LendingClub Dataset

This project contains supervised learning's classification modeling on the famous LendingClub The Dataset consists of the loan information from the years 2007 to 2018. I will be working on the years 2012 to 2014 of this Dataset (https://www.kaggle.com/wordsforthewise/lending-club).

About LendingClub (https://www.lendingclub.com/company/about-us): Since 2007 LendingClub has been bringing borrowers and investors together, transforming the way people access credit. Over the last 10 years, they have helped millions of people take control of their debt, grow their small businesses, and invest for the future. In this project, I will use the data to extract useful insights that will help the future lending club members in **predicting the loan status** of a user. Also, there will be useful insights provided in the notebook that will **help the money lenders in understanding the customer base and people analytics better.**

- **Step 1 – collect data** Download the data files from the kaggle LendingClub website and subset the data in the ac- cepted_2007_2018q4.csv file for the years 2012-2014.

- **Step 2 – exploring and preparing the data** Summarize the important features of the data. Summarize some of the numeric and some of the categorical features, there maybe too many to summarize. If you have time and are interested, see if you can get the trelliscope package to work for visualization. Create Training and Testing datasets. Use a 75-25 split. (What was used for the accuracies on the github?)

- **Step 3 – training a model on the data** Try out the applicable classification models such as Random Forest, Logistic Regressiong, KNN, Boosting, Decising Tree, Support Vector.

- **Step 4 – evaluating model performance** Compute the accuracy and, if appropriate, the area under the ROC curve (AUC) to rank the classification accuracy of each model.

- **Step 5 – improving model performance** Try to tune the parameters in each model to achieve best performance. In the Conclusion section clearly state what you believe the best ML learning model is for classifying Loan Status, variable is loan_status.

I would like to thank Professor [Eric A Suess ('http://cox.csueastbay.edu/~esuess/')](http://cox.csueastbay.edu/~esuess/) for guiding and helping me complete this project.

In [1]:

```
import re
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# Standard plotly imports
import plotly as py
import plotly.graph_objs as go
from plotly.offline import iplot, init_notebook_mode
# Using plotly + cufflinks in offline mode
import cufflinks
cufflinks.go_offline(connected=True)
init_notebook_mode(connected=True)

import hvplot.pandas
import holoviews as hv

%matplotlib inline
```

In [2]:

```
df = pd.read_csv('accepted_2007_to_2018Q4.csv', low_memory=False
)
```

In [3]:

```
df
```

Out[3]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt
0	68407277	NaN	3600.0	3600.0	36
1	68355089	NaN	24700.0	24700.0	247
2	68341763	NaN	20000.0	20000.0	200
3	66310712	NaN	35000.0	35000.0	350
4	68476807	NaN	10400.0	10400.0	104
...	...	...	...	...	
2260696	88985880	NaN	40000.0	40000.0	400
2260697	88224441	NaN	24000.0	24000.0	240
2260698	88215728	NaN	14000.0	14000.0	140
2260699	Total amount funded in policy code 1: 1465324575	NaN	NaN	NaN	
2260700	Total amount funded in policy code 2: 521953170	NaN	NaN	NaN	

2260701 rows × 151 columns

EDA

This project will include analysis performed for the years 2012 to 2014. We will subset our data to those years.

Issue Date

In [4]:

```
df.issue_d.unique()
df.issue_d.isna().sum()
```

Out[4]:

```
33
```

In [5]:

```
df['issue_d'] =  pd.to_datetime(df['issue_d'])
df["issue_d"].fillna(df["issue_d"].mean(), inplace = True)
```

In [6]:

```
new_df = df[(df.issue_d >= '2012-01-01 00:00:00') & (df.issue_d
< '2015-01-01 00:00:00')]
new_df = new_df.reset_index(drop=True)
```

In [7]:

```
new_df.info()
new_df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 423810 entries, 0 to 423809
Columns: 151 entries, id to settlement_term
dtypes: datetime64[ns](1), float64(113), object(37)
memory usage: 488.2+ MB
```

Out[7]:

	member_id	loan_amnt	funded_amnt	funded_amnt_inv	
count	0.0	423810.000000	423810.000000	423810.000000	4238
mean	NaN	14641.033659	14639.914938	14631.905082	
std	NaN	8300.162717	8299.264937	8295.368633	
min	NaN	1000.000000	1000.000000	950.000000	
25%	NaN	8000.000000	8000.000000	8000.000000	
50%	NaN	12800.000000	12800.000000	12800.000000	
75%	NaN	20000.000000	20000.000000	20000.000000	
max	NaN	35000.000000	35000.000000	35000.000000	

8 rows × 113 columns

In [8]:

```
new_df.head()
```

Out[8]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	t
0	36805548	NaN	10400.0	10400.0	10400.0	mo
1	38098114	NaN	15000.0	15000.0	15000.0	mo
2	37822187	NaN	9600.0	9600.0	9600.0	mo
3	37662224	NaN	7650.0	7650.0	7650.0	mo
4	37612354	NaN	12800.0	12800.0	12800.0	mo

5 rows × 151 columns

Let's get the missing values of the data that will help us understand the features better.

In [9]:

```
total_missing = new_df.isnull().sum().sort_values(ascending=False)
percent_missing = (new_df.isnull().sum()/new_df.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total_missing, percent_missing], axis=1, keys=['Total', 'Percent'])
missing_data.head(90)
```

Out[9]:

	Total	Percent
inq_fi	423810	1.000000
inq_last_12m	423810	1.000000
sec_app_chargeoff_within_12_mths	423810	1.000000
sec_app_num_rev_accts	423810	1.000000
sec_app_open_act_il	423810	1.000000
...	...	...
num_bc_sats	16055	0.037883
bc_util	11723	0.027661
percent_bc_gt_75	11585	0.027335
bc_open_to_buy	11470	0.027064
mths_since_recent_bc	11074	0.026130

90 rows × 2 columns

The columns of the dataset containing null values will be dropped to simplify the machine learning purpose. Note: Most columns had over 70% Null values. Using these columns for ML purpose would provide a biased and unauthentic output.

In [10]:

```
new_df = new_df.loc[:, new_df.isnull().mean() <= 0.0]
```

In [11]:

```
total_missing1 = new_df.isnull().sum().sort_values(ascending=False)
percent_missing1 = (new_df.isnull().sum()/new_df.isnull().count()).sort_values(ascending=False)
missing_data1 = pd.concat([total_missing1, percent_missing1], axis=1, keys=['Total', 'Percent'])
print('Columns without null values:')
missing_data1.head(36)
```

Columns without null values:

Out[11]:

	Total	Percent
debt_settlement_flag	0	0.0
open_acc	0	0.0
fico_range_high	0	0.0
fico_range_low	0	0.0
earliest_cr_line	0	0.0
delinq_2yrs	0	0.0
dti	0	0.0
addr_state	0	0.0
zip_code	0	0.0
purpose	0	0.0
url	0	0.0
pymnt_plan	0	0.0
loan_status	0	0.0
issue_d	0	0.0
verification_status	0	0.0
annual_inc	0	0.0
home_ownership	0	0.0

