In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
train=pd.read_csv("train.csv")

test=pd.read_csv("test.csv")
```

In [57]:
```
import seaborn as sns
```

In [2]:
```
train['Months.since.last.delinquent']=train['Months.since.last.delinquent'].fillna(0)
```

In [3]:
```
train['Credit.Score']=train['Credit.Score'].fillna(0)
test['Credit.Score']=test['Credit.Score'].fillna(0)
test['Months.since.last.delinquent']=test['Months.since.last.delinquent'].fillna(0)
```

In [ ]:
```
train['Credit.Score']=train['Credit.Score'].fillna(700)
test['Credit.Score']=test['Credit.Score'].fillna(700)
```

In [4]:
```
train['Annual.Income']=train['Annual.Income'].fillna(72000)
test['Annual.Income']=test['Annual.Income'].fillna(72000)
```

In [ ]:
```
test['Annual.Income']=test['Annual.Income'].fillna(0)
train['Annual.Income']=train['Annual.Income'].fillna(0)
```

In [ ]:
```
train['Credit.Score'].fillna(train['Credit.Score'].mean(), inplace=True)
test['Credit.Score'].fillna(test['Credit.Score'].mean(), inplace=True)
train['Annual.Income'].fillna(train['Annual.Income'].mean(), inplace=True)
test['Annual.Income'].fillna(test['Annual.Income'].mean(), inplace=True)
train['Months.since.last.delinquent'].fillna(train['Months.since.last.delinquent'].mean(), inplace=True)
test['Months.since.last.delinquent'].fillna(test['Months.since.last.delinquent'].mean(), inplace=True)
```

In [ ]:
```
test['Months.since.last.delinquent']=test['Months.since.last.delinquent'].fillna(0)
```

In [5]:
```
train['Term'] = train['Term'].map({'Short Term':0,'Long Term':1}).astype(np.int)
test['Term'] = test['Term'].map({'Short Term':0,'Long Term':1}).astype(np.int)
```

In [6]:
```
train['Years.in.current.job'] = train['Years.in.current.job'].map({'9 years':9
, '':0, '6 years':6,'4 years':4, '5 years':5, 'less than  1 year':0.5, '1 yea
r':1, '2 years':2, '3 years':3, '8 years':8, '7 years':7, '10 years':10, '10+
 years':12,}).astype(np.float)
```

In [7]:
```
train['Years.in.current.job']=train['Years.in.current.job'].fillna(0)
```

In [8]:
```
test['Years.in.current.job'] = test['Years.in.current.job'].map({'9 years':9,
'':0, '6 years':6,'4 years':4, '5 years':5, 'less than  1 year':0.5, '1 year':
1, '2 years':2, '3 years':3, '8 years':8, '7 years':7, '10 years':10, '10+ yea
rs':12,}).astype(np.float)
```

In [9]:
```
test['Years.in.current.job']=test['Years.in.current.job'].fillna(0)
```

In [10]:
```
train['Home.Ownership'] = train['Home.Ownership'].map({'Home Mortgage':0,'Ren
t':1, 'Own Home': 2, 'HaveMortgage':3}).astype(np.int)
test['Home.Ownership'] = test['Home.Ownership'].map({'Home Mortgage':0,'Rent':
1, 'Own Home': 2, 'HaveMortgage':3}).astype(np.int)
```

In [11]:
```
train['Purpose'] = train['Purpose'].map({'Debt Consolidation':1,'Other':2, 'ot
her': 2, 'Home Improvements':0, 'Business Loan': 3, 'Buy a Car':4, 'Medical Bi
lls':5, 'Buy House':6, 'Take a Trip':7, 'major_purchase':8, 'small_business':9
, 'moving':10, 'Educational Expenses': 11, 'vacation': 12, 'wedding': 13, 'ren
ewable_energy':14}).astype(np.int)
test['Purpose'] = test['Purpose'].map({'Debt Consolidation':1,'Other':2, 'othe
r': 2, 'Home Improvements':0, 'Business Loan': 3, 'Buy a Car':4, 'Medical Bill
s':5, 'Buy House':6, 'Take a Trip':7, 'major_purchase':8, 'small_business':9,
'moving':10, 'Educational Expenses': 11, 'vacation': 12, 'wedding': 13, 'renew
able_energy':14}).astype(np.int)
```

In [12]:
```
train['Bankruptcies']=train['Bankruptcies'].fillna(0)
train['Tax.Liens']=train['Tax.Liens'].fillna(0)
test['Bankruptcies']=test['Bankruptcies'].fillna(0)
test['Tax.Liens']=test['Tax.Liens'].fillna(0)
```

In [ ]:
```
train.head()
```

In [13]:
```
train['Current.Loan.Amount.log'] = np.log(train['Current.Loan.Amount'])
test['Current.Loan.Amount.log'] = np.log(test['Current.Loan.Amount'])
```

In [14]:
```
train['Annual.Income.log'] = np.log(train['Annual.Income'])
test['Annual.Income.log'] = np.log(test['Annual.Income'])
```

In [15]:
```
train = train.drop('Loan.ID', axis=1)
```

In [19]:
```
X = train.drop('Loan_Status', 1)
y = train.Loan_Status
```

In [16]:
```
train = train.rename(columns={'Loan.Status': 'Loan_Status'})
```

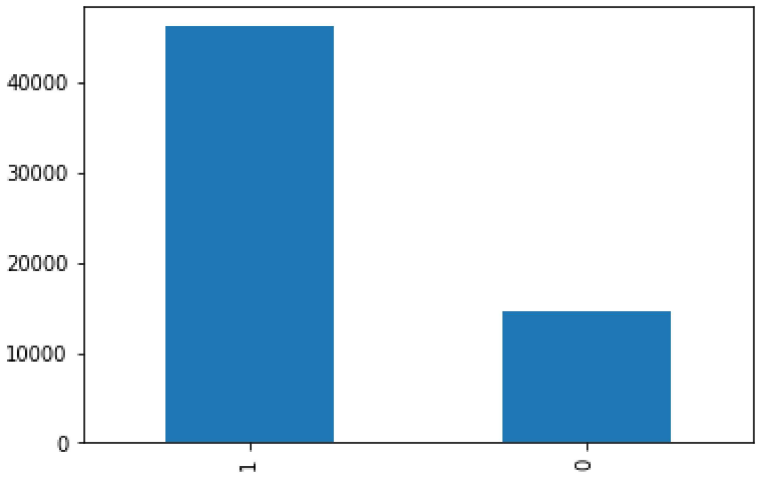
In [20]:
```
from sklearn.model_selection import train_test_split
```

In [21]:
```
test = test.drop('Loan.ID', axis=1)
```

In [22]:
```
test = test.drop('Loan.Status', axis=1)
```

In [53]:
```
train['Loan_Status'].value_counts().plot.bar()
```

Out[53]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x194acfe4ba8>
```



