Lending Club Loan Data Analysis

[2]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

[3]:
```
#Import Libraries
#Import the usual libraries for pandas and plotting. You can import sklearn␣
↪later on.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[4]:
```
loans = pd.read_csv('loan_data.csv')
```

[5]:
```
#Check out the info(), head(), and describe() methods on loans.

loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   credit.policy      9578 non-null   int64
 1   purpose            9578 non-null   object
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64
 11  delinq.2yrs        9578 non-null   int64
 12  pub.rec            9578 non-null   int64
 13  not.fully.paid     9578 non-null   int64
```

```
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

[6]:
```
loans.describe()
```

[6]:
```
       credit.policy     int.rate  installment  log.annual.inc          dti  \
count    9578.000000  9578.000000  9578.000000     9578.000000  9578.000000
mean        0.804970     0.122640   319.089413       10.932117    12.606679
std         0.396245     0.026847   207.071301        0.614813     6.883970
min         0.000000     0.060000    15.670000        7.547502     0.000000
25%         1.000000     0.103900   163.770000       10.558414     7.212500
50%         1.000000     0.122100   268.950000       10.928884    12.665000
75%         1.000000     0.140700   432.762500       11.291293    17.950000
max         1.000000     0.216400   940.140000       14.528354    29.960000

              fico  days.with.cr.line     revol.bal   revol.util  \
count  9578.000000        9578.000000  9.578000e+03  9578.000000
mean    710.846314        4560.767197  1.691396e+04    46.799236
std      37.970537        2496.930377  3.375619e+04    29.014417
min     612.000000         178.958333  0.000000e+00     0.000000
25%     682.000000        2820.000000  3.187000e+03    22.600000
50%     707.000000        4139.958333  8.596000e+03    46.300000
75%     737.000000        5730.000000  1.824950e+04    70.900000
max     827.000000       17639.958330  1.207359e+06   119.000000

       inq.last.6mths  delinq.2yrs      pub.rec  not.fully.paid
count     9578.000000  9578.000000  9578.000000     9578.000000
mean         1.577469     0.163708     0.062122        0.160054
std          2.200245     0.546215     0.262126        0.366676
min          0.000000     0.000000     0.000000        0.000000
25%          0.000000     0.000000     0.000000        0.000000
50%          1.000000     0.000000     0.000000        0.000000
75%          2.000000     0.000000     0.000000        0.000000
max         33.000000    13.000000     5.000000        1.000000
```