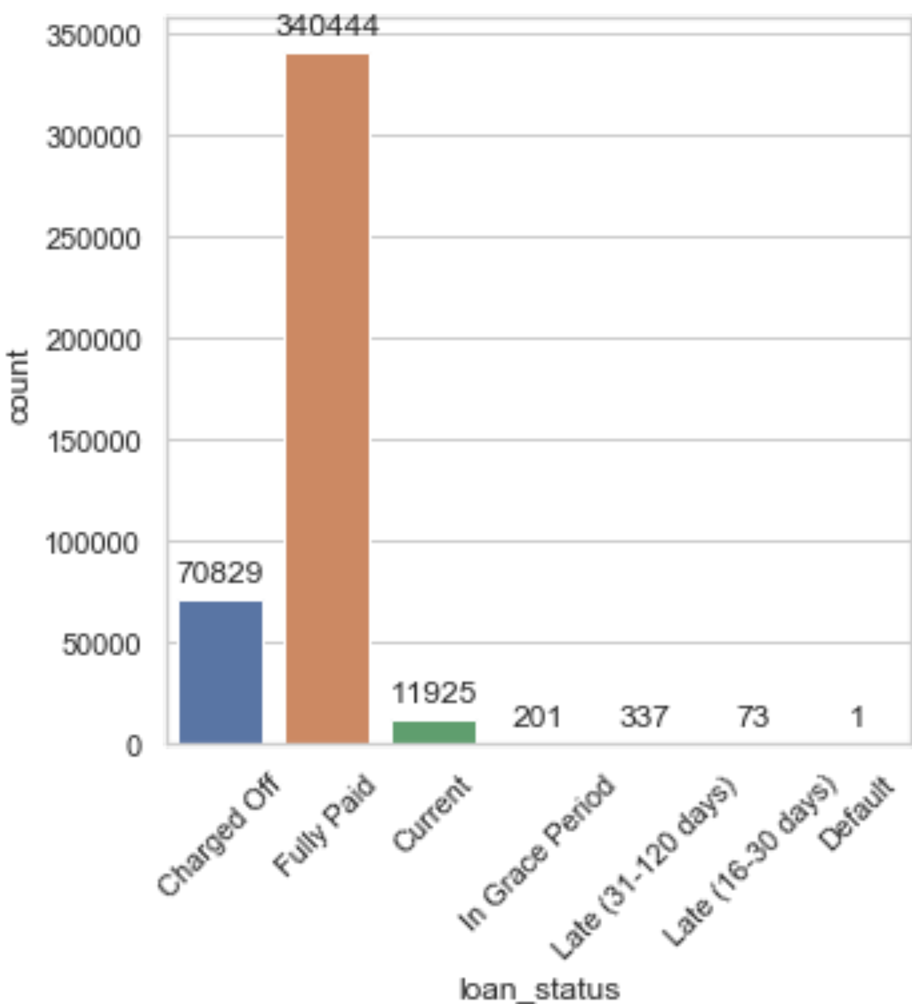
sub_grade	0	0.0
grade	0	0.0
installment	0	0.0
int_rate	0	0.0
term	0	0.0
funded_amnt_inv	0	0.0
funded_amnt	0	0.0
loan_amnt	0	0.0
inq_last_6mths	0	0.0
pub_rec	0	0.0
disbursement_method	0	0.0
revol_bal	0	0.0
hardship_flag	0	0.0
tax_liens	0	0.0
pub_rec_bankruptcies	0	0.0
delinq_amnt	0	0.0
chargeoff_within_12_mths	0	0.0
acc_now_delinq	0	0.0
application_type	0	0.0

Visualization

Loan Status (The target variable)

In [12]:

```
plt.figure(figsize = (5,5))
new_df['loan_status'].value_counts()

import seaborn as sns

sns.set(style="whitegrid")
splot = sns.countplot(x="loan_status",data=new_df[['loan_status'
]])
plt.xticks(rotation = 45)
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.0f'),
                   (p.get_x() + p.get_width() / 2.,
                    p.get_height()),
                   ha = 'center', va = 'center', xytext = (0, 10
), textcoords = 'offset points')
```



Our aim is to predict if the user is charged off or current

Grades

In [13]:

```
from itertools import cycle
plt.style.use('bmh')
color_cycle = cycle(plt.rcParams['axes.prop_cycle'].by_key()['co
lor'])
```

In [14]:

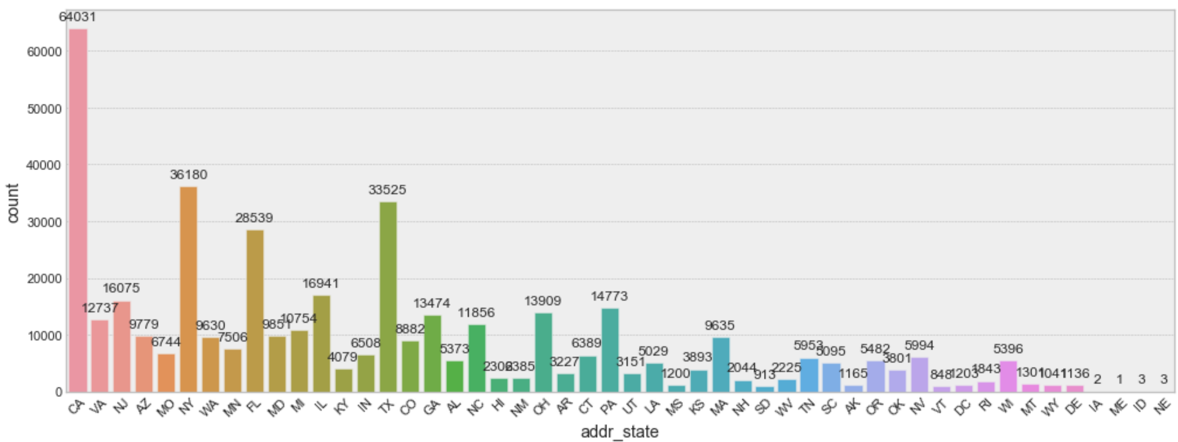
```
new_df.groupby('grade').count()['id'] \
    .sort_values() \
    .plot(kind='barh', figsize=(15, 5), title='Count of members
by Grades')
plt.show()
```

Let's find the annual income and the loan status based on the different cities.

Annual Income, Loan Status, Different States

In [15]:

```
plt.figure(figsize = (17,6))
new_df['addr_state'].value_counts()

splot = sns.countplot(x="addr_state",data=new_df[['addr_state']]
)
plt.xticks(rotation = 45)
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.0f'),
                   (p.get_x() + p.get_width() / 2.,
                    p.get_height()),
                   ha = 'center', va = 'center', xytext = (0, 10
), textcoords = 'offset points')
```

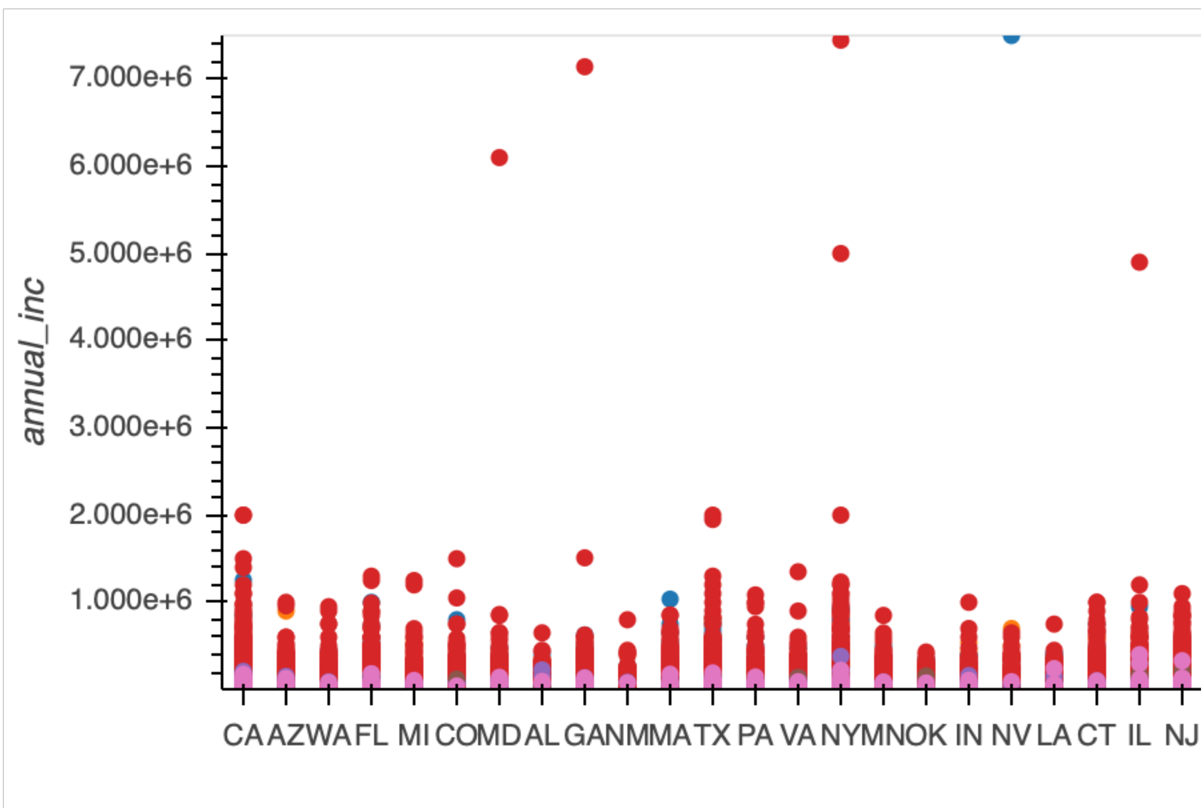


California had the highest number of money lending and borrowing business during 2012 to 2014. Could this be related to the highest druaght experience by CA in 2012? or maybe it was for the all famous Apple 4S release?