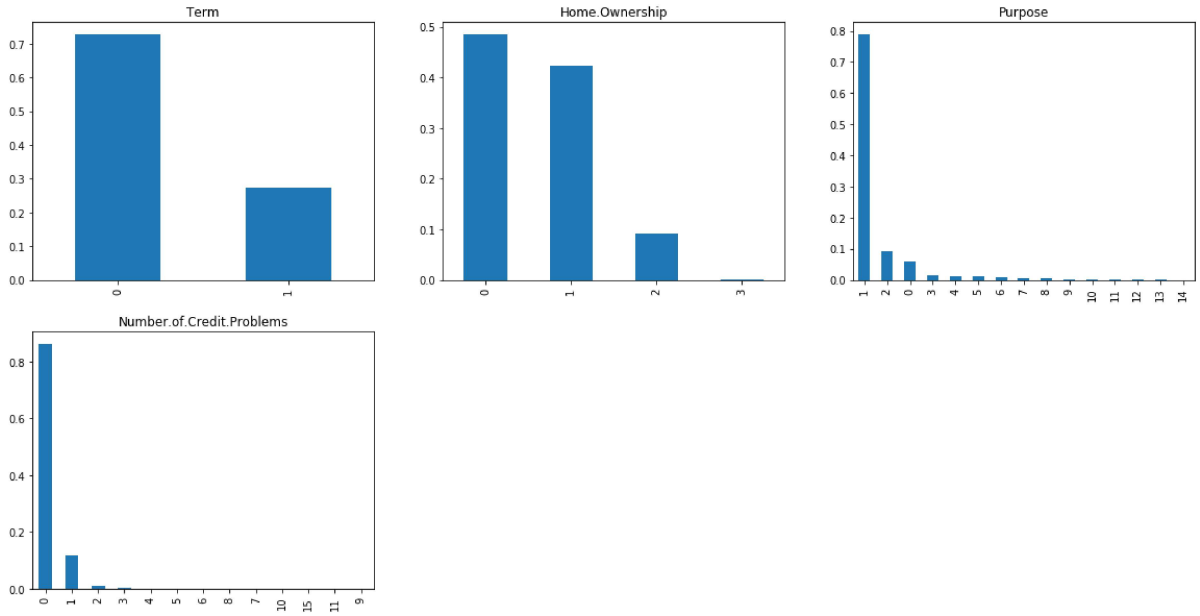
In [23]:
```
X = pd.get_dummies(X)
train1 = pd.get_dummies(train)
test1 = pd.get_dummies(test)
```