[7]:
```
loans.head()
```

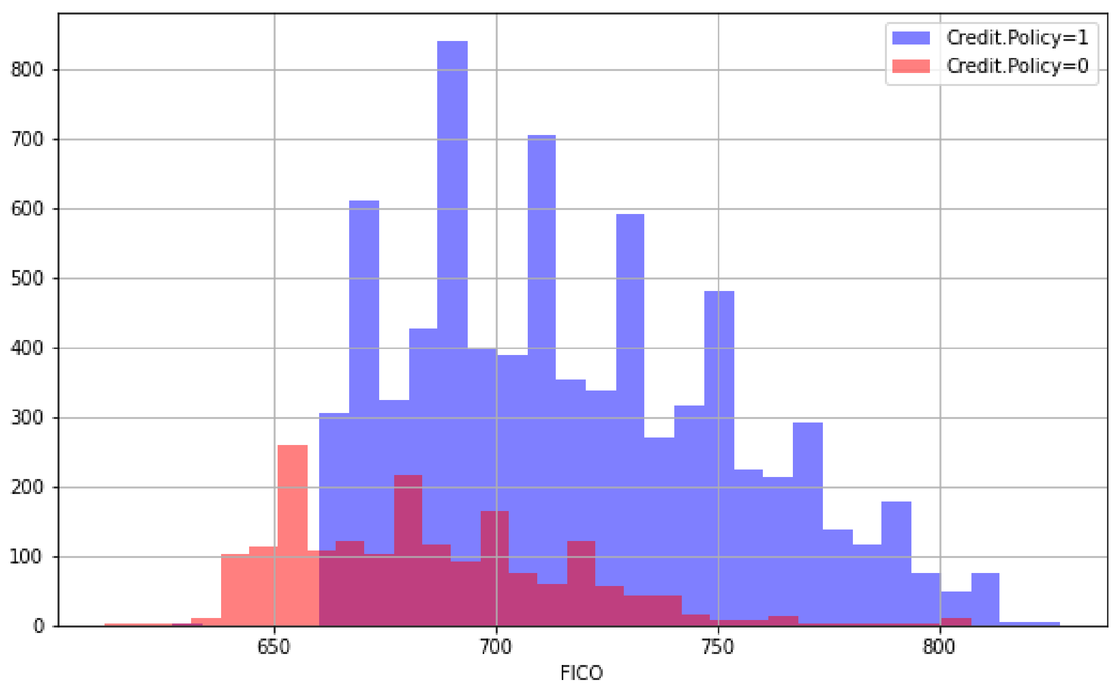
[7]:
```
   credit.policy             purpose  int.rate  installment  log.annual.inc  \
0              1  debt_consolidation    0.1189       829.10       11.350407
1              1         credit_card    0.1071       228.22       11.082143
2              1  debt_consolidation    0.1357       366.86       10.373491
3              1  debt_consolidation    0.1008       162.34       11.350407
4              1         credit_card    0.1426       102.92       11.299732

     dti  fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
0  19.48   737        5639.958333      28854        52.1               0
1  14.29   707        2760.000000      33623        76.7               0
```

```
2  11.63  682   4710.000000   3511   25.6   1
3   8.10  712   2699.958333  33667   73.2   1
4  14.97  667   4066.000000   4740   39.5   0

   delinq.2yrs  pub.rec  not.fully.paid
0            0        0               0
1            0        0               0
2            0        0               0
3            0        0               0
4            1        0               0
```

[ ]:
```
#Exploratory Data Analysis¶
#Let's do some data visualization! We'll use seaborn and pandas built-in␣
↪plotting capabilities, but feel free to use whatever library you want.

#Creating a histogram of two FICO distributions on top of each other, one for␣
↪each credit.policy outcome.
```

[8]:
```
plt.figure(figsize=(10,6))
loans[loans['credit.policy']==1]['fico'].hist(alpha=0.5,color='blue',
                                              bins=30,label='Credit.Policy=1')
loans[loans['credit.policy']==0]['fico'].hist(alpha=0.5,color='red',
                                              bins=30,label='Credit.Policy=0')
plt.legend()
plt.xlabel('FICO')
```