In [16]:

```
hv.extension('bokeh')
img = new_df.hvplot(kind='points', x='addr_state', y='annual_inc
', by='loan_status')
img.options(frame_width=800)
```

Out[16]:



7.000e+6
6.000e+6
5.000e+6
4.000e+6
3.000e+6
2.000e+6
1.000e+6
annual_inc
CA AZ WA FL MI CO MD AL GA NM MA TX PA VA NY MN OK IN NV LA CT IL NJ

It is clear and evident that the population of states with higher income had their loans paid off.

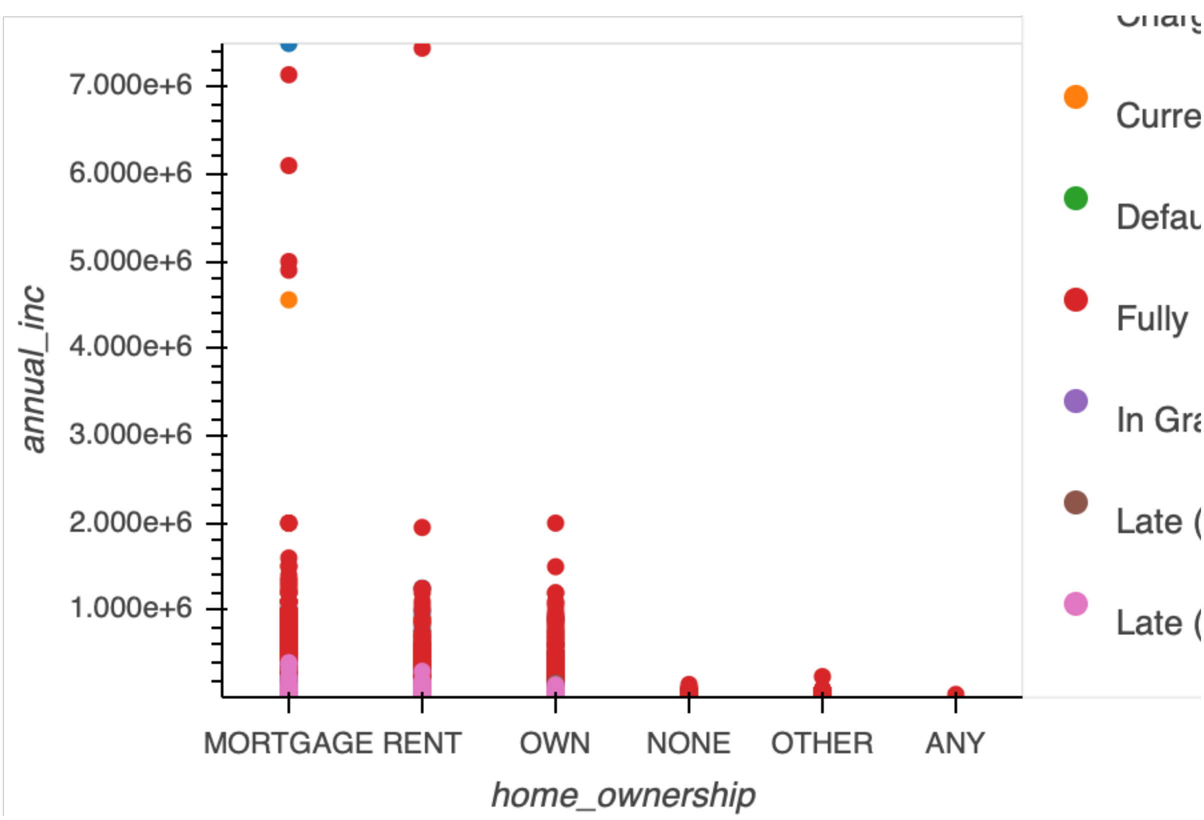
The "Late" or "Grace Period" loan status were persistent among population with

Now let's find how the Home Ownership and their Annual Income by their Loan Status

In [18]:

```
hv.extension('bokeh')
img = new_df.hvplot(kind='scatter', x='home_ownership', y='annua
l_inc', by='loan_status')
img.options(frame_width=300)
```

Out[18]:



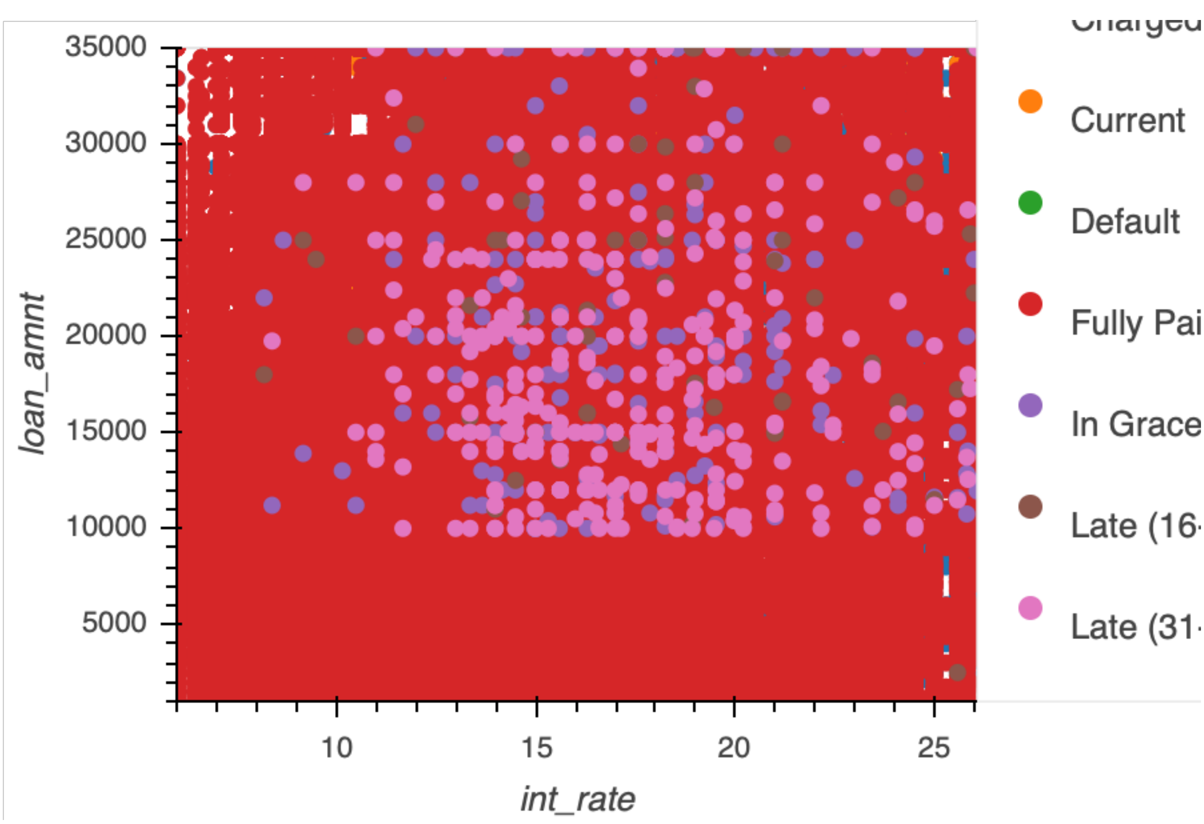
It can be see that population that had high income had their home mortgaged. The houses that were "owned" actually belonged to the class of population who had income from 1.000e+6 to 2.000e+6.

Now it's time to find the interest rates for different loan amounts.