In [54]:
```
train.head()
```

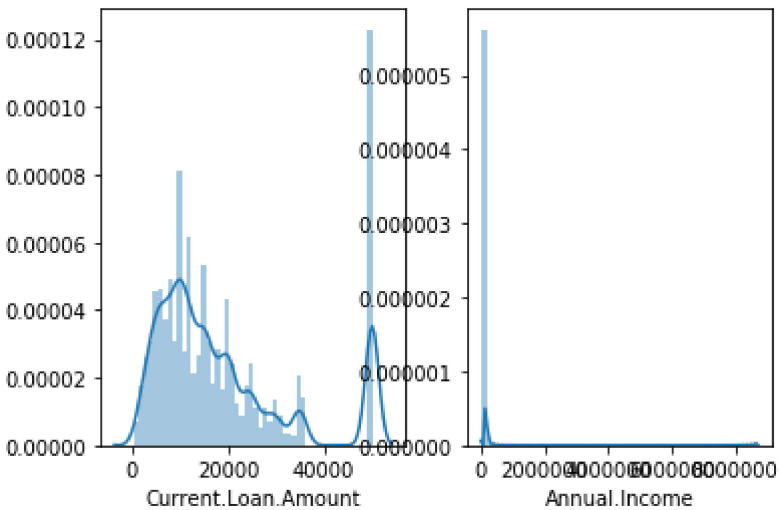
Out[54]:

	Current.Loan.Amount	Term	Credit.Score	Years.in.current.job	Home.Ownership	Annual.Income
0	17879	0	739.0	6.0	0	95357.(
1	50000	1	619.0	6.0	1	54406.(
2	50000	0	738.0	0.5	1	40480.(
3	11200	0	738.0	4.0	1	53965.(
4	3608	0	731.0	12.0	0	47709.(

In [55]:
```
plt.subplot(231)
train['Term'].value_counts(normalize=True).plot.bar(figsize=(20,10), title= 'Term')

plt.subplot(232)
train['Home.Ownership'].value_counts(normalize=True).plot.bar(title= 'Home.Ownership')

plt.subplot(233)
train['Purpose'].value_counts(normalize=True).plot.bar(title= 'Purpose')

plt.subplot(234)
train['Number.of.Credit.Problems'].value_counts(normalize=True).plot.bar(title= 'Number.of.Credit.Problems')


plt.show()
```