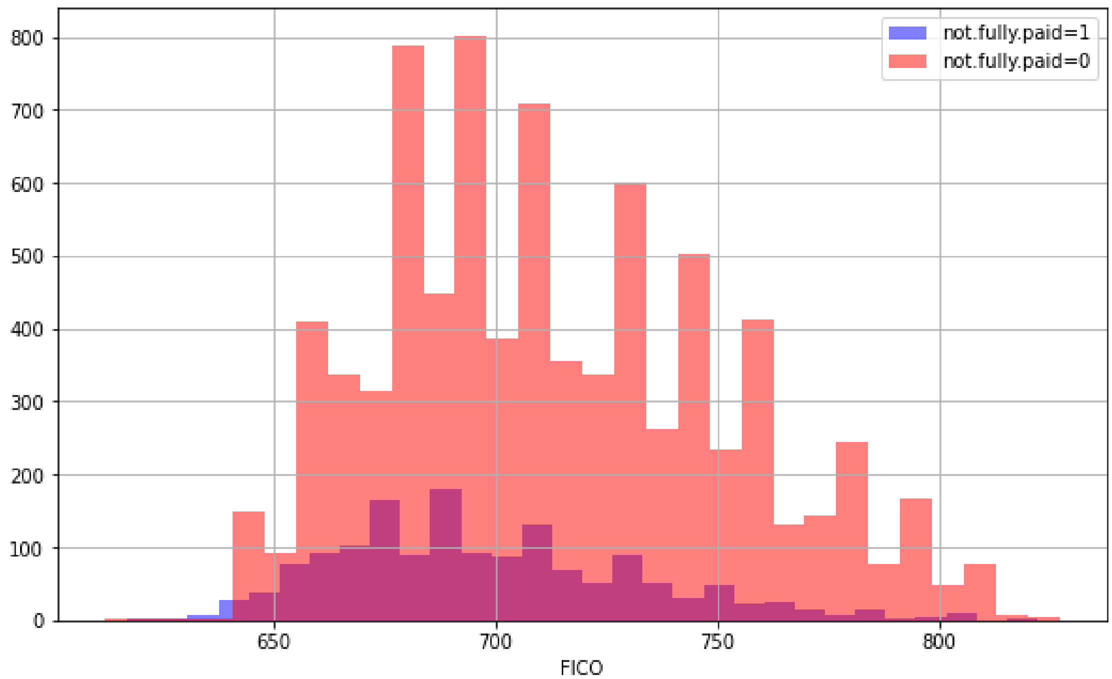
[8]:
```
Text(0.5, 0, 'FICO')
```



```
[ ]: #Now let us createe a similar figure, except this time select by the not.fully.
     ↪paid column.
```

```
[9]: plt.figure(figsize=(10,6))
     loans[loans['not.fully.paid']==1]['fico'].hist(alpha=0.5,color='blue',
                                               bins=30,label='not.fully.paid=1')
     loans[loans['not.fully.paid']==0]['fico'].hist(alpha=0.5,color='red',
                                               bins=30,label='not.fully.paid=0')
     plt.legend()
     plt.xlabel('FICO')
```

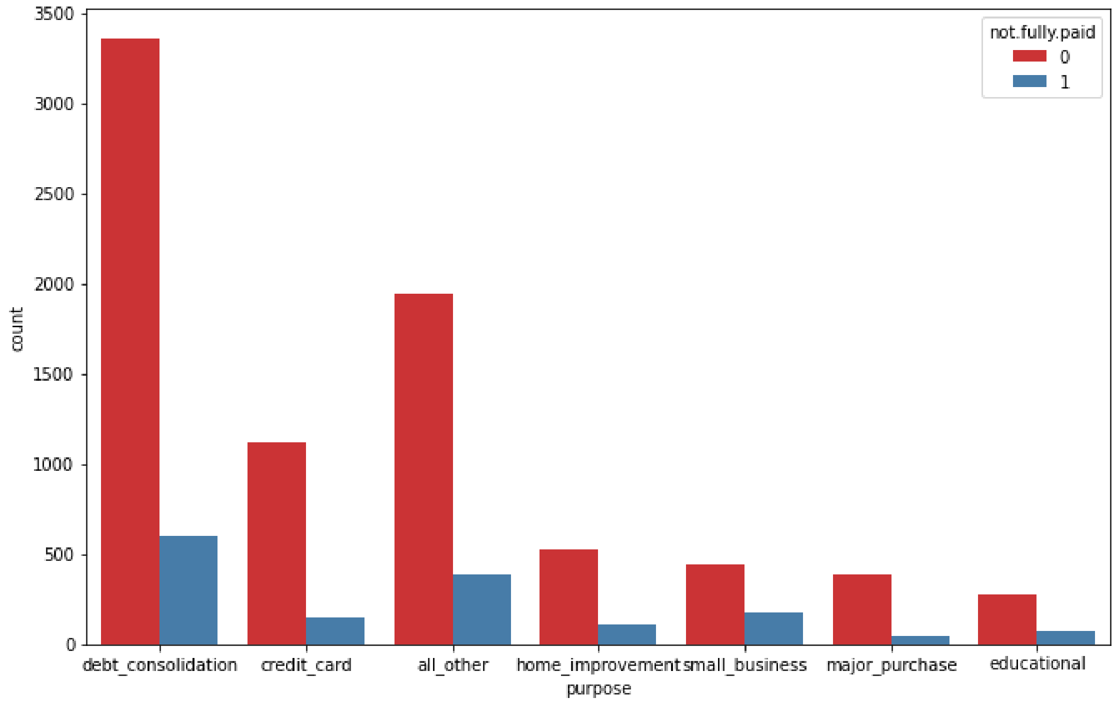
```
[9]: Text(0.5, 0, 'FICO')
```




```
[ ]: #Create a countplot using seaborn showing the counts of loans by purpose, with␣
     ↪the color hue defined by not.fully.paid.
```

```
[10]: plt.figure(figsize=(11,7))
      sns.countplot(x='purpose',hue='not.fully.paid',data=loans,palette='Set1')
```