In [19]:

```
hv.extension('bokeh')
img = new_df.hvplot(kind='scatter', x='int_rate', y='loan_amnt',
by='loan_status')
img.options(frame_width=300)
```

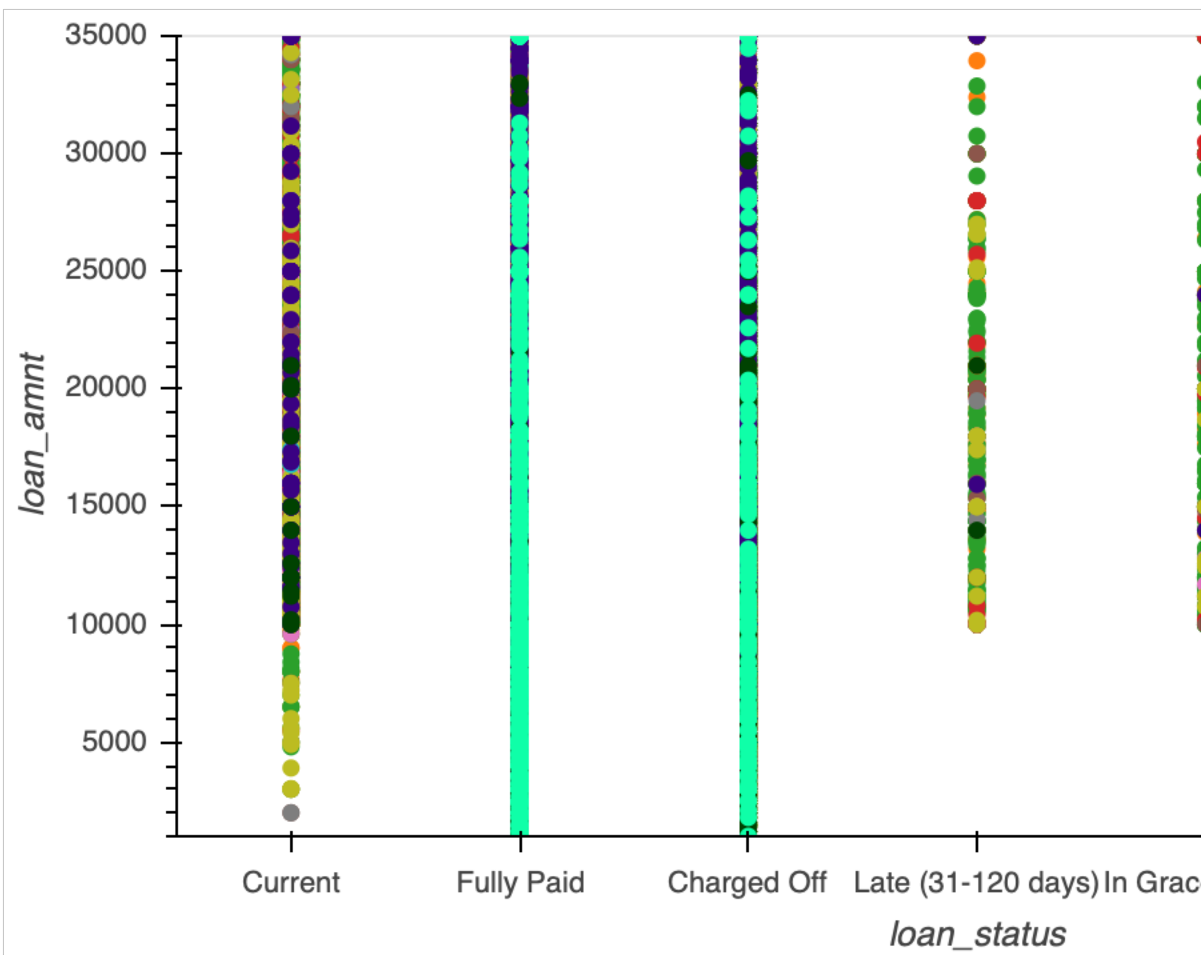
Out[19]:



In [20]:

```
hv.extension('bokeh')
img = new_df.hvplot(kind='scatter', x='loan_status', y='loan_amn
t', by='purpose')
img.options(frame_width=600, frame_height=300)
```

Out[20]:



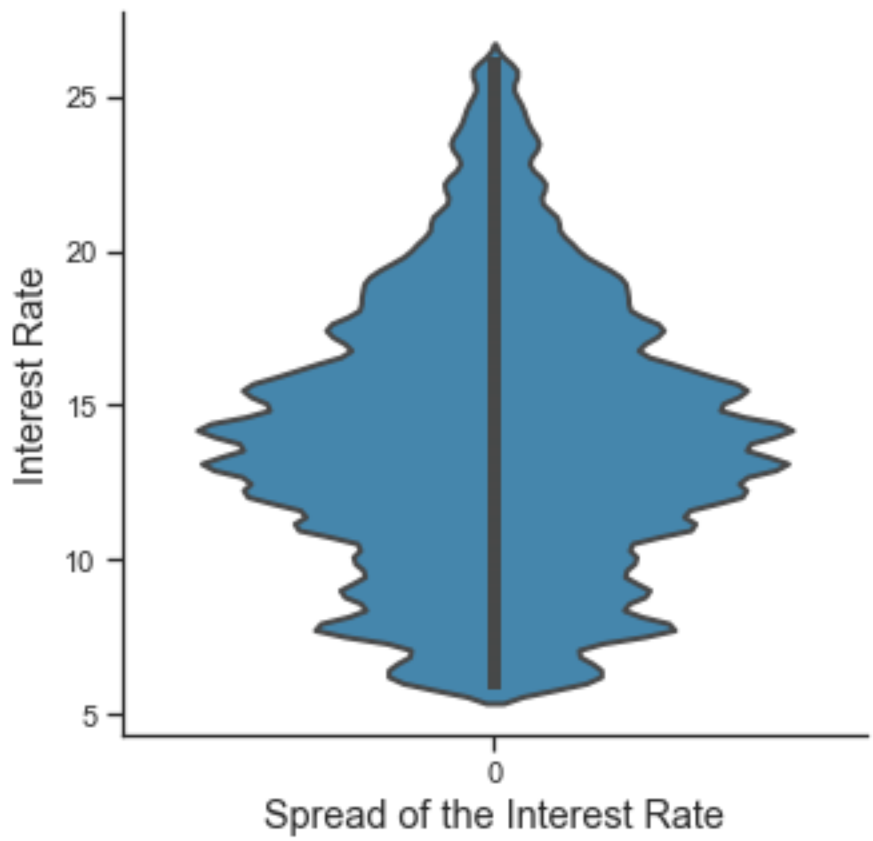
35000
30000
25000
20000
15000
10000
5000
loan_amnt
Current
Fully Paid
Charged Off
Late (31-120 days)
In Grac
loan_status

We can see that borrowers whose purpose were either wedding or small business can be trusted with the loan. The ones that are delaying to pay back their loans were found to be from debt consolidation, home_improvements , major purchases other few.

Let's Check the spread of the interest rate to find out the common interest rate that is normally offered.

In [21]:

```
# plot
sns.set_style('ticks')
fig, ax = plt.subplots()
# the size of A4 paper
fig.set_size_inches(5, 5)
plt.xlabel('Spread of the Interest Rate')
plt.ylabel('Interest Rate')
sns.violinplot(data=new_df['int_rate'], inner="points", ax=ax)
sns.despine()
```



So 15% interest rate is the most widely used.