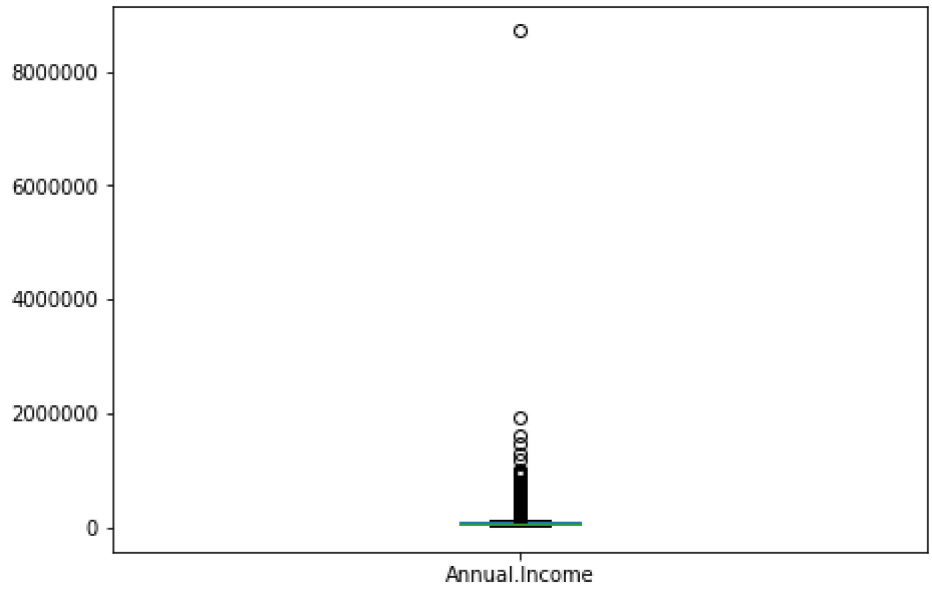


In [59]:
```
plt.subplot(121)
sns.distplot(train['Current.Loan.Amount']);
```



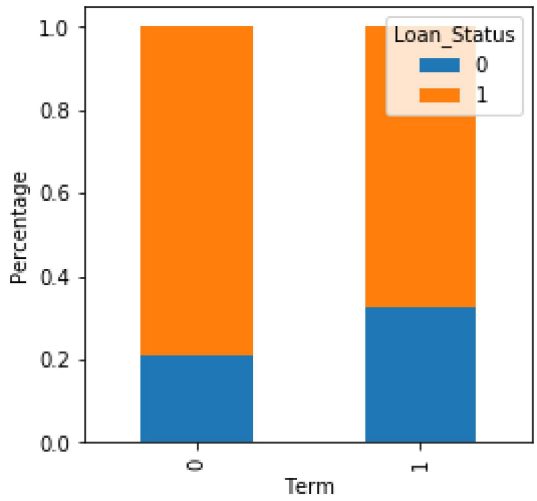
In [66]:
```
plt.subplot(122)
train['Annual.Income'].plot.box(figsize=(16,5))
```

Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x194c996ea20>



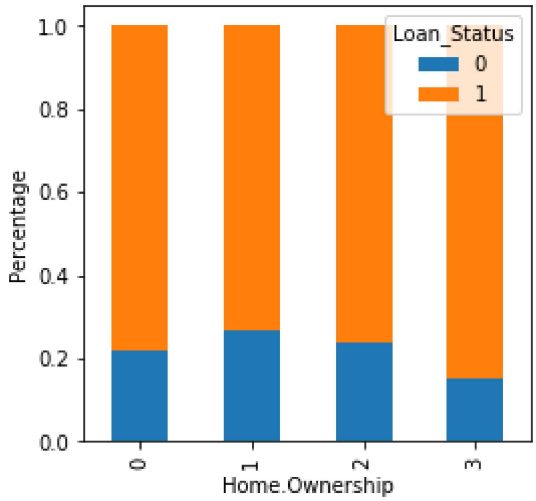
In [67]:
```
print(pd.crosstab(train['Term'],train['Loan_Status']))

Gender = pd.crosstab(train['Term'],train['Loan_Status'])
Gender.div(Gender.sum(1).astype(float), axis = 0).plot(kind="bar", stacked=True, figsize=(4,4))
plt.xlabel('Term')
p = plt.ylabel('Percentage')
```

```
Loan_Status     0      1
Term
0            9198  34986
1            5428  11192
```



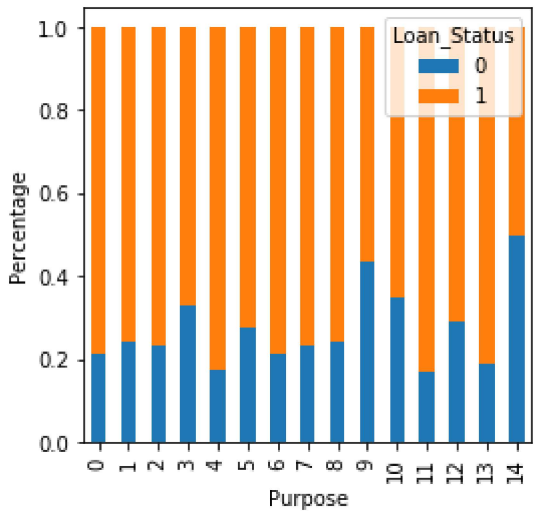
In [68]:
```
print(pd.crosstab(train['Home.Ownership'],train['Loan_Status']))

Gender = pd.crosstab(train['Home.Ownership'],train['Loan_Status'])
Gender.div(Gender.sum(1).astype(float), axis = 0).plot(kind="bar", stacked=True, figsize=(4,4))
plt.xlabel('Home.Ownership')
p = plt.ylabel('Percentage')
```

```
Loan_Status         0      1
Home.Ownership
0                6425  23022
1                6868  18869
2                1317   4199
3                  16     88
```