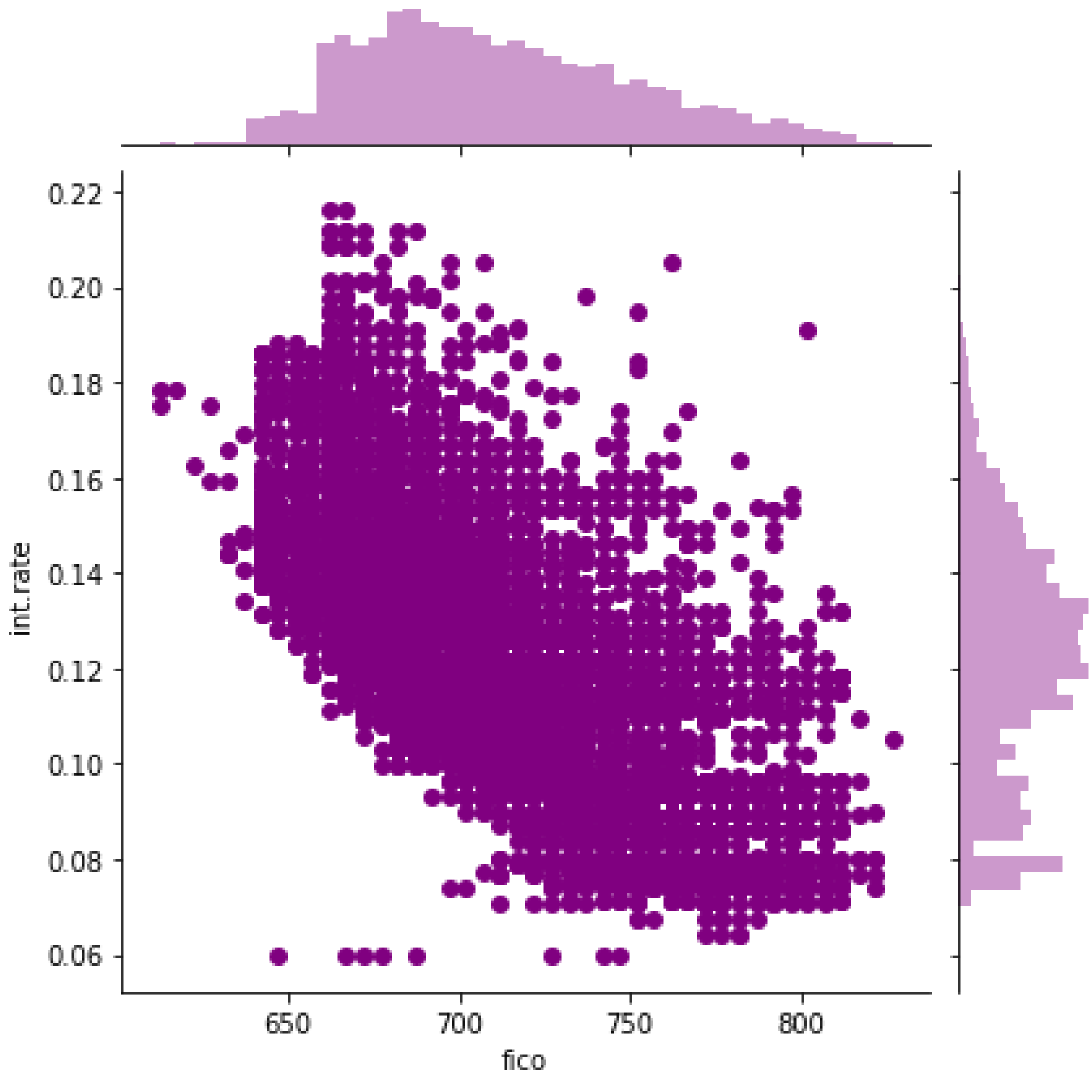
```
[10]: <AxesSubplot:xlabel='purpose', ylabel='count'>
```



