32            48818
```

[268]:

```
random_data2 = random_data.to_numpy()
```

[269]:

```
from csv import reader
from math import sqrt


def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(1, len(row1)):
        distance += (row1[i] - row2[i])**2
    return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors

# Make a classification prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[0] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction
```

```
prediction = predict_classification(random_data2, random_data2[0], 10)
print('Expected %d, Got %d.' % (random_data2[0][0], prediction))
```

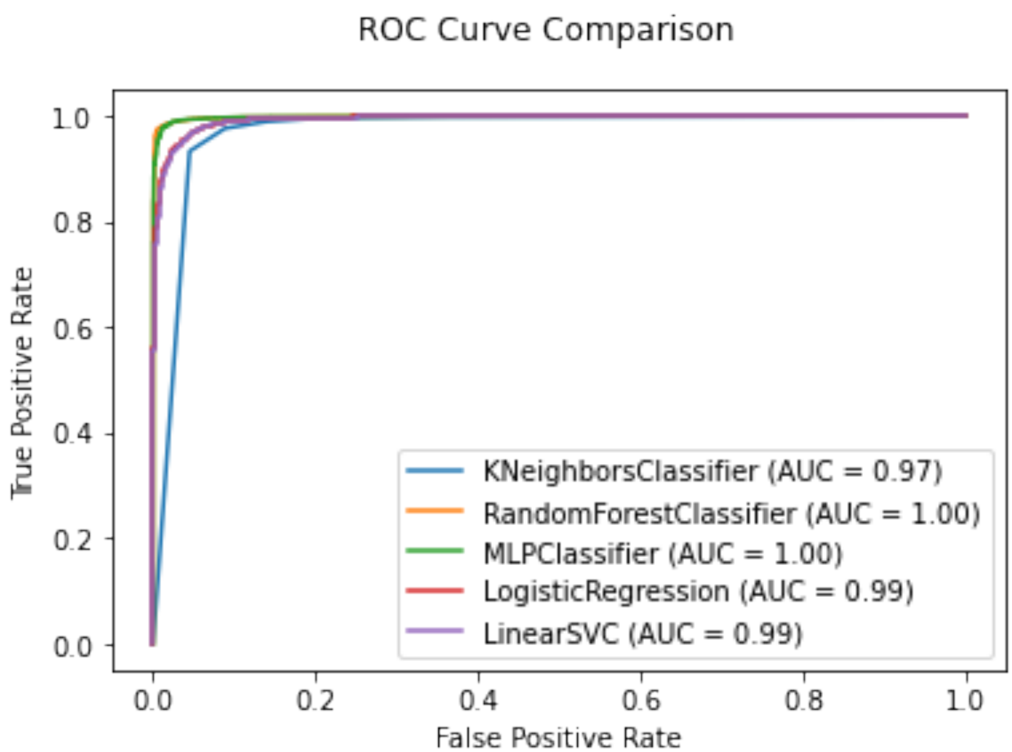
```
Expected 0, Got 0.
```

[ ]:

```
```

[276]:

```
knn_disp = plot_roc_curve(classifier, X_test, y_test)
rf_disp = plot_roc_curve(rf_model, X_test, y_test, ax=knn_disp.ax_)
nn_disp = plot_roc_curve(nn_model, X_test, y_test, ax=knn_disp.ax_)
lr_disp = plot_roc_curve(logreg, X_test, y_test, ax=knn_disp.ax_)
svm_disp = plot_roc_curve(ls2, X_test, y_test, ax=knn_disp.ax_)


knn_disp.figure_.suptitle("ROC Curve Comparison")

plt.show()
```



[ ]:

```
```