In [69]:
```
print(pd.crosstab(train['Purpose'],train['Loan_Status']))

Gender = pd.crosstab(train['Purpose'],train['Loan_Status'])
Gender.div(Gender.sum(1).astype(float), axis = 0).plot(kind="bar", stacked=True, figsize=(4,4))
plt.xlabel('Purpose')
p = plt.ylabel('Percentage')
```

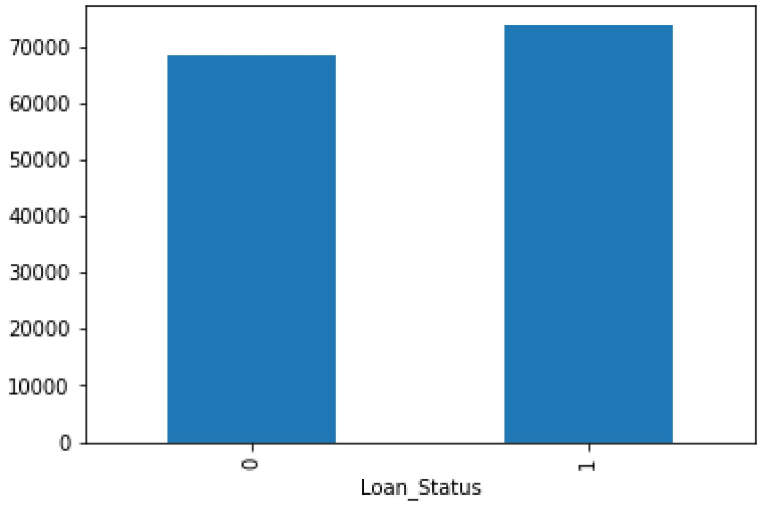
```
Loan_Status      0      1
Purpose
0              747   2788
1            11590  36298
2             1310   4318
3              298    612
4              135    630
5              191    504
6               89    331
7               68    228
8               51    162
9               74     96
10              29     54
11              12     58
12              19     46
13              12     52
14               1      1
```



In [70]:
```
print(train.groupby('Loan_Status')['Annual.Income'].mean())

train.groupby('Loan_Status')['Annual.Income'].mean().plot.bar()
```

```
Loan_Status
0    68362.405237
1    73701.790961
Name: Annual.Income, dtype: float64
```