[ ]:
```
#Let's see the trend between FICO score and interest rate.
```

[11]:
```
sns.jointplot(x='fico',y='int.rate',data=loans,color='purple')
```

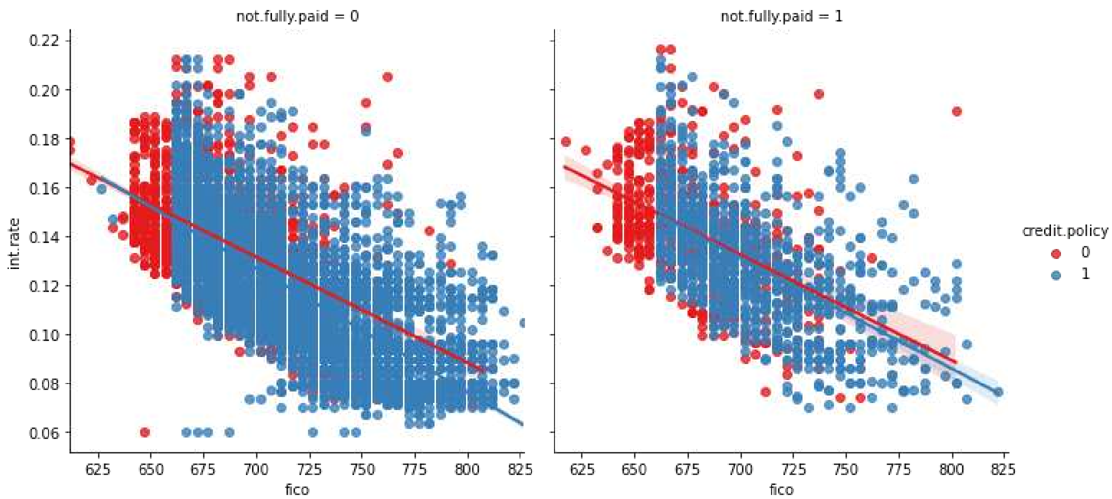
[11]: <seaborn.axisgrid.JointGrid at 0x7f9c946f0890>




```
[ ]: #Let us create the following lmplots to see if the trend differed between not.
     ↪fully.paid and credit.policy.
```

```
[12]: plt.figure(figsize=(11,7))
      sns.lmplot(y='int.rate',x='fico',data=loans,hue='credit.policy',
                 col='not.fully.paid',palette='Set1')
```

```
[12]: <seaborn.axisgrid.FacetGrid at 0x7f9c94342b90>

      <Figure size 792x504 with 0 Axes>
```



[ ]:
```
#Setting up the Data
Let's get ready to set up our data for our Random Forest Classification Model!

Categorical Features
Notice that the purpose column as categorical

That means we need to transform them using dummy variables so sklearn will be␣
↪able to understand them.

#Create a list of 1 element containing the string 'purpose'. Call this list␣
↪cat_feats.
```