In [22]:

```
## Subsetting the dataset to our prediction variable.
new_df = new_df.loc[new_df['loan_status'].isin(['Current','Charged Off'])]
## Reset Index
new_df = new_df.reset_index(drop=True)
```

In [23]:

```
new_df.loan_status.unique()
```

Out[23]:

```
array(['Charged Off', 'Current'], dtype=object)
```

Numeric Columns

In [24]:

```
numeric_columns = new_df.select_dtypes(['int64', 'float64']).columns
print(numeric_columns)
print('-------------------------')
print("The number of numerical columns is {}".format(len(numeric_columns)))
```

```
Index(['loan_amnt', 'funded_amnt', 'funded_amnt_inv'
, 'int_rate',
       'installment', 'annual_inc', 'dti', 'delinq_2
yrs', 'fico_range_low',
       'fico_range_high', 'inq_last_6mths', 'open_ac
c', 'pub_rec', 'revol_bal',
       'total_acc', 'out_prncp', 'out_prncp_inv', 't
otal_pymnt',
       'total_pymnt_inv', 'total_rec_prncp', 'total_
rec_int',
       'total_rec_late_fee', 'recoveries', 'collecti
on_recovery_fee',
       'last_pymnt_amnt', 'last_fico_range_high', 'l
ast_fico_range_low',
       'collections_12_mths_ex_med', 'policy_code',
'acc_now_delinq',
       'chargeoff_within_12_mths', 'delinq_amnt', 'p
ub_rec_bankruptcies',
       'tax_liens'],
      dtype='object')
------------------------
The number of numerical columns is 34
```

Non numeric columns

In [25]:

```
non_numeric_columns = new_df.select_dtypes(['object']).columns
print(non_numeric_columns)
print("The number of non-numerical columns is {}".format(len(non
_numeric_columns)))
```

```
Index(['id', 'term', 'grade', 'sub_grade', 'home_own
ership',
       'verification_status', 'loan_status', 'pymnt_
plan', 'url', 'purpose',
       'zip_code', 'addr_state', 'earliest_cr_line',
'initial_list_status',
       'application_type', 'hardship_flag', 'disburs
ement_method',
       'debt_settlement_flag'],
      dtype='object')
The number of non-numerical columns is 18
```

In [26]:

```
## Dropping columns that are least important like 'url', 'zip_co
de', etc
new_df.drop(columns=['id','home_ownership',
       'verification_status', 'pymnt_plan', 'url', 'purpose', 'z
ip_code',
       'addr_state','application_type', 'hardship_flag', 'disbur
sement_method','debt_settlement_flag'], inplace=True)
```

In order to deal with a large number of non numeric columns let's find out the top highest correlated non numeric columns.

For all pairs of the categorical features of combining and evaluating two variables as a combination, `comb_cat_feat` let's calculate the Cramer's V correlation coefficient that is expressed through the chi-square statistic χ^2 of the contingency table:

$$V = \sqrt{\frac{\chi^2}{n(\min(K_1, K_2) - 1)}}$$

where n is the sum of all elements in the contingency table, K_1 and K_2 are the dimensions of the contingency table. Note that Pearson's R correlation coefficient isn't applicable to categorical features and shouldn't be used.

In [27]:

```
## Categorical Features
cat_feat = new_df.select_dtypes('object').columns.values
new_df[cat_feat].nunique().sort_values()
```

Out[27]:

```
term                    2
loan_status             2
initial_list_status     2
grade                   7
sub_grade              35
earliest_cr_line      611
dtype: int64
```

In [28]:

```
from scipy.stats import chi2_contingency
from itertools import combinations

cat_feat = new_df.select_dtypes('object').columns.values
comb_cat_feat = np.array(list(combinations(cat_feat, 2)))
corr_cat_feat = np.array([])
for comb in comb_cat_feat:
    table = pd.pivot_table(new_df, values='loan_amnt', index=comb[0], columns=comb[1], aggfunc='count').fillna(0)
    corr = np.sqrt(chi2_contingency(table)[0] / (table.values.sum() * (np.min(table.shape) - 1) ) )
    corr_cat_feat = np.append(corr_cat_feat, corr)
```

In [29]:

```
high_corr_cat = comb_cat_feat[corr_cat_feat >= 0.5]
high_corr_cat
```

Out[29]:

```
array([['grade', 'sub_grade']], dtype='<U19')
```

The two non-numeric features that were highly correlated were Grade and Sub Grade. We will include them in our machine learning model.

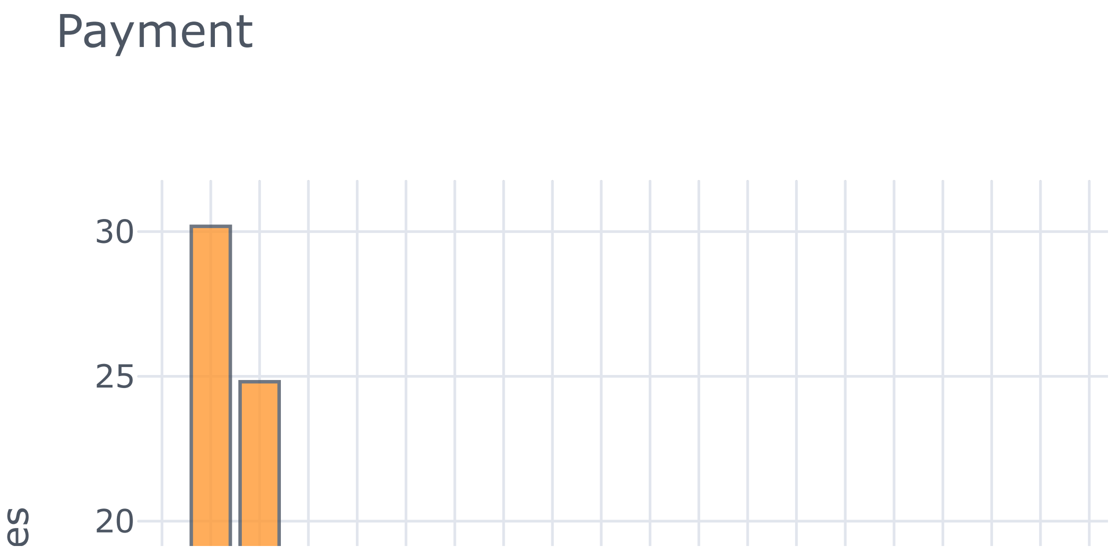
Let's check the distribution of grade and sub grades as they were highly correlated to eachother.

In [30]:

```
def configure_plotly_browser_state():
    import IPython
    display(IPython.core.display.HTML('''
        <script src="/static/components/requirejs/require.js"></
script>
        <script>
          requirejs.config({
            paths: {
              base: '/static/base',
              plotly: 'https://cdn.plot.ly/plotly-latest.min.js?
noext',
            },
          });
        </script>
        '''))
```

In [31]:

```
configure_plotly_browser_state()
init_notebook_mode(connected=False)

new_df[['grade','sub_grade']].iplot(
    kind='hist',
    histnorm='percent',
    barmode='overlay',
    xTitle='Grades',
    yTitle='(%) of Grades',
    title='Payment')
```



Let's Label Encode them for ML purpose

In [32]:

```
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()
new_df.grade.unique()
new_df.grade = LE.fit_transform(new_df.grade)
```

In [33]:

```
new_df.grade = pd.to_numeric(new_df.grade, errors = 'coerce')
new_df.grade.unique()
```

Out[33]:

```
array([0, 2, 3, 1, 4, 6, 5])
```

In [34]:

```
new_df.sub_grade = LE.fit_transform(new_df.sub_grade)
new_df.sub_grade = pd.to_numeric(new_df.sub_grade, errors = 'coe
rce')
```

In [35]:

```
new_df.sub_grade.unique()
```

Out[35]:

```
array([ 2, 12, 18, 13, 19,  9, 24, 23,  7, 22, 14, 1
7, 10, 15,  5, 31, 20,
       11, 26, 27, 21, 30,  6, 32,  8, 25, 16,  4,
1, 28,  3, 29, 34,  0,
       33])
```

Now lets Engineer some features that will have high correlation with the target variable, loan_status.

There is a high correlation between **"total_rec_prncp"** and **"total_pymnt_inv"**. There is also another high correlation between **"out_prncp"** and **"total_rec_prncp"**. Lets create **interaction** between them.

In [36]:

```
new_df.drop(columns=['earliest_cr_line','term','initial_list_sta
tus'], inplace=True)
```

Let's parse the Timestamp data and then convert them to numeric.