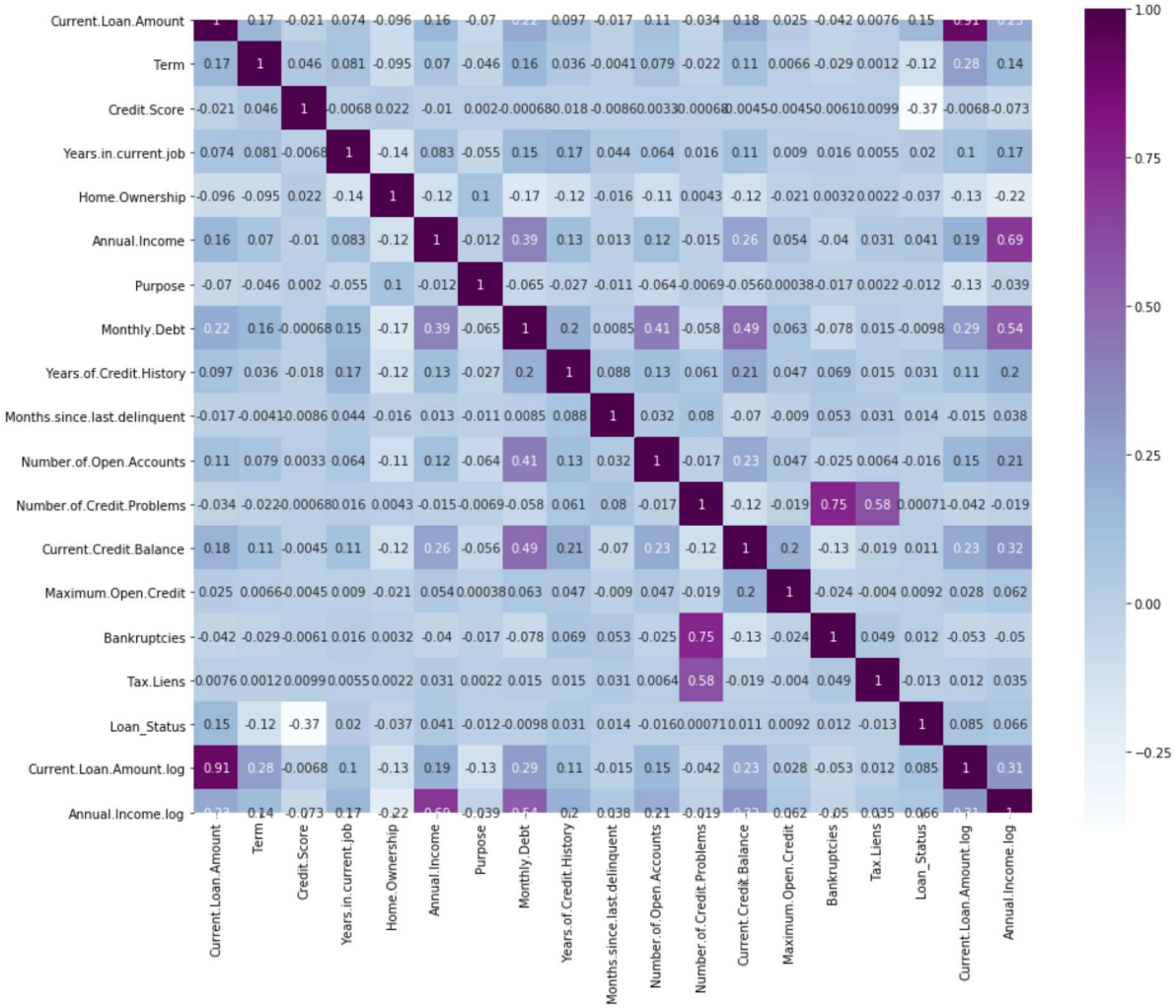
Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x194c9b692e8>



In [78]:
```
matrix = train.corr()
f, ax = plt.subplots(figsize=(15, 12))
sns.heatmap(matrix, vmax=1, square=True, cmap="BuPu", annot=True)

matrix
```

Out[78]:

	Current.Loan.Amount	Term	Credit.Score	Years.in.current.job	H
Current.Loan.Amount	1.000000	0.174637	-0.021230	0.074315	
Term	0.174637	1.000000	0.045890	0.080855	
Credit.Score	-0.021230	0.045890	1.000000	-0.006761	
Years.in.current.job	0.074315	0.080855	-0.006761	1.000000	
Home.Ownership	-0.095977	-0.095406	0.021796	-0.143432	
Annual.Income	0.159007	0.069615	-0.010236	0.083137	
Purpose	-0.069832	-0.046388	0.001980	-0.055492	
Monthly.Debt	0.224643	0.159331	-0.000680	0.149716	
Years.of.Credit.History	0.097078	0.036304	-0.017873	0.172360	
Months.since.last.delinquent	-0.016915	-0.004087	-0.008552	0.044174	
Number.of.Open.Accounts	0.107169	0.078650	0.003253	0.063580	
Number.of.Credit.Problems	-0.033819	-0.021511	-0.000683	0.016307	
Current.Credit.Balance	0.180664	0.106202	-0.004532	0.105737	
Maximum.Open.Credit	0.024673	0.006590	-0.004474	0.009013	
Bankruptcies	-0.042455	-0.028545	-0.006074	0.016016	
Tax.Liens	0.007603	0.001153	0.009947	0.005460	
Loan_Status	0.148848	-0.123478	-0.371880	0.020491	
Current.Loan.Amount.log	0.908839	0.279247	-0.006826	0.104445	
Annual.Income.log	0.232575	0.135242	-0.072722	0.166736	



In [ ]:
```
x_train, x_cv, y_train, y_cv = train_test_split(X, y, test_size=0.11, random_state=2)
```