[13]:
```
cat_feats = ['purpose']
```

[ ]:
```
#Now use pd.get_dummies(loans,columns=cat_feats,drop_first=True) to create a␣
↪fixed larger dataframe that has new feature columns with dummy variables.␣
↪Set this dataframe as final_data.
```

[14]:
```
final_data = pd.get_dummies(loans,columns=cat_feats,drop_first=True)
```

[ ]:
```
#Notice we have drop the first column to avoid collinearity in our data.␣
↪MultiCollinearity can have a bigger impact in our data.
```

[15]:
```
final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 19 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
```

```
 0   credit.policy              9578 non-null   int64
 1   int.rate                   9578 non-null   float64
 2   installment                9578 non-null   float64
 3   log.annual.inc             9578 non-null   float64
 4   dti                        9578 non-null   float64
 5   fico                       9578 non-null   int64
 6   days.with.cr.line          9578 non-null   float64
 7   revol.bal                  9578 non-null   int64
 8   revol.util                 9578 non-null   float64
 9   inq.last.6mths             9578 non-null   int64
 10  delinq.2yrs                9578 non-null   int64
 11  pub.rec                    9578 non-null   int64
 12  not.fully.paid             9578 non-null   int64
 13  purpose_credit_card        9578 non-null   uint8
 14  purpose_debt_consolidation 9578 non-null   uint8
 15  purpose_educational        9578 non-null   uint8
 16  purpose_home_improvement   9578 non-null   uint8
 17  purpose_major_purchase     9578 non-null   uint8
 18  purpose_small_business     9578 non-null   uint8
dtypes: float64(6), int64(7), uint8(6)
memory usage: 1.0 MB
```

[ ]:
```
#Train Test Split¶
Now its time to split our data into a training set and a testing set!
```

[16]:
```
from sklearn.model_selection import train_test_split
```

[17]:
```
X = final_data.drop('not.fully.paid',axis=1)
y = final_data['not.fully.paid']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
↪random_state=101)
```

[ ]:
```
#Training a Decision Tree Model
Let's start by training a single decision tree first!
```

[18]:
```
from sklearn.tree import DecisionTreeClassifier
```

[19]:
```
dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train)
```

[19]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, 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, presort='deprecated',
                       random_state=None, splitter='best')
```

[ ]:
```python
#Predictions and Evaluation of Decision Tree
```

[20]:
```python
predictions = dtree.predict(X_test)
```

[21]:
```python
from sklearn.metrics import classification_report,confusion_matrix
```

[22]:
```python
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      0.82      0.84      2431
           1       0.19      0.23      0.20       443

    accuracy                           0.73      2874
   macro avg       0.52      0.52      0.52      2874
weighted avg       0.75      0.73      0.74      2874
```

[23]:
```python
print(confusion_matrix(y_test,predictions))
```

```
[[1998  433]
 [ 343  100]]
```

[ ]:
```python
#Training the Random Forest model¶
Now its time to train our model!
```

[24]:
```python
from sklearn.ensemble import RandomForestClassifier
```

[25]:
```python
rfc = RandomForestClassifier(n_estimators=600)
```

[26]:
```python
rfc.fit(X_train,y_train)
```

[26]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=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=600,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[ ]:
```python
#Predictions and Evaluation¶
#Let's predict off the y_test values and evaluate our model.
```

[27]:
```python
predictions = rfc.predict(X_test)
```

[28]:
```python
from sklearn.metrics import classification_report,confusion_matrix
```

[29]:
```python
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2431
           1       0.53      0.02      0.03       443

    accuracy                           0.85      2874
   macro avg       0.69      0.51      0.48      2874
weighted avg       0.80      0.85      0.78      2874
```

[30]:
```python
print(confusion_matrix(y_test,predictions))
```

```
[[2424    7]
 [ 435    8]]
```

[ ]:
```python
#the recall for each class for the models.
Neither did very well, Looks like we need more feature engineering to predict
 ↪our model more accurately
```