In [37]:

```
new_df[['issue_d']] = new_df[['issue_d']].astype(np.int64) // 10
**9
new_df.issue_d
```

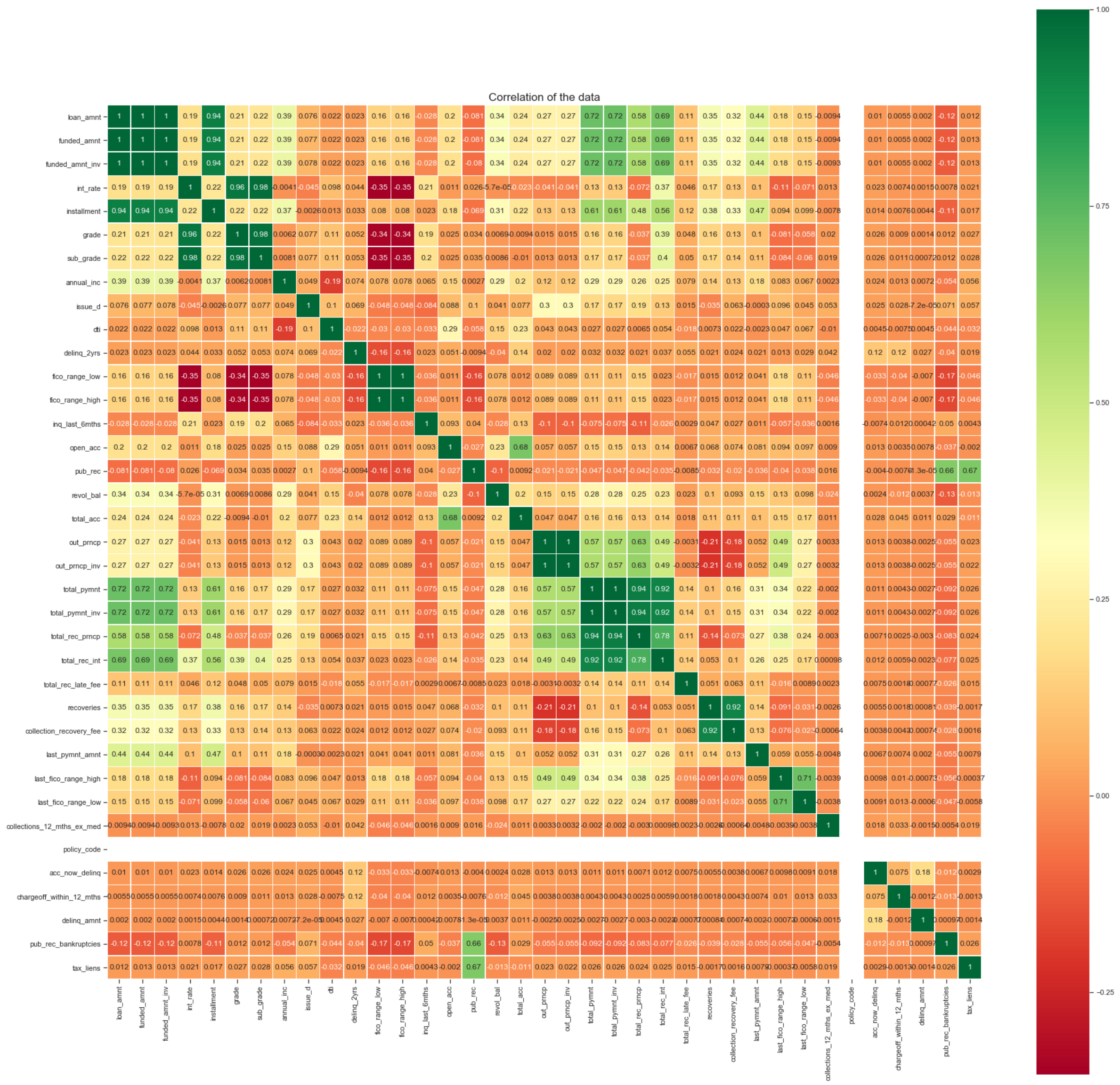
Out[37]:

```
0        1417392000
1        1417392000
2        1417392000
3        1417392000
4        1417392000
            ...
82749    1325376000
82750    1325376000
82751    1325376000
82752    1325376000
82753    1325376000
Name: issue_d, Length: 82754, dtype: int64
```

Heat Map to find the Correlation of each feature:

In [38]:

```
plt.figure(figsize=(30, 30))
sns.heatmap(new_df.corr(), square=True, annot=True, linewidths=.
5,  cmap='RdYlGn')
plt.title("Correlation of the data")
plt.show()
```



There are many features that are highly correlated to each other such as total payment, total received inv, total received interest, total amount, total amount invested.

Modeling

In [39]:

```
X = new_df.drop('loan_status', 1)
Y = new_df.loan_status
```

In [40]:

```
X
```

Out[40]:

	loan_amnt	funded_amnt	funded_amnt_inv	int_rate	installment
0	10400.0	10400.0	10400.000000	6.99	321.08
1	7650.0	7650.0	7650.000000	13.66	260.20
2	12800.0	12800.0	12800.000000	17.14	319.08
3	23325.0	23325.0	23325.000000	14.31	800.71
4	12975.0	12975.0	12975.000000	17.86	468.17
...	...	...	...	...	...
82749	15200.0	15200.0	14443.634553	17.27	379.97
82750	4900.0	4900.0	4900.000000	16.77	121.18
82751	17500.0	16800.0	16775.000000	22.74	471.10
82752	35000.0	22550.0	22550.000000	14.27	527.87
82753	12000.0	12000.0	12000.000000	16.29	423.61

82754 rows × 37 columns

In [41]:

```
# Standarizing the features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

In [42]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(X,Y, test_si
ze = 0.25)
```

In [43]:

```
X_train.columns
```

Out[43]:

```
Index(['loan_amnt', 'funded_amnt', 'funded_amnt_inv'
, 'int_rate',
       'installment', 'grade', 'sub_grade', 'annual_
inc', 'issue_d', 'dti',
       'delinq_2yrs', 'fico_range_low', 'fico_range_
high', 'inq_last_6mths',
       'open_acc', 'pub_rec', 'revol_bal', 'total_ac
c', 'out_prncp',
       'out_prncp_inv', 'total_pymnt', 'total_pymnt_
inv', 'total_rec_prncp',
       'total_rec_int', 'total_rec_late_fee', 'recov
eries',
       'collection_recovery_fee', 'last_pymnt_amnt',
'last_fico_range_high',
       'last_fico_range_low', 'collections_12_mths_e
x_med', 'policy_code',
       'acc_now_delinq', 'chargeoff_within_12_mths',
'delinq_amnt',
       'pub_rec_bankruptcies', 'tax_liens'],
      dtype='object')
```

In [44]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import ensemble
lr_r = ensemble.RandomForestClassifier(n_estimators=5)
lr_r.fit(X_train,y_train)
predictions_lrr = lr_r.predict(X_train)
```

In [45]:

```
report = classification_report(y_train, predictions_lrr)

print(report)
```

```
              precision    recall  f1-score   suppor
t

 Charged Off       1.00      1.00      1.00     5316
8
     Current       1.00      1.00      1.00      889
7

    accuracy                           1.00     6206
5
   macro avg       1.00      1.00      1.00     6206
5
weighted avg       1.00      1.00      1.00     6206
5
```

In [45]:

```
predictions_lrr = lr_r.predict(X_test)
report = classification_report(y_test, predictions_lrr)

print(report)
```

```
              precision    recall  f1-score   suppor
t

 Charged Off       1.00      1.00      1.00      1774
2
     Current       1.00      0.99      1.00       294
7

    accuracy                           1.00      2068
9
   macro avg       1.00      1.00      1.00      2068
9
weighted avg       1.00      1.00      1.00      2068
9
```

In [46]:

```
from sklearn.metrics import accuracy_score
print (accuracy_score(y_test,predictions_lrr))
```

```
0.9987432935376287
```

Logistic Classifier

In [47]:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='lbfgs', penalty='l2', max_iter=1
0000)
lr.fit(X_train, y_train)

test_score = lr.score(X_test, y_test)
train_score = lr.score(X_train, y_train)

print('Score on training data: ', train_score)
print('Score on test data: ', test_score)
```

```
Score on training data:  0.8553452026101668
Score on test data:  0.857557155976606
```

In [48]:

```
predictions_lg = lr.predict(X_test)

print (accuracy_score(y_test,predictions_lg))
```

```
0.857557155976606
```