In [25]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [79]:
```
model = LogisticRegression()
model.fit(x_train, y_train)
```

```
C:\Users\LENOVO\Anaconda3\envs\tensorflow_cpu\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[79]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [80]:
```
pred_cv = model.predict(x_cv)
```

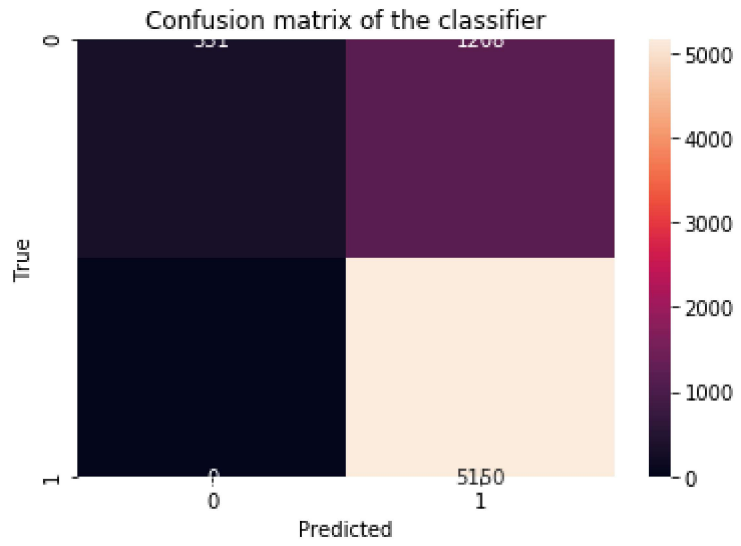
In [81]:
```
accuracy_score(y_cv, pred_cv)
```

Out[81]:
```
0.819404993272537
```

In [82]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_cv, pred_cv)
print(cm)

# f, ax = plt.subplots(figsize=(9, 6))
sns.heatmap(cm, annot=True, fmt="d")
plt.title('Confusion matrix of the classifier')
plt.xlabel('Predicted')
plt.ylabel('True')
```

```
[[ 331 1208]
 [   0 5150]]
```

Out[82]:
```
Text(33.0, 0.5, 'True')
```



In [83]:
```
from sklearn.metrics import classification_report
print(classification_report(y_cv, pred_cv))
```

```
              precision    recall  f1-score   support

           0       1.00      0.22      0.35      1539
           1       0.81      1.00      0.90      5150

    accuracy                           0.82      6689
   macro avg       0.91      0.61      0.62      6689
weighted avg       0.85      0.82      0.77      6689
```

In [84]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)
```

Out[84]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [86]:
```
y1_pred = gnb.predict(x_cv)

from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_cv, y1_pred)*100)
```

```
Gaussian Naive Bayes model accuracy(in %): 81.41725220511287
```

In [ ]:
```
pred_test = model.predict(test)
```

In [ ]:
```
submission = pd.read_csv("SubmissionSample.csv")
```

In [ ]:
```
submission['Loan_Status'] = pred_test
```

In [ ]:
```
submission.head()
```

In [ ]:
```
submission.to_csv('logisticfin.csv', index=False)
```

In [29]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [30]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [31]:
```
clf = RandomForestClassifier(max_depth=2, random_state=8)
```

In [32]:
```
clf.fit(x_train, y_train)
```

```
C:\Users\LENOVO\Anaconda3\envs\tensorflow_cpu\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[32]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=2, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=8, verbose=0,
                       warm_start=False)
```

In [33]:
```
pred_cv = clf.predict(x_cv)
```

In [34]:
```
accuracy_score(y_cv, pred_cv)
```

Out[34]:
```
0.819404993272537
```

In [35]:
```
import tensorflow as tf
```

```
C:\Users\LENOVO\Anaconda3\envs\tensorflow_cpu\lib\site-packages\tensorflow\py
thon\framework\dtypes.py:526: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be und
erstood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\LENOVO\Anaconda3\envs\tensorflow_cpu\lib\site-packages\tensorflow\py
thon\framework\dtypes.py:527: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be und
erstood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\LENOVO\Anaconda3\envs\tensorflow_cpu\lib\site-packages\tensorflow\py
thon\framework\dtypes.py:528: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be und
erstood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\LENOVO\Anaconda3\envs\tensorflow_cpu\lib\site-packages\tensorflow\py
thon\framework\dtypes.py:529: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be und
erstood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\LENOVO\Anaconda3\envs\tensorflow_cpu\lib\site-packages\tensorflow\py
thon\framework\dtypes.py:530: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be und
erstood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\LENOVO\Anaconda3\envs\tensorflow_cpu\lib\site-packages\tensorflow\py
thon\framework\dtypes.py:535: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be und
erstood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [36]:
```
import keras
```

```
Using TensorFlow backend.
```

In [37]:
```
from keras.models import Sequential
from keras.layers import Dense
```

In [38]:
```
model = Sequential()
model.add(Dense(12, input_dim=18, activation='softmax'))
model.add(Dense(8, activation='softmax'))
model.add(Dense(1, activation='tanh'))
```

```
WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\envs\tensorflow_cpu\lib\sit
e-packages\tensorflow\python\ops\resource_variable_ops.py:435: colocate_with
(from tensorflow.python.framework.ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Colocations handled automatically by placer.
```

In [39]:
```
model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [40]:
```
model.fit(x_train, y_train, epochs=10, batch_size=10)
```

```
WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\envs\tensorflow_cpu\lib\sit
e-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.
python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/10
54115/54115 [==============================] - 4s 72us/step - loss: 12.2201 -
accuracy: 0.2418
Epoch 2/10
54115/54115 [==============================] - 3s 60us/step - loss: 12.2201 -
accuracy: 0.2418
Epoch 3/10
54115/54115 [==============================] - 5s 85us/step - loss: 12.2201 -
accuracy: 0.2418
Epoch 4/10
54115/54115 [==============================] - 5s 86us/step - loss: 12.2201 -
accuracy: 0.2418
Epoch 5/10
54115/54115 [==============================] - 4s 71us/step - loss: 12.2201 -
accuracy: 0.2418
Epoch 6/10
54115/54115 [==============================] - 4s 72us/step - loss: 12.2201 -
accuracy: 0.2418
Epoch 7/10
54115/54115 [==============================] - 5s 85us/step - loss: 12.2201 -
accuracy: 0.2418
Epoch 8/10
54115/54115 [==============================] - 4s 78us/step - loss: 12.2201 -
accuracy: 0.2418
Epoch 9/10
54115/54115 [==============================] - 4s 78us/step - loss: 12.2201 -
accuracy: 0.2418
Epoch 10/10
54115/54115 [==============================] - 4s 79us/step - loss: 12.2201 -
accuracy: 0.2418
```

Out[40]:
```
<keras.callbacks.callbacks.History at 0x194a10037b8>
```

In [41]:
```
pred_cv = model.predict(x_cv)
```

In [43]:
```
pred_cv.head()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-43-cffc6c05d047> in <module>
----> 1 pred_cv.head()

AttributeError: 'numpy.ndarray' object has no attribute 'head'
```

In [42]:
```
accuracy_score(y_cv, pred_cv)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-42-8366a2fb48dd> in <module>
----> 1 accuracy_score(y_cv, pred_cv)

~\Anaconda3\envs\tensorflow_cpu\lib\site-packages\sklearn\metrics\classification.py in accuracy_score(y_true, y_pred, normalize, sample_weight)
    174
    175     # Compute accuracy for each possible representation
--> 176     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    177     check_consistent_length(y_true, y_pred, sample_weight)
    178     if y_type.startswith('multilabel'):

~\Anaconda3\envs\tensorflow_cpu\lib\site-packages\sklearn\metrics\classification.py in _check_targets(y_true, y_pred)
     79     if len(y_type) > 1:
     80         raise ValueError("Classification metrics can't handle a mix of {0} "
---> 81                          "and {1} targets".format(type_true, type_pred))
     82
     83     # We can't have more than one value on y_type => The set is no more needed

ValueError: Classification metrics can't handle a mix of binary and continuous targets
```

In [44]:
```
clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
```

In [45]:
```
clf.fit(x_train, y_train)
```

Out[45]:
```
Pipeline(memory=None,
         steps=[('standardscaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('svc',
                 SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
                     decision_function_shape='ovr', degree=3, gamma='auto',
                     kernel='rbf', max_iter=-1, probability=False,
                     random_state=None, shrinking=True, tol=0.001,
                     verbose=False))],
         verbose=False)
```

In [48]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [49]:
```
clf=RandomForestClassifier(n_estimators=10)
```

In [50]:
```
clf.fit(x_train,y_train)
```

Out[50]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=No
ne,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [51]:
```
pred_cv = clf.predict(x_cv)
```

In [52]:
```
accuracy_score(y_cv, pred_cv)
```

Out[52]:
```
0.7887576618328599
```

In [ ]:
```
#Try adaboost and xgboost
```

In [ ]:
```
pred_test = clf.predict(test)
```

In [ ]:
```
submission.to_csv('ada.csv', index=False)
```

In [ ]:
```

```