In [49]:

```
eport = classification_report(y_test, predictions_lg)

print(report)
```

```
/Users/sajithgowthaman/opt/anaconda3/lib/python3.7/s
ite-packages/sklearn/metrics/classification.py:1437:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples.


              precision    recall  f1-score   suppor
t

 Charged Off       1.00      1.00      1.00      1774
2
     Current       1.00      0.99      1.00       294
7

    accuracy                           1.00      2068
9
   macro avg       1.00      1.00      1.00      2068
9
weighted avg       1.00      1.00      1.00      2068
9
```

KNN

In [50]:

```
from sklearn import neighbors
### Tuned KNN Model
knn = neighbors.KNeighborsClassifier(n_neighbors=15, leaf_size=5
, p=1)
knn.fit(X_train, y_train)

print(knn.score(X_train, y_train))

knn_w = neighbors.KNeighborsClassifier(n_neighbors=24, weights='
distance')
knn_w.fit(X_train, y_train)

print(knn_w.score(X_train, y_train))
```

```
0.987932006767099
1.0
```

In [51]:

```
predictions_knn = knn.predict(X_test)
report = classification_report(y_test, predictions_knn)
print(report)
print('---------------------------------------')
print('The Accuracy score with KNN is {}'.format(accuracy_score(
y_test,predictions_knn)))
```

```
              precision    recall  f1-score   suppor
t

 Charged Off       1.00      0.99      0.99     1774
2
     Current       0.93      0.98      0.95      294
7

    accuracy                           0.99     2068
9
   macro avg       0.96      0.98      0.97     2068
9
weighted avg       0.99      0.99      0.99     2068
9

---------------------------------------
The Accuracy score with KNN is 0.9859345545942289
```

Boosting Model

In [52]:

```
params_new = {'n_estimators': 1000,
          'max_depth': 3,
          'loss': 'exponential'}

# Initialize and fit the model.
clf_b = ensemble.GradientBoostingClassifier(**params_new)
clf_b.fit(X_train, y_train)

predict_train_new = clf_b.predict(X_train)
predict_test_new = clf_b.predict(X_test)

test_score = clf_b.score(X_test, y_test)
train_score = clf_b.score(X_train, y_train)

print('Score on training data: ', train_score)
print('Score on test data: ', test_score)
```

```
Score on training data:  1.0
Score on test data:  0.9997099908163759
```

Support Vector Classifier

In [53]:

```
from sklearn.svm import SVC
svc = SVC(gamma = 'auto')
svc.fit(X_train,y_train)
print(svc.score(X_train, y_train))
```

```
1.0
```

In [54]:

```
predict_train_svc = svc.predict(X_train)
predict_test_svc = svc.predict(X_test)

test_score = svc.score(X_test, y_test)
train_score = svc.score(X_train, y_train)
```

In [55]:

```
print('Score on training data: ', train_score)
print('Score on test data: ', test_score)
```

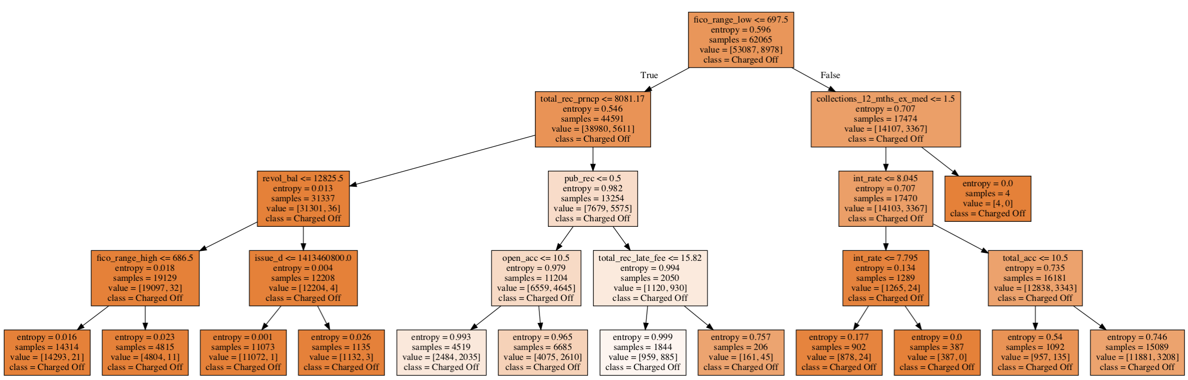
```
Score on training data:  1.0
Score on test data:  0.857557155976606
```

Decision Tree

In [56]:

```python
# This is the model we'll be using.
from sklearn import tree

# A convenience for displaying visualizations.
from IPython.display import Image

# Packages for rendering our tree.
import pydotplus
import graphviz

# Initialize and train our tree.
decision_tree = tree.DecisionTreeClassifier(
    criterion='entropy',
    max_features=1,
    max_depth=4,
    random_state = 1337
)
decision_tree.fit(X_train, y_train)

# Render our tree.
dot_data = tree.export_graphviz(
    decision_tree, out_file=None,
    feature_names=X_train.columns,
    class_names=['Charged Off', 'Current'],
    filled=True
)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

Out[56]:



In [57]:

```
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()
new_df.loan_status = new_df.loan_status.astype(str)
new_df.loan_status = LE.fit_transform(new_df.loan_status)
```

In [58]:

```
X = new_df.drop('loan_status', 1)
Y = new_df.loan_status
```

In [59]:

```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_si
ze = 0.25)
```

In [60]:

```
##Decision Tree
lr_dt = tree.DecisionTreeClassifier()
lr_dt.fit(X_train,y_train)
predictions_lrdt = lr_dt.predict(X_test)
report = classification_report(y_test, predictions_lrdt)

print(report)
```

```
              precision    recall  f1-score   suppor
t

           0       1.00      1.00      1.00     1770
8
           1       0.99      0.99      0.99      298
1

    accuracy                           1.00     2068
9
   macro avg       1.00      1.00      1.00     2068
9
weighted avg       1.00      1.00      1.00     2068
9
```

In [61]:

```
print('The Accuracy score with Decision Tree Classifier is {}'.f
ormat(accuracy_score(y_test,predictions_lrdt)))
```

```
The Accuracy score with Decision Tree Classifier is
0.9981632751703804
```

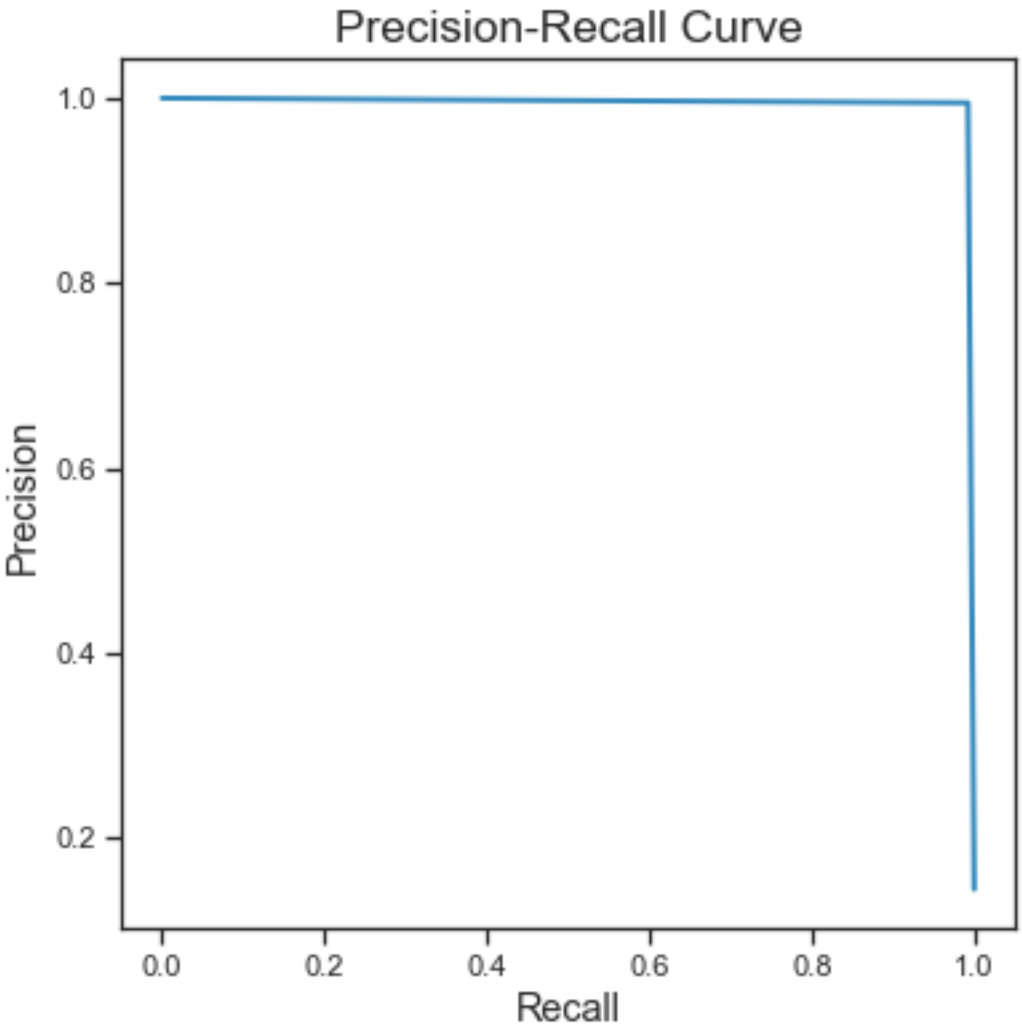
In [62]:

```
from sklearn.metrics import roc_curve, precision_recall_curve
probs = lr_dt.predict_proba(X_test)[:, 1]
print(probs[1:30])
fpr, tpr, thresholds = roc_curve(y_test, probs)
```

```
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
```

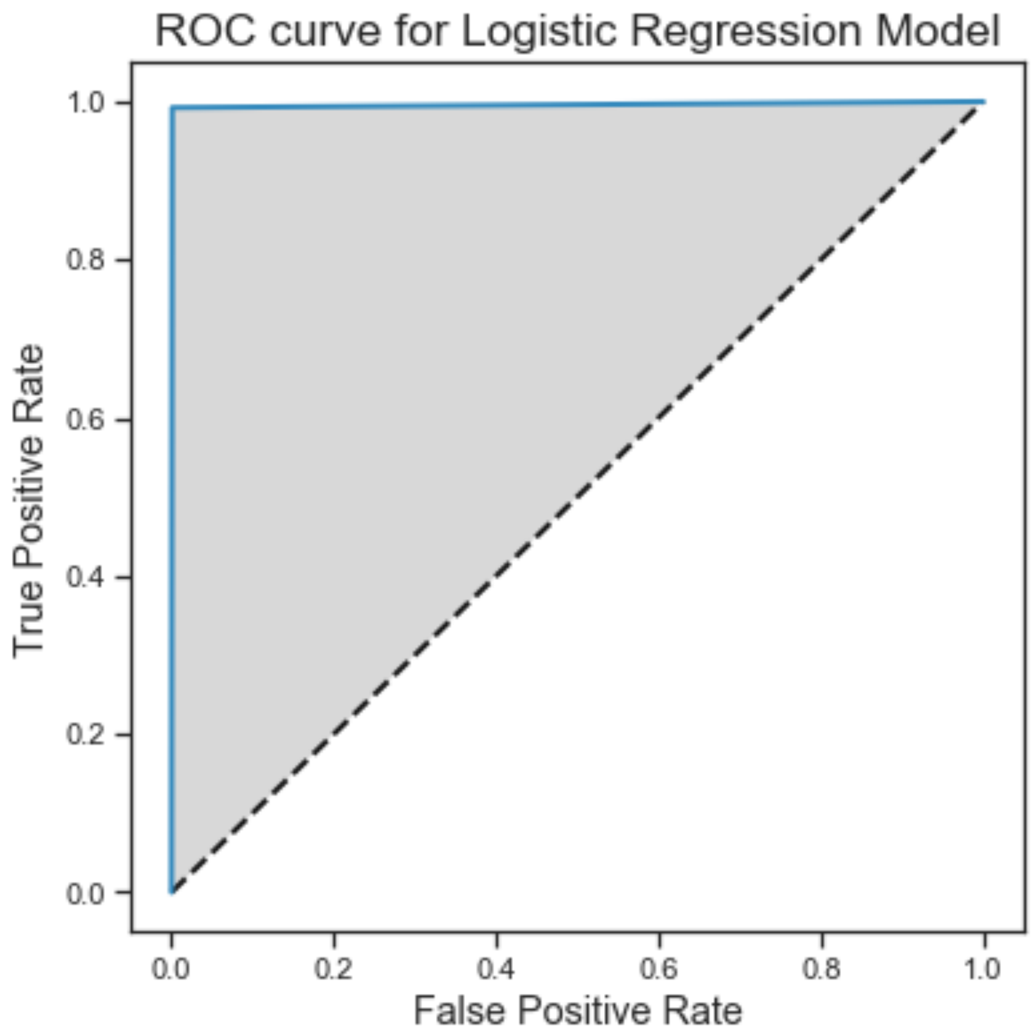
In [63]:

```
pres, rec, thresholds = precision_recall_curve(y_test, predictio
ns_lrdt)
fig = plt.figure(figsize = (6, 6))
plt.plot(rec, pres)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()
```


Precision-Recall Curve
Precision
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0
Recall

In [64]:

```
from sklearn.metrics import roc_curve, auc
fig = plt.figure(figsize = (6, 6))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.fill(fpr, tpr, 'grey', alpha=0.3)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for Logistic Regression Model')
plt.show()
```



Random Forest and KNN Model classifiers were the highest at **98% to 100%.** The accuracy was validated with the train, test score and plotted ROC curve to find the true positives at the top left corner of the graph.

Summary

- We were able to explore the data and extract useful insights from the data.
- The EDA and visualization will help the money lenders/borrowers to make the right decision
- The model was created to predict a classification for loan_status.
- Accuracy was boosted from a baseline of 75% accuracy to a high 98 percent.
- This model can be used to predict the loan_status for any given features that matches with this dataframe.

Project by

- **NAME: SAJITH GOWTHAMAN**
- **NET ID: ek5282**

Extra Credit

In [46]:

```
df_2015 = df[(df.issue_d >= '2015-01-01 00:00:00') & (df.issue_d
< '2016-01-01 00:00:00')]
df_2015 = new_df.reset_index(drop=True)
```

In [47]:

```
## Subsetting the dataset to our prediction variable.
df_2015 = df_2015.loc[df_2015['loan_status'].isin(['Current','Ch
arged Off'])]
## Reset Index
df_2015 = df_2015.reset_index(drop=True)
```

In [48]:

```
df_2015 = df_2015.loc[:, df_2015.isnull().mean() <= 0.0]
```

In [50]:

```
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()
new_df.grade.unique()
new_df.grade = LE.fit_transform(df_2015.grade)
```

In [51]:

```
df_2015.grade = pd.to_numeric(df_2015.grade, errors = 'coerce')
df_2015.grade.unique()
```

Out[51]:

```
array([0, 2, 3, 1, 4, 6, 5])
```

In [52]:

```
df_2015.sub_grade = LE.fit_transform(df_2015.sub_grade)
df_2015.sub_grade = pd.to_numeric(df_2015.sub_grade, errors = 'c
oerce')
```

In [54]:

```
df_2015[['issue_d']] = df_2015[['issue_d']].astype(np.int64) //
10**9
df_2015.issue_d
```

Out[54]:

```
0        1
1        1
2        1
3        1
4        1
        ..
82749    1
82750    1
82751    1
82752    1
82753    1
Name: issue_d, Length: 82754, dtype: int64
```

Let's Predict the Loan status for 2015 using our best models KNN and Random Forest.

In [55]:

```
X = new_df.drop('loan_status', 1)
Y = df_2015.loan_status
```

In [57]:

```
from sklearn.metrics import classification_report

from sklearn.ensemble import RandomForestClassifier
from sklearn import ensemble
lr_r = ensemble.RandomForestClassifier(n_estimators=5)
lr_r.fit(X,Y)
predictions_lrr = lr_r.predict(X)
```

In [59]:

```
predictions_lrr
```

Out[59]:

```
array(['Charged Off', 'Charged Off', 'Current', ...,
'Charged Off',
       'Charged Off', 'Charged Off'], dtype=object)
```

We were able to predict the Loan status for the year 2015!
:D

In [ ]:

